

The encoder delivers the encoded media in an MPEG-2 Transport Stream over the local network to the stream segmenter.

Note: MPEG-2 transport streams should not be confused with MPEG-2 video compression. The transport stream is a packaging format that can be used with a number of different compression formats. Only MPEG-2 transport streams with H.264 video and AAC audio are supported at this time for audio-video content. Audio-only content can be either MPEG-2 transport or MPEG elementary audio streams, either in AAC format with ADTS headers or in MP3 format.

Important: The video encoder should not change stream settings—such as video dimensions or codec type—in the midst of encoding a stream.

Stream Segmenter

The stream segmenter is a process—typically software—that reads the Transport Stream from the local network and divides it into a series of small media files of equal duration. Even though each segment is in a separate file, video files are made from a continuous stream which can be reconstructed seamlessly.

The segmenter also creates an index file containing references to the individual media files. Each time the segmenter completes a new media file, the index file is updated. The index is used to track the availability and location of the media files. The segmenter may also encrypt each media segment and create a key file as part of the process.

Media segments are saved as `.ts` files (MPEG-2 transport stream files). Index files are saved as `.M3U8` playlists.

File Segmenter

If you already have a media file encoded using supported codecs, you can use a file segmenter to encapsulate it in an MPEG stream and break it into segments of equal length. The file segmenter allows you to use a library of existing audio and video files for sending video on demand via HTTP Live Streaming. The file segmenter performs the same tasks as the stream segmenter, but it takes files as input instead of streams.

Media Segment Files

The media segment files are normally produced by the stream segmenter, based on input from the encoder, and consist of a series of `.ts` files containing segments of an MPEG-2 Transport Stream carrying H.264 video and AAC audio. For an audio-only broadcast, the segmenter can produce MPEG elementary audio streams containing either AAC audio with ADTS headers or MP3 audio.

Index Files (Playlists)

Index files are normally produced by the stream segmenter or file segmenter, and saved as `.M3U8` playlists, an extension of the `.m3u` format used for MP3 playlists.

Note: Because the index file format is an extension of the .m3u playlist format, and because the system also supports .mp3 audio media files, the client software may also be compatible with typical MP3 playlists used for streaming Internet radio.

Here is a very simple example of an index file, in the form of an .M3U8 playlist, that a segmenter might produce if the entire stream were contained in three unencrypted 10-second media files:

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:10,
http://media.example.com/segment0.ts
#EXTINF:10,
http://media.example.com/segment1.ts
#EXTINF:10,
http://media.example.com/segment2.ts
#EXT-X-ENDLIST
```

Note: You can use the file segmenter provided by Apple to generate a variety of example playlists, using an MPEG-4 video or AAC or MP3 audio file as a source. For details, see “Media File Segmenter” (page 17).

The index file may also contain URLs for encryption key files and alternate index files for different bandwidths. For details of the index file format, see the IETF Internet-Draft of the HTTP Live Streaming specification.

Index files are normally created by the same segmenter that creates the media segment files. Alternatively, it is possible to create the .M3U8 file and the media segment files independently, provided they conform the published specification. For audio-only broadcasts, for example, you could create an .M3U8 file using a text editor, listing a series of existing .MP3 files.

Distribution Components

The distribution system is a web server or a web caching system that delivers the media files and index files to the client over HTTP. No custom server modules are required to deliver the content, and typically very little configuration is needed on the web server.

Recommended configuration is typically limited to specifying MIME-type associations for .M3U8 files and .ts files.

File extension	MIME type
.M3U8	application/x-mpegURL or vnd.apple.mpegURL
.ts	video/MP2T

Tuning time-to-live (TTL) values for .M3U8 files may also be necessary to achieve desired caching behavior for downstream web caches, as these files are frequently overwritten during live broadcasts, and the latest version should be downloaded for each request.

Client Component

The client software begins by fetching the index file, based on a URL identifying the stream. The index file in turn specifies the location of the available media files, decryption keys, and any alternate streams available. For the selected stream, the client downloads each available media file in sequence. Each file contains a consecutive segment of the stream. Once it has a sufficient amount of data downloaded, the client begins presenting the reassembled stream to the user.

The client is responsible for fetching any decryption keys, authenticating or presenting a user interface to allow authentication, and decrypting media files as needed.

This process continues until the client encounters the `#EXT-X-ENDLIST` tag in the index file. If no `#EXT-X-ENDLIST` tag is present, the index file is part of an ongoing broadcast. During ongoing broadcasts, the client loads a new version of the index file periodically. The client looks for new media files and encryption keys in the updated index and adds these URLs to its queue.

Using HTTP Live Streaming

Download the Tools

There are several tools available that can help you set up an HTTP Live Streaming service. The tools include a media stream segmenter, a media file segmenter, a stream validator, an id3 tag generator, and a variant playlist generator.

The tools are frequently updated, so you should download the current version of the HTTP Live Streaming Tools from the Apple Developer website. You can access them if you are a member of the iPhone Developer Program. One way to navigate to the tools is to log onto connect.apple.com, then click iPhone under the Downloads heading.

Media Stream Segmenter

The `mediastreamsegmenter` command-line tool takes an MPEG-2 transport stream as an input and produces a series of equal-length files from it, suitable for use in HTTP Live Streaming. It can also generate index files (also known as playlists), encrypt the media, produce encryption keys, optimize the files by reducing overhead, and create the necessary files for automatically generating multiple stream alternates. For details, type `man mediastreamsegmenter` from the terminal window.

Usage example: `mediastreamsegmenter -s 3 -D -f /Library/WebServer/Documents/stream 239.4.1.5:20103`

The usage example captures a live stream from the network at address `239.4.1.5:20103` and creates media segment files and index files from it. The index files contain a list of the current three media segment files (`-s 3`). The media segment files are deleted after use (`-D`). The index files and media segment files are stored in the directory `/Library/WebServer/Documents/stream`.

Media File Segmenter

The `mediafilesegmenter` command-line tool takes an encoded media file as an input, wraps it in an MPEG-2 transport stream, and produces a series of equal-length files from it, suitable for use in HTTP Live Streaming. The media file segmenter can also produce index files (playlists) and decryption keys. The file segmenter behaves very much like the stream segmenter, but it works on existing files instead of streams coming from an encoder. For details, type `man mediafilesegmenter` from the terminal window.

Media Stream Validator

The `mediastreamvalidator` command-line tool examines the index files, stream alternates, and media segment files on a server and tests to determine whether they will work with HTTP Live Streaming clients. For details, type `man mediastreamvalidator` from the terminal window.

Variant Playlist Creator

The `variantplaylistcreator` command-line tool creates a master index file, or playlist, listing the index files for alternate streams at different bit rates, using the output of the stream or file segmenter. The segmenter must be invoked with the `-generate-variant-playlist` argument to produce the required output for the variant playlist creator. For details, type `man variantplaylistcreator` from the terminal window.

Metadata Tag Generator

The `id3taggenerator` command-line tool generates ID3 metadata tags. These tags can either be written to a file or inserted into outgoing stream segments. For details, see “Adding Metadata” (page 23).

Session Types

The HTTP Live Streaming protocol supports live broadcast sessions and video on demand (VOD) sessions. Live sessions can be presented as a complete record of an event, or as a sliding window with a limited time range the user can seek within.

For live sessions, as new media files are created and made available, the index file is updated. The new index file lists the new media files. Older media files can be removed from the index and discarded, presenting a moving window into a continuous stream—this type of session is suitable for continuous broadcasts. Alternatively, the index can simply add new media files to the existing list—this type of session can be easily converted to VOD after the event completes.

For VOD sessions, media files are available representing the entire duration of the presentation. The index file is static and contains a complete list of all files created since the beginning of the presentation. This kind of session allows the client full access to the entire program.

It is possible to create a live broadcast of an event that is instantly available for video on demand. To convert a live broadcast to VOD, do not remove the old media files from the server or delete their URLs from the index file; instead, add an `#EXT-X-ENDLIST` tag to the index when the event ends. This allows clients to join the broadcast late and still see the entire event. It also allows an event to be archived for rebroadcast with no additional time or effort.

VOD can also be used to deliver “canned” media. HTTP Live Streaming offers advantages over progressive download for VOD, such as support for media encryption and dynamic switching between streams of different data rates in response to changing connection speeds. (QuickTime also supports multiple-data-rate movies using progressive download, but QuickTime movies do not support dynamically switching between data rates in mid-movie.)

Content Protection

Media files containing stream segments may be individually encrypted. When encryption is employed, references to the corresponding key files appear in the index file so that the client can retrieve the keys for decryption.

When a key file is listed in the index file, the key file contains a cipher key that must be used to decrypt subsequent media files listed in the index file. Currently HTTP Live Streaming supports AES-128 encryption using 16-octet keys. The format of the key file is a packed array of these 16 octets in binary format.

The media stream segmenter available from Apple provides encryption and supports three modes for configuring encryption.

The first mode allows you to specify a path to an existing key file on disk. In this mode the segmenter inserts the URL of the existing key file in the index file. It encrypts all media files using this key.

The second mode instructs the segmenter to generate a random key file, save it in a specified location, and reference it in the index file. All media files are encrypted using this randomly generated key.

The third mode instructs the segmenter to generate a new random key file every n media segments, save it in a specified location, and reference it in the index file. This mode is referred to as key rotation. Each group of n files is encrypted using a different key.

Note: All media files may be encrypted using the same key, or new keys may be required at intervals. The theoretical limit is one key per media file, but because each media key adds a file request and transfer to the overhead for presenting the subsequent media segments, changing to a new key periodically is less likely to impact system performance than changing keys for each segment.

You can serve key files using either HTTP or HTTPS. You may also choose to protect the delivery of the key files using your own session-based authentication scheme. For details, see “Serving Key Files Securely Over HTTPS” (page 29).

Key files require an initialization vector (IV) to decode encrypted media. The IVs can be changed periodically, just as the keys can.

Caching and Delivery Protocols

HTTPS is commonly used to deliver key files. It may also be used to deliver the media segment files and index files, but this is not recommended when scalability is important, since HTTPS requests often bypass web server caches, causing all content requests to be routed through your server and defeating the purpose of edge network distribution systems.

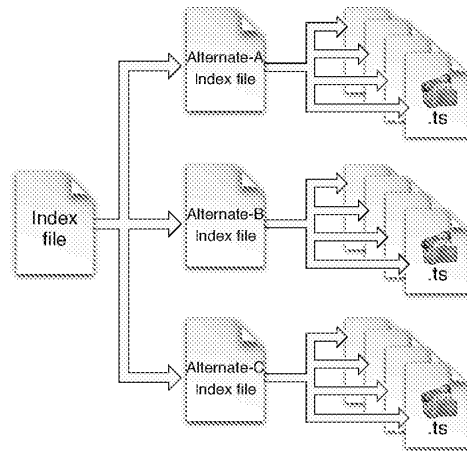
For this very reason, however, it is important to make sure that any content delivery network you use understands that the `.M3U8` index files are not to be cached for longer than one media segment duration for live broadcasts, where the index file is changing dynamically.

Stream Alternates

A master index file may reference alternate streams of content. References can be used to support delivery of multiple streams of the same content with varying quality levels for different bandwidths or devices. HTTP Live Streaming supports switching between streams dynamically if the available bandwidth changes. The client software uses heuristics to determine appropriate times to switch between the alternates. Currently, these heuristics are based on recent trends in measured network throughput.

The master index file points to alternate streams of media by including a specially tagged list of other index files, as illustrated in Figure 2-1

Figure 2-1 Stream alternates



Both the master index file and the alternate index files are in .M3U8 playlist format. The master index file is downloaded only once, but for live broadcasts the alternate index files are reloaded periodically. The first alternate listed in the master index file is the first stream used—after that, the client chooses among the alternates by available bandwidth.

Note that the client may choose to change to an alternate stream at any time, such as when a mobile device enters or leaves a WiFi hotspot. All alternates should use identical audio to allow smooth transitions among streams.

You can create a set of stream alternates by using the `variantplaylistcreator` tool and specifying the `-generate-variant-playlist` option for either the `mediasegmenter` tool or the `mediastreamsegmenter` tool (see “Download the Tools” (page 17) for details).

When using stream alternates, it is important to bear the following considerations in mind:

- The first entry in the variant playlist is played when a user joins the stream and is used as part of a test to determine which stream is most appropriate. The order of the other entries is irrelevant.
- Where possible, encode enough variants to provide the best quality stream across a wide range of connection speeds. For example, encode variants at 150 kbps, 350 kbps, 550 kbps, 900 kbps, 1500 kbps.
- When possible, use relative path names in variant playlists and in the individual .M3U8 playlist files
- The audio track in all streams should be exactly the same, to prevent audio artifacts when the stream changes.

Note: One way to synchronize different VOD streams is to copy the audio track from one media file and paste it into each of the other media files, after compression but before segmenting.

- The video aspect ratio on alternate streams *must* be exactly the same, but alternates can have different pixel dimensions, as long as they have the same aspect ratio. For example, two stream alternates with the same 4:3 aspect ratio could have dimensions of 400 x 300 and 800 x 600.
- If you are an iOS app developer, you can query the user's device to determine whether the initial connection is cellular or WiFi and choose an appropriate master index file.

To ensure the user has a good experience when the stream is first played, regardless of the initial network connection, you should have more than one master index file consisting of the same alternate index files but with a different first stream.

A 150k stream for the cellular variant playlist is recommended.

A 240k or 440k stream for the Wi-Fi variant playlist is recommended.

Note: For details on how to query an iOS-based device for its network connection type, see the following sample code: *Reachability*.

- When you specify the bitrates for stream variants, it is important that the `BANDWIDTH` attribute closely match the actual bandwidth required by a given stream. If the actual bandwidth requirement is substantially different than the `BANDWIDTH` attribute, automatic switching of streams may not operate smoothly or even correctly.

Video Over Cellular Networks

When you send video to a mobile device such as iPhone or iPad, the client's Internet connection may move to or from a cellular network at any time.

HTTP Live Streaming allows the client to choose among stream alternates dynamically as the network bandwidth changes, providing the best stream as the device moves between cellular and WiFi connections, for example, or between 3G and EDGE connections. This is a significant advantage over progressive download.

It is strongly recommended that you use HTTP Live Streaming to deliver video to all cellular-capable devices, even for video on demand, so that your viewers have the best experience possible under changing conditions.

In addition, you should provide cellular-capable clients an alternate stream at 64 Kbps or less for slower data connections. If you cannot provide video of acceptable quality at 64 Kbps or lower, you should provide an audio-only stream, or audio with a still image.

A good choice for pixel dimensions when targeting cellular network connections is 400 x 224 for 16:9 content and 400 x 300 for 4:3 content (see "Preparing Media for Delivery to iOS-Based Devices" (page 24)).

Requirements for Apps



Warning: iOS apps submitted for distribution in the App Store must conform to these requirements.

If your app delivers video over cellular networks, and the video exceeds either 10 minutes duration or 5 MB of data in a five minute period, you are required to use HTTP Live Streaming. (Progressive download may be used for smaller clips.)

If your app uses HTTP Live Streaming over cellular networks, you are required to provide at least one stream at 64 Kbps or lower bandwidth (the low-bandwidth stream may be audio-only or audio with a still image).

These requirements apply to iOS apps submitted for distribution in the App Store for use on Apple products. Non-compliant apps may be rejected or removed, at the discretion of Apple.

Failover Protection

If your playlist contains alternate streams, they can not only operate as bandwidth or device alternates, but as failure fallbacks. Starting with iOS 3.1, if the client is unable to reload the index file for a stream (due to a 404 error, for example), the client attempts to switch to an alternate stream.

In the event of an index load failure on one stream, the client chooses the highest bandwidth alternate stream that the network connection supports. If there are multiple alternates at the same bandwidth, the client chooses among them in the order listed in the playlist.

You can use this feature to provide redundant streams that will allow media to reach clients even in the event of severe local failures, such as a server crashing or a content distributor node going down.

To implement failover protection, create a stream—or multiple alternate bandwidth streams—and generate a playlist file as you normally would. Then create a parallel stream, or set of streams, on a separate server or content distribution service. Add the list of backup streams to the playlist file, so that the backup stream at each bandwidth is listed after the primary stream. For example, if the primary stream comes from server ALPHA, and the backup stream is on server BETA, your playlist file might look something like this:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://ALPHA.mycompany.com/1o/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://BETA.mycompany.com/1o/prog_index.m3u8

#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://ALPHA.mycompany.com/md/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://BETA.mycompany.com/md/prog_index.m3u8
```

Note that the backup streams are intermixed with the primary streams in the playlist, with the backup at each bandwidth listed after the primary for that bandwidth.

You are not limited to a single backup stream set. In the example above, ALPHA and BETA could be followed by GAMMA, for instance. Similarly, you need not provide a complete parallel set of streams. You could provide a single low-bandwidth stream on a backup server, for example.

Adding Timed Metadata

You can add various kinds of metadata to media stream segments. For example, you can add the album art, artist's name, and song title to an audio stream. As another example, you could add the current batter's name and statistics to video of a baseball game.

If an audio-only stream includes an image as metadata, the Apple client software automatically displays it. Currently, the only metadata that is automatically displayed by the Apple-supplied client software is a still image accompanying an audio-only stream.

If you are writing your own client software, however, using either `MPMoviePlayerController` or `AVPlayerItem`, you can access streamed metadata using the `timedMetadata` property.

You can add timed metadata by specifying a metadata file in the `-F` command line option to either the stream segmenter or the file segmenter. The specified metadata source can be a file in ID3 format or an image file (JPEG or PNG). Metadata specified this way is automatically inserted into every media segment.

This is called timed metadata because it is inserted into a media stream at a given time offset. Timed metadata can optionally be inserted into all segments after a given time.

To add timed metadata to a live stream, use the `id3taggenerator` tool, with its output set to the stream segmenter. The tool generates ID3 metadata and passes it the stream segmenter for inclusion in the outbound stream.

The tag generator can be run from a shell script, for example, to insert metadata at the desired time, or at desired intervals. New timed metadata automatically replaces any existing metadata.

Once metadata has been inserted into a media segment, it is persistent. If a live broadcast is re-purposed as video on demand, for example, it retains any metadata inserted during the original broadcast.

Adding timed metadata to a stream created using the file segmenter is slightly more complicated.

1. First, generate the metadata samples. You can generate ID3 metadata using the `id3taggenerator` command-line tool, with the output set to file.
2. Next, create a **metadata macro file**—a text file in which each line contains the time to insert the metadata, the type of metadata, and the path and filename of a metadata file.

For example, the following metadata macro file would insert a picture at 1.2 seconds into the stream, then an ID3 tag at 10 seconds:

```
1.2 picture /meta/images/picture.jpg
10 id3 /meta/id3/title.id3
```

3. Finally, specify the metadata macro file by name when you invoke the media file segmenter, using the `-M` command line option.

For additional details, see the man pages for `mediastreamsegmenter`, `mediafilesegmenter`, and `id3taggenerator`.

Adding Closed Captions

HTTP Live Streaming supports adding closed captions to streams.

Note: Closed captions should not be confused with subtitles.

If you are using the stream segmenter, you need to add CEA-608 closed captions to the MPEG-2 transport stream (in the main video elementary stream) as specified in ATSC A/72.

If you are using the file segmenter, you should encapsulate your media in a QuickTime movie file and add a closed caption track ('c1cp').

One tool you can use to add closed captions to movie files or MPEG-2 transport streams is Compressor, included with Final Cut Studio. Compressor works in conjunction with Scenarist software to generate closed captions. See the Compressor documentation for details.

If you are writing an app, the AVFoundation framework supports playback of closed captions.

Preparing Media for Delivery to iOS-Based Devices

The recommended encoder settings for streams used with iOS-based devices are shown in the following four tables. For live streams, these settings should be available from your hardware or software encoder. If you are re-encoding from a master file for video on demand, you can use a video editing tool such as Compressor.

File format for the file segmenter can be a QuickTime movie, MPEG-4 video, or MP3 audio, using the specified encoding.

Stream format for the stream segmenter must be MPEG elementary audio and video streams, wrapped in an MPEG-2 transport stream, and using the following encoding.

- Encode video using H.264 compression
 - iPhone 3G and later, and iPod touch, 2nd generation and later, support H.264 Baseline 3.1. If your app runs on older versions of iPhone or iPod touch, however, you should use H.264 Baseline 3.0 for compatibility.
 - For iPad, Apple TV 2 and later, and iPhone 4 and later, you can use Baseline profile 3.0, Baseline profile 3.1, or Main profile 3.1.

Note: Baseline profiles are not the same as Main profiles. iPhone and iPod touch use Baseline profiles. iPad and Apple TV can use Baseline or Main profiles. Use Main profiles if your content is for large-screen devices only; use Baseline profiles if your content is meant to be viewed on small screens only, or on both large and small screens.

- A frame rate of 10 fps is recommended for video streams under 200 kbps. For video streams under 300 kbps, a frame rate of 12 to 15 fps is recommended. For all other streams, a frame rate of 29.97 is recommended.
- Encode audio as either of the following:
 - HE-AAC or AAC-LC, stereo
 - MP3 (MPEG-1 Audio Layer 3), stereo
- Audio sample rate of 22.05 kHz and audio bit rate of 40 kbps is recommended in all cases.

Table 2-1 Encoder settings for iPhone, iPod touch, iPad, and Apple TV, 16:9 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
Cellular	480 x 320	64 kbps	audio only	none
Cellular	480 x 224	150 kbps	110 kbps	30
Cellular	480 x 224	240 kbps	200 kbps	45
Cellular	480 x 224	440 kbps	400 kbps	90
WiFi	640 x 360	640 kbps	600 kbps	90

Table 2-2 Encoder settings for iPhone, iPod touch, iPad, and Apple TV, 4:3 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
Cellular	480 x 320	64 kbps	audio only	none
Cellular	480 x 300	150 kbps	110 kbps	30
Cellular	480 x 300	240 kbps	200 kbps	45
Cellular	480 x 300	440 kbps	400 kbps	90
WiFi	640 x 480	640 kbps	600 kbps	90

Table 2-3 Additional encoder settings for iPad and Apple TV only, 16:9 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	640 x 360	1240 kbps	1200 kbps	90
WiFi	960 x 540	1840 kbps	1800 kbps	90

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	1280 x 720	2540 kpbs	1500 kpbs	90
WiFi	1280 x 720	4540 kpbs	4500 kpbs	90

Table 2-4 Additional encoder settings for iPad and Apple TV only, 4:3 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	640 x 480	1240 kpbs	1200 kpbs	90
WiFi	960 x 720	1840 kpbs	1800 kpbs	90
WiFi	960 x 720	2540 kpbs	2500 kpbs	90
WiFi	1280 x 960	4540 kpbs	4500 kpbs	90

Sample Streams

There are a series of HTTP streams available for testing on Apple's developer site. These examples show proper formatting of HTML to embed streams, .M3U8 files to index the streams, and .ts media segment files. The streams can be accessed at the following URLs:

- ◆ <http://devimages.apple.com/iphone/samples/bipbopgear1.html>
- ◆ <http://devimages.apple.com/iphone/samples/bipbopgear2.html>
- ◆ <http://devimages.apple.com/iphone/samples/bipbopgear3.html>
- ◆ <http://devimages.apple.com/iphone/samples/bipbopgear4.html>
- ◆ <http://devimages.apple.com/iphone/samples/bipbopall.html>

The samples show the same NTSC test pattern at four different resolutions and data rates. The last sample streams at multiple data rates. The stream starts with sample 1 and switches to the fastest sample the connection supports.

You must install iOS version 3.0 or later to play these samples on your iPhone or iPod touch. Safari 4.0 or later is required for playback on the desktop.

Deploying HTTP Live Streaming

To actually deploy HTTP Live Streaming, you need to create either an HTML page for browsers or a client app to act as a receiver. You also need the use of a web server and a way to either encode live streams as MPEG-2 transport streams or to create MP3 or MPEG-4 media files with H.264 and AAC encoding from your source material.

You can use the Apple-provided tools to segment the streams or media files, and to produce the index files and variant playlists (see “Download the Tools” (page 17)).

You should use the Apple-provided media stream validator prior to serving your streams, to ensure that they are fully compliant with HTTP Live Streaming.

You may want to encrypt your streams, in which case you probably also want to serve the encryption key files securely over HTTPS, so that only your intended clients can decrypt them.

Creating an HTML Page

The easiest way to distribute HTTP Live Streaming media is to create a webpage that includes the HTML5 `<video>` tag, using an .M3U8 playlist file as the video source. An example is shown in Listing 3-1.

Listing 3-1 Serving HTTP Live Streaming in a webpage

```
<html>
<head>
  <title>HTTP Live Streaming Example</title>
</head>
<body>
  <video
    src="http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8"
    height="300" width="400"
  >
</video>
</body>
</html>
```

For browsers that don't support the HTML5 `video` element, or browsers that don't support HTTP Live Streaming, you can include fallback code between the `<video>` and `</video>` tags. For example, you could fall back to a progressive download movie or an RTSP stream using the QuickTime plug-in. See *Safari HTML5 Audio and Video Guide* for examples.

Configuring a Web Server

HTTP Live Streaming can be served from an ordinary web server; no special configuration is necessary, apart from associating the MIME types of the files being served with their file extensions.

Configure the following MIME types for HTTP Live Streaming:

File Extension	MIME Type
.M3U8	application/x-mpegURL or vnd.apple.mpegURL
.ts	video/MP2T

If your web server is constrained with respect to MIME types, you can serve files ending in .m3u with MIME type audio/mpegURL for compatibility.

Index files can be long and may be frequently redownloaded, but they are text files and can be compressed very efficiently. You can reduce server overhead by enabling on-the-fly .gzip compression of .M3U8 index files; the HTTP Live Streaming client automatically unzips compressed index files.

Shortening time-to-live (TTL) values for .M3U8 files may also be needed to achieve proper caching behavior for downstream web caches, as these files are frequently overwritten during live broadcasts, and the latest version should be downloaded for each request. Check with your content delivery service provider for specific recommendations. For VOD, the index file is static and downloaded only once, so caching is not a factor.

Validating Your Streams

The `mediastreamvalidator` tool is a command-line utility for validating HTTP Live Streaming streams and servers (see “Download the Tools” (page 17) for details on obtaining the tool).

The media stream validator simulates an HTTP Live Streaming session and verifies that the index file and media segments conform to the HTTP Live Streaming specification. It performs several checks to ensure reliable streaming. If any errors or problems are found, a detailed diagnostic report is displayed.

You should always run the validator prior to serving a new stream or alternate stream set.

The media stream validator shows a listing of the streams you provide, followed by the timing results for each of those streams. (It may take a few minutes to calculate the actual timing.) An example of validator output follows.

```
Validating http://devimages.apple.com/iphone/samples/bipbop/gear3/prog_index.m3u8
against iPhone OS 3.1.0
```

```
Average segment duration: 8.77 seconds
Average segment bitrate: 510.05 kbit/s
Average segment structural overhead: 96.37 kbit/s (18.89 %)
```

```
Video codec: avc1
Video resolution: 480x360 pixels
```

```
Video frame rate: 29.97 fps
Average video bitrate: 407.76 kbit/s
H.264 profile: Baseline
H.264 level: 2.1
```

```
Audio codec: aac
Audio sample rate: 22050 Hz
Average audio bitrate: 5.93 kbit/s
```

Serving Key Files Securely Over HTTPS

You can protect your media by encrypting it. The file segmenter and stream segmenter both have encryption options, and you can tell them to change the encryption key periodically. Who you share the keys with is up to you.

Key files require an initialization vector (IV) to decode encrypted media. The IVs can be changed periodically, just as the keys can. Current recommendations for encrypting media while minimizing overhead is to change the key every 3-4 hours and change the IV after every 50 Mb of data.

Even with restricted access to keys, however, it is possible for an eavesdropper to obtain copies of the key files if they are sent over HTTP. One solution to this problem is to send the keys securely over HTTPS.

Before you attempt to serve key files over HTTPS, you should do a test serving the keys from an internal web server over HTTP. This allows you to debug your setup before adding HTTPS to the mix. Once you have a known working system, you are ready to make the switch to HTTPS.

There are three conditions you must meet in order to use HTTPS to serve keys for HTTP Live Streaming:

- You need to install an SSL certificate signed by a trusted authority on your HTTPS server.
- The authentication domain for the key files must be the same as the authentication domain for the first playlist file. The simplest way to accomplish this is to serve the variant playlist file from the HTTPS server—the variant playlist file is downloaded only once, so this shouldn't cause an excessive burden. Other playlist files can be served using HTTP.
- You must either initiate your own dialog for the user to authenticate, or you must store the credentials on the client device—HTTP Live Streaming does not provide user dialogs for authentication. If you are writing your own client app, you can store credentials, whether cookie-based or HTTP digest based, and supply the credentials in the `didReceiveAuthenticationChallenge` callback (see “Using `NSURLConnection`” and “Authentication Challenges” for details). The credentials you supply are cached and reused by the media player.

Important: You must obtain an SSL certificate signed by a trusted authority in order to use an HTTPS server with HTTP Live Streaming.

If your HTTPS server does not have an SSL certificate signed by a trusted authority, you can still test your setup by creating a self-signed SSL Certificate Authority and a leaf certificate for your server. Attach the certificate for the certificate authority to an email, send it to a device you want to use as a Live Streaming client, and tap on the attachment in Mail to make the device trust the server.

Frequently Asked Questions

1. What kinds of encoders are supported?

The protocol specification does not limit the encoder selection. However, the current Apple implementation should interoperate with encoders that produce MPEG-2 Transport Streams containing H.264 video and AAC audio (HE-AAC or AAC-LC). Encoders that are capable of broadcasting the output stream over UDP should also be compatible with the current implementation of the Apple provided segmenter software.

2. What are the specifics of the video and audio formats supported?

Although the protocol specification does not limit the video and audio formats, the current Apple implementation supports the following formats:

- ◆ Video: H.264 Baseline Level 3.0, Baseline Level 3.1, and Main Level 3.1.
- ◆ Audio:
 - ◆ HE-AAC or AAC-LC up to 48 kHz, stereo audio
 - ◆ MP3 (MPEG-1 Audio Layer 3) 8 kHz to 48 kHz, stereo audio

Note: iPad, iPhone 3G, and iPod touch (2nd generation and later) support H.264 Baseline 3.1. If your app runs on older versions of iPhone or iPod touch, however, you should use H.264 Baseline 3.0 for compatibility. If your content is intended solely for iPad, Apple TV, iPhone 4 and later, and Mac OS X computers, you should use Main Level 3.1.

3. What duration should media files be?

The main point to consider is that shorter segments result in more frequent refreshes of the index file, which might create unnecessary network overhead for the client. Longer segments will extend the inherent latency of the broadcast and initial startup time. A duration of 10 seconds of media per file seems to strike a reasonable balance for most broadcast content.

4. How many files should be listed in the index file during a continuous, ongoing session?

The normal recommendation is 3, but the optimum number may be larger.

The important point to consider when choosing the optimum number is that the number of files available during a live session constrains the client's behavior when doing play/pause and seeking operations. The more files in the list, the longer the client can be paused without losing its place in the broadcast, the further back in the broadcast a new client begins when joining the stream, and the wider the time range within which the client can seek. The trade-off is that a longer index file adds to network overhead—during live broadcasts, the clients are all refreshing the index file regularly, so it does add up, even though the index file is typically small.

5. What data rates are supported?

The data rate that a content provider chooses for a stream is most influenced by the target client platform and the expected network topology. The streaming protocol itself places no limitations on the data rates that can be used. The current implementation has been tested using audio-video streams with data rates as low as 64 Kbps and as high as 3 Mbps to iPhone. Audio-only streams at 64 Kbps are recommended as alternates for delivery over slow cellular connections.

For recommended data rates, see “Preparing Media for Delivery to iOS-Based Devices” (page 24).

Note: If the data rate exceeds the available bandwidth, there is more latency before startup and the client may have to pause to buffer more data periodically. If a broadcast uses an index file that provides a moving window into the content, the client will eventually fall behind in such cases, causing one or more segments to be dropped. In the case of VOD, no segments are lost, but inadequate bandwidth does cause slower startup and periodic stalling while data buffers.

6. What is a .ts file?

A .ts file contains an MPEG-2 Transport Stream. This is a file format that encapsulates a series of encoded media samples—typically audio and video. The file format supports a variety of compression formats, including MP3 audio, AAC audio, H.264 video, and so on. Not all compression formats are currently supported in the Apple HTTP Live Streaming implementation, however. (For a list of currently supported formats, see “Media Encoder” (page 12).

MPEG-2 Transport Streams are containers, and should not be confused with MPEG-2 compression.

7. What is an .M3U8 file?

An .M3U8 file is an extensible playlist file format. It is an m3u playlist containing UTF-8 encoded text. The m3u file format is a de facto standard playlist format suitable for carrying lists of media file URLs. This is the format used as the index file for HTTP Live Streaming. For details, see IETF Internet-Draft of the HTTP Live Streaming specification.

8. How does the client software determine when to switch streams?

The current implementation of the client observes the effective bandwidth while playing a stream. If a higher-quality stream is available and the bandwidth appears sufficient to support it, the client switches to a higher quality. If a lower-quality stream is available and the current bandwidth appears insufficient to support the current stream, the client switches to a lower quality.

Note: For seamless transitions between alternate streams, the audio portion of the stream should be identical in all versions.

9. Where can I find a copy of the media stream segmenter from Apple?

The media stream segmenter, file stream segmenter, and other tools are frequently updated, so you should download the current version of the HTTP Live Streaming Tools from the Apple Developer website. See “Download the Tools” (page 17) for details.

10. What settings are recommended for a typical HTTP stream, with alternates, for use with the media segmenter from Apple?

See “Preparing Media for Delivery to iOS-Based Devices” (page 24).

These settings are the current recommendations. There are also certain requirements. The current `mediastreamsegmenter` tool works only with MPEG-2 Transport Streams as defined in ISO/IEC 13818. The transport stream must contain H.264 (MPEG-4, part 10) video and AAC or MPEG audio. If AAC audio is used, it must have ADTS headers. H.264 video access units must use Access Unit Delimiter NALs, and must be in unique PES packets.

The segmenter also has a number of user-configurable settings. You can obtain a list of the command line arguments and their meanings by typing `man mediastreamsegmenter` from the Terminal application. A target duration (length of the media segments) of 10 seconds is recommended, and is the default if no target duration is specified.

