This chapter describes how you can create a control panel extension to add a panel to an existing control panel. Some of the control panels provided with the Macintosh system software allow you to install additional panels to control settings for your own devices. You can also install additional panels to allow the user to manipulate other system-wide settings or configuration data not directly associated with any hardware.

You need to read this chapter if you are developing hardware or software that provides system-wide services and that has one or more settings that a user might want to alter. However, you need to read this chapter only if some existing control panel is extensible in the way described in the next section, "About Control Panel Extensions." Currently, only certain versions of the Sound control panel and the Video control panel allow you to add panels by creating control panel extensions. In all other cases, you'll need to create a control panel to handle any necessary user interaction. For a complete description of how to create a control panel, see the chapter "Control Panels" in *Inside Macintosh: More Macintosh Toolbox*. (Also see the chapter "Control Panels" if you are the manufacturer of a video card and need to create an extension to the Monitors control panel.)

To use this chapter, you should already be familiar with creating dialog boxes and handling user actions in them. See the chapters "Dialog Manager" and "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information about these topics. Because control panel extensions are components, you also need to be familiar with the Component Manager, described in *Inside Macintosh: More Macintosh Toolbox*.

Note
The programming interface to control panel extensions described in this chapter is virtually identical to the programming interface to sequence grabber panel components, described in the chapter "Sequence Grabber Panel Components" in *Inside Macintosh: QuickTime Components*. If you are programming in C, you might find it useful to consult the source code samples, which are in C in that chapter. ◆

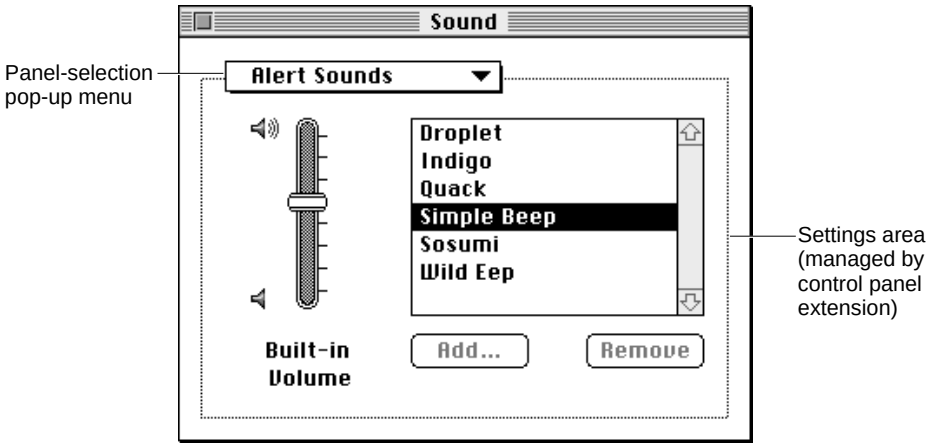
About Control Panel Extensions

A ***control panel*** manages the settings of a system-wide feature, such as the amount of memory allocated to a disk cache, the speed at which the cursor moves relative to movement of the mouse, the background pattern used on the desktop, or the picture displayed by a screen saver. On the screen, a control panel appears as a modeless dialog box with controls that let users specify basic settings and preferences for the feature. A control panel such as the General Controls or Color control panel usually defines the contents of its display area and manages the settings of its own controls; however, a control panel such as the Sound or Video control panel may use one or more control panel extensions to manage parts of its display area. The rest of this chapter discusses control panels that use control panel extensions and describes how to write a control panel extension. For information on control panels that do not use control panel extensions, see the chapter "Control Panels" in *Inside Macintosh: More Macintosh Toolbox*.

A ***control panel extension*** works in conjunction with and at the request of a control panel to manage a certain part of the control panel's display area. The area managed by a control panel extension is called a ***panel***. A panel contains controls and other items related to the features managed by the control panel extension. These items are the same items used in dialog and alert boxes. The control panel extension is responsible for handling events in its panel and for responding to requests from its associated control panel. A control panel that uses control panel extensions typically includes a pop-up menu, from which the user chooses which panel to view. The control panel displays the current panel's items within a dotted-line border extending from its pop-up menu.

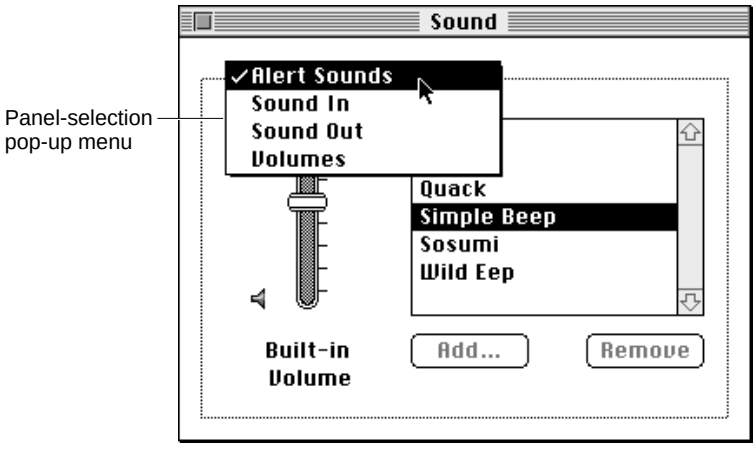
Figure 5-1 shows the Sound control panel introduced with version 3.0 of the Sound Manager. The Sound control panel manages the pop-up menu in its display area. When the user chooses a menu item from the pop-up menu, the Sound control panel uses a control panel extension to display the panel corresponding to the user's choice. The control panel extension is responsible for managing the area within its panel.

Figure 5-1 A control panel with a panel



As shown in Figure 5-1, control panels that use control panel extensions typically include a pop-up menu from which the user can choose one or more items. Each item typically corresponds to a feature managed by a control panel extension. For example, Figure 5-2 shows the menu items in the pop-up menu of the Sound control panel. This pop-up menu can have the items Alert Sounds, Sound In, Sound Out, or Volumes as well as items corresponding to other control panel extensions. Apple supplies the control panel extensions for Alert Sounds, Sound In, Sound Out, and Volumes.

Figure 5-2 Panel-selection pop-up menu in a control panel



As shown in Figure 5-2, when the user chooses the Alert Sounds pop-up menu item, the Sound control panel calls the Alert Sounds control panel extension to display a panel and manage the items associated with the extension. The Alert Sounds control panel extension is responsible for the items within its panel: the volume slider, the scrollable list of sounds, and the two buttons.

The user interface for a panel consists of the display area defined by the owning control panel and includes the items defined and managed by your panel. Each control panel that supports control panel extensions defines the bounding area in which panels can place items. For example, the panel inserted into the Sound control panel is given a default rectangle size of 185 pixels in height, and 302 pixels in width. All of the items for this panel must be placed at least 13 pixels from the dialog's border.

