

Movie Controller Components

This chapter describes movie controller components. Movie controller components provide a high-level interface that allows your application to present movies to users quickly and easily. **Movie controllers**, the controls managed by movie controller components, present a user interface for playing and editing movies. Movie controller components eliminate much of the complexity of working with movies by assuming primary responsibility for the movie, freeing your application to focus on the unique services it offers to users.

This chapter has been divided into the following sections:

- “About Movie Controller Components” describes the capabilities of movie controller components in general and discusses the movie controller component supplied by Apple.
- “Spatial Properties” discusses the display regions that are supported by movie controller components—your application can manipulate these regions to control how the controller is displayed.
- “Using Movie Controller Components” provides sample code that shows you how to play, edit, and customize movies with movie controller components.
- “Movie Controller Components Reference” describes the functions provided to your application by movie controller components.
- “Summary of Movie Controller Components” provides a condensed listing of the constants, data structures, and functions supported by these components.

If you are developing an application that can play movies, you should consider using movie controller components to manage your movie user interface. They provide a consistent user interface that shields you from the details of using the Movie Toolbox. To learn about the capabilities of movie controllers, read “About Movie Controller Components.” If your application allows the user to play movies, read “Spatial Properties.” If you anticipate doing event management, read “Customizing Movie Controllers” beginning on page 2-13 and “Application-Defined Function” beginning on page 2-61 as well. All movie controller functions are described in “Movie Controller Components Reference”—you should read the portions that are relevant to your application.

If you are developing a movie controller component, the information in this chapter describes the interface that your component must support. In addition, you should be familiar with the material in the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox*, which describes how to build a component.

About Movie Controller Components

Movie controller components provide movie playback and editing capabilities to applications. In so doing, movie controller components remove from your application much of the burden of presenting an interface for movie playback and editing. It is possible to have the controller do nearly all the work involved with playing movies, including updating and idling. Alternatively, your application can take care of some or all of these tasks.

You can think of movie controller components in terms of more familiar Macintosh controls. Movie controller components, in addition to handling update, activate, and mouse-down events, also know how to interact with the data that they control. Consequently, the movie controller components can actually perform the commands requested by users (the controls handled by the Control Manager merely report user actions to your application). In this way, your application is relieved of much of the work of controlling movies. Furthermore, movie controller components can be updated to provide improved functionality with no impact on your application.

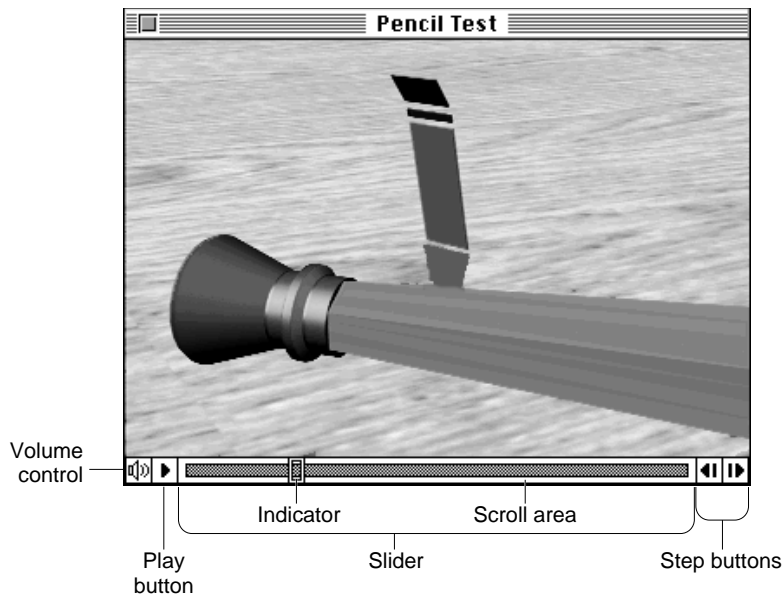
Movie controller components have a component type value of 'play'. You can use the following constant to specify this value.

```
#define MovieControllerComponentType 'play'
```

Apple has defined the functional interface that is supported by movie controller components so that you can create a wide variety of movie controls. For example, you could create a control that is separate from the movie image. Consequently, the interface is a bit more complex than might seem necessary for simple controls that support only playback. For details on the functions that your component must support, see “Movie Controller Components Reference,” which begins on page 2-14.

The Elements of a Movie Controller

The movie controller component provided with QuickTime by Apple provides control elements for regulating sound, starting, stopping, pausing, single-stepping (forward and backward), and moving to a specified time. Figure 2-1 shows the controls supported by Apple's movie controller component. If the user resizes the controller so that there is not enough space to display all the individual control elements, the movie controller component eliminates elements from the display. Note that this controller allows the user to start and stop the movie by clicking the movie image itself. This is an important feature, because it allows the user to control the movie even in circumstances where no control elements are visible.

Figure 2-1 The standard movie controller

The movie controller presented by Apple's movie controller component contains a number of individual controls, as shown in Figure 2-1. These controls include:

- **A volume control.** This control allows the user to adjust the sound volume—holding down the mouse button while the cursor is on this control causes the controller to display a slider that allows the user to change the sound volume while the movie is playing (if a movie does not have any sound, the movie controller component disables the volume control).
- **A play button.** This control allows the user to start and stop the movie. Clicking the play button causes the movie to start playing; in addition, the movie controller component changes the play button into a pause button. Clicking the pause button causes the movie to stop playing. If the user starts the movie and does not stop it, the movie controller plays the movie once and then stops the movie.
- **A slider.** This control allows the user to quickly navigate through a movie's contents. Dragging the indicator within the slider displays a single frame of the movie that corresponds to the position of the indicator. Clicking within the slider causes the indicator to jump to the location of the mouse click and causes the movie controller component to display the corresponding movie data.
- **Step buttons.** These controls allow the user to move through the movie frame by frame, either forward or backward. Holding the mouse button down while the cursor is on a step button causes the movie controller to step through the movie, frame by frame, in the appropriate direction.

Badges

The movie controller component supplied by Apple allows your application to distinguish movies from static graphics in documents by the use of a badge. A **badge** is a visual element that the movie controller can display as part of a movie when the other controls are not visible and the movie is not playing. Figure 2-2 shows a movie with a badge.

Figure 2-2 A movie with a badge



The badge lets the user know that the image represents a movie rather than a static image. A badge appears under the following conditions:

- the movie is in badge mode—that is, the `mcActionSetUseBadge` movie controller action was called with a value of `true`
- the movie is not playing
- the movie controller is hidden

When the user double-clicks the movie, the movie starts playing and the badge disappears; a single click stops the movie, and the badge reappears. When the user clicks the badge itself, the movie controller component displays the controls, as shown in Figure 2-1.

Your application can control whether the movie controller component displays a badge with a movie. Use the `NewMovieController` function (described on page 2-28) to create a new controller.

Spatial Properties

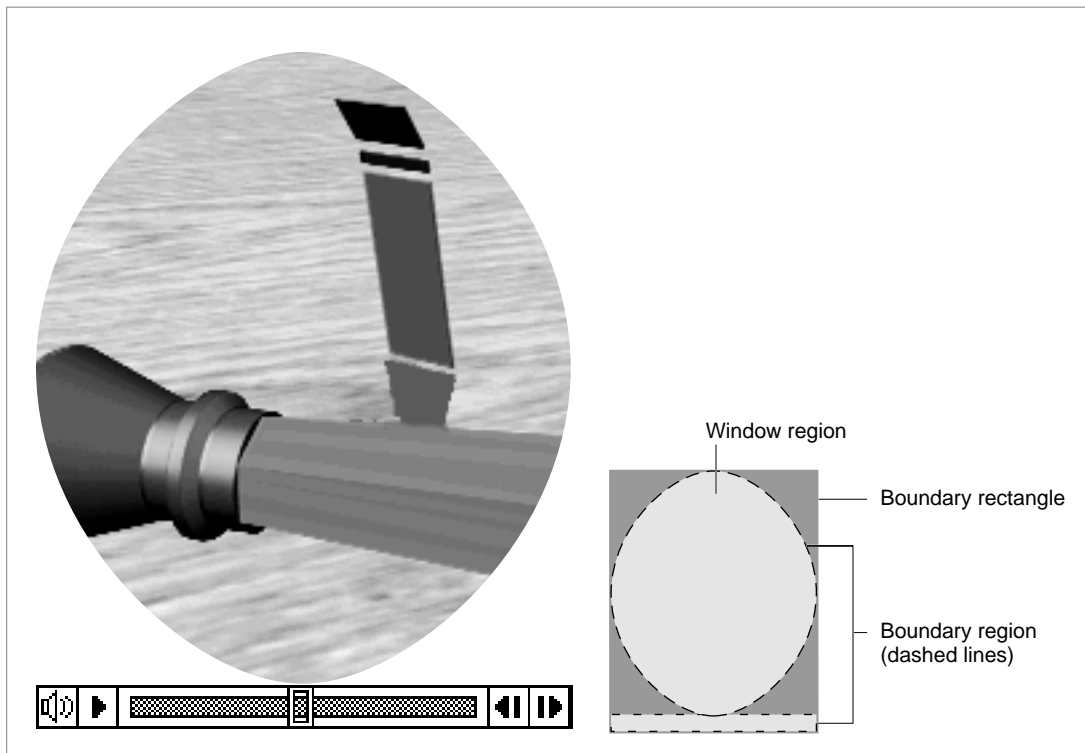
Movie controller components define several display regions that govern how a controller and its movie are displayed. In addition, movie controller components support a number

of functions that allow your application to manipulate these regions and thereby control the display of a controller and its associated movie. This section discusses each of these regions and the movie controller component functions that your application can use to work with these regions.

The displayed representation of a movie controller consists of two parts: the movie and the controller itself. The movie consists of the QuickTime movie image. The controller consists of the visual elements that allow the user to control the movie. Figure 2-1 on page 2-5 shows a sample controller. In this figure, note that the movie is attached to the controller—that is, the movie and the controller are contiguous. Movie controller components also allow you to create controllers that are separate from, or detached from, their associated movies. You use the `MCSetControllerAttached` function (described on page 2-34) to control this attribute. This gives you the freedom to position the movie and the controller.

Movie controller components define several spatial elements that allow your application to control the display of a movie and its controller. Figure 2-3 shows the relationships between these spatial elements for **attached controllers**, whereas Figure 2-4 shows the relationships between these spatial elements for **detached controllers**.

Figure 2-3 Movie controller spatial elements for attached controllers

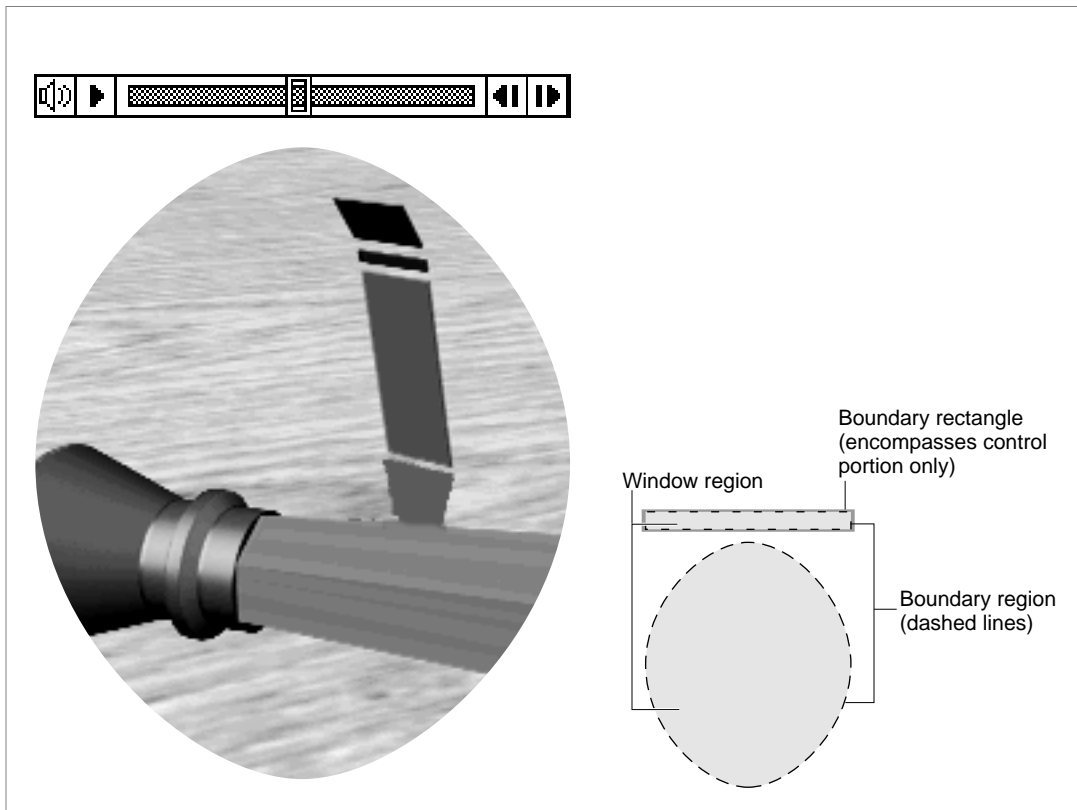


Movie Controller Components

The **controller boundary rectangle** is a rectangle that completely encloses the controller. If the controller is attached to its movie, the controller boundary rectangle also encloses the movie. The width of this rectangle corresponds to the widest part of the displayed representation of the controller (and its attached movie). Similarly, its height is derived from the highest part of the controller (and its attached movie). You can use the `MCSetControllerBoundsRect` function to modify the controller boundary rectangle to define display transformations to be applied to a controller and its movie. You can retrieve a controller's boundary rectangle by calling the `MCGetControllerBoundsRect` function (described on page 2-39).