11. How can I specify what codecs or H.264 profile are required to play back my stream?

Use the `CODECS` attribute of the `EXT-X-STREAM-INF` tag. When this attribute is present, it must include all codecs and profiles required to play back the stream. The following values are currently recognized:

AAC-LC	"mp4a.40.2"
HE-AAC	"mp4a.40.5"
MP3	"mp4a.40.34"
H.264 Baseline Profile level 3.0	"avc1.42001e" or "avc1.66.30" Note: Use "avc1.66.30" for compatibility with iOS versions 3.0 to 3.12.
H.264 Baseline Profile level 3.1	"avc1.42001f"
H.264 Main Profile level 3.0	"avc1.4d001e" or "avc1.77.30" Note: Use "avc1.77.30" for compatibility with iOS versions 3.0 to 3.12.
H.264 Main Profile level 3.1	"avc1.4d001f"

The attribute value must be in quotes. If multiple values are specified, one set of quotes is used to contain all values, and the values are separated by commas. An example follows.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
mid_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=800000
wifi_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=3000000, CODECS="avc1.4d001e,mp4a.40.5"
h264main_heaac_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=64000, CODECS="mp4a.40.5"
aacaudio_index.M3U8
```

12. How can I create an audio-only stream from audio/video input?

Add the `-audio-only` argument when invoking the stream or files segmenter.

13. How can I add a still image to an audio-only stream?

Use the `-meta-file` argument when invoking the stream or file segmenter with `-meta-type=picture` to add an image to every segment. For example, this would add an image named `poster.jpg` to every segment of an audio stream created from the file `track01.mp3`:

```
mediafilesegmenter -f /Dir/outputFile -a --meta-file=poster.jpg --meta-type=picture
track01.mp3
```

Remember that the image is typically resent every ten seconds, so it's best to keep the file size small.

14. How can I specify an audio-only alternate to an audio-video stream?

Use the CODECS and BANDWIDTH attributes of the EXT-X-STREAM-INF tag together.

The BANDWIDTH attribute specifies the bandwidth required for each alternate stream. If the available bandwidth is enough for the audio alternate, but not enough for the lowest video alternate, the client switches to the audio stream.

If the CODECS attribute is included, it must list all codecs required to play the stream. If only an audio codec is specified, the stream is identified as audio-only. Currently, it is not required to specify that a stream is audio-only, so use of the CODECS attribute is optional.

The following is an example that specifies video streams at 500 Kbps for fast connections, 150 Kbps for slower connections, and an audio-only stream at 64 Kbps for very slow connections. All the streams should use the same 64 Kbps audio to allow transitions between streams without an audible disturbance.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
mid_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=150000
3g_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=64000, CODECS="mp4a.40.5"
aacaudio_index.M3U8
```

15. What are the hardware requirements or recommendations for servers?

See question #1 for encoder hardware recommendations.

The Apple stream segmenter is capable of running on any Intel-based Mac. We recommend using a Mac with two Ethernet network interfaces, such as a Mac Pro or an XServe. One network interface can be used to obtain the encoded stream from the local network, while the second network interface can provide access to a wider network.

16. Does the Apple implementation of HTTP Live Streaming support DRM?

No. However, media can be encrypted and key access can be limited using HTTPS authentication.

17. What client platforms are supported?

iPhone, iPad, and iPod touch (requires iOS version 3.0 or later), Apple TV (version 2 and later), and Mac OS X computers.

18. Is the protocol specification available?

Yes. The protocol specification is an IETF Internet-Draft, at <http://tools.ietf.org/html/draft-pantos-http-live-streaming>.

19. Does the client cache content?

The index file can contain an instruction to the client that content should not be cached. Otherwise, the client may cache data for performance optimization when seeking within the media.

20. Is this a real-time delivery system?

No. It has inherent latency corresponding to the size and duration of the media files containing stream segments. At least one segment must fully download before it can be viewed by the client, and two may be required to ensure seamless transitions between segments. In addition, the encoder and segmenter must create a file from the input; the duration of this file is the minimum latency before media is available for download. Typical latency with recommended settings is in the neighborhood of 30 seconds.

21. What is the latency?

Approximately 30 seconds, with recommended settings. See question #15.

22. Do I need to use a hardware encoder?

No. Using the protocol specification, it is possible to implement a software encoder.

23. What advantages does this approach have over RTP/RTSP?

HTTP is less likely to be disallowed by routers, NAT, or firewall settings. No ports need to be opened that are commonly closed by default. Content is therefore more likely to get through to the client in more locations and without special settings. HTTP is also supported by more content-distribution networks, which can affect cost in large distribution models. In general, more available hardware and software works unmodified and as intended with HTTP than with RTP/RTSP. Expertise in customizing HTTP content delivery using tools such as PHP is also more widespread.

Also, HTTP Live Streaming is supported in Safari and the media player framework on iOS. RTSP streaming is not supported.

24. Why is my stream's overall bit rate higher than the sum of the audio and video bitrates?

MPEG-2 transport streams can include substantial overhead. They utilize fixed packet sizes that are padded when the packet contents are smaller than the default packet size. Encoder and multiplexer implementations vary in their efficiency at packing media data into these fixed packet sizes. The amount of padding can vary with frame rate, sample rate, and resolution.

25. How can I reduce the overhead and bring the bit rate down?

Using a more efficient encoder can reduce the amount of overhead, as can tuning the encoder settings.

Note: Optimization in the stream segmenter is now on by default.

26. Do all media files have to be part of the same MPEG-2 Transport Stream?

No. You can mix media files from different transport streams, as long as they are separated by `EXT-X-DISCONTINUITY` tags. See the protocol specification for more detail. For best results, however, all video media files should have the same height and width dimensions in pixels.

27. Where can I get help or advice on setting up an HTTP audio/video server?

You can visit the Apple Developer Forum at <http://devforums.apple.com/>.

Also, check out *Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad*.

CHAPTER 4

Frequently Asked Questions

Document Revision History

This table describes the changes to *HTTP Live Streaming Overview*.

Date	Notes
2011-04-01	Added description of closed caption support, best practices for preparing media for iOS-based devices, and description of how to serve key files over HTTPS.
2010-11-15	Added description of timed metadata.
2010-03-25	Updated to include iPad and -optimize option for stream segmenter.
2010-02-05	Defines requirement for apps to use HTTP Live Streaming for video over cellular networks and requirement for 64 Kbps streams.
2010-01-20	Updated links to tools and sample streams. Added documentation of CODECS attribute and video streams with audio-only alternates.
2009-11-17	Fixed typos and corrected URLs for samples.
2009-09-09	Updated to include failover support.
2009-06-04	Added sample code, QuickTime X support. Reorganized document for readability.
2009-05-22	Changed title from iPhone Streaming Media Guide for Web Developers. Added URL for downloading media segmenter.
2009-03-15	New document that describes live streaming of audio and video over HTTP for iPhone.

REVISION HISTORY

Document Revision History

Multimedia Programming Interface and Data Specifications 1.0

Issued as a joint design by IBM Corporation and
Microsoft Corporation

August 1991

This document describes the programming interfaces and data specifications for multimedia that are common to both OS/2 and Windows environments. These specifications may be enhanced to incorporate new technologies or modified based on customer feedback and, as such, specifications incorporated into any final product may vary.

Microsoft is a registered trademark, and Windows is a trademark of Microsoft Corp.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

Contents

Contents	
Chapter 1	Overview of Multimedia Specifications
Resource Interchange File Format	1-1
Multimedia File Formats	1-1
Media Control Interface	1-2
Registering Multimedia Formats	1-2
Chapter 2	Resource Interchange File Format
About the RIFF Tagged File Format	2-1
Notation Conventions	2-1
Chunks	2-2
RIFF Forms	2-3
Defining and Registering RIFF Forms	2-3
Registered Form and Chunk Types	2-4
Unregistered (Form-Specific) Chunk Types	2-4
Notation for Representing Sample RIFF Files	2-5
Basic Notation for Representing RIFF Files	2-5
Escape Sequences for Four-Character Codes and String Chunks	2-7
Extended Notation for Representing RIFF Form Definitions	2-8
Atomic Labels	2-10
A Sample RIFF Form Definition and RIFF Form	2-11
Storing Strings in RIFF Chunks	2-12
NULL-Terminated String (ZSTR) Format	2-12
String Table Format	2-13
NULL-Terminated, Byte Size Prefix String (BZSTR) Series	2-13
Multiline String Format	2-13
Choosing a Storage Method	2-13
LIST Chunk	2-14
INFO List Chunk	2-14
CSET (Character Set) Chunk	2-16
Country Codes	2-16
Language and Dialect Codes	2-17
JUNK (Filler) Chunk	2-18
Compound File Structure	2-18
Structural Overview	2-19

Compound File Table of Contents (CTOC) Chunk	2-19
Structural Overview	2-19
Header Information	2-21
Parameter Table Definition	2-21
Header Parameter Table	2-22
CTOC Table Entries	2-22
Usage Codes for Extra Header and Extra Entry Fields	2-24
Compression of Compound File Elements	2-26
Compound File Element Group (CGRP) Chunk	2-27
Placement of the CTOC and CGRP Chunks	2-27
Chapter 3 Multimedia File Formats	
Bundle File Format	3-1
Device Independent Bitmap File Format	3-1
Overview of DIB Structure	3-2
Bitmap File Header	3-2
Bitmap Information Header	3-3
Information Header Structures	3-4
Bitmap Color Table	3-6
Color Table Structure	3-6
Order of Colors	3-6
Field Descriptions	3-6
Locating the Color Table	3-7
Interpreting the Color Table	3-7
Bitmap Data	3-8
Windows 3.0 Bitmap Compression Formats	3-8
Compression of 8-Bit-Per-Pixel DIBs	3-8
Compression of 4-Bit-Per-Pixel DIBs	3-9
RIFF Device-Independent Bitmap File Format	3-10
Simple RDIB Format	3-10
Extended RDIB Format	3-10
Bitmap Header Chunk	3-11
Transitional Compression	3-16
CCC Compression	3-17
Palette Chunk	3-17
External Palette Chunk	3-17
Bitmap Data Chunk	3-17
MIDI and RIFF MIDI File Formats	3-18
Palette File Format	3-18
Simple PAL Format	3-18
Extended PAL Format	3-19
Rich Text Format (RTF)	3-22
Waveform Audio File Format (WAVE)	3-22
WAVE Format Chunk	3-22
WAVE Format Categories	3-23
Pulse Code Modulation (PCM) Format	3-24
Storage of WAVE Data	3-26
FACT Chunk	3-26
Cue-Points Chunk	3-27
Examples of File Position Values	3-28
Playlist Chunk	3-29
Associated Data Chunk	3-29
Label and Note Information	3-30
Text with Data Length Information	3-30
Embedded File Information	3-31
Chapter 4 Media Control Interface	
MCI Command Strings	4-1
Example of MCI Command Use	4-2
Categories of MCI Command Strings	4-2
Command Syntax Conventions	4-3

System Commands	4-3
Required Commands	4-3
Basic Commands	4-4
Extended Commands	4-4
Extended Commands Reserved for Future Use	4-4
Creating a Command String	4-5
About MCI Device Types	4-6
Using MCI Command Strings	4-6
Opening a Device	4-6
Opening Simple Devices	4-7
Opening Compound Devices	4-7
Using the Shareable Flag	4-8
Using the Alias Flag	4-8
Opening New Device Elements	4-8
Closing a Device	4-8
Shortcuts and Variations for MCI Commands	4-9
Using All as a Device Name	4-9
Combining the Device Type and Device Element Name	4-9
Automatic Open	4-9
Automatic Close	4-9
Using Wait and Notify Flags	4-10
Using the Notify Flag	4-10
Obtaining Information From MCI Devices	4-11
The Play Command	4-11
Stop, Pause, and Resume Commands	4-11
MCI System Commands	4-12
Required Commands for All Devices	4-13
Basic Commands for Specific Device Types	4-14
CD Audio (Redbook) Commands	4-17
MIDI Sequencer Commands	4-20
Videodisc Player Commands	4-25
Waveform Audio Commands	4-29

Chapter 1

Overview of Multimedia Specifications

This document describes the file format and control interface specifications for multimedia. These specifications allow developers to use common file format and device control interfaces.

Resource Interchange File Format

The Resource Interchange File Format (RIFF), a tagged file structure, is a general specification upon which many file formats can be defined. The main advantage of RIFF is its extensibility; file formats based on RIFF can be future-proofed, as format changes can be ignored by existing applications.

The RIFF file format is suitable for the following multimedia tasks:

- Playing back multimedia data
- Recording multimedia data

- Exchanging multimedia data between applications and across platforms

Chapter 2, Resource Interchange File Format, describes the RIFF format.

Multimedia File Formats

A number of RIFF-based and non-RIFF file formats have been defined for the storage of multimedia data. Chapter 3, Multimedia File Formats, describes the following file formats:

- Bundle File Format
- Device-Independent Bitmap (DIB) and RIFF DIB file formats
- Musical Instrument Digital Interface (MIDI) and RIFF MIDI file formats
- Palette File Format
- Rich Text File Format
- Waveform Audio File Format

Media Control Interface

The Media Control Interface (MCI) is a high-level control mechanism that provides a device-independent interface to multimedia devices and resource files.

The Media Control Interface (MCI) provides a command set for playing and recording multimedia devices and resource files. Developers creating multimedia applications are encouraged to use this high-level command interface rather than the low-level functions specific to each platform. The MCI command set acts as a platform-independent layer that sits between multimedia applications and the underlying system software.

The MCI command set is extensible in two ways:

- Developers can incorporate new multimedia devices and file formats in the MCI command set by creating new MCI drivers to interpret the commands.
- New commands and command options can be added to support special features or functions required by new multimedia devices or file formats.

Using MCI, an application can control multimedia devices using simple command strings like open, play,

and close. The MCI command strings provide a generic interface to different multimedia devices, reducing the number of commands a developer needs to learn. A multimedia application might even accept MCI commands from an end user and pass them unchanged to the MCI driver, which parses the command and performs the appropriate action.

Chapter 3, Media Control Interface, describes MCI and its command set in detail.

Registering Multimedia Formats

This document discusses several multimedia codes and formats that require registration. These multimedia elements include the following:

- Compression techniques
- RIFF form types, chunk IDs, and list types
- Compound-file usage codes
- Waveform audio format codes

To register these multimedia elements, request a Multimedia Developer Registration Kit from the following group:RIFF forms;registering

Microsoft Corporation

Multimedia Systems Group
Product Marketing
One Microsoft Way
Redmond, WA 98052-6399

The Multimedia Developer Registration Kit also lists currently defined multimedia elements.

Chapter 2 Resource Interchange File Format

The Resource Interchange File Format (RIFF) is a tagged file structure developed for use on multimedia platforms. This chapter defines RIFF and describes the file structures based on RIFF. If your application requires a new file format, you should define it using the RIFF tagged file structure described in this chapter.

About the RIFF Tagged File Format

RIFF (Resource Interchange File Format) is the tagged file structure developed for multimedia resource files. The structure of a RIFF file is similar to the

structure of an Electronic Arts IFF file. RIFF is not actually a file format itself (since it does not represent a specific kind of information), but its name contains the words interchange file format in recognition of its roots in IFF. Refer to the EA IFF definition document, EA IFF 85 Standard for Interchange Format Files, for a list of reasons to use a tagged file format.

RIFF has a counterpart, RIFX, that is used to define RIFF file formats that use the Motorola integer byte-ordering format rather than the Intel format. A RIFX file is the same as a RIFF file, except that the first four bytes are RIFX instead of RIFF, and integer byte ordering is represented in Motorola format. RIFF files; structureFile formats; Resource Interchange (RIFF)

Notation Conventions

The following table lists some of the notation conventions used in this document. Further conventions and the notation for documenting RIFF forms are presented later in the document in the section Notation for Representing Sample RIFF Files. RIFF files; notational conventions

Notation	Description
<element label>	RIFF file element with the label element label
<element label: TYPE>	RIFF file element with data type TYPE
[<element label>]	Optional RIFF file element
<element label>...	One or more copies of the specified element
[<element label>]...	Zero or more copies of the specified element

Chunks

The basic building block of a RIFF file is called a chunk. Using C syntax, a chunk can be defined as follows: RIFF chunk; defined in C syntax

```
typedef unsigned long DWORD;
typedef unsigned char BYTE;

typedef DWORD FOURCC;           // Four-character code

typedef FOURCC CKID;           // Four-character-code
                                chunk identifier
typedef DWORD CKSIZE;          // 32-bit unsigned size
                                value

typedef struct {                // Chunk structure
    CKID    ckID;               // Chunk type identifier
    CKSIZE   ckSize;            // Chunk size
}
```

```

    field (size of ckData)
        BYTE      ckData[ckSize];          // Chunk data
    } CK;

```

A FOURCC is represented as a sequence of one to four ASCII alphanumeric characters, padded on the right with blank characters (ASCII character value 32) as required, with no embedded blanks. DWORD, in RIFF chunkRIFF files; FOURCC code in BYTE, in RIFF chunkFOURCCdata type; in RIFF chunk

For example, the four-character code FOO is stored as a sequence of four bytes: 'F', 'O', 'O', ' ' in ascending addresses. For quick comparisons, a four-character code may also be treated as a 32-bit number.

The three parts of the chunk are described in the following table: RIFF chunk; IDRIFF chunk; size

Part	Description
ckID	A four-character code that identifies the representation of the chunk data. A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it simply skips the number of bytes specified by ckSize plus the pad byte, if present.
ckSize	A 32-bit unsigned value identifying the size of ckData. This size value does not include the size of the ckID or ckSize fields or the pad byte at the end of ckData.
ckData	Binary data of fixed or variable size. The start of ckData is word-aligned with respect to the start of the RIFF file. If the chunk size is an odd number of bytes, a pad byte with value zero is written after ckData. Word aligning improves access speed (for chunks resident in memory) and maintains compatibility with EA IFF. The ckSize value does not include the pad byte.

We can represent a chunk with the following notation (in this example, the ckSize and pad byte are implicit): RIFF chunk; notation

```
<ckID> ( <ckData> )
```

Two types of chunks, the LIST and RIFF chunks, may contain nested chunks, or subchunks. These special chunk types are discussed later in this document. All other chunk types store a single element of binary data in <ckData>.

RIFF Forms

A RIFF form is a chunk with a RIFF chunk ID. The term

also refers to a file format that follows the RIFF framework. The following is the current list of registered RIFF forms. Each is described in Chapter 3, Multimedia File Formats.

Form Type	Description
PAL	RIFF Palette Format
RDIB	RIFF Device Independent Bitmap Format
RMID	RIFF MIDI Format
RMMP	RIFF Multimedia Movie File Format
WAVE	Waveform Audio Format

Using the notation for representing a chunk, a RIFF form looks like the following: RIFF chunk; RIFF form notation

RIFF (<formType> <ck>...)

The first four bytes of a RIFF form make up a chunk ID with values R, I, F, F. The ckSize field is required, but for simplicity it is omitted from the notation.

The first DWORD of chunk data in the RIFF chunk (shown above as <formType>) is a four-character code value identifying the data representation, or form type, of the file. Following the form-type code is a series of subchunks. Which subchunks are present depends on the form type. The definition of a particular RIFF form typically includes the following: RIFF chunk; in RIFF form

- A unique four-character code identifying the form type
- A list of mandatory chunks
- A list of optional chunks
- Possibly, a required order for the chunks

Defining and Registering RIFF Forms

The form-type code for a RIFF form must be unique. To guarantee this uniqueness, you must register any new form types before release. See Registering Multimedia Formats in Chapter 1, Overview of Multimedia Specifications, for information on registering RIFF forms.

Like RIFF forms, RIFX forms must also be registered. Registering a RIFF form does not automatically register the RIFX counterpart. No RIFX form types are currently defined.

Registered Form and Chunk Types

By convention, the form-type code for registered form types contains only digits and uppercase letters. Form-type codes that are all uppercase denote a registered, unique form type. Use lowercase letters for temporary or prototype chunk types. FOURCC data type; and registering form type RIFF chunk; registering

Certain chunk types are also globally unique and must also be registered before use. These registered chunk types are not specific to a certain form type; they can be used in any form. If a registered chunk type can be used to store your data, you should use the registered chunk type rather than define your own chunk type containing the same type of information.

For example, a chunk with chunk ID INAM always contains the name or title of a file. Also, within all RIFF files, filenames or titles are contained within chunks with ID INAM and have a standard data format.

Unregistered (Form-Specific) Chunk Types

Chunk types that are used only in a certain form type use a lowercase chunk ID. A lowercase chunk ID has specific meaning only within the context of a specific form type. After a form designer is allocated a registered form type, the designer can choose lowercase chunk types to use within that form. See Registering Multimedia Formats in Chapter 1, Overview of Multimedia Specifications, for information on registering form types.

For example, a chunk with ID scln inside one form type might contain the number of scan lines. Inside some other form type, a chunk with ID scln might mean secondary lambda number.

Notation for Representing Sample RIFF Files

RIFF is a binary format, but it is easier to comprehend an ASCII representation of a RIFF file. This section defines a standard notation used to present samples of various types of RIFF files. If you define a RIFF form, we urge you to use this notation in any file format samples you provide in your documentation. RIFF files; form notation

Basic Notation for Representing RIFF Files

The following table summarizes the elements of the RIFF notation required for representing sample RIFF files:

Notation	Description
<ckID> (<ckData>)	The chunk with ID <ckID> and data

<ckData>. As previously described, <ckID> is a four-character code which may be enclosed by single quotes for emphasis.

For example, the following notation describes a RIFF chunk with a form type of QRST. The data portion of this chunk contains a FOO subchunk. RIFF('QRST' FOO(17 23))

The following example describes an ICOP chunk containing the string Copyright Encyclopedia International.:

'ICOP' ("Copyright Encyclopedia International."Z)

<number>[<modifier>]

A number in Intel format, where <number> is an optional sign (+ or -) followed by one or more digits and modified by the optional <modifier>. Valid <modifier> values follow:

Modifier	Meaning
None	16-bit number in decimal format
H	16-bit number in hexadecimal format
C	8-bit number in decimal format
CH	8-bit number in hexadecimal format
L	32-bit number in decimal format
LH	32-bit number in hexadecimal format

Several examples follow:

0
65535
-1
0L
4a3c89HL
-1C
2LHC

Note that -1 and 65535 represent the same value. The application reading this file must know whether to interpret the number as signed or unsigned.

'<chars>'RIFF chunk;FOURCC code notationFOURCC data type;RIFF chunk notation

A four-character code (32-bit quantity) consisting of a sequence of zero to four ASCII characters <chars> in the given order. If <chars> is less than four characters long, it is implicitly padded on the right with blanks. Two single quotes is equivalent to four blanks. Examples follow.
'RIFF'
'xyz'

<chars> can include escape sequences, which are combinations of characters introduced by a backslash (\) and used to represent other characters. Escape sequences are listed in the following section.

The sequence of ASCII characters contained in <string> and modified by the optional modifier <modifier>. The quoted text can include any of the escape sequences listed in the following section. Valid <modifier> values follow:

Modifier	Meaning
none	No NULL terminator or size prefix.
Z	String is NULL-terminated
B	String has an 8-bit (byte) size prefix
W	String has a 16-bit (word) size prefix
BZ	String has a byte-size prefix and is NULL-terminated
WZ	String has a word-size prefix and is NULL-terminated

NULL-terminated means that the string is followed by a character with ASCII value 0. A size prefix is an unsigned integer, stored as a byte or a word in Intel format preceding the string characters, that specifies the length of the string. In the case of strings with BZ or WZ modifiers, the size prefix specifies the size of the string without the terminating NULL. The various string formats referred to above are discussed in "Storing Strings in RIFF Chunks," following later in this section., +

Examples follow:

- "No prefix, no NULL terminator"
- "No prefix, NULL terminator"Z
- "Byte prefix, NULL terminator"BZ

Escape Sequences for Four-Character Codes and String Chunks

The following escape sequences can be used in four-character codes and string chunks:

Strings, in RIFF chunk;	escape sequences RIFF
chunk;	escape sequence; list of FOURCC
data type;	escape sequences RIFF files;
escape sequences	New line escape sequence
Carriage return escape sequence	Line feed escape sequence
Tab escape sequence	Backspace escape sequence
Backslash escape	

sequenceQuote character escape sequence

Escape Sequence	ASCII Value	Description
\n	10	Newline character
\t	9	Horizontal tab character
\b	8	Backspace character
\r	13	Carriage return character
\f	12	Form feed character
\\	92	Backslash
\'	39	Single quote
\"	34	Double quote
\ddd	Octal ddd	Arbitrary character

Extended Notation for Representing RIFF Form Definitions

To unambiguously define the structure of new RIFF forms, document the RIFF form using the basic notation along with the following extended notation: RIFF files; sample file notation

Notation Description

<name>

A label that refers to some element of the file, where <name> is the name of the label. Examples follow:

```
<NAME-ck>
<GOBL-form>
<bitmap-bits>
<foo>
```

Conventionally, a label that refers to a chunk is named <ckID-ck>, where ckID is the chunk ID. Similarly, a label that refers to a RIFF form is named <formType-form>, where formType is the name of the form's type.

<name> ¥ elements

The actual data represented by <name> is defined as elements. This states that <name> is an abbreviation for elements, where elements is a sequence of other labels and literal data. An example follows:

```
<GOBL-form> ¥ RIFF ( 'GOBL' <form-data> )
```

This example defines label <GOBL-form> as representing a RIFF form with chunk ID GOBL and data equal to <form-data>, where <form-data> is a label that would be defined in another rule. Note that a label may represent any data, not just a RIFF chunk or form.

Note: A number of atomic labels are defined in the section Atomic Labels later in

this document. These labels refer to primitive data types.

<name:type
e>

This is the same as <name>, but it also defines <name> to be equivalent to <type>. This notation obviates the following rule:
<name> Ý <type>

This allows you to give a symbolic name to an element of a file format and to specify the element data type. An example follows:

<xyz-coordinate> Ý <x:INT> <y:INT> <z:INT>

This defines <xyz-coordinate> to consist of three parts concatenated together: <x>, <y>, and <z>. The definition also specifies that <x>, <y>, and <z> are integers. This notation is equivalent to the following:

<xyz-coordinate> Ý <x> <y> <z>

<x> Ý <INT>

<y> Ý <INT>

<z> Ý <INT>

[elements
]

An optional sequence of labels and literal data. Surrounded by square brackets, it may be considered an element itself. An example follows:

<FOO-form> Ý RIFF('FOO' [<header-ck>] <data-ck>)

This example defines form FOO with an optional header chunk followed by a mandatory data chunk.

el1 | el2

| ... |

elN

Exactly one of the listed elements must be present. An example follows:

<hdr-ck> Ý hdr(<hdr-x> | <hdr-y> | <hdr-z>)

This example defines the hdr chunk's data as containing one of <hdr-x>, <hdr-y>, or <hdr-z>.

element..

One or more occurrences of element may be present. An ellipsis has this meaning only if it follows an element; in cases such as el1 | el2 | ... | elN, the ellipsis has its ordinary English meaning. If there is any possibility of confusion, an ellipsis should only be used to indicate one or more occurrences. An example follows:

<data-ck> Ý data(<count:INT> <item:INT>...)

This example defines the data of the data chunk to contain an integer <count>, followed by one or more occurrences of the integer <item>.

[element]

....

Zero or more occurrences of element may be present. An example follows.

```
<data-ck> Y data(<count:INT>
[<item:INT>]...)
```

This example defines the data of the data chunk to contain an integer <count> followed by zero or more occurrences of an integer <item>.

```
{elements
}
```

The group of elements within the braces should be considered a single element. An example follows:

```
<blorg> Y <this> | (<that> | <other>)...
```

This example defines <blorg> to be either <this> or one or more occurrences of <that> or <other>, intermixed in any way. Contrast this with the following example:

```
<blorg> Y <this> | <that> | <other>...
```

This example defines <blorg> to be either <this> or <that> or one or more occurrences of <other>.

```
struct {
...} name
```

A structure defined using C syntax. This can be used instead of a sequence of labels if a C header (include) file is available that defines the structure. The label used to refer to the structure should be the same as the structure's typedef name. An example follows:

```
<3D_POINT> Y struct {
    coordinate    INT x;           // x-
    coordinate    INT y;           // y-
    coordinate    INT z;           // z-
    coordinate
} 3D_POINT
```

Wherever possible, the types used in the structure should be the types listed in the following section, Atomic Labels, because these types are more portable than C types such as int. The structure fields are assumed to be present in the file in the order given, with no padding or forced alignment.

Unless the RIFF chunk ID is RIFX, integer byte ordering is assumed to be in Intel format.

```
//
comment
```

An explanatory comment to a rule. An

```

example follows:
<weekend> Y 'Sat' | 'Sun' // Four-character
code
                        // for day

```

Atomic Labels

The following are atomic labels, which are labels that refer to primitive data types. Where available, the equivalent Microsoft C data type is also listed.

Datatypes Atomic labels RIFF files; primitive data type labels

Label	Meaning	MS C Type
<CHAR>	8-bit signed integer	signed char
<BYTE>	8-bit unsigned quantity	unsigned char
<INT>	16-bit signed integer in Intel format	signed int
<WORD>	16-bit unsigned quantity in Intel format	unsigned int
<LONG>	32-bit signed integer in Intel format	signed long
<DWORD>	32-bit unsigned quantity in Intel format	unsigned long
<FLOAT>	32-bit IEEE floating point number	float
<DOUBLE>	64-bit IEEE floating point number	double
<STR>	String (a sequence of characters)	
<ZSTR>	NULL-terminated string	
<BSTR>	String with byte (8-bit) size prefix	
<WSTR>	String with word (16-bit) size prefix	
<BZSTR>	NULL-terminated string with byte size prefix	
<WZSTR>	NULL-terminated string with word size prefix	

NULL-terminated means that the string is followed by a character with ASCII value 0.

A size prefix is an unsigned integer, stored as a byte or a word in Intel format, that specifies the length of the string. In the case of strings with BZ or WZ modifiers, the size prefix specifies the size of the string without the terminating NULL.

Note: The WINDOWS.H header file defines the C types BYTE, WORD, LONG, and DWORD. These types correspond to

labels <BYTE>, <WORD>, <LONG>, and <DWORD>, respectively.

A Sample RIFF Form Definition and RIFF Form

The following example defines <GOBL-form>, the hypothetical RIFF form of type GOBL. To fully document a new RIFF form definition, a developer would also provide detailed descriptions of each file element, including the semantics of each chunk and sample files documented using the standard notation. RIFF forms; example of definition

```
<GOBL-form> Ý      RIFF( 'GOBL'          // RIFF form
header
                    [<org-ck>]          // Origin chunk
                    (default (0,0,0))
                    <obj-list>)        // Series of
graphical objects
...
<org-ck> Ý      org(  <origin:3D_POINT> )
// Object-list origin

// An object is a:
<obj-list> Ý      LIST(  'obj' { <sqr-ck> |
// square,
                    <circ-ck> |
// circle,
                    <poly-ck> }... )
// or polygon

<sqr-ck> Ý      sqr( <pt1:3D_POINT>      // one vertex
                    <pt2:3D_POINT>      // another
vertex
                    <pt3:3D_POINT> )    // a third
vertex

<circ-ck> Ý      circ(  <center:3D_POINT>
// Center of circle
                    <circumPt:3D_POINT> )    //
Point on circumference

<poly-ck> Ý      poly( <pt:3D_POINT>... )    // List of
points in a polygon

<3D_POINT> Ý      struct                // Defined in
"gobl.h"
                    { INT x;              // x-coordinate
                    INT y;              // y-coordinate
                    INT z;              // z-coordinate
                    } 3D_POINT
```

Sample RIFF Form

The following sample RIFF form adheres to the form definition for form type GOBL. The file contains three subchunks: RIFF forms; example RIFF chunk; example

- An INFO list

- An org chunk
- An obj chunk

The INFO list and org chunk each have two subchunks. The INFO list is a registered global chunk that can be used within any RIFF file. The INFO list is described in the INFO List Chunk, later in this chapter. RIFF chunk; INFO chunk INFO chunk RIFF files; INFO chunk

Since the definition of the GOBL form does not refer to the INFO chunk, software that expects only org and obj chunks in a GOBL form would ignore the unknown INFO chunk.

```
RIFF( 'GOBL'
    LIST('INFO'          // INFO list containing
        filename and copyright
        INAM("A House"Z)
        ICOP("(C) Copyright Encyclopedia
        International 1991"Z)
    )

    org(2, 0, 0)          // Origin of object list

    LIST('obj'           // Object list containing two
        polygons
        poly(0,0,0 2,0,0 2,2,0, 1,3,0, 0,2,0)
        poly(0,0,5 2,0,5 2,2,5, 1,3,5, 0,2,5)
    )
)                          // End of form
```

Storing Strings in RIFF Chunks

This section lists methods for storing text strings in RIFF chunks. While these guidelines may not make sense for all applications, you should follow these conventions if you must make an arbitrary decision regarding string storage. RIFF chunk; text strings in

NULL-Terminated String (ZSTR) Format ZSTR format, in RIFF chunk

A NULL-terminated string (ZSTR) consists of a series of characters followed by a terminating NULL character. The ZSTR is better than a simple character sequence (STR) because many programs are easier to write if strings are NULL-terminated. ZSTR is preferred to a string with a size prefix (BSTR or WSTR) because the size of the string is already available as the <ckSize> value, minus one for the terminating NULL character. WSTR format, in RIFF chunk BSTR format, in RIFF chunk String Table Format String table format, in RIFF chunk

In a string table, all strings used in a structure are stored at the end of the structure in packed format. The structure includes fields that specify the offsets from the beginning of the string table to the individual strings. An example follows:

```
typedef struct
{
    INT    iWidgetNumber;    // the widget number
    WORD offszWidgetName;    // an offset to a string in
    <rgchStrTab>
    WORD offszWidgetDesc;    // an offset to a string in
    <rgchStrTab>
    INT    iQuantity;        // how many widgets
    CHAR rgchStrTab[1];      // string table (allocate
    as large as needed)
} WIDGET;
```

If multiple chunks within the file need to reference variable-length strings, you can store the strings in a single chunk that acts as a string table. The chunks that refer to the strings contain offsets relative to the beginning of the data part of the string table chunk.

NULL-Terminated, Byte Size Prefix String (BZSTR)
SeriesBZSTR format, in RIFF chunk

In a BZSTR series, a series of strings is stored in packed format. Each string is a BZSTR, with a byte size prefix and a NULL terminator. This format retains the ease-of-use characteristics of the ZSTR while providing the string size, allowing the application to quickly skip unneeded strings.