Control panel extensions are implemented as components. A control panel uses the Component Manager to request services from the appropriate control panel extension as needed. For example, when the user opens a control panel, the Finder sends the control panel an initialization request. In response to this request, the control panel uses the Component Manager to determine which control panel extensions are available and includes the name of each available extension in its pop-up menu.

The control panel then uses the Component Manager to open the control panel extension associated with the current pop-up menu item and set up the panel. (For example, if the Sound control panel determines that its panel area should display information for Alert Sounds panel, the Sound control panel opens the Alert Sounds control panel extension.) As directed, the control panel extension returns information about its controls and other items in its panel area and sets initial values for these items. The control panel continues to use the Component Manager to communicate with the control panel extension, requesting it to respond to user events within the panel area. When the user closes the control panel, the control panel uses the Component Manager to close the current control panel extension before the control panel terminates.

This chapter describes the general structure of a control panel extension. For information on providing a control panel extension for a specific control panel, see the documentation describing that control panel. For example, for information on the Video control panel, see the chapter "Sequence Grabber Panel Components" in *Inside Macintosh: QuickTime Components*.

Writing a Control Panel Extension

A control panel extension is a component that works with a control panel to manage a panel—a certain part of an existing control panel's display area. Because a control panel extension is a component, it must be able to respond to standard request codes sent by the Component Manager. In addition, a control panel extension must

- return information about the items in its panel
- handle user actions and other events in its panel
- get and set the values of its items

This section describes how to write a control panel extension. You need to read this section if you want to create a new panel for an existing control panel.

Creating a Component Resource for a Control Panel Extension

A control panel extension is stored as a component resource. It contains a number of resources, including icons, strings, pictures, and the standard component resource (a resource of type `'thng'`) required of any Component Manager component. In addition, a control panel extension must contain code to handle required request codes passed to it by the Component Manager as well as panel-specific request codes. A control panel extension also usually contains an item list resource (`'DITL'`) that defines the items for the panel.

Note
For complete details on components and their structure, see the chapter "Component Manager" in *Inside Macintosh: More Macintosh Toolbox*. This section provides specific information about control panel extensions. ◆

The component resource binds together all the relevant resources contained in a component; its structure is defined by the `ComponentResource` data type.

```
TYPE ComponentResource =
   RECORD
      cd:               ComponentDescription;
      component:        ResourceSpec;
      componentName:    ResourceSpec;
```

```
      componentInfo:    ResourceSpec;
      componentIcon:    ResourceSpec;
   END;
```

The `cd` field contains a component description record that specifies the component type, subtype, manufacturer, and flags. The `component` field specifies the resource type and resource ID of the component's executable code. By convention, this resource should be of the same type as the `componentType` field of the component description record referenced through the `cd` field. (You can, however, specify some other resource type if you wish.) The resource ID can be any integer greater than or equal to 128. See the next section, "Dispatching to Control Panel Extension-Defined Routines," for further information about this code resource. The `ResourceSpec` data type has this structure:

```
TYPE ResourceSpec =
   RECORD
      resourceType:     ResType;
      resourceID:       Integer;
   END;
```

The `componentName` field of the `ResourceSpec` data type specifies the resource type and resource ID of the resource that contains the component's name. Usually the name is contained in a resource of type `'STR '`. This string should be as short as possible.

The `componentInfo` field specifies the resource type and resource ID of the resource that contains a description of the component. Usually the description is contained in a resource of type `'STR '`. This information is not currently used by control panels, but some development tools may use it.

The `componentIcon` field specifies the resource type and resource ID of the resource that contains an icon for the component. Usually the icon is contained in a resource of type `'ICON'`. This icon is not currently used by control panels, but some development tools may use it.

As previously described, the `cd` field of the `ComponentResource` structure is a component description record, which includes additional information about the component. A component description record is defined by the `ComponentDescription` data structure.

```
TYPE ComponentDescription =
   RECORD
      componentType:          LongInt;
      componentSubType:       LongInt;
      componentManufacturer:  LongInt;
      componentFlags:         LongInt;
      componentFlagsMask:     LongInt;
   END;
```

For control panel extensions, the `componentType` field must be set to a value associated with an existing control panel. Currently, you can specify one of two available component types for control panel extensions:

```
CONST
   SoundPanelType             = 'sndP';   {sound panel}
   VideoPanelType             = 'vidP';   {video panel}
```

In addition, the `componentSubType` field must be set to a value that indicates the type of control panel services your panel provides. For example, the Apple-supplied control panel extensions for the Sound control panel have these subtypes:

```
CONST
   kAlertSoundsPanel          = 'alrt';   {alert sounds panel}
   kInputsPanel               = 'mics';   {input devices panel}
   kOutputsPanel              = 'spek';   {output devices panel}
   kVolumesSubType            = 'vols';   {volumes panel}
```

If you add panels to the Sound control panel, you should assign some other subtype.

Note
Apple reserves for its own uses all types and subtypes composed solely of lowercase letters. ◆

You can assign any value you like to the `componentManufacturer` field; typically, you put the signature of your control panel extension in this field.

The `componentFlags` field of the component description for a control panel extension contains bit flags that encode information about the extension. Currently, you can use this field to specify whether the control panel should open your extension's resource file.

```
CONST
   channelFlagDontOpenResFile = 2;     {do not open resource file}
```

The `channelFlagDontOpenResFile` bit indicates to the owning control panel whether or not to open the component's resource file. When bit 2 is cleared (set to 0), the control panel opens the component's resource file for you. In general, this is the most convenient way to gain access to your extension's resources. However, if the component is linked with an application and does not have its own resource file, you might not want the control panel to try to open the resource file. In that case, set this bit to 1.

You should set the `componentFlagsMask` field to 0.

Your control panel extension is contained in a resource file. The creator of the file can be any type you wish, but the type of the file must be `'thng'`. If the extension contains a `'BNDL'` resource, then the file's bundle bit must be set. Control panel extensions should be located in the Control Panels folder (or Extensions folder if the component needs automatic registration).

Listing 5-1 shows the Rez listing of a component resource that describes a control panel extension.

Listing 5-1 A component resource for a control panel extension

```
resource 'thng' (kExamplePanelID, kExampleName, purgeable) {
   kExamplePanelComponentType,   /*component type*/
   kExamplePanelSubType,         /*component subtype*/
   kExampleManufacturer,         /*component manufacturer*/
   cmpWantsRegisterMessage,      /*control flags*/
   0,                            /*control flags mask*/
                                 /*code res type, res ID*/
   kExamplePanelCodeType, kExamplePanelCodeID,
   'STR ', kExamplePanelNameID,  /*name res type, res ID*/
   'STR ', kExamplePanelInfoID,  /*info res type, res ID*/
   'ICON', kExamplePanelIconID   /*icon res type, res ID*/
};
```

Dispatching to Control Panel Extension-Defined Routines

As explained in the previous section, the code stored in the control panel extension component should be contained in a resource whose resource type matches the type stored in the `componentType` field of the component description record. The Component Manager expects that the entry point in this resource is a function having this format:

```
FUNCTION MyPanelDispatch (VAR params: ComponentParameters;
                          storage: Handle): ComponentResult;
```

Whenever the Component Manager receives a request for your control panel extension, it calls your component's entry point and passes any parameters, along with information about the current connection, in a component parameters record. The Component Manager also passes a handle to the global storage (if any) associated with that instance of your component.

