After Dark is a screen saving shell. It provides all of the support and tools a programmer needs to quickly create unique screen saving graphics. To create a new screen saver, you simply write a new *graphics module*. A screen saver graphics module is a small, separately compiled code resource. It is loaded into memory by After Dark when the user places the pointer in a chosen corner of the screen, or after a specified delay. You can develop a graphics module in any development system that allows generation of code resources. This Programmer's Manual is a description of the structure of a graphics module.

This manual is split into two parts. The first is a "Getting Started" section that discusses the basic design of a graphics module and provides the minimum material necessary to write a graphics module. The second part covers advanced topics. These topics include After Dark resources, full descriptions of data structures, and special features provided by After Dark.

Included on the After Dark disk is source code in C and Pascal for the MPW and Think programming environments. The source code demonstrates how to write an After Dark graphics module.

To program a graphics module one need not be a "Macintosh Guru." In fact, the simplest module (one that just blanks the screen) is less than 20 lines of code. Some familiarity with Macintosh programming is expected but simple modules can be created fairly quickly by novice programmers.

In addition to letting you create your own screen saver graphics, After Dark allows you to implement sliders, menus, check boxes, text strings, and buttons in the Control Panel to change the behavior of your screen saver. Sound can also be added to enhance the effect of your screen saver. After Dark also provides graphics modules with information about the number, size and pixel depth of the monitors in use, the availability of Color QuickDraw, and a copy of the QuickDraw global variables. A description of these are found in the "Advanced Features" section of this manual.

A graphics module is a code resource of type `'ADgm'`. When After Dark calls a graphics module, it calls the function `main()` in the `'ADgm'` code resource. We have written a generic function `main()` in each of the example projects. It can be used as a basis to create a new graphics module.

When activated, After Dark calls a graphics module and passes it a message. The message asks the module to perform a certain task. Basic messages are:

- Ask the module to initialize itself.
- Ask the module to blank the screen.
- Ask the module to draw a frame of animation.
- Ask the module to finish and clean up.

If the user is in the Control Panel, three other messages can be passed:

- Tell the module a button has been pressed.
- Tell the module it has been selected.
- Ask the module to display help or information about itself.

These last messages are optional and are described further in the advanced section.

For further information about code resources, consult your development system's documentation.

When the function `main()` is called, four parameters are passed:

`storage`	–	A handle to storage allocated by the graphics module. This memory is used to store variables used by the module.
`blankRgn`	–	A handle to the region the graphics module is allowed to draw in. This region encompasses the screens of all attached monitors. Drawing is clipped to this region.
`message`	–	This value tells the graphics module which task After Dark wants it to perform.
`params`	–	This value points to a structure that contains information about the Macintosh that After Dark is running on. It also contains the values of any controls the module has created. The fields of this structure are described in depth in "Advanced Features."

Note: all examples in the manual are in Pascal. Examples in C are on the After Dark disk.

The function `main()` is defined as:

```
function main(
             VAR storage: Handle;
             blankRgn: RgnHandle;
             message: integer;
             params: GMParamBlockPtr;) : OSErr;
```

`main()` acts as a "dispatcher." It looks at the message passed and calls a function to perform the task specified in the message. Messages are passed in a specific order.

On the After Dark disk, we provide a sample `main()` function that you can use as-is. To use our sample `main()` function, just write your own routines `DoInitialize()`, `DoBlank()`, `DoDrawFrame()`, and `DoClose()` as described briefly in the next section. Link them with `main()` to make a code resource of type `'ADgm'`. Put the resource in a file of type `'ADgm'` and creator `'ADrk'`. Your module is ready to run.

When a user invokes After Dark, the function `main()` is called repeatedly. In the example code, `main()` calls a different message handling function for each message. The messages are (in order passed):

`Initialize – main()` calls your function `DoInitialize()`. This function should allocate memory needed by the graphics module and initialize the module's variables. The graphics module should allocate a Handle to a block of memory that it will use to hold any variables it needs. It should then assign it to the parameter `storage`. No drawing should be done at this time.

`Blank – main()` calls your function `DoBlank()`. This function should blank the screen. A banner titling the graphics module or giving credit to its author might be drawn here. No further drawing should happen.