Multiline String Format

When storing multiline strings, separate lines with a carriage return/line feed pair (ASCII 13/ASCII 10 pair). Although applications vary in their requirements for new line symbols (carriage return only, line feed only, or both), it is generally easier to strip out extra characters than to insert extra ones. Inserting characters might require reallocating memory blocks or pre-scanning the chunk before allocating memory for it. Multiline string format, in RIFF chunk

Choosing a Storage Method

The following lists guidelines for deciding which storage method is appropriate for your application. Strings, in RIFF chunk; storage methods

Usage

Recommended Format

Chunk data contains nothing except a string	ZSTR (NULL-terminated string) format.
Chunk data contains a number of fields, some of which are variable-length strings	String-table format
Multiple chunks within the file need to reference variable-length strings	String-table format
Chunk data stores a sequence of strings, some of which the application may want to skip	BZSTR (NULL-terminated string with byte size prefix) series
Chunk data contains multiline strings	A multiline string format

LIST Chunk

A LIST chunk contains a list, or ordered sequence, of subchunks. A LIST chunk is defined as follows:

```
LIST( <list-type> [<chunk>]... )
```

The <list-type> is a four-character code that identifies the contents of the list.

If an application recognizes the list type, it should know how to interpret the sequence of subchunks. However, since a LIST chunk may contain only subchunks (after the list type), an application that does not know about a specific list type can still walk through the sequence of subchunks.

Like chunk IDs, list types must be registered, and an all-lowercase list type has meaning relative to the form that contains it. See Registering Multimedia Formats in Chapter 1, Overview of Multimedia Specifications, for information on registering list types.

INFO List Chunk

The INFO list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An INFO list is a LIST chunk with list type INFO. The following shows a sample INFO list chunk:

```
LIST('INFO' INAM("Two Trees"Z)
      ICMT("A picture for the opening screen"Z) )
```

An INFO list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an INFO list. Each chunk contains a ZSTR, or null-terminated text string.

Chunk ID	Description
IARL	Archival Location. Indicates where the subject of the file is archived.
IART	Artist. Lists the artist of the original subject of the file. For example, Michaelangelo.
ICMS	Commissioned. Lists the name of the person or organization that commissioned the subject of the file. For example, Pope Julian II.
ICMT	Comments. Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do not include newline characters.
ICOP	Copyright. Records the copyright information for the file. For example, Copyright Encyclopedia International 1991. If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	Creation date. Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example, 1553-05-03 for May 3, 1553.
ICRP	Cropped. Describes whether an image has been cropped and, if so, how it was cropped. For example, lower right corner.
IDIM	Dimensions. Specifies the size of the original subject of the file. For example, 8.5 in h, 11 in w.
IDPI	Dots Per Inch. Stores dots per inch setting of the digitizer used to produce the file, such as 300.
IENG	Engineer. Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example, Smith, John; Adams, Joe.
IGNR	Genre. Describes the original work, such as, landscape, portrait, still life, etc.
IKEY	Keywords. Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example, Seattle; aerial view; scenery.
ILGT	Lightness. Describes the changes in lightness settings on the digitizer required to produce the file. Note that the format of this information depends on hardware used.
IMED	Medium. Describes the original subject of the file, such as, computer image, drawing, lithograph, and so forth.
INAM	Name. Stores the title of the subject of the file, such as, Seattle From Above.

IPLT	Palette Setting. Specifies the number of colors requested when digitizing an image, such as 256.
IPRD	Product. Specifies the name of the title the file was originally intended for, such as Encyclopedia of Pacific Northwest Geography.
ISBJ	Subject. Describes the contents of the file, such as Aerial view of Seattle.
ISFT	Software. Identifies the name of the software package used to create the file, such as Microsoft WaveEdit.
ISHP	Sharpness. Identifies the changes in sharpness for the digitizer required to produce the file (the format depends on the hardware used).
ISRC	Source. Identifies the name of the person or organization who supplied the original subject of the file. For example, Trey Research.
ISRF	Source Form. Identifies the original form of the material that was digitized, such as slide, paper, map, and so forth. This is not necessarily the same as IMED.
ITCH	Technician. Identifies the technician who digitized the subject file. For example, Smith, John.

CSET (Character Set) Chunk

To define character-set and language information for a RIFF file, use the CSET chunk. The CSET chunk defines the code page and country, language, and dialect codes for the file. These values can be overridden for specific file elements; see Usage Codes for Extra Header and Extra Entry Fields, later in this chapter, for information on specifying character set information in a compound file.

The CSET chunk is defined as follows:

```
<CSET chunk> Ÿ CSET( <wCodePage:WORD>
                     <wCountryCode:WORD>
                     <wLanguageCode:WORD>
                     <wDialect:WORD> )
```

The fields are as follows:

Field	Description
wCodePage	Specifies the code page used for file elements. If the CSET chunk is not present, or if this field has value zero, assume standard ISO 8859/1 code page (identical to code page 1004 without code points defined in hex columns 0, 1, 8, and 9).
wCountryCode	Specifies the country code used

wLanguage,
wDialect

for file elements. See Country Codes, following this section, for a list of currently defined country codes.
If the CSET chunk is not present, or if this field has value zero, assume USA (country code 001).
Specify the language and dialect used for file elements. See Language and Dialect Codes, later in this chapter, for a list of language and dialect codes.
If the CSET chunk is not present, or if these fields have value zero, assume US English (language code 9, dialect code 1).

Country Codes

Use one of the following country codes in the wCountryCode field:

Country Code	Country
000	None (ignore this field)
001	USA
002	Canada
003	Latin America
030	Greece
031	Netherlands
032	Belgium
033	France
034	Spain
039	Italy
041	Switzerland
043	Austria
044	United Kingdom
045	Denmark
046	Sweden
047	Norway
049	West Germany
052	Mexico
055	Brazil
061	Australia
064	New Zealand
081	Japan
082	Korea
086	Peoples Republic of China
088	Taiwan
090	Turkey
351	Portugal
352	Luxembourg
354	Iceland
358	Finland

Language and Dialect Codes

Specify one of the following pairs of language-code and dialect-code values in the wLanguage and wDialect fields:

Language Code	Dialect Code	Language
0	0	None (ignore these fields)
1	1	Arabic
2	1	Bulgarian
3	1	Catalan
4	1	Traditional Chinese
4	2	Simplified Chinese
5	1	Czech
6	1	Danish
7	1	German
7	2	Swiss German
8	1	Greek
9	1	US English
9	2	UK English
10	1	Spanish
10	2	Spanish Mexican
11	1	Finnish
12	1	French
12	2	Belgian French
12	3	Canadian French
12	4	Swiss French
13	1	Hebrew
14	1	Hungarian
15	1	Icelandic
16	1	Italian
16	2	Swiss Italian
17	1	Japanese
18	1	Korean
19	1	Dutch
19	2	Belgian Dutch
20	1	Norwegian - Bokmal
20	2	Norwegian - Nynorsk
21	1	Polish
22	1	Brazilian Portuguese
22	2	Portuguese
23	1	Rhaeto-Romanic
24	1	Romanian
25	1	Russian
26	1	Serbo-Croatian (Latin)
26	2	Serbo-Croatian (Cyrillic)
27	1	Slovak
28	1	Albanian
29	1	Swedish
30	1	Thai
31	1	Turkish
32	1	Urdu
33	1	Bahasa

JUNK (Filler) Chunk

A JUNK chunk represents padding, filler or outdated information. It contains no relevant data; it is a space filler of arbitrary size. The JUNK chunk is defined as follows:

```
<JUNK chunk> Ý JUNK( <filler> )
```

where <filler> contains random data.

Compound File Structure

The compound file structure is a RIFF-based structure upon which multimedia file formats can be defined. The compound file structure is a parameterized structure that provides for the following:

- Storage of multimedia data elements
- Direct access to multimedia data elements (as opposed to sequential searching)

The goals of the compound file structure are to maximize flexibility and extensibility while minimizing implementation costs. Using the compound file structure, developers of multimedia data formats can define both simple and complex file formats.

The structure is flexible enough to be used for many purposes, but it can be simplified for use with simple file formats. Designers of new multimedia file formats can restrict the use of standard header fields, requiring some and removing others.

For example, a developer might define a compound file format that stores a series of bitmaps in a single file, thus reducing compact disc seek times. Another developer might define a compound file format that contains a special type of audio resource, using the compound file header information to identify the attributes of the audio data stored within.

Structural Overview

Files based upon the compound file structure contain the following two RIFF chunks at their topmost level:

- Compound File Table of Contents (CTOC) chunk
- Compound File Element Group (CGRP) chunk

The CTOC chunk indexes the CGRP chunk, which contains the actual multimedia data elements. Defined using the standard chunk notation, a compound file is represented as follows:

<compound file> Ý RIFF('type' <CTOC> <CGRP>)

where 'type' is a FOURCC indicating the file type.

This section describes the CTOC and CGRP chunks in detail.

Compound File Table of Contents (CTOC) Chunk

The CTOC chunk functions mainly as an index, allowing direct access to elements within a compound file. The CTOC chunk also contains information about the attributes of the entire file and of each media element within the file.

To provide the maximum flexibility for defining compound file formats, the CTOC chunk can be customized at several levels. The CTOC chunk contains fields whose length and usage is defined by other CTOC fields. This parameterization adds complexity, but it provides flexibility to file format designers and allows applications to correctly read data without necessarily knowing the specific file format definition.

Structural Overview

The CTOC chunk defines the contents of the CGRP chunk. The CTOC chunk has the following components:

- Header information defining the size of the CTOC chunk, the number of entries in the CGRP chunk, the size of the CGRP chunk, and general information about the entire header file
- A parameter table definition defining the size and contents of the header parameter table and CTOC table entries
- A header parameter table defining attributes that apply to the entire compound file.
- CTOC table entries defining the location, size, name, and attributes of the compound file elements contained in the CGRP chunk.

These components appear sequentially in the CTOC chunk. The individual fields in the CTOC chunk are the following:

```
<CTOC-chunk>Ý CTOC (
    <dwHeaderSize:DWORD>           // Header
information
    <dwEntriesTotal:DWORD>
    <dwEntriesDeleted:DWORD>
    <dwEntriesUnused:DWORD>
    <dwBytesTotal:DWORD>
    <dwBytesDeleted:DWORD>
```

```

<dwHeaderFlags:DWORD>

    <wEntrySize:WORD>          // Parameter table
definition
    <wNameSize:WORD>
    <wExHdrFields:WORD>
    <wExEntFields:WORD>
    <awExHdrFldUsage:WORD[wExHdrFields]>
    <awExEntFldUsage:WORD[wExEntFields]>

                                // Header parameter table
    <adwExHdrField:DWORD[wExHdrFields]>
    [<bHeaderPad:BYTE>]
    [<CTOC-table-entry>]      // CTOC table
entries
)

```

Each CTOC table entry is defined as follows:

```

<CTOC-table-entry> Y
    <dwOffset:DWORD>
    <dwSize:DWORD>
    <dwMedType:DWORD>
    <dwMedUsage:DWORD>
    <dwCompressTech:DWORD>
    <dwUncompressBytes:DWORD>
    <adwExEntField:DWORD[wExEntFields]>
    <bEntryFlags:BYTE>
    <achName:CHAR[wNameSize]>
    [<bEntryPad:BYTE>]...

```

The following sections describe each field in detail.
Header Information

The header information section defines general information about the CTOC header and about the entire compound file. It contains the following fields:

Field Name	Description
dwHeaderSize	Combined size of header information, parameter table definition, and header parameter table. Use this value to locate the start of the CTOC table entries within the CTOC chunk.
dwEntriesTotal	Total number of CTOC table entries, including unused entries and entries corresponding to deleted elements.
dwEntriesDeleted	Number of CTOC table entries that correspond to deleted elements.
dwEntriesUnused	Number of CTOC table entries that are unused.
dwBytesTotal	Combined size of all CGRP elements, including deleted elements.
dwBytesDeleted	Combined size of all deleted CGRP elements.
dwHeaderFlags	Flags that give information about the

entire compound file. The following flags may be used:

CTOC_HF_SEQUENTIAL

Valid CTOC table entries are arranged in sequential order. If this flag is not set, the CTOC table entries may be in an arbitrary order.

CTOC_HF_MEDSUBTYPE

The dwMedUsage field of each CTOC table entry contains a FOURCC that indicates how the element is used. If this flag is not set, the dwMedUsage field contains information as defined by the form type.

Parameter Table Definition

The parameter table definition defines the size and contents of the header parameter table and CTOC table. It contains the following fields:

Field Name	Description
wEntrySize	Size of each CTOC table entry, including any pad bytes.
wNameSize	Size of the achName field of each CTOC table entry. Each achName field must be padded with null characters to this length. The achName field is a null-terminated string, so it always contains at least one trailing null character.
wExHdrFields	Number of extra header fields, or entries in the awExHdrFldUsage and adwExHdrField arrays.
wExEntFields	Number of extra entry fields, or entries in the awExEntFldUsage and adwExEntField arrays.
awExHdrFldUsage	Array of extra header field usage fields. Each entry in this array corresponds to the same numbered entry in the adwExHdrField array and defines how that entry is interpreted. Valid usage codes for each field in this array are listed in Usage Codes for Extra Header and Extra Entry Fields, later in this chapter. The number of WORDs in this array is defined by the wExHdrFields value.
awExEntFldUsage	Array of extra entry field usage fields. Each entry in this array corresponds to the same numbered entry in the adwExEntField array, present in each CTOC table entry, and defines how that entry is interpreted. Valid usage

codes for each field in this array are listed in Usage Codes for Extra Header and Extra Entry Fields, later in this chapter. The number of WORDs in this array is defined by the wExEntFields value.

Header Parameter Table

The header parameter table is an optional component generally used to define attributes of the entire compound file.

Field Name	Type
adwExHdrField	Extra header fields. The usage of each cell in the array is defined by the corresponding cell in the awExHdrFldUsage array. The number of DWORDs in this array is defined by the wExHdrFields value.
bHeaderPad	Zero or more NULL pad bytes. There must be enough padding in this field to make the CTOC header an even number of bytes in length.

CTOC Table Entries

The CTOC table entries define the location, size, name, and other information about the individual compound file elements contained in the CGRP chunk. The number of CTOC table entries is determined by the dwEntriesTotal field in the header information of the CTOC chunk. Each CTOC table entry is a structure containing the following fields:

Field Name	Description
dwOffset	Byte offset of the compound file element measured from the beginning of the data portion of the CGRP chunk. For example, if dwOffset is 1000 and the chunk ID of the CGRP chunk is at offset 500, the element is at offset 1508 (1000+500+4 (chunk ID)+4 (chunk size field)).
dwSize	Size of the element in bytes.
dwMedType	FOURCC value identifying the media element type of the compound file element. This field may be zero if the compound file element is not to be interpreted as a standalone file. If the compound file element is a RIFF form, then the media element type is the same as the RIFF form type.

dwMedUsage Extra usage information for the compound file element. If the CTOC_HF_MEDSUBTYPE flag is set in the dwHeaderFlags field, this field contains a FOURCC that indicates how the element is used. To avoid name conflicts, this FOURCC must be registered. See Registering Multimedia Formats in Chapter 1, Overview of Multimedia Specifications, for information on usage codes. If the CTOC_HF_MEDSUBTYPE flag is not set in the dwHeaderFlags field, this field contains 32 bits of information interpreted as defined by the form type.

dwCompressTech Compression technique used to compress the media element. If this value is zero, the element is not compressed. See Compression of Compound File Elements, later in this chapter, for more information.

dwUncompressBytes Number of bytes the compound file element occupies in memory after decompression. This value assumes the decompression technique identified in the dwCompressTech field. If the dwCompressTech field is 0, then the compound file element is not compressed, and this field should equal the dwSize field.

adwExEntField Array of extra entry fields defining attributes of this compound file element. The usage of each cell in the array is defined by the corresponding cell in the awExEntFldUsage array. The number of DWORDs in this array is defined by the wExEntFields value.

bEntryFlags Flags giving information about the compound file element or this CTOC table entry. Possible values follow; these may be combined:

- CTOC_EF_DELETED
Compound file is marked as deleted and should not be accessed. Do not combine this flag with the CTOC_EF_UNUSED flag.
- CTOC_EF_UNUSED
CTOC table entry is unused and does not refer to any compound file element. This entry can be used to refer to a new compound file element. Do not combine this flag with the CTOC_EF_DELETED flag.

achName Array of characters containing the name of the compound file element. The number of bytes in this array is defined by the wNameSize value. The string must be padded on the right

with NULL characters and must be terminated by at least one NULL character. This field must be an odd number of bytes in length and must be at least one byte long.

bEntryPad Zero or more NULL pad bytes as needed to make the table entry an even number of bytes in length.

Usage Codes for Extra Header and Extra Entry Fields

The following are valid usage codes for elements in the awExHdrFldUsage and awExEntFldUsage arrays, both of which are fields of the CTOC header. These arrays define the meaning of data stored in the adwExHdrField and adwExEntField extra fields. All usage codes apply to both header fields and entry fields, unless explicitly stated otherwise.

Values marked in the extra header field arrays generally apply to all elements in the CFRG chunk, while values marked in the extra entry field arrays generally apply only to the element referenced by the corresponding CTOC table entry.

Flag	Description
CTOC_EFU_UNUSED (0x00)	The field is unused. This usage code may be used to logically delete a header field.
CTOC_EFU_LASTMODTIM E (0x01)	When used to describe an extra header field, the field contains the time that any portion of the CTOC or CGRP was last modified. When used to describe an extra entry field, the field contains the time that the corresponding CTOC table entry, or the compound file element it refers to, was last modified. The field is interpreted as a DWORD containing the number of seconds that have elapsed since 00:00:00 Greenwich Mean Time (GMT), January 1, 1970.
CTOC_EFU_CODEPAGE	The field contains the code page and country code for the achName field. These values override any values specified in a CSET chunk. When used to describe an extra header field, the field contains code-page and country-code information for all CTOC table entries. When used to describe an extra entry field, the field contains information for that specific CTOC table entry. The low-order word of the field

contains one of the following code page values:

Zero

Use standard ISO 8859/1 code page. This is identical to code page 1004 without code points defined in hex columns 0, 1, 8, and 9.

CTOC_CHARSET_CODEPAGE

(0x0000nnnn)

Use code page 0xn timer, where 0xn timer is the 16-bit code page number. For example, 0x00000352 for OS/2 code page 850, or 0x000004E4 for Windows 3.1 code page 1252.

The high-order word contains one of the following country codes:

Zero

Ignore this field.

Country code

See Country Codes, earlier in this chapter, for a list of currently defined country codes.

CTOC_EFU_LANGUAGE

The field contains language and dialect information for the achName field. These values override any values specified in a CSET chunk.

When used to describe an extra header field, the field contains language information for all CTOC table entries. When used to describe an extra entry field, the field contains information for that specific CTOC table entry.

The low-order word of the field contains one of the following language codes:

Zero

Ignore this field.

Language code

See Language and Dialect Codes, earlier in this chapter, for a list of currently defined language codes.

The high-order word of the field contains one of the following dialect codes:

Zero

Ignore this field.

Dialect code

See Language and Dialect Codes, earlier in this chapter, for a list of currently defined dialect codes.

CTOC_EFU_COMPRESSPA Specifies a compression
 RAM0 (0x05) through parameter. See Compression of
 CTOC_EFU_COMPRESSPA Compound File Elements, later in
 RAM9 (0x14) this chapter.

Compression of Compound File Elements

Compound file elements can be compressed. The
 dwCompressTech field of a CTOC table entry contains a
 FOURCC compression technique identifier for the
 corresponding compound file element. If the field is
 zero, the compound file element is not compressed.

The definition of a specific compression technique may
 specify that either the entire compound file element is
 compressed, or that some specific subset, for example
 one or more RIFF chunks, is compressed.

The dwUncompressSize field contains the number of bytes
 that the compound file element will occupy in memory
 after decompression. If the compound file element is
 not compressed, this field contains the same value as
 the dwSize field, which identifies the file size of the
 compound file element.

Compression techniques may demand extra header fields
 or extra entry fields for decompression parameters.
 Compression technique identifiers, and any new entry
 fields corresponding to decompression technique
 parameters, must be unique. See Registering Multimedia
 Formats in Chapter 1, Overview of Multimedia
 Specifications, for information on registering
 compression techniques.

Compound File Element Group (CGRP) Chunk

The actual elements of data referenced by the CTOC
 chunk are stored in a compound file Element Group
 (CGRP) chunk. The CGRP chunk contains all the compound
 file elements, concatenated together into one
 contiguous block of data. Some of the elements in the
 CGRP chunk might be unused, if the element was marked
 for deletion or was altered and stored elsewhere within
 the CGRP chunk.

Elements within the CGRP chunk are of arbitrary size
 and can appear in a specific or arbitrary order,
 depending upon the file format definition. Each element
 is identified by a corresponding CTOC table entry.

Using the standard RIFF notation, the CGRP chunk is
 defined as follows:

```
<CGRP-chunk> Ȳ CGRP([<compound file element>]...)
```

Placement of the CTOC and CGRP Chunks

The specific file format definition can specify which of the two chunks appear first the data file. Generally, the CTOC chunk is placed at the front of the file to reduce the seek and read times required to access it. During authoring time, an application might place the CTOC chunk at the end of the file, so it can be expanded as elements are added to the CGRP chunk.

Chapter 3 Multimedia File Formats

This chapter describes the multimedia file formats. Most of these file formats are based on the Resource Interchange File Format (RIFF), described in Chapter 2.

This chapter describes the following file formats:

- Bundle File Format (BND)
- Device Independent-Bitmap File Format (DIB)
- RIFF DIB File Format (RDIB)
- Musical Instrument Digital Interface File Format (MIDI)
- RIFF MIDI File Format (RMID)
- Palette File Format (PAL)
- Rich Text Format (RTF)
- Waveform Audio File Format (WAVE)

Bundle File Format

The Bundle (BND) format contains a series of RIFF chunks or other multimedia files. The BND file is defined as follows:

```
<BND-file> Y RIFF('BND' <CTOC-chunk> <CGRP-chunk> )
```

The <CTOC-chunk> and <CGRP-chunk> formats are defined in Compound File Structure, in Chapter 2, Resource Interchange File Format.

Each compound file element must be capable of standing alone as an independent file. An element may not be a random chunk (except the RIFF chunk, indicating a RIFF file) or random binary data (unless the binary data is supposed to be treated as a file).

Device Independent Bitmap File Format

The Device Independent Bitmap (DIB) format represents

bitmap images in a device-independent manner. Bitmaps can be represented at 1, 4, and 8 bits per pixel, with a palette containing colors represented in 24 bits. Bitmaps can also be represented at 24 bits per pixel without a palette and in a run-length encoded format. File formats; Device-Independent Bitmap (DIB) DIB files; Multimedia Windows

This documentation describes three types of DIB files:

- Windows version 3.0 device-independent bitmap files
- OS/2 Presentation Manager version 1.2 device-independent bitmap files
- RIFF device-independent bitmap files

The Windows 3.0 and Presentation Manager 1.2 DIBs are similar, so they are discussed together.

Overview of DIB Structure

Windows 3.0 and Presentation Manager 1.2 DIB files consist of the following sequence of data structures: DIB files; structure

- A file header
- A bitmap information header
- A color table
- An array of bytes that defines the bitmap bits

The following sections describe each of these structures.

Bitmap File Header

The bitmap file header contains information about the type, size, and layout of a device-independent bitmap (DIB) file. In both the Windows 3.0 and Presentation Manager 1.2 DIBs, it is defined as a BITMAPFILEHEADER data structure: BITMAPFILEHEADER data structure; in DIB file; DIB files; bitmap file header

```
typedef struct tagBITMAPFILEHEADER {
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER;
```

The following table describes the fields.

Field	Description
bftype	Specifies the file type. It must consist of the character sequence BM (WORD value 0x4D42).
bfsz	Specifies the file size in bytes.
bfReserved1	Reserved. Must be set to zero.
bfReserved2	Reserved. Must be set to zero.
bfOffBits	Specifies the byte offset from the BITMAPFILEHEADER structure to the actual bitmap data in the file.

Bitmap Information Header

The BITMAPINFO and BITMAPCOREINFO data structures define the dimensions and color information for Windows 3.0 and Presentation Manager 1.2 DIBs, respectively. They are defined as follows:

Windows 3.0 DIB	Presentation Manager 1.2 DIB
typedef struct	typedef struct
tagBITMAPINFO {	_BITMAPCOREINFO {
BITMAPINFOHEADER	BITMAPCOREHEADER
bmiHeader;	bmciHeader;
RGBQUAD bmiColors[1];	RGBTRIPLE
} BITMAPINFO;	bmciColors[1];
	} BITMAPCOREINFO;

These structures are essentially alike, and this section discusses them simultaneously. Each field name for the Windows BITMAPINFO structure is followed by the corresponding field name for the Presentation Manager BITMAPCOREINFO 1.2 structure, in parentheses.

The following table describes the fields.

Windows (PM) Field	Description
bmiHeader (bmciHeader)	Specifies information about the dimensions and color format of the DIB. The BITMAPINFOHEADER and BITMAPCOREHEADER data structures are described in the next section.
bmiColors (bmciColors)	Specifies the DIB color table. The RGBQUAD and RGBTRIPLE data structures are described in Bitmap Color Table, later in this chapter.

Information Header Structures

The BITMAPINFOHEADER and BITMAPCOREHEADER structures contain information about the dimensions and color format of Windows 3.0 and Presentation Manager 1.2

DIBs, respectively. They are defined as follows:
 DIB files; bitmap information header BITMAPINFOHEADER
 data structure; in DIB file

Windows 3.0 DIB <pre>typedef struct tagBITMAPINFOHEADER { DWORD biSize; DWORD biWidth; DWORD biHeight; WORD biPlanes; WORD biBitCount; DWORD biCompression; DWORD biSizeImage; DWORD biXPelsPerMeter; DWORD biYPelsPerMeter; DWORD biClrUsed; DWORD biClrImportant; } BITMAPINFOHEADER;</pre>	Presentation Manager 1.2 DIB <pre>typedef struct tagBITMAPCOREHEADER { DWORD bcSize; WORD bcWidth; WORD bcHeight; WORD bcPlanes; WORD bcBitCount; } BITMAPCOREHEADER;</pre>
--	---

Because these structures are essentially alike, except for the added fields in the Windows 3.0 structure, this section discusses them simultaneously. Each field name for the Windows structure is followed by the corresponding field name for the Presentation Manager structure, in parentheses.

Common Fields

The following fields are present in both the Windows 3.0 and Presentation Manager 1.2 formats:

Windows (PM) Field	Description
biSize (bcSize)	Specifies the number of bytes required by the BITMAPINFOHEADER structure. You can use this field to distinguish between Windows 3.0 and Presentation Manager 1.2 DIBs.
biWidth (bcWidth)	Specifies the width of the DIB in pixels.
biHeight (bcHeight)	Specifies the height of the DIB in pixels.
biPlanes (bcPlanes)	Specifies the number of planes for the target device. Must be set to 1.
wBitCount (bcBitCount)	Specifies the number of bits-per-pixel. See Interpreting the Color Table, later in this section, for more information.

Windows Fields

The following fields are present only in the Windows 3.0 BITMAPINFOHEADER structure:

Field	Description
biCompression	Specifies the type of compression for a compressed bitmap. It can be one of the following values:
Value	Meaning
BI_RGB	Specifies that the bitmap is not compressed.
BI_RLE4	Specifies a run-length encoded format for bitmaps with 4 bits-per-pixel. The compression format is a two-byte format consisting of a count byte followed by two word-length color indexes.
BI_RLE8	Specifies a run-length encoded format for bitmaps with 8 bits-per-pixel. The compression format is a two-byte format consisting of a count byte followed by a color-index byte.

See Windows 3.0 Bitmap Compression Formats later in this document for information about the encoding schemes.

biSizeImage	Specifies the size in bytes of the image.
biXPelsPerMeter	Specifies the horizontal resolution in pixels per meter of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.
biYPelsPerMeter	Specifies the vertical resolution in pixels per meter of the target device for the bitmap.
biClrUsed	Specifies the number of color values in the color table actually used by the bitmap. Possible values follow.
Value	Result
0	Bitmap uses the maximum number of colors corresponding to the value of the wBitCount field.
Nonzero	If the wBitCount value is less than 24, the biClrUsed value indicates the actual number of colors which the graphics engine or device driver will access.

If the wBitCount value is 24, the biClrUsed value indicates the size of the reference color table used to optimize performance of Windows color palettes.

If the bitmap is a packed bitmap (that is, a bitmap in which the bitmap

array immediately follows the BITMAPINFO header and which is referenced by a single pointer), the biClrUsed field must be set to 0 or to the actual size of the color table. Packed bitmap See Interpreting the Color Table, later in this section, for more information on how this field affects the interpretation of the color table.

biClrImportant Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0, then all colors are important.

Bitmap Color Table

The color table is a collection of 24-bit RGB values. There are as many entries in the color table as there are colors in the bitmap. The color table isn't present for bitmaps with 24 color bits because each pixel is represented by 24-bit RGB values in the actual bitmap data area. Color table; and bitmap colors[bitmap]DIB files; bitmap color table RGBQUAD data structure; in DIB file RGBTRIPLE data structure; in DIB file

Color Table Structure

The color table for Windows 3.0 and Presentation Manager 1.2 DIBs consists of an array of RGBQUAD and RGBTRIPLE structures, respectively. These structures are defined as follows:

Windows 3.0 DIB <pre>typedef struct tagRGBQUAD { BYTE rgbBlue; BYTE rgbGreen; BYTE rgbRed; BYTE rgbReserved; } RGBQUAD;</pre>	Presentation Manager 1.2 DIB <pre>typedef struct tagRGBTRIPLE { BYTE rgbtBlue; BYTE rgbtGreen; BYTE rgbtRed; } RGBTRIPLE;</pre>
--	--

Because these structures are essentially alike, this section discusses them simultaneously. Each field name for the Windows RGBQUAD structure is followed by the corresponding field name for the Presentation Manager RGBTRIPLE structure, in parentheses.

Order of Colors

The colors in the table should appear in order of importance. This can help a device driver render a bitmap on a device that cannot display as many colors

as there are in the bitmap. If the DIB is in Windows 3.0 format, the driver can use the biClrImportant field of the BITMAPINFOHEADER structure to determine which colors are important. Color table; color order RGB values; color table intensity Color table; color intensity BITMAPINFOHEADER data structure; bitmap color importance

Field Descriptions

The RGBQUAD (RGEBTRIPLE) structure contains the following fields:

Windows (PM) Field	Description
rgbBlue (rgbtBlue)	Specifies the blue intensity.
rgbGreen (rgbtGreen)	Specifies the green intensity.
rgbRed (rgbtRed)	Specifies the red intensity.
rgbReserved (no PM equivalent)	Not used. Must be set to 0.

Locating the Color Table

An application can use the biSize (bcSize) field of the BITMAPINFOHEADER (BITMAPCOREHEADER) structure to locate the color table. Each of the following statements assigns the pColor variable the byte offset of the color table from the beginning of the file: BITMAPINFOHEADER data structure; color table location BITMAPCOREHEADER data structure; color table location

```
// Windows 3.0 DIB
pColor = (LPSTR)pBitmapInfo + (WORD)pBitmapInfo->biSize
```

```
// Presentation Manager 1.2 DIB
pColor = (LPSTR)pBitmapCoreInfo +
(WORD)pBitmapCoreInfo->bcSize
```

Interpreting the Color Table

The biSize (bcSize) field of the BITMAPINFOHEADER (BITMAPCOREHEADER) structure specifies how many bits define each pixel and specifies the maximum number of colors in the bitmap. Its value affects your interpretation of the color table. BITMAPINFOHEADER data structure; color definition BITMAPCOREHEADER data structure; color definition Color table; defining number of colors

The biSize (bcSize) field can have any of the following values:

Value	Meaning
1	The bitmap is monochrome, and the color

- table contains two entries. Each bit in the bitmap array represents a pixel. If the bit is clear, the pixel is displayed with the color of the first entry in the color table. If the bit is set, the pixel has the color of the second entry in the table.
- 4 Monochrome bitmap
The bitmap has a maximum of 16 colors. Each pixel in the bitmap is represented by a four-bit index into the color table. For example, if the first byte in the bitmap is 0x1F, then the byte represents two pixels. The first pixel contains the color in the second table entry, and the second pixel contains the color in the 16th table entry.
- 8 The bitmap has a maximum of 256 colors. Each pixel in the bitmap is represented by a byte-sized index into the color table. For example, if the first byte in the bitmap is 0x1F, then the first pixel has the color of the thirty-second table entry.
- 24 The bitmap has a maximum of 224 colors. The bmiColors (bmciColors) field is NULL, and each three bytes in the bitmap array represent the relative intensities of red, green, and blue, respectively, of a pixel.

Note on Windows DIBs

For Windows 3.0 DIBs, the field of the BITMAPINFOHEADER structure specifies the number of color indexes in the color table actually used by the bitmap. If the biClrUsed field is set to 0, the bitmap uses the maximum number of colors corresponding to the value of the field.

Bitmap Data

The bits in the array are packed together, but each line of pixels, or scan line, must be zero-padded to end on a LONG boundary. When the bitmap is in memory, segment boundaries can appear anywhere in the bitmap. The origin of the bitmap is the lower-left corner. The following section discusses compression formats for the Windows 3.0 bitmap data.

Windows 3.0 Bitmap Compression Formats

Windows supports run-length encoded formats for compressing 4- and 8-bit bitmaps. Compression reduces the disk and memory storage required for the bitmap. The following sections describe the compression formats.

DIB files; Windows compression formats

Compression of 8-Bit-Per-Pixel DIBs

When the biCompression field is set to BI_RLE8, the bitmap is compressed using a run-length encoding format for an 8-bit bitmap. This format uses two modes: Compression format; 8-bit DIBs[eight] Run-length encoded format; 8-bit DIBs[eight]

- Encoded mode
- Absolute mode

Both modes can occur anywhere throughout a single bitmap.

Encoded Mode

Encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. Encoded mode; 8-bit DIBs[eight]

Also, the first byte of the pair can be set to zero to indicate an escape that denotes an end of line, end of bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. In encoded mode, the second byte has a value of 0 to 2.

The following table shows the meaning of the second byte:

Second Byte	Meaning
0	End of line.
1	End of bitmap.
2	Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offset of the next pixel from the current position.

Absolute Mode

Absolute mode is signalled by the first byte set to zero and the second byte set to a value between 03H and FFH.

The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. Absolute mode; 8-bit DIBs[eight]

Each run must be aligned on a word boundary.

The following example shows the hexadecimal values of an 8-bit RLE bitmap. Under Expanded Data, the two-digit values represent a color index for a single pixel.