When your component receives a request, it should examine the parameters to determine the nature of the request, perform the appropriate processing, set an error code if necessary, and return an appropriate function result to the Component Manager.

The component parameters record is defined by a data structure of type `ComponentParameters`. The `what` field of this record contains a value that specifies the type of request. Your component's entry point should interpret the request code and possibly dispatch to some other subroutine. Your extension must be able to handle the required request codes, defined by these constants:

```
CONST
   kComponentOpenSelect          = -1;
   kComponentCloseSelect         = -2;
   kComponentCanDoSelect         = -3;
   kComponentVersionSelect       = -4;
```

Note
For complete details on required component request codes, see the chapter "Component Manager" in *Inside Macintosh: More Macintosh Toolbox*. ◆

In addition, your extension must be able to respond to panel-specific request codes. Currently, you need to be able to handle these request codes:

```
CONST
   kPanelGetDitlSelect        = 0;  {get panel's item list}
   kPanelGetTitleSelect       = 1;  {get panel's name}
   kPanelInstallSelect        = 2;  {restore item settings}
   kPanelEventSelect          = 3;  {handle event in panel}
   kPanelItemSelect           = 4;  {handle click in a panel item}
   kPanelRemoveSelect         = 5;  {panel is about to be removed}
   kPanelValidateInputSelect  = 6;  {validate panel settings}
   kPanelGetSettingsSelect    = 7;  {get panel settings}
   kPanelSetSettingsSelect    = 8;  {set panel settings}
```

You should respond to these request codes by performing the requested action. To service the request, your component may need to access additional information provided in the `params` field of the component parameters record. The `params` field is an array that contains the parameters specified by the control panel that called your component. You can directly extract the parameters from this array, or you can use the `CallComponentFunction` or `CallComponentFunctionWithStorage` function to extract the parameters from this array and pass these parameters to a subroutine of your component.

Listing 5-2 illustrates how to define the entry-point routine for a control panel extension.

Listing 5-2 Handling Component Manager request codes

```
FUNCTION MyPanelDispatch (VAR params: ComponentParameters; storage: Handle)
                         : ComponentResult;
CONST
   kPanelVersion = 1;
   kExamplePanelDITLID = 128;
   kDefaultButton = 1;
   kExampleOtherButton = 2;
   kExampleBeepButton = 3;
   kExampleRadioButton1 = 4;
   kExampleRadioButton1 = 5;
TYPE
   PanelGlobalsRec =          {global storage for this component instance}
      RECORD
         itemOffset:    Integer;
```

```
         mySelf:        ComponentInstance;
      END;
   PanelGlobalsPtr = ^PanelGlobalsRec;
   PanelGlobalsHandle = ^PanelGlobalsPtr;
VAR
   myGlobals:     PanelGlobalsHandle;
   selector:      Integer;
BEGIN
   CASE params.what OF
      kComponentOpenSelect:                  {component is opening}
         BEGIN
            myGlobals :=
               PanelGlobalsHandle(NewHandleClear(SizeOf(PanelGlobalsRec)));
            IF myGlobals <> NIL THEN
            BEGIN
               myGlobals^^.mySelf := ComponentInstance(params.params[0]);
               SetComponentInstanceStorage(myGlobals^^.mySelf,
                                           Handle(myGlobals));
               MyPanelDispatch := noErr;
            END
            ELSE
               MyPanelDispatch := MemError;
         END;

      kComponentCloseSelect:                 {component is closing; clean up}
         BEGIN
            IF storage <> NIL THEN
               DisposeHandle(storage);
            MyPanelDispatch := noErr;
         END;

      kComponentCanDoSelect:                 {indicate whether component }
                                             { supports this request code}
         BEGIN
            selector := Integer((Ptr(params.params)^));
            IF (((kComponentVersionSelect <= selector)
                  AND (selector <= kComponentOpenSelect))
               OR ((kPanelGetDitlSelect <= selector)
                  AND (selector <= kPanelSetSettingsSelect))) THEN
               MyPanelDispatch := 1 {valid request}
            ELSE
               MyPanelDispatch := 0;{invalid request}
         END;
```

```
kComponentVersionSelect:{return version number}
   MyPanelDispatch := kPanelVersion;

kPanelGetDitlSelect:    {get panel's item list}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelGetDITL));

kPanelInstallSelect:    {restore items' settings if necessary}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelInstall));

kPanelEventSelect:      {handle event in panel}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelEvent));

kPanelItemSelect:       {handle hit in one of panel's items}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelItem));

kPanelRemoveSelect:  {panel is about to be removed, respond as needed}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelRemove));
kPanelValidateInputSelect:{validate panel settings}
   MyPanelDispatch :=
               CallComponentFunctionWithStoMyPanelValidateInputrage
                           (storage, params,
                            ComponentFunction(@MyPanelValidateInput));

kPanelGetTitleSelect:   {get panel's name}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                         ComponentFunction(@MyPanelGetTitle));

kPanelGetSettingsSelect:   {get panel settings}
   MyPanelDispatch := CallComponentFunctionWithStorage
                        (storage, params,
                        ComponentFunction(@MyPanelGetSettings));
```

```
      kPanelSetSettingsSelect:{set panel settings}
         MyPanelDispatch := CallComponentFunctionWithStorage
                              (storage, params,
                              ComponentFunction(@MyPanelSetSettings));

      OTHERWISE               {unrecognized request code}
         MyPanelDispatch := badComponentSelector;
   END; {of CASE}
END;
```

The `MyPanelDispatch` function defined in Listing 5-2 simply inspects the `what` field of the component parameters record to determine which request code to handle. For panel-specific request codes, it dispatches to the appropriate function in the control panel extension. See the following sections for more details on handling panel-specific request codes.

Your extension can be dynamically loaded or unloaded at any time. When the owning control panel first discovers the extension, it loads it into a subheap of some existing heap. In all likelihood, your extension is loaded into either the system heap or temporary memory. In some cases, however, your extension might be loaded into an application's heap. Your extension is guaranteed 32 KB of available heap space. You should do all allocation in that heap using normal Memory Manager routines.

If you need to access resources that are stored in your control panel extension, you can use the `OpenComponentResFile` and `CloseComponentResFile` functions (which are provided by the Component Manager), or you can allow the control panel to open your resource fork for you automatically by setting the appropriate component flag. The `OpenComponentResFile` routine requires the `ComponentInstance` parameter supplied to your routine. You should not call the Resource Manager routines `OpenResFile` or `CloseResFile`.

▲ **WARNING**
Do not leave any resource files open when your control panel extension is closed. Their maps will be left in the subheap when the subheap is freed, causing the Resource Manager to crash. ▲

The following sections illustrate how to write control panel extension functions that respond to panel-specific request codes.

