

## Preview Components

This chapter discusses preview components. **Preview components** are used by the Image Compression Manager's standard file preview functions to display and create visual previews for files. Previews usually consist of a single image, but they may contain many kinds of data, including sound. In QuickTime, the Image Compression Manager is the primary client of preview components. Rarely, if ever, do applications call preview components directly. However, you may want to develop your own preview component for use by the Image Compression Manager. Therefore, this chapter focuses on what you must do to create a preview component.

- “About Preview Components” presents some general information about how preview components obtain, store, and use preview data.
- “Creating Preview Components” presents a sample program for the implementation of a preview component that displays PICT images.
- “Preview Components Reference” describes the functions and resources that are specific to preview components.
- “Summary of Preview Components” provides summaries of the preview component constants, data structures, and functions in C and in Pascal.

Before learning about preview components, you must be familiar with QuickTime movie previews. See the chapter “Movie Toolbox” in *Inside Macintosh: QuickTime* for a complete description of movie previews and of the Image Compression Manager functions that support standard file previews.

## About Preview Components

---

Preview components provide two basic services: they draw and create previews. This section describes how preview components obtain preview data, what kind of information is stored with the file, and what they do with the preview data.

### Obtaining Preview Data

---

Preview components obtain data from

- a small data cache
- a reference they create to another resource in the file
- the file for which they are invoked

## Preview Components

The preview component can create a small data cache containing the preview. Although creation of the preview cache may be time-consuming, the cache can then be stored in the file and used to display the preview for the file rapidly on subsequent occasions. The picture file preview component, which creates a thumbnail picture for the file and stores it in the file's resource fork, is one way of getting information from a data cache.

The preview component can create a reference to another resource in the file. For example, some file types already contain a picture preview in them. The preview component can then create a pointer to that existing data, rather than making another copy of it. The movie preview component works in this way when the preview for the movie is actually the movie's preview, rather than only its poster picture.

If the preview component can display the preview for the file quickly enough in every case, there is no need for a cache. Such a preview component reinterprets the data in the file each time it is invoked, rather than creating a preview cache once. This method of getting the information allows the file to remain untouched, requires no disk space, and does not demand that the user or the application make any special effort to create the preview. Unfortunately, in most cases, it is not possible to interpret the data quickly enough to use this approach. Preview components that handle this type of preview should set the `pnotComponentNeedsNoCache` flag in their component flags field.

```
enum {
    pnotComponentNeedsNoCache = 2
};
```

If a preview component relies on other system software services, it must make sure they are present. For example, if your preview component uses the Movie Toolbox, it is responsible for calling the Movie Toolbox's `EnterMovies` and `ExitMovies` functions.

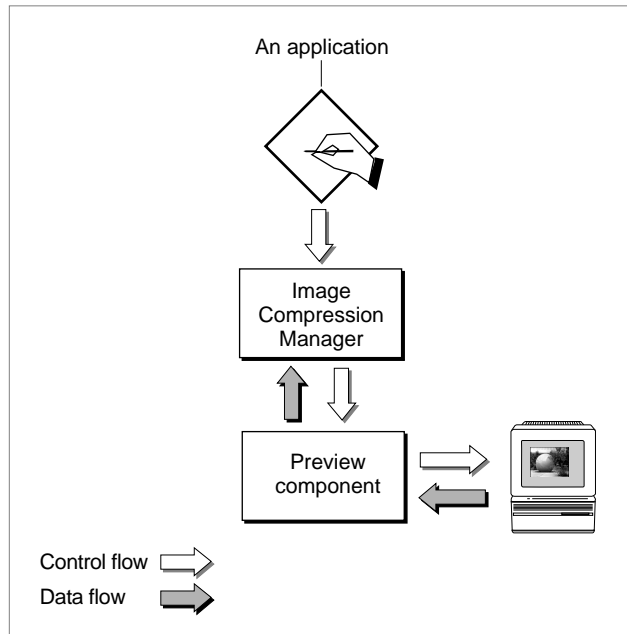
When previewing is complete, the component receives a normal Component Manager close request. If you add any controls to the window, you should dispose of them while you are calling the Component Manager's `CloseComponent` function.