`DrawFrame – main()` calls your function `DoDrawFrame()`. This function is responsible for drawing the animation on the screen. It is called repeatedly and should draw the next frame of the animation sequence. The basic idea is to "erase old, draw new."

`Close - main()` calls your function `DoClose()`. At this point, the screen saver is finished. Any memory allocated should be disposed of and any other "house cleaning" should be performed. Drawing is allowed, but your module should return quickly since the user is waiting to regain control. After Dark handles updating the screen.

A graphics module may call any of the routines documented in *Inside Macintosh* with *one exception*. Calls to the Window Manager and Palette Manager are not recommended since After Dark does not draw into a window.

If you are ready to build a simple graphics module, go for it! With this information you should be able to build a basic screen saver. Try using the source code included on the disk as a starting point. If you want to learn more about the features of After Dark, read on.

This section describes the advanced features available in After Dark and provides a complete description of all data structures and resources used by After Dark. This section also explains how to create controls for graphics modules such as sliders, menus, text strings, check boxes, and buttons, how to include sound in your module, and how to add a "Help Box" to your module.

The function `main()` must be the first routine in your graphics module code resource. `main()` is passed a sequence of messages to direct its behavior:

`Initialize`
When this message is received, your module should allocate a Handle to memory and assign it to the parameter `storage`. This block of memory is used to store the graphics module's variables.

It is important to understand why variables might be stored this way. In many development systems a code resource cannot have global variables. If your development system does not support globals, a structure must be declared and memory allocated as in our example code. The fields of this structure can be any data type.

Remember, the `storage` handle is *not* locked down when it is passed to `main()` so it is the module's responsibility to lock the handle when necessary, and unlock it before returning.

`Blank`
When you receive the blank message, blank the screen if you want. Call your major animation routines when you receive the `DrawFrame` message.

`DrawFrame`
The frequency with which this message is received depends on how loaded down the system is. This is because After Dark tries to cooperate with the system by calling graphics modules only at idle times. In keeping with this philosophy, a graphics module should not try to do too much at one time.

`Close`
It is important to dispose of any memory you have allocated. If you do not, the Macintosh will think it has less memory available than it actually does. Each time your module is used, the memory available to your Macintosh will decrease.

`ButtonMessage`
If a user clicks on a button in the Control Panel (buttons are described in the section "Control Resources"), the message `ButtonMessage` is passed to `main()`. The value of `ButtonMessage`

depends on which button was hit. The value of each button is distinct and is defined in a resource. All messages with a value of *8 or greater* are button messages. In the example code, `ButtonMessage` is handled by the `DoSetup()` function.

The graphics module could use this kind of message to open a dialog box allowing the user to set values not easily handled by sliders, menus, or check boxes. A button could allow the user to select a color from the "Color Picker" or choose a picture from a scrapbook, for example.

`ModuleSelected`
After Dark can send this message to your module when a user selects it in the Control Panel. This feature can be used to setup your module's controls before they are displayed. For example, a menu of the fonts in the system can be built when your module is selected.

To receive this message, add the `'Cals'` resource to your module and set it as described in the "Graphics Module Resources" section.

`DoHelp`
As described in the "Graphics Module Resources" section, modules may include a "Help Box" that provides the user with additional information or help. The "Help Box" is displayed when the user clicks in the "Credits" area. While no programming is necessary to display text or a picture, your module may want to handle the display itself (MultiModule does this).

As above, to receive this message, add a `'Cals'` resource to your module and set it as described in the "Graphics Module Resources" section.

Normally a graphics module returns the `OSErr` type value `noErr`. There are two conditions where this will not be the case:

In `DoInitialize()` a graphics module usually attempts to allocate memory. If there is an error in allocating the storage, the graphics module should return the value `ModuleError` (defined as integer value -1). If errors occur on later calls, the graphics module should dispose of any memory allocated and return this error value to After Dark.

There are three special values that the graphics module can use to ask After Dark to perform certain functions.

Returning the value `RestartMe` (defined as the integer value 1) tells After Dark to restart the graphics module immediately with an `Initialize` message without an intervening `Close` message. The graphics module should close itself before returning this value. The graphics module might use this service if it finds that it has been passed changing parameters that require it to reallocate storage.