Compressed Data	Expanded Data
03 04	04 04 04
05 06	06 06 06 06 06

```

00 03 45 56 67 00      45 56 67
02 78                  78 78
00 02 05 01            move 5 right and 1 down
02 78                  78 78
00 00                  end of line
09 1E                  1E 1E 1E 1E 1E 1E 1E 1E
00 01                  end of RLE bitmap

```

Compression of 4-Bit-Per-Pixel DIBs

When the field is set to BI_RLE4, the bitmap is compressed using a run-length encoding format for a 4-bit bitmap. This format uses two modes: Compression format; 4-bit DIBs[four]

- Encoded mode
- Absolute mode

Encoded ModeEncoded mode; 4-bit DIBs[four]

In encoded mode, the first byte of the pair contains the number of pixels to be drawn using the color indexes in the second byte.

The second byte contains two color indexes, one in its high-order nibble (that is, its low-order four bits) and one in its low-order nibble.

The first of the pixels is drawn using the color specified by the high-order nibble, the second is drawn using the color in the low-order nibble, the third is drawn with the color in the high-order nibble, and so on, until all the pixels specified by the first byte have been drawn.

Also, the first byte of the pair can be set to zero to indicate an escape that denotes an end of line, end of bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair. In encoded mode, the second byte has a value from 00H to 02H.

Absolute ModeAbsolute mode; 4-bit DIBs[four]

In absolute mode, the first byte contains zero, the second byte contains the number of color indexes that follow, and subsequent bytes contain color indexes in their high- and low-order nibbles, one color index for each pixel.

Each run must be aligned on a word boundary.

The end-of-line, end-of-bitmap, and delta escapes valid for BI_RLE8 also apply to BI_RLE4. The following example shows the hexadecimal values of a 4-bit RLE bitmap. Under Expanded Data, the one-digit values represent a color index for a single pixel.

Compressed Data	Expanded Data
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67 00	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	move 5 right and 1 down
04 78	7 8 7 8
00 00	end of line
09 1E	1 E 1 E 1 E 1 E 1
00 01	end of RLE bitmap

RIFF Device-Independent Bitmap File Format

This document describes two types of RIFF Device-Independent Bitmap (RDIB) format:

- A simple RDIB consisting of a DIB file enclosed in a RIFF chunk.
- An extended RDIB that allows the creation of more complex bitmaps

To ensure that the maximum number of programs will accept an RDIB file, programs that adopt the extended RDIB format should also accept simple RDIB files. Both formats are described in the following sections.

Simple RDIB Format

The simple RDIB format consists of a Windows 3.0 or Presentation Manager 1.2 DIB enclosed in a RIFF chunk. Enclosing the DIB in a RIFF chunk allows the file to be consistently identified; for example, an INFO list can be included in the file.

The simple RDIB form is defined as follows, using the standard RIFF form definition notation: DIB files; RDIB format RIFF files; RDIB form RDIB file format File formats; RIFF Device-Independent Bitmap (RDIB) RDIB files; description

```
<RDIB-form> Y RIFF ( 'RDIB data( <DIB-data> ) )
```

The <DIB-data> format is defined in Device Independent Bitmap File Format, earlier in this chapter.

Extended RDIB Format

The extended RDIB format, designed to incorporate enhancements such as compression, is defined as follows:

```
<RDIB-form> Y
```

```

        RIFF('RDIB'
        <bmhd-ck>           // Bitmap
header chunk
        [ <pal-file> |     //
Internal palette chunk
        <XPAL-ck> ]       //
External palette chunk
        <bitmap-data> )    // Bitmap
data

```

The <pal-file> chunk can be any of the palette-file formats discussed in Palette File Format, later in this chapter. The <bmhd-ck>, <XPAL-chunk>, and <bitmap-data> are described in the following sections.

Bitmap Header Chunk

The <bmhd-ck> bitmap header chunk is defined as follows:

```

<bmhd-chunk> Y bmhd( struct {
    DWORD dwMemSize;           // If dwPelFormat is
'data', only these
    DWORD dwPelFormat;        // four fields are
present.
    WORD wTransType;
    DWORD dwTransVal;
    DWORD dwHdrSize;          // Fields from dwHdrSize
forward match
    DWORD dwWidth;            // the Windows
BITMAPINFOHEADER
    DWORD dwHeight;           // structure, though some
fields can
    WORD dwPlanes;            // contain new values.
    WORD dwBitCount;
    DWORD dwCompression;
    DWORD dwSizeImage;
    DWORD dwXPelsPerMeter;
    DWORD dwYPelsPerMeter;
    DWORD dwClrUsed;
    DWORD dwClrImportant;
} )

```

If the dwCompression field equals BI_RGB or BI_RLE8 or BI_RLE4, then the extended RDIB has the same bitmap format as a simple RDIB.

Each pixel format defines the orientation, or position of the bitmap origin. Windows bitmaps (identified by a value of data in the dwPelFormat field) have the origin at the bottom left. By default, the other formats have the origin at the top left.

Field	Description
dwMemSize	Equal to the size of the bitmap bits if the bits are uncompressed. For RDIBs with dwPelFormat equal to data, dwMemSize has one of the

following values:

Image Type	Field Value
Non-RLE	Same as dwSizeImage value
8-bit RLE	Size as an uncompressed, 8-bit image
4-bit RLE	Size as an uncompressed, 4-bit image

dwPelFormat Specifies a FOURCC code defining the pixel format of the bitmap data. The bitmap data is stored in a chunk (or chunks) that has the same chunk ID as is contained in dwPelFormat. The compression scheme and pixel depth of the bitmap data are recorded in the dwCompression and dwBitCount fields. The current bitmap data values are as follows:

Value	Bitmap Data Location and Format
'data'	Bitmap data is stored in a data chunk using the format defined for Windows 3.0 device independent bitmaps (DIBs). An application can display the bitmap properly even if the fields after (and including) dwMemSize are ignored.
'palb'	Bitmap data is stored in a palb chunk. The pixel format is one of the Windows 3.0 RGB palettized formats (1 to 8 bpp, depending on the value of the dwBitCount field).
'rgbb'	Bitmap data is stored in a rgbb chunk. Pixel format is packed, unpalettized RGB represented at 16, 24, or 32 bits per pixel. The following shows the ordering of the RGB bits for each pixel-depth value. The first extra bit (if present) is the high-order bit.

dwBitCount	Extra	Red	Green	Blue
15	1	5	5	5
16	0	5	6	5
24	0	8	8	8
32	8	8	8	8

'yuvb' Bitmap data is stored in a yuvb chunk. Pixel format is packed, unpalettized YUV. The exact pixel format is currently undefined. By the time this draft is final, the pixel format

will be defined similarly to the
rgbb definition.

wTransType Specifies the type of transparency representation, if any, used for this image. This is normally used for either image overlay applications, where one image may be visually on top of another, and all pels of the transparency color should not be drawn. Examples include sprites, clip art and motion video overlay. Wherever the transparency color occurs in the picture, the background should be visible. This information is stored with the image, so that multiple images that use the same color map may all have different transparency color. There are 5 different values for the transparency variable. These are:

Value	Result
BITT_NONE (0x0000)	No pels are considered transparent in this image.
BITT_MAPIND EX (0x0001)	One of the color map/palette entries should be considered the transparency color. All instances of this pel should NOT be drawn, and the existing background should be allowed to show through.
BITT_SINGLE COLOR (0x0002)	A single RGB or YUV value is considered transparent and should not be drawn.
BITT_BITPLA NE (0x0003)	An individual bit plane is considered transparent, and all pels that have that bit or bits on should not be drawn.
BITT_MULTIL EVEL (0x0004)	A set of bits indicate multiple levels of transparency or opacity. This is usually used with 32-bit RGB, where the high 8 bits indicate transparency.

dwTransVal These bytes allow the image definition to indicate the exact information about the transparent color. The information is dependent on the value of the wTransType as follows:

wTransType	dwTransVal Contents
BITT_NONE	Not used.
BITT_MAPIND	Specifies a palette index,
EX	either 0 through 16 or 0

through 255, depending on the number of palette entries.

BITT_SINGLE COLOR Specifies an RGB or YUV value (2 to 4 bytes in size, depending on the pixel format specified by dwPixelFormat). All pels that match dwTransVal should be considered transparent.

BITT_BITPLANE Specifies a bit mask identifying the bits used to indicate a transparent pel. Any pel that has this set of bits set is totally transparent. This allows multiple colors to be considered transparent. This method works for palettized images; in this case, the value refers to a map entry that is considered transparent.

BITT_MULTILEVEL Specifies bits to use for transparency levels. These bits act as a mask on every pel, and each pel can be matched to the mask to determine the transparency level for the pel.

For example, if dwTransVal has value 0xFF000000, then there are 256 levels of transparency. Each pel can be evaluated against the mask. If the pel has a value FFxxxxxx, then it is fully transparent. If the pel has a value 00xxxxxx, then it is fully visible. If the pel has a value 7Fxxxxxx, then the pel is half visible.

dwHdrSize Specifies the size of the data portion of the <bmhdr> chunk. This is always 40, the size of the BITMAPINFOHEADER structure.

dwWidth Specifies the width of the DIB in pixels.

dwHeight Specifies the height of the DIB in pixels.

wPlanes Specifies the number of planes. This value is normally 1, but it can be 3 or 4 for 24-bit RGB and 32-bit RGB images, respectively. In a multiplane DIB, each color component (for example, red, green, and blue) is

stored as a separate plane, and each plane is stored in a separate bitmap data chunk. For example, in a 3-plane, 24-bit rgb image, the red colors are stored in one rgb chunk, the green colors in a second rgb chunk, and the blue colors in a third rgb chunk.

Allowing the separate RGB planes to be compressed independently can dramatically improve the compression ratio. The wPlanes value must be 1 if dwPixelFormat equals data.

wBitCount

Specifies the number of bits per pixel. If the dwPixelFormat field equals data, this field must contain values compatible with the Windows 3.0 DIB definition.

dwCompression

Specifies the type of compression for a compressed bitmap. It can be one of the following values:

Value	Meaning
BI_NONE (0xFFFF0000)	Specifies that the bitmap is not compressed. Pixel values are not padded to four-byte boundaries.
BI_RGB (0x00000000)	Specifies that the bitmap is an uncompressed, 1-, 4-, 8-, or a 24-bit image. For 24-bit images, the palette is optional. Bitmap bits are represented as defined by Windows 3.0 for BI_RGB DIBs. The dwPixelFormat field must be set to data.
BI_RLE8 (0x00000001)	Specifies a run-length encoded, compressed bitmap (as defined by Windows 3.0 BI_RLE8 DIBs). The palette is required. The dwPixelFormat field must be set to data.
BI_RLE4 (0x00000002)	Specifies a run-length encoded, compressed bitmap (as defined by Windows 3.0 BI_RLE4 DIBs). The palette is required. The dwPixelFormat field must be set to data.
BI_PACK (0xFFFF0001)	Specifies a simple PACKBITS byte compression scheme consisting of one-byte counts followed by byte data, in the form: <count byte n><data byte1><data byte2>...<data byte n> <count byte n><data byte to repeat> The high-order bit of the count byte n is a decision

bit:

n ValueData Representation
 n < 0x80 A run of n+1 non-repeating bytes follows.
 n > 0x80 Data byte is repeated (n-0x80+1) times.
 n = 0x80 Reserved.

BI_TRANS (0xFFFFF002) Specifies transitional compression, using a table of byte transitions or sequences. See Transitional Compression, following this table.
 BI_CCC (0xFFFFF003) Specifies CCC compression, a method involving encoding each 4-by-4 block of the image using two colors. See CCC Compression, following this table.
 BI_JPEGN (0xFFFFF004) To be defined later, when the ISO completes the official specification.

dwSizeImage Specifies the size in bytes of the compressed image.
 dwXPelsPerMeter Specifies the horizontal resolution in pixels per meter of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device. This field is set to zero if unused.
 dwYPelsPerMeter Specifies the vertical resolution in pixels per meter of the target device for the bitmap. This field is set to zero if unused.
 dwClrUsed Specifies the number of palette entries actually used by the bitmap. Possible values follow.

Value	Result
0	Bitmap uses the maximum number of colors corresponding to the value of the wBitCount field.
Nonzero	If the wBitCount is less than 24, dwClrUsed specifies the actual number of colors which the graphics engine or device driver will access. If the wBitCount field is set to 24, dwClrUsed specifies the size of the reference color table used to optimize performance of Windows color palettes.

`dwClrImportant` Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0, then all colors are important.

Transitional Compression

If the `dwCompression` field is set to `BI_TRANS`, the data is transitionally compressed using a table of byte transitions or sequences. Values in the data indicate a table position to start at, and the table provides continuing references to other table positions. Transitional compression applies only to eight-bit data, either from an eight-bit palettized image or from a multi-plane image in which each color component is represented in eight bits.

The table consists of up to 256 16-byte rows at the beginning of the data section of the object. Nibbles (half-bytes) in the data section indicate an offset into a table row, at which location is stored the actual byte value. The actual value then becomes the row applicable to the next data nibble. The transitional encoding scheme is described more fully in a separate IBM document.

In transitional compression, the data section is a two-part compound object having the following items:

- A transition table
- The compressed image data

The transition table consists of an integer indicating the table size in bytes and a table of 16-byte rows. The first byte in each row is a row number and the next 15 are transition values. Rows are in descending sequence. The image is compressed according to the following rules:

- Data is in nibbles (half-bytes) or in nibble-pairs (successive half-bytes which may cross a byte boundary).
- The first byte is a nibble-pair. It is the first byte of the image and also the first row number.
- Following a nibble-pair is a series of transition nibbles (1-15) ended by a terminator (0). Each transition nibble indicates an offset in the current row at which the next byte in the image is found; this value is also the next row number.
- The terminator indicates that the next image byte is not in the table, but instead in the following nibble-pair. This value is also the next row number.

- If the picture has an odd number of nibbles (i.e., it ends in the first half of the last byte), an extra zero nibble is included.

CCC Compression

TBD.

Palette Chunk

A PLT chunk represents a color table and consists of a valid PAL file. The PAL file format is defined in Palette File Format, later in this chapter.

External Palette Chunk

Instead of a PLT chunk, an RDIB may contain an XPLT chunk, which indicates that the bitmap's palette is stored outside the bitmap. The palette might be stored in a separate file or as a separate compound file element. The XPLT chunk indicates the name and location of the external palette chunk and is defined as follows:

```
<XPLT-chunk> Y XPLT(<fccLocation:FOURCC>
<szPaletteName::ZSTR>)
```

The fccLocation contains one of the following FOURCC values specifying the location of the external chunk:

fccLocation Value	Chunk Location
full	Palette is located in an external file, and the szPaletteName value specifies a complete filename with path.
file	Palette is located in an external file, and the szPaletteName value specifies a filename without path.
elem	Palette is located in the same compound file containing the DIB. The szPaletteName value specifies the name of the compound file element.

The szPaletteName consists of a null-terminated string (ZSTR) containing the name of the external chunk containing the palette.

Bitmap Data Chunk

The <bitmap-data> contains bitmap data in the format specified by the biPelFormat field of the <bmhd-chunk>.

MIDI and RIFF MIDI File Formats

The Musical Instrument Digital Interface (MIDI) file format represents a Standard MIDI File, as defined by the MIDI Manufacturers Association. A MIDI file contains commands instructing instruments to play specific notes and perform other operations. File formats;Musical Instrument Digital Interface (MIDI)

The specifications for MIDI and MIDI files can be obtained from the following organization:MIDI files;specifications

International MIDI Association (IMA)
5316 W. 57th Street
Los Angeles, CA 90056
(213) 649-6434.

The RMID format consists of a standard MIDI file enclosed in a RIFF chunk. Enclosing the MIDI file in a RIFF chunk allows the file to be consistently identified; for example, an INFO list can be included in the file. RIFF files;RMID formatFile formats;RIFF Musical Instrument Digital Interface (MIDI)RMID file formatMIDI files;RMID format

The RMID form is defined as follows, using the standard RIFF form definition:

```
<RMID-form>  Ÿ  RIFF ('RMID' data( <MIDI-data> ))
```

The <MIDI-data> is equivalent to a Standard MIDI File.

Palette File Format

The Palette (PAL) File Format represents a logical palette, which is a collection of colors represented as RGB values. There are two types of PAL formats:

- A simple PAL format
- An extended PAL format

Simple PAL Format

The simple PAL format is defined as follows:Palette files;PAL file formatPAL file formatFile formats;Palette (PAL)

```
RIFF('PAL' data( <palette:LOGPALETTE> ))
```

LOGPALETTE is the Windows 3.0 logical palette structure, defined as follows:Palette files;logical palette structureLOGPALETTE data structure;with palette filesLogical palette structure

```
typedef struct tagLOGPALETTE {
    WORD        palVersion;
```

```

        WORD            palNumEntries;
    PALETTEENTRY    palPalEntry[];
} LOGPALETTE;

```

The LOGPALETTE structure fields are as follows:

Field	Description
palVersion	Specifies the Windows version number for the structure.
palNumEntries	Specifies the number of palette color entries.
palPalEntry[]	Specifies an array of PALETTEENTRY data structures that define the color and usage of each entry in the logical palette.

The colors in the palette entry table should appear in order of importance. This is because entries earlier in the logical palette are most likely to be placed in the system palette.

The PALETTEENTRY data structure specifies the color and usage of an entry in a logical color palette. The structure is defined as follows: PALETTEENTRY data structure; with palette files; setting intensity; RGB values; palette intensity

```

typedef struct tagPALETTEENTRY {
    BYTE    peRed;
    BYTE    peGreen;
    BYTE    peBlue;
    BYTE    peFlags;
} PALETTEENTRY;

```

The PALETTEENTRY structure fields are as follows:

Field	Description
peRed	Specifies the intensity of red for the palette entry color.
peGreen	Specifies the intensity of green for the palette entry color.
peBlue	Specifies the intensity of blue for the palette entry color.
peFlags	Specifies how the palette entry is to be used.

Extended PAL Format

The extended PAL format includes the following:

- A palette-header chunk
- A data chunk containing an RGB palette (consisting of a LOGPALETTE structure) or some other palette type, including YUV and XYZ palettes.

For an RGB palette, the extended PAL format is represented as follows:

```
RIFF('PAL' plth( <palette-header> ) data( <LOGPALETTE-
data> ))
```

For a YUV palette, the extended PAL format is represented as follows:

```
RIFF('PAL' plth( <palette-header> ) yuvp( <YUV-
LOGPALETTE-data> ))
```

Both the <LOGPALETTE-data> and <YUV-LOGPALETTE-data> use the Windows 3.0 LOGPALETTE structure, described in Simple PAL Format, earlier in this section. The <YUV-LOGPALETTE-data> contains YUV values instead of RGB values.

The plth chunk is defined as follows:

```
<plth-ck> Y PLT( struct {
    DWORD dwMapType;
    WORD wWhite;          // Fields from this point
on are
    WORD wBlack;          // optional. If they are
included
    WORD wBorder;         // but not used, set them
to 0xFFFF.
    WORD wRegisteredMap;
    WORD wCustomBase;     // If an application
encounters a
    WORD wCustomCnt;      // 'PLT' chunk smaller
than shown
    WORD wRsvBase;        // here, it should treat
the missing
    WORD wRsvCount;       // fields as unused.
    WORD wArtBase;
    WORD wArtCnt;
    WORD wNumIntense;
} )
```

The structure fields are described in the following:

Field	Description
dwMapType	FOURCC code specifying the type of palette. Currently, the following palette types are identified:
	Code Description
data	Specifies an RGB palette. Data chunk contains a LOGPALETTE structure.
yuvp	Specifies a YUV palette. Data chunk contains a YUV palette.
xyzp	Specifies an XYZ palette. Data chunk contains a XYZ palette.

wWhite Specify palette-map indices corresponding to the closest value of white and black. These identify the pair of colors with the best contrast for use in cursors, calibration, etc. These values are usually changed if the palette changes. Ignore these fields if they contain 0xFFFF.
 wBlack
 wBorder Specifies the index of the palette entry to be used for any display-border regions, if supported by the display device. Ignore this field if it contains 0xFFFF.
 wRegisteredMap Specifies how many palette entries correspond to a registered color map. Registered entries are stored at the front of the palette. Ignore this field if it contains 0xFFFF. Registered map entries are always stored at the beginning of the palette, so wRegisteredMap also indicates the index of the first custom color in the palette. Registered color maps include predefined palettes for general use, forest/nature, or seashores. Currently defined values are the following:

Value	Description
PAL_UNREGISTERED (0xFFFF)	Color map does not contain colors from a registered color map.
PAL_VGA (0x0000)	Color map contains the standard 16 VGA colors.
PAL_AVC198 (0x0001)	Standard AVC 198-entry palette.

wCustomBase Specifies the index of the first custom color of the palette. The beginning of the palette contains the entries of the registered map, so wCustomBase also indicates the number of entries in the registered palette. Map entries starting with wCustomBase comprise additional custom colors used in the bitmap. Ignore this value if wRegisteredPalette is PAL_UNREGISTERED, or if wCustomBase contains 0xFFFF.
 wCustomCnt Specifies the number of custom colors in the palette. Ignore this value if wRegisteredPalette is PAL_UNREGISTERED, or if this field contains 0xFFFF.
 wRsvBase Specifies the index of the first reserved color of the palette. Reserved colors are those reserved for

	menus, text, and other screen elements. Reserved colors must be stored contiguously. Ignore this field if it contains 0xFFFF.
wRsvCnt	Specifies the number of reserved entries. Ignore this field if it contains 0xFFFF.
wArtBase	Specifies the index of the first art color of the palette. Art colors are colors used for text and drawing. Art colors consist of a number of hues, each of which has multiple intensities. The various intensities are used for anti-aliasing, a method of using different shades of a color to improve the quality of images displayed on low-resolution devices. For example, if the first art color is red anti-aliased to black with three intensities, the first three entries in the palette would be dark red, medium red, and bright red. The art colors constitute an array, and all hues have the same number of intensities. The user can set both the number of hues and the number of intensities. Ignore these fields if they contain 0xFFFF.
wArtCnt	Specifies the number of art colors. Ignore this field if it contains 0xFFFF.
wNumIntense	Specifies the number of palette entries reserved for the anti-aliased levels of a given art color. This field must be present if wArtBase is present. Ignore this field if it contains 0xFFFF.

Rich Text Format (RTF)

The Rich Text Format (RTF) is a standard method of encoding formatted text and graphics using only 7-bit ASCII characters. Formatting includes different font sizes, faces, and styles, as well as paragraph alignment, justification, and tab control. Rich Text Format file format

RTF is described in the Microsoft Word Technical Reference: For Windows and OS/2, published by Microsoft Press.

Waveform Audio File Format (WAVE)

This section describes the Waveform format, which is

<http://www.kk.iij4u.or.jp/~kondo/wave/mpidata.txt>

3/6/2006

used to represent digitized sound. RIFF
forms; WAVE waveform files; WAVE format file
formats; Waveform Audio (WAVE)

The WAVE form is defined as follows. Programs must expect (and ignore) any unknown chunks encountered, as with all RIFF forms. However, <fmt-ck> must always occur before <wave-data>, and both of these chunks are mandatory in a WAVE file.

```
<WAVE-form> Ý
    RIFF( 'WAVE'
        <fmt-ck>                // Format
        [<fact-ck>]             // Fact chunk
        [<cue-ck>]              // Cue points
        [<playlist-ck>]         // Playlist
        [<assoc-data-list>]     //
    Associated data list
        <wave-data> )          // Wave data
```

The WAVE chunks are described in the following sections.

WAVE Format Chunk

The WAVE format chunk <fmt-ck> specifies the format of the <wave-data>. The <fmt-ck> is defined as follows:

```
<fmt-ck> Ý    fmt( <common-fields>
                  <format-specific-fields> )

<common-fields> Ý
    struct
    {
        WORD wFormatTag;                // Format
    category
        WORD wChannels;                // Number of
    channels
        DWORD dwSamplesPerSec;         //
    Sampling rate
        DWORD dwAvgBytesPerSec;        // For
    buffer estimation
        WORD wBlockAlign;              // Data block size
    }
```

The fields in the <common-fields> chunk are as follows:

Field	Description
wFormatTag	A number indicating the WAVE format category of the file. The content of the <format-specific-fields> portion of the fmt chunk, and the interpretation of the waveform data, depend on this value. You must register any new WAVE format categories. See Registering Multimedia Formats in Chapter 1, Overview of Multimedia

Specifications, for information on registering WAVE format categories. Wave Format Categories, following this section, lists the currently defined WAVE format categories.

wChannels The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.

dwSamplesPerSec The sampling rate (in samples per second) at which each channel should be played.

dwAvgBytesPerSec The average number of bytes per second at which the waveform data should be transferred. Playback software can estimate the buffer size using this value.

wBlockAlign The block alignment (in bytes) of the waveform data. Playback software needs to process a multiple of wBlockAlign bytes of data at a time, so the value of wBlockAlign can be used for buffer alignment.

The <format-specific-fields> consists of zero or more bytes of parameters. Which parameters occur depends on the WAVE format category-see the following section for details. Playback software should be written to allow for (and ignore) any unknown <format-specific-fields> parameters that occur at the end of this field. Samples, setting bits per sample

WAVE Format Categories

The format category of a WAVE file is specified by the value of the wFormatTag field of the fmt chunk. The representation of data in <wave-data>, and the content of the <format-specific-fields> of the fmt chunk, depend on the format category. Waveform files; format categories

The currently defined open non-proprietary WAVE format categories are as follows:

wFormatTag Value	Format Category
WAVE_FORMAT_PCM (0x0001)	Microsoft Pulse Code Modulation (PCM) format

The following are the registered proprietary WAVE format categories:

wFormatTag Value	Format Category
IBM_FORMAT_MULAW (0x0101)	IBM mu-law format
IBM_FORMAT_ALAW (0x0102)	IBM a-law format

IBM_FORMAT_ADPCM
(0x0103)

IBM AVC Adaptive
Differential Pulse Code
Modulation format

The following sections describe the Microsoft
WAVE_FORMAT_PCM format.

Pulse Code Modulation (PCM) Format

If the wFormatTag field of the <fmt-ck> is set to
WAVE_FORMAT_PCM, then the waveform data consists of
samples represented in pulse code modulation (PCM)
format. For PCM waveform data, the <format-specific-
fields> is defined as follows:

```
<PCM-format-specific> {
    struct
    {
        WORD wBitsPerSample; // Sample size
    }
}
```

The wBitsPerSample field specifies the number of bits
of data used to represent each sample of each channel.
If there are multiple channels, the sample size is the
same for each channel.

For PCM data, the wAvgBytesPerSec field of the fmt
chunk should be equal to the following formula rounded
up to the next whole number:

The wBlockAlign field should be equal to the following
formula, rounded to the next whole number: Data
blocks; aligning waveform

Data Packing for PCM WAVE Files

In a single-channel WAVE file, samples are stored
consecutively. For stereo WAVE files, channel 0
represents the left channel, and channel 1 represents
the right channel. The speaker position mapping for
more than two channels is currently undefined. In
multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for a 8-
bit mono and stereo WAVE files:

```
Sample 1    Sample 2    Sample 3    Sample 4
Channel 0    Channel 0    Channel 0    Channel 0
Data Packing for 8-Bit Mono PCM
```

```
Sample 1          Sample 2
Channel 0    Channel 1    Channel 0    Channel 0
(left)      (right)      (left)      (right)
Data Packing for 8-Bit Stereo PCM
```

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:

```

      Sample 1          Sample 2
Channel 0  Channel 0  Channel 0  Channel 0
low-order  high-order  low-order  high-order
  byte      byte      byte      byte
      Data Packing for 16-Bit Mono PCM

```

```

              Sample 1
Channel 0    Channel 0  Channel 1  Channel 1
(left)      (left)    (right)   (right)
low-order   high-order low-order   high-order
  byte      byte      byte      byte
      Data Packing for 16-Bit Stereo PCM

```

Data Format of the Samples -

Each sample is contained in an integer *i*. The size of *i* is the smallest number of bytes required to contain the specified sample size. The least significant byte is stored first. The bits that represent the sample amplitude are stored in the most significant bits of *i*, and the remaining bits are set to zero.

For example, if the sample size (recorded in *nBitsPerSample*) is 12 bits, then each sample is stored in a two-byte integer. The least significant four bits of the first (least significant) byte is set to zero. Amplitude, setting waveform

The data format and maximum and minimum values for PCM waveform samples of various sizes are as follows:

Sample Size	Data Format	Maximum Value	Minimum Value
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer <i>i</i>	Largest positive value of <i>i</i>	Most negative value of <i>i</i>

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format	Maximum Value	Minimum Value	Midpoint Value
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767 (0x7FFF)	-32768 (-0x8000)	0

Examples of PCM WAVE Files

Example of a PCM WAVE file with 11.025 kHz sampling rate, mono, 8 bits per sample:

```
RIFF( 'WAVE'      fmt(1, 1, 11025, 11025, 1, 8)
      data( <wave-data> ) )
```

Example of a PCM WAVE file with 22.05 kHz sampling rate, stereo, 8 bits per sample:

```
RIFF( 'WAVE'      fmt(1, 2, 22050, 44100, 2, 8)
      data( <wave-data> ) )
```

Example of a PCM WAVE file with 44.1 kHz sampling rate, mono, 20 bits per sample:

```
RIFF( 'WAVE'      INFO(INAM("O Canada"Z))
      fmt(1, 1, 44100, 132300, 3, 20)
      data( <wave-data> ) )
```

Storage of WAVE Data

The <wave-data> contains the waveform data. It is defined as follows:

```
<wave-data> Ý    [ <data-ck> | <data-list> ]

<data-ck> Ý      data( <wave-data> )

<wave-list> Ý    LIST( 'wavl' {      <data-ck> |
// Wave samples
                                <silence-ck> }... ) //
Silence

<silence-ck> Ý    slnt( <dwSamples:DWORD> )
// Count of
                                // silent
samples
```

Note: The slnt chunk represents silence, not necessarily a repeated zero volume or baseline sample. In 16-bit PCM data, if the last sample value played before the silence section is a 10000, then if data is still output to the D to A converter, it must maintain the 10000 value. If a zero value is used, a click may be heard at the start and end of the silence section. If play begins at a silence section, then a zero value might be used since no other information is available. A click might be created if the data following the silent section starts with a nonzero value.

FACT Chunk

The <fact-ck> fact chunk stores important information about the contents of the WAVE file. This chunk is defined as follows:

```
<fact-ck> Ý      fact( <dwFileSize:DWORD> )
// Number of samples
```

The fact chunk is required if the waveform data is contained in a wavl LIST chunk and for all compressed audio formats. The chunk is not required for PCM files using the data chunk format.

The "fact" chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the <dwFileSize> field. Applications can use the chunk size field to determine which fields are present.

Cue-Points Chunk

The <cue-ck> cue-points chunk identifies a series of positions in the waveform data stream. The <cue-ck> is defined as follows:

```
<cue-ck> Y      cue(  <dwCuePoints:DWORD>      //
Count of cue points
                    <cue-point>... )          // Cue-
point table

<cue-point> Y    struct {
                    DWORD  dwName;
                    DWORD  dwPosition;
                    FOURCC fccChunk;
                    DWORD  dwChunkStart;
                    DWORD  dwBlockStart;
                    DWORD  dwSampleOffset;
                }
```

The <cue-point> fields are as follows:

Field	Description
dwName	Specifies the cue point name. Each <cue-point> record must have a unique dwName field.
dwPosition	Specifies the sample position of the cue point. This is the sequential sample number within the play order. See Playlist Chunk, later in this document, for a discussion of the play order.
fccChunk	Specifies the name or chunk ID of the chunk containing the cue point.
dwChunkStart	Specifies the file position of the start of the chunk containing the cue point. This is a byte offset relative to the start of the data section of the wavl LIST chunk.
dwBlockStart	Specifies the file position of the start of the block containing the position. This is a byte offset relative to the start of the data section of the wavl LIST chunk.
dwSampleOffset	Specifies the sample offset of the cue point relative to the start of the block.

Examples of File Position Values

The following table describes the <cue-point> field values for a WAVE file containing multiple data and slnt chunks enclosed in a wavl LIST chunk:

Cue Point Location	Field	Value
In a slnt chunk	fccChunk	FOURCC value slnt.
	dwChunkStart	File position of the slnt chunk relative to the start of the data section in the wavl LIST chunk.
	dwBlockStart	File position of the data section of the slnt chunk relative to the start of the data section of the wavl LIST chunk.
	dwSampleOffset	Sample position of the cue point relative to the start of the slnt chunk.
In a PCM data chunk	fccChunk	FOURCC value data.
	dwChunkStart	File position of the data chunk relative to the start of the data section in the wavl LIST chunk.
	dwBlockStart	File position of the cue point relative to the start of the data section of the wavl LIST chunk.
	dwSampleOffset	Zero value.
In a compressed data chunk	fccChunk	FOURCC value data.
	dwChunkStart	File position of the start of the data chunk relative to the start of the data section of the wavl LIST chunk.
	dwBlockStart	File position of the enclosing block relative to the start of the data section of the wavl LIST chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

The following table describes the <cue-point> field

values for a WAVE file containing a single data chunk:

Cue Point Location	Field	Value
Within PCM data	fccChunk	FOURCC value data.
	dwChunkStart	Zero value.
	dwBlockStart	Zero value.
	dwSampleOffset	Sample position of the cue point relative to the start of the data chunk.
In a compressed data chunk	fccChunk	FOURCC value data.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the enclosing block relative to the start of the data chunk. The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

Playlist Chunk

The <playlist-ck> playlist chunk specifies a play order for a series of cue points. The <playlist-ck> is defined as follows:

```
<playlist-ck> Y    plst(
                    <dwSegments:DWORD>    // Count of
play segments      <play-segment>... )    // Play-segment
table

<play-segment> Y    struct {
                    DWORD dwName;
                    DWORD dwLength;
                    DWORD dwLoops;
                    }
```

The <play-segment> fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwLength	Specifies the length of the section in samples.
dwLoops	Specifies the number of times to play the section.