A preview component should never write back to the file directly. The caller of the preview component is responsible for actually modifying the file. You should open all access paths to the file with read permission only.

## Preview Components

Figure 12-1 illustrates the relationships of a preview component, the Image Compression Manager, and an application.

**Figure 12-1** Relationships of a preview component, the Image Compression Manager, and an application



## Storing Preview Data in Files

A preview may or may not contain sound or text data or other types of information. In addition to the visual preview, QuickTime provides the preview resource, described on page 12-14, which also allows you to store

- a brief description of the file
- a list of keywords
- an associated language code to allow use of a single file in more than one region
- a modification date to help applications determine when the data has been changed

## Using the Preview Data

Preview components may

- create a preview
- draw a preview
- create and draw a preview

## Preview Components

Some preview components only create a preview and rely on another component to display it. For example, by default, the movie preview component creates a picture preview for the file. This is displayed by the picture preview component.

Most preview components simply draw the preview. These are the simplest type of display components. They do not require any other event processing—including the scheduling of idle time—for example, to play a movie. The picture preview component is an example of this type of component.

Preview components that do not require a cache should have a subtype that matches the type of file for which they can display previews.

A preview component for sound would require event processing, since it would need time to play the sound. If your preview component requires event processing, you must have the `pnotComponentWantsEvents` flag set in its component flags field.

```
enum {
    pnotComponentWantsEvents = 1,
};
```

## Creating Preview Components

---

This section describes how to create your own preview component.

Preview components that create previews have a type of `'pmak'` and a subtype that matches the type of the file for which they create previews.

Preview components that display previews have a type of `'pnot'` and a subtype that matches the type of the resource that they display.

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

```
enum {
    kPreviewShowDataSelector          = 1, /* PreviewShowData */
    kPreviewMakePreviewSelector       = 2, /* PreviewMakePreview */
    kPreviewMakePreviewReferenceSelector = 3, /* PreviewMakePreviewReference */
    kPreviewEventSelector             = 4  /* PreviewEvent */
};
```

This section presents a sample program that displays a preview component for the display of PICS animation files. First it implements the required Component Manager functions. Then it converts the PICT image data into a format for display as a preview.

## Implementing Required Component Functions

Listing 12-1 supplies the component dispatchers for the preview component together with the can do, version, open, and close functions.

**Listing 12-1** Implementing the required Component Manager functions

```
typedef struct {
    ComponentInstance self;
} PICSPreviewRecord, **PICSPreviewGlobals;

/* entry point for all Component Manager requests */
pascal ComponentResult PICSPreviewDispatch
    (ComponentParameters *params, Handle store)
{
    OSErr err = badComponentSelector;
    ComponentFunction componentProc = 0;

    switch (params->what) {
        case kComponentOpenSelect:
            componentProc = PICSPreviewOpen; break;
        case kComponentCloseSelect:
            componentProc = PICSPreviewClose; break;
        case kComponentCanDoSelect:
            componentProc = PICSPreviewCanDo; break;
        case kComponentVersionSelect:
            componentProc = PICSPreviewVersion; break;
        case kPreviewShowDataSelector:
            componentProc = PICSPreviewShowData; break;
    }

    if (componentProc)
        err = CallComponentFunctionWithStorage (store, params,
                                                componentProc);

    return err;
}

pascal ComponentResult PICSPreviewCanDo
    (PICSPreviewGlobals store, short ftnNumber)
{
    switch (ftnNumber) {
        case kComponentOpenSelect:
        case kComponentCloseSelect:
        case kComponentCanDoSelect:
```

## Preview Components

```

        case kComponentVersionSelect:
        case kPreviewShowDataSelector:
            return true;
        default:
            return false;
    }
}

pascal ComponentResult PICSPreviewVersion
    (PICSPreviewGlobals store)
{
    return 0x00010001;
}

pascal ComponentResult PICSPreviewOpen (PICSPreviewGlobals store,
    ComponentInstance self)
{
    store = (PICSPreviewGlobals)NewHandle
        (sizeof (PICSPreviewRecord));
    if (!store) return MemError();
    SetComponentInstanceStorage (self, (Handle)store);
    (**store).self = self;

    return noErr;
}

pascal ComponentResult PICSPreviewClose
    (PICSPreviewGlobals store,
    ComponentInstance self)
{
    if (store) DisposeHandle ((Handle)store);
    return noErr;
}

```

## Displaying Image Data as a Preview

---

To display a file's image preview, your `PreviewShowData` function is called.

Listing 12-2 includes the `PICSPreviewShowData` function, which previews a PICS file. The function loads the first PICT image from the PICS file and uses the PICT file preview component to display it.

**Listing 12-2** Converting data into a form that can be displayed as a preview

```

pascal ComponentResult PICSPreviewShowData
    (PICSPreviewGlobals store,
     OSType dataType, Handle data,
     const Rect *inHere)
{
    OSErr err = noErr;
    short resRef = 0, saveRes = CurResFile();
    FSSpec theFile;
    Boolean whoCares;
    Handle thePict = nil;
    ComponentInstance ci;

    /* because your component has the pnotComponentNeedsNoCache
       flag set, it should only be called to display files */
    if (dataType != rAliasType)
        return paramErr;

    /* open up the file to preview */
    if (err = ResolveAlias (nil, (AliasHandle)data, &theFile,
                          &whoCares)) goto bail;
    resRef = FSpOpenResFile (&theFile, fsRdPerm);
    if (err = ResError()) goto bail;

    /* get the first 'PICT' */
    UseResFile (resRef);
    thePict = Get1IndResource ('PICT', 1);
    if (!thePict) goto bail;

    /* use the PICT preview component to display the preview */
    if (ci = OpenDefaultComponent (ShowFilePreviewComponentType,
                                  'PICT')) {
        PreviewShowData (ci, 'PICT', thePict, inHere);
        CloseComponent (ci);
    }

bail:
    if (resRef) CloseResFile (resRef);
    if (thePict) DisposeHandle (thePict);
    UseResFile (saveRes);
    return err;
}

```

## Preview Components Reference

---

This section describes the functions and resources that are specific to preview components.

### Functions

---

This section describes the functions for displaying previews, handling events in previews, and creating previews that are provided by preview components. These functions are described from the perspective of the Image Compression Manager, which is most likely to call preview components. If you are developing a preview component, your component must behave as described here.

### Displaying Previews

---

The preview component supplies a single function for displaying movie previews. If your preview component does not handle events (that is, does not contain time-based data), you should use this function.

### PreviewShowData

---

The `PreviewShowData` function allows you to display a preview if your preview component does not handle events.

```
pascal ComponentResult PreviewShowData (pnotComponent p,
                                         OSType dataType,
                                         Handle data,
                                         const Rect *inHere);
```

<code>p</code>	Specifies your preview component. You obtain this identifier from the Component Manager's <code>OpenComponent</code> function. See the chapter "Component Manager" in <i>Inside Macintosh: More Macintosh Toolbox</i> for details.
<code>dataType</code>	Contains the type of handle pointing to the data to be displayed in the preview.
<code>data</code>	Contains a handle to the data, which is typically the same as the subtype of your preview component.
<code>inHere</code>	Contains a pointer to a rectangle that defines the area into which you draw the preview. The current port is set to the correct graphics port for drawing. You must not draw outside the given rectangle.



**DESCRIPTION**

If your preview component can display the data for the preview quickly enough that it does not need a cache (that is, you have set the `pnotComponentNeedsNoCache` flag), you should consider the `PreviewShowData` function an initialization function. Therefore, you should remember the location of the preview rectangle and set up any necessary data structures. An update event is generated after this function for your initial drawing. In this case, the type of the handle in the `data` parameter is an alias (that is, it is the `rAliasType` resource type), and the handle contains an alias to the file to be previewed.

**Handling Events**

The `PreviewEvent` function is provided so that your preview component can do standard event filtering. See *Inside Macintosh: Files* for details on the standard dialog event filter function.

**PreviewEvent**

If your preview component handles events, the `PreviewEvent` function is called as appropriate.

```
pascal ComponentResult PreviewEvent (pnotComponent p,
                                     EventRecord *e,
                                     Boolean *handledEvent);
```

**p** Specifies your preview component. You obtain this identifier from the Component Manager's `OpenComponent` function. See the chapter "Component Manager" in *Inside Macintosh: More Macintosh Toolbox* for details.

**e** Contains a pointer to the event structure for this operation.

**handledEvent** Contains a pointer to a Boolean value. If you completely handle an event such as a mouse-down event or keystroke, you should set the `handledEvent` parameter to `true`. Otherwise, set it to `false`.

**Creating Previews**

Two functions are available for use in creating previews. The `PreviewMakePreview` function creates previews by allocating a handle to data to be added to the file. On the other hand, the `PreviewMakePreviewReference` function makes previews by returning the type and identification number of a resource within the file to be used as the preview for the file.

## PreviewMakePreview

---

The `PreviewMakePreview` function creates previews by allocating a handle to data that is to be added to the file.

```
pascal ComponentResult PreviewMakePreview (pnotComponent p,
                                           OSType *previewType,
                                           Handle *previewResult,
                                           const FSSpec *sourceFile,
                                           ProgressProcRecordPtr progress);
```

<code>p</code>	Specifies your preview component. You obtain this identifier from the Component Manager's <code>OpenComponent</code> function. See the chapter "Component Manager" in <i>Inside Macintosh: More Macintosh Toolbox</i> for details.
<code>previewType</code>	Contains a pointer to the type of preview component that should be used to display the preview.
<code>previewResult</code>	Contains a pointer to a handle of cached preview data created by this function.
<code>sourceFile</code>	Contains a pointer to a reference to the file for which the preview is created.
<code>progress</code>	Points to a progress function. For details on progress functions, see the chapter "Image Compression Manager" in <i>Inside Macintosh: QuickTime</i> . If the process of creating a preview takes more than a few seconds, you should call the progress function that is provided.

### DESCRIPTION

Your preview component should not actually write the preview to the given file. It should simply return the handle. The data is added to the file by the caller.

## PreviewMakePreviewReference

---

Instead of creating a handle to data that is to be added to the file, the `PreviewMakePreviewReference` function returns the type and identification number of a resource within the file to be used as the preview for the file.

<pre>pascal ComponentResult PreviewMakePreviewReference                                 (pnotComponent p, OSType *previewType,                                  short *resID, const FSSpec *sourceFile);</pre>	
<code>p</code>	Specifies your preview component. You obtain this identifier from the Component Manager's <code>OpenComponent</code> function. See the chapter "Component Manager" in <i>Inside Macintosh: More Macintosh Toolbox</i> for details.
<code>previewType</code>	Contains a pointer to the type of preview component that should be used to display the preview.
<code>resID</code>	Contains a pointer to the identification number of a resource within the file to be used as the preview for the file.
<code>sourceFile</code>	Contains a pointer to a reference to the file for which the preview is created.

### DESCRIPTION

If your preview component creates previews by reference, you must also implement the `PreviewMakePreview` function, described in the previous section. However, you should return an error from it. `PreviewMakePreview` is always called first. If it fails, `PreviewMakePreviewReference` is tried next.

## Resources

---

This section describes the preview resource and the preview resource item structures. The preview component uses the preview resource to store visual preview information. The preview resource item structure stores an unlimited number of additional pieces of file data.

## The Preview Resource

---

QuickTime uses the preview resource (defined by the `pnotResource` data type) with a resource ID of 0 to store the visual preview information. The structure of the preview resource is shown in Listing 12-3.

### ▲ WARNING

If you parse this resource directly, please do extensive error checking in your code so as not to hinder future expansion of the data structure. In particular, if you encounter unknown version bits, exercise caution. Unexpected results may occur. ▲

---

**Listing 12-3**     The preview resource

```
typedef struct pnotResource {

    unsigned long   modDate;      /* modification date */
    short           version;      /* version number of preview
                                   resource */
    OSType          resType;      /* type of resource used as preview
                                   cache */
    short           resID;        /* resource identification number
                                   of resource used as preview
                                   cache */
    short           numResItems; /* number of additional file
                                   descriptions */
    pnotResItem     resItem[ ]; /* array of file descriptions */
} pnotResource;
```

### Field descriptions

<code>modDate</code>	Contains the modification time (in standard Macintosh seconds since midnight, January 1, 1904) of the file for which the preview was created. This parameter allows you to find out if the preview is out of date with the contents of the file.
<code>version</code>	Contains the version number of the preview resource. The low bit of the version is a flag for preview components that only reference their data. If the bit is set, it indicates that the resource identified in the preview resource is not owned by the preview component, but is part of the file. It is not removed when the preview is updated or removed (using the Image Compression Manager's <code>MakeFilePreview</code> or <code>AddFilePreview</code> function), as it would be if the version number were 0.
<code>resType</code>	Contains the type of a resource used as a preview cache for the given file. The type of the resource determines the subtype of the preview component that should be used to display the preview.
<code>resID</code>	Contains the identification number of a resource used as a preview cache for the specified file.

Preview Components

numResItems	Specifies the number of additional file descriptions stored with this preview.
resItem	Contains the preview resource item structure (defined by the pnotResItem data type), which is described next.

The Preview Resource Item Structure

The preview resource item structure is an array that allows you to store an unlimited number of additional pieces of file information. Each piece of data contains a reference to its information using the structure defined by the pnotResItem data type, which is shown in Listing 12-4.

**Listing 12-4** The preview resource item structure

```
typedef struct pnotResItem {
    unsigned long  modDate; /* last modification date of item */
    OSType         useType; /* what type of data */
    OSType         resType; /* resource type containing item */
    short          resID;   /* resource ID containing this item */
    short          rgnCode; /* region code */
    long           reserved; /* set to 0 */
} pnotResItem; *pnotResItemPtr;
```

Field descriptions

modDate	Contains the modification time (in standard Macintosh seconds since midnight, January 1, 1904) of this item. This parameter allows you to find out if the item is out of date with the rest of the items in the array.
useType	Indicates the meaning of the data pointed to by this item. Two values are currently defined for this field.  KeyW            Indicates that this item points to a list of keywords, typically stored in an 'STR#' resource.  Desc           Indicates that the item points to a brief text description of the file, typically stored in a 'TEXT' resource.  Developers are encouraged to expand the list of types to include additional relevant kinds of information.
resType	Contains the type of a resource used as a preview cache for the file associated with the given item. The type of the resource determines which preview component should be used to display the preview.
resID	Contains the identification number of a resource used as a preview cache for the specified file.
rgnCode	Contains the region code for this item.
reserved	Reserved for use by Apple. Set this field to 0.

## Summary of Preview Components

---

### C Summary

---

#### Constants

---

```
enum {
    pnotComponentWantsEvents    = 1,  /* component requires events */
    pnotComponentNeedsNoCache  = 2    /* component does not require cache */
};

enum {
    kPreviewShowDataSelector      = 1,  /* PreviewShowData */
    kPreviewMakePreviewSelector    = 2,  /* PreviewMakePreview */
    kPreviewMakePreviewReferenceSelector= 3,
                                    /* PreviewMakePreviewReference */
    kPreviewEventSelector        = 4    /* PreviewEvent */
};

#define ShowFilePreviewComponentType 'pnot'    /* creates previews */
#define CreateFilePreviewComponentType 'pmak'  /* displays previews */
```

#### Data Types

---

```
typedef ComponentInstance pnotComponent;

typedef struct pnotResource {
    unsigned long  modDate;    /* modification date */
    short          version;    /* version number of preview resource */
    OSType         resType;    /* type of resource used as preview cache */
    short          resID;      /* resource identification number
                               of resource used as preview cache */
    short          numResItems; /* number of additional file descriptions */
    pnotResItem    resItem[ ]; /* array of file descriptions */
} pnotResource;
```

## Preview Components

```
typedef struct pnotResItem {
    unsigned long  modDate; /* last modification date of item */
    OSType         useType; /* what type of data */
    OSType         resType; /* resource type containing item */
    short          resID;   /* resource ID containing this item */
    short          rgnCode; /* region code */
    long           reserved; /* set to 0 */
} pnotResItem; *pnotResItemPtr;
```

## Functions

---

### Displaying Previews

```
pascal ComponentResult PreviewShowData
    (pnotComponent p, OSType dataType,
     Handle data, const Rect *inHere);
```

### Handling Events

```
pascal ComponentResult PreviewEvent
    (pnotComponent p, EventRecord *e,
     Boolean *handledEvent);
```

### Creating Previews

```
pascal ComponentResult PreviewMakePreview
    (pnotComponent p, OSType *previewType,
     Handle *previewResult,
     const FSSpec *sourceFile,
     ProgressProcRecordPtr progress);

pascal ComponentResult PreviewMakePreviewReference
    (pnotComponent p, OSType *previewType,
     short *resID, const FSSpec *sourceFile);
```

## Pascal Summary

---

### Constants

---

```
CONST
    {flags for component flags field for your preview component}
    pnotComponentWantsEvents = 1; {component requires events}
    pnotComponentNeedsNoCache = 2; {component does not require cache}
```

## Preview Components

```

{selectors for preview components}
kPreviewShowDataSelector          = 1;  {PreviewShowData}
kPreviewMakePreviewSelector       = 2;  {PreviewMakePreview}
kPreviewMakePreviewReferenceSelector = 3;  {PreviewMakePreviewReference}
kPreviewEventSelector            = 4;  {PreviewEvent}

{component types and subtypes}
ShowFilePreviewComponentType      'pnot'  {creates previews}
CreateFilePreviewComponentType    'pmak'  {displays previews}

```

## Data Types

---

### TYPE

```

pnotComponent = ComponentInstance; {preview component type}

pnotResource =
RECORD
    modDate:      LongInt;    {modification date}
    version:      Integer;    {version number of preview }
                                { resource}
    resType:      OSType;     {type of resource used as preview }
                                { cache}
    resID:        Integer;    {resource identification number }
                                { of resource used as preview }
                                { cache}
    numResItems:  Integer;    {number of additional file }
                                { descriptions}
    ARRAY OF resItem[ ]: pnotResItem;
                                {array of file descriptions}
END;

pnotResItem =
RECORD
    modDate:      LongInt;    {last modification date of item}
    useType:      OSType;     {what type of data}
    resType:      OSType;     {resource type containing item}
    resID:        Integer;    {resource ID containing this item}
    rgnCode:      Integer;    {region code}
    reserved:     LongInt;    {set to 0}
END;

```



## Routines

---

### Displaying Previews

```
FUNCTION PreviewShowData (p: pnotComponent; dataType: OSType;  
    data: Handle; VAR inHere: Rect):  
    ComponentResult;
```

### Handling Events

```
FUNCTION PreviewEvent (p: pnotComponent; VAR e: EventRecord;  
    VAR handledEvent: Boolean): ComponentResult;
```

### Creating Previews

```
FUNCTION PreviewMakePreview (p: pnotComponent; VAR previewType: OSType;  
    VAR previewResult: Handle;  
    VAR sourceFile: FSSpec;  
    progress: ProgressProcRecordPtr):  
    ComponentResult;  
  
FUNCTION PreviewMakePreviewReference (p: pnotComponent; VAR previewType: OSType;  
    VAR resID: Integer; VAR sourceFile: FSSpec):  
    ComponentResult;
```