Installing and Removing Panel Items

After opening your control panel extension, the control panel calls your control panel extension with a get-item list request followed by an install request. When your component receives a get-item list request, it should return the item list that defines the items in its panel. When your component receives an install request, it should set the default values of any items in the panel or set up any user items in the panel. For example, your component can restore previous settings as set by the user or create lists

at this time. When your component receives a remove request, it should perform any processing that is necessary before the panel is removed from the display area of the control panel.

A control panel that uses your control panel extension calls your component with the get-item list request (followed by an install request) before displaying the panel to the user. If your component returns a result code of `noErr` in response to both of these request codes, the control panel displays your panel to the user.

The relevant fields in the component parameters record when your component receives a get-item list request are:

Field	Description
`what`	This field is set to `kPanelGetDitlSelect`.
`params`	The first entry in this array contains a handle to a block of memory. Your component should resize the handle as necessary and then use this memory to return an item list of the items supported by your control panel extension.

In response to a get-item list request, set your component's function result to `noErr` if your component successfully placed the item list in memory; otherwise, set it to a nonzero value.

Listing 5-3 shows an example of a control panel extension-defined routine that handles the get-item list request.

Listing 5-3 Responding to the get-item list request

```
FUNCTION MyPanelGetDITL(globals: PanelGlobalsHandle;
                        ditlHandle: Handle): ComponentResult;
BEGIN
   MyPanelGetDITL := resNotFound; {set default return value}
   ditlHandle := Get1Resource('DITL', kExamplePanelDITLID);
   IF (ditlHandle <> NIL)
   BEGIN
      DetachResource(ditlHandle);
      MyPanelGetDITL := noErr;
   END;
END;
```

The relevant fields in the component parameters record when your component receives an install request are:

Field	Description
what	This field is set to kPanelInstallSelect.
params	The first entry in this array contains the dialog pointer of the owning control panel. The dialog box can be a color dialog box on systems that support color windows. The second entry contains the item offset to your panel's first item.

In response to an install request, set your component's function result to noErr if your component successfully handled the request; otherwise, set it to a nonzero value.

Listing 5-4 shows an example of a control panel extension-defined routine that handles the install request.

Listing 5-4 Responding to the install request

```
FUNCTION MyPanelInstall(globals: PanelGlobalsHandle;
                        cpDialogPtr: DialogPtr;
                        itemOffset: Integer): ComponentResult;
BEGIN
   {restore previous settings of panel items as set by user}
   MyPanelInstall := MyRestoreSettings(globals, itemOffset,
                                       cpDialogPtr);
END;
```

The MyPanelInstall function shown in Listing 5-4 calls one of its own routines (MyRestoreSettings) to set the panel's items to the last settings chosen by the user. In response to the install request, you can also create other elements needed by your panel, such as lists.

The relevant fields in the component parameters record when your component receives a remove request are:

Field	Description
what	This field is set to kPanelRemoveSelect.
params	The first entry in this array contains the dialog pointer of the owning control panel. The dialog box can be a color dialog box on systems that support color windows. The second entry contains the item offset to your panel's first item.

In response to a remove request, dispose of any additional dialog data you created (for example, if you created a list, call LDispose), but do not dispose of your component's global storage. Also, set your component's function result to noErr if your component successfully handled the request; otherwise, set it to a nonzero value.

Handling Panel Items

Your control panel extension typically receives an item-select request (indicated by the `kPanelItemSelect` request code) when the user clicks in one of your panel's items. When your component receives an item-select request, it should perform the appropriate action for the selected item.

Note that when a click in one of your panel's items occurs, the owning control panel first sends your component an event-select request, giving your component a chance to filter the event, if necessary. A control panel sends your component an item-select request only if your component returns `FALSE` in the `handled` parameter in response to an event-select request. Typically, your component only returns `FALSE` in response to an event-select request if the event is a mouse event. The event-select request is discussed in detail in the next section, "Handling Events in a Panel" beginning on page 5-17.

The relevant fields in the component parameters record when your component receives an item-select request are:

Field	Description
`what`	This field is set to `kPanelItemSelect`.
`params`	The first entry in this array contains the dialog pointer of the owning control panel. The dialog box can be a color dialog box on systems that support color windows. The second entry contains the item number of the item selected by the user. Note that to map the item number to an item in your panel, you must offset the item number by the number of items in the owning control panel.

You must set your component's function result to `noErr` in response to an item-select request; otherwise, the owning control panel closes the panel.

Listing 5-5 shows an example of a control panel extension-defined routine that handles an item-select request.

Listing 5-5 Responding to an item-select request

```
FUNCTION MyPanelItemSelect(globals: PanelGlobalsHandle;
                           cpDialogPtr: DialogPtr;
                           itemHit: Integer): ComponentResult;
BEGIN
   MyPanelItemSelect := noErr; {set return value}
   {adjust item number to take into account control panel's items}
   itemHit := itemHit - (globals^^).itemsOffset;
   CASE itemHit OF
      kExampleBeepButton:  {user clicked beep button}
         SysBeep(40);
      kExampleOtherButton: {user clicked this button}
         MyPanelItemSelect := MyDoButton(cpDialogPtr, itemHit);
      kExampleRadioButton1:{user clicked this radio button}
```

```
         MyPanelItemSelect := MySetRadioButton(cpDialogPtr,
                                               itemHit);
      kExampleRadioButton2:{user clicked this radio button}
         MyPanelItemSelect := MySetRadioButton(cpDialogPtr,
                                               itemHit);
      kDefaultButton:      {user clicked the default button}
         MyPanelItemSelect :=
                        MyDoDefaultButtonAction(cpDialogPtr,
                                                itemHit);
   END; {of CASE}
END;
```

Handling Events in a Panel

A control panel sends an event-select request (indicated by the `kPanelEventSelect` request code) to your extension whenever an event occurs in your panel. The event-select request is intended to provide your extension with the ability to respond just like an event filter function specified in calls to the `ModalDialog` procedure or other Dialog Manager routines. A control panel sends your extension the event-select request to give it an opportunity to intercept events in its panel and handle events before, or instead of the owning control panel. For example, you can change a keystroke into a click on an item, use idle time during null events, or track the movement of the cursor through mouse events.

The relevant fields in the component parameters record when your component receives an event-select request are:

Field	Description
`what`	This field is set to `kPanelEventSelect`.
`params`	The first entry in this array contains the dialog pointer of the owning control panel. The second entry contains the item offset to your panel's first item. Note that to map the item number to an item in your panel, you must offset the item number by the number of items in the owning control panel. The third entry contains an event record describing the event. If your extension handles the event, it should return in the fourth entry the item number of the associated panel item. On exit, your extension should indicate in the fifth entry whether it has handled the event by returning `TRUE` (handled the event) or `FALSE` (did not handle the event).