Returning the value `ImDone` (defined as the integer value 2) tells After Dark to take over the drawing. The graphics module uses this when it is done drawing but wants something else to happen on the screen.

Returning the value `RefreshResources` (defined as the integer value 3) updates all sliders, buttons, check boxes, menus, and text strings in the Control Panel to their values in the module's resource file. This allows you to change the items in a menu or the values of text strings in your module's resource file and have these changes immediately reflected in the Control Panel. This value should only be returned in response to `ButtonMessage` or `ModuleSelected.`

When a graphics module is called, it is passed three parameters. The first two are simple data types, but the third, "params", is a pointer to a data structure that holds useful information about the Macintosh that After Dark is running on. This includes information about monitors, Color QuickDraw, and any controls your module may have created. The structure "params" points to is defined as:

```
GMParamBlock = record
       controlValues: array[0..3] of integer;
       monitors: MonitorsInfoPtr;
       colorQDAvail: Boolean;
       systemConfig: integer;
       qdGlobalsCopy: QDGlobalsPtr;
       brightness: integer;
       demoRect: Rect;
       errorMessage: StringPtr;
       sndChannel: SndChannelPtr;
       adVersion: integer;
end;
GMParamBlockPtr = ^GMParamBlock;
```

controlValues
This array contains the current value of any slider, menu, text string, or check box control created by the graphics module (slider, menu, text strings, and check box controls are resources and are described later in "Control Resources"). Slider values can range from 0 to 100 and can be interpreted however the programmer wishes. Menu values range from 1 to the number of items in the menu with the value representing the item currently selected. Text strings values range from 0 to the number of strings defined. Check box values are either 0 or 1.

These values can be changed at any time by your module and will be reflected in the Control Panel. To avoid confusion on the user's part, do not change these values unless the user requests it (an exception are text strings which may need to be changed to provide new information to the user).

monitors
This field is a pointer to a structure that contains the number of monitors attached and an array of structures with information about the monitors. The structures are defined as:

```
MonitorsInfo = record
       monitorCount: integer;
       monitorList[0..0]: MonitorData;
end;
MonitorsInfoPtr = ^MonitorsInfo;

MonitorData = packed record
       bounds: Rect;
       synchFlag: Boolean;
       curDepth: Byte;
end;
MonitorData = ^MonitorsInfo;
```

In the `MonitorsInfo` structure, the "`monitorCount`" field is the number of monitors attached to the Macintosh. The "`monitorList`" field is an array of `MonitorData` structures which contains information about each monitor attached. The first monitor in the list is always the monitor with the menu bar (also called the "main" monitor).

In the `MonitorData` structure, the "`bounds`" field is the monitor's rectangle in global coordinates.

The "synchFlag" field is used to help create smoother animation when drawing on the screen. The flag is set to true by After Dark when a vertical blanking ("VBL") interrupt occurs. Sufficiently rapid drawing begun immediately after this flag goes "true" should be flicker-free. For more information about VBL interrupts, see the Vertical Retrace Manager in *Inside Macintosh Volume 2*. The "curDepth" field is set to the pixel depth of the monitor.

If more information about a monitor is needed, the routines in *Inside Macintosh* can be used.

colorQDAvail
This field indicates whether or not Color QuickDraw is present on the Macintosh. This does *not* mean the Macintosh is running in color or has a color monitor. It only means the Color QuickDraw routines can be called. On a Macintosh SE/30 "colorQDAvail" is true.

systemConfig
This 16 bit value is a set of flags holding information about the configuration of the Macintosh. The flags are defined as:

Bit 0: cqdAvail - The Macintosh has Color QuickDraw.
Bit 1: anyMultibit - A monitor is set to greater than 1 bit in depth.
Bit 2: allMultibit - All monitors are set to greater than 1 bit in depth.
Bit 3: anyColor - A monitor is in color mode.
Bit 4: allColor - All monitors are in color mode.
Bit 5: anyCLUTDevice - A monitor is a CLUT device.
Bit 6: allCLUTDevice - All monitors are CLUT devices.
Bit 7: allCanDim - All monitors can dim.
Bit 8: mainMonCanDim - The main monitor can dim.
Bit 9: moduleMayNotAnimate - Module may not animate the color table.
Bit 10: multiModuleRunning - Module is running under MultiModule.
Bits 11-13: Reserved.
Bit 14: extensionsAvailable - Module is running under After Dark 2.0u or later.
Bit 15: soundAvail - This version of After Dark supports sound.