Associated Data Chunk

The <assoc-data-list> associated data list provides the ability to attach information like labels to sections of the waveform data stream. The <assoc-data-list> is defined as follows:

```

<assoc-data-list> Y      LIST(      'adtl'
                           <labl-ck>      // Label
                           <note-ck>      // Note
                           <ltxt-ck>      // Text
with data length
                           <file-ck> )      //
Media file
<labl-ck> Y              labl( <dwName:DWORD>
                              <data:ZSTR> )
<note-ck> Y              note( <dwName:DWORD>
                              <data:ZSTR> )
<ltxt-ck> Y              ltxt( <dwName:DWORD>
                              <dwSampleLength:DWORD>
                              <dwPurpose:DWORD>
                              <wCountry:WORD>
                              <wLanguage:WORD>
                              <wDialect:WORD>
                              <wCodePage:WORD>
                              <data:BYTE>... )
<file-ck> Y              file( <dwName:DWORD>
                              <dwMedType:DWORD>
                              <fileData:BYTE>... )

```

Label and Note Information

The labl and note chunks have similar fields. The labl chunk contains a label, or title, to associate with a cue point. The note chunk contains comment text for a cue point. The fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
data	Specifies a NULL-terminated string containing a text label (for the labl chunk) or comment text (for the note chunk).

Text with Data Length Information

The ltxt chunk contains text that is associated with a data segment of specific length. The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwSampleLength	Specifies the number of samples in the segment of waveform data.
dwPurpose	Specifies the type or purpose of the text. For example, dwPurpose can specify a FOURCC code like scrp for script text or capt for close-caption text.
wCountry	Specifies the country code for the text. See Country Codes in Chapter 2, Resource Interchange File Format, for a current list of country codes.
wLanguage, wDialect	Specify the language and dialect codes for the text. See Language and Dialect Codes in Chapter 2, Resource Interchange File Format, for a current list of language and dialect codes.
wCodePage	Specifies the code page for the text.

Embedded File Information

The file chunk contains information described in other file formats (for example, an RDIB file or an ASCII text file). The chunk fields are as follows:

Field	Description
dwName	Specifies the cue point name. This value must match one of the names listed in the <cue-ck> cue-point table.
dwMedType	Specifies the file type contained in the fileData field. If the fileData section contains a RIFF form, the dwMedType field is the same as the RIFF form type for the file.
fileData	This field can contain a zero value. Contains the media file.

Chapter 4 Media Control Interface

The Media Control Interface (MCI) is a high-level command control interface to multimedia devices and resource files. MCI provides applications with device-independent capabilities for controlling audio and visual peripherals. Your application can use MCI to control any multimedia device, including audio playback and recording, as well as videodisc and videotape players.

MCI provides a standard command set for playing and recording multimedia devices and resource files. Developers creating multimedia applications are encouraged to use this high-level command interface rather than the low-level functions specific to each platform. The MCI command set acts as a platform-independent layer that sits between multimedia applications and the underlying system software.

The command set is extensible in two ways:

- Developers can incorporate new multimedia devices and file formats in the MCI command set by creating new MCI drivers to interpret the commands.
- New commands and command options can be added to support special features or functions required by new multimedia devices or file formats.

MCI Command Strings

Using MCI, an application can control multimedia devices using simple command strings like open, play, and close. The MCI commands provide a generic interface to different multimedia devices, reducing the number of commands a developer needs to learn. A multimedia application might even accept MCI commands from an end user and pass them unchanged to the MCI driver, which parses the command and performs the appropriate action.

A set of basic commands is supported by all MCI devices. Developers can also define MCI commands and command options specific to a particular multimedia device or file format. These device-specific commands and command options are needed only when the basic command set does not support a feature specific to the device or file format.

Example of MCI Command Use

The following example shows a series of MCI commands that play track 6 of an audio compact disc:

```
open cdaudio
set cdaudio time format tmsf
play cdaudio from 6 to 7
close cdaudio
```

The next example shows a similar series of MCI commands that play the first 10,000 samples of a waveform audio file:

```
open c:\mmdata\purplefi.wav type waveaudio alias finch
set finch time format samples
play finch from 1 to 10000 wait
close finch
```

Notice the following:

- The same basic commands (open, play, and close) are used with both devices.
- The open command for the waveaudio device includes a filename specification. The waveaudio device is a compound device (one associated with a media element), while the cdaudio device is a simple device (one without an associated media element).
- The set commands both specify time formats, but the time format options for the cdaudio device are different from those used with the waveaudio device.
- The parameters used with the from and to flags are appropriate to the respective device. For the cdaudio device, the parameters specify a range of tracks; for the waveaudio device, the parameters specify a range of samples.

Categories of MCI Command StringsMedia Control Interface;commands defined

MCI command strings divide into the following categories:

- System commands are interpreted directly by MCI rather than being relayed to a device.
- Required commands are recognized by all MCI devices. If a device does not support a required command, it can return unsupported function in response to the message. Media Control Interface;required commands@defined
- Basic commands are optional commands. If a device uses a basic command, it must respond to all options for that command. If a device does not use a basic command, it can return unrecognized command in response to the message. Media Control Interface;basic commands@defined
- Extended commands are specific to a device type or device class; for example, videodisc players. These commands contain both unique commands and extensions to the required and basic commands. <\$IMedia Control Interface;extended commands@defined

Command Syntax Conventions

This chapter uses the following documentation conventions:

Conventio Description

n

bold MCI command or flag keyword.

italics Command parameter to be replaced with a valid string, number, or rectangle specification.

quotes Parameter text to be typed exactly as shown.

[Optional flags or parameters

brackets

]

Media Control Interface; commands definedSystem Commands

The following list summarizes the system commands. MCI supports these commands directly rather than passing them to MCI devices. Media Control Interface; system messages@defined

Message	Description
sound	Play system sounds defined in a system setup file.
sysinfo	Returns information about MCI devices.

Required Commands

The following list summarizes the required commands. All devices recognize these messages. If a device does not support a required command, it can return unsupported function in response to the message.

Message	Description
capabilit	Obtains the capabilities of a device.
y	
close	Closes the device.
info	Obtains textual information from a device.
open	Initializes the device.
status	Returns various status information from the device.

Basic Commands

The following list summarizes the basic commands. MCI devices are not required to recognize these commands. If the device does not recognize a basic command, it can return unrecognized command in response to the message. Media Control Interface; basic messages@defined

Message	Description
load	Recalls data from a disk file.
pause	Stops playing.
play	Starts transmitting output data.
record	Starts recording input data.
resume	Resumes playing or recording from a paused state.
save	Saves data to a disk file.

seek	Seeks forward or backward.
set	Sets the operating state of the device.
status	Obtains status information about the device. (The flags for this command supplement the flags for the command in the required command group.)
stop	Stops playing.

Extended Commands

MCI devices can have additional commands or extend the definition of the required and basic commands. While some extended commands only apply to a specific device driver most of them apply to all devices of a particular type. For example, the MIDI sequencer command set extends the set command to add time formats needed by MIDI sequencers. You can find descriptions of extended commands in the command tables in this chapter.

Extended Commands Reserved for Future Use

The following commands can be defined as extended commands. With the exception of the delete command, they are not currently defined for any MCI devices.

Message	Description
copy	Copies data to the Clipboard. Parameters and flags for this message vary according to the selected device.
cut	Moves data from the MCI element to the Clipboard. Parameters and flags for this message vary according to the selected device.
delete	Removes data from the MCI element. Parameters and flags for this message vary according to the selected device.

Creating a Command String

There are three components associated with each command string: the command, the name or ID of the device receiving the command, and the command arguments. A command string has the following form: Media Control Interface;command string defined

```
command    device_name    arguments
```

These components contain the following information:

- The command includes a command from the system, required, basic, or extended command set. Examples of commands include open, close, and play.

- The device_name designates the target of the command. MCI accepts the names of MCI device types and names of media elements for the device_name. An example of a device name is cdAudio.
- The arguments specify the flags and parameters used by the command. Flags are key words recognized by the MCI command, and parameters are variables associated with the MCI command or flag. Parameters specify variable data values such as filenames, track or frame numbers, or speed values. You can use the following data types for the parameters in a string command:
 - Strings-String data types can be delimited by leading and trailing white space or by matching quotation marks. If MCI encounters a single (unmatched) quotation mark, it ignores the quotation mark. To embed a quote in string, use two quotes (""). To specify an empty string, you can use double quotes ("") for the string.
 - Signed long integers-Signed long integer data types are delimited by leading and trailing white space. Unless otherwise specified, integers can be positive or negative. If using negative integers, do not embed white space between the negative sign and the first digit.
 - Rectangle-Rectangle data types are an ordered list of four signed integer values. White space delimits this data type as well as separates each integer in the list.

For example, the play command uses the arguments from position to position to specify starting and ending points for the playback. The from and to arguments are flags, and the two position values are parameters.

For example, the following command string instructs the CD audio player cdaudio to play from the start of the waveform to position 500:

```
play cdaudio from 0 to 500
```

Unspecified command arguments assume a default value. For example, if the flag from was unspecified in the previous example, the audio player would start playing at the current position.

About MCI Device Types

Your application identifies an MCI device by specifying an MCI device type. A device type indicates the physical type of device. The following table lists the currently defined MCI device types:

Device Type	Description
-------------	-------------

cdaudio1	CD audio player
dat	Digital audio tape player
digitalvideo	Digital video in a window (not GDI based)
other	Undefined MCI device
scanner	Image scanner
sequencer1	MIDI sequencer
vcr	Videotape recorder or player
videodisc1	Videodisc player
waveaudio1	Audio device that plays digitized waveform files

1An extended command set is provided for these devices.

If you have a particular device type installed more than once, the device type names in the system setup file have integers appended to them. This creates unique names for each MCI device type entry. For example, if the cdaudio device type is installed twice, the names cdaudio1 and cdaudio2 are used to create unique names for each occurrence of the device type. Each name usually refers to a different CD audio player in the system. Device types;drivers

Using MCI Command Strings

The tables at the end of this chapter describe command strings for the MCI devices. The following sections describe commonly used command strings.

Opening a DeviceMedia Control Interface;opening a device

Before using a device, you must initialize it with the open command. The number of devices you can have open depends on the amount of available memory. The open command has the following syntax:

```
open device_name [shareable] [type device_type] [ alias
alias ]
```

The parameters for the open command are:

Parameters	Description
device_name	Specifies the destination device or MCI element name (filename).
shareable	Allows applications to share a common device or device element.
type	Specifies the device when the device_name refers to an MCI element.
device_type	
alias alias	Specifies an alternate name for the device.

MCI classifies device drivers as compound and simple. Compound device drivers use a device element-a media

element associated with a device-during operation. For most compound device drivers, the device element is the source or destination data file. For file elements, the element name references a file and its path.

Simple device drivers do not require a device element for playback. For example, compact disc audio device drivers are simple device drivers.

Opening Simple Devices

Simple devices require only the device_name for operation. You don't need to provide any additional information (such as a name of a data file) to open these devices. For these devices, substitute the name of a device type obtained from the system setup file. For example, you can open a videodisc device with the following command:

```
open videodisc1
```

Opening Compound Devices

There are three ways to open a compound device:

- By specifying just the device type
- By specifying both the element name and the device type
- By specifying just the element name

To determine the capabilities of a device, you can open a device by specifying only the device type. When opened this way, most compound devices will let you determine their capabilities and close them. For example, you can open the sequencer with the following command:

```
open sequencer
```

To associate a device element with a particular device, you must specify the element name and device type. In the open command, substitute the element name for the device_name, add the type flag, and substitute the name of the device you want to use for device_type. This combination lets your application specify the MCI device it needs to use. For example, you can open a device element of the waveaudio device with the following command:

```
open right.wav type waveaudio
```

To associate a default MCI device with a device element, you can specify just an element name. In this case, MCI uses the filename extension of the element name to select the device type.

Using the Shareable Flag

The shareable flag lets multiple applications or tasks concurrently access the same device (or element) and device instance.

If your application opens a device or device element without the shareable flag, no other application can access it simultaneously. If your application opens a device or device element as shareable, other applications can also access it by also opening it as shareable. The shared device or device element gives each application the ability to change the parameters governing the operating state of the device or device element. Each time that a device or device element is opened as shareable, a unique device ID is returned (even though the device IDs refer to the same instance)

If you make a device or device element shareable, your application should not make any assumptions about the state of a device. When working with shared devices, your application might need to compensate for changes made by other applications using the same services.

If a device can service only one application or task it will fail an open with the shareable flag.

While most compound device elements are not shareable, you can open multiple elements (where each element is unique), or you can open a single element multiple times. If you open a single file element multiple times, MCI creates an independent instance for each open device. Each file element opened within a task must have a unique name. The alias flag described in the next section lets you use a unique name for each element.

Using the Alias Flag

The alias flag specifies an alternate name for the given device. The alias provides a shorthand notation for compound devices with lengthy pathnames. If your application creates a device alias, it must use the alias rather than the device name for all subsequent references.

Opening New Device Elements

To create a new device element for a task such as capturing a sound using waveform recording, specify new as a device_name. MCI does not save a new file element until you save it with the save command. When creating a new file, you must include a device alias with the open command. The following commands open a new waveaudio device element, start and stop recording, save the file element, and close the device element:

```

open new type waveaudio alias capture
record capture
stop capture
save capture orca.wav
close capture

```

Closing a DeviceMedia Control Interface;closing a device

The close command releases access to a device or device element. To help MCI manage the devices, your application must explicitly close each device or device element when it is finished with it.

Shortcuts and Variations for MCI Commands

The MCI string interface lets you use several shortcuts when working with MCI devices.

Using All as a Device Name

You can specify all as a device_name for any command that does not return information. When you specify all, the command is sent to all devices opened by your application. For example, close all closes all open devices and play all starts playing all devices opened by the task. Because MCI sends the commands to each device, there is a delay between when the first device receives the command and when the last device receives the command.

Combining the Device Type and Device Element Name

You can eliminate the type flag in the open command if you combine the device type with the device element name. MCI recognizes this combination when you use the following syntax:

```
device_type!element_name
```

The exclamation mark separates the device type from the element name. The following example opens the right.wav element with the waveaudio device:

```
open waveaudio!right.wav
```

Automatic Open

If MCI cannot identify the device_name as an already open device, MCI tries to automatically open the specified device. Automatic open does not let your application specify the type flag. If the device type is not supplied, MCI determines the device type from the element (filename) extensions listed in the system setup file. If you want to use a specific device, you can combine the device type name with the device element name using the exclamation mark.

Only the command-string interface supports automatic open. Automatic open will fail for device-specific commands. For example, a command to unlock the front panel of a videodisc player will fail an automatic open because this capability is specific to the particular videodisc player.

A device that was opened using the automatic open feature will not respond to a command that uses all as a device name.

Automatic Close

MCI automatically closes any device automatically opened using the command-string interface. MCI closes a device when the command completes, when you abort the command, when you request notification with a subsequent command, or when MCI detects a failure. Automatic open/close

Using Wait and Notify Flags

Normally, MCI commands return to the user immediately, even if it takes several minutes to complete the action initiated by the command. For example, after a VCR device receives a rewind command, it returns before the tape has finished rewinding. You can use either of the following required MCI flags to modify this default behavior:

Flag	Description
notify	Directs the device to send an MM_MCINOTIFY message to a window when the requested action is complete.
wait	Directs the device to wait until the requested action is complete before returning to the application.

Using the Notify FlagMedia Control Interface;callbacks@sending messages to

The notify flag directs the device to post an MM_MCINOTIFY message when the device completes an action. Your application must have a window procedure to process the MM_MCINOTIFY message for notification to have any effect. While the results of a notification are application-dependent, the application's window procedure can act upon four possible conditions associated with the notify message:

- Notification will occur when the notification conditions are satisfied.
- Notification can be superseded.

- Notification can be aborted.
- Notification can fail.

A successful notification occurs when the conditions required for initiating the callback are satisfied and the command completed without interruption.

A notification is superseded when the device has a notification pending and you send it another notify request. When a notification is superseded, MCI resets the callback conditions to correspond to the notify request of the new command.

A notification is aborted when you send a new command that prevents the callback conditions set by a previous command from being satisfied. For example, sending the stop command cancels a notification pending for the Splay to 500 command. If your command interrupts a command that has a notification pending, and your command also requests notification, MCI will abort the first notification immediately and respond to the second notification normally.

A notification fails if a device error occurs while a device is executing the MCI command. For example, MCI posts this message when a hardware error occurs during a play command.

Obtaining Information From MCI Devices

Every device responds to the capability, status, and info commands. These commands obtain information about the device. For example, your application can determine if a videodisc requires a device element using the following command:

```
capability videodisc compound file
```

For most videodisc devices, this example would return false. The flags listed for the required and basic commands provide a minimum amount of information about a device. Many devices supplement the required and basic flags with extended flags to provide additional information about the device.

When you request information with the capability, status, or info command, the argument list can contain only one flag requesting information. The string interface can only return one string or value in response to a capability, status, or info command.

The Play Command

The play command starts playing a device. Without any flags, the play command starts playing from the current position and plays until the command is halted or until the end of the media or file is reached. For example, play cdaudio starts playing an audio disc from the

position where it was stopped.

Most devices support the play command also support the from and to flags. These flags indicate the position at which the device should start and stop playing. For example, play cdaudio from 0 plays the audio disc from the beginning of the first track. The units assigned to the position value depend on the device. For example, the position is normally specified in frames for CAV videodiscs, and milliseconds for digital audio.

As an extended command, devices add flags to use the capabilities of a particular device. For example, the play command for videodisc players adds the flags fast, slow, reverse, and scan.

Stop, Pause, and Resume Commands

The stop command suspends the playing or recording of a device. Many devices include the basic command pause, which also suspends these sessions. The difference between stop and pause depends on the device. Usually pause suspends operation but leaves the device ready to resume playing or recording immediately.

Using play or record to restart a device will reset the to and from positions specified before the device was paused or stopped. Without the from flag, these commands reset the start position to the current position. Without the to flag, they reset the end position to the end of the media. If you want to continue playing or recording but want to stop at a position previously specified, use the to flag with these commands and repeat the position value.

Some devices include the resume command to restart a paused device. This command does not change the to and from positions specified with the play or record command, which preceded the pause command.

MCI System Commands

The following commands are interpreted directly by MCI. The remaining command tables list commands interpreted by the devices.

Command	Description
sound	The device name of this command specifies a sound defined in a system setup file.. If it is not found, MCI uses a system default sound.
sysinfo	Obtains MCI system information. One of the following items modifies sysinfo:
item	installname Returns the name used to install the device.
	quantity Returns the number of MCI devices of the type

	specified by the device-name field. The device-name field must contain a standard MCI device type. Any digits after the name are ignored. The special device name all returns the total number of MCI devices in the system.
quantity	Returns the number of open
open	MCI devices of the type specified by the device name. The device name must be a standard MCI device type. Any digits after the name are ignored. The special device name all returns the total number of MCI devices in the system that are open.
name index	Returns the name of an MCI device. The index ranges from 1 to the number of devices of that type. If all is specified for the device name, index ranges from 1 to the total number of devices in the system.
name index	Returns the name of an open
open	MCI device. The index ranges from 1 to the number of devices of that type. If all is specified for the device name, index ranges from 1 to the total number of devices in the system.

Required Commands for All Devices

The following commands are recognized by all devices. Extended commands can add other options to these commands. A list of the errors common to all the commands follows the required command table.

Command	Description
capability	Requests information about a particular capability of a device. While other capabilities are defined for specific devices and device types, the following items are always available:
can eject	Returns true if the device can eject the media.
can play	Returns true if the device can play.
can record	Returns true if the device supports recording.
can save	Returns true if the device can save data.
compound	Returns true if the device

device requires an element name.
 device type Returns one of the following:
 audio tape
 cdaudio
 digital audio tape
 scanner
 sequencer
 videodisc
 videotape
 waveaudio
 has audio Returns true if the device supports audio playback.
 has video Returns true if the device supports video.
 uses files Returns true if the element of a compound device is a file pathname.
 close When sent to a simple device, closes the device. When sent to a compound device element, closes the element and any resources associated with it. MCI unloads a device when it is no longer being used.
 info item Fills a user-supplied buffer with a NULL-terminated string containing textual information. One of the following item modifies info:
 product Returns a description of the hardware associated with a device. This usually includes the manufacturer and model information.
 open items Initializes the device. The following optional items modify open:
 alias Specifies an alternate name for the given device. If specified, it must be used for subsequent references.
 shareable Initializes the device or element as shareable. Subsequent attempts to open it fail unless you specify shareable in both the original and subsequent open commands. MCI returns an error if it is already open and not shareable.
 type Specifies the compound device that controls a device element. As an alternative to type, MCI can use the filename extension entries to select the device based on the extension used by the device element.
 status item Obtains status information for the device. One of the following items modifies status:
 mode Returns the current mode of the device.

	Commonly supported standard modes are: not ready, paused, playing, stopped, open, recording, and seeking
ready	Returns true if the device is ready.

Basic Commands for Specific Device Types

In addition to the commands described previously, each device supports a set of commands specific to its device type. Where possible, these type-specific commands are identical between types. When type-specific commands are common to multiple devices, they are considered basic commands. For example, the basic play command is identical for videodisc and videotape players. Other basic commands are listed in the following table. Although these commands are optional for a device, if a command is used it must recognize all options listed in this table. The options generally provide for a minimum set of capabilities, but some devices may return unsupported function if an option is used which clearly doesn't apply.

Command	Description
load item	Load a device element from disk. The following optional item modifies load: filename Specifies the source path and file.
pause	Pauses playing or recording.
play items	Start playing the device. The following optional items modify play: from Specifies the position to start and stop playing. If to position from is omitted, the play starts from the current position; if to is omitted, the play stops at the end of the media.
record items	Start recording data. All data recorded after a file is opened is discarded if the file is closed without saving it. The following optional items modify record: insert Specifies that new data is added to the device element at the current position. from Specifies the positions to start and stop recording. to position If from is omitted, the device starts recording at the current position; if to is omitted, the device

records until a stop or pause command is received.

overwrite Specifies that new data will replace data in the device element.

The default recording mode (insert or overwrite) depends on the specific device. Each device should define a default recording mode.

resume Resumes playing or recording following a pause.

save item Saves the MCI element. The following optional item modifies save:

filename Specifies the destination path and file.

seek item Moves to the specified position and stops. One of the following is required for item :

to position Specifies the position to stop the seek.

to start Seeks to the start of the media or device element.

to end Seeks to the end of the media or device element.

set items Sets the various control items:

audio all Enables or disables audio output

off

audio all on

audio left Enables or disables output to the left audio channel.

off

audio left on

audio right Enables or disables output to the right audio channel.

off

audio right on

door closed Loads the media and closes the door if possible.

door open Opens the door and ejects the tray if possible.

time format Sets time format to milliseconds. All position information is this format after this command. You can abbreviate milliseconds as ms.

video off Enables or disables video output.

video on

status item Obtains status information for the device. One of the following items modifies status:

current Returns the current track.

track

length Returns the total length of the segment.

length Returns the length of the track

track serial track specified by track_number.

track_number

number of tracks	Returns the number of tracks on the media.
position	Returns the current position.
position track	Returns the position of the start of the track specified by track_number.
track_number	
start	Returns the starting position of the media or device element.
time format	Returns the time format.
stop	Stops the device.

CD Audio (Redbook) Commands

The CD audio command set provides a common method for playing CD audio sequences. CD audio devices support the following core-set of commands:

Command	Description
capability item	Requests information about the capabilities of the CD audio device. One of the following items is required:
can eject	Returns true if the CD audio device can eject the media.
can play	Returns true if the CD audio device can play the media.
can record	Returns false.
can save	Returns false.
compound device	Returns false.
device type	Returns CDAudio.
has audio	Returns true.
has video	Returns false.
uses files	Returns false..
close	Closes the device.
info item	Fills a user-supplied buffer with a NULL-terminated string containing textual information. One of the following optional item modifies info:
product	Returns the product name and model of the current audio device.
open items	Initializes the device. MCI reserves cdaudio for the compact disc audio device type. The following optional items modify open:
alias	Specifies an alternate name for the given device.
device_	
alias	If specified, it must also be used for subsequent references.
shareable	Initializes the device as

shareable. Subsequent attempts to open it fail unless you specify shareable in both the original and subsequent open commands. MCI returns an error if it is already open and not shareable.

pause Pauses playing.

play items Starts playing audio. The following optional items modify play:

- from Specifies the position to position to start and stop playing.
- position

resume Resumes playing from a paused state.

seek item Moves to the specified location on the disc. If already playing or recording, the device is stopped. One of the following items modifies seek:

- to position Specifies the destination position for the seek. If it is greater than the length of the disc, an out-of-range error is returned.
- to start Specifies to seek to the start of the audio data on the CD.
- to end Specifies to seek to the end of the audio data on the CD.

set items Sets the various control items:

- audio all Enables or disables audio output..
- off
- audio all on
- audio left Enables or disables output to the left audio channel.
- off
- audio left on
- audio right Enables or disables output to the right audio channel.
- off
- audio right on
- door closed Retracts the tray and closes the door if possible.
- door open Opens the door and ejects the tray if possible.
- time format Sets the time format to milliseconds. All position information is this format after this command. You can abbreviate milliseconds as ms.
- millisecond
- s
- time format Sets the time format to mm:ss:ff, where mm is minutes, ss is seconds, and ff is frames. All position information is in
- msf

this format after this command. On input, ff can be omitted if 0, and ss can be omitted if both it and ff are 0. These fields have the following maximum values:

Minutes	99
Seconds	59
Frames	74

time format Sets the time format to tt:mm:ss:ff where tt is tracks, mm is minutes, ss is seconds, and ff is frames. All position information is in this format after this command. On input ff can be omitted if 0, ss can be omitted if both it and ff are 0, and mm can be omitted if it, ss and ff are 0. These fields have the following maximum values:

Tracks	99
Minutes	99
Seconds	59
Frames	74

status item Obtains status information for the device. One of the following items modifies status:

current	Returns the current track.
track	
length	Returns the total length of the disc.
length	Returns the length of the track specified by track_number.
track_number	
r	
media	Returns true if the CD is inserted in the drive; otherwise, it returns false.
present	
mode	Returns not ready, open, paused, playing, seeking, or stopped for the current mode of the drive.
number of	Returns the number of tracks on the CD.
tracks	
position	Returns the current position.
position	Returns the starting position of the track specified by track_no.
track	
track_no	
ready	Returns true if the drive is ready.
start	Returns the starting position of the CD.
position	
time format	Returns the current time

stop format.
 Stops playing.

MIDI Sequencer Commands

The MIDI sequencer supports the following set of commands:

Command	Description
capability item	Requests additional information about the capabilities of the MIDI sequencer. One of the following items is required: can eject Returns false.. can play Returns true if the sequencer can play. can record Returns true if the sequencer can record MIDI data. can save Returns true if the sequencer can save MIDI data. compound device Generally returns true; most sequencers are compound devices.. device type Returns sequencer. has audio Returns true. has video Returns false. uses files Returns true.
close	Closes the sequencer element and the port and file associated with it.
info item	Fills a user-supplied buffer with a NULL-terminated string containing textual information. One of the following optional item modifies info: product Returns the product name of the current MIDI sequencer.
open items	Initializes the sequencer. The following optional items modify open: alias Specifies an alternate name for the sequencer element. If specified, it must also be used for subsequent references. shareable Initializes the sequencer element as shareable. Subsequent attempts to open it fail unless you specify shareable in both the original and subsequent open commands. MCI returns an invalid device error if it is already open and not shareable. type device_ MCI reserves sequencer for the MIDI sequencer device

	type	type. As an alternative to type, MCI can use the element filename extension entries to select the sequencer.
pause		Pauses playing.
play items		Starts playing the sequencer. The following optional items modify play:
	from	Specifies the positions to
	position	start and stop playing. If
	to position	from is omitted, play
		starts at the current
		position; if to is
		omitted, play stops at the
		end of the file.
record items		Starts recording MIDI data. All data
		recorded after a file is opened is
		discarded if the file is closed
		without saving it. The following
		optional items modify record:
	insert-	Specifies that new data is
		added to the device
		element.
	from	Specifies the positions to
	position	start and stop recording.
	to position	If from is omitted, the
		device starts recording at
		the current position; if
		to is omitted, the device
		records until a stop or
		pause command is received.
	overwrite	Specifies that new data
		will replace data in the
		device element.
resume		Resumes playing or recording following
		a pause.
save item		Saves the MCI element. The following
		item modifies save:
	filename	The filename specifies the
		destination path and file.
seek item		Moves to the specified position in the
		file. One of the following items is
		required:
	to position	Specifies the final
		position for the seek.
	to start	Specifies to seek to the
		start of the sequence.
	to end	Specifies to seek to the
		end of the sequence.
set items		Sets the various control items:
	audio all	Enables or disables audio
	off	output..
	audio all	
	on	
	audio left	Enables or disables output
	off	to the left audio channel.
	audio left	
	on	
	audio right	Enables or disables output
	off	to the right audio

audio right channel.
 on
 master MIDI Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in MIDI format.
 master none Inhibits the sequencer from sending synchronization data.
 master SMPTE Sets the MIDI sequencer as the synchronization source. Synchronization data is sent in SMPTE format.
 offset time Sets the SMPTE offset time in colon form (hours:minutes:seconds:frames). The offset is the beginning time of a SMPTE based sequence.
 port
 port_number Sets the MIDI port receiving the MIDI messages. This command will fail if the port you are trying to open is being used by another application.
 port mapper Sets the MIDI mapper as the port receiving the MIDI messages. This command will fail if the MIDI mapper or a port it needs is being used by another application.
 port none Disables the sending of MIDI messages..
 slave file Sets the MIDI sequencer to use file data as the synchronization source. This is the default.
 slave MIDI Sets the MIDI sequencer to use incoming data MIDI for the synchronization source. The sequencer recognizes synchronization data with the MIDI format.
 slave none Sets the MIDI sequencer to ignore synchronization data.
 slave SMPTE Sets the MIDI sequencer to use incoming MIDI data for the synchronization source. The sequencer recognizes synchronization data with the SMPTE format.
 tempo
 tempo_value Sets the tempo of the sequence according to the current time format. For a

ppqn-based file, the integer is interpreted as beats per minute. For a SMPTE-based file, the integer is interpreted as frames per second.

time format Sets time format to
millisecond milliseconds. All position
s information is specified as milliseconds following this command. The sequence file sets the default format to ppqn or SMPTE. You can abbreviate milliseconds as ms.

time format Sets time format to song
song pointer (sixteenth notes).
pointer This can only be performed for a sequence of division type ppqn.

time format Sets time format to SMPTE
SMPTE 24 24 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

time format Sets time format to SMPTE
SMPTE 25 25 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

time format Sets time format to SMPTE
SMPTE 30 30 frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

time format Sets time format to SMPTE
SMPTE 30 drop 30 drop frame rate. All position information is specified in SMPTE format following this command. The sequence file sets the default format to ppqn or SMPTE.

status item Obtains status information for the MIDI sequencer. One of the following items modifies status:

current Returns the current track number.

track

division Returns one of the following file division type:

type PPQN, SMPTE 24

frame, SMPTE 25 frame, SMPTE 30 drop frame, or SMPTE 30 frame. Use this information to determine the format of the MIDI file, and the meaning of tempo and position information.

length Returns the length of a sequence in the current time format. For ppqn files, this will be song pointer units. For SMPTE files, this will be in colon form (hours:minutes:seconds:frames).

length track track_number Returns the length of a sequence using the current time format. For ppqn files, this will be song pointer units. For SMPTE files, this will be in colon form (hours:minutes:seconds:frames).

master Returns midi, none, or smpte depending on the type of synchronization set.

media present mode The sequencer returns true. Returns not ready, paused, playing, seeking, or stopped.

number of tracks offset Returns the number of tracks. Returns the offset of a SMPTE-based file. The time is returned in colon form (hours:minutes:seconds:frames). The offset is the starting time of a SMPTE based sequence.

port Returns the MIDI port number assigned to the sequence.

position Returns the current position of a sequence in the current time format. For ppqn files, this will be song pointer units. For SMPTE files, this will be in colon form (hours:minutes:seconds:frames).

position track track_number Returns the current position of the track specified by track_number in the current time

	format. For ppqn files, this will be song pointer units. For SMPTE files, this will be in colon form (hours:minutes:seconds:frames).
ready	Returns true if the device is ready.
slave	Returns file, midi, none, or smpte depending on the type of synchronization set.
start position	Returns the starting position of the media or device element.
tempo	Returns the current tempo of a sequence in the current time format. For files with ppqn format, the tempo is in beats per minute. For files with SMPTE format, the tempo is in frames per second.
time format	Returns the time format.
stop	Stops playing.