When your extension receives an event-select request, it indicates (through the fifth entry in `params`) whether it handled the event or not. Typically, your extension responds to an event-select request in this manner:

- maps the Return or Enter key to the default button, performs the action corresponding to the default button, and returns `TRUE` and the item number of the default button through entries in `params`

- maps the Esc (Escape) key or Command-period combination to the Cancel button (if any), performs the action corresponding to the Cancel button, and returns TRUE and the item number through entries in params
- updates the panel if needed (typically updating only those items that need updating apart from the standard updating performed by the Dialog Manager, such as user-defined panel items or lists) and returns TRUE and the item number of the default button through entries in params
- activates certain panel items (such as lists) as necessary and returns TRUE
- maps keyboard equivalents (if any) to corresponding item numbers, performs the corresponding action for that item number, and returns TRUE
- tracks movement of the cursor as needed (typically tracking the cursor only in those items, such as user-defined items or lists, that the Dialog Manager doesn't handle) and returns TRUE

In general, for all other events, your extension should return FALSE (in the fifth entry of params) and allow the owning control panel to handle the event. However, note that if your extension returns FALSE, the owning control panel calls your extension with the item-select request code. See the previous section, "Handling Panel Items" on page 5-16 for information on handling clicks in your panel's items.

Listing 5-6 shows an example of a control panel extension-defined routine that handles the event-select request.

Listing 5-6 Responding to an event-select request

```
FUNCTION MyPanelEvent (globals: Handle; dialog: DialogPtr;
                       itemOffset: Integer;
                       theEvent: eventRecord;
                       VAR itemHit: Integer;
                       VAR handled: Boolean): ComponentResult;
VAR
   itemType:      Integer;
   itemHandle:    Handle;
   itemRect:      Rect;
   finalTicks:    LongInt;
BEGIN
   MyPanelEvent := noErr;
   CASE theEvent.what OF
      keyDown, autoKey:
      BEGIN
         CASE ((char)(theEvent->message & charCodeMask))
            kEnterKey, kReturnKey:
            BEGIN    {respond as if user clicked Default button}
               itemHit := kDefaultButton + itemOffset;
```

```
                GetDialogItem(dialog, itemHit, itemType,
                              itemHandle, itemRect);
                HiliteControl(ControlHandle(itemHandle),inButton);
                Delay(kVisualDelay, finalTicks);
                HiliteControl(ControlHandle(itemHandle),0);
                MyPanelEvent :=
                        MyDoDefaultButtonAction(dialog, itemHit);
             END;
          OTHERWISE
       {let control panel/Dialog Mgr handle other keyboard events}
             handled := FALSE;
       END; {of CASE keyDown, autoKey}
       updateEvt:
          DoUpdatePanel(globals, dialog);
       OTHERWISE
       {let owning control panel & Dialog Mgr handle other events}
          handled := FALSE;
   END; {of CASE}
END;
```

Handling Title Requests

A control panel may send your control panel extension a title request to determine the name it should display for the panel in the control panel's pop-up menu. Note that a control panel usually uses the name of your component as the name to display.

The relevant fields in the component parameters record when your component receives a title request are:

Field	Description
`what`	This field is set to `kPanelGetTitleSelect`.
`params`	The first entry in this array contains a value that identifies a specific instance of your component. In the second entry of this array, your component should return the name you want displayed in the pop-up menu associated with your panel.

Note
Current versions of the Sound and Video control panels do not send the `kPanelGetTitleSelect` request code. ◆

Managing Control Panel Settings

A control panel may send the `kPanelValidateInputSelect`, `kPanelGetSettingsSelect`, or `kPanelSetSettingsSelect` request codes to your extension to request it to validate the settings of its items, or return or set the current

settings of its items. If a control panel sends this request code, your extension should respond appropriately.

Note
Current versions of the Sound and Video control panels do not send the `kPanelValidateInputSelect`, `kPanelGetSettingsSelect`, or `kPanelSetSettingsSelect` request code. ◆

Control Panel Extensions Reference

This section describes the extension-defined routines that you can write to handle the panel-specific request codes that your control panel extension receives. See "Writing a Control Panel Extension" beginning on page 5-6 for information on creating a component that contains these extension-defined routines.

Control Panel Extension-Defined Routines

This section describes the routines you'll need to define in order to write a control panel extension. You need to write routines that respond to panel-specific request codes. The panel-specific request codes request your control panel extension to perform various actions. These actions include:

- returning an item list describing the panel's items and setting up the initial values of these items
- receiving and handling events in the panel
- getting and setting a panel's settings

Your control panel extension-defined routines should always return result codes of type `ComponentResult`. If a routine succeeds, it should return `noErr`.

See "Dispatching to Control Panel Extension-Defined Routines" beginning on page 5-9 for a description of how you call these routines from within a control panel extension.

Managing Panel Components

A control panel extension should respond to the `kPanelGetDitlSelect`, `kPanelInstallSelect`, `kPanelGetTitleSelect`, and `kPanelRemoveSelect` request codes. You typically define subroutines that the main program of your control panel extension calls (using `CallComponentFunctionWithStorage`) to handle these requests. You can choose any name for these subroutines, but by convention they're called `MyPanelGetDITL`, `MyPanelInstall`, `MyPanelGetTitle`, and `MyPanelRemove`.

When the appropriate control panel prepares to add a control panel extension's items to a control panel, it obtains a list of those items by calling the extension and specifying the

kPanelGetDitlSelect request code. The control panel extension typically responds by calling a subroutine (for example, MyPanelGetDITL) to handle the request. Once the control panel has installed the items, it calls the extension and specifies the kPanelInstallSelect request code to give the extension the opportunity to set any default values in the panel. The extension's MyPanelInstall function responds to this request code.

Before the control panel removes the panel from its display, it calls the extension and specifies the kPanelRemoveSelect request code. The extension's MyPanelRemove function responds to this request code. The kPanelGetTitleSelect request code is currently optional for control panel extensions. If your extension responds to this request code, it should return the name that the control panel should display for the panel in the control panel's pop-up menu. The extension's MyPanelGetTitle function responds to this request code.

MyPanelGetDITL

A control panel extension must respond to the kPanelGetDitlSelect request code. A control panel sends this request code to an extension to obtain a list of the panel's items. A control panel extension typically responds to the kPanelGetDitlSelect request code by calling an extension-defined subroutine (for example, MyPanelGetDITL) to handle the request.

```
FUNCTION MyPanelGetDITL (globals: Handle; VAR ditl: Handle)
                         : ComponentResult;
```

globals	A handle to the control panel extension's global data.
ditl	On entry, a handle to a block of memory in your application heap. On exit, a handle to an item list.

DESCRIPTION

Your MyPanelGetDITL function should return, through the ditl parameter, an item list of the items supported by your extension. The control panel then places those items into the control panel and, after installing the panel, displays the panel to the user. When the control panel creates the panel, it places the items at the locations specified in the item list.

On entry to your MyPanelGetDITL function, the ditl parameter contains a handle to a block of memory in your application heap. You should resize the handle as necessary to hold the item list you return to the control panel. (If you use a Resource Manager routine such as Get1Resource, the Resource Manager automatically resizes the handle for you.)