Note: For more information about CLUT devices, see the Color Manager in *Inside Macintosh Volume 5.*

A bit is set to 1 if the corresponding statement is true. Bit 0 is the rightmost (least-significant) bit.

If a module requires more information about the Macintosh than "hasColorQD" and "systemConfig" provide, use the SysEnvirons() or Gestalt() routines documented in *Inside Macintosh*.

qdGlobalsCopy
A code resource cannot directly access the QuickDraw global variables. Because they are quite useful, a *read-only copy* of them has been provided. The field "qdGlobalsCopy" is a pointer to the structure QDGlobals. This structure is defined as:

```
QDGlobals = record
       qdThePort:       GrafPtr;
       qdWhite:         Pattern;
       qdBlack:         Pattern;
       qdGray:          Pattern;
       qdLtGray:        Pattern;
       qdDkGray:        Pattern;
       qdArrow:         Cursor;
       qdScreenBits:    BitMap;
       qdRandSeed:      Longint;
```

```
end;
QDGlobalsPtr = ^QDGlobals;
```

If you are unfamiliar with any of these values or QuickDraw Globals in general, see *Inside Macintosh Volume 1*.

brightness
This field is used by a graphics module to tell After Dark to dim the monitors to a certain level. The values range from 0 (black) to 255 (normal). A module need only place a value in this field and After Dark will dim all of the monitors to the specified level. This only works if the `"allCanDim"` or `"mainMonCanDim"` bit in the `systemConfig` field is set.

demoRect
When After Dark is in "Demo Mode", this field contains the global coordinates of the controls area. When After Dark is not in "Demo Mode" it contains an empty rectangle. This field is useful if a graphics module does something special when After Dark is in "Demo Mode."

errorMessage
If your module returns an error code to After Dark, a short message can be displayed explaining the problem. Before your module returns, copy a message into the `Str255` string pointed at by `'errorMessage'`. This can be done with the Macintosh routine `BlockMove()`. After Dark will display this message on the screen and will take over animation.

sndChannel
If a module wants to use sound, it needs a sound channel. By adding a resource of type `'Chnl'` (described in the section "Sound Resources") to your module, After Dark will allocate a sound channel and place a pointer to it in the `'sndChannel'` field. To add sound to your module, see the next section "Using Sound".

adVersion
This field contains the version of After Dark in Binary Coded Decimal (BCD) format. For After Dark 2.0, this field contains the value 0x0200 hexadecimal, 512 decimal.

After Dark allows modules to play sounds while saving the screen. To use sound in your module you must have a resource of type `'Chnl'` (described in the section "Sound Resources"). You also need to use two libraries on the After Dark disk, Sound and EntryPoints. The Sound library sends the sound calls your module makes on to After Dark (which carries them out). The EntryPoints library coordinates the communication between the Sound library and After Dark. If you are using MPW C, MPW Pascal or Think Pascal use the libraries Sound.o and EntryPoints.o. If you are using Think C use the libraries Sound.π and EntryPoints.π.

Note: This schema for handling sound is different from previous versions of After Dark and will only work with After Dark 2.0u or later. You must check for the presence of the sound routines in After Dark. To do this we have defined a bit in the `systemConfig` field of `GMParamBlock`. Bit 14 is set to 1 if the sound routines are present. In `DoInitialize()` your module should check this bit before attempting to play sound.

The sound routines support the playing of `'snd '` resources (also known as `sampledSynth` type sounds). There are four routines used to play sound. All of the routines take the parameter `"sndChannel"` (`OpenSound()` also takes `"params"` as a parameter). The routines are: `OpenSound()`, `PlaySound()`, `QuietSound()`, and `CloseSound()`. Call `OpenSound()` in your module's `DoInitialize()` routine to setup the playing of sound. To play a sound, call `PlaySound()` with a handle to a `'snd '` resource. If you want to stop a sound being played, call `QuietSound()`. Call `CloseSound()` in you module's `DoClose()` routine to shutdown sound playing.