Videodisc Player Commands

Videodisc players support the following core set of commands:

Command	Description
capability item	Reports the capabilities of the device. The device should report capabilities according to the type of disc (CAV or CLV) inserted in the drive. If no disc is inserted, the device should assume CAV. One of the following optional items modifies capability:
can eject	Returns true if the device can eject the media.
can play	Returns true if the device supports playing.
can record	Returns true if the video device can record.
can reverse	Returns true if the device can play in reverse, false otherwise. This is always false if a CLV disc is inserted.
can save	Returns false.
compound device	Returns false.
device type	Returns videodisc.
fast play rate	Returns the standard fast play rate of the player in frames per second. Returns

	0 if the device cannot play fast.
has audio	Returns true if the videodisc player has audio.
has video	Returns true.
media type	Returns CAV, CLV, or other, depending on the type of videodisc.
normal play rate	Returns the normal play rate in frames per second. Returns 0 for CLV discs.
slow play rate	Returns the standard slow play rate in frames per second. Returns 0 if the device cannot play slow.
uses files	Returns false.
close	Closes the device.
escape item	Sends custom information to a device. The following item modifies escape:
string	Specifies the custom information sent to the device.
info item	Fills a user-supplied buffer with a NULL-terminated string containing textual information. The following optional item modifies info:
product	Returns the product name of the device that the peripheral is controlling..
open items	Initializes the device. MCI reserves video disc for the videodisc device type. The following optional items modify open:
alias	Specifies an alternate name for the given device.
device_	If specified, it must also be used for subsequent references.
alias	
shareable	Initializes the device as shareable. Subsequent attempts to open it fail unless you specify shareable in both the original and subsequent open commands. MCI returns an invalid device error if it is already open and not shareable.
pause	Stops playing. If a CAV disc is playing, it also freezes the video frame. If a CLV disc is playing, the player is stopped.
play items	Starts playing. The following optional items modify play:
fast	Indicates that the device should play faster or
slow	slower than normal. To determine the exact speed

	on a particular player, use the status speed command. To specify the speed more precisely, use the fps flag. Slow applies only to CAV discs.
from position to position	Specifies the positions to start and stop playing. Positions are in frames for CAV discs and in seconds for CLV discs, unless chapter is also used (in which case, the position is given in chapters). If from is omitted, play starts at the current position; if to is omitted, the play stops at the end of the disc.
reverse	Sets the play direction to backwards. This applies only to CAV discs.
scan	Indicates the play speed is as fast as possible, possibly with audio disabled. This applies only to CAV discs.
speed integer	Specifies the rate of play. Currently supported speed values are measured in frames per second, which is the default. This applies only to CAV discs.
resume	Resumes playing.
seek item	Searches using fast forward or fast reverse with video and audio off. The following optional items modify seek:
reverse	Indicates the seek direction on CAV discs is backwards. This modifier is invalid if to is specified.
to position	Specifies the end position to stop the seek. If to is not specified, the seek continues until the end of the media is reached.
to start	Specifies to seek to the start of the disc.
to end	Specifies to seek to the end of the disc.
set items	Sets the various control items:
audio all	Enables or disables audio
off	output.
audio all	
on	
audio left	Enables or disables output
off	to the left audio channel.
audio left	

	on	
	audio right	Enables or disables output
	off	to the right audio
	audio right	channel.
	on	
	door open	Opens the door and ejects
		the tray, if possible.
	door closed	Retracts the tray and
		closes the door, if
		possible.
	time format	Sets the position format
	frames	to frames on CAV discs.
		All position information
		is specified in this
		format following this
		command. This is the
		default for CAV discs.
	time format	Sets position format to
	hms	h:mm:ss where h is hours,
		mm is minutes, and ss is
		seconds. All position
		information is specified
		in this format following
		this command. On input, h
		may be omitted if 0, and
		mm may be omitted if both
		it and h are 0. This is
		the default for CLV discs.
	time format	Sets the position format
	millisecond	to milliseconds. All
	s	position information is
		specified in this format
		following this command.
		You can abbreviate
		milliseconds as ms.
	time format	Sets the position format
	track	to tracks (chapters). All
		position information is
		specified in this format
		following this command.
	video on	Turns the video on or off.
	video off	
spin item		Starts the disc spinning or stops the
		disc from spinning. One of the
		following items modifies status:
	down	Stops the disc from
		spinning.
	up	Starts the disc spinning.
status item		Obtains status information for the
		device. One of the following items
		modifies status:
	current	Returns the current track
	track	(chapter) number.
	disc size	Returns either 8 or 12 to
		indicate the size of the
		loaded disc in inches.
	forward	Returns true if the play
		direction is forward or if
		the device is not playing;
		false if the play

	length	direction is backward. Returns the total length of the segment.
	length track	Returns the length of the track (chapter) specified by track_number.
	track_number	
	media present	Returns true if a disc is inserted in the device, false otherwise.
	media type	Returns either CAV, CLV, or other depending on the type of videodisc.
	mode	Returns not ready, opened, paused, parked, playing, scanning, seeking, or stopped.
	number of tracks	Returns the number of tracks (chapters) on the media.
	position	Returns the current position.
	position track	Returns the position of the start of the track (chapter) specified by track_number.
	track_number	
	ready	Returns true if the device is ready.
	side	Returns 1 or 2 to indicate which side of the disc is loaded.
	speed	Returns the speed in frames per second.
	start position	Returns the starting position of the disc.
	time format	Returns the time format.
step items		Step the play one or more frames forward or backward. The default action is to step one frame forward. The step command applies only to CAV discs. The following items modifies step:
	by frames	Specifies the number of frames to step. If a negative value is used, the reverse flag is ignored.
	reverse	Step backward.
stop		Stop playing.

Waveform Audio Commands

Waveform audio drivers must support the following core set of commands:

Command	Description
capability item	Requests additional information about the capabilities of the waveform audio

driver. One of the following items modify capability:

- can eject Returns false.
- can play Returns true if the device can play. The wave audio device returns true if an output device is available.
- can record Returns true if the waveform driver can record. The waveform audio device returns true if an input device is available.
- can save Returns true if the wave audio device can save data.
- compound device Generally returns true; most waveform audio devices are compound devices.
- device-type Returns waveaudio.
- has audio Returns true
- has video Returns false.
- inputs Returns the total number of input devices.
- outputs Returns the total number of output devices.
- uses files Returns true.

close Closes the device element and any resources associated with it.

cue item Prepares for playing or recording. The cue command does not have to be issued prior to playing or recording. However, depending on the device, it might reduce the delay associated with the play or record command. This command fails if playing or recording is in progress. The item is one of the following:

- input Prepares for recording.
- output Prepares for playing. This is the default.

delete items Deletes a data segment from the MCI element. The following optional items modify delete:

- from Specifies the positions to
- position start and stop deleting
- to position data. If from is omitted, the deletion starts at the current position; if to is omitted, the deletion stops at the end of the file or waveform.

info item Fills a user-supplied buffer with a NULL-terminated string containing textual information. One of the following items modifies info:

- file Returns the current filename.
- product Returns the product name

		of the current audio output device.
	input	Returns the product name of the current waveform input device or none if no device is set.
	output	Returns the product name of the current waveform output device or none if no device is set.
open items		Initializes the device. The following items are optional:
	alias	Specifies an alternate name for the given device.
	device_	
	alias	If specified, it must also be used the alias for references.
	buffer	Sets the size in seconds
	buffer_size	of the buffer used by the wave audio device. The default size of the buffer is set when the wave audio device is installed or configured. Typically, the buffer size is set to 4 seconds.
	shareable	Initializes the device element as shareable. Subsequent attempts to open it fail unless you specify shareable in both the original and subsequent open commands. MCI returns an error if it is already open and not shareable.
	type	Specifies the compound device used to control a device element. MCI reserves waveaudio for the waveform audio device type. As an alternative to type, MCI can use the element filename extension entries to select the controlling device
	device_type	
pause		Pauses playing or recording.
play items		Starts playing audio. The following optional items modify play:
	from	Specifies the positions to start and stop playing. If from is omitted, play starts at the current position; if to is omitted, play stops at the end of the file or waveform.
	position	
	to position	
record items		Starts recording audio. All data recorded after a file is opened is discarded if the file is closed

without saving it. The following optional items modify record:

- insert Specifies that new data is added to the device element.
- from Specifies the positions to start and stop recording.
- position to position If from is omitted, the device starts recording at the current position; if to is omitted, the device records until a stop or pause command is received.
- overwrite Specifies that new data will replace data in the device element.

resume Resumes playing or recording following a pause.

save item Saves the MCI element in its current format. The following item modifies save:

- filename Specifies the file and pathname used to save data.

seek item Moves to the specified location in the file. Playback or recording is stopped after the seek. One of the following items modify seek:

- to position Specifies the stop position.
- to start Specifies to seek to the first sample.
- to end Specifies to seek to the last sample.

set items Sets the following control items:

- alignment Sets the alignment of data blocks. The file is saved in the new format.
- any input Use any input that supports the current format when recording. This is the default.
- any output Use any output that supports the current format when playing. This is the default.
- audio all Enables or disables audio output.
- off
- audio all on
- audio left Enables or disables output to the left audio channel.
- off
- audio left on
- audio right Enables or disables output to the right audio channel.
- off
- audio right on
- bitpersamp Sets the number of bits per sample played or
- le

bit_count	recorded. The file is saved in this format.
bytespersec	Sets the average number of bytes per second played or recorded. The file is saved in this format.
byte_rate	Sets the average number of bytes per second played or recorded. The file is saved in this format.
channels	Sets the channel count for playing and recording. The file is saved in this format.
channel_count	Sets the channel count for playing and recording. The file is saved in this format.
format tag	Sets the format type for playing and recording. The file is saved in this format.
tag	Sets the format type for playing and recording. The file is saved in this format.
format tag	Sets the format type to PCM for playing and recording. The file is saved in this format.
pcm	Sets the format type to PCM for playing and recording. The file is saved in this format.
input	Sets the audio channel used as the input.
integer	Sets the audio channel used as the input.
output	Sets the audio channel used as the output.
integer	Sets the audio channel used as the output.
samplespers	Sets the sample rate for playing and recording. The file is saved in this format.
ec integer	Sets the sample rate for playing and recording. The file is saved in this format.
time format	Sets the time format to bytes. All position information is specified as bytes following this command.
bytes	Sets the time format to bytes. All position information is specified as bytes following this command.
time format	Sets the time format to milliseconds. All position information is specified as milliseconds following this command. You can abbreviate milliseconds as ms.
millisecond	Sets the time format to milliseconds. All position information is specified as milliseconds following this command. You can abbreviate milliseconds as ms.
s	Sets the time format to milliseconds. All position information is specified as milliseconds following this command. You can abbreviate milliseconds as ms.
time format	Sets the time format to samples. All position information is specified as samples following this command.
samples	Sets the time format to samples. All position information is specified as samples following this command.
status item	Obtains status information for the device. One of the following items modifies status:
alignment	Returns the block alignment of data in bytes.
bitspersamp	Returns the bits per sample.
le	Returns the bits per sample.
bytespersec	Returns the average number of bytes per second played or recorded.
channels	Returns the number of channels set (1 for mono, 2 for stereo).
current	Returns the index of the current track.
track	Returns the index of the current track.
format tag	Returns the format tag.

input	Returns the currently set input. If no input is set, the error returned indicates that any device can be used.
length	Returns the total length of the waveform.
length	Returns the length of the waveform track.
track	
track_number	
level	Returns the current audio sample value.
media	Returns true.
present	
mode	Returns not ready, paused, playing, stopped, recording, or seeking.
number of	Returns the number of tracks (chapters).
tracks	
output	Returns the currently set output. If no output is set, the error returned indicates that any device can be used.
position	Returns the current position.
position	Returns the position of the track specified by track_number.
track	
track_number	
ready	Returns true if the device is ready.
samplespers	Returns the number of samples per second played or recorded.
ec	
start	Returns the starting position of the waveform data.
position	
time format	Returns the current time format.
stop	Stops playing or recording.

Search for IT White Papers, Webcasts, and more at the TechWeb Digital Library

Welcome Guest. Log In Register Benefits

InformationWeek

THE BUSINESS VALUE OF TECHNOLOGY



InformationWeek
Our New iPad App

Home News Blogs Videos Slideshows Tech Centers

Software Security Cloud Mobility Social Business Personal Tech Hardware Windows Global CIO Government Health
SME

GovCloud
Building Secure, On-Demand Clouds For Government

Ranger Pavilion, D.C.
October 20, 2011



INFORMATIONWEEK
CALENDAR

FRONT END: DAILY DOSE

July 20, 1999

Internet On Wheels



Road warriors looking for ubiquitous network connectivity now have the opportunity to check E-mail, access the Internet, and query company databases while on the road—literally.

Startup CellPort Labs Inc. next week will release a server-based router that can be tucked under the driver's seat, under the hood, or in the trunk. Funded by AT&T Wireless among other cellular and satellite communications providers, CellPort has developed the CP2100, which connects to the Internet or intranets over a variety of wireless connections. "This gives IT managers the ability to 'Internet-use' vehicles," says CellPort CEO Patrick Kennedy.

The devices are initially being used by automakers to provide real-time online performance metrics on cars they are testing, while the military is testing them to provide satellite communications for Humvee vehicles. But Kennedy envisions a time, maybe three years from now, when an Ethernet port will appear on the dashboard of your Buick, maybe somewhere between the cigarette lighter and the radio. Mobile users will be able to plug notebook computers into their ports to connect to their company's intranet while stuck in traffic.

Let's just hope they drive better than cell-phone users do today.

--Brian Riggs

Past Items

IBM Boosts Privacy Services

Archived Daily Doses

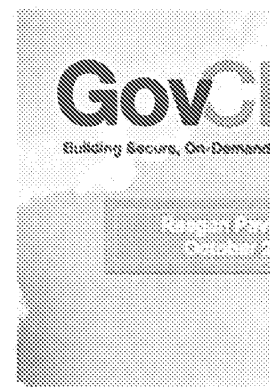
Send news, gossip and more to FrontEnd@cmp.com

Get InformationWeek

Don't miss each day's hottest news alerts directly to your inbox, including news alerts.

<http://www.informationweek.com/frontend/dose0720.htm>

Search for IT White Papers, Webcasts, and more at the TechWeb Digital Library



THIS WEEK'S TOP 10



- Subsc
- Read 1
- Read 1
- Downl

Free Print E

Back issues

CURRENT HEALTHCARE



Back issues

- In this
- EHR E the most practice: billions c stimulus use.
 - Flawed Patients how far respond: system: to die.
 - And:
 - Read

<https://dl.dropboxusercontent.com/static/javascript/external/pdf-js/viewer.html?file=https...> 3/26/2014

TECHNOLOGY WHITE PAPER

- Bathwick Group - The Mainframe role of the mainframe as a host enterprise cloud computing
- Clabby Analytics report: The N Advantage: How to Save Over
- How to Secure Virtualized Env Accelerate Cloud Computing
- Data security and privacy: A h
- Future-Proofing your Data Center Scalable Density

<http://www.informationweek.com/frontend/dose0720.htm>

<https://dl.dropboxusercontent.com/static/javascript/external/pdf-js/viewer.html?file=https...> 3/26/2014

Fleet system opts for mobile server - ITS International - News Article



The Leading Website and Magazine for Traffic Management and Urban Mobility

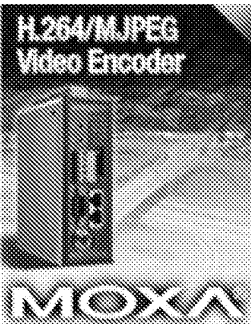
search

search website

Advanced search 

sections

- News
- Industry Index
- Events
- Feature articles
- Products
- Our latest issue
- Press releases
- Marketing services
- About us
- Contact us
- Portal / Links



H.264/MJPEG
Video Encoder

MOXA

news

Thursday 26 August 1999

Fleet system opts for mobile server

Computer Associates International has announced plans to use the CP2100 mobile network server from CellPort Labs (ITS International, July-August 1999) as an application platform and wireless communications gateway for its Unicenter TNG Fleet Management System.

Using the CP2100 as an in-vehicle computing platform, CA's Unicenter TNG Fleet Management System lets fleet managers manage large vehicle fleets remotely via real-time, wireless connections.

CellPort's CP2100 is installed within fleet vehicles where it acts as a network server capable of managing a wide range of data, including vehicle speed, odometer reading, fuel/oil levels, engine RPM, diagnostic trouble codes, door lock/unlock, air bag deployment and GPS data. The CP2100 then communicates vehicle data to the Unicenter TNG Fleet Management System via real time wireless links, extending the reach of corporate intranets and extranets to vehicles regardless of their location.

"Connecting mobile devices to traditional IT networks offers tremendous value to a wide range of business applications," says Pat Kennedy, CEO of CellPort Labs. "Allowing fleet managers the ability to track the health and welfare of vehicle assets from a centralised location is clearly one of the most exciting applications for this technology and our partnership with CA."

 Comment on this article

 Email this page to a friend

 More news stories

Want to license this story for your website or reprint it to promote your business? Email us [here](#)

Put our news headlines on your website for FREE

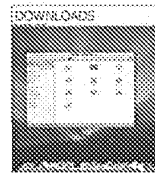
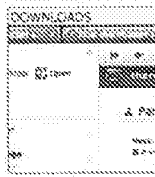
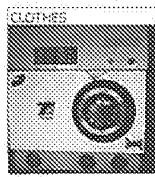
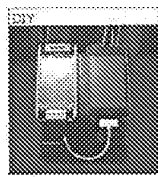
Contact ITS International | Terms & Conditions | Disclaimer | Copyright | Privacy Policy

se
Ad

H2
Vic



<http://www.itsinternational.com/News/article.cfm?recordID=547>



BOOKMARKLETS

Boxqueue Bookmarklet Saves Videos for Later Boxee Watching

By Kevin Purdy, 7:00 AM on Mon Jun 15 2009, 1,949 views



Media center Boxee lets you access a lot of web video sources, but what about that clip your friend emailed you earlier today? Boxqueue is a bookmarklet/webapp combo that feeds bookmarked clips into your Boxee feed.

It's a clever and mostly elegant little hack, playing off the simplicity of an "Add to Boxqueue" bookmark bar button

and Boxee's RSS versatility. After signing into Boxqueue and adding a personalized feed to your Boxee account, you're halfway there. After that, you simply find a clip you want to watch through your (likely TV-connected) Boxee setup later, hit the bookmarklet, and when you're in Boxee, head to the "My Feeds" section to see a queue of items you've saved for later.

Free to use, works with most any streaming video site, and requires a sign-in with your Google and Boxee accounts to activate.

Boxqueue [via boxee blog]

Ads by Google

Hook PC To TV

Turn Your PC Into A Media

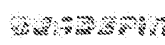
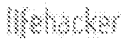
Media Center Pc

HP Official Store - Save Big on

<http://lifehacker.com/5291087/boxqueue-bookmarklet-saves-videos-for-later-boxee-watchi...> 6/16/2009

Center.
In Stock \$149 Free Shipping
www.SolidSignal.com

HP MediaSmart Servers. Shop
Now!
www.shopping.hp.com

[Advertising](#) [Legal](#) [Help](#) [Report a Bug](#)

under a Creative Commons License permitting non-commercial sharing with attribution.



[Log in](#) | [Sign up](#) | [Why join?](#)

Search: [Reviews](#) [Go](#)

[Today on CNET](#) [Reviews](#) [News](#) [Downloads](#) [Tips & Tricks](#) [CNET TV](#) [Compare Prices](#)
[Cell phones](#) | [Desktops](#) | [Digital cameras](#) | [Laptops](#) | [MP3 players](#) | [TVs](#) | [All Categories](#)

[Nominate your Web 2.0 favorites](#)

PRODUCT FINDER

[By price](#)

[By manufacturer](#)

[By product type](#)

[See all video product types](#)

DIGITAL MEDIA RECEIVERS

BUYING ADVICE

Editors' top TVs
Editors' top DVD decks
Editors' top DVRs (TiVo)
TV buying guide

Find all home-video reviews

DISCUSSIONS

Home video blog
Home video forums
Discuss home-theater gear: The Screening Room

RELATED PRODUCTS

DVD players
TV/HDTV tuners and receivers
Digital video recorders
Televisions
Home audio
Camcorders
Projectors

RELATED FEATURES

CNET quick guide: HD-DVD vs. Blu-ray
Digital Home DIY: Set up a home theater
DVD 2.0: HD-DVD and Blu-ray
HDTV World
Show Us Yours
Hot Houses
Fully Equipped
How we test: DVD players
How we test: DVD recorders

OTHER CNET FEATURES

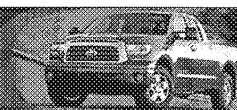
Digital Home
Ultimate Office
Go Mobile
Cell phone battery life chart

CNET'S FREE DRIVING ISSUE

Each Issue Includes the

advertisement

Most Significant Vehicle of 2007
Smart Car Buyers Start Here [CLICK HERE](#)
Free Local Price Quotes • Road Tests & Specs • Photos & Specs • Vehicle Ratings & Reviews



[Printer friendly version](#) [Email to a friend](#) [Editorial policies](#) [del.icio.us](#) [Digg](#)

Linksys Wireless-B Media Adapter

[Add to my products](#) [What is this?](#)

[REVIEW](#) | [USER OPINIONS](#) | [SPECIFICATIONS](#) | [COMPARE](#) | [SHOP](#) | [TIPS & TRICKS](#)

CNET video and photos



© 2007 CNET Networks, Inc.



Special sponsor stores

[Samsung](#)

[Polk Audio](#)

CNET editors' review

Very good
7.4
out of 10

The good: Good audio and graphics quality; easy setup; intuitive onscreen menu; includes audio and video cables.

The bad: Can't handle video, Internet radio, or audio CDs; requires Windows XP and .Net framework; hogs host PC's resources; incompatible with firewall; meager one-year warranty.

The bottom line: The Linksys Wireless-B media adapter does a good job of letting media hounds beam MP3s to their stereo or digital pictures to their TV, but it's pricey considering what it still lacks.

[Read editors' review below](#)

Average user rating from 35 users

Mediocre
4.1
out of 10

[Read user opinions](#)

How would you rate this product?
[Write your own review](#)

Buying choices

for **Linksys Wireless-B Media Adapter** from online stores:

This product is currently **not in stock** at any of our online merchants.

Find Linksys Wireless-B Media Adapter from our auction partner, eBay

[Email me when this product is available](#)

latest on our comprehensive coverage of digital TVs, DVD players and recorders, A/V receivers, DVRs, and much more.
Delivered Fridays

► Sign up now

► All free newsletters

▼ sponsored

TOP PRODUCTS FROM TOP BRANDS
Dell products
HP products
Sony products
Samsung

▼ advertisement



CNET editors' review

Reviewed by: Brian Nadel

Reviewed on 7/30/03 Release date: 7/15/03

The latest entry in the emerging category of digital audio receivers and media adapters is the Linksys Wireless-B media adapter, also known as the WMA11B. It uses your existing wireless home network to link your PC and entertainment center. The WMA11B works as advertised, streaming MP3s to your stereo and displaying digital snapshots on your TV, but considering that this 802.11b media adapter doesn't support video and lacks certain security features, it's pricey. Still, if you're itching to connect your living room to the rest of your wireless home network, the WMA11B is the best media adapter we've seen so far.

Setup and ease of use of Linksys Wireless-B Media Adapter

The media adapter's package includes a setup CD; an AC adapter, Ethernet, RCA, and S-Video cables, and a remote control. Documentation consists of an installation guide and a 56-page electronic manual. To use the media adapter, your TV must have a composite-video or an S-Video input, and your stereo needs an RCA jack. Your computer must have 802.11b wireless or Ethernet installed with TCP/IP enabled. Finally, you need to have Windows XP Home Edition or Professional as your installed OS; other Windows users, not to mention Mac users, are out of luck.

Setting up the media adapter is complicated, but the included quick-install guide breaks the process into a series of simple tasks. First, you connect the device to your computer using an Ethernet cable and run the setup CD, which walks you through network configuration. You could do this through the TV, but it's more awkward that way. Next, you'll need to load the Media Adapter Utility onto your PC. For this, you'll need Microsoft's .Net framework, which became a standard part of the Windows XP OS only recently. The setup CD automatically checks your computer and installs .Net, if necessary. Last, you use the Media Adapter Utility to tell the device where your pictures and music are stored on the PC.

When everything's set, it's time to plug in the media adapter to the stereo and TV. If your stereo and TV are placed far apart, you may need to buy an extension cable. Once connected, the media adapter's menu appears automatically on the TV screen.

Features and security of Linksys Wireless-B Media Adapter

Other media adapters and digital audio receivers hit the market earlier this year, but the cd30 is the only direct competitor we've reviewed that fully integrates itself into an existing wireless LAN, while others such as the Lyra Wireless RCA RD900W and the X10 Lola are more limited as far as networking goes. Even the cd30 fails to offer the flexibility and easy LAN integration you'll get with the WMA11B.

The media adapter's vertical profile and compact size (1.97 inches wide by 6.3 inches deep by 7.48 inches tall) make it easy to tuck into a small space. A set of LEDs on its front show network activity. Along the back are all of the connectors, a Reset (to default settings) button, and an uplink button that toggles the Ethernet port between MDI and MDIX cable settings.

The Wireless-B media adapter's remote is fairly intuitive. Select either the Pictures or the Music button to access the content folders on your PC. Image and music files can be retrieved simultaneously. You hit the Setup button to check the media adapter's wireless and network settings and make changes with the onscreen keyboard. You may find it awkward to navigate the keyboard and select characters using the remote's arrow buttons, though.

The remote control's features are a mixed bag. You have plenty of options for viewing images or playing audio tracks. You can't back out of a selection, although it's easy to start over by pressing Menu, Music, or Pictures. A universal remote would have helped cut down on clicker-clutter. In a minor but puzzling development, the volume control didn't work on our circa-2000 Samsung LCD TV or TEAC stereo. Neither we nor Linksys could figure out why.



The remote makes it easy to play media files on your TV or stereo.

The media adapter cannot address all of your music needs. The Utility can access content on external drives, but it can't play audio CDs. The Media Adapter won't process WAV audio files, video of any kind, or Internet radio, though Linksys is considering these features for future upgrades.

The media adapter can join a Wi-Fi network that uses 64- or 128-bit WEP encryption, but it does not support the more advanced WPA standard. Then again, no other product in this developing category offers WPA, either. Linksys plans to support WPA later this year through a free firmware upgrade. Also, the device won't work with most firewalls, including Windows XP's.

Performance of Linksys Wireless-B Media Adapter

In CNET Labs' tests, the Linksys Wireless-B media adapter achieved 5.3Mbps maximum throughput, putting it on a par with the best 802.11b devices. Its 100-foot range gives it an operational zone of more than 30,000 square feet—plenty for the typical home or apartment. As range increases, however, the media adapter's throughput will decrease, and you should expect compromised audio quality at the

User Opinions

See all 35 user opinions

How would you rate this product?

Write your own review

1 out of 10 Abysmal
"Great idea but very difficult to set up (read more)"
by fbcourse (see profile), April 24, 2005

2 out of 10 Terrible
"Doesn't work with Firewall!!?!!? (read more)"
December 8, 2004

3 out of 10 Poor
"It works, but Linksys needs to UPDATE (read more)"
December 13, 2003

See all 35 user opinions

▼ advertisement

Wireless

Find everything you need to go wireless at Dell Small Business!
www.dell.com

Wireless Media Players

Save on Wireless Media Players Order Today. Ships Today - Buy!
www.tigerdirect.com

Data Transmitter-Receiver

RS232/485, 2.4GHz & 900MHz FHSS, high rates, long range, certified.
www.aerocomm.com

(about)

Discover new gorgeous gadgets

Cell phones your friends will envy
Laptops that will turn heads
All the latest gadget gossip & news is here

GO TO CRAVE

a blog from CNET

crave

▼ advertisement

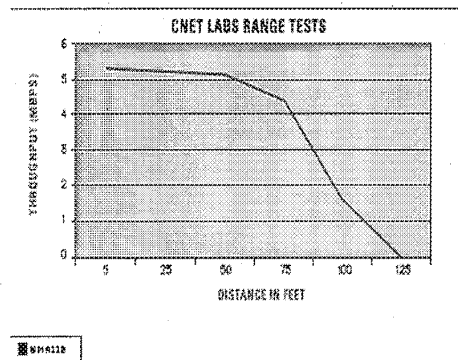
NOW ON

Most Popular Digital Media Receivers

fringes of its range. In our informal tests, the media adapter connected without a hitch to seven different Wi-Fi devices and continued to send an uninterrupted audio stream while the others were sending and receiving data. The composite-video link produced fuzzy viewing of still photographs, but the S-Video connection was pinpoint perfect, and the audio was static-free.

The media adapter is definitely a resource hog, though. The computer running the Media Adapter Utility occasionally hit peaks of 100 percent memory use, and we also noted 70 percent CPU-use spikes.

1. Apple TV
2. Netgear EVA8000 Digital Entertainer HD
3. D-Link MediaLounge DSM-520
4. Sonos Digital Music System (ZP80 bundle)
5. Roku SoundBridge M1000/M1001



For practical throughput tests, CNET Labs uses NetIQ's Chariot 4.3 software with Chariot 4.4 Endpoints as its benchmark. For wireless testing, the clients and the routers are set up to transmit at various distances from the access point and to automatically select the best transmit speed. All tests are run with Chariot software using TCP and are run in our CNET offices over channel 1. Our tests indicate the range that you can expect in a typical office environment, but range in your own home or office may differ. You may be able to achieve better performance in situations where you can establish a clear line of sight. For more details on how we test networking devices, see the CNET Labs site.

Service and support of Linksys Wireless-B Media Adapter

The Wireless-B media adapter's service-and-support policies could be better for the price. The media adapter comes with lifetime, 24/7 phone and e-mail support, but its one-year warranty is much shorter than the three years' coverage that Linksys offers on most of its wireless products. At least the Linksys Web site has plenty to offer, from manuals and firmware updates to troubleshooting, warranty info, and online product registration.

People who looked at this product also looked at

D-Link MediaLounge DSM-520

Netgear EVA8000 Digital Entertainer HD

(about)

Wireless

Find everything you need to go wireless at Dell Small Business!
www.dell.com

[Visit Web Site](#)

Wireless Media Players

Save on Wireless Media Players Order Today, Ships Today - Buy!
www.tigerdirect.com

[Visit Web Site](#)

HD Friend Plus

High Definition Media Player perfect for digital signage, kiosks
www.electronicmedianetworks.com

[Visit Web Site](#)

Data Transmitter-Receiver

RS232/485, 2.4GHz & 900MHz FHSS, high rates, long range, certified.
www.aerocomm.com

[Visit Web Site](#)

D-Link MediaLounge

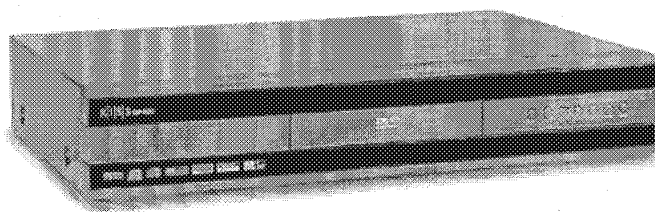
Stream Video, Music, And Photos From An Intel® ViiV™ PC To Your TV!

[Visit Web Site](#)

advertisement

Dell Home PCs

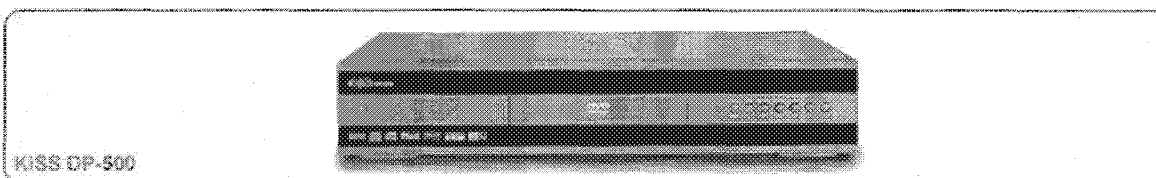
Dell • Home • Systems



KISS DP-500

DivX compatible DVD player with Ethernet connection

The DP-500 is based on the ground breaking DP-450 with its DivX, XviD, Mpeg 4, MP3, Ogg Vorbis reading capabilities. An Ethernet 10/100 port has been added to the player along with a Sigma Designs EM8500 progressive DVD decoder chip which enable any user with a 1MB xDSL connection to stream DVD quality movies directly from a connected PC. To enhance the user's experience and to take the best out of this connection, the DP-500 ships with the KISS PC-Link software which makes possible to read any of the supported file format straight from your PC to your DVD player and your TV via a straight forward and user-friendly interface. To change your DP-500 experience from exceptional to unforgettable, KISS introduces Webradio. Accessing webradio.kiss-technology.com will provide you with a number of radio stations that can be streamed from the internet to your DVD player and your TV. To make this session comfortable and according to your own taste, once registered, Webradio will allow you to choose, customize, add as many radio station as you wish. The KISS DP-500 also comes complete with the standard CD/MP3, Ogg Vorbis, DivX, XviD, CD-RW and DVD+-RW playback.

[back](#)
[Know more about DivX ?](#)
[go](#)


Specifications

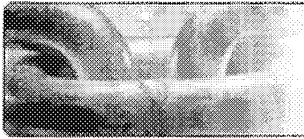
- ? Full DVD/MPEG-4/DivX/XviD/CD/MP3/Ogg Vorbis/CD-RW/DVD+-RW compatibility
- ? JPEG Picture CD viewing
- ? Progressive Scan Video Out
- ? Time Search
- ? Playback Control: Play/Pause/Stop/FF/RW/NEXT/PREV
- ? Repeat Playback
- ? Title/chapter select
- ? Parental lock, Multi Language, Multi Subtitle, Multi Angle
- ? Remote Control
- ? TV Type (PAL and NTSC)
- ? Configuration Setup Menu
- ? Aspect Ratio (4:3 Pan and Scan, 4:3 Letter Box and 16:9 Wide)
- ? S-Video Output
- ? Composite Video Output
- ? Component via SCART Output
- ? RGB via SCART Output
- ? Dolby Digital, MPEG Audio Decoding
- ? Dolby Digital/MPEG 5.1 Digital Output
- ? S/PDIF Outputs
- ? Analogue stereo
- ? Ethernet 10/100
- ? Webradio/KISS PC-Link
- ? KISS PC-Link Software for PC

Special features



Best ever DivX compatibility with KISS

KISS DVD players play all DivX format, DivX 3.11, DivX 4 and DivX 5. What does this mean for you? It means that now, for the first time, you can download full-screen, full-motion videos from the internet that actually look and sound like what you get from your television or DVD player. With DivX compression, you can download and watch high-quality video on your television set or other convergence devices.



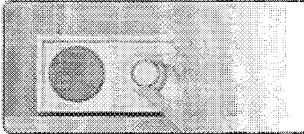
KISS PC-Link

KISS PC-Link is a software that enables the user to play music, see pictures and view movies from a PC to the DivX compatible DVD player thanks to the ethernet connection found on the back of the player. KISS PC-Link is a user-friendly software to install on a PC running at least Windows 98. With a simple interface KISS PC-Link makes it easy to select any media files on a PC's hard disk.



Internet Upgradable

KISS is keen in keeping a special relation with its customers, so, even after a player has been sold, its software and technology keeps moving. The firmware of the player keeps updated with the latest research and development and KISS is able to release a firmware update on a regular basis. This latest firmware is available from our website in the support section.



KISS WebRadio

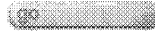
Thanks to its ethernet connection, this player will provide users with a direct connection to the Internet. This internet connection enables it to play internet radio. Just select this feature on the Player, and you will have access to around 5000 radio stations that can be streamed from a friendly user interface and played on your TV through the KISS DVD Player.

Downloads

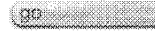
View the DP-500 presentation
(requires Flash 6 player)



Download brochure



Download latest firmware



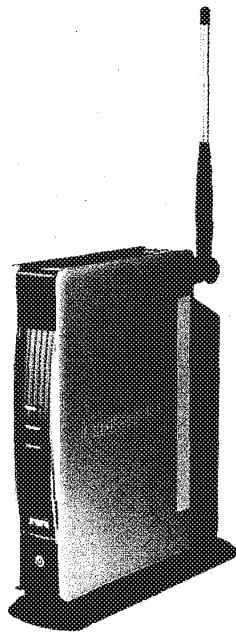
© 1992-2007 Cisco Systems, Inc. All rights reserved.

[Terms & Conditions](#) | [Privacy Policy](#) | [Legal Notice and Compliance](#) | [Careers](#) | [Contact Us](#)





Enjoy your digital music and pictures on your home entertainment center, without stringing wires!



The Linksys Wireless-B Media Adapter lets you bring the digital music and pictures stored on your computer to your Home Entertainment Center, without running cables through the house. Using a wireless connection, the Media Adapter displays your digital photographs on the TV for the whole family to enjoy. And your digital music collection is finally freed from those little computer speakers and can play in full glory through your stereo system.