In general, the owning control panel disposes of the handle you pass it once it's finished constructing the panel. As a result, you must make sure that the handle you pass to the control panel is not a resource handle. If you obtain your item list by reading it into memory from a resource, you should call the Resource Manager's DetachResource

procedure to convert that resource handle into one that is suitable for use with the `MyPanelGetDITL` function.

The `componentFlags` field of the component description record for a control panel extension contains a bit flag, `channelFlagDontOpenResFile`, that indicates whether the control panel should open your extension's resource file before calling your extension.

Set the `channelFlagDontOpenResFile` component flag to 0 if you want the control panel to open your extension's resource file before calling your extension. Set the `channelFlagDontOpenResFile` component flag to 1 to specify that the control panel should not open your extension's resource file before calling your extension.

RESULT CODES

Your `MyPanelGetDITL` function should return `noErr` if successful, or an appropriate result code otherwise.

SEE ALSO

For an example of the `MyPanelGetDITL` function, see Listing 5-3 on page 5-14.

MyPanelInstall

A control panel extension must respond to the `kPanelInstallSelect` request code. A control panel sends this request code to an extension immediately after sending the `kPanelGetDitlSelect` request code (which initially adds your panels's items to the control panel) and just before displaying the panel to the user. A control panel extension typically responds to the `kPanelInstallSelect` request code by calling an extension-defined subroutine (for example, `MyPanelInstall`) to handle the request.

```
FUNCTION MyPanelInstall (globals: Handle; dialog: DialogPtr;
                         itemOffset: Integer): ComponentResult;
```

`globals` A handle to the control panel extension's global data.

`dialog` A pointer to the dialog record of the owning control panel. The owning control panel displays your panel's items in the dialog box referenced through this parameter.

`itemOffset`
An offset to the panel's first item.

DESCRIPTION

Your `MyPanelInstall` function should perform any processing that must occur after the panel is created but before it is displayed to the user. For example, your

MyPanelInstall function can set or restore default values of various items in the panel. You can also use this opportunity to create user items (such as lists) in the panel.

The itemOffset parameter specifies the offset from 1 to the first item in your panel. The items installed by your control panel extension are contained in a larger dialog box containing other items; as a result, if you call the GetDialogItem procedure to obtain a handle to an item, you need to increment the itemNo parameter passed to GetDialogItem by the value of itemOffset.

In most cases, you'll need to save the value passed in the itemOffset parameter in your extension's global storage for later use. For example, you usually need this value to determine which panel item the user selected when your extension responds to the kPanelItemSelect request code.

The value passed to your MyPanelInstall function in the itemOffset parameter may be different each time MyPanelInstall is called. You should not assume it is always the same value.

RESULT CODES

Your MyPanelInstall function should return noErr if successful, or an appropriate result code otherwise.

SEE ALSO

For an example of the MyPanelInstall function, see Listing 5-4 on page 5-15.

MyPanelGetTitle

A control panel extension should respond to the kPanelGetTitleSelect request code but is not required to do so. A control panel sends this request code to your extension to get the name of your panel extension. A control panel extension typically responds to the kPanelGetTitleSelect request code by calling an extension-defined subroutine (for example, MyPanelGetTitle) to handle the request.

```
FUNCTION MyPanelGetTitle (self: ComponentInstance; title: Str255)
                         : ComponentResult;
```

self	A component instance identifying the specific instance of your control panel extension.
title	On exit, the name of your control panel extension as you want it to appear in the panel-selection pop-up menu of the control panel.

DESCRIPTION

Your `MyPanelGetTitle` function should return, through the `title` parameter, a string that is the desired title of your control panel extension. This name appears as a menu item in the pop-up menu that lets the user select which panel to view.

SPECIAL CONSIDERATIONS

Currently, all control panels use the component name as the title of the control panel extension. The `MyPanelGetTitle` function is intended to allow your extension to assign a title different from the component name. Future control panels are likely to call your `MyPanelGetTitle` function.

RESULT CODES

Your `MyPanelGetTitle` function should return `noErr` if successful, or an appropriate result code otherwise.

MyPanelRemove

A control panel extension must respond to the `kPanelRemoveSelect` request code. A control panel sends this request code to an extension just before removing the panel from the enclosing dialog box. A control panel extension typically responds to the `kPanelRemoveSelect` request code by calling an extension-defined subroutine (for example, `MyPanelRemove`) to handle the request.

```
FUNCTION MyPanelRemove (globals: Handle; dialog: DialogPtr;
                        itemOffset: Integer): ComponentResult;
```

`globals` A handle to the control panel extension's global data.

`dialog` A pointer to the dialog record of the owning control panel.

`itemOffset`
An offset to the panel's first item.

DESCRIPTION

Your `MyPanelRemove` function should perform any processing that must occur before your panel is removed from the enclosing dialog box. For example, your `MyPanelRemove` function can save the current values of any items in the dialog box. You can also use this opportunity to dispose of any user items (such as lists) in the dialog box. If the control panel opened your component's resource file, that file is still open at the time `MyPanelRemove` is called.

The `itemOffset` parameter specifies the offset from 1 to the first item in your control panel. The dialog items installed by your control panel extension are contained in a larger dialog box containing other items; as a result, if you call the `GetDialogItem`

procedure to obtain a handle to a dialog item, you need to increment the `itemNo` parameter passed to `GetDialogItem` by the value of `itemOffset`.

The value passed to your `MyPanelRemove` function in the `itemOffset` parameter may be different each time `MyPanelRemove` is called. You should not assume it is always the same value.

RESULT CODES

Your `MyPanelRemove` function should return `noErr` if successful, or an appropriate result code otherwise.

Handling Panel Events

A control panel extension should respond to the `kPanelItemSelect` and `kPanelEventSelect` request codes. You typically define subroutines that the main program of your control panel extension calls (using the `CallComponentFunctionWithStorage` function) to handle these requests. You can choose any name for these subroutines, but by convention they're called `MyPanelItem` and `MyPanelEvent`. These two routines should respond to mouse clicks and other events in the items of the panel.

MyPanelItem

A control panel extension must respond to the `kPanelItemSelect` request code. In general, a control panel sends this request code to your extension whenever the user clicks an item in your panel. A control panel extension typically responds to the `kPanelItemSelect` request code by calling an extension-defined subroutine (for example, `MyPanelItem`) to handle the request.

```
FUNCTION MyPanelItem (globals: Handle; dialog: DialogPtr;
                      itemOffset: Integer; itemNum: Integer)
                      : ComponentResult;
```

`globals` A handle to the control panel extension's global data.

`dialog` A pointer to the dialog record of the owning control panel. The owning control panel displays your panel's items in the dialog box (of the control panel) referenced through this parameter.

`itemOffset`
An offset to the panel's first item.

`itemNum` The item number of the item selected by the user. This item number is an index into the list of items in the dialog box. To map this value to the item list you passed to the control panel (in the `MyPanelGetDITL` function), you need to compensate for the offset reported in the `itemOffset` parameter.