For more information about sound, see the Sound Manager chapter in *Inside Macintosh Volume 6*.

Only one resource is necessary in a graphics module. This is the code resource of type `'ADgm'`. The rest of the resources described are optional.

After Dark makes use of several different resource types to implement its features. A few are common but the majority are new. Most of these resources are used to construct sliders, buttons, check boxes, and menus in the Control Panel. Below is a description of all the resource types used by After Dark.

A graphics module may have the following two resource types, but both are optional:

CREDIT LINE
This resource is a string that is displayed in the Control Panel when your graphics module is selected. This can be your name, the version number of your module, or any other information. Here is its format:

`STR`	`"Credit Line"`	`ID = 128`	String Value

HELP BOX
After Dark allows modules to include help or additional information that will not fit in the "Credits" area. The information is displayed when the user clicks in the "Credits" area. No programming is necessary to display text or a picture. To display text, create a resource of type `'TEXT'` with `ID = 1000`. Enter the text you wish displayed into this resource. To display a picture, create a resource of type `'PICT'` with `ID = 1000`. Here are the formats:

`TEXT`	`"Help Text"`	`ID = 1000`	String Value
`PICT`	`"Help Picture"`	`ID = 1000`	Picture

MESSAGES SUPPORTED
This resource allows your module to tell After Dark which messages it wants to receive. While your module should always request the `Initialize`, `Blank`, `DrawFrame`, and `Close` messages, it may or may not want the `ModuleSelected` or `DoHelp` messages (described in the "More About Messages" section). The resource is a byte value and each bit is a flag for a message. Bit 0 is a flag for the `Initialize` message. If it is 1, After Dark will send your module the message. Bit 1 is a flag for the `Blank` message, Bit 2 = `DrawFrame`, Bit 3 = `Close`, Bit 4 = `ModuleSelected`, and Bit 5 = `DoHelp`. Here is its format:

`Cals`	`"Messages Supported"`	`ID = 1`	Byte Value

MEMORY NEEDED
The `'sysz'` resource tells After Dark, Randomizer, and MultiModule the number of bytes of memory a graphics module needs. To determine the amount of memory needed, add the amount of memory allocated for your structure/record, the size of your code resource (the `'ADgm'` resource in your module), and any additional memory your module allocates. On startup, After Dark allocates 30K of memory for graphics modules unless a module has a `'sysz'` resource requesting more. Randomizer and MultiModule use a graphics module's `'sysz'` resource to determine how much memory they should allocate for it. Here is its format:

`sysz`	`"Memory required"`	`ID = 0`	Long word value

The following resources are used to create and manage sliders, menus, text strings, check boxes, and buttons in the Control Panel. A graphics module can have any combination of up to four sliders, menus, text strings, check boxes, and buttons. The resources for controls are numbered 1000, 1001, 1002, and 1003 with 1000 representing the topmost control in the Control Panel, 1001 representing the second control from the top, etc. The current value of each control defined is passed to your graphics module in the `"controlValues"` field of the `"params"` structure.

SLIDERS

To create a slider, create a resource of type `'sVal'`. Its resource `ID` determines where it appears in the Control Panel. The resources name (`"Slider 1"` in this case) is displayed next to the slider in the Control Panel. Slider values range from 0 to 100. The integer value of the `'sVal'` resource is the slider's current value. Here is its format:

```
sVal   "Slider 1"                   ID = 1000          Integer value
```

If only an `'sVal'` resource is created, the numbers 0 to 100 are displayed as the slider is used. If your module needs a different number range or a label ("seconds", "pixels") for its slider, an `'sUnt'` resource can be created to do this. The `'sUnt'` resource has the same ID number as its associated `'sVal'` resource. An `'sUnt'` resource consists of pairs of integers and strings, and it is prefaced by an integer which is a count of the number of pairs. Each pair of integers and strings specifies a string that is to be displayed when the value of the slider is *greater than or equal to* the integer in the pair and *less than* the integer in the next pair. Here is its format:

```
sUnt   "Slider 1"                   ID = 1000          Integer (count) and pairs of
                                                              strings and
integers value.
```

Let's say you wanted a slider that would display "0 sec.", "1 sec.", and "2 sec." as the slider is moved. Create an `'sVal'` resource and an `'sUnt'` resource:

Note: This is a pseudo-resource format. Use ResEdit, RMaker, or MPW Rez to create the actual resource.

```
sVal   "Seconds"                    ID = 1000
       Value = 0

sUnt   "Seconds"                    ID = 1000
       Count = 3
       Lower Limit = 0,             String = "0 sec."
       Lower Limit = 33,            String = "1 sec."
       Lower Limit = 66,            String = "2 sec."
```

The `'sVal'` resource sets the slider to 0. When the slider value is between 0 and 33, the string "0 sec." is displayed under the slider. When the slider value is between 33 and 66, the string "1 sec." is displayed and when the slider value is greater than or equal to 66, the string "2 sec." is displayed. Remember, an integer from *0 to 100* is passed to your module in the `"controlValues"` field as the slider's current value, *not* the string value being displayed or which range of the `'sUnt'` resource you are in. Your module is responsible for interpreting this number in a way that is consistent with what the user sees. If your understanding of this resource is unclear, examine an `'sUnt'` resource in an After Dark module with ResEdit. ResEdit templates for the After Dark resource types are provided. See the section "Resource Templates" at the end of this section for more information about them.

POPUP MENUS

Create an `‘mVal’` resource. The name of the resource (`“Menu 1”` in this case) is displayed next to the popup menu. Its resource `ID` determines where it appears in the Control Panel. The integer value of the resource is the item number of the currently selected item in the popup menu. This value ranges from 1 to the number of items in the menu. The number passed to the graphics module in the `“controlValues”` field is the current menu item selected. Here is its format:

`mVal`	`“Menu 1”`	`ID = 1000`	Integer

A standard `‘MENU’` resource must also be created. It must have the same resource `ID` number as its associated `‘mVal’` resource. This manual does not describe how to build a `‘MENU’` resource. If you are unfamiliar with this process, refer to *Inside Macintosh Volume 1* for more information. Here is its format:

`MENU`	`“Menu 1”`	`ID = 1000`	Standard menu resource

CHECK BOXES

Check boxes are useful when a control only needs a true/false value. When the box is checked, the associated statement is true. For example, if a module can display in color and black & white, a check box could be created with the statement “Draw in color.” To create a check box, create a resource of type `‘xVal’`. The name of the resource is the string displayed next to the check box. Its resource `ID` determines where it appears in the Control Panel. The value of the resource determines if the box is checked or not. If the value is 0 it is not checked. If the value is 1 it is checked. This value is passed to your module in the `“controlValues”` field. Here is its format:

`xVal`	`“Check Box 1”`	`ID = 1000`	Integer value

TEXT STRINGS

Strings of text can also be displayed in the Control Panel. This can be used to inform the user that the module only works on a Macintosh II or that a control is not present because the monitor is not in color. To do this, two resources are created. The first is a standard `‘STR#’` resource. It is simply a list of strings. The second is a resource of type `‘tVal’`. The value of this resource determines which, if any, of the strings in the `‘STR#’` resource appear in the Control Panel. The value ranges from 0 to the number of strings in the `‘STR#’` resource. A value of 0 means no string is displayed in the Control Panel. A value of 1 or greater displays the corresponding numbered string in the `‘STR#’` resource. This value is passed to your module in the `“controlValues”` field. Both resources must have the same `ID` and the resource `ID` determines where the string appears in the Control Panel. Here are their formats:

`STR#`	`“String List 1”`	`ID = 1000`	List of Strings
`tVal`	`“String List 1”`	`ID = 1000`	Integer Value

BUTTONS

To implement a button in a graphics module, create a `‘bVal’` resource. The resource name (`“Button 1”` in this case) appears in the button. Its resource `ID` determines where it appears in the Control Panel. The message value passed to `main()` identifies which button was clicked. This value must be *8 or greater* and *unique* for each button defined. Here is its format:

`bVal`	`“Button 1”`	`ID = 1000`	Integer value

CONTROL DISPLAY

The last control resource allows controls to be conditionally displayed. For example, a button can be displayed only if the Macintosh has Color QuickDraw. Furthermore, a slider could be displayed if the Macintosh has a color monitor and a button could be displayed if it did not. Here is its format:

```
µVal  "Control 1"                     ID = 1000          Integer value
```