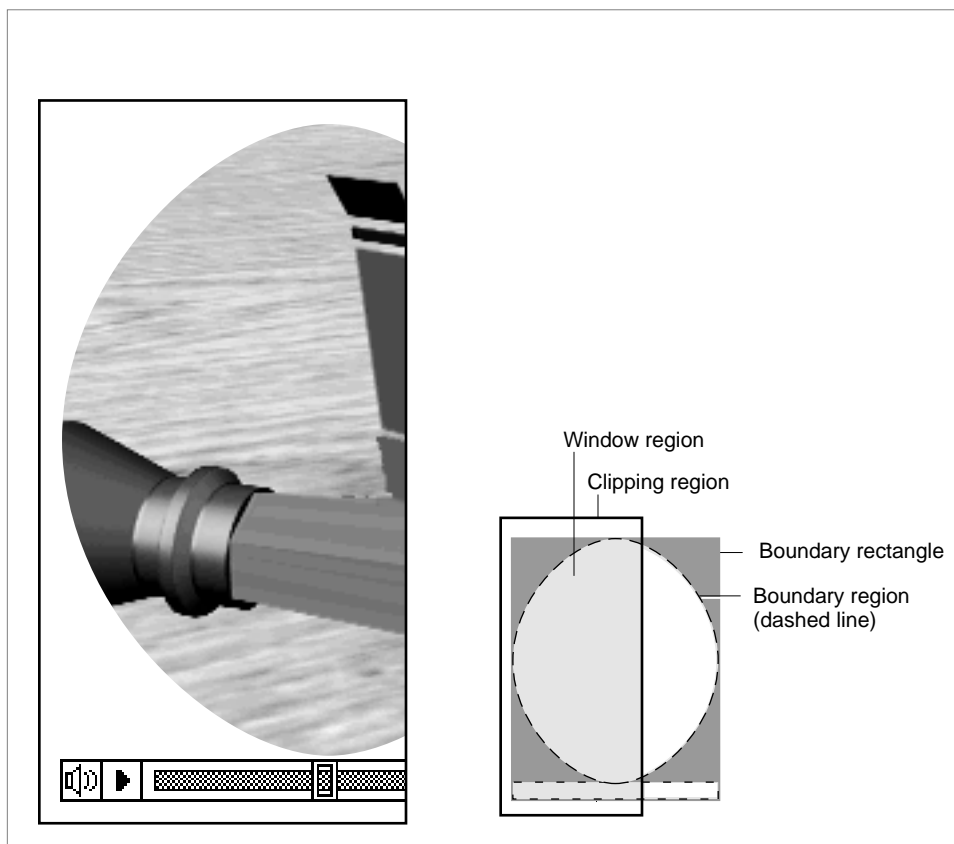
The **controller boundary region** defines the region occupied by the controller. If the movie is attached to the controller, the controller boundary region also includes the movie. The controller boundary region corresponds exactly to the display footprint of the controller (and its attached movie). You can retrieve the boundary region of a controller by calling the `MCGetControllerBoundsRgn` function (described on page 2-40).

Figure 2-4 Movie controller spatial elements for detached controllers



The controller boundary rectangle and controller boundary region both work with the unclipped display representation of the controller and its movie. The **controller window region** represents the portion of the controller and its movie that is actually displayed on the computer screen, after clipping by the **controller clipping region**. The controller window region always includes both the controller and its movie, whether the controller is attached or detached. You can retrieve a controller's window region by calling the `MCGetWindowRgn` function (described on page 2-41). You can manipulate a controller's clipping region by calling the `MCSetClip` and `MCGetClip` functions (described on page 2-42 and page 2-43, respectively). Figure 2-5 shows how the controller clipping region affects the controller window region.

Figure 2-5 Clipping the controller window region with the controller clipping region



Using Movie Controller Components

This section supplies examples of how to use the standard movie controller to play movies. It also provides sample code for customizing movie controller components.

Playing Movies

The following sample code demonstrates how to use the standard movie controller component to play a movie. The `GetMovie` function prompts the user to select a movie file and then get a movie out of it. It then opens the movie and allows the user to play it.

Listing 2-1 Playing a movie with a movie controller component

```
MovieController    gController;
WindowPtr          gWindow;
Rect               windowRect;
Movie              gMovie;
Boolean            gDone;
OSErr              gErr;
ComponentResult    gCErr;
Boolean            gResult;
EventRecord        gTheEvent;
WindowPtr          whichWindow;
short              part;

pascal Movie       GetMovie(void);
pascal Movie       GetMovie(void)
{
    OSErr            err;
    SFTypelist        typeList;
    StandardFileReply reply;
    Movie             aMovie;
    short             movieResFile;
    short             movieResID;
    Str255            movieName;
    Boolean           wasChanged;

    aMovie = nil;
    typeList[0] = MovieFileType;
    StandardGetFilePreview ( (FileFilterProcPtr)nil, 1,
                            typeList, &reply);
```


Movie Controller Components

```

    if (reply.sfGood) {
        err = OpenMovieFile (&reply.sfFile, &movieResFile,
                               fsRdPerm);

        if (err == noErr) {
            movieResID = 0;
            err = NewMovieFromFile (&aMovie, movieResFile,
                                    &movieResID,
                                    movieName,
                                    newMovieActive,
                                    &wasChanged);

            err = CloseMovieFile (movieResFile);
        }
    }
    return aMovie;
}

void main(void);
void main(void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(nil);
    gErr = EnterMovies();

    SetRect (&windowRect, 100, 100, 200, 200);
    gWindow = NewCWindow (nil,
                          &windowRect,
                          "\pMovie",
                          false,
                          noGrowDocProc,
                          (WindowPtr)-1,
                          true,
                          0);
    SetPort (gWindow);
    gMovie = GetMovie();
    if (gMovie != nil) {
        SetRect(&windowRect, 0, 0, 100, 100);
        gController = NewMovieController (gMovie, &windowRect,
                                           mcTopLeftMovie);
    }
}

```

Movie Controller Components

```

if (gController != nil) {
    gCErr = MCGetControllerBoundsRect (gController,
                                       &windowRect);
    SizeWindow (gWindow, windowRect.right, windowRect.bottom,
               true);
    ShowWindow (gWindow);
    gCErr = MCDoAction (gController, mcActionSetKeysEnabled,
                       (Ptr>true);
    gDone = false;

    while (! gDone) {
        gResult = GetNextEvent (everyEvent, &gTheEvent);
        if (MCIsPlayerEvent (gController, &gTheEvent) == 0) {
            switch (gTheEvent.what) {
                case updateEvt:
                    whichWindow = (WindowPtr)gTheEvent.message;
                    BeginUpdate (whichWindow);
                    EraseRect (&(*whichWindow).portRect);
                    EndUpdate (whichWindow);
                    break;
                case mouseDown:
                    part = FindWindow (gTheEvent.where,
                                       &whichWindow);
                    if (whichWindow == gWindow) {
                        switch (part) {
                            case inGoAway:
                                gDone = TrackGoAway (whichWindow,
                                                       gTheEvent.where);
                                break;

                            case inDrag:
                                DragWindow (whichWindow,
                                             gTheEvent.where,
                                             &(qd.screenBits.bounds) );
                                break;

                        }
                    }
            }
        }
    }
    DisposeMovieController(gController);
}
DisposeMovie(gMovie);

```

```

    }
    DisposeWindow(gWindow);
}

```

Customizing Movie Controllers

Movie controller components allow you to create an action filter function in your application. The component calls your action filter function whenever an action occurs in the control. (An **action** is an integer constant used by the movie controller component.) You can then customize the behavior of the control or simply monitor user actions. You establish an action filter function by calling the `MCSetActionFilterWithRefCon` function, which is described on page 2-47.

The sample code in Listing 2-2 demonstrates the use of an action filter function. This filter function resizes the window whenever the user hides the controller. Therefore, this sample function handles the `mcActionControllerSizeChanged` action. Your application should include a similar action filter function so that you can determine when the user resizes the controller. This function supports only attached controllers.

Listing 2-2 Using a movie controller filter function

```

pascal Boolean myMCActionFilter ( MovieController mc,
                                short* Action, long* params);

{
    RgnHandle    controllerRgn;
    Rect         controllerBox;
    WindowPtr    movieWindow;
    switch (*Action) {
        case mcActionControllerSizeChanged:
            /* size of controller/movie has changed */
            movieWindow = (WindowPtr)MCGetControllerPort(mc);
            controllerRgn = MCGetWindowRgn(mc, movieWindow);
            if (controllerRgn != nil) {
                controllerBox = (**controllerRgn).rgnBBox;
                DisposeRgn (controllerRgn);
                SizeWindow (movieWindow, controllerBox.right,
                           controllerBox.bottom, true);
            }
            break;
    }
    return false;
}

```

Movie Controller Components Reference

This section describes some of the constants and functions associated with movie controller components.

You can use the following constants to refer to the request codes for each of the functions that your movie controller component must support.

```
enum {
    kMCSetMovieSelect          = 2,  /* MCSetMovie */
    kMCRemoveMovieSelect      = 3,  /* MCRemoveMovie */
    kMCIsPlayerEventSelect    = 7,  /* MCIsPlayerEvent */
    kMCSetActionFilterSelect  = 8,  /* MCSetActionFilter */
    kMCDoActionSelect         = 9,  /* MCDoAction */
    kMCSetControllerAttachedSelect = 10,
                                /* MCSetControllerAttached */
    kMCIsControllerAttachedSelect = 11,
                                /* MCIsControllerAttached */
    kMCSetControllerPortSelect = 12, /* MCSetControllerPort */
    kMCGetControllerPortSelect = 13, /* MCGetControllerPort */
    kMCGetVisibleSelect       = 14, /* MCGetVisible */
    kMCSetVisibleSelect       = 15, /* MCSetVisible */
    kMCGetControllerBoundsRectSelect
                                = 16,
                                /* MCGetControllerBoundsRect */
    kMCSetControllerBoundsRectSelect
                                = 17,
                                /* MCSetControllerBoundsRect */
    kMCGetControllerBoundsRgnSelect = 18,
                                /* MCGetControllerBoundsRgn */
    kMCGetWindowRgnSelect         = 19, /* MCGetWindowRgn */
    kMCMovieChangedSelect        = 20, /* MCMovieChanged */
    kMCSetDurationSelect         = 21, /* MCSetDuration */
    kMCGetCurrentTimeSelect      = 22, /* MCGetCurrentTime */
    kMCNewAttachedControllerSelect = 23,
                                /* MCNewAttachedController */
    kMCDrawSelect                = 24, /* MCDraw */
    kMCActivateSelect            = 25, /* MCActivate */
    kMCIdleSelect                = 26, /* MCIdle */
    kMCKeySelect                 = 27, /* MCKey */
    kMCClickSelect               = 28, /* MCClick */
    kMCEnableEditingSelect       = 29, /* MCEnableEditing */
}
```

Movie Controller Components

```

    kMCIsEditingEnabledSelect      = 30, /* MCIsEditingEnabled */
    kMCCopySelect                  = 31, /* MCCopy */
    kMCCutSelect                   = 32, /* MCCut */
    kMCPasteSelect                 = 33, /* MCPaste */
    kMCClearSelect                 = 34, /* MCClear */
    kMCUndoSelect                  = 35, /* MCUndo */
    kMCPositionControllerSelect    = 36,
                                   /* MCPositionController */
    kMCGetControllerInfoSelect      = 37,
                                   /* MCGetControllerInfo */
    kMCSetClipSelect               = 40, /* MCSetClip */
    kMCGetClipSelect               = 41, /* MCGetClip */
    kMCDrawBadgeSelect             = 42  /* MCDrawBadge */
    kMCSetUpEditMenuSelect         = 43, /* MCSetUpEditMenu */,
    kMCGetMenuStringSelect         = 44, /* MCGetMenuString */
    kMCSetActionFilterWithRefConSelect = 45
                                   /* MCSetActionFilterWithRefCon */
};

```

Movie Controller Actions

This section discusses actions, which are integer constants (defined by the `mcAction` data type) used by movie controller components. Applications that use movie controller components can invoke these actions by calling the `MCDoAction` function, which is described on page 2-46. If your application includes an action filter function, that function may receive any of these actions (see the discussion of the `MCSetActionFilterWithRefCon` function on page 2-47 for more information about action filter functions).

Your action filter function should refer any actions that you do not want to handle back to the calling movie controller component. Your function refers actions back to the movie controller component by returning a value of `false`. If your function returns a value of `true`, the movie controller component performs no further processing for the action.

If you use any Movie Toolbox functions that modify the movie in your action filter function, be sure to call the `MCMovieChanged` function (described on page 2-49).

```

enum {
    mcActionIdle                  = 1,  /* give event-processing time to
                                         movie controller */
    mcActionDraw                  = 2,  /* send update event to movie
                                         controller */
    mcActionActivate              = 3,  /* activate movie controller */
    mcActionDeactivate           = 4,  /* deactivate controller */
    mcActionMouseDown            = 5,  /* pass mouse-down event */
};

```

Movie Controller Components

mcActionKey	= 6, /* pass key-down or auto-key event */
mcActionPlay	= 8, /* start playing movie */
mcActionGoToTime	= 12, /* move to specific time in a movie */
mcActionSetVolume	= 14, /* set a movie's volume */
mcActionGetVolume	= 15, /* retrieve a movie's volume */
mcActionStep	= 18, /* play a movie a specified number of frames at a time */
mcActionSetLooping	= 21, /* enable or disable looping */
mcActionGetLooping	= 22, /* find out if movie is looping */
mcActionSetLoopIsPalindrome	= 23, /* enable palindrome looping */
mcActionGetLoopIsPalindrome	= 24, /* find out if palindrome looping is on */
mcActionSetGrowBoxBounds	= 25, /* set limits for resizing a movie */
mcActionControllerSizeChanged	= 26, /* user has resized movie controller */
mcActionSetSelectionBegin	= 29, /* start time of movie's current selection */
mcActionSetSelectionDuration	= 30, /* set duration of movie's current selection */
mcActionSetKeysEnabled	= 32, /* enable or disable keystrokes for movie */
mcActionGetKeysEnabled	= 33, /* find out if keystrokes are enabled */
mcActionSetPlaySelection	= 34, /* constrain playing to the current selection */
mcActionGetPlaySelection	= 35, /* find out if movie is constrained to playing within selection */
mcActionSetUseBadge	= 36, /* enable or disable movie's playback badge */
mcActionGetUseBadge	= 37, /* find out if movie controller is using playback badge */
mcActionSetFlags	= 38, /* set movie's control flags */
mcActionGetFlags	= 39, /* retrieve movie's control flags */
mcActionSetPlayEveryFrame	= 40, /* instruct controller to play all frames in movie */
mcActionGetPlayEveryFrame	= 41, /* find out if controller is playing every frame in movie */
mcActionGetPlayRate	= 42, /* determine playback rate */
mcActionShowBalloon	= 43, /* find out if controller wants to display Balloon Help */

Movie Controller Components

```

mcActionBadgeClick          = 44, /* user clicked movie's badge */
mcActionMovieClick          = 45, /* user clicked in movie */
mcActionSuspend              = 46, /* suspend event received */
mcActionResume               = 47 /* resume event received */
typedef short mcAction;
};

```

The action descriptions that follow are divided into those used by your application and those received by your action filter.

Actions for Use by Applications**mcActionIdle**

Your application can use this action to grant event-processing time to a movie controller.

There are no parameters for this action.

mcActionDraw

Your application can use this action to send an update event to a movie controller.

The parameter for this action is a pointer to a window.

mcActionActivate

Your application can use this action to activate a movie controller.

There are no parameters for this action.

mcActionDeactivate

Your application can use this action to deactivate a movie controller.

There are no parameters for this action.

mcActionMouseDown

Your application can use this action to pass a mouse-down event to a movie controller.

The parameter data must contain a pointer to an event structure—the message field in the event structure must specify the window in which the user clicked.

mcActionKey

Your application can use this action to pass a key-down or auto-key event to a movie controller.

The parameter data must contain a pointer to an event structure that describes the key event.

Your action filter function receives this action when the movie controller has received a key-down or auto-key event.

mcActionPlay

Your application can use this action to start or stop playing a movie.

The parameter data must contain a fixed value that indicates the rate of play. Values greater than 0 correspond to forward rates; values less than 0 play the movie backward. A value of 0 stops the movie.

Movie Controller Components

`mcActionGotoTime`

Your application can use this action to move to a specific time in a movie.

The parameter data must contain a pointer to a time structure that specifies the target position in the movie.

`mcActionSetVolume`

Your application can use this action to set a movie's volume.

The parameter data must contain a pointer to a 16-bit, fixed-point number that indicates the relative volume of the movie. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`mcActionGetVolume`

Your application can use this action to determine a movie's volume.

The parameter data must contain a pointer to a 16-bit, fixed-point number that indicates the relative volume of the movie. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`mcActionStep`

Your application can use this action to play a movie while skipping a specified number of frames at a time.

The parameter data must contain a long integer value that specifies the number of steps (that is, the frames and the play direction). Positive values step the movie forward the specified number of frames; negative values step the movie backward. A value of 0 steps the movie forward one frame.

`mcActionSetLooping`

Your application can use this action to enable or disable looping for a movie.

The parameter data must contain a Boolean value—a value of `true` indicates that looping is to be enabled.

`mcActionGetLooping`

Your application can use this action to determine whether a movie is looping.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetLoopIsPalindrome`

Your application can use this action to enable palindrome looping.

Palindrome looping causes a movie to play alternately forward and backward. Looping must also be enabled for palindrome looping to take effect.

The parameter data must contain a Boolean value—a value of `true` indicates that palindrome looping is to be enabled.

`mcActionGetLoopIsPalindrome`

Your application can use this action to determine whether palindrome looping is enabled for a movie. Looping must also be enabled for palindrome looping to take effect.

Movie Controller Components

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if palindrome looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetGrowBoxBounds`

Your application can use this action to set the limits for resizing a movie.

The parameter data consists of a `rect` structure.

`mcActionSetSelectionBegin`

Your application can use this action to set the start time of a movie's current selection. After using this action, you must use the `mcActionSetSelectionDuration` action to set the duration of the selection.

The parameter data must contain a pointer to a time structure specifying the starting time of the movie's current selection.

`mcActionSetSelectionDuration`

Your application can use this action to set the duration of a movie's current selection. You can only use this action immediately after the `mcActionSetSelectionBegin` action.

The parameter data must contain a pointer to a time structure specifying the ending time of the movie's current selection.

Your action filter function receives this action when the movie controller has received a request to set the movie's current selection duration.

`mcActionSetKeysEnabled`

Your application can use this action to enable or disable keystrokes for a movie.

The parameter data must contain a Boolean value—a value of `true` indicates that keystrokes are to be enabled. By default, this value is set to `false`.

`mcActionGetKeysEnabled`

Your application can use this action to determine whether keystrokes are enabled for a movie controller.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if keystrokes are enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetPlaySelection`

Your application can use this action to constrain playing to the current selection.

The parameter data must contain a Boolean value—a value of `true` indicates that playing within the current selection is to be enabled.

`mcActionGetPlaySelection`

Your application can use this action to determine whether a movie has been constrained to playing within its selection.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if playing is constrained to the current selection. Otherwise, it sets the value to `false`.

Movie Controller Components

`mcActionSetUseBadge`

Your application can use this action to enable or disable a movie's playback badge. If a controller's badge is enabled, then the badge is displayed whenever the controller is not visible. When the controller is visible, the badge is not displayed. If the badge is disabled, the badge is never displayed.

The parameter data must contain a Boolean value—a value of `true` indicates that the playback badge is to be enabled.

`mcActionGetUseBadge`

Your application can use this action to determine whether a controller is using a badge. If a controller's badge is enabled, then the badge is displayed whenever the controller is not visible. When the controller is visible, the badge is not displayed. If the badge is disabled, the badge is never displayed.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if the controller is using a badge. Otherwise, it sets the value to `false`.

`mcActionSetFlags`

Your application can use this action to set a movie's control flags.

The parameter data must contain a long integer that contains the new control flag values. The following flags are defined:

`mcFlagSuppressMovieFrame`

Controls whether the controller displays a frame around the movie. If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

`mcFlagSuppressStepButtons`

Controls whether the controller displays the step buttons. The step buttons allow the user to step the movie forward or backward a frame at a time. If this flag is set to 1, the controller does not display the step buttons. By default, this flag is set to 0.

`mcFlagSuppressSpeakerButton`

Controls whether the controller displays the speaker button. The speaker button allows the user to control the movie's sound. If this flag is set to 1, the controller does not display the speaker button. By default, this flag is set to 0.

`mcActionGetFlags`

Your application can use this action to retrieve a movie's control flags.

The parameter data must contain a pointer to a long integer. The movie controller places the movie's control flags into that long integer. The following movie control flags are defined:

`mcFlagSuppressMovieFrame`

Controls whether the controller displays a frame around the movie. If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

Movie Controller Components

`mcFlagSuppressStepButtons`

Controls whether the controller displays the step buttons. The step buttons allow the user to step the movie forward or backward a frame at a time. If this flag is set to 1, the movie controller does not display the step buttons. By default, this flag is set to 0.

`mcFlagSuppressSpeakerButton`

Controls whether the controller displays the speaker button. The speaker button allows the user to control the movie's sound. If this flag is set to 1, the movie controller does not display the speaker button. By default, this flag is set to 0.

`mcFlagsUseWindowPalette`

Controls whether the controller manages the palette for the window containing the movie. This ensures that a movie's colors are reproduced as accurately as possible. This flag is particularly useful for movies with custom color tables. If this flag is set to 1, the movie controller does not manage the window palette. By default, this flag is set to 0.

`mcActionSetPlayEveryFrame`

Your application can use this action to instruct the movie controller to play every frame in a movie. In this case, the movie controller may play the movie at a slower rate than you specify with the `mcActionPlay` action. However, the controller does not play the movie faster than the movie rate. In addition, the controller does not play the movie's sound tracks.

The parameter data must contain a Boolean value—a value of `true` instructs the controller to play every frame in the movie, even if that means playing the movie at a slower rate than you previously specified.

`mcActionGetPlayEveryFrame`

Your application can use this action to determine whether the movie controller has been instructed to play every frame in a movie. You tell the controller to play every frame by using the `mcActionSetPlayEveryFrame` action, which is described earlier in this section.

The parameter data must contain a pointer to a Boolean value—the movie controller sets this value to `true` if the controller has been instructed to play every frame in the movie, even if that means playing the movie at a slower rate than you previously specified. Otherwise, the controller sets the value to `false`.

Movie Controller Components

`mcActionSetGrowBoundsBox`

The parameter data must contain a pointer to a rectangle—set the rectangle to the boundary coordinates for the movie. If you want to prevent the movie from being resized, supply an empty rectangle (note that enabling or disabling the size box may change the appearance of some movie controllers). By default, movie controllers do not have size boxes. You must use this action to establish a size box for a movie controller.

If the movie controller's boundary rectangle intersects the lower-right corner of your window, your window cannot have a size box.

`mcActionGetPlayRate`

Your application can use this action to determine a movie's playback rate. You set the playback rate when you start a movie playing by using the `mcActionPlay` action.

The parameter data must contain a pointer to a fixed value. The movie controller returns the movie's playback rate in that fixed value. Values greater than 0 correspond to forward rates; values less than 0 play the movie backward. A value of 0 indicates that the movie is stopped.

`mcActionBadgeClick`

Indicates that the badge was clicked. The parameter is a pointer to a Boolean value. On entry, the Boolean is set to `true`. Set the Boolean to `false` if you want the controller to ignore the click in the badge.

`mcActionMovieClick`

Indicates that the movie was clicked. The parameter is a pointer to an event structure containing the mouse-down event. If you want the controller to ignore the mouse-down event, change the `what` field of the event structure to a null event.

`mcActionSuspend`

Indicates that a suspend event has been received. There is no parameter.

`mcActionResume`

Indicates that a resume event has been received. There is no parameter.

Actions for Use by Action-Filter Functions`mcActionIdle`

Your action filter function receives this action when the application has granted null event-processing time to the movie controller.

There are no parameters for this action.

`mcActionDraw`

Your filter function receives this action when the controller has received an update event.

The parameter for this action is a pointer to a window.

`mcActionActivate`

Your filter function receives this action when the controller has received an activate or resume event.

There are no parameters for this action.

Movie Controller Components

`mcActionDeactivate`

Your filter function receives this action when the controller has received a deactivate or suspend event.

There are no parameters for this action.

`mcActionMouseDown`

Your action filter function receives this action when the movie controller has received a mouse-down event.

The parameter data must contain a pointer to an event structure—the `message` field in the event structure must specify the window in which the user clicked.

`mcActionKey`

Your action filter function receives this action when the movie controller has received a key-down or auto-key event.

The parameter data must contain a pointer to an event structure that describes the key event.

`mcActionPlay`

Your action filter receives this action when the movie controller has received a request to start or stop playing a movie.

The parameter data must contain a fixed value that indicates the rate of play. Values greater than 0 correspond to forward rates; values less than 0 play the movie backward. A value of 0 stops the movie.

`mcActionGotoTime`

Your action filter function receives this action when the movie controller has received a request to go to a specified time in the movie.

The parameter data must contain a pointer to a time structure that specifies the target position in the movie.

`mcActionSetVolume`

Your action filter function receives this action when the movie controller has received a request to set the movie's volume.

The parameter data must contain a pointer to a 16-bit, fixed-point number that indicates the relative volume of the movie. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`mcActionGetVolume`

Your action filter function receives this action when the movie controller has received a request to retrieve the movie's volume.

The parameter data must contain a pointer to a 16-bit, fixed-point number that indicates the relative volume of the movie. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

Movie Controller Components

`mcActionStep`

Your action filter function receives this action when the movie controller has received a request to play a movie while advancing a specified number of frames at a time.

The parameter data must contain a long integer value that specifies the number of steps (that is, the frames and the play direction). Positive values step the movie forward the specified number of frames; negative values step the movie backward. A value of 0 steps the movie forward one frame.

`mcActionSetLooping`

Your action filter function receives this action when the movie controller has received a request to turn looping on or off.

The parameter data must contain a Boolean value—a value of `true` indicates that looping is to be enabled.

`mcActionGetLooping`

Your action filter function receives this action when the controller has received a request to indicate whether looping is enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetLoopIsPalindrome`

Your action filter function receives this action when the movie controller has received a request to turn palindrome looping on or off. Palindrome looping causes a movie to play alternately forward and backward. Looping must also be enabled for palindrome looping to take effect.

The parameter data must contain a Boolean value—a value of `true` indicates that palindrome looping is to be enabled.

`mcActionGetLoopIsPalindrome`

Your action filter function receives this action when the controller has received a request to indicate whether palindrome looping is enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if palindrome looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionControllerSizeChanged`

Your filter function receives this action when the user has resized the movie controller—the controller component issues this action before it updates the screen, allowing your application to change the controller's location or appearance before the user sees the resized controller.

There are no parameters for this action.

Note

Your application should never use this action. ♦

Movie Controller Components

`mcActionSetSelectionBegin`

Your action filter function receives this action when the movie controller has received a request to set the movie's current selection start time.

The parameter data must contain a pointer to a time structure specifying the starting time of the movie's current selection.

`mcActionSetSelectionDuration`

Your action filter function receives this action when the movie controller has received a request to set the movie's current selection duration.

The parameter data must contain a pointer to a time structure specifying the ending time of the movie's current selection.

`mcActionSetKeysEnabled`

Your action filter function receives this action when the movie controller has received a request to enable or disable keystrokes.

The parameter data must contain a Boolean value—a value of `true` indicates that keystrokes are to be enabled. By default, this value is set to `false`.

`mcActionGetKeysEnabled`

Your filter function receives this action when the controller has received a request to indicate whether keystrokes are enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if keystrokes are enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetPlaySelection`

Your action filter function receives this action when the movie controller has received a request to constrain playing to the current selection.

The parameter data must contain a Boolean value—a value of `true` indicates that playing within the current selection is to be enabled.

`mcActionGetPlaySelection`

Your action filter function receives this action when the movie controller has received a request to indicate whether playing is constrained to the current selection.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if playing is constrained to the current selection. Otherwise, it sets the value to `false`.

`mcActionSetUseBadge`

Your action filter function receives this action when the movie controller has received a request to turn the playback badge on or off.

The parameter data must contain a Boolean value—a value of `true` indicates that the playback badge is to be enabled.

`mcActionGetUseBadge`

Your action filter function receives this action when the controller has received a request to indicate whether it is using a badge during playback.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if the controller is using a badge. Otherwise, it sets the value to `false`.

Movie Controller Components

`mcActionSetFlags`

Your action filter function receives this action when the movie controller has received a request to set the movie's control flags.

The parameter data must contain a long integer that contains the new control flag values. The following flags are defined:

`mcFlagSuppressMovieFrame`

Controls whether the controller displays a frame around the movie. If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

`mcFlagSuppressStepButtons`

Controls whether the controller displays the step buttons. The step buttons allow the user to step the movie forward or backward a frame at a time. If this flag is set to 1, the controller does not display the step buttons. By default, this flag is set to 0.

`mcFlagSuppressSpeakerButton`

Controls whether the controller displays the speaker button. The speaker button allows the user to control the movie's sound. If this flag is set to 1, the controller does not display the speaker button. By default, this flag is set to 0.

`mcActionGetFlags`

Your action filter function receives this action when the movie controller has received a request to retrieve the movie's control flags.

The parameter data must contain a pointer to a long integer. The movie controller places the movie's control flags into that long integer. The following movie control flags are defined:

`mcFlagSuppressMovieFrame`

Controls whether the controller displays a frame around the movie. If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

`mcFlagSuppressStepButtons`

Controls whether the controller displays the step buttons. The step buttons allow the user to step the movie forward or backward a frame at a time. If this flag is set to 1, the movie controller does not display the step buttons. By default, this flag is set to 0.

`mcFlagSuppressSpeakerButton`

Controls whether the controller displays the speaker button. The speaker button allows the user to control the movie's sound. If this flag is set to 1, the movie controller does not display the speaker button. By default, this flag is set to 0.

Movie Controller Components

`mcFlagsUseWindowPalette`

Controls whether the controller manages the palette for the window containing the movie. This ensures that a movie's colors are reproduced as accurately as possible. This flag is particularly useful for movies with custom color tables. If this flag is set to 1, the movie controller does not manage the window palette. By default, this flag is set to 0.

`mcActionSetPlayEveryFrame`

Your action filter function receives this action when the movie controller has received a request to play every frame in a movie.

The parameter data must contain a Boolean value—a value of `true` instructs the controller to play every frame in the movie, even if that means playing the movie at a slower rate than you previously specified.

`mcActionGetPlayEveryFrame`

Your action filter function receives this action when the movie controller has received a request to indicate whether it has been instructed to play every frame in a movie.

The parameter data must contain a pointer to a Boolean value—the movie controller sets this value to `true` if the controller has been instructed to play every frame in the movie, even if that means playing the movie at a slower rate than you previously specified. Otherwise, the controller sets the value to `false`.

`mcActionSetGrowBoundsBox`

Your action filter function receives this action when the movie controller has received a request to set the limits for resizing the movie.

The parameter data contains a pointer to a rectangle—the rectangle defines the boundary coordinates for the movie. If the rectangle is empty, the application wants to disable the size box. You may change the appearance of your controller in response to such a request.

`mcActionShowBalloon`

Your action filter function receives this action when the controller wants to display Balloon Help. Your filter function instructs the controller whether to display the Balloon Help. This action allows you to override the movie controller's default Balloon Help behavior.

The parameter data contains a pointer to a Boolean value. Set the value to `true` to display the appropriate Balloon Help. Otherwise, set the value to `false`.

Note

Your application should never use this action. ♦

Movie Controller Functions

This section describes the functions that are supported by movie controller components. It is divided into the following topics:

- “Associating Movies With Controllers,” which describes the movie controller component functions that allow applications to assign movies to controllers
- “Managing Controller Attributes,” which discusses the movie controller component functions that allow applications to alter the display characteristics of the controller
- “Handling Movie Events,” which discusses the movie controller component functions that applications use to handle movie actions
- “Editing Movies,” which describes the movie controller component functions that help applications edit movies
- “Getting and Setting Movie Controller Time,” which discusses the movie component controller functions that allow applications to get and set movie controller time information
- “Customizing Event Processing,” which describes movie controller component functions that allow applications to perform customized event processing

These functions are discussed from the perspective of the developer of an application that uses movie controllers. If you are developing a movie controller component, your component must behave as described here.

Associating Movies With Controllers

Once your application has established a connection to a movie controller component, you may associate one movie with a movie controller. By default, the new controller has editing and keystroke processing turned off.

You create a new movie controller and assign it to a movie by calling the `NewMovieController` function. This is the easiest way to use a movie controller component.

If you want to exert more control over the assignment of movies to controllers, you can use other movie controller functions. If you want to assign a movie to an existing controller, you can use the `MCNewAttachedController` function. Use the `MCSetMovie` function to assign a movie to or remove a movie from a controller. You can use the `MCGetMovie` function to retrieve a reference to the movie that is assigned to a controller.

When you are done with a controller, use the `DisposeMovieController` function to dispose of the controller.

The `NewMovieController` function locates a movie controller component for you and assigns a movie to that controller. This function always creates a controller that is attached to a movie.

Movie Controller Components

This function is actually implemented by the Movie Toolbox, not by movie controller components. If you are creating your own movie controller component, you do not need to support this function.

```
pascal MovieController NewMovieController (Movie theMovie,
                                           const Rect *movieRect,
                                           long someFlags);
```

theMovie Identifies the movie to be associated with the movie controller.

movieRect Points to the display rectangle that is to contain the movie and its controller.

someFlags Contains flags that control the operation. If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by the `movieRect` parameter and scales the movie to fit in that rectangle. The control portion of the controller is also placed within that rectangle. You may control how the movie and the control are drawn by setting one or more of the following flags to 1:

mcTopLeftMovie

If this flag is set to 1, the movie controller component places the movie into the upper-left corner of the display rectangle specified by the `movieRect` parameter. The component scales the movie to fit into the rectangle. Note that the control portion of the controller may fall outside of the rectangle, depending upon the results of the scaling operation.

mcScaleMovieToFit

If this flag is set to 1, the movie controller component resizes the movie to fit into the display rectangle specified by the `movieRect` parameter after it places the control portion of the controller into the rectangle.

If you set this flag and the `mcScaleMovieToFit` flag to 1, the movie controller component resizes the movie to fit into the specified rectangle and places the control portion of the controller outside of the rectangle.

mcWithBadge

Controls whether the movie controller uses a badge (see “Badges,” which begins on page 2-6, for more information about movie badges). If you set this flag to 1, the movie controller component displays the movie with a badge whenever the controller portion is not displayed. If you set this flag to 0, the movie controller component does not use a badge.

mcNotVisible

Controls whether the controller portion is visible. If you set this flag to 0, the movie controller component displays the controller along with the movie. If you set this flag to 1, the component does not display the controller. If you have set

Movie Controller Components

the `mcWithBadge` flag to 1, specifying that the component uses a badge, the component displays a badge whenever the controller is not visible.

`mcWithFrame`

Specifies whether the component displays a frame around the movie as part of the controller. If you set this flag to 1, the component displays a frame around the movie, including the movie's name. If you set this flag to 0, the component does not display a frame as part of the controller.

DESCRIPTION

The `NewMovieController` function returns a movie controller identifier value. This value identifies a connection to a movie controller component, and it is a component instance.

MCNewAttachedController

The `MCNewAttachedController` function associates a specified movie with a movie controller.

```
pascal ComponentResult MCNewAttachedController (MovieController
                                                mc, Movie theMovie,
                                                WindowPtr w,
                                                Point where);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function.
<code>theMovie</code>	Identifies the movie to be associated with the movie controller.
<code>w</code>	Identifies the window in which the movie is to be displayed. The movie controller component sets the movie's graphics world to match this window. If you set the <code>w</code> parameter to <code>nil</code> , the component uses the current window.
<code>where</code>	Specifies the upper-left corner of the movie within the window specified by the <code>w</code> parameter. The movie controller component uses the movie's boundary rectangle to determine the size of the movie (the Movie Toolbox's <code>GetMovieBox</code> function returns this rectangle).

DESCRIPTION

The `MCNewAttachedController` function forces the controller to be attached to the movie and sets the controller to be visible.

MCSetMovie

The `MCSetMovie` function associates a movie with a specified movie controller.

```
pascal ComponentResult MCSetMovie (MovieController mc,
                                    Movie theMovie,
                                    WindowPtr movieWindow,
                                    Point where);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>theMovie</code>	Identifies the movie to be associated with the movie controller. Set this value to <code>nil</code> to remove the movie from the controller.
<code>movieWindow</code>	Identifies the window in which the movie is to be displayed. The movie controller component sets the movie's graphics world to match this window. If you set the <code>w</code> parameter to <code>nil</code> , the component uses the current window.
<code>where</code>	Specifies the upper-left corner of the movie within the window specified by the <code>movieWindow</code> parameter. The movie controller component uses the movie's boundary rectangle to determine the size of the movie (the Movie Toolbox's <code>GetMovieBox</code> function returns this rectangle).

DESCRIPTION

You can also use the `MCSetMovie` function to remove a movie from its controller.

SEE ALSO

If you want to scale the movie, call the Movie Toolbox's `SetMovieBox` function (described in *Inside Macintosh: QuickTime*) before calling `MCSetMovie`.

MCGetMovie

The `MCGetMovie` function allows your application to retrieve the movie reference for a movie that is associated with a movie controller. The movie controller component returns the movie's identifier value.

```
pascal Movie MCGetMovie (MovieController mc);
```

Movie Controller Components

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCGetMovie` function returns the movie identifier for the movie that is assigned to the specified controller. If there is no movie assigned to the controller, the returned movie identifier is set to `nil`.

DisposeMovieController

The `DisposeMovieController` function disposes of a movie controller. Your application is responsible for disposing of the movie that is associated with the movie controller. Do not dispose of the movie before disposing of the controller.

```
pascal void DisposeMovieController (MovieController mc);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `DisposeMovieController` function is implemented by the Movie Toolbox, not by movie controller components. If you are creating your own movie controller component, you do not have to support this function.

Managing Controller Attributes

Movie controller components provide a number of functions that allow your application to control the display attributes of a movie controller. For example, you can detach the controller from its movie, so that the controller and movie can be managed as separate graphics entities. In addition, movie controller components provide a number of functions that allow you to work with a controller's boundary rectangles and regions. For a complete discussion of these rectangles and regions, see "Spatial Properties," which begins on page 2-6.

The `MCSetControllerAttached` function lets you control whether the movie controller is attached to its movie. The `MCIsControllerAttached` function allows you to determine if a controller is attached to its movie.

You can use the `MCSetControllerPort` and `MCGetControllerPort` functions to work a movie controller's graphics port.

The `MCSetVisible` and `MCGetVisible` functions enable you to control the visibility of the movie controller.

The `MCSetControllerBoundsRect` and `MCGetControllerBoundsRect` functions help you work with a movie controller's boundary rectangle. You can use the `MCGetControllerBoundsRgn` and `MCGetControllerWindowRgn` functions if the controller is not rectangular. You can position a controller and its movie separately by calling the `MCPositionController` function.

MCPositionController

The `MCPositionController` function allows you to control the position of a movie and its controller on the computer display.

```
pascal ComponentResult MCPositionController (MovieController mc,
                                             const Rect *movieRect,
                                             const Rect *controllerRect,
                                             long someFlags);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>movieRect</code>	Points to a <code>Rect</code> structure that specifies the coordinates of the movie's boundary rectangle. (For details on the <code>Rect</code> structure, see the chapter "Basic QuickDraw" in <i>Inside Macintosh: Imaging</i> .)
<code>controllerRect</code>	Points to a <code>Rect</code> structure that specifies the coordinates of the controller's boundary rectangle. The movie controller component always centers the control portion of the controller inside this rectangle. The movie controller component only uses this parameter when the control portion of the controller is detached from the movie. If you are working with an attached controller, you can set this parameter to <code>nil</code> .
<code>someFlags</code>	If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by <code>movieRect</code> and scales the movie to fit in that rectangle. You may control how the movie is drawn by setting one or more of the following flags to 1:
<code>mcTopLeftMovie</code>	If this flag is set to 1, the movie controller component places the movie into the upper-left corner of the display rectangle specified by the <code>movieRect</code> parameter. The component scales the movie to fit into the rectangle. Note that the control portion of the controller may fall outside of the rectangle, depending upon the results of the scaling operation.

Movie Controller Components

`mcScaleMovieToFit`

If this flag is set to 1, the movie controller component resizes the movie to fit into the display rectangle specified by the `movieRect` parameter after it places the control portion of the controller into the rectangle.

If you set this flag and the `mcTopLeftMovie` flag to 1, the movie controller component resizes the movie to fit into the specified rectangle and places the control portion of the controller outside of the rectangle.

`mcPositionDontInvalidate`

If this flag is set to 1, the movie controller component is requested not to invalidate areas of the window that are changed as a result of repositioning the movie or the controller. This flag is useful for applications that use the movie controller as part of a larger document. In particular, if the document is scrolled using QuickDraw's `ScrollRect` routine, optimal redrawing occurs (that is, scrolled areas are not redrawn) if this flag is set. For details on `ScrollRect`, see the chapter “Basic QuickDraw” in *Inside Macintosh: Imaging*.

DESCRIPTION

The `MCPositionController` function works with controllers that are attached to movies and controllers that are not attached to movies.

MCSetControllerAttached

The `MCSetControllerAttached` function allows your application to control whether a movie controller is attached to its movie or detached from it. “About Movie Controller Components,” which begins on page 2-4, discusses the differences between attached and detached movie controllers.

```
pascal ComponentResult MCSetControllerAttached
                                (MovieController mc,
                                 Boolean attach);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

attach Specifies the action for this function. Set the `attach` parameter to `true` to cause the controller to be attached to its movie. Set this parameter to `false` to detach the controller from its movie.

DESCRIPTION

By default, a new movie controller is attached to its movie.

SPECIAL CONSIDERATIONS

Your application should not make any assumptions about the location of an attached movie controller with respect to its movie. The controller may be above, below, or surrounding the movie image.

SEE ALSO

If you need to know the location of the controller, you can use the `MCGetControllerBoundsRect` function, described on page 2-39, to obtain its boundary rectangle.

MCIsControllerAttached

The `MCIsControllerAttached` function returns a value that indicates whether a movie controller is attached to its movie.

```
pascal ComponentResult MCIsControllerAttached
                                (MovieController mc);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCIsControllerAttached` function returns a `ComponentResult` value that indicates whether a movie controller is attached to its movie. If the controller is attached, the returned value is set to 1. If the controller is not attached, the returned value is set to 0.

SEE ALSO

You can use the `MCSetControllerAttached` function, described in the previous section, to attach or detach a movie controller.

MCSetVisible

The `MCSetVisible` function allows your application to control the visibility of a movie controller.

```
pascal ComponentResult MCSetVisible (MovieController mc,
                                     Boolean visible);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>visible</code>	Specifies the action for this function. Set the <code>visible</code> parameter to <code>true</code> to cause the controller to be visible. Set this parameter to <code>false</code> to make the controller invisible.

DESCRIPTION

Movie controller components support badges, which allow you to create a visual cue that helps the user distinguish between static images and images that represent movies. The movie controller component displays a badge in the movie image whenever the movie is visible but the control portion of the controller is not visible. To work with movie controller badges, you must use the `mcActionSetUseBadge` action, which is described in “Movie Controller Actions” beginning on page 2-15.

SPECIAL CONSIDERATIONS

By default, a new controller is hidden so that your application can freely set the display attributes before showing the controller to the user. You should note, however, that the `MCNewAttachedController` function (described on page 2-30) automatically sets the movie controller to be visible. Your application must make the controller visible before the user can work with its associated movie.

SEE ALSO

You can use the `MCGetVisible` function, described in the next section, to determine the visibility of a movie controller.

MCGetVisible

The `MCGetVisible` function returns a value that indicates whether a movie controller is visible.

```
pascal ComponentResult MCGetVisible (MovieController mc);
```

Movie Controller Components

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCGetVisible` function returns a `ComponentResult` value that indicates whether a movie controller is visible. If the controller is visible, the returned value is set to 1. If the controller is not showing, the returned value is set to 0.

SEE ALSO

Use the `MCSetVisible` function, described in the previous section, to change the visibility of a movie controller.

MCDrawBadge

The `MCDrawBadge` function allows you to display a controller's badge. This function places the badge in an appropriate location based on the location of the controller's movie.

```
pascal ComponentResult MCDrawBadge (MovieController mc,
                                     RgnHandle movieRgn,
                                     RgnHandle *badgeRgn);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

movieRgn Specifies the boundary region of the controller's movie.

badgeRgn Points to a region that is to receive information about the location of the badge—your application must dispose of this handle. The movie controller returns the region where the badge is displayed. If you are not interested in this information, you may set this parameter to `nil`.

DESCRIPTION

The `MCDrawBadge` function can be useful in circumstances where you are using a movie controller component but do not want to incur the overhead of having the QuickTime movie in memory all the time. This function allows you to display the badge without having to display the movie. In addition, you can use the badge region to perform mouse-down event testing.

MCSetControllerBoundsRect

The `MCSetControllerBoundsRect` function lets you change the position and size of a movie controller. A controller's boundary rectangle encloses the control portion of the controller. In addition, in cases where the movie is attached to the controller, the boundary rectangle also encloses the movie. Note that changing the size of the boundary rectangle may result in the movie being resized as well, if the movie is attached to the controller.

```
pascal ComponentResult MCSetControllerBoundsRect
                                (MovieController mc,
                                 const Rect *bounds);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>bounds</code>	Points to a rectangle structure that contains the new boundary rectangle for the movie controller.

DESCRIPTION

Movie controller components can reject your request for a number of reasons. For example, some movie controller components may support only fixed-size controllers or controllers whose size is fixed in one dimension. Also, note that your application cannot change the location of an attached controller.

The movie controller component returns a value of `controllerBoundsNotExact` if the boundary rectangle has been changed but does not correspond to the rectangle you specified. In this case, the new boundary rectangle is always smaller than the requested rectangle.

RESULT CODES

<code>controllerBoundsNotExact</code>	-9996	Controller has altered the bounds you supplied
<code>controllerHasFixedHeight</code>	-9998	You cannot change the height of this controller
<code>cannotMoveAttachedController</code>	-9999	You cannot move an attached controller

SEE ALSO

To find the dimensions of the new boundary rectangle, call the `MCGetControllerBoundsRect` function, described in the next section.

MCGetControllerBoundsRect

The `MCGetControllerBoundsRect` function returns a movie controller's boundary rectangle. This rectangle reflects the size and location of the controller even if the controller is currently hidden. If the controller is detached from its movie, the rectangle encompasses only the controller, not the movie. If the controller is attached to its movie, the rectangle encompasses both the controller and the movie.

```
pascal ComponentResult MCGetControllerBoundsRect
                                (MovieController mc,
                                 Rect *bounds);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>bounds</code>	Contains a pointer to a <code>rect</code> structure that is to receive the coordinates of the movie controller's boundary rectangle. If there is insufficient screen space to display the controller, the function may return an empty rectangle.

DESCRIPTION

The returned rectangle is the boundary rectangle for the region occupied by the controller and its movie (if the movie is attached to the controller), even if the controller is not rectangular.

SPECIAL CONSIDERATIONS

Note that if the controller cannot obtain enough screen space, the movie controller component may return an empty rectangle.

SEE ALSO

You can use the `MCGetControllerBoundsRgn` function, described in the next section, to obtain the boundary region for a controller. You can use the `MCGetWindowRgn` function, described on page 2-41, to determine the portion of the window that is currently in use by the controller.

MCGetControllerBoundsRgn

Some movie controllers may not be rectangular in shape. The `MCGetControllerBoundsRgn` function returns the actual region occupied by the controller and its movie, if the movie is attached to the controller. If the movie is not attached to its controller, the boundary region encloses only the control portion of the controller. The rectangle returned by the `MCGetControllerBoundsRect` function (described in the previous section) bounds the region returned by `MCGetControllerBoundsRgn`.

```
pascal RgnHandle MCGetControllerBoundsRgn (MovieController mc);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function.

DESCRIPTION

As with the `MCGetControllerBoundsRect` function, the `MCGetControllerBoundsRgn` function returns a region that reflects the size, shape, and location of the controller, even if the controller is hidden. Your application must dispose of the returned region.

The `MCGetControllerBoundsRgn` function returns a handle to the boundary region. Your application must dispose of this region.

RESULT CODES

Memory Manager errors

SEE ALSO

You can use the `MCGetWindowRgn` function, described in the next section, to determine the portion of the window that is currently in use by the controller.

MCGetWindowRgn

The `MCGetWindowRgn` function allows your application to determine the window region that is actually in use by a controller and its movie. The region returned by this function contains only the visible portions of the controller and its movie.

```
pascal RgnHandle MCGetWindowRgn (MovieController mc, WindowPtr w);
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
w	Identifies the window in which the movie controller and its movie are displayed, if the control portion of the controller is attached to the movie. If the controller is detached and in a separate window from the movie, specify one of the windows.

DESCRIPTION

The returned region may consist of several discontinuous areas. For example, if a controller is detached from its movie, the window region may define separate areas for the movie and the controller. If you want to consider just the controller, you must subtract the movie from the returned region.

Your application must dispose of the returned region.

The `MCGetWindowRgn` function returns a handle to the window region. Your application must dispose of this region.

RESULT CODES

Memory Manager errors

SEE ALSO

You can control the clipping region that is applied to the controller by calling the `MCSetClip` function, which is described in the next section.

MCSetClip

The `MCSetClip` function allows you to set a movie controller's clipping region. This clipping region is equivalent to the movie display clipping region supported by the Movie Toolbox.

```
pascal ComponentResult MCSetClip (MovieController mc,
                                   RgnHandle theClip,
                                   RgnHandle movieClip);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>theClip</code>	Contains a handle to a region that defines the controller's clipping region. This clipping region affects the entire movie controller and its movie, including the controller's badge and associated controls. Set this parameter to <code>nil</code> to clear the controller's clipping region.
<code>movieClip</code>	Contains a handle to a region that defines the clipping region of the controller's movie. This clipping region affects only the movie and the badge, not the movie controller. Set this parameter to <code>nil</code> to clear the movie clipping region.

DESCRIPTION

Your application must dispose of the regions you supply to the `MCSetClip` function.

SPECIAL CONSIDERATIONS

Do not use the Movie Toolbox's `SetMovieDisplayClipRgn` function to modify movies that are associated with movie controllers.

RESULT CODES

Memory Manager errors

SEE ALSO

You can retrieve information about a controller's clipping information by calling the `MCGetClip` function, which is described in the next section.

MCGetClip

The `MCGetClip` function allows you to obtain information describing a movie controller's clipping regions.

```
pascal ComponentResult MCGetClip (MovieController mc,
                                   RgnHandle *theClip,
                                   RgnHandle *movieClip);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>theClip</code>	Contains a pointer to a field that is to receive a handle to the clipping region of the entire movie controller. You must dispose of this region when you are done with it. If you are not interested in this information, you may set this parameter to <code>nil</code> .
<code>movieClip</code>	Contains a pointer to a field that is to receive a handle to the clipping region of the controller's movie. You must dispose of this region when you are done with it. If you are not interested in this information, you may set this parameter to <code>nil</code> .

RESULT CODES

Memory Manager errors

SEE ALSO

You can set a controller's clipping information by calling the `MCSetClip` function, which is described in the previous section.

MCSetControllerPort

The `MCSetControllerPort` function allows your application to set the graphics port for a movie controller. You can use this function to place a movie and its associated movie controller in different graphics ports. If you are using an attached controller, both the controller and the movie's graphics ports are changed. If you are using a detached controller, this function changes only the graphics port of the control portion of the controller. You must use the Movie Toolbox's `SetMovieGWorld` function followed by the `MCMovieChanged` function to change other portions.

```
pascal ComponentResult MCSetControllerPort (MovieController mc,
                                             CGrafPtr gp);
```

Movie Controller Components

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>gp</code>	Points to the new graphics port for the movie controller. Set this parameter to <code>nil</code> to use the current graphics port.

DESCRIPTION

The movie controller component may use the foreground and background colors from the graphics port at the time the `MCSetController` function is called to colorize the movie controller.

Movie controller components use the `MCSetControllerPort` function each time you create a new movie controller. Hence, your component must be set to a valid port before creating a new movie controller.

MCGetControllerPort

The `MCGetControllerPort` function returns a movie controller's color graphics port.

```
pascal CGrafPtr MCGetControllerPort (MovieController mc);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
-----------------	--

Handling Movie Events

Movie controller components provide functions that handle movie controller actions. Your application must call these functions whenever an event occurs. Consider this event loop:

```
#if whatIsHandleEvent
    while (! gDoneFlag) {
        gResult = GetNextEvent (everyEvent, &gEventRec);
        if (( MCIsPlayerEvent(gMCPlay, &gEventRec) == 0 )) {
            if (gResult) {
                /* player didn't handle the event */
                HandleEvent(gEventRec);
            }
        }
    }
#endif
```

Movie Controller Components

```
#if 0
/* interface for application-defined routine: */
pascal Boolean MyPlayerFilter ( MovieController mc,
                               short* action, long* params);
#endif
```

If the movie controller component handles the event, your application can loop to wait for the next event. Otherwise, your application must take care of the event as part of its normal event handling.

Movie controller components support an action filter. You can instruct the filter to invoke a function in your application whenever actions occur. This action filter function can then perform specialized processing or refer the action back to the movie controller component. The actions supported by movie controller components are discussed in “Movie Controller Actions,” which begins on page 2-15.

The `MCIsPlayerEvent` function lets you pass events to a movie controller component. The `MCSetActionFilterWithRefCon` function allows you to specify your action filter function for a movie controller.

You can use the `MCDoAction` function to request action processing from a movie controller.

If you use any Movie Toolbox functions to change the characteristics of a movie that is associated with a movie controller, you must inform the movie controller—use the `MCMovieChanged` function.

You can obtain information about the current state of the movie controller and its movie by calling the `MCGetControllerInfo` function.

MCIsPlayerEvent

The `MCIsPlayerEvent` function handles all events for a movie controller. Your application should call this function in its main event loop. Call `MCIsPlayerEvent` for each active movie controller until the event is handled.

This function returns a long integer indicating whether the movie controller component handled the event. The component sets this long integer to 1 if it handled the event. Your application should then skip the rest of its event loop and wait for the next event. The return value is 0 otherwise. Your application must then handle the event as part of its normal event processing.

The movie controller component does everything necessary to support the movie controller and its associated movie. For example, the component calls the Movie Toolbox’s `MoviesTask` function for each movie. The movie controller component also handles suspend and resume events. It treats suspend events as deactivate requests and resume events as activate requests.

Movie Controller Components

You can provide an action filter function that is called by the movie controller component. See “Application-Defined Function,” which begins on page 2-61, for details. The component calls your filter function after it decides to process a particular action, but before it actually does so. In this manner, your application can perform custom action processing for a movie controller. Set your action filter function with the `MCSetActionFilterWithRefCon` function, described on page 2-47.

```
pascal ComponentResult MCIsPlayerEvent (MovieController mc,
                                       const EventRecord *e);
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
e	Points to the current event structure.

DESCRIPTION

The `MCIsPlayerEvent` function returns a long integer indicating whether it handled the event. If the movie controller component handled the event, this function sets the returned value to 1. Your application should then skip the rest of its event loop and wait for the next event. If the component did not handle the event, the `MCIsPlayerEvent` function returns a value of 0. Your application must then handle the event.

MCDoAction

Your application can use the `MCDoAction` function to invoke a movie controller component and have it perform a specified action.

```
pascal ComponentResult MCDoAction (MovieController mc,
                                   short action, void *params;
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function.
action	Specifies the action to be taken. See “Movie Controller Actions,” which begins on page 2-15, for descriptions of the actions supported by movie controller components.
params	Points to the parameter data appropriate to the action. See the individual action descriptions in “Movie Controller Actions,” which begins on page 2-15, for information about the parameters required for each supported action.

DESCRIPTION

For example, your application might define a menu item that stops all currently playing movies. When the user selects this menu item, your application could use the `MCDoAction` function to instruct each controller to stop playing. You would do so by specifying an `mcActionPlay` action with the parameters set to 0 to indicate that the controller should stop playing the movie.

MCSetActionFilterWithRefCon

The `MCSetActionFilterWithRefCon` function allows your application to establish an action filter function for a movie controller. The movie controller component calls your action filter function each time the component receives an action for its movie controller. Your filter function is then free to handle the action or to refer it back to the movie controller component. If you refer it back to the movie controller component, the component handles the action. See “Movie Controller Actions,” which begins on page 2-15, for a description of the actions supported by movie controller components.

<pre>pascal ComponentResult MCSetActionFilterWithRefCon (MovieController mc, MCActionFilter filter, long refCon);</pre>	
mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager’s <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
filter	Points to your action filter function. Set this parameter to <code>nil</code> to remove your action filter function.
refCon	Contains a reference constant value. The movie controller component passes this reference constant to your action filter function each time it calls your function.

DESCRIPTION

Movie controller components allow your application to field movie controller actions. You define an action filter function in your application and assign it to a controller by calling the `MCSetActionFilterWithRefCon` function.

You can use the constants described in “Movie Controller Actions,” which begins on page 2-15, to refer to movie controller actions.

If your filter function handles an action, you can handle the action in any way you desire. For example, your filter function could change the operation of movie controller buttons. More commonly, applications use the action filter function to monitor actions of the controller. For instance, your filter function might enable you to find out when the

Movie Controller Components

user clicks the play button, so that your application can enable appropriate menu selections. Alternatively, you can use the filter function to detect when the user resizes the movie.

SEE ALSO

If you use any Movie Toolbox functions that modify the movie in your action filter function, be sure to call the `MCMovieChanged` function (described on page 2-49).

MCGetControllerInfo

Your application can use the `MCGetControllerInfo` function to determine the current status of a movie controller and its associated movie. You can use this information to control your application's menu highlighting.