DESCRIPTION

Your `MyPanelItem` function should handle mouse clicks on specific items in your panel. The owning control panel calls your control panel extension with the `kPanelItemSelect` whenever your component returns `FALSE` in response to an event-select request. Your `MyPanelItem` function is therefore typically invoked each time the user clicks on some item in your panel. Your function should respond appropriately, according to the item that was clicked.

As just described, note that when a click in one of your panel's items occurs, the owning control panel first sends your component an event-select request, giving your component a chance to filter the event, if necessary. In this case, if your component returns `FALSE` in the `handled` parameter, then the control panel sends your component the item-select request code; if your component returns `TRUE` in the `handled` parameter, the control panel does not send your component the subsequent item-select request code.

RESULT CODES

Your `MyPanelItem` function should return `noErr` if successful, or an appropriate result code otherwise.

SEE ALSO

For an example of the `MyPanelItem` function, see Listing 5-5 on page 5-16. For information on responding to events, see the description of the `MyPanelEvent` function in the next section.

MyPanelEvent

A control panel extension must respond to the `kPanelEventSelect` request code. A control panel sends this request code to your extension whenever an event occurs in your panel. A control panel extension typically responds to the `kPanelEventSelect` request code by calling an extension-defined subroutine (for example, `MyPanelEvent`) to handle the request.

```
FUNCTION MyPanelEvent (globals: Handle; dialog: DialogPtr;
                       itemOffset: Integer;
                       theEvent: eventRecord;
                       VAR itemHit: Integer;
                       VAR handled: Boolean): ComponentResult;
```

`globals` A handle to the control panel extension's global data.

`dialog` A pointer to the dialog record of the owning control panel. The owning control panel displays your items in the dialog box (of the control panel) referenced through this parameter.

`itemOffset`
: An offset to the panel's first item.

`theEvent`
: An event record describing the event being reported to your control panel extension.

`itemHit`
: On entry, the item number of an item. This number is valid only for mouse events (on input, do not interpret this parameter for any other type of event). On exit, if the `MyPanelEvent` function has handled the event, it should return the item number of the associated item in this parameter.

`handled`
: On entry, the value `FALSE` for mouse events; the value `TRUE` for all other events. On exit, the `MyPanelEvent` function should return a Boolean value that indicates whether it has handled the event (`TRUE`) or has not handled the event (`FALSE`).

DESCRIPTION

Your `MyPanelEvent` function is called whenever an event occurs in your panel. The parameter `theEvent` contains a complete description of the event. A control panel handles events in its own items and also gives your component a chance to handle events in its own panel.

The `MyPanelEvent` function is intended to operate just like an event filter function specified in calls to the `ModalDialog` procedure or other Dialog Manager routines. The main difference between `MyPanelEvent` and other event filter functions is that `MyPanelEvent` does not return a Boolean value as its function result. Instead, it indicates whether it handled the event in the `handled` parameter.

If the specified event is a mouse event, you might prefer your extension's `MyPanelItem` function to handle the event. In that case, you should return `FALSE` in the `handled` parameter. Otherwise, you should attempt to handle the event.

If your `MyPanelEvent` function does handle the event, it should update the `itemHit` parameter to reflect the affected item and return `TRUE` in the `handled` parameter. If you set `handled` to `FALSE`, the owning control panel sends your panel an item-select request.

RESULT CODES

Your `MyPanelEvent` function should return `noErr` if successful, or an appropriate result code otherwise.

SEE ALSO

For an example `MyPanelEvent` function, see Listing 5-6 on page 5-18. See the description of `MyPanelItem` on page 5-25 for information on handling clicks in dialog items. For a description of the fields of the event record, see the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Managing Panel Settings

A control panel extension should respond to the `kPanelValidateInputSelect`, `kPanelGetSettingsSelect`, and `kPanelSetSettingsSelect` request codes. You typically define subroutines that the main program of your control panel extension calls (using the routine `CallComponentFunctionWithStorage`) to handle these requests. You can choose any name for these subroutines, but by convention they're called `MyPanelValidateInput`, `MyPanelGetSettings`, and `MyPanelSetSettings`. These routines should manage item settings in a panel.

Note
Current versions of the Sound and Video control panels do not send the `kPanelValidateInputSelect`, `kPanelGetSettingsSelect`, or `kPanelSetSettingsSelect` request code. ◆

MyPanelValidateInput

A control panel extension must respond to the `kPanelValidateInputSelect` request code. A control panel sends this request code to your extension whenever the user clicks a control panel's close box. A control panel extension typically responds to the `kPanelValidateInputSelect` request code by calling an extension-defined subroutine (for example, `MyPanelValidateInput`) to handle the request.

```
FUNCTION MyPanelValidateInput (globals: Handle; VAR ok: Boolean)
                               : ComponentResult;
```

`globals`	A handle to the control panel extension's global data.
`ok`	On return, a Boolean value that indicates whether the panel's current values are valid (`TRUE`) or invalid (`FALSE`).

DESCRIPTION

Your `MyPanelValidateInput` function should perform any processing necessary to validate the current settings in the panel. For example, if your panel contains any editable text items, you might need to ensure that the text they contain makes sense. The control panel calls this function when the user clicks the control panel's close box.

If the current settings of the panel items are acceptable, set the `ok` parameter to `TRUE` before returning from `MyPanelValidateInput`. If the current settings are not valid, set `ok` to `FALSE`. When you set `ok` to `FALSE`, the control panel ignores any of the user's subsequent clicks in the panel's OK button.

RESULT CODES

Your `MyPanelValidateInput` function should return `noErr` if successful, or an appropriate result code otherwise.

MyPanelGetSettings

A control panel extension must respond to the `kPanelGetSettingsSelect` request code. A control panel sends this request code to your extension to get the panel's current settings. A control panel extension typically responds to the `kPanelGetSettingsSelect` request code by calling an extension-defined subroutine (for example, `MyPanelGetSettings`) to handle the request.

```
FUNCTION MyPanelGetSettings (globals: Handle; VAR ud: UserData;
                             flags: LongInt): ComponentResult;
```

`globals`	A handle to the control panel extension's global data.
`ud`	A handle to the control panel's configuration data.
`flags`	Reserved. This parameter is always 0.

DESCRIPTION

Your `MyPanelGetSettings` function should return, through the `ud` parameter, a copy of the panel's current settings. This copy is maintained privately by the control panel. The control panel may subsequently restore your panel's settings by passing those settings to your `MyPanelSetSettings` function.

Your control panel extension is responsible for allocating storage for the configuration data to which `ud` is a handle. You might do that when the Component Manager passes your extension the `kComponentOpenSelect` parameter. Your extension should not dispose of that storage until it closes (that is, when the Component Manager passes it the `kComponentCloseSelect` parameter).

You can arrange the panel configuration data in any way you like. The data needs to contain whatever information is necessary for your `MyPanelSetSetting` function to set all relevant panel items to specified values. For example, the standard Apple sound panels save information such as the component type of the default sound output device, the current volumes levels, the current alert beep, and so forth. You might want to begin the configuration data with a version number so that you can easily change the format of the rest of the data, if necessary.