To display a control conditionally, create a `'µVal'` resource ('µ' is option - m) with the same `ID` number as the control it is associated with. This resource consists of boolean flags that correspond to the bits in `"systemConfig"` field of the `GMParamBlock` structure/record (see "Graphics Module Parameters" above). The `"systemConfig"` field contains information about the configuration of the Macintosh. If you want a control to be displayed only under a certain configuration, set the appropriate flags in the `'µVal'` resource for that configuration. After Dark compares the `'µVal'` resource to the `"systemConfig"` field. If all the bits set in the `'µVal'` resource are set in the `"systemConfig"` field, After Dark displays the control. For example, if you want a control to be displayed only if Color QuickDraw is present, set the value of the `'µVal'` resource to 1. The `"cqdAvail"` flag is the first (lowest order) bit in `"systemConfig"` field. After Dark checks the first bit in the `"systemConfig"` field. If it is set, the control is displayed.

If you want a different control to appear depending on the configuration, construct the controls (you can have as many controls as there are configurations) and their associated `'µVal'` resources. The resource `ID` for each control differs by 1000. One control is `"ID = 1000"`, the next control is `"ID = 2000"`, etc. For example if you want a `"Choose Color"` button to appear if all the monitors are in color mode and a `"Choose Pattern"` if they are not, define the following resources:

```
bVal  "Choose Color"                  ID = 1000          Value = 8
µVal  "Choose Color"                  ID = 1000          Value = 16
bVal  "Choose Pattern"                ID = 2000          Value = 9
```

The `"Choose Color"` button passes an 8 and the `"Choose Pattern"` button passes a 9 to your `main()` function in your graphics module if they are clicked. The `'µVal'` resource for the `"Choose Color"` button is set to 16. This sets the flag that "all monitors must be in color mode." After Dark checks the `"systemConfig"` field to see if "all the monitors are in color mode" flag is set. If it is, the `"Choose Color"` button is displayed. If it is not, the `"Choose Pattern"` is displayed as the default since it does not have a `'µVal'` resource.

If you are unclear about this use ResEdit on a copy of the `"FadeAway"` graphics module to see an example of a `'µVal'` resource.

If your module wants to use sound, it must have a `'Chnl'` resource with `ID = 0`. This resource consists of two integer values, `"Channel Kind"` and `"Volume"`. The `"Channel Kind"` value tells After Dark what type of sound channel your module wants. This value can be `noteSynth = 1`, `waveTable = 3`, or `sampledSynth = 5`. As described in the section "Using Sound", After Dark currently only supports the sampledSynth type sound channel. The `"Volume"` value is current volume setting of your module. This value can be an integer from 0 to 7. Here is its format:

```
Chnl  "Sound Channel"                 ID = 0             Two integer values
```

Adding color to a graphics module can enhance it greatly. A module may want to use a specific set of colors when it is drawing. This is accomplished by creating a resource of type `'clut'` (For more information about `'clut'` resources, see the Color QuickDraw chapter in *Inside Macintosh Volume 5*.) After Dark sets this resource as the current color lookup table when your module is activated. Since different `'clut'` resources are needed for different monitor pixel depths, a specific `'clut'` resource can be used according to the pixel depth. The resource ID determines which `'clut'` resource is used. If the monitor is set to 1 bit depth, the `'clut'` resource with ID = 1001 is used. If the the monitor is set to 2 bit depth, the `'clut'` resource with ID = 1002 is used. For a bit depth of 4, the resource ID = 1004 and for a bit depth of 8, the resource ID = 1008. Here is its format:

`clut  "1 Bit clut"` `ID = 1001` Standard `'clut'` resource

There are two guidelines when using this resource. A `'clut'` resource should begin with the color white and end with the color black. If this is not done, the Control Panel is displayed with random colors when the module is used in "Demo Mode". Modules should also black the screen in `DoClose()`. This prevents the screen from displaying a "color flash" when the color lookup table is set back the original one.

Since many of the resources used by After Dark are new, resource templates for ResEdit have been created to assist in creating these resources. The file `"After Dark Templates"` has eight resources of type `'TMPL'`. Open this file and paste these resources into your copy of ResEdit. You are now able to create and edit the resources used by After Dark more easily.