The Wireless-B Media Adapter sits by your home stereo and television and connects to them using standard consumer electronics cables. Then it connects to your home network by Wireless-B (802.11b) wireless networking, or if you prefer, it can be connected via standard 10/100 Ethernet cabling. Using the included remote control and the user-friendly menus on your TV, you can browse through the digital pictures on your computer by folder, filename, or thumbnail. You can view pictures one at a time, or watch an automatically created slideshow of all the pictures in a given folder. The Media Adapter supports four popular picture formats: JPG, GIF, TIF and BMP. Use the remote control's Zoom button to get a close-up of the details in your pictures.

You can also use the remote to browse your MP3 or WMA formatted music collection by title, artist, genre, folder, or playlist. Choose the music you want, and let the Wireless-B Media Adapter play it through your stereo system. You can even let music play in the background while you browse your pictures.

Let the Linksys Wireless-B Media Adapter bring your digital media out into the living room for the whole family to enjoy.

Connects your Home Entertainment Center to a Wireless-B (802.11b) network

Listen to your MP3 and WMA digital music collection through your stereo system

Browse your digital pictures on your television (JPG, GIF, TIF, and BMP)

Select songs and pictures from on-screen menus with the easy-to-use remote control

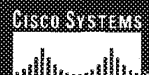
2.4GHz
802.11b

Wireless-B Media Adapter

Product Data

Wireless

Model No. **WMA11B**



2.4GHz 802.11b Wireless-B Media Adapter

Features

- Supports popular audio formats (MP3 and WMA), image formats (JPG, GIF, TIF and BMP), and playlist formats (M3U and ASX)
- S-Video output for best video quality
- RCA connectors for left and right stereo audio output and composite video output
- Easy TV setup and convenient Setup Wizard for configuration
- User-friendly utility runs on your PC to manage music and picture folders
- Wireless security with up to 128-bit WEP encryption
- Compatibility with 802.11b (2.4GHz) standard
- Up to 11Mbps, high-speed wireless data transfer rate
- RJ-45, 10/100Mbps Ethernet connection

Specifications

Model	WMA11B
Standards	IEEE 802.3, IEEE 802.3u, and IEEE 802.11b
Ports	One 10/100 RJ-45 Port, Power Port, S-Video Output, Composite Video Output, RCA Audio Output
Buttons	Power, Uplink
Channels	802.11b 11 Channels (US, Canada) 13 Channels (Europe) 14 Channels (Japan)
LEDs	Ready, Wireless, Ethernet
Transmitted Power	16 dBm
Receive Sensitivity	-82 dBm @ 11Mbps (typical) -85 dBm @ 5.5Mbps (typical) -89 dBm @ 2Mbps (typical) -91 dBm @ 1Mbps (typical)
Peak Gain of the Antenna	1.5 dBi
Security Features	WEP Encryption
WEP Key Bits	64, 128-bit
Warranty	1 Year

Environmental

Dimensions (W x H x D)	6.30" x 7.48" x 1.97" (160 mm x 190 mm x 50 mm)
Unit Weight	13.05 oz. (0.37 kg)
Power	External, 5V DC, 2.0 A
Certifications	FCC
Operating Temp.	0°C to 60°C (32°F to 140°F)
Storage Temp.	0°C to 85°C (32°F to 185°F)
Operating Humidity	10% to 85%, Non-Condensing
Storage Humidity	5% to 90%, Non-Condensing

Linksys
A Division of Cisco Systems, Inc.
18552 Teller Avenue
Irvine, CA 92612 USA
E-mail: sales@linksys.com
support@linksys.com
Web: <http://www.linksys.com>
Linksys products are available in more than 50 countries, supported by 12 Linksys Regional Offices throughout the world. For a complete list of local Linksys Sales and Technical Support contacts, visit our Worldwide Web Site at www.linksys.com.

Minimum Requirements

- One TV with Composite Video or S-Video input
- One Stereo or A/V Receiver with RCA Audio input
- One PC (600MHz or Faster Processor) with:
 - Windows XP Home Edition or Professional
 - 128MB RAM
 - 25MB available Hard Drive space
 - CD-ROM
 - 802.11b Wireless Adapter with TCP/IP protocol installed or Network Adapter with TCP/IP protocol installed and Ethernet Network Cable

Package Contents

- Wireless-B Media Adapter (with base and cable fixed)
- Infrared Remote Control (2 AAA batteries included)
- Audio/Video Cable with 3 RCA connectors
- S-Video Cable
- Setup CD-ROM with User Guide
- Power Adapter
- Ethernet Network Cable
- Quick Installation and Registration Card

Specifications are subject to change without notice. Linksys is a registered trademark or trademark of Cisco Systems, Inc. and/or its affiliates in the U.S. and certain other countries. Copyright © 2006 Cisco Systems, Inc. All rights reserved. Other brands and product names are trademarks or registered trademarks of their respective holders.

WMA11B-US-80211b-A

Product Data

Model No. **WMA11B**

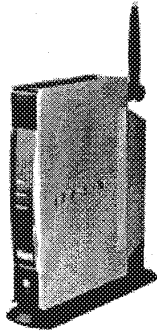


WMA11B

Home » Products » Wireless » Entertainment » Music, Picture, and Video Links » WMA11B

Product Features

Wireless-B Media Adapter



- Connects your Home Entertainment Center to a Wireless-B (802.11b) network
- Listen to your MP3 and WMA digital music collection through your stereo system
- Browse your digital pictures on your television (JPG, GIF, TIF, and BMP)
- Select songs and pictures from on-screen menus with the easy-to-use remote control

Product Description

The Linksys Wireless-B Media Adapter lets you bring the digital music and pictures stored on your computer to your Home Entertainment Center, without running cables through the house. Using a wireless connection, the Media Adapter displays your digital photographs on the TV for the whole family to enjoy. And your digital music collection is finally freed from those little computer speakers and can play in full glory through your stereo system.

The Wireless-B Media Adapter sits by your home stereo and television and connects to them using standard consumer electronics cables. Then it connects to your home network by Wireless-B (802.11b) wireless networking, or if you prefer, it can be connected via standard 10/100 Ethernet cabling. Using the included remote control and the user-friendly menus on your TV, you can browse through the digital pictures on your computer by folder, filename, or thumbnail. You can view pictures one at a time, or watch an automatically created slideshow of all the pictures in a given folder. The Media Adapter supports four popular picture formats: JPG, GIF, TIF and BMP. Use the remote control's Zoom button to get a close-up of the details in your pictures.

You can also use the remote to browse your MP3 or WMA formatted music collection by title, artist, genre, folder, or playlist. Choose the music you want, and let the Wireless-B Media Adapter play it through your stereo system. You can even let music play in the background while you browse your pictures.

Let the Linksys Wireless-B Media Adapter bring your

More Information

- » [Data Sheet](#)  pdf
- » [Product Image](#)
- » [Product Image](#) (high res)
- » [User Guide](#)  pdf
- » [Driver](#)
- » [Quick Install](#)  pdf

Related Products



BEFSR41
[EtherFast® Cable/DSL Router with 4-Port Switch](#)

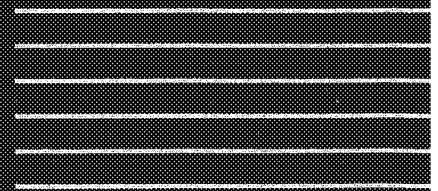
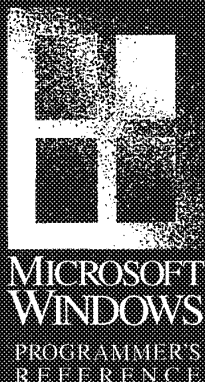
digital media out into the living room for the whole family
to enjoy.



1992-2007 Cisco Systems, Inc. All rights reserved.



Microsoft Windows Multimedia Programmer's Reference

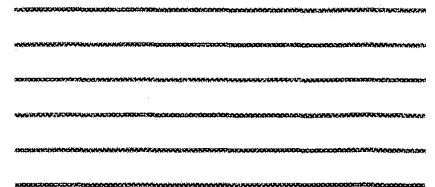
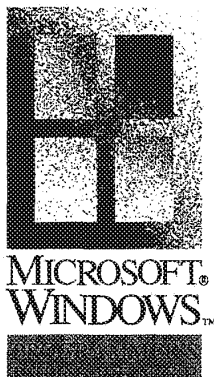


Microsoft® Windows™

Multimedia Programmer's Reference

*Written, edited, and produced by
Microsoft Corporation*

Distributed by Microsoft Press



PUBLISHED BY

Microsoft Press

A Division of Microsoft Corporation

One Microsoft Way, Redmond, Washington 98052-6399

Copyright © 1991 by Microsoft Corporation. All rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Microsoft.

Library of Congress Cataloging-in-Publication Data

Microsoft Windows multimedia programmer's reference / Microsoft Corporation.

p. cm. -- (Microsoft Windows multimedia programmer's reference library)

Includes index.

ISBN 1-55615-389-9

1. Computer animation. 2. Computer sound processing.

3. Microsoft Windows (Computer program) I. Microsoft. II. Series.

TR897.5.M535 1991

006.6'76--dc20

91-15490

CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 MLML 6 5 4 3 2 1

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

Distributed to the book trade outside the United States and Canada by Penguin Books Ltd.

Penguin Books Ltd., Harmondsworth, Middlesex, England

Penguin Books Australia Ltd., Ringwood, Victoria, Australia

Penguin Books N.Z. Ltd., 182-190 Wairau Road, Auckland 10, New Zealand

British Cataloging-in-Publication Data available.

Apple® and Macintosh® are registered trademarks and QuickDraw™ is a trademark of Apple Computer, Inc. IBM® is a registered trademark of International Business Machines Corp. Intel® is a registered trademark of Intel Corp. MacroMind Director® is a registered trademark of MacroMind, Inc. Microsoft®, MS®, MS-DOS®, and the Microsoft logo are registered trademarks and Windows™ is a trademark of Microsoft Corp. Motorola® is a registered trademark of Motorola, Inc. Pioneer® is a registered trademark of Pioneer Kabushiki Kaisha. Sound Blaster™ is a trademark of Creative Labs, Inc.

Document Number: MM20347-0691

Chapter 8

Multimedia File Formats

This chapter describes the file formats used with the Multimedia extensions. The chapter describes the structure of each file type and includes detailed lists of the data structures and fields contained in the files. The chapter also presents examples of multimedia files.

Several file formats used with the Multimedia extensions are based on the Resource Interchange File Format (RIFF). This chapter defines RIFF, the preferred format for new multimedia file types. If your application requires a new file format, you should define it using the RIFF tagged file structure described in this chapter.

This chapter describes the following file formats:

- Device Independent Bitmap (DIB) File Format
- RIFF DIB File Format (RDIB)
- Musical Instrument Digital Interface (MIDI) File Format
- RIFF MIDI File Format (RMID)
- Multimedia Movie File Format (MMM)
- Palette File Format (PAL)
- Rich Text Format (RTF)
- Waveform Audio File Format (WAVE)

About the RIFF Tagged File Format

RIFF (Resource Interchange File Format) is the tagged file structure developed for multimedia resource files. The structure of a RIFF file is similar to the structure of an Electronic Arts IFF file. In fact, RIFF is not actually a file format itself (since it does not represent a specific kind of information), but its name contains the words “interchange file format” in recognition of its roots in IFF. Refer to the EA IFF definition document, *EA IFF 85 Standard for Interchange Format Files*, for a list of reasons to use a tagged file format.

The Multimedia file I/O services provide support for working with RIFF files. For information on the file I/O services, see Chapter 9, “Multimedia File I/O Services,” in the *Programmer's Workbook*. Also, functions, messages, and data structures provided by the Multimedia file I/O services are described in this Reference.

Chunks

The basic building block of a RIFF file is called a *chunk*, which, defined using C syntax, looks like the following:

```
typedef unsigned long DWORD;
typedef unsigned char BYTE;
typedef DWORD FOURCC;           // Four-character code

typedef struct {
    FOURCC ckID;
    DWORD ckSize;                // the size of field <ckData>
    BYTE ckData[ckSize];        // the actual data of the chunk
} CK;
```

Four-character codes (FOURCCs) are used extensively in RIFF files; they identify the sections of data contained in the file. A four-character code has the following characteristics:

- A 32-bit quantity represented as a sequence of one to four ASCII alphanumeric characters
- Padded on the right with blank characters (ASCII character value 32)
- Contains no embedded blanks.

For example, the four-character code “FOO” is stored as a sequence of four bytes ('F' 'O' 'O' ' ') in ascending addresses. For quick comparisons, a four-character code may also be treated as a 32-bit number.

The chunk fields are as follows:

Part	Description
ckID	Chunk ID. This four-character code identifies the representation of the chunk data. A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it skips the number of bytes specified by the ckSize field plus the pad byte, if present.
ckSize	Chunk size. This is a 32-bit unsigned value identifying the size of ckData . This size value includes does not include the size of the ckID or ckSize fields or the pad byte at the end of the ckData field.
ckData	Chunk data. This is binary data of fixed or variable size. The start of ckData is word-aligned with the start of the RIFF file. If the size of the chunk is an odd number of bytes, a pad byte with value zero is written after ckData . Word aligning is done to improve access speed (for chunks resident in memory) and for compatibility with EA IFF. The ckSize value does not include the pad byte.

Two types of chunks, the "LIST" and "RIFF" chunks, may contain nested chunks, or subchunks. These special chunk types are discussed later in this document. All other chunk types store a single element of binary data in **ckData**.

Chunks are represented using the following notation (in this example, the **ckSize** field and pad byte are implicit):

```
<ckID> ( <ckData> )
```

For example, a chunk with chunk ID "FOO" might be represented as follows:

```
FOO ( <foo-Data> )
```

It's common to refer to chunks by their chunk ID; the chunk shown above would be called a "FOO" chunk.

RIFF Forms

A RIFF form is a chunk with a “RIFF” chunk ID. The term also refers to a file format that follows the RIFF framework. The following lists currently registered RIFF forms. Each is described later in this document.

Form Type	Description
PAL	Palette File Format
RDIB	RIFF Device Independent Bitmap Format
RMID	RIFF MIDI Format
RMMP	RIFF Multimedia Movie File Format
WAVE	Waveform Audio Format

Using the notation for representing a chunk, a RIFF form looks like the following:

```
RIFF ( <formType> <ck>... )
```

The first four bytes of a RIFF form make up a chunk ID with values ‘R’ ‘I’ ‘F’ ‘F’. The chunk size (**ckSize**) field is required, but for simplicity it is omitted from the notation.

Note RIFF has a counterpart, RIFX, that you can use to define RIFF file formats that use the Motorola integer byte-ordering format rather than the Intel format. A RIFX form is the same as a RIFF form, except that the first four bytes contain the character sequence ‘R’ ‘I’ ‘F’ ‘X’ rather than RIFF. Integer byte ordering is represented in Motorola format.

The first DWORD of chunk data in the “RIFF” chunk (shown above as **<formType>**) is a four-character code value identifying the data representation, or *form type*, of the file. Following the form-type code is a series of subchunks. Which subchunks are present depends on the form type. The definition of a particular RIFF form typically includes the following:

- A unique four-character code identifying the form type
- A list of mandatory chunks
- A list of optional chunks
- Possibly, a required order for the chunks

Defining and Registering RIFF Forms

The form-type code for a RIFF form must be unique. To guarantee this uniqueness, you must register any new form types before release. To register form types, and to get a current list of registered RIFF forms, request a *Multimedia Developer Registration Kit* from the following group:

Microsoft Corporation
Multimedia Systems Group
Product Marketing
One Microsoft Way
Redmond, WA 98052-6399

Like RIFF forms, RIFX forms must also be registered. Registering a RIFF form does not automatically register the RIFX counterpart. No RIFX form types are currently defined.

When you document the RIFF form, you should use the notation described in "Notation for Representing RIFF Files," later in this chapter.

Registered Form and Chunk Types

By convention, the form-type code for registered form types contains only digits and uppercase letters. Form-type codes that are all uppercase denote a registered, unique form type. Use lowercase letters for temporary or prototype chunk types.

Certain chunk types are also globally unique and must also be registered before use. These registered chunk types are not specific to a certain form type; they can be used in any form. If a registered chunk type can be used to store your data, you should use the registered chunk type rather than define your own chunk type containing the same type of information.

For example, a chunk with chunk ID "INAM" always contains the name or title of a file. Also, within all RIFF files, filenames or titles are contained within chunks with ID "INAM" and have a standard data format.

Unregistered (Form-Specific) Chunk Types

Chunks types that are used only in a certain form type use a lowercase chunk ID. A lowercase chunk ID has specific meaning only within the context of a specific form type. After a form designer is allocated a registered form type, the designer can choose lowercase chunk types to use within that form. See the *Multimedia Developers Registration Kit* for details.

For example, a chunk with ID "scln" inside one form type might contain the "number of scan lines." Inside some other form type, a chunk with ID "scln" might mean "secondary lambda number."

Notation for Representing RIFF Files

RIFF is a binary format, but it is easier to comprehend an ASCII representation of a RIFF file. This section defines a standard notation used to present samples of RIFF files. If you define a RIFF form, we urge you to use this notation in any file format samples you provide in your documentation.

Element Notation Conventions

The following table lists notation conventions used to describe elements of RIFF files. Further conventions are presented in the following sections.

Notation	Description
<code><element label></code>	RIFF file element with the label "element label"
<code><element label: TYPE></code>	RIFF file element with data type "TYPE"
<code>[<element label>]</code>	Optional RIFF file element
<code><element label>...</code>	One or more copies of the specified element
<code>[<element label>]...</code>	Zero or more copies of the specified element

Basic Notation for Representing RIFF Files

The following table summarizes the elements of the RIFF notation required for representing sample RIFF files:

Notation	Description
<code><ckID> (<ckData>)</code>	The chunk with ID <code><ckID></code> and data <code><ckData></code>. The <code><ckID></code> field is a four-character code that may be enclosed by single quotes for emphasis. For example, the following notation describes a "RIFF" chunk with a form type of "QRST." The data portion of this chunk contains a "FOO" subchunk. <pre>RIFF('QRST' F00(17 23))</pre> The following example describes an "ICOP" chunk containing the string "Copyright Encyclopedia International.": <pre>'ICOP' ("Copyright Encyclopedia International."Z)</pre>

Notation	Description														
<number>[<modifier>]	A number consisting of an optional sign (+ or -) followed by one or more digits and modified by the optional <modifier>. Valid <modifier> values follow: <table> <tr> <th><modifier></th><th>Meaning</th></tr> <tr> <td>none</td><td>16-bit number in decimal format</td></tr> <tr> <td>H</td><td>16-bit number in hexadecimal format</td></tr> <tr> <td>C</td><td>8-bit number in decimal format</td></tr> <tr> <td>CH</td><td>8-bit number in hexadecimal format</td></tr> <tr> <td>L</td><td>32-bit number in decimal format</td></tr> <tr> <td>LH</td><td>32-bit number in hexadecimal format</td></tr> </table>	<modifier>	Meaning	none	16-bit number in decimal format	H	16-bit number in hexadecimal format	C	8-bit number in decimal format	CH	8-bit number in hexadecimal format	L	32-bit number in decimal format	LH	32-bit number in hexadecimal format
<modifier>	Meaning														
none	16-bit number in decimal format														
H	16-bit number in hexadecimal format														
C	8-bit number in decimal format														
CH	8-bit number in hexadecimal format														
L	32-bit number in decimal format														
LH	32-bit number in hexadecimal format														

Several examples follow:

```
0
65535
-1
0L
4a3c89HL
-1C
21HC
```

Note that -1 and 65535 represent the same value. The application reading this file must know whether to interpret the number as signed or unsigned.

'<chars>'

A four-character code (32-bit quantity) consisting of a sequence of zero to four ASCII characters (<chars>) in the given order. If <chars> is less than four characters long, it is implicitly padded on the right with blanks. Two single quotes is equivalent to four blanks. Examples follow.

```
'RIFF'
'xyz'
''
```

<chars> can include escape sequences, which are combinations of characters introduced by a backslash (\) used to represent other characters. Escape sequences are listed in the following section.

Notation	Description
----------	-------------

"<string>"[<modifier>]

The sequence of ASCII characters contained in <string> and modified by the optional modifier <modifier>. The quoted text can include any of the escape sequences listed in the following section. Valid <modifier> values follow:

<modifier>	Meaning
none	No NULL terminator or size prefix.
Z	String is NULL-terminated
B	String has an 8-bit (byte) size prefix
W	String has a 16-bit (word) size prefix
BZ	String has a byte-size prefix and is NULL-terminated
WZ	String has a word-size prefix and is NULL-terminated

NULL-terminated means that the string is followed by a character with ASCII value 0. A size prefix is an unsigned integer, stored as a byte or a word in Intel format preceding the string characters, that specifies the length of the string. In the case of strings with BZ or WZ modifiers, the size prefix specifies the size of the string without the terminating NULL.

The various +
string formats
are described
in "Storing
Strings in
RIFF
Chunks," later
in this chapter.

Examples follow:

"No prefix, no NULL terminator"
"No prefix, NULL terminator"Z
"Byte prefix, NULL terminator"BZ

Escape Sequences for Four-Character Codes and String Chunks

The following escape sequences can be used in four-character codes and string chunks:

Escape Sequence	ASCII Value (Decimal)	Description
\n	10	Newline character
\t	9	Horizontal tab character
\b	8	Backspace character
\r	13	Carriage return character
\f	12	Form feed character
\\	92	Backslash
\'	39	Single quote
\"	34	Double quote
\ddd	Octal <i>ddd</i>	Arbitrary character

Extended Notation for Representing RIFF Form Definitions

When documenting RIFF forms that you create, use the notation listed in the preceding section along with the extended notation listed in the following table to unambiguously define the structure of the new form.

Notation	Description
<name>	A label that refers to some element of the file, where name is the name of the label. Examples follow: <pre> <NAME-ck> <GOBL-form> <bitmap-bits> <foo> </pre> Conventionally, a label that refers to a chunk is named <ckID-ck>, where <i>ckID</i> is the chunk ID. Similarly, a label that refers to a RIFF form is named <formType-form>, where <i>formType</i> is the name of the form's type.

Notation	Description
----------	-------------

<name> → elements

The actual data represented by <name> is defined as **elements**. This states that <name> is an abbreviation for **elements**. An example follows.

<GOBL-form> → RIFF ('GOBL'<form-data>)

This example defines label <GOBL-form> as representing a RIFF form with chunk ID "GOBL" and data equal to <form-data>, where <form-data> is a label that would be defined in another rule. Note that a label may represent any data, not just a RIFF chunk or form.

Note A number of atomic labels are defined in "Atomic Labels," later in this chapter. These labels refer to primitive data types.

<name:type>

This is the same as <name>, but it also defines <name> to be equivalent to <type>. This notation obviates the following rule:

<name> → <type>

This allows you to give a symbolic name to an element of a file format and to specify the element data type. An example follows:

<xyz-coordinate> → <x:INT> <y:INT> <z:INT>

This defines <xyz-coordinate> to consist of three parts concatenated together: <x>, <y>, and <z>. The definition also specifies that <x>, <y>, and <z> are integers. This notation is equivalent to the following:

<xyz-coordinate> → <x> <y> <z>

<x> → <INT>

<y> → <INT>

<z> → <INT>

[elements]

An optional sequence of labels and literal data. Surrounded by square brackets, it may be considered an element itself. An example follows:

<FOO-form> → RIFF('FOO' [<header-ck>] <data-ck>)

This example defines form "FOO" with an optional <header> chunk followed by a mandatory <data> chunk.

Notation	Description
<code>e1 e2 ... eN</code>	Exactly one of the listed elements must be present. An example follows: <pre><hdr-ck> → hdr(<hdr-x> <hdr-y> <hdr-z>)</pre> This example defines the “hdr” chunk’s data as containing one of <hdr-x>, <hdr-y>, or <hdr-z>.
<code>element...</code>	One or more occurrences of element may be present. Note that an ellipsis has this meaning only if it follows an element; in other cases (such as, <code>A B ... Z</code>), the ellipsis has its ordinary English meaning. If there is any possibility of confusion, an ellipsis should only be used to indicate one or more occurrences. An example follows: <pre><data-ck> → data(<count:INT> <item:INT>...)</pre> This example defines the data of the “data” chunk to contain an integer <count>, followed by one or more occurrences of the integer <item>.
<code>[element]...</code>	Zero or more occurrences of element may be present. An example follows. <pre><data-ck> → data(<count:INT> [<item:INT>]...)</pre> This example defines the data of the “data” chunk to contain an integer <count> followed by zero or more occurrences of an integer <item>.
<code>{ elements }</code>	The group of elements within the braces should be considered a single element. An example follows: <pre><blorg> → <this> <that> <other>...</pre> This example defines <blorg> to be either <this> or <that> or one or more occurrences of <other>. The next example defines <blorg> to be either <this> or one or more occurrences of <that> or <other>, intermixed in any way. <pre><blorg> → <this> {<that> <other>}...</pre>

Notation	Description
----------	-------------

struct { ... } name

A structure defined using C syntax. This can be used instead of a sequence of labels if a C header (include) file is available that defines the structure. The label used to refer to the structure should be the same as the structure's typedef name. An example follows:

```
<3D_POINT> → struct          {  
                        INT x;      // x-coordinate  
                        INT y;      // y-coordinate  
                        INT z;      // z-coordinate  
                    } 3D_POINT
```

Wherever possible, the types used in the structure should be the types listed in the following section, "Atomic Labels," because these types are more portable than C types such as int. The structure fields are assumed to be present in the file in the order given, with no padding or forced alignment.

Unless the RIFF chunk ID is "RIFX" integer byte ordering is assumed to be in Intel format.

// comment

An explanatory comment to a rule. An example follows:

```
<weekend> → 'Sat'|'Sun'    // 4-char code for day
```

Atomic Labels

The following are atomic labels, or labels that refer to primitive data types. Where available, the equivalent Microsoft C data type is also listed.

Label	Meaning	MS C Type
<CHAR>	8-bit signed integer	signed char
<BYTE>	8-bit unsigned quantity	unsigned char
<INT>	16-bit signed integer in Intel format	signed int
<WORD>	16-bit unsigned quantity in Intel format	unsigned int
<LONG>	32-bit signed integer in Intel format	signed long
<DWORD>	32-bit unsigned quantity in Intel format	unsigned long
<FLOAT>	32-bit IEEE floating point number	float
<DOUBLE>	64-bit IEEE floating point number	double
<STR>	String (a sequence of characters)	
<ZSTR>	NULL-terminated string	
<BSTR>	String with byte (8-bit) size prefix	
<WSTR>	String with word (16-bit) size prefix	
<BZSTR>	NULL-terminated string with byte size prefix	
<WZSTR>	NULL-terminated string with word size prefix	

NULL-terminated means that the string is followed by a character with ASCII value 0.

A size prefix is an unsigned integer, stored as a byte or a word in Intel format, that specifies the length of the string. In the case of strings with BZ or WZ modifiers, the size prefix specifies the size of the string without the terminating NULL.

Note The WINDOWS.H header file defines the C types BYTE, WORD, LONG, and DWORD. These types correspond to labels <BYTE>, <WORD>, <LONG>, and <DWORD>, respectively.

A Sample RIFF Form Definition and RIFF Form

The following example defines <GOBL-form>, the hypothetical RIFF form of type "GOBL." To fully document a new RIFF form definition, a developer would also provide detailed descriptions of each file element, including the semantics of each chunk and sample files documented using the standard notation.

```

<GOBL-form> → RIFF( 'GOBL'                                // RIFF form header
                  [<org-ck>]                                // Origin chunk (default (0,0,0))
                  <obj-list> )                             // Series of graphical objects

<org-ck> →      org( <origin:3D_POINT> )                   // Origin of object list

<obj-list> →    LIST('obj' { <sqr-ck> |                    // An object is: a square
                           <circ-ck> |                    // or a circle
                           <poly-ck> }... )               // or a polygon

<sqr-ck> →      sqr( <pt1:3D_POINT>                        // one vertex
                    <pt2:3D_POINT>                        // another vertex
                    <pt3:3D_POINT> )                      // a third vertex

<circ-ck> →     circ( <center:3D_POINT>                    // center of circle
                     <circumPt:3D_POINT> )               // a point on the circumference

<poly-ck> →     poly( <pt:3D_POINT>... )                  // list of points in a polygon

<3D_POINT> →    struct                                     // defined in "gobl.h"
    {
        INT x;                                           // x-coordinate
        INT y;                                           // y-coordinate
        INT z;                                           // z-coordinate
    } 3D_POINT

```

Sample RIFF Form

The following sample RIFF form adheres to the form definition for form type GOBL. The file contains three subchunks:

- An "INFO" list
- An "org" chunk
- An "obj" chunk

The "INFO" list and "org" chunk each have two subchunks. The "INFO" list is a registered global chunk that can be used within any RIFF file. The "INFO" list is described in the "The INFO List Chunk," later in this chapter.

Since the definition of the GOBL form does not refer to the INFO chunk, software that expects only “org” and “obj” chunks in a GOBL form would ignore the unknown “INFO” chunk.

```
RIFF( 'GOBL'
    LIST('INFO'          // INFO list containing filename and copyright
        INAM("A House"Z)
        ICOP("(C) Copyright Joe Inc. 1991"Z)
    )

    org (2, 0, 0)         // Origin of object list

    LIST('obj'           // Object list containing two polygons
        poly(0,0,0 2,0,0 2,2,0, 1,3,0, 0,2,0)
        poly(0,0,5 2,0,5 2,2,5, 1,3,5, 0,2,5)
    )

    // End of form
```

Storing Strings in RIFF Chunks

This section lists methods for storing text strings in RIFF chunks. While these guidelines may not make sense for all applications, you should follow these conventions if you must make an arbitrary decision regarding string storage.

NULL-Terminated String (ZSTR) Format

A NULL-terminated string (ZSTR) consists of a series of characters followed by a terminating NULL character. The ZSTR is better than a simple character sequence (STR) because many programs are easier to write if strings are NULL-terminated. ZSTR is preferred to a string with a size prefix (BSTR orWSTR) because the size of the string is already available as the `<ckSize>` value, minus one for the terminating NULL character.

String Table Format

In a string table, all strings used in a structure are stored at the end of the structure in packed format. The structure includes fields that specify the offsets from the beginning of the string table to the individual strings. An example follows:

```
typedef struct
{
    INT      iWidgetNumber;        // the widget number
    WORD     offszWidgetName;      // an offset to a string in <rgchStrTab>
    WORD     offszWidgetDesc;      // an offset to a string in <rgchStrTab>
    INT      iQuantity;           // how many widgets
    CHAR     rgchStrTab[1];        // string table (allocate as large as needed)
} WIDGET;
```

If multiple chunks within the file need to reference variable-length strings, you can store the strings in a single chunk that acts as a string table. The chunks that refer to the strings contain offsets relative to the beginning of the data part of the string table chunk.

NULL-Terminated, Byte Size Prefix String (BZSTR) Series

In a BZSTR series, a series of strings is stored in packed format. Each string is a BZSTR, with a byte size prefix and a NULL terminator. This format retains the ease-of-use characteristics of the ZSTR while providing the string size, allowing the application to quickly skip unneeded strings.

Multiline String Format

When storing multiline strings, separate lines with a carriage return/line feed pair (ASCII 13/ASCII 10 pair). Although applications vary in their requirements for new line symbols (carriage return only, line feed only, or both), it is generally easier to strip out extra characters than to insert extra ones. Inserting characters might require reallocating memory blocks or pre-scanning the chunk before allocating memory for it.

Choosing a Storage Method

The following lists guidelines for deciding which storage method is appropriate for your application.

Usage	Recommended Format
Chunk data contains nothing except a string	ZSTR (NULL-terminated string) format.
Chunk data contains a number of fields, some of which are variable-length strings	String-table format
Multiple chunks within the file need to reference variable-length strings	String-table format
Chunk data stores a sequence of strings, some of which the application may want to skip	BZSTR (NULL-terminated string with byte size prefix) series
Chunk data contains multiline strings	A multiline string format

LIST Chunk

A LIST chunk is defined as follows:

```
LIST( <list-type> [<chunk>]... )
```

A LIST chunk contains a list, or ordered sequence, of subchunks. The **<list-type>** is a four-character code that identifies the contents of the list.

If an application recognizes the list type, it should know how to interpret the sequence of subchunks. However, since a LIST chunk may contain only subchunks (after the list type), an application that does not know about a specific list type can still walk through the sequence of subchunks.

Like chunk IDs, list types must be registered, and an all-lowercase list type has meaning relative to the form that contains it.

The INFO List Chunk

The “INFO” list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An “INFO” list is a “LIST” chunk with list type “INFO.” The following shows a sample “INFO” list chunk:

```
LIST('INFO' INAM("Two Trees"Z)
      ICMT("A picture for the opening screen"Z) )
```

An “INFO” list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn’t understand. The chunks listed below may only appear in an “INFO” list. Each chunk contains a ZSTR, or null-terminated text string.

Chunk ID	Description
IARL	<i>Archival Location.</i> Indicates where the subject of the file is archived.
IART	<i>Artist.</i> Lists the artist of the original subject of the file. For example, “Michaelangelo.”
ICMS	<i>Commissioned.</i> Lists the name of the person or organization that commissioned the subject of the file. For example, “Pope Julian II.”
ICMT	<i>Comments.</i> Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do <i>not</i> include newline characters.

Chunk ID	Description
ICOP	<i>Copyright.</i> Records the copyright information for the file. For example, "Copyright Encyclopedia International 1991." If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	<i>Creation date.</i> Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example, "1553-05-03" for May 3, 1553.
ICRP	<i>Cropped.</i> Describes whether an image has been cropped and, if so, how it was cropped. For example, "lower right corner."
IDIM	<i>Dimensions.</i> Specifies the size of the original subject of the file. For example, "8.5 in h, 11 in w."
IDPI	<i>Dots Per Inch.</i> Stores dots per inch setting of the digitizer used to produce the file, such as "300."
IENG	<i>Engineer.</i> Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example, "Smith, John; Adams, Joe."
IGNR	<i>Genre.</i> Describes the original work, such as, "landscape," "portrait," "still life," etc.
IKEY	<i>Keywords.</i> Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example, "Seattle; aerial view; scenery."
ILGT	<i>Lightness.</i> Describes the changes in lightness settings on the digitizer required to produce the file. Note that the format of this information depends on hardware used.
IMED	<i>Medium.</i> Describes the original subject of the file, such as, "computer image," "drawing," "lithograph," and so forth.
INAM	<i>Name.</i> Stores the title of the subject of the file, such as, "Seattle From Above."
IPLT	<i>Palette Setting.</i> Specifies the number of colors requested when digitizing an image, such as "256."
IPRD	<i>Product.</i> Specifies the name of the title the file was originally intended for, such as "Encyclopedia of Pacific Northwest Geography."
ISBJ	<i>Subject.</i> Describes the contents of the file, such as "Aerial view of Seattle."
ISFT	<i>Software.</i> Identifies the name of the software package used to create the file, such as "Microsoft WaveEdit."

Chunk ID	Description
ISHP	<i>Sharpness.</i> Identifies the changes in sharpness for the digitizer required to produce the file (the format depends on the hardware used).
ISRC	<i>Source.</i> Identifies the name of the person or organization who supplied the original subject of the file. For example, "Trey Research."
ISRF	<i>Source Form.</i> Identifies the original form of the material that was digitized, such as "slide," "paper," "map," and so forth. This is not necessarily the same as IMED.
ITCH	<i>Technician.</i> Identifies the technician who digitized the subject file. For example, "Smith, John."

Device Independent Bitmap (DIB) File Format

The Device Independent Bitmap (DIB) format represents bitmap images in a device-independent manner. Bitmaps can be represented at 1, 4, and 8 bits per pixel, with a palette containing colors represented in 24 bits. Bitmaps can also be represented at 24 bits per pixel without a palette and in a run-length encoded format. This documentation describes three types of DIB files:

- Windows version 3.0 device-independent bitmap files
- OS/2 Presentation Manager version 1.2 device-independent bitmap files
- RIFF device-independent bitmap files

The Windows 3.0 and Presentation Manager 1.2 DIBs are similar, so they are discussed together.

Overview of DIB Structure

Windows 3.0 and Presentation Manager 1.2 DIB files contain the following sequence of data structures:

- A file header
- A bitmap information header
- A color table
- An array of bytes that defines the bitmap bits

The following sections describe each of these structures.

Bitmap File Header

The bitmap file header contains information about the type, size, and layout of a device-independent bitmap (DIB) file. In both the Windows 3.0 and Presentation Manager 1.2 DIBs, it is defined as a **BITMAPFILEHEADER** data structure:

```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

The following table describes the fields.

Field	Description
bfType	Specifies the file type. It must consist of the character sequence BM (WORD value 0x4D42).
bfSize	Specifies the file size in bytes.
bfReserved1	Reserved. Must be set to zero.
bfReserved2	Reserved. Must be set to zero.
bfOffBits	Specifies the byte offset from the BITMAPFILEHEADER structure to the actual bitmap data in the file.

Bitmap Information Header

The **BITMAPINFO** and **BITMAPCOREINFO** data structures define the dimensions and color information for Windows 3.0 and Presentation Manager 1.2 DIBs, respectively. They are defined as follows:

Windows 3.0 DIB	Presentation Manager 1.2 DIB
<pre>typedef struct tagBITMAPINFO { BITMAPINFOHEADER bmiHeader; RGBQUAD bmiColors[1]; } BITMAPINFO;</pre>	<pre>typedef struct _BITMAPCOREINFO { BITMAPCOREHEADER bmciHeader; RGBTRIPLE bmciColors[1]; } BITMAPCOREINFO;</pre>

These structures are essentially alike, and this section discusses them simultaneously. Each field name for the Windows **BITMAPINFO** structure is followed by the corresponding

Microsoft Media Platform: Player Framework

Search all Comments

Home Downloads Documentation Discussions Issue Tracker Source Code People License

View All Comments | Print View | Page Info | Change History (all pages)
None

Search

Microsoft Media Platform: Player Framework v2.5 (formerly Silverlight Media Framework)

Want to learn more? Check out the [SMF "How To" Documentation](#) for videos, online examples, sample code and more!

Overview

What is the Microsoft Media Platform: Player Framework (MMPF)?

Microsoft's open source media player framework - a component of the Microsoft Media Platform. This project was formerly known as the Silverlight Media Framework (SMF). enables developers to quickly deploy a robust, scalable, customizable media player for [IIS Smooth Streaming](#) delivery. The MMPF builds on the core functionality of the [Smooth Streaming Client](#) (formerly known as the "Smooth Streaming Player Development Kit") and adds a large number of additional features, including an extensibility API that allows developers to create plugins for the framework. The MMPF also now includes full support for Windows Phone 7 so developers can incorporate high-end video playback experiences in their Windows Phone 7 applications.

The MMPF is built on a proven code base with dozens of leading Smooth Streaming deployments, including Wimbledon, Sunday Night Football on NBCSports, the UEFA Super Cup on Canal+, Roland Garros and the Tour de France on France Télévisions and many others and by providing developers with source code they can more easily build these experiences for their deployments. Key features in the framework include DVR, rewind, alternate language tracks, in-stream data feeds and analytics tracking. The MMPF is designed for future third-party extensibility and component modularity, as well as support for other media delivery scenarios beyond Smooth Streaming.

New in version 2.x

The SMFv2 release introduces a whole new, more modular architecture with an extensive plug-in API for 3rd-party developers and partners. It will also enable developers to selectively exclude libraries that aren't needed in their projects in order to avoid unnecessary file size increases. The v2 framework will include plug-ins for [Fixed Text](#) (DFXP), URL frame linking, a metadata framework, support for the [Microsoft Silverlight Analytics Framework](#), support for multiple audio tracks, improved bitrate monitoring, support for Silverlight 4 global styling, a JavaScript API, a logging plug-in, and a host of other new features. It also takes advantage of the new [Managed Extensibility Framework \(MEF\)](#) in Silverlight 4. More information in this [thread](#) ...

Features

New in v2

- » Re-architected to be more modular, flexible
- » Plugin API
- » Accessibility support via W3C Timed Text standard for closed captioning (DFXP format)
- » Support for multiple audio tracks
- » Support for multiple SSME instances (used for multicam like in Sunday Night Football app)
- » URL frame linking
- » Improved bitrate monitoring
- » Support for Silverlight 4 global styling
- » Support for the Managed Extensibility Framework (MEF)
- » JavaScript API
- » Logging plugin
- » Support for the [Microsoft Silverlight Analytics Framework](#)
- » Documentation (API docs available now, detailed User Guide coming soon)
- » Media plugins (IIS Smooth Streaming, Progressive Download, Windows Media Streaming, etc)
- » Design-time DLL for improved authoring experience in Blend and Visual Studio
- » MSI installer so all the right goo gets put in all the right places
- » Smooth Streaming data graph overlay (used for demonstrating bitrate switching in live demos, useful for debugging)

New in SMF 2.2

- » Added support for Windows Phone 7, this includes a new set of binaries and a new Solution included in the source download
- » Updated SMF and SMF for Windows Phone to v1.1 of the Smooth Streaming Media Element
- » Fixed an issue in SMFPlayer that was preventing the databinding of SMFPlayer.CurrentPlaylistItem from working correctly
- » Added PlaylistItem.Duration
- » Added a fix to TimedTextMarkerProvider for an issue that was causing captions to only display for the first Playlist Item.
- » Fixed an issue with TimedTextMarkerProvider that was preventing captions from displaying when SMFPlayer.IsStartPositionOffset = true
- » In the previous release the Smooth Streaming and Progressive player examples were missing their HTML and ReadMe files, these have been added back for this release.

New in SMF 2.3

- » Added support for the latest version of the [IIS Smooth Streaming Client \(SSME v1.5\)](#)
- » Fixed several bugs
- » Added several API enhancements
- » Improved design templates for Windows Phone 7
- » See the downloads page for a full, detailed listing of changes in v2.3

235 people are

CURRENT: MMP
(form
DATE: Sun /
STATUS: Stabl
RATING: 1507
MORE: [View](#)

Recent review

The 2.4
the previous ones.
to... (more)

Activity

Page Views
Visits
Downloads
Application Runs
View

Related Projects

» [Microsoft Silv](#)

New in SMF 2.4

- » A new SMF plugin designed to capture health and diagnostic information for smooth streaming players. Captured information can then be sent to a variety of destinations including the new [Health Monitoring API](#).
- » New smooth streaming health monitoring tool to visualize the health of a smooth streaming player.
- » Sample app demonstrating how to capture basic health information locally within your player.
- » Sample app demonstrating how to capture health information and send it to a remote server.
- » See the downloads page for a full, detailed listing of changes in v2.4

New in SMF 2.5

- » Stereoscopic 3D support
- » Standards based advertising support
- » Fixed several bugs
- » Uses SSME 1.5 Update 1
- » See the downloads page for a full, detailed listing of changes in v2.5

Also includes all features from SMFv1.1 [more...](#)

Download

Click on the "Downloads" tab at the top of this page to access downloads for the SMFv2

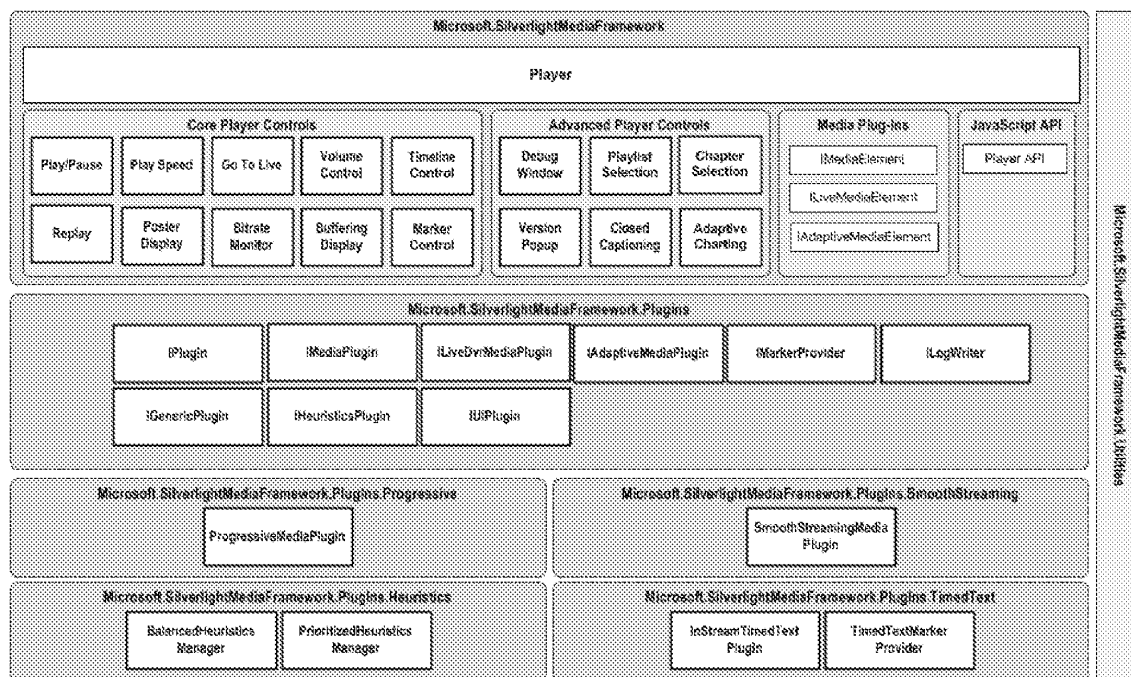
Planned for future releases:

- » Support for advertising standards like [VAST](#), [MAGI](#), and VPAID via the plugin API
- » Metadata framework

NOTE: SMFv2 and the [MS Smooth Streaming Client 1.0](#) require Silverlight 4 for deployment. For deployment to earlier versions of the runtime, take a look at [Silverlight Media Framework 1.1](#) in the "Downloads" section of this site.

SMF v.2 Architecture

Last Updated: July 7, 2010



Last edited May 3 at 8:57 PM by [jmcgregorfield](#), version 61

Microsoft Media Platform: Player Framework

[Home](#) | [Downloads](#) | [Documentation](#) | [Discussions](#) | [Issue Tracker](#) | [Source Code](#) | [People](#) | [License](#) | [RSS](#)
[Show Page Comments](#) | [Print View](#) | [Page Info](#) | [Change History \(all pages\)](#)
[Search Wiki & Documentation](#)
[Home](#) > [SMF v1.1](#)

Silverlight Media Framework v1.1

(Released Jan-2010)

The SMFv1.1 project took the purpose-built, high-end video player developed for the NBC Olympics site, cleaned up and refactored the code, and made it available to Silverlight developers via an open source Ms-PL license. Microsoft has released a new version of the framework that re-architects the purpose-built player and turns it into a modular, flexible player framework with an extensive plugin API. For more information see the SMFv2 page.

SMFv1.1 can be downloaded from the "Downloads" page.

Features

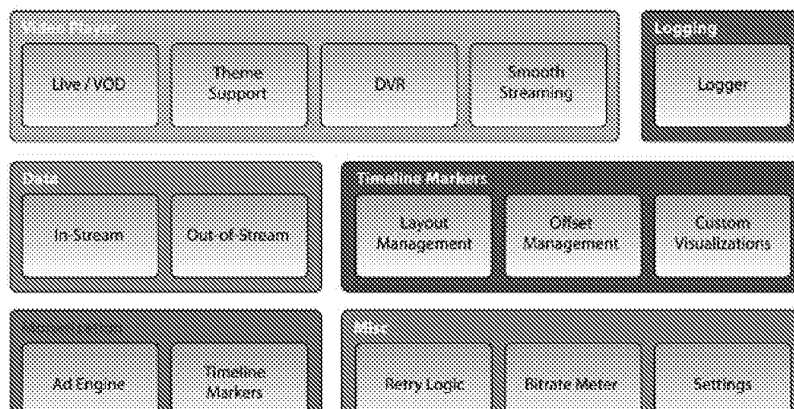
Player Elements

- » Play / pause
- » Rewind
- » Fast forward
- » Replay
- » Slow motion
- » Next chapter marker
- » Previous chapter marker
- » Timeline Scrubber
- » Current position / total position
- » Go to live
- » Bitrate meter
- » Volume
- » Full screen / restore

Advanced Features

- » Brand your player - Use Blend to re-invent how users experience your content!
- » Emphasize defining moments by embedding Markers in your Timeline.
- » Communicate and respond to bandwidth utilization by using the Bitrate element.
- » Integrate external data sources using the Data framework.
- » Deliver differentiated Live and On-Demand experiences.

SMFv1.1 Architecture Diagram



236 people are following this project ([follow](#))

Download

CURRENT MMP Player Framework 2.5 (formerly called SMF)

DATE Sun Apr 10 2011

STATUS Stable

RATING 1 rating

15074 downloads

MORE [View all downloads](#)

Recent reviews [more](#)

The 2.5 version is even better, that the previous ones. Great work, that saves you a lot... [\(more\)](#)

Activity 7 [All days](#)

Page Views	6075
Visits	1845
Downloads	1141
Application Runs	N/A

[View Detailed Stats](#)

Related Projects

- » [Microsoft Silverlight Analytics Framework](#)



How-To Videos

Live Smooth Streaming Tutorial - Playback (added 3/1/10)

<http://team.iis.net/page.aspx/801/live-smooth-streaming-tutorial---playback/>

Embedding The Video Player

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Embedding-Video-Player/>

Styling And Templating The Video Player

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Styling-Templating-Video-Player/>

Extending The Video Player

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Extending-Video-Player/>

Out Of Stream Data Access

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Out-Of-Stream-Data-Access/>

Using Application Settings

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Using-Application-Settings/>

Creating Custom Timeline Markers

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Creating-Custom-Timeline-Markers/>

Building A Custom Bitrate Meter

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Building-Custom-Bitrate-Meter/>

Using SMF Logging

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Using-SMF-Logging/>

Live Mode vs. VOD

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Live-Mode-vs-VOD/>

Hosting VOD using IIS Media Services

<http://www.silverlight.net/learn/videos/Silverlight-Media-Framework/Hosting-VOD-using-IIS-Media-Services/>

Additional Information

[Silverlight Media Framework announcement and future thinking](#)

Last edited Jun 7 2010 at 5:51 PM by [mdowney](#), version 4

Comments

No comments yet.

[Sign in to add a comment](#)



Quick Links | Home | Worldwide

Search Microsoft.com for:

Microsoft Windows XP Media Center Edition 2005

Windows XP Home

Security At Home | Microsoft At Home | Microsoft At Work

Windows XP Media Center Edition

Product Information

How to Buy

Upgrading

Using Media Center

Partners

Resources

Using Windows XP

Downloads

Support

Expert Zone

Worldwide Sites

Microsoft Update

Windows Marketplace

Windows Family





Frequently asked Questions

This page provides answers to frequently asked questions about Microsoft Windows XP Media Center Edition 2005.

If you do not find an answer to your question in this FAQ, try the following resources:

- To learn more about Windows XP Media Center Edition, download this [PDF datasheet](#).
- For more detailed information about Windows XP Media Center Edition 2005, read about its [features](#).
- For expert advice, free downloads, and other useful information about Windows XP Media Center Edition 2005, check out the [Expert Zone](#).
- To submit a question or chat with other technical enthusiasts, visit [Windows XP Media Center newsgroups](#).

Quick Links

- [Take me to another topic.](#)
- [Setting It Up](#)
- [Getting Started](#)
- [Beyond the Basics](#)

Get Our Newsletter

[Media Center and Windows tips.](#)

Sign up for the Exploring Windows newsletter and get the latest tips, tricks, and news for Windows XP and Media Center.

On this page

- General information
- Installation
- Playback

General information

What is Windows XP Media Center Edition 2005?

Windows XP Media Center Edition 2005, available preinstalled on Media Center PCs, allows you to pause and rewind live TV or radio, digitally record an entire TV series or program category, watch DVDs and videos, organize and play your music collection, and showcase digital photos. Windows XP Media Center Edition 2005 makes it easier to access digital entertainment wherever you are. Here are some of the top features:

- Multi-tuner support (record two shows at the same time and watch another previously recorded show)
- Support for digital and high-definition local TV
- Built-in CD and DVD burning
- Windows Movie Maker 2.1 with DVD burning
- Internet radio
- Enhanced setup wizard for easier installation

Is Windows XP Media Center Edition 2005 available in any other languages?

Localized versions are now available for Australia, Austria, China, France, Germany, Italy, Japan, Korea, Netherlands, Switzerland, and the United Kingdom.

Where can I purchase a Media Center PC with Windows XP Media Center Edition 2005?

You can purchase Media Center PCs from a variety of retailers or directly from the manufacturers. See the Windows XP Media Center Edition 2005 [How to buy page](#) for complete details.

Can I use the Media Center PC for my computing needs?

All Media Center PCs can run the same Microsoft programs as other Windows XP-based computers. You can use your Media Center PC to surf the Internet or to send and receive e-mail.

Can my local PC shop build a Media Center PC?

Starting this year, your local PC shop and other smaller manufacturers can offer Windows XP Media Center Edition 2005 preinstalled on new PCs.

Installation

What equipment will I need to install Media Center?

- Media Center PC.
- Monitor or TV.
- Keyboard.
- Mouse.
- Remote control and remote control receiver.
- Internet connection.
- TV signal connection from an antenna, cable, or satellite.
- Cables to connect the various components. For a complete list, consult the documentation that came with your Media Center PC.

Note: DVD Burning, TV features and remote control support may vary by manufacturer and/or require additional purchases.

What are the basic steps for setting up Media Center?

1. Decide where the components go. The new experiences in Windows XP Media Center Edition 2005 are designed around using the remote from the other side of the room; Media Center works well with the TV as the primary monitor.
2. Connect the peripherals. The simplest way to connect the peripherals is to set up a wireless network and add a wireless network card to your PC.
3. Hook up audio and video, and connect the devices. Media Center works with speakers from your PC, TV, or audio/video (A/V) system.
4. If you connect the PC to more than one display, configure the video card to show video on both screens at the same time.

To display the same video on both your TV and your PC monitor:

- Click **Start**, and then click **Control Panel**.
- Click **Appearance and Themes**, and then click **Display**.
- On the **Settings** tab, click **Advanced**.
- On the **View** tab, click **Clone**.

I want to set up my Media Center in my family room. What should I use to make the job easier?

- USB extension cable for extending USB keyboard, mouse, and remote sensor
- SVG extension cable for connecting monitor and PC
- RG6 coaxial cable for connecting cable or satellite feed to PC
- Wireless keyboard and mouse, or keyboard and mouse extension cables
- Remote control and remote control receiver.

- Optical mouse (if you don't want to use a mouse pad)
- Composite video cable or SVGA cable for connecting PC to TV

For more information, review the information on TV connection cables and on connecting your TV to your Media Center PC.

Do I need to purchase a TV tuner when I buy a Media Center PC?

Many Media Center PCs come equipped with a single TV tuner that captures your TV signal from a cable, satellite, or antenna source. Media Center PCs can also be equipped to handle multiple tuners. For example, if your Media Center PC comes with more than one TV tuner, you can record two shows while you watch a third recorded show.

If your Media Center PC is not equipped with a TV tuner, it is possible to upgrade.

Will Windows XP Media Center PCs support surround sound and work with my existing A/V system?

Many Media Center PCs support multi-channel output for speakers and connected A/V devices (using digital interconnect outputs such as SPDIF or Toslink). Media Center PCs also deliver Dolby Digital, DTS, or PCM surround decoding.

What kind of displays can I connect to a Windows XP Media Center PC?

The Windows XP Media Center PC can connect to a variety of displays:

- Media Center PC and desktop monitor. With computing and entertainment functionality in one place, this setup is ideal for locations where space is at a premium.
- Media Center PC, desktop monitor, and TV. Use the desktop display for computing and the TV for entertainment.
- Media Center PC and high-resolution display. A high-resolution display, such as a plasma or projection TV, provides sufficient resolution for desktop tasks, like reading e-mail or surfing the Internet, and offers a home theater style experience for TV, home movies, and DVDs. When your Media Center PC is connected to a widescreen display, 16:9 support lets you toggle between normal, zoom, and stretch video modes.

What is a Media Center Extender?

Windows Media Center Extender devices are used to connect Media Center PCs with TVs and monitors in the home, extending Media Center experiences to any room in the house. You can also connect an Xbox 360 with Media Center Extender technology to your PC running Microsoft Windows XP Media Center Edition 2005 through your wired or wireless home network, and access all of your TV, music, photos, movies, and Online Spotlight content through the Xbox 360.

Can I connect a new PC running Windows XP Media Center Edition 2005 to a work network or domain?

While you can access network resources on a work network or a domain, you cannot join a Windows XP Media Center Edition 2005 PC to the domain. PCs running Windows XP Media Center Edition 2005 are designed specifically for home use. Windows XP Professional features, specifically Domain Join and Cached Credentials (Credentials Manager for logons) are not included. As a result, you will be prompted for your logon user name and password to access network resources after you reboot or log back on to the PC. In addition, file shares or network resources that are set to require a domain-joined PC for access will not be available. Remote Desktop and Encrypting File System support are still included.

Playback

Can I use my computer and watch TV at the same time?

Yes. You can watch TV while using your computer at the same time.

What does the Media Center remote do?

From the comfort of your easy chair or across your living room, you can use the remote to browse Media Center, including the TV program guide listings, recorded TV, music, photos, and videos. You can also make your selections by artist, show or album title, or genre. When you're not using the remote, you can also access Media Center by using a mouse and keyboard.

What are the capabilities of the Media Center TV and Personal Video Recorder (PVR) features?

You can have Media Center record TV based on selections you make. You can select shows to record based on

keywords, categories, actors, or directors.

What are the new TV features?

With multiple TV tuners, you can record up to the same number of shows as you have tuners, while watching a recorded TV show. It's your decision whether to record a single show or an entire series. Here are some more new features:

- Use Movie Finder to browse top-rated movies, new releases, and movies playing right now or showing in the next hour.
- Get rid of recording conflicts to ensure that Media Center records the shows that you want.
- Use the My TV page to display recent recordings and upcoming recordings.
- Press the MORE INFO button on your remote to record the show, launch Messenger, or get help.

For complete information, check out the Windows Media Center features page.

How do I find out what's on TV?

Using Media Center's Program Guide, you can view and scroll through a listing of current TV shows that is periodically downloaded from your Microsoft TV listings service provider. The Guide displays channel and network information, along with the title and scheduled broadcast time of each show. When you select a show title, a short description of the show displays at the bottom of the screen.

Can I record multiple programs at once, or watch one show while recording another? Can I watch a recorded show while Media Center is recording another show?

Some Media Center PCs are equipped to handle multiple tuners. For example, if your Media Center PC comes with two TV tuners, you can record two shows while watching a third recorded show. Check with the manufacturer to find out how many TV tuners are included in your Media Center PC.

What happens if I try to record multiple shows at once and I don't have enough tuners to support it?

Media Center uses advanced conflict resolution features to help you prioritize recordings or find alternate airings of programs when preset recordings conflict. Here's how the system works.

A recording conflict notification appears:

- If you have one TV tuner enabled, and you schedule a recording for the same time as another recording.
- If you have more than one TV tuner enabled, and you schedule more recordings than you have tuners available at one time.

Follow the on-screen instructions to resolve the conflict when prompted. If you do not resolve the conflict, Media Center resolves the conflict automatically and selects a show to record. Sometimes Media Center can locate an alternate date and time for one of the shows and adjusts the recording schedule to record both shows.

Otherwise, Media Center selects which show to record based on the following criteria:

- A one-time recording (not part of a series)
- Series priority ranking (Series, Keyword, and Manual recordings can all be ranked in Series Priorities)
- The show that was scheduled first

If Media Center encounters a recording conflict between two episodes from different series recording requests, Media Center uses the Series Priorities list to decide which show to record. You can rank the series you most want recorded at the top of the Series Priorities list to help ensure that it is successfully recorded in the case of conflicts. When you schedule a new series, it is added to the bottom of the list and has the lowest priority.

Does Windows XP Media Center Edition 2005 support HDTV? What do I need?

Windows XP Media Center Edition 2005 is the first PC operating system to support playback, pause, and recording of free, over-the-air high-definition television (HDTV) right out of the box. Just connect the Media Center PC that has HDTV support to an antenna that can receive a high-definition TV signal, and get up to six times the image quality of standard TV. Media Center PCs with HDTV support are available through various manufacturers and retailers.

Can I receive HDTV over cable or satellite?

Not at the current time. You can use standard connectors such as S-Video to connect your cable box or satellite receiver to your Media Center PC, and then watch and record high-definition TV programs. However, the HDTV programs will be in standard TV format. In order to watch and record a true HDTV signal using Media Center, you must have access to local channels broadcast over the air and receive these freely available signals with an HDTV antenna. For more information on HDTV antennas and over-the-air reception, please visit the [CEA Antennaweb.org](http://www.cea-antennaweb.org) Web site.

What file format does Media Center record in?

Media Center uses a new file format called DVR-MS. DVR-MS creates an MPEG-2 file that includes metadata about the recorded program to be stored.

How many hours of TV can I store on my Media Center PC?

Media Center offers four levels of recording quality—Fair, Good, Better, and Best. The quality level affects the amount of disk space needed: the higher the quality level, the larger the recorded TV file, and the more disk space needed.

The approximate amount of disk space that each recording level will consume is:

- Fair: 1 GB/hour
- Good: 2 GB/hour
- Better: 2.5 GB/hour
- Best: 3 GB/hour

What is content protection, and how does Media Center use it?

Content owners and/or broadcasters can set copy protection flags to indicate that a program is subject to content protection. When Media Center detects that this flag is set, it will protect the content by limiting the ability to copy and distribute the program.

Can I watch copy-protected recorded TV files on another PC?

Protected files can be watched only on the Media Center PC that originally recorded the content.

Can I watch unprotected recorded TV files on another PC?

Yes, as long as that PC can play DVR-MS files, either through Media Center or an updated Microsoft DirectShow player such as Microsoft Windows Media Player 10. Watch what you want when you want.

Can I play back copy-protected recorded TV files on the same Media Center PC by using Windows Media Player 10 or other applications that work with DirectShow?

Protected files can be watched only through Media Center on the Media Center PC that originally recorded the content.

Can I burn DVDs to watch using a dedicated DVD player?

Yes. Windows XP Media Center Edition 2005 includes built-in support for burning video and data DVDs from photos, music, video, and TV content on a wide range of media types and formats. DVDs can be burned in one of the following formats:

- Data DVD: Designed for storing files on disc, such as creating a backup of music or photos.
- Video DVD: Designed for converting recorded TV or home movies into video for playback on most home DVD players.
- DVD slide show: Designed for creating a slide show with pictures and music for playback on consumer DVD players and computers equipped with DVD players.

In addition, you can continue to use your Media Center PC normally while you burn DVDs in the background.

How can I enjoy Windows XP Media Center Edition 2005 content around the home or on portable devices?

Access your digital media anywhere in the home with Windows Media Center Extender devices, using your Media Center PC as your entertainment hub. Or go mobile with Portable Media Center running Windows Mobile software and enjoy high-quality digital video, recorded TV, movies, music, and photos—anywhere, anytime.

Do Media Center Extender devices support wireless home networks?

Media Center PCs and Media Center Extender devices can both be connected to wireless home networks, although certain configurations may not be supported. See the [Media Center Extender](#) page for additional information.

What are the capabilities of the My Music feature in Media Center?

Media Center allows you to save playlists, copy CDs to your hard drive, burn CDs, and specify the file format that best suits your needs. With your music collection in one place, you can use the remote to browse your music by album, artist, song, or genre.

What are the capabilities of the Media Center Radio features?

The Media Center Radio features let you tune in to FM broadcasts and control the way you listen. Just like with TV, if you're interrupted while listening, you can pause (up to 30 minutes), rewind, and skip ahead in radio broadcasts. Or, press the REPLAY button on your remote to skip back 7 seconds, so you can catch the title of a tune.

What are the capabilities of the My Pictures and My Videos features in Media Center?

With My Pictures you navigate thumbnail images or launch a slide show of your photos. My Videos gives you greater control when viewing home movies or videos.

What is Online Spotlight, and what are the experiences available within this feature?

Online Spotlight lets you connect to the latest news, movies, and other content by using your remote. You can rent and watch digital movies on demand, find and download new music, watch movie previews, play games, and more. (Additional fees may apply.)

What information is sent back to Microsoft about my viewing/listening habits? What information is shared with partners?

Microsoft is committed to protecting consumer privacy. Microsoft does not collect personal information about your use of Media Center features, such as your name, home address, phone number, e-mail address, credit card number, Social Security number, or driver's license number. Anonymous and aggregate usage information may be shared with partners. For more information on Media Center terms of use and the complete privacy statement, visit [the privacy section](#).

Are any extra fees required to activate Windows XP Media Center Edition 2005?

Some partners that can be found in Online Spotlight may charge for their services. Microsoft does not.



[Manage Your Profile](#) | [Contact Us](#) | [Free Newsletter](#) | [RSS Feeds](#)

© 2007 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)



[Quick Links](#)[Home](#)[Worldwide](#)

Search Microsoft.com for:

Microsoft Windows XP Media Center Edition 2005

[Windows XP Home](#)[Security At Home](#)[Microsoft At Home](#)[Microsoft At Work](#)[Windows XP Media Center Edition](#)[Product Information](#)[How to Buy](#)[Upgrading](#)[Using Media Center](#)[Partners](#)

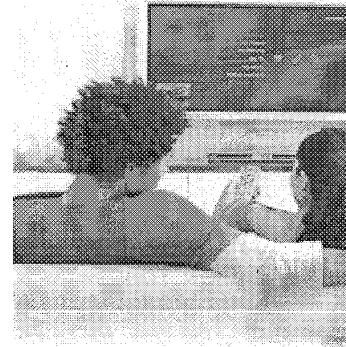
Resources

[Using Windows XP](#)[Downloads](#)[Support](#)[Expert Zone](#)[Worldwide Sites](#)[Microsoft Update](#)[Windows Marketplace](#)[Windows Family](#)

Media Center PC Features

Access more entertainment with less effort

Microsoft Windows XP Media Center Edition 2005 lets you do everything other Windows XP PCs do—and a whole lot more. Browse the Web, play your favorite PC games, e-mail and instant message your friends, and install and use programs designed for Windows XP. Media Center also delivers a powerful yet familiar way to enjoy all of your digital entertainment—photos, music, TV, movies, home videos, radio, and a world of applications and services whether you're sitting in front of your Windows desktop or across the room with a remote control. Media Center is your all-in-one PC and home entertainment center.

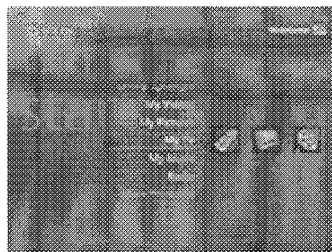


Play DVDs



Enjoy DVD movies on your Media Center PC. You'll get scan DVD playback with online chapter lookup on Mac equipped with DVD decoders manufactured by Media (standard hardware on new PC configurations).

My TV



My TV offers a simple way to browse, watch, and record multiple channels at the same time (both analog and digital TV features work with an antenna, cable, or satellite TV). fast and easy setup. My TV includes Live TV, Recorder Guide, Guide Search, and the new Movie Finder2 service, monthly service fee.

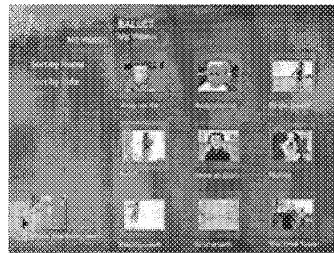
My Music



My Music offers access to your personal and online music. It's easier than ever to play a music CD, copy it into your library, or download music from the Internet.

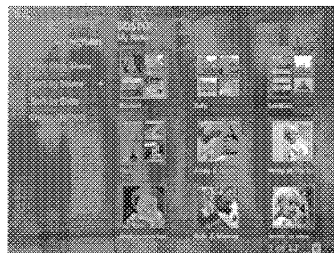
create a playlist, and edit album details.

My Video



My Videos offers an alternate view of your My Videos in Windows XP, organizing files and folders by date or by name and displaying a thumbnail image for each clip. My Videos supports content including standard avi, .mpg, and .wmv files.

My Pictures



My Pictures provides a fun new way to instantly enjoy digital photos. Download pictures from your digital camera, create a near-instant slide show, burn photos to CD—all with your remote control.

Copy to CD/DVD



Burn recorded TV to CDs and DVDs with built-in CD/DVD burning. Share pictures and home movies, and burn CD mixes.

Online Spotlight

Online Spotlight is your portal into a wide and growing range of on-demand entertainment. Online Spotlight offers a quick and easy access to a wide range of music and radio services, movie and TV services, as well as information services which give you instant access to customized entertainment programming online.¹

Sync to Device



Enjoy hours of TV, music, video, and digital photos on the go by transferring your favