I N S I D E   M A C I N T O S H

Programming With the Mac OS 8.5 Control Manager

January 20, 1999
Technical Publications

Contents

Using the Mac OS 8.5 Control Manager

Contents

Your program can use the Control Manager to create and manage controls, onscreen objects that the user can manipulate with the mouse. By manipulating controls, the user can take an immediate action or change settings to modify a future action.

This document describes the Control Manager application programming interface (API) introduced with Mac OS 8.5 and Appearance Manager 1.1. Note that pre-existing Control Manager functions, types, and constants are not discussed in this document. For a description of the Mac OS 8 Control Manager API, see *Mac OS 8 Control Manager Reference*. For descriptions of the pre–Mac OS 8 Control Manager API, see *Inside Macintosh: Macintosh Toolbox Essentials*.

See the following sections for discussions of some programming topics for the Mac OS 8.5 Control Manager.

- “Creating a Proportional Scroll Box” (page 7)
- “Validating Editable Text” (page 8)

Creating a Proportional Scroll Box

Your application should call the function `SetControlViewSize` (page 26) to support proportional scroll boxes. If the user selects the systemwide Appearance preference for “Smart Scrolling” and your application doesn’t call `SetControlViewSize`, your application displays the traditional square scroll boxes.

To support a proportional scroll box, simply pass the size of the view area—in terms of whatever units the scroll bar uses—to `SetControlViewSize`. The system automatically handles resizing the scroll box, once your application supplies this information. Listing 0-1 shows some typical code for setting a scroll bar according to a TextEdit handle, which includes making the scroll box proportional.

Listing 1-1 Adjusting a scroll bar to the viewable text

```
static pascal void AdjustScrollBarToText
    (TEHandle teh, ControlHandle scrollBar)
{
    // get the values needed to reflect the state of the TEHandle
    short nLines    = (**teh).nLines; // number of lines in doc.
    long totalHeight= TEGetHeight (nLines, 1, teh); // total doc. height
    Rect viewRect   = (**teh).viewRect; // visible area of doc.
    Rect destRect   = (**teh).destRect; // total area of doc.
    short viewHeight= viewRect.bottom - viewRect.top; // vis. doc. height

    // set the min, max, and current value of the scroll bar
    SetControl32BitMinimum (scrollBar, 1);
    SetControl32BitMaximum (scrollBar, totalHeight - viewHeight);
    SetControl32BitValue (scrollBar, viewRect.top - destRect.top);

    // set the scroll bar view size to create a proportional scroll box
    SetControlViewSize (scrollBar, viewHeight);
}
```

Validating Editable Text

Your application typically uses a `MyControlEditTextValidationProc` (page 30) function in conjunction with a key filter function to ensure that editable text is valid in cases such as a cut, paste, or clear, where a key filter cannot be called.

Note that if you are using the inline input editable text control variant, the Control Manager will not call your `MyControlEditTextValidationProc` function during inline input. Instead, you may install your own Text Services Manager `TSMTEPostUpdateUPP` callback function to validate text during inline input, or your application can validate the input itself, immediately prior to using the text.

Listing 0-2 shows how you can use a `MyControlEditTextValidationProc` function to ensure that a user-supplied file name does not contain any illegal characters. Note that to enhance readability, no error-checking is included in this sample; a real application would, however, check for errors.

Listing 1-2 Validating a file name with a MyControlEditTextValidationProc function

```
pascal void MyControlEditTextValidationProc (ControlHandle control)
{
    Str31 text;
    Size actualSize;
    UInt8 i;

    // Get the text to be examined from the control.
    GetControlData (control, kControlNoPart, kControlEditTextTextTag,
        sizeof(Str31) - 1, (Ptr)&text[1], &actualSize);

    // Set the string’s length byte appropriately for the number of
    // characters in the text, limited to the (current) max filename.
    if (actualSize <= 31)
        text[0] = actualSize;
    else
        text[0] = 31;

    // Replace any colons with dashes.
    // Note: This only works with Roman script systems!
    for (i = 1; i <= text[0]; i++)
    {
        if (text[i] == ':')
            text[0] = '-';
    }

    // If this were a real app, there’d be code here to check to see
    // whether any text was actually replaced before bothering to redraw.

    // Put the replaced text into the control and redraw.
    SetControlData(control, kControlNoPart, kControlEditTextTextTag,
        text[0], (Ptr)&text[1]);
    DrawOneControl(control);
}
```

Mac OS 8.5 Control Manager Reference

Contents

This chapter describes the Control Manager application programming interface (API) introduced with Mac OS 8.5 and Appearance Manager 1.1, as follows:

- "Gestalt Selector for the Mac OS 8.5 Control Manager" (page 13)
- "Functions for the Mac OS 8.5 Control Manager" (page 14)
- "Application-Defined Functions for the Mac OS 8.5 Control Manager" (page 30)
- "Data Types for the Mac OS 8.5 Control Manager" (page 32)
- "Constants for the Mac OS 8.5 Control Manager" (page 34)
- "Result Codes for the Mac OS 8.5 Control Manager" (page 45)

Note that pre-existing Control Manager functions, types, and constants are not discussed in this document. For a description of the Mac OS 8 Control Manager API, see *Mac OS 8 Control Manager Reference*. For descriptions of the pre–Mac OS 8 Control Manager API, see *Inside Macintosh: Macintosh Toolbox Essentials*.

