# Preview Components

Preview Components are used by the Standard File Preview calls to display and create previews for files.

Some preview components may 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 simpliest type of display components. They do not require any other event processing, including time. The picture preview component is an example of this. A preview component for sound would require events, since it would need time to play the sound. If a preview component requires events, it must have the “pnotComponentWantsEvents” flag set in its Component Flags field.

There are potentially three different sources of data for the preview component.
1. The preview component creates a small data cache containing the preview. Creating the preview cache may be a somewhat time consuming process, but after the preview cache is created, it can be stored in the file, and used to rapidly display the preview for the file on future occasions. The Picture File preview component works in this way. It creates a thumbnail picture for the file and stores, which is stored in the files resource fork.
2. 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 just create a pointer to that existing data to use, 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 just its poster picture.
3. The preview component can display the preview for the file quickly enough in all cases, that there is no need for a cache. These preview components reinterpret the data in the file each time they are invoked, rather than creating a preview cache once. The advantage is that the file remains untouched, no disk space is required, and neither the user or the application have to 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 which handle this type of preview should set the “pnotComponentNeedsNoCache” flag in their Component flags field.

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 EnterMovies and ExitMovies.

When previewing is complete, the component receives a normal Component Manager close call.

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 that create previews have a type of “pmak” and a subType that matches the type of the file that they create previews for.

Preview Components that display previews have a type of “pnot” and a subType that matches the type of the resource that they cache. The exception to this rule is preview components that do not require a cache (3 above), which should have a subType that matches the type of file they can display previews for.

## Data Structures

```
typedef ComponentInstance pnotComponent;

enum {
            pnotComponentWantsEvents = 1,
            pnotComponentNeedsNoCache = 2
};

typedef struct pnotResource {
            unsigned long     modDate;
            short             version;
            OSType            resType;
            short             resID;
} pnotResource;
```

The pnotResource format is currently defined as explained below. 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, beware.

The “modDate” is the modification time (in standard Macintosh seconds since whenever) of the file that the preview was created for. This can be used to tell if the preview is out of date with the contents of the file.

The “resType” and “resID” fields contain the type and id of a resource which is being used as a preview cache for the given file. The type of the resource will be used to determine which Preview Component should be used to display the preview.

The “version” field was always set to zero for QuickTime 1.0. In QuickTime 1.5, the low bit of the version has been claimed as a flag for preview components which only reference their data. If the bit is set, it indicates that the resource identified in the pnot resource is not owned by the preview component, but is part of the file. It will not be removed when the preview is updated or removed (MakeFilePreview, AddFilePreview), as it would be if the version number is zero.

**Warning**: Apple is currently considering several extensions to the pnotResource data structure. If your application updates exiting pnotResources directly you should take care not to destroy any data which comes after the currently defined fields.

## Routines

```
pascal ComponentResult PreviewShowData(pnotComponent p, OSType dataType, Handle data, cons
            Rect *inHere) = ComponentCallNow(1, 12);
```

For components which do not handle events, this routine is the sole display routine. In this routine, you are passed a typed handle, and a rectangle to draw the preview into. The current port is set to the correct GrafPort for drawing. You should not draw outside the given rectangle. The type of the handle will typically be of the same type as the subType of your preview component.

If your preview component has the “pnotComponentNeedsNoCache” flag set, this call should be considered an “initialize.” You should remember the rectangle, and set up any necessary data structures. An update event will be generated after this call for your initial drawing. In this case, the type of the handle will be “alis” (rAliasType) and the handle will contain an alias to the file to be previewed. If you add any controls to the window, you should dispose of them during your CloseComponent call.

```
pascal ComponentResult PreviewMakePreview(pnotComponent p, OSType *previewType, Handle
            *previewResult, const FSSpec *sourceFile, ProgressProcRecordPtr progress) =
            ComponentCallNow(2, 16);
```

This is the most common call for creating previews. It should create a handle of cached preview data (for example, a PICT) and return it along with the type of preview component that should be used to display that preview. This type should be the same as the type of data the handle contains. The “sourceFile” parameter provides a reference to the file to create the preview for. If the process of creating a preview will take more than a few seconds, you should call the progress procedure provided. The progress proc is of a standard Image Compression Manager type.

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

```
pascal ComponentResult PreviewMakePreviewReference(pnotComponent p, OSType *previewType,
            short *resID, const FSSpec *sourceFile) = ComponentCallNow(3, 12);
```

This call is very similar to PreviewMakePreview, except that instead of creating a handle of data to be added to the file, you just return the type and id of a resource within the file to be used as the preview for the file.

If your Preview Component creates previews by reference, you must also implement the PreviewMakePreview routine. However, you should return an error from it. PreviewMakePreview is always called first. If it fails, PreviewMakePreviewReference is tried next.

```
pascal ComponentResult PreviewEvent(pnotComponent p, EventRecord *e, Boolean *handledEvent
            = ComponentCallNow(4, 8);
```

If your preview component handles events, this routine will be called as appropriate. It is basically just a hook onto the standard dialog event filter routine. If you completely handle an event such as a mouse down or key stroke, you should set the “handledEvent” parameter to true. Otherwise, set it to false.