```
pascal ComponentResult MCGetControllerInfo (MovieController mc,
                                           long *someFlags);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

someFlags Contains a pointer to flags that specify the current status and capabilities of the controller. The following flags are defined (more than one flag may be set to 1):

`mcInfoUndoAvailable`

The user has edited the movie. If this flag is set to 1, you can call the `MCUndo` function (described on page 2-54).

`mcInfoCutAvailable`

The user has selected some material in the movie and editing is enabled. If this flag is set to 1, you can call the `MCCut` function (described on page 2-51).

`mcInfoCopyAvailable`

The user has selected some material in the movie. If this flag is set to 1, you can call the `MCCopy` function (described on page 2-52).

`mcInfoPasteAvailable`

There is movie data in the scrap and editing is enabled. If this flag is set to 1, you can call the `MCPaste` function (described on page 2-53).

If your application maintains a private scrap, this flag does not reflect the state of that scrap.

`mcInfoClearAvailable`

The user has selected some material in the movie and editing is enabled. If this flag is set to 1, you can call the `MCClear` function (described on page 2-54).

`mcInfoHasSound`

The movie has sound. If this flag is set to 1, the controller can play a movie's sound.

`mcInfoIsPlaying`

If this flag is set to 1, the movie is playing.

`mcInfoIsLooping`

The controller is currently set to play its movie repeatedly. If this flag is set to 1, the movie is looping.

`mcInfoIsInPalindrome`

The controller is currently set to play its movie repeatedly, alternating between forward and backward playback. If this flag is set to 1, the movie is in palindrome looping mode.

`mcInfoEditingEnabled`

The user can edit the movie associated with this controller. If this flag is set to 1, you have enabled editing by calling the `MCEnableEditing` function (described on page 2-50).

MCMovieChanged

The `MCMovieChanged` function lets you inform a movie controller component that your application has used the Movie Toolbox to change the characteristics of its associated movie.

```
pascal ComponentResult MCMovieChanged (MovieController mc,
                                       Movie theMovie);
```

`mc` Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

`theMovie` Identifies the movie that has been changed.

DESCRIPTION

Your application should be able to make most movie changes using the `MCDoAction` function (described on page 2-46). However, if your application uses Movie Toolbox functions to change the characteristics of a movie that is associated with a movie controller, you must inform the controller so that it can update itself accordingly. For

Movie Controller Components

instance, if your application changes the size of the movie without informing the movie controller component, the control portion of the controller may no longer be the proper size for the movie.

RESULT CODES

Memory Manager errors

Editing Movies

Movie controller components can provide editing capabilities. This section describes the functions that your application can use to alter movies that are associated with movie controllers.

Your application can use the `MCEnableEditing` function to enable editing for a specified movie controller. Movie controller components may return an error code indicating that editing is not supported. Use the `MCIsEditingEnabled` function to find out if editing is enabled for a specified controller.

The `MCCopy`, `MCCut`, `MCPaste`, `MCClear`, and `MCUndo` functions support normal editing operations on movies associated with movie controllers. These functions operate on the current movie selection.

Two functions are also provided that facilitate work with Edit menus. You can use the `MCSetUpEditMenu` function to highlight and name the items in the Edit menu for your application. The `MCGetMenuString` function is provided for you to use with a non-standard Edit menu.

MCEnableEditing

The `MCEnableEditing` function allows your application to enable and disable editing for a movie controller. Once editing is enabled for a controller, the user may edit the movie associated with the controller.

```
pascal ComponentResult MCEnableEditing (MovieController mc,
                                         Boolean enabled);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>enabled</code>	Specifies whether to enable or disable editing for the controller. Set this parameter to <code>true</code> to enable editing; set the <code>enabled</code> parameter to <code>false</code> to disable editing.

DESCRIPTION

By default, editing is turned off when you create a new movie controller. If you want to allow the user to edit, you must use the `MCEnableEditing` function to enable editing.

SPECIAL CONSIDERATIONS

Note that a movie controller component may not support editing. Therefore, your application should check the component result from this function before continuing with other movie-editing operations.

MCIsEditingEnabled

The `MCIsEditingEnabled` function allows your application to determine whether editing is currently enabled for a movie controller. The movie controller component returns a long value reflecting the edit state of the controller. Once editing is enabled for a controller, the user may edit the movie associated with the controller.

```
pascal long MCIsEditingEnabled (MovieController mc);
```

mc Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCIsEditingEnabled` function returns a long integer that contains a value indicating the current edit state of the controller. This returned value is set to 1 if editing is enabled. This returned value is set to 0 if editing is disabled or if the controller component does not support editing.

MCCut

The `MCCut` function returns a copy of the current movie selection from the movie associated with a specified controller and then removes the current movie selection from the source movie. Your application is responsible for the returned movie. If you want to allow the user to paste the movie selection, use the Movie Toolbox's `PutMovieOnScrap` function to place the movie selection onto the scrap. Be sure to dispose of the movie afterward, using the Movie Toolbox's `DisposeMovie` function.

```
pascal Movie MCCut (MovieController mc);
```

Movie Controller Components

`mc` Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCCut` function returns a movie containing the current selection from the movie associated with the specified controller. If the user has not made a selection, the returned movie reference is set to `nil`.

SEE ALSO

The `MCCut` function is analogous to the Movie Toolbox's `CutMovieSelection` function.

MCCopy

The `MCCopy` function returns a copy of the current movie selection from the movie associated with a specified controller. The selection remains active after this operation. Your application is responsible for the returned movie.

If you want to allow the user to paste the movie selection, use the Movie Toolbox's `PutMovieOnScrap` function to place the movie selection onto the scrap. Be sure to dispose of the movie afterward, using the Movie Toolbox's `DisposeMovie` function.

```
pascal Movie MCCopy (MovieController mc);
```

`mc` Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` function (described on page 2-28).

DESCRIPTION

The `MCCopy` function returns a movie containing the current selection from the movie associated with the specified controller. If the user has not made a selection, the returned movie reference is set to `nil`.

SEE ALSO

This function is analogous to the Movie Toolbox's `CopyMovieSelection` function.

MCPaste

The `MCPaste` function inserts a specified movie at the current movie time in the movie associated with a specified controller.

```
pascal ComponentResult MCPaste (MovieController mc,
                                Movie srcMovie);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>srcMovie</code>	Specifies the movie to be inserted into the current selection in the movie associated with the movie controller specified by the <code>mc</code> parameter. If you set this parameter to <code>nil</code> , the movie controller component retrieves the source movie from the scrap.

DESCRIPTION

All of the tracks from the source movie are placed in the destination movie. If the duration of the destination movie's current selection is 0, the source movie is inserted at the starting time of the current selection. If the current selection duration is nonzero, the function clears the current selection and then inserts the tracks from the source movie. After the paste operation, the current selection time is set to the start of the tracks that were inserted and the duration is set to the source movie's duration.

SEE ALSO

This function is analogous to the Movie Toolbox's `PasteMovieSelection` function.

SPECIAL CONSIDERATIONS

The preferred way to use the `MCPaste` function is to set the `srcMovie` parameter to `nil`. This causes the movie controller to use movie import components to paste other types of data than movies.

MCClear

The `MCClear` function removes the current movie selection from the movie associated with a specified controller.

```
pascal ComponentResult MCClear (MovieController mc);
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function.
----	---

DESCRIPTION

After removing the segment, the duration of the movie's current selection is set to 0. This function removes empty tracks from the resulting movie.

SEE ALSO

This function is analogous to the Movie Toolbox's `ClearMovieSelection` function.

MCUndo

The `MCUndo` function allows your application to discard the effects of the most recent edit operation.

```
pascal ComponentResult MCUndo (MovieController mc);
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
----	--

SEE ALSO

Your movie controller component could use the Movie Toolbox's edit state functions to implement this function. (See the chapter "Movie Toolbox" in *Inside Macintosh: QuickTime* for more information about the edit state functions.)

MCSetUpEditMenu

The `MCSetUpEditMenu` function correctly highlights and names the items in your application's Edit menu.

```
pascal ComponentResult MCSetUpEditMenu (MovieController mc,
                                         long modifiers,
                                         MenuHandle mh);
```

<code>mc</code>	Specifies the movie controller for this operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function.
<code>modifiers</code>	Indicates the current modifiers from the mouse-down or key-down event to which you are responding.
<code>mh</code>	Specifies a menu handler to your current Edit menu. The first six items in your Edit menu should be the standard editing commands: Undo, a blank line, Cut, Copy, Paste, and Clear.

DESCRIPTION

When your application is highlighting its menus, you should call `MCSetUpEditMenu` immediately before you use the Menu Manager's `MenuSelect` or `MenuKey` functions. For details on `MenuSelect` and `MenuKey`, see *Inside Macintosh: Macintosh Toolbox Essentials*.

MCGetMenuString

If your application has a non-standard Edit menu, you can use the `MCGetMenuString` function together with the `MCGetControllerInfo` function to assign names correctly to the items in your application's Edit menu.

```
pascal ComponentResult MCGetMenuString (MovieController mc,
                                         long modifiers,
                                         short item,
                                         Str255 aString);
```

<code>mc</code>	Specifies the movie controller for this operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function.
<code>modifiers</code>	Indicates the current modifiers from the mouse-down or key-down event to which you are responding.

Movie Controller Components

<code>item</code>	Contains one of the appropriate movie controller Edit menu constants returned in the <code>aString</code> parameter.
<code>aString</code>	Contains (on return) an appropriate string to set the menu item text. The following flags are available:
<code>mcMenuUndo</code>	Contains the string to set the menu item text to the Undo command.
<code>mcMenuCut</code>	Contains the string to set the menu item text to the Cut command.
<code>mcMenuCopy</code>	Contains the string to set the menu item text to the Copy command.
<code>mcMenuPaste</code>	Contains the string to set the menu item text to the Paste command.
<code>mcMenuClear</code>	Contains the string to set the menu item text to the Clear command.

DESCRIPTION

The `MCGetMenuString` function is used by the `MCSetUpEditMenu` function, which is described in the previous section.

SEE ALSO

To highlight menu items, use the `MCGetControllerInfo` function, which is described on page 2-48, to determine which items should be enabled.

Getting and Setting Movie Controller Time

Movie controller components provide functions that allow your application to work with temporal aspects of movie controllers. You can use the `MCSetDuration` function to set the duration of a movie controller to some arbitrary value. The `MCGetCurrentTime` function lets you retrieve the time value represented by the indicator on the movie controller's slider.

MCSetDuration

The `MCSetDuration` function allows your application to set a controller's duration in the case where a controller does not have a movie associated with it.

```
pascal ComponentResult MCSetDuration (MovieController mc,
                                       TimeValue duration);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>duration</code>	Specifies the new duration for the movie. This duration value must be in the controller's time scale.

DESCRIPTION

The controller's duration remains at this new value until you assign a movie to the controller.

SEE ALSO

You can use the `MCGetCurrentTime` function, which is described in the next section, to obtain the time scale for the controller.

MCGetCurrentTime

Your application can use the `MCGetCurrentTime` function to obtain the time value represented by the indicator on the movie controller's slider. This time value is appropriate to the movie currently being affected by the movie controller. You can also obtain the time scale for this time value.

```
pascal TimeValue MCGGetCurrentTime (MovieController mc,
                                     TimeScale *scale);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>scale</code>	Contains a pointer to a field that is to receive the time scale for the controller.

DESCRIPTION

The `MCGetCurrentTime` function returns the time value that corresponds to the current setting of the indicator on the movie controller's slider.

Customizing Event Processing

Movie controller components provide a number of functions that allow your application to customize event processing. If your application does not use the `MCIsPlayerEvent` function (described on page 2-45), you can use these functions to direct movie controller events to the appropriate movie controller component. The component then attempts to handle the event.

Your application obtains the values for many of the function parameters from the appropriate event structure.

Each function returns a value that indicates whether it handled the event. If the controller component completely handles the event, the function sets the return value to 1. If the controller component does not handle the event, the function sets the return value to 0. Your application must then handle the event.

MCActivate

Your application can use the `MCActivate` function in response to activate, deactivate, suspend, and resume events.

```
pascal ComponentResult MCActivate (MovieController mc,
                                   WindowPtr w,
                                   Boolean activate);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>w</code>	Specifies the window in which the event has occurred.
<code>activate</code>	Indicates the nature of the event. Set this parameter to <code>true</code> for activate and resume events. Set it to <code>false</code> for deactivate and suspend events.

DESCRIPTION

The `MCActivate` function returns a value indicating whether it handled the event. The function sets the returned value to 1 if it handles the event. The function sets the returned value to 0 if it does not handle the event. In this case, your application is responsible for the event.

MCClick

Your application should call the `MCClick` function when the user clicks in a movie controller window.

```
pascal ComponentResult MCClick (MovieController mc, WindowPtr w,
                                Point where, long when,
                                long modifiers);
```

mc	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
w	Specifies the window in which the event has occurred.
where	Indicates the location of the click. This value is expressed in the local coordinates of the window specified by the <code>w</code> parameter. Your application must convert this value from the global coordinates returned in the event structure.
when	Indicates when the user pressed the mouse button. You obtain this value from the event structure.
modifiers	Specifies modifier flags for the event. You obtain this value from the event structure.

DESCRIPTION

The `MCClick` function returns a value indicating whether it handled the event. The function sets the returned value to 1 if it handles the event. The function sets the returned value to 0 if it does not handle the event. In this case, your application is responsible for the event.

MCDraw

Your application should call the `MCDraw` function in response to an update event. The movie controller component updates the movie controller if the controller is in the window that received the update event. The controller component updates the movie associated with the controller only if the movie is contained in the window that received the event.

```
pascal ComponentResult MCDraw (MovieController mc, WindowPtr w);
```

Movie Controller Components

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>w</code>	Points to the window in which the update event has occurred.

DESCRIPTION

The `MCDraw` function returns a value indicating whether it handled the event. The function sets the returned value to 1 if it handles the event. The function sets the returned value to 0 if it does not handle the event. In this case, your application is responsible for the event.

MCIdle

The `MCIdle` function performs idle processing for a movie controller. This idle processing includes calling the Movie Toolbox's `MoviesTask` function for each movie that is associated with the controller. Your application should call the `MCIdle` function as often as possible, in order to ensure consistent movie play behavior.

```
pascal ComponentResult MCIdle (MovieController mc);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
-----------------	--

DESCRIPTION

The `MCIdle` function returns a value indicating whether it handled the event. The function sets the returned value to 1 if it handles the event. The function sets the returned value to 0 if it does not handle the event. In this case, your application is responsible for the event.

MCKey

The `MCKey` function handles keyboard events for a movie controller. You can call this function only if you have enabled keystroke processing in the controller. By default, keystroke processing is turned off when you create a movie controller. You can enable and disable keystroke processing using the `mcActionSetKeysEnabled` action with the `MCDoAction` function (described on page 2-46).

```
pascal ComponentResult MCKey (MovieController mc, char key,
                              long modifiers);
```

<code>mc</code>	Specifies the movie controller for the operation. You obtain this identifier from the Component Manager's <code>OpenComponent</code> or <code>OpenDefaultComponent</code> function, or from the <code>NewMovieController</code> function (described on page 2-28).
<code>key</code>	Specifies the keystroke. You obtain this value from the event structure.
<code>modifiers</code>	Specifies modifier flags for the event. You obtain this value from the event structure.

DESCRIPTION

The `MCKey` function returns a value indicating whether it handled the event. The function sets the returned value to 1 if it handles the event. The function sets the returned value to 0 if it does not handle the event. In this case, your application is responsible for the event.

Application-Defined Function

Movie controller components provide an action filter function that you establish with the `MCSetActionFilterWithRefCon` function (described on page 2-47).

MyPlayerFilterWithRefCon

Your action filter function, `MyPlayerFilterWithRefCon`, should be in this form:

```
Boolean MyPlayerFilterWithRefCon (MovieController mc,
                                  short action,
                                  void *params, long refCon);
```

<code>mc</code>	Specifies the movie controller for the operation.
-----------------	---

Movie Controller Components

<code>action</code>	A short integer containing the action code. The movie controller component sets this parameter to point to the <code>what</code> field in the appropriate action structure. (Although this action is passed as a variable, it should not be changed by the filter.) See “Movie Controller Actions,” which begins on page 2-15, for a description of the actions supported by movie controller components.
<code>params</code>	Contains a pointer to the parameter data appropriate to the action—for example, setting the playback rate. See the individual descriptions of the actions beginning on page 2-15 for information about the parameters supplied for each supported action.
<code>refCon</code>	Contains a reference constant value. The movie controller component passes this reference constant to your action filter function each time it calls your function.

DESCRIPTION

Your filter function must return a Boolean value indicating whether it handled the action. Set the returned Boolean value to `true` if your function completely handles the action. In this case, the movie controller component performs no additional processing for the action. Set the returned value to `false` if your function does not handle the action. The movie controller component then performs the appropriate processing for the action.

Summary of Movie Controller Components

C Summary

Constants

```
enum {
    kMCSetMovieSelect          = 2,  /* MCSetMovie */
    kMCRemoveMovieSelect      = 3,  /* MCRemoveMovie */
    kMCIsPlayerEventSelect    = 7,  /* MCIsPlayerEvent */
    kMCSetActionFilterSelect  = 8,  /* MCSetActionFilter */
    kMCDoActionSelect         = 9,  /* MCDoAction */
    kMCSetControllerAttachedSelect= 10, /* MCSetControllerAttached */
    kMCIsControllerAttachedSelect = 11, /* MCIsControllerAttached */
    kMCSetControllerPortSelect = 12, /* MCSetControllerPort */
    kMCGetControllerPortSelect  = 13, /* MCGetControllerPort */
    kMCGetVisibleSelect        = 14, /* MCGetVisible */
    kMCSetVisibleSelect        = 15, /* MCSetVisible */
    kMCGetControllerBoundsRectSelect
                                = 16, /* MCGetControllerBoundsRect */
    kMCSetControllerBoundsRectSelect
                                = 17, /* MCSetControllerBoundsRect */
    kMCGetControllerBoundsRgnSelect
                                = 18, /* MCGetControllerBoundsRgn */
    kMCGetWindowRgnSelect      = 19, /* MCGetWindowRgn */
    kMCMovieChangedSelect      = 20, /* MCMovieChanged */
    kMCSetDurationSelect       = 21, /* MCSetDuration */
    kMCGetCurrentTimeSelect    = 22, /* MCGetCurrentTime */
    kMCNewAttachedControllerSelect= 23, /* MCNewAttachedController */
    kMCDrawSelect              = 24, /* MCDraw */
    kMCActivateSelect          = 25, /* MCActivate */
    kMCIdleSelect              = 26, /* MCIdle */
    kMCKeySelect               = 27, /* MCKey */
    kMCClickSelect             = 28, /* MCClick */
    kMCEnableEditingSelect     = 29, /* MCEnableEditing */
    kMCIsEditingEnabledSelect  = 30, /* MCIsEditingEnabled */
    kMCCopySelect              = 31, /* MCCopy */
    kMCCutSelect               = 32, /* MCCut */
}
```

Movie Controller Components

```

kMCPasteSelect          = 33, /* MCPaste */
kMCClearSelect          = 34, /* MCClear */
kMCUndoSelect           = 35, /* MCUndo */
kMCPositionControllerSelect = 36, /* MCPositionController */
kMCGetControllerInfoSelect = 37, /* MCGetControllerInfo */
kMCSetClipSelect        = 40, /* MCSetClip */
kMCGetClipSelect        = 41, /* MCGetClip */
kMCDrawBadgeSelect      = 42, /* MCDrawBadge */
kMCSetUpEditMenuSelect  = 43, /* MCSetUpEditMenu */
kMCGetMenuStringSelect  = 44, /* MCGetMenuString */
kMCSetActionFilterWithRefConSelect
                        = 45
                        /* SetActionFilterWithRefConSelect */
};

enum {
    mcActionIdle          = 1, /* give event-processing time to
                                movie controller */
    mcActionDraw           = 2, /* send update event to movie
                                controller */
    mcActionActivate       = 3, /* activate movie controller */
    mcActionDeactivate     = 4, /* deactivate controller */
    mcActionMouseDown      = 5, /* pass mouse-down event */
    mcActionKey            = 6, /* pass key-down or auto-key event */
    mcActionPlay           = 8, /* start playing movie */
    mcActionGoToTime       = 12, /* move to specific time in a movie */
    mcActionSetVolume      = 14, /* set a movie's volume */
    mcActionGetVolume      = 15, /* retrieve a movie's volume */
    mcActionStep           = 18, /* play a movie a specified number
                                of frames at a time */
    mcActionSetLooping     = 21, /* enable or disable looping */
    mcActionGetLooping     = 22, /* find out if movie is looping */
    mcActionSetLoopIsPalindrome = 23, /* enable palindrome looping */
    mcActionGetLoopIsPalindrome = 24, /* find out if palindrome looping
                                is on */
    mcActionSetGrowBoxBounds = 25, /* set limits for resizing a movie */
    mcActionControllerSizeChanged = 26, /* user has resized movie
                                controller */
    mcActionSetSelectionBegin = 29, /* start time of movie's current
                                selection */
    mcActionSetSelectionDuration = 30, /* set duration of movie's current
                                selection */
    mcActionSetKeysEnabled  = 32, /* enable or disable keystrokes for
                                movie */

```

Movie Controller Components

```

mcActionGetKeysEnabled      = 33, /* find out if keystrokes are
                                   enabled */
mcActionSetPlaySelection    = 34, /* constrain playing to the current
                                   selection */
mcActionGetPlaySelection    = 35, /* find out if movie is constrained to
                                   playing within selection */
mcActionSetUseBadge         = 36, /* enable or disable movie's
                                   playback badge */
mcActionGetUseBadge         = 37, /* find out if movie controller is
                                   using playback badge */
mcActionSetFlags            = 38, /* set movie's control flags */
mcActionGetFlags            = 39, /* retrieve movie's control flags */
mcActionSetPlayEveryFrame   = 40, /* instruct controller to play all
                                   frames in movie */
mcActionGetPlayEveryFrame   = 41, /* find out if controller is playing
                                   every frame in movie */
mcActionGetPlayRate         = 42, /* determine playback rate */
mcActionShowBalloon         = 43, /* find out if controller wants to
                                   display Balloon Help */
mcActionBadgeClick          = 44, /* user clicked movie's badge */
mcActionMovieClick          = 45, /* user clicked movie */
mcActionSuspend             = 46, /* suspend action */
mcActionResume              = 47  /* resume action */
};

enum {
    mcTopLeftMovie          = 1<<0,    /* places movie in upper-left corner of
                                           display rectangle */
    mcScaleMovieToFit       = 1<<1,    /* resizes movie to fit into display
                                           rectangle */
    mcWithBadge              = 1<<2,    /* controls whether badge is displayed */
    mcNotVisible             = 1<<3,    /* controls whether controller portion
                                           is visible */
    mcWithFrame              = 1<<4     /* specifies whether component shows
                                           frame around movie */
};

enum {
    mcFlagSuppressMovieFrame = 1<<0,    /* controls display of frame */
    mcFlagSuppressStepButtons = 1<<1,    /* controls display of step
                                           buttons */
};

```

Movie Controller Components

```

    mcFlagSuppressSpeakerButton    = 1<<2,  /* controls display of speaker
                                              button */
    mcFlagsUseWindowPalette        = 1<<3    /* controls display of window
                                              palette */
};

enum {
    mcInfoUndoAvailable            = 1<<0,  /* MUndo function available */
    mcInfoCutAvailable             = 1<<1,  /* MCCut function available */
    mcInfoCopyAvailable            = 1<<2,  /* MCCopy function available */
    mcInfoPasteAvailable           = 1<<3,  /* MCPaste function available */
    mcInfoClearAvailable           = 1<<4,  /* MCClear function available */
    mcInfoHasSound                 = 1<<5,  /* controller can play movie's
                                              sound */

    mcInfoIsPlaying                = 1<<6,  /* movie is playing */
    mcInfoIsLooping                = 1<<7,  /* movie is looping */
    mcInfoIsInPalindrome           = 1<<8,  /* movie is alternating between
                                              forward and backward playback */

    mcInfoEditingEnabled           = 1<<9    /* MEnableEditing function
                                              available */
};

enum {
    mcMenuUndo    = 1,  /* Undo command */
    mcMenuCut     = 3,  /* Cut command */
    mcMenuCopy    = 4,  /* Copy command */
    mcMenuPaste   = 5,  /* Paste command */
    mcMenuClear   = 6   /* Clear command */
};

enum {
    mcPositionDontInvalidate = 1<<5  /* do not invalidate areas of window
                                      changed due to repositioning of movie
                                      or controller */
};

```

Data Types

```

typedef short mcAction;

typedef unsigned long MCFlags;

```


Movie Controller Functions

Associating Movies With Controllers

```
pascal MovieController NewMovieController
    (Movie theMovie, const Rect *movieRect,
     long someFlags);

pascal ComponentResult MCNewAttachedController
    (MovieController mc, Movie theMovie,
     WindowPtr w, Point where);

pascal ComponentResult MCSetMovie
    (MovieController mc, Movie theMovie,
     WindowPtr movieWindow, Point where);

pascal Movie MCGetMovie      (MovieController mc);
pascal void DisposeMovieController
    (MovieController mc);
```

Managing Controller Attributes

```
pascal ComponentResult MCPositionController
    (MovieController mc, const Rect *movieRect,
     const Rect *controllerRect, long someFlags);

pascal ComponentResult MCSetControllerAttached
    (MovieController mc, Boolean attach);

pascal ComponentResult MCIsControllerAttached
    (MovieController mc);

pascal ComponentResult MCSetVisible
    (MovieController mc, Boolean visible);

pascal ComponentResult MCGetVisible
    (MovieController mc);

pascal ComponentResult MCDrawBadge
    (MovieController mc, RgnHandle movieRgn,
     RgnHandle *badgeRgn);

pascal ComponentResult MCSetControllerBoundsRect
    (MovieController mc, const Rect *bounds);

pascal ComponentResult MCGetControllerBoundsRect
    (MovieController mc, Rect *bounds);

pascal RgnHandle MCGetControllerBoundsRgn
    (MovieController mc);

pascal RgnHandle MCGetWindowRgn
    (MovieController mc, WindowPtr w);

pascal ComponentResult MCSetClip
    (MovieController mc, RgnHandle theClip,
     RgnHandle movieClip);
```

Movie Controller Components

```

pascal ComponentResult MCGetClip
    (MovieController mc, RgnHandle *theClip,
     RgnHandle *movieClip);

pascal ComponentResult MCSetControllerPort
    (MovieController mc, CGrafPtr gp);

pascal CGrafPtr MCGetControllerPort
    (MovieController mc);

```

Handling Movie Events

```

pascal ComponentResult MCIsPlayerEvent
    (MovieController mc, const EventRecord *e);

pascal ComponentResult MCDoAction
    (MovieController mc, short action,
     void *params);

pascal ComponentResult MCSetActionFilterWithRefCon
    (MovieController mc, MCActionFilter filter,
     long refCon);

pascal ComponentResult MCGetControllerInfo
    (MovieController mc, long *someFlags);

pascal ComponentResult MCMovieChanged
    (MovieController mc, Movie theMovie);

```

Editing Movies

```

pascal ComponentResult MCEnableEditing
    (MovieController mc, Boolean enabled);

pascal long MCIsEditingEnabled
    (MovieController mc);

pascal Movie MCCut
    (MovieController mc);

pascal Movie MCCopy
    (MovieController mc);

pascal ComponentResult MCPaste
    (MovieController mc, Movie srcMovie);

pascal ComponentResult MCClear
    (MovieController mc);

pascal ComponentResult MCUndo
    (MovieController mc);

pascal ComponentResult MCSetUpEditMenu
    (MovieController mc, long modifiers,
     MenuHandle mh);

pascal ComponentResult MCGetMenuString
    (MovieController mc, long modifiers, short item,
     Str255 aString);

```

Getting and Setting Movie Controller Time

```
pascal ComponentResult MCSetDuration
    (MovieController mc, TimeValue duration);

pascal TimeValue MCGetCurrentTime
    (MovieController mc, TimeScale *scale);
```

Customizing Event Processing

```
pascal ComponentResult MCActivate
    (MovieController mc, WindowPtr w,
     Boolean activate);

pascal ComponentResult MCClick
    (MovieController mc, WindowPtr w, Point where,
     long when, long modifiers);

pascal ComponentResult MCDraw
    (MovieController mc, WindowPtr w);

pascal ComponentResult MCIdle
    (MovieController mc);

pascal ComponentResult MCKey
    (MovieController mc, char key, long modifiers);
```

Application-Defined Function

```
Boolean MyPlayerFilterWithRefCon
    (MovieController mc, short *action,
     void *params, long refCon);
```

Pascal Summary**Constants**

```
CONST
    MovieControllerComponentType = 'play';

    {movie controller selectors}
    kMCSetMovieSelect             = 2;  {MCSetMovie}
    kMCGetMovie                   = 5;  {MCGetMovie}
    kMCIsPlayerEventSelect        = 7;  {MCIsPlayerEvent}
    kMCSetActionFilterSelect      = 8;  {MCSetActionFilter}
    kMCDoActionSelect             = 9;  {MCDoAction}
    kMCSetControllerAttachedSelect = $A; {MCSetControllerAttached}
```

Movie Controller Components

```

kMCIsControllerAttachedSelect    = $B; {MCIsControllerAttached}
kMCSetControllerPortSelect      = $C; {MCSetControllerPort}
kMCGetControllerPortSelect      = $D; {MCGetControllerPort}
kMCGetVisibleSelect            = $E; {MCGetVisible}
kMCSetVisibleSelect            = $F; {MCSetVisible}
kMCGetControllerBoundsRectSelect = $10; {MCGetControllerBoundsRect}
kMCSetControllerBoundsRectSelect = $11; {MCSetControllerBoundsRect}
kMCGetControllerBoundsRgnSelect  = $12; {MCGetControllerBoundsRgn}
kMCGetWindowRgnSelect           = $13; {MCGetWindowRgn}
kMCMovieChangedSelect          = $14; {MCMovieChanged}
kMCSetDurationSelect            = $15; {MCSetDuration}
kMCGetCurrentTimeSelect         = $16; {MCGetCurrentTime}
kMCNewAttachedControllerSelect  = $17; {MCNewAttachedController}
kMCDrawSelect                   = $18; {MCDraw}
kMCActivateSelect               = $19; {MCActivate}
kMCIdleSelect                   = $1A; {MCIdle}
kMCKeySelect                    = $1B; {MCKey}
kMCClickSelect                  = $1C; {MCClick}
kMCEnableEditingSelect          = $1D; {MCEnableEditing}
kMCIsEditingEnabledSelect       = $1E; {MCIsEditingEnabled}
kMCCopySelect                   = $1F; {MCCopy}
kMCCutSelect                    = $20; {MCCut}
kMCPasteSelect                  = $21; {MCPaste}
kMCClearSelect                  = $22; {MCClear}
kMCUndoSelect                   = $23; {MCUndo}
kMCPositionControllerSelect     = $24; {MCPositionController}
kMCGetControllerInfoSelect      = $25; {MCGetControllerInfo}
kMCSetClipSelect                = $28; {MCSetClip}
kMCGetClipSelect                = $29; {MCGetClip}
kMCDrawBadgeSelect              = $2A; {MCDrawBadge}
kMCSetUpEditMenuSelect          = $2B; {MCSetUpEditMenu}
kMCGetMenuStringSelect          = $2C; {MCGetMenuString}
kMCSetActionFilterWithRefConSelect = $2D; {MCSetActionFilterWithRefConSelect}

{movie controller actions}
mcActionIdle                    = 1;  {give event-processing }
                                { time to movie controller}
mcActionDraw                    = 2;  {send update event to movie }
                                { controller}
mcActionActivate                = 3;  {activate controller}
mcActionDeactivate              = 4;  {deactivate controller}
mcActionMouseDown               = 5;  {pass mouse-down event}
mcActionKey                     = 6;  {pass key-down or auto-key event}

```

Movie Controller Components

mcActionPlay	= 8; {start playing movie}
mcActionGoToTime	= 12; {move to specific time in } { a movie}
mcActionSetVolume	= 14; {set a movie's volume}
mcActionGetVolume	= 15; {retrieve a movie's volume}
mcActionStep	= 18; {play movie skipping specified } { number of frames at a time}
mcActionSetLooping	= 21; {enable/disable looping } { for a movie}
mcActionGetLooping	= 22; {determine whether a } { movie is looping}
mcActionSetLoopIsPalindrome	= 23; {enable palindrome looping}
mcActionGetLoopIsPalindrome	= 24; {is palindrome looping on?}
mcActionSetGrowBoxBounds	= 25; {set limits for resizing a movie}
mcActionControllerSizeChanged	= 26; {user has resized movie controller}
mcActionSetSelectionBegin	= 29; {start time of movie's } { current selection}
mcActionSetSelectionDuration	= 30; {set duration of movie's } { current selection}
mcActionSetKeysEnabled	= 32; {enable/disable } { keystrokes for movie}
mcActionGetKeysEnabled	= 33; {are keystrokes enabled?}
mcActionSetPlaySelection	= 34; {constrain playing to the } { current selection}
mcActionGetPlaySelection	= 35; {is movie constrained to } { playing within selection}
mcActionSetUseBadge	= 36; {enable/disable movie's } { playback badge}
mcActionGetUseBadge	= 37; {is movie controller } { using playback badge?}
mcActionSetFlags	= 38; {set movie's control flags}
mcActionGetFlags	= 39; {get movie's control flags}
mcActionSetPlayEveryFrame	= 40; {instruct controller to } { play all frames in movie}
mcActionGetPlayEveryFrame	= 41; {is controller playing } { every frame in movie?}
mcActionGetPlayRate	= 42; {determine playback rate}
mcActionShowBalloon	= 43; {controller wants to } { display balloon help}
mcActionBadgeClick	= 44; {user clicked movie's badge}
mcActionMovieClick	= 45; {user clicked movie}
mcActionSuspend	= 46; {suspend action}
mcActionResume	= 47; {resume action}

Movie Controller Components

```

{controller creation flags}
mcTopLeftMovie          = $1; {places movie in upper-left }
                           { corner of display rectangle}
mcScaleMovieToFit       = $2; {resizes movie to fit into }
                           { display rectangle}
mcWithBadge             = $4; {controls whether badge }
                           { is displayed}
mcNotVisible            = $8; {controls whether controller }
                           { portion is visible}
mcWithFrame             = $10; {specifies whether component }
                           { shows frame around movie}

{movie control flags}
mcFlagSuppressMovieFrame = $1; {controls display of frame}
mcFlagSuppressStepButtons = $2; {controls display of step buttons}
mcFlagSuppressSpeakerButton = $4; {controls display of speaker }
                           { button}
mcFlagsUseWindowPalette = $5; {controls display of window }
                           { palette}

(movie controller information flags)
mcInfoUndoAvailable     = $1;    {MCUndo function available}
mcInfoCutAvailable      = $2;    {MCCut function available}
mcInfoCopyAvailable     = $4;    {MCCopy function available}
mcInfoPasteAvailable    = $8;    {MCPaste function available}
mcInfoClearAvailable    = $10;   {MCClear function available}
mcInfoHasSound          = $20;   {controller can play movie's }
                           { sound}
mcInfoIsPlaying         = $40;   {movie is playing}
mcInfoIsLooping         = $80;   {movie is looping}
mcInfoIsInPalindrome    = $100;  {movie is alternating between }
                           { forward and backward playback}
mcInfoEditingEnabled     = $200;  {MCEnableEditing function }
                           { available}

mcMenuUndo = 1; {Undo command}
mcMenuCut  = 3; {Cut command}
mcMenuCopy = 4; {Copy command}
mcMenuPaste = 5; {Paste command}
mcMenuClear = 6; {Clear command}

```

Movie Controller Components

```
mcPositionDontInvalidate = 32;    {do not invalidate areas of window }
                                   { changed due to repositioning of }
                                   { movie or controller}
```

Data Types

TYPE

```
mcAction          = Integer;
mcFlags           = LongInt;
```

Movie Controller Routines

Associating Movies With Controllers

```
FUNCTION NewMovieController (theMovie: Movie; movieRect: Rect;
                             someFlags: LongInt): MovieController;

FUNCTION MCNewAttachedController
    (mc: MovieController; theMovie: theMovie;
     w: WindowPtr; where: Point): ComponentResult;

FUNCTION MCSetMovie
    (mc: MovieController; theMovie: Movie;
     movieWindow: WindowPtr; where: Point):
    ComponentResult;

FUNCTION MCGetMovie
    (mc: MovieController): Movie;

PROCEDURE DisposeMovieController
    (mc: MovieController);
```

Managing Controller Attributes

```
FUNCTION MCPositionController
    (mc: MovieController; VAR movieRect: Rect;
     VAR controllerRect: Rect; someFlags: LongInt):
    ComponentResult;

FUNCTION MCSetControllerAttached
    (mc: MovieController;
     attach: Boolean): ComponentResult;

FUNCTION MCIsControllerAttached
    (mc: MovieController): ComponentResult;

FUNCTION MCSetVisible
    (mc: MovieController; visible: Boolean):
    ComponentResult;

FUNCTION MCGetVisible
    (mc: MovieController): ComponentResult;

FUNCTION MCDrawBadge
    (mc: MovieController; movieRgn: RgnHandle;
     VAR badgeRgn: RgnHandle): ComponentResult;
```

Movie Controller Components

```

FUNCTION MCSetControllerBoundsRect
    (mc: MovieController; bounds: Rect):
        ComponentResult;
FUNCTION MCGetControllerBoundsRect
    (mc: MovieController; VAR bounds: Rect):
        ComponentResult;
FUNCTION MCGetControllerBoundsRgn
    (mc: MovieController): RgnHandle;
FUNCTION MCGetWindowRgn    (mc: MovieController; w: WindowPtr): RgnHandle;
FUNCTION MCSetClip         (mc: MovieController; theClip: RgnHandle;
    movieClip: RgnHandle): ComponentResult;
FUNCTION MCGetClip         (mc: MovieController; VAR theClip: RgnHandle;
    VAR movieClip: RgnHandle): ComponentResult;
FUNCTION MCSetControllerPort
    (mc: MovieController; gp: CGrafPtr):
        ComponentResult;
FUNCTION MCGetControllerPort
    (mc: MovieController): CGrafPtr;

```

Handling Movie Events

```

FUNCTION MCIsPlayerEvent    (mc: MovieController; e: EventRecord):
    ComponentResult;
FUNCTION MCDoAction         (mc: MovieController; action: Integer;
    params: Ptr): ComponentResult;
PROCEDURE MCSetActionFilterWithRefCon
    (mc: MovieController; filter: MCActionFilter;
    refCon: LongInt);
FUNCTION MCGetControllerInfo
    (mc: MovieController; VAR someFlags: LongInt):
        ComponentResult;
FUNCTION MCMovieChanged     (mc: MovieController; theMovie: Movie):
    ComponentResult;

```

Editing Movies

```

FUNCTION MCEnableEditing    (mc: MovieController; enabled: Boolean):
    ComponentResult;
FUNCTION MCIsEditingEnabled
    (mc: MovieController): LongInt;
FUNCTION MCCut              (mc: MovieController): Movie;
FUNCTION MCCopy             (mc: MovieController): Movie;
FUNCTION MCPaste            (mc: MovieController; srcMovie: Movie):
    ComponentResult;

```


Movie Controller Components

```

FUNCTION MCClear          (mc: MovieController): ComponentResult;
FUNCTION MCUndo           (mc: MovieController): ComponentResult;
FUNCTION MCSetUpEditMenu  (mc: MovieController; modifiers: LongInt;
                           mh: MenuHandle): ComponentResult;
FUNCTION MCGetMenuString  (mc: MovieController; modifiers: LongInt;
                           item: Integer; VAR aString: Str255):
                           ComponentResult;

```

Getting and Setting Movie Controller Time

```

FUNCTION MCSetDuration    (mc: MovieController; duration: TimeValue):
                           ComponentResult;
FUNCTION MCGetCurrentTime (mc: MovieController; VAR scale: TimeScale):
                           TimeValue;

```

Customizing Event Processing

```

FUNCTION MCActivate       (mc: MovieController; w: WindowPtr;
                           activate: Boolean): ComponentResult;
FUNCTION MCClick           (mc: MovieController; w: WindowPtr;
                           where: Point; when: LongInt;
                           modifiers: LongInt): ComponentResult;
FUNCTION MCDraw            (mc: MovieController; w: WindowPtr):
                           ComponentResult;
FUNCTION MCIdle            (mc: MovieController): ComponentResult;
FUNCTION MCKey             (mc: MovieController; key: Byte;
                           modifiers: LongInt): ComponentResult;

```

Application-Defined Routine

```

FUNCTION MyPlayerFilterWithRefCon
                           (mc: MovieController; VAR action: Integer;
                           VAR params: LongInt; refCon: LongInt): Boolean;

```

Result Codes

badControllerHeight	-9994	Invalid height
editingNotAllowed	-9995	Controller does not support editing
controllerBoundsNotExact	-9996	Boundary rectangle not exact
cannotSetWidthOfAttachedController	-9997	Cannot change controller width
controllerHasFixedHeight	-9998	Cannot change controller height
cannotMoveAttachedController	-9999	Cannot move attached controllers