Gestalt Selector for the Mac OS 8.5 Control Manager

Before calling any functions dependent upon the Control Manager, your application should pass the selector `gestaltControlMgrAttr` to the `Gestalt` function to determine which Control Manager functions are available.

```
enum {
    gestaltControlMgrAttr   = 'cntl',
    gestaltControlMgrPresent= (1L << 0)
};
```

Constant descriptions

`gestaltControlMgrAttr`
: The `Gestalt` selector passed to determine what features of the Control Manager are present. This selector is available with Mac OS 8.5 and later. The `Gestalt` function produces a 32-bit value whose bits you should test to determine what Control Manager functionality is available.

`gestaltControlMgrPresent`
If the bit specified by this mask is set, the Control Manager functionality for Appearance Manager 1.1 is available. This bit is set for Mac OS 8.5 and later.

Functions for the Mac OS 8.5 Control Manager

The Mac OS 8.5 Control Manager provides new functions in the following areas:

- "Changing Control Settings" (page 14)
- "Associating Data With Controls" (page 20)
- "Displaying Controls" (page 25)
- "Validating Controls" (page 28)
- "Obtaining Control Part Regions" (page 29)

Changing Control Settings

The Mac OS 8.5 Control Manager provides the following functions for changing control settings:

- `SetControl32BitValue` (page 19) changes the current setting of a control and redraws it accordingly.
- `GetControl32BitValue` (page 16) obtains the current setting of a control.
- `SetControl32BitMinimum` (page 18) changes the minimum setting of a control and, if appropriate, redraws it accordingly.
- `GetControl32BitMinimum` (page 15) obtains the minimum setting of a control.
- `SetControl32BitMaximum` (page 17) changes the maximum setting of a control and, if appropriate, redraws it accordingly.
- `GetControl32BitMaximum` (page 15) obtains the maximum setting of a control.

GetControl32BitMaximum

Obtains the maximum setting of a control.

```
pascal SInt32 GetControl32BitMaximum (
                        ControlHandle theControl);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose maximum setting you wish to obtain.
function result	A signed 32-bit integer equal to the maximum setting of the control.

DISCUSSION

Your application may use the `GetControl32BitMaximum` function to obtain a 32-bit value previously set with the function `SetControl32BitMaximum` (page 17).

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control maximum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMaximum` or by accessing the `contrlMax` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

“Settings Values for Standard Controls” in *Mac OS 8 Control Manager Reference*.

GetControl32BitMinimum

Obtains the minimum setting of a control.

```
pascal SInt32 GetControl32BitMinimum (
                        ControlHandle theControl);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose minimum setting you wish to obtain.
function result	A signed 32-bit integer equal to the minimum setting of the control.

DISCUSSION

Your application may use the `GetControl32BitMinimum` function to obtain a 32-bit value previously set with the function `SetControl32BitMinimum` (page 18).

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control minimum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMinimum` or by accessing the `contrlMin` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

“Settings Values for Standard Controls” in *Mac OS 8 Control Manager Reference*.

GetControl32BitValue

Obtains the current setting of a control.

```
pascal SInt32 GetControl32BitValue (
                        ControlHandle theControl);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose current setting you wish to obtain.
function result	A signed 32-bit integer equal to the current setting of the control.

DISCUSSION

Your application may use the `GetControl32BitValue` function to obtain a 32-bit value previously set with the function `SetControl32BitValue` (page 19).

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlValue` or by accessing the `contrlValue` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

“Settings Values for Standard Controls” in *Mac OS 8 Control Manager Reference*.

SetControl32BitMaximum

Changes the maximum setting of a control and, if appropriate, redraws it accordingly.

```
pascal void SetControl32BitMaximum (
                        ControlHandle theControl,
                        SInt32 newMaximum);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose maximum setting you wish to change.
`newMaximum`	A signed 32-bit integer. Pass a value specifying the new maximum setting of the control. In general, to avoid unpredictable behavior, do not set the maximum control value lower than the current minimum value.

DISCUSSION

Your application may use the `SetControl32BitMaximum` function to set a 32-bit value as the maximum setting for a control.

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control maximum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMaximum` or by accessing the `contrlMax` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitMaximum` (page 15).

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

"Settings Values for Standard Controls" in *Mac OS 8 Control Manager Reference*.

SetControl32BitMinimum

Changes the minimum setting of a control and, if appropriate, redraws it accordingly.

```
pascal void SetControl32BitMinimum (
                        ControlHandle theControl,
                        SInt32 newMinimum);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose minimum setting you wish to change.
`newMinimum`	A signed 32-bit integer. Pass a value specifying the new minimum setting of the control. In general, to avoid unpredictable behavior, do not set the minimum control value higher than the current maximum value.

DISCUSSION

Your application may use the `SetControl32BitMinimum` function to set a 32-bit value as the minimum setting for a control.

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control minimum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMinimum` or by accessing the `contrlMin` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitMinimum` (page 15).

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

“Settings Values for Standard Controls” in *Mac OS 8 Control Manager Reference*.

SetControl32BitValue

Changes the current setting of a control and redraws it accordingly.

```
pascal void SetControl32BitValue (
                    ControlHandle theControl,
                    SInt32 newValue);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose current setting you wish to change.
`newValue`	A signed 32-bit integer. Pass a value specifying the new setting of the control. If the specified value is less than the minimum setting for the control, `SetControl32BitValue` sets the current setting of the control to its minimum setting. If the specified value is greater than the maximum setting, `SetControl32BitValue` sets the control to its maximum.

DISCUSSION

Your application may use the `SetControl32BitValue` function to set a 32-bit value as the current setting for a control.

SPECIAL CONSIDERATIONS

If your application uses a 32-bit control value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlValue` or by accessing the `contrlValue` field of the `ControlRecord` structure, because the stored 16-bit value will not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitValue` (page 16).

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

"Settings Values for Standard Controls" in *Mac OS 8 Control Manager Reference*.

Associating Data With Controls

The Mac OS 8.5 Control Manager provides the following functions for associating data with controls:

- `SetControlProperty` (page 24) associates data with a control.
- `GetControlProperty` (page 21) obtains a piece of data that has been previously associated with a control.
- `GetControlPropertySize` (page 22) obtains the size of a piece of data that has previously been associated with a control.
- `RemoveControlProperty` (page 23) removes a piece of data that has been previously associated with a control.

GetControlProperty

Obtains a piece of data that has been previously associated with a control.

```
pascal OSStatus GetControlProperty (
                    ControlHandle control,
                    OSType propertyCreator,
                    OSType propertyTag,
                    UInt32 bufferSize,
                    UInt32 *actualSize,
                    void *propertyBuffer);
```

`control`	A value of type `ControlHandle`. Pass a handle to the control whose associated data you wish to obtain.
`propertyCreator`	A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.
`propertyTag`	A four-character code. Pass the application-defined code identifying the data.
`bufferSize`	An unsigned 32-bit integer. Pass a value specifying the size of the data to be obtained. If the size of the data is unknown, use the function `GetControlPropertySize` (page 22) to get the data's size. If the size specified in the `bufferSize` parameter does not match the actual size of the property, `GetControlProperty` only retrieves data up to the size specified or up to the actual size of the property, whichever is smaller, and an error is returned.
`actualSize`	A pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the associated data. You may pass `nil` for the `actualSize` parameter if you are not interested in this information.
`propertyBuffer`	A pointer to a buffer. On return, this buffer contains a copy of the data that is associated with the specified control.

function result A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

DISCUSSION

You may use the function `GetControlProperty` to obtain a copy of data previously set by your application with the function `SetControlProperty` (page 24).

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

The function `RemoveControlProperty` (page 23).

GetControlPropertySize

Obtains the size of a piece of data that has previously been associated with a control.

```
pascal OSStatus GetControlPropertySize (
                    ControlHandle control,
                    OSType propertyCreator,
                    OSType propertyTag,
                    UInt32 *size);
```

`control` A value of type `ControlHandle`. Pass a handle to the control whose associated data you wish to examine.

`propertyCreator`
A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

`propertyTag` A four-character code. Pass the application-defined code identifying the data.

`size` A pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the data.

function result A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

DISCUSSION

If you want to retrieve a piece of associated data with the function `GetControlProperty` (page 21), you will typically need to use the `GetControlPropertySize` function beforehand to determine the size of the associated data.

VERSION NOTES

Available with Mac OS 8.5 and later.

RemoveControlProperty

Removes a piece of data that has been previously associated with a control.

```
pascal OSStatus RemoveControlProperty (
                    ControlHandle control,
                    OSType propertyCreator,
                    OSType propertyTag);
```

`control` A value of type `ControlHandle`. Pass a handle to the control whose associated data you wish to remove.

`propertyCreator`
A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

`propertyTag` A four-character code. Pass the application-defined code identifying the associated data.

function result A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

DISCUSSION

Your application may dissociate data it has previously set with the `SetControlProperty` (page 24) function by calling the `RemoveControlProperty` function.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

The function `GetControlProperty` (page 21).

SetControlProperty

Associates data with a control.

```
pascal OSStatus SetControlProperty (
                    ControlHandle control,
                    OSType propertyCreator,
                    OSType propertyTag,
                    UInt32 propertySize,
                    void *propertyData);
```

`control` A value of type `ControlHandle`. Pass a handle to the control with which you wish to associate data.

`propertyCreator`
A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not

have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

`propertyTag`	A four-character code. Pass a value identifying the data. You define the tag your application uses to identify the data.
`propertySize`	An unsigned 32-bit integer. Pass a value specifying the size of the data.
`propertyData`	A pointer to data of any type. Pass a pointer to a buffer containing the data to be associated; this buffer should be at least as large as the value specified in the `propertySize` parameter.
function result	A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

DISCUSSION

Your application may use the `SetControlProperty` function to associate any type of data with a control.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

The function `GetControlProperty` (page 21).

The function `RemoveControlProperty` (page 23).

Displaying Controls

The Mac OS 8.5 Control Manager provides the following functions for displaying controls:

- `GetControlViewSize` (page 26) obtains the size of the content to which a control's size is proportioned.
- `SetControlViewSize` (page 26) informs the Control Manager of the size of the content to which a control's size is proportioned.

- `SetUpControlTextColor` (page 27) prepares a control to be drawn with a text color consistent with that of any controls in which it is embedded and with the current theme.

GetControlViewSize

Obtains the size of the content to which a control's size is proportioned.

```
pascal SInt32 GetControlViewSize (
                        ControlHandle theControl);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose view size you wish to obtain.
function result	A signed 32-bit integer. The `GetControlViewSize` function returns a value equal to the current size of the content being displayed, expressed in terms of the same units of measurement as are used for the minimum, maximum, and current settings of the control.

DISCUSSION

Your application should call the `GetControlViewSize` function to obtain the current view size of a control. This value is used by the function `SetControlViewSize` (page 26) to support proportional scroll boxes.

VERSION NOTES

Available with Mac OS 8.5 and later.

SetControlViewSize

Informs the Control Manager of the size of the content to which a control's size is proportioned.

```
pascal void SetControlViewSize (
                        ControlHandle theControl,
                        SInt32 newViewSize);
```

`theControl`	A value of type `ControlHandle`. Pass a handle to the control whose view size is to be set.
`newViewSize`	A signed 32-bit integer. Pass a value specifying the size of the content being displayed. This value should be expressed in terms of the same units of measurement as are used for the minimum, maximum, and current settings of the control.

DISCUSSION

Your application should call the `SetControlViewSize` function to support proportional scroll boxes. If the user selects the systemwide Appearance preference for proportional scroll boxes and your application doesn't call `SetControlViewSize`, it will still have the traditional square scroll boxes.

To support a proportional scroll box, simply pass the size of the view area—in terms of whatever units the scroll bar uses—to `SetControlViewSize`. The system automatically handles resizing the scroll box, once your application supplies this information. Listing 0-1 in "Creating a Proportional Scroll Box" (page 7) shows some typical code for setting a scroll bar according to a TextEdit handle, which includes making the scroll box proportional.

VERSION NOTES

Available with Mac OS 8.5 and later.

SEE ALSO

The function `GetControlViewSize` (page 26).

SetUpControlTextColor

Prepares a control to be drawn with a text color consistent with that of any controls in which it is embedded and with the current theme.

```
pascal OSErr SetUpControlTextColor (
                    ControlHandle inControl,
                    SInt16 inDepth,
                    Boolean inIsColorDevice);
```

`inControl`	A value of type `ControlHandle`. Pass a handle to the control whose text color is to be set.
`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in pixels) of the current graphics port.
`inIsColorDevice`	A value of type `Boolean`. Set to `true` to indicate that you are drawing on a color device. Set to `false` for a monochrome device.
function result	A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

DISCUSSION

Your control definition function may use the `SetUpControlTextColor` function inside a `DeviceLoop` drawing procedure to set a control's text color to coordinate with the current theme. Applications do not typically need to use this function.

VERSION NOTES

Available with Mac OS 8.5 and later.

Validating Controls

The Mac OS 8.5 Control Manager provides the following function for confirming the validity of a control handle:

- `IsValidControlHandle` (page 28) reports whether a given handle is a control handle.

IsValidControlHandle

Reports whether a given handle is a control handle.

```
pascal Boolean IsValidControlHandle (
                        ControlHandle theControl);
```

`theControl`	A value of type `ControlHandle`. Pass the handle to be examined.

function result A value of type `Boolean`. The `IsValidControlHandle` function returns `true` if the specified handle is a valid control handle; otherwise, `false`.

DISCUSSION

The `IsValidControlHandle` function confirms whether a given handle is a valid control handle, but it does not check the validity of the data contained in the control itself.

VERSION NOTES

Available with Mac OS 8.5 and later.

Obtaining Control Part Regions

The Mac OS 8.5 Control Manager provides the following function for obtaining the region corresponding to a control part:

- `GetControlRegion` (page 29) obtains the region corresponding to a given control part.

GetControlRegion

Obtains the region corresponding to a given control part.

```
pascal OSStatus GetControlRegion (
                    ControlHandle inControl,
                    ControlPartCode inPart,
                    RgnHandle outRegion);
```

`inControl`	A value of type `ControlHandle`. Pass a handle to the control containing the part for which a region is to be obtained.
`inPart`	A value of type `ControlPartCode`. Pass a constant identifying the control part for which a region is to obtained; you may specify the `kControlStructureMetaPart` and `kControlContentMetaPart` control part codes, as well as the standard control part codes. See “Control Part Code Constants” (page 44) for descriptions of possible values.

`outRegion` Pass a value of type `RgnHandle`. On return, `GetControlRegion` sets the region to contain the actual dimensions and position of the control part, in local coordinates.

function result A result code. See "Result Codes for the Mac OS 8.5 Control Manager" (page 45).

VERSION NOTES

Available with Mac OS 8.5 and later.

Application-Defined Functions for the Mac OS 8.5 Control Manager

The Mac OS 8.5 Control Manager provides the following function for validating the content of an editable text control:

■ `MyControlEditTextValidationProc` (page 30) validates editable text.

MyControlEditTextValidationProc

Ensures that the content of an editable text control is valid.

This is how you would declare an editable text validation function, if you were to name the function `MyControlEditTextValidationProc`:

```
pascal void MyControlEditTextValidationProc (
                        ControlHandle control);
```

`control` A value of type `ControlHandle`. You are passed a handle to the control containing the editable text to be validated.

DISCUSSION

Your application typically uses a `MyControlEditTextValidationProc` function in conjunction with a key filter function to ensure that editable text is valid in cases such as a cut, paste, or clear, where a key filter cannot be called. Use the `kControlEditTextValidationProcTag` control data tag constant, described in

"Control Data Tag Constants" (page 34), with the functions `SetControlData` and `GetControlData` to set or retrieve a `MyControlEditTextValidationProc` function.

Note that if you are using the inline input editable text control variant, the Control Manager will not call your `MyControlEditTextValidationProc` function during inline input. Instead, you may install your own Text Services Manager `TSMTEPostUpdateUPP` callback function to validate text during inline input, or your application can validate the input itself, immediately prior to using the text.

Listing 0-2 in "Validating Editable Text" (page 8) demonstrates how you can use a `MyControlEditTextValidationProc` function to ensure that a user-supplied file name does not contain any illegal characters.

When you implement your editable text validation function, the pointer that you pass to `SetControlData` must be a universal procedure pointer of the following type:

```
typedef ControlEditTextValidationProcPtr ControlEditTextValidationUPP;
```

To create a universal procedure pointer for your application-defined editable text validation function, you should use the `NewControlEditTextValidationProc` macro as follows:

```
ControlEditTextValidationUPP myControlEditTextValidationUPP;
myControlEditTextValidationUPP = NewControlEditTextValidationProc
(MyControlEditTextValidationProc);
```

You can then pass `myControlEditTextValidationUPP` in the `inData` parameter of `SetControlData`. When you no longer need the universal procedure pointer, you should remove it using the `DisposeRoutineDescriptor` function.

If you need to call your application-defined function from outside your application for some reason (for example, from a plug-in), you should use the `CallControlEditTextValidationProc` macro to make the call, as follows:

```
ControlEditTextValidationUPP myControlEditTextValidationUPP;
CallControlEditTextValidationProc (myControlEditTextValidationUPP,
control);
```

Using this macro ensures that the call is made through a universal procedure pointer.

VERSION NOTES

Available with Mac OS 8.5 and later.

Data Types for the Mac OS 8.5 Control Manager

The following data types are available with the Mac OS 8.5 Control Manager:

- `ControlApplyTextColorRec` (page 32)
- `ControlEditTextValidationProcPtr` (page 33)
- `ControlGetRegionRec` (page 33)

ControlApplyTextColorRec

If you implement a custom control definition function, when the Control Manager passes the message `kControlMsgApplyTextColor` in your control definition function's `message` parameter, it also passes a pointer to a structure of type `ControlApplyTextColorRec` in the `param` parameter. The Control Manager sets the `ControlApplyTextColorRec` structure to contain data describing the current drawing environment, and your control definition function is responsible for using that data to apply the proper text color to the current graphics port.

See "Control Definition Message Constants" (page 42) for more details on the `kControlMsgApplyTextColor` message.

```
struct ControlApplyTextColorRec
{
    SInt16  depth;
    Boolean colorDevice;
    Boolean active;
};
typedef struct ControlApplyTextColorRec ControlApplyTextColorRec;
typedef ControlApplyTextColorRec* ControlApplyTextColorPtr;
```

Field descriptions

`depth`	A signed 16-bit integer. The Control Manager sets this field to specify the bit depth (in pixels) of the current graphics port.
`colorDevice`	A value of type `Boolean`. The Control Manager passes a value of `true` if you are drawing on a color device; otherwise, `false`.
`active`	A value of type `Boolean`. The Control Manager passes a value of `true` to specify a color suitable for active text; otherwise, `false`.

ControlEditTextValidationProcPtr

You may want to designate an application-defined function in conjunction with a key filter function to ensure that editable text is valid in cases such as a cut, paste, or clear, where a key filter cannot be called. Such a function has the following type definition:

```
typedef pascal void (ControlEditTextValidationProcPtr) (ControlHandle
control);
```

See `MyControlEditTextValidationProc` (page 30) for more information about how to implement the application-defined function.

ControlGetRegionRec

If you implement a custom control definition function, when the Control Manager passes the message `kControlMsgGetRegion` in your control definition function's `message` parameter, it also passes a pointer to a structure of type `ControlGetRegionRec` in the `param` parameter. Your control definition function is responsible for setting the `region` field of the `ControlGetRegionRec` structure to the region that contains the control part which the Control Manager specifies in the `part` field.

See "Control Definition Message Constants" (page 42) for more details on the `kControlMsgGetRegion` message.

```
struct ControlGetRegionRec
{
    RgnHandle       region;
    ControlPartCode part;
};
typedef struct ControlGetRegionRec ControlGetRegionRec;
typedef ControlGetRegionRec* ControlGetRegionPtr;
```

Field descriptions

`region`	A value of type `RgnHandle` allocated by the Control Manager. Your control definition function should set this field to the region that contains the control part specified in the `part` field.
`part`	A value of type `ControlPartCode`. The Control Manager passes a constant identifying the control part for which a region is to be obtained. For descriptions of possible values, see “Control Part Code Constants” (page 44).

Constants for the Mac OS 8.5 Control Manager

The following constants are available with the Mac OS 8.5 Control Manager:

- “Control Data Tag Constants” (page 34)
- “Control Definition Feature Constants” (page 38)
- “Control Definition IDs” (page 39)
- “Control Definition Message Constants” (page 42)
- “Control Font Style Flag Constant” (page 43)
- “Control Key Script Behavior Constants” (page 43)
- “Control Part Code Constants” (page 44)

Control Data Tag Constants

The Mac OS 8.5 Control Manager defines the following new control data tag constants. These constants are passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data (for example, text in an editable text control). These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The descriptions below show the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

For details on other control data tag constants, see *Mac OS 8 Control Manager Reference.*

```
enum {
    kControlScrollTextBoxDelayBeforeAutoScrollTag   = 'stdl',
    kControlScrollTextBoxDelayBetweenAutoScrollTag  = 'scdl',
    kControlScrollTextBoxAutoScrollAmountTag        = 'samt',
    kControlScrollTextBoxContentsTag                = 'tres',
    kControlEditTextKeyScriptBehaviorTag            = 'kscr',
    kControlEditTextLockedTag                       = 'lock',
    kControlEditTextFixedTextTag                    = 'ftxt',
    kControlEditTextValidationProcTag               = 'vali',
    kControlEditTextInlinePreUpdateProcTag          = 'prup',
    kControlEditTextInlinePostUpdateProcTag         = 'poup',
    kControlStaticTextTruncTag                      = 'trun',
    kControlIconResourceIDTag                       = 'ires',
    kControlIconContentTag                          = 'cont',
    kControlBevelButtonScaleIconTag                 = 'scal',
    kControlPushButtonCancelTag                     = 'cncl',
    kControlPopupButtonExtraHeightTag               = 'exht',
    kControlGroupBoxTitleRectTag                    = 'trec'
};
```

Constant descriptions

`kControlScrollTextBoxDelayBeforeAutoScrollTag`

Gets or sets the number of ticks to delay before the initial scrolling of an auto-scrolling text box control begins. Data type retrieved or set: `UInt32`

`kControlScrollTextBoxDelayBetweenAutoScrollTag`
: Gets or sets the number of ticks to delay between each unit of scrolling, for an auto-scrolling text box control. (The unit of scrolling for the auto-scrolling text box control is one pixel at a time, unless your application changes this value by calling the `SetControlData` function.)
Data type retrieved or set: `UInt32`

`kControlScrollTextBoxAutoScrollAmountTag`
: Gets or sets the number of pixels by which an auto-scrolling text box control scrolls; default is 1.
Data type retrieved or set: `UInt16`

`kControlScrollTextBoxContentsTag`
: Sets the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used as the content in a scrolling or auto-scrolling text box control.
Data type set: `SInt16`

`kControlEditTextKeyScriptBehaviorTag`
: Gets or sets the kind of behavior to be used in an editable text control with respect to changing and locking the keyboard menu as the field is focused.
Data type retrieved or set: `ControlKeyScriptBehavior`. The default for password fields is `kControlKeyScriptBehaviorPrefersRoman`. The default for non-password fields is `kControlKeyScriptBehaviorAllowAnyScript`. See "Control Key Script Behavior Constants" (page 43) for descriptions of possible values.

`kControlEditTextLockedTag`
: Gets or sets whether the text in an editable text control is currently editable.
Data type retrieved or set: `Boolean`; if `true`, the text is locked and cannot be edited; if `false`, the text is editable.

`kControlEditTextFixedTextTag`
: Gets or sets inline input text in an editable text control, after confirming any text in the active input area with the Text Services Manager function `FixTSMDocument`.
Data type retrieved or set: character buffer

`kControlEditTextValidationProcTag`
: Gets or sets a universal procedure pointer to a callback

function such as that described in `MyControlEditTextValidationProc` (page 30), which can be used to validate editable text after an operation that changes the text, such as inline text entry, a cut, or paste.
Data type retrieved or set: `ControlEditTextValidationUPP`

`kControlEditTextInlinePreUpdateProcTag`
Gets or sets a universal procedure pointer to a Text Services Manager pre-update callback function. See Technote TE 27, "Inline Input for TextEdit with TSMTE" for a description of the `TSMTEPreUpdateUPP` type.
Data type retrieved or set: `TSMTEPreUpdateUPP`

`kControlEditTextInlinePostUpdateProcTag`
Gets or sets a universal procedure pointer to a Text Services Manager post-update callback function. See Technote TE 27, "Inline Input for TextEdit with TSMTE" for a description of the `TSMTEPostUpdateUPP` type.
Data type retrieved or set: `TSMTEPostUpdateUPP`

`kControlStaticTextTruncTag`
Gets or sets how text is truncated at the end of a line for a static text control.
Data type retrieved or set: `TruncCode`; the value `truncEnd` indicates that characters are truncated off the end of the string; the value `truncMiddle` indicates that characters are truncated from the middle of the string. Default is a value of -1, which indicates that no truncation occurs and the text is instead wrapped.

`kControlIconResourceIDTag`
Gets or sets the resource ID of the icon to use.
Data type retrieved or set: `SInt16`

`kControlIconContentTag`
Gets or sets the type of content to be used in an icon control.
Data type retrieved or set: `ControlButtonContentInfo`; see *Mac OS 8 Control Manager Reference* for a description of this data type.

`kControlBevelButtonScaleIconTag`
Gets or sets whether, when the proper icon size is unavailable, an icon should be scaled for use with a given bevel button. This tag is only for use with icon suites or the

IconRef data type.
Data type retrieved or set: Boolean. If true, indicates that if an icon of the ideal size isn't available, a larger or smaller icon should be scaled to the ideal size. If false, no scaling should occur; instead, a smaller icon should be drawn or a larger icon clipped. Default is false.

kControlPushButtonCancelTag
Gets or sets whether a given push button in a dialog or alert should be drawn with the theme-specific adornments for a Cancel button.
Data type retrieved or set: Boolean; default is false.

kControlPopupButtonExtraHeightTag
Gets or sets the amount of extra vertical white space in a pop-up menu button.
Data type retrieved: SInt16; default is 0.

kControlGroupBoxTitleRectTag
Gets the rectangle that contains the title of a group box (and any associated control, such as a checkbox or other button).
Data type retrieved: Rect

Control Definition Feature Constants

With the Mac OS 8.5 Control Manager, your control definition function may report the following new feature flags to reflect the features that your control supports. For other control definition feature flags, see "Defining Your Own Control Definition Function" in *Mac OS 8 Control Manager Reference.*

```
enum {
    kControlAutoToggles          = 1 << 14,
    kControlSupportsGetRegion    = 1 << 17
};
```

Constant descriptions

kControlAutoToggles
If the bit specified by this mask is set, the control definition function supports automatically changing among various states (on, off, mixed) in response to user actions.

kControlSupportsGetRegion
If the bit specified by this mask is set, the control definition

function supports the `kControlMsgGetRegion` message, described in “Control Definition Message Constants” (page 42).

Control Definition IDs

The Mac OS 8.5 Control Manager defines the following new control definition IDs. For details on other standard control definition IDs, see *Mac OS 8 Control Manager Reference*.

```
enum {
    kControlEditTextInlineInputProc     = 276,
    kControlIconRefProc                 = 324,
    kControlIconRefNoTrackProc          = 325,
    kControlCheckBoxAutoToggleProc      = 371,
    kControlRadioButtonAutoToggleProc   = 372,
    kControlScrollTextBoxProc           = 432,
    kControlScrollTextBoxAutoScrollProc = 433
};
```

Constant descriptions

`kControlEditTextInlineInputProc`
: Identifies the inline input variant of the editable text control (`'CDEF'` resource ID 17), which supports 2-byte script systems. This variant cannot be combined with the password variant of the editable text box.

`kControlIconRefProc`
: Identifies the variant of the icon control (`'CDEF'` resource ID 20) that supports all standard types of icon-based content. Note that you do not supply content for this control upon its creation with a call to the `NewControl` function. Rather, after the control’s creation you can set or change its content at any time by passing the `SetControlData` function the `kControlIconContentTag` control data tag constant and a `ControlButtonContentInfo` structure containing any of the allowable data types. Supported data types for this icon control variant are specified with the following `ControlContentType` values: `kControlContentIconSuiteRes`, `kControlContentCIconRes` (uses a black-and-white `'ICON'` resource if the color resource isn’t available),

`kControlContentIconSuiteHandle`, `kControlContentCIconHandle`, and `kControlContentIconRef`. Note, too, that if you supply the `kControlContentIconRef` value, you must first use Icon Services functions to register your resources and generate `IconRef` values. See *Mac OS 8 Control Manager Reference* for a description of the `ControlButtonContentInfo` data type and the `ControlContentType` constants.

`kControlIconRefNoTrackProc`

Identifies the non-tracking variant of the icon control (`'CDEF'` resource ID 20) that supports all standard types of icon-based content. This control immediately returns `kControlIconPart` as the part code hit without tracking. Note that you do not supply content for this control upon its creation with a call to the `NewControl` function. Rather, after the control's creation you can set or change its content at any time by passing the `SetControlData` function the `kControlIconContentTag` control data tag constant and a `ControlButtonContentInfo` structure containing any of the allowable data types. Supported data types for this icon control variant are specified with the following `ControlContentType` values: `kControlContentIconSuiteRes`, `kControlContentCIconRes` (uses a black-and-white `'ICON'` resource if the color resource isn't available), `kControlContentIconSuiteHandle`, `kControlContentCIconHandle`, and `kControlContentIconRef`. Note, too, that if you supply the `kControlContentIconRef` value, you must first use Icon Services functions to register your resources and generate `IconRef` values. See *Mac OS 8 Control Manager Reference* for a description of the `ControlButtonContentInfo` data type and the `ControlContentType` constants.

`kControlCheckBoxAutoToggleProc`

Identifies a checkbox control (`'CDEF'` resource ID 23) that automatically changes among its various states (on, off, mixed) in response to user actions. Your application must only call the function `GetControl32BitValue` (page 16) to get the checkbox's new state—there is no need to manually change the control's value after tracking successfully.

`kControlRadioButtonAutoToggleProc`
: Identifies a radio button control (`'CDEF'` resource ID 23) that automatically changes among its various states (on, off, mixed) in response to user actions. Your application must only call the function `GetControl32BitValue` (page 16) to get the radio button's new state—there is no need to manually change the control's value after tracking successfully.

`kControlScrollTextBoxProc`
: Identifies the standard variant of the scrolling text box (`'CDEF'` resource ID 27), which contains a scroll bar. Your application can use the `kControlScrollTextBoxProc` ID to create a scrolling box of non-editable text, such as is used for an "About" box. You must pass the `NewControl` function the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used for the initial value of the control. The minimum and maximum values are reserved for the `kControlScrollTextBoxProc` variant and should be set to 0.

`kControlScrollTextBoxAutoScrollProc`
: Identifies the auto-scrolling variant of the scrolling text box (`'CDEF'` resource ID 27); this variant does not contain a scroll bar. Your application can use the `kControlScrollTextBoxAutoScrollProc` ID to create a scrolling box of non-editable text, such as is used for an "About" box. You must pass the `NewControl` function the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used for the initial value of the control. For the minimum value of the control, pass a value equal to the delay, in ticks, before the control begins scrolling; this delay will also be used between when scrolling completes and when it begins again. For the maximum value of the control, pass a value equal to the delay, in ticks, between each unit of scrolling. The unit of scrolling for the auto-scrolling text box control is one pixel at a time, unless your application changes this value by calling the `SetControlData` function. Note that in order to advance the content of the text box—that is, to scroll the content—you must call the `IdleControls` function on a periodic basis, such as whenever you receive a null event.

Control Definition Message Constants

The Mac OS 8.5 Control Manager may pass the following new constants in the `message` parameter of your control definition function to specify the actions that your function must perform. For other control definition message constants, see "Defining Your Own Control Definition Function" in *Mac OS 8 Control Manager Reference*.

```
enum {
    kControlMsgApplyTextColor   = 30,
    kControlMsgGetRegion        = 31
};
```

Constant descriptions

`kControlMsgApplyTextColor`
: Set the foreground color to be consistent with the current drawing environment and suitable for display against the background color or pattern. To indicate that your control definition function supports this message, set the `kControlHasSpecialBackground` feature bit. When this message is sent, the Control Manager passes a pointer to a structure of type `ControlApplyTextColorRec` (page 32) in your control definition function's `param` parameter. The Control Manager sets the `ControlApplyTextColorRec` structure to contain data describing the current drawing environment. Your control definition function is responsible for using this data to apply a text color which is consistent with the current theme and optimally readable on the control's background. Your control definition function should return 0 as the function result.

`kControlMsgGetRegion`
: Obtain the region occupied by the specified control part. To indicate that your control definition function supports this message, set the `kControlSupportsGetRegion` feature bit. When this message is sent, the Control Manager passes a pointer to a structure of type `ControlGetRegionRec` (page 33) in your control definition function's `param` parameter. Your control definition function is responsible for setting the `region` field of the `ControlGetRegionRec` structure to the region that contains the control part which the Control Manager specifies in the `part` field. Your control definition

function return a result code of type `OSStatus` as the function result.

Control Font Style Flag Constant

With the Mac OS 8.5 Control Manager, you can pass the following new control font style flag constant in the `flags` field of the `ControlFontStyleRec` structure to specify the fields of the structure that should be applied to the control. For more details on control font style flag constants and the `ControlFontStyleRec` structure, see *Mac OS 8 Control Manager Reference*.

```
enum {
    kControlAddToMetaFontMask   = 0x0200
};
```

Constant description

`kControlAddToMetaFontMask`

If the bit specified by this mask is set, the control may use a meta-font while also adding other attributes to the font. If the bit specified by this mask is not set, but a meta-font is specified for the control, any additional attributes set for the font are ignored.

Control Key Script Behavior Constants

With the Mac OS 8.5 Control Manager, you can use the following constants of type `ControlKeyScriptBehavior` to specify the kind of behavior to be used in an editable text control with respect to changing and locking the keyboard menu as the field is focused. The `ControlKeyScriptBehavior` constants are set and retrieved with the `kControlEditTextKeyScriptBehaviorTag` control data tag constant; for details on `kControlEditTextKeyScriptBehaviorTag`, see “Control Data Tag Constants” (page 34).

```
enum {
    kControlKeyScriptBehaviorAllowAnyScript = 'any ',
    kControlKeyScriptBehaviorPrefersRoman   = 'prmn',
    kControlKeyScriptBehaviorRequiresRoman  = 'rrmn'
};
typedef UInt32 ControlKeyScriptBehavior;
```

Constant descriptions

`kControlKeyScriptBehaviorAllowAnyScript`
: Does not change the current keyboard and allows the user to change the keyboard at will. This is the default for non-password fields.

`kControlKeyScriptBehaviorPrefersRoman`
: Changes the current keyboard to Roman whenever the editable text field receives focus but allows the user to change the keyboard at will. This is the default for password fields.

`kControlKeyScriptBehaviorRequiresRoman`
: Changes the current keyboard to Roman whenever the editable text field receives focus and does not allow the user to change the keyboard.

Control Part Code Constants

The Mac OS 8.5 Control Manager defines the following new control part code constants. You can use control part codes with various functions to identify control parts. For details on other control part code constants, see *Mac OS 8 Control Manager Reference*.

```
enum {
    kControlClockHourDayPart        = 9,
    kControlClockMinuteMonthPart    = 10,
    kControlClockSecondYearPart     = 11,
    kControlClockAMPMPart           = 12,
    kControlStructureMetaPart       = -1,
    kControlContentMetaPart         = -2
};
```

Constant descriptions

`kControlClockHourDayPart`
: Identifies the part of a clock control that contains the hour or the day.

`kControlClockMinuteMonthPart`
: Identifies the part of a clock control that contains the minute or the month.

`kControlClockSecondYearPart`
Identifies the part of a clock control that contains the second or the year.

`kControlClockAMPMPart`
Identifies the part of a clock control that contains the AM/PM information.

`kControlStructureMetaPart`
Identifies the entire region occupied by a control. The `kControlStructureMetaPart` constant is for use only with `GetControlRegion` (page 29).

`kControlContentMetaPart`
Identifies the area of a control in which other controls may be embedded; this region is only defined for controls that can be embedders. The `kControlContentMetaPart` constant is for use only with `GetControlRegion` (page 29).

Result Codes for the Mac OS 8.5 Control Manager

The new result codes returned by the Mac OS 8.5 Control Manager are listed below.

`controlPropertyInvalid`	–5603	`'macs'` property signature not allowed
`controlPropertyNotFoundErr`	–5604	Specified property does not exist
`controlInvalidDataVersionErr`	–30597	Incorrect version of data passed
`controlHandleInvalidErr`	–30599	Invalid control handle

Document Version History

This document has had the following releases:

Table A-1 *Programming With the Mac OS 8.5 Control Manager* Revision History

Version	**Notes**
Jan. 12, 1999	Initial public release. The following changes were made from the prior version: Changed “Control Manager 2.0” to “Mac OS 8.5 Control Manager” throughout to reflect final versioning. “Control Definition IDs” (page 39). Added descriptions of `kControlIconRefProc` and `kControlIconRefNoTrackProc`. “Result Codes for the Mac OS 8.5 Control Manager” (page 45). Changed the `controlPropertyInvalidErr` constant to `controlPropertyInvalid`. Added “Using the Mac OS 8.5 Control Manager” chapter to contain code listings moved from “Mac OS 8.5 Control Manager Reference” chapter.
Jul. 23, 1998	First seed draft release.

A P P E N D I X

Document Version History

Index

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Donna S. Lee

PRODUCTION EDITOR
Glen Frank

Acknowledgments to Guy Fullerton, Pete Gontier, and Chris Thomas.