The information you return to the control panel may get stored as part of the owner's configuration information and might therefore persist across system restarts. As a result, you should not store values that might change without the control panel's knowledge (such as component ID numbers, file reference numbers, and similar volatile information).

RESULT CODES

Your `MyPanelGetSettings` function should return `noErr` if successful, or an appropriate result code otherwise.

MyPanelSetSettings

A control panel extension must respond to the `kPanelSetSettingsSelect` request code. A control panel sends this request code to your extension to request that your extension set the panel's current settings to the specified values. A control panel extension typically responds to the `kPanelSetSettingsSelect` request code by calling an extension-defined subroutine (for example, `MyPanelSetSettings`) to handle the request.

```
FUNCTION MyPanelSetSettings (globals: Handle; ud: UserData;
                             flags: LongInt): ComponentResult;
```

`globals`	A handle to the control panel extension's global data.
`ud`	A handle to the control panel's configuration data.
`flags`	Reserved. This parameter is always 0.

DESCRIPTION

Your `MyPanelSetSettings` function should parse the block of configuration data passed in the `ud` parameter and set the values of the items in the panel based on that data. The control panel calls this function just before your panel is displayed to the user and whenever a user cancels changes to your panel. You can assume that the data passed in the `ud` parameter was created by a previous call to your extension's `MyPanelGetSetting` function.

It's possible that your extension might not able to set the value of one or more panel items to the values specified in the configuration data. (For example, the hardware environment might have changed since the configuration data was last stored by the control panel.) When this happens, you should try to match the specified panel settings as closely as possible. If you cannot match perfectly, you should return some nonzero result code.

RESULT CODES

Your `MyPanelSetSettings` function should return `noErr` if successful, or an appropriate result code otherwise.

Summary of Control Panel Extensions

Pascal Summary

Constants

```
CONST
   {component types}
   SoundPanelType               = 'sndP';   {sound panel}
   VideoPanelType               = 'vidP';   {video panel}

   {component subtypes}
   kAlertSoundsPanel            = 'alrt';   {alert sounds panel}
   kInputsPanel                 = 'mics';   {input devices panel}
   kOutputsPanel                = 'spek';   {output devices panel}
   kVolumesSubType              = 'vols';   {volumes panel}

   {component flags}
   channelFlagDontOpenResFile   = 2;        {do not open resource file}

   {Component Manager request codes for routines}
   kPanelGetDitlSelect          = 0;        {get panel's item list}
   kPanelGetTitleSelect         = 1;        {get panel's name}
   kPanelInstallSelect          = 2;        {restore item settings}
   kPanelEventSelect            = 3;        {handle event in panel}
   kPanelItemSelect             = 4;        {handle click in a panel item}
   kPanelRemoveSelect           = 5;        {panel is about to be removed}
   kPanelValidateInputSelect    = 6;        {validate panel settings}
   kPanelGetSettingsSelect      = 7;        {get panel settings}
   kPanelSetSettingsSelect      = 8;        {set panel settings}
```

Control Panel Extension-Defined Routines

Managing Panel Components

```
FUNCTION MyPanelGetDITL     (globals: Handle; VAR ditl: Handle)
                             : ComponentResult;
```

```
FUNCTION MyPanelInstall     (globals: Handle; dialog: DialogPtr;
                             itemOffset: Integer): ComponentResult;
FUNCTION MyPanelGetTitle    (self: ComponentInstance; title: Str255)
                             : ComponentResult;
FUNCTION MyPanelRemove      (globals: Handle; dialog: DialogPtr;
                             itemOffset: Integer): ComponentResult;
```

Handling Panel Events

```
FUNCTION MyPanelItem        (globals: Handle; dialog: DialogPtr;
                             itemOffset: Integer; itemNum: Integer)
                             : ComponentResult;
FUNCTION MyPanelEvent       (globals: Handle; dialog: DialogPtr;
                             itemOffset: Integer; theEvent: eventRecord;
                             VAR itemHit: Integer; VAR handled: Boolean)
                             : ComponentResult;
```

Managing Panel Settings

```
FUNCTION MyPanelValidateInput
                            (globals: Handle; VAR ok: Boolean)
                             : ComponentResult;
FUNCTION MyPanelGetSettings (globals: Handle; VAR ud: UserData;
                             flags: LongInt): ComponentResult;
FUNCTION MyPanelSetSettings (globals: Handle; ud: UserData;
                             flags: LongInt): ComponentResult;
```

C Summary

Constants

```
/*component types*/
#define SoundPanelType          'sndP'   /*sound panel*/
#define VideoPanelType          'vidP'   /*video panel*/

/*component subtypes*/
#define kAlertSoundsPanel       'alrt'   /*alert sounds panel*/
#define kInputsPanel            'mics'   /*input devices panel*/
#define kOutputsPanel           'spek'   /*output devices panel*/
#define kVolumesSubType         'vols'   /*volumes panel*/
```

```
/*component flags*/
enum {
   channelFlagDontOpenResFile    = 2      /*do not open resource file*/
};

/*Component Manager request codes for routines*/
enum {
   kPanelGetDitlSelect           = 0,     /*get panel's item list*/
   kPanelGetTitleSelect,                  /*get panel's name*/
   kPanelInstallSelect,                   /*restore item settings*/
   kPanelEventSelect,                     /*handle event in panel*/
   kPanelItemSelect,                      /*handle click in a panel item*/
   kPanelRemoveSelect,                    /*panel is about to be removed*/
   kPanelValidateInputSelect,             /*validate panel settings*/
   kPanelGetSettingsSelect,               /*get panel settings*/
   kPanelSetSettingsSelect                /*set panel settings*/
};
```

Control Panel Extension-Defined Routines

Managing Panel Components

```
pascal ComponentResult MyPanelGetDITL
                           (Handle globals, Handle *ditl);
pascal ComponentResult MyPanelInstall
                           (Handle globals, DialogPtr dialog,
                            short itemOffset);
pascal ComponentResult MyPanelGetTitle
                           (ComponentInstance self, StringPtr title);
pascal ComponentResult MyPanelRemove
                           (Handle globals, DialogPtr dialog,
                            short itemOffset);
```

Handling Panel Events

```
pascal ComponentResult MyPanelItem
                           (Handle globals, DialogPtr dialog,
                            short itemOffset, short itemNum);
pascal ComponentResult MyPanelEvent
                           (Handle globals, DialogPtr dialog,
                            short itemOffset, eventRecord *theEvent,
                            short *itemHit, Boolean *handled);
```

Managing Panel Settings

```
pascal ComponentResult MyPanelValidateInput
                            (Handle globals, Boolean *ok);
pascal ComponentResult MyPanelGetSettings
                            (Handle globals, UserData *ud, long flags);
pascal ComponentResult MyPanelSetSettings
                            (Handle globals, UserData *ud, long flags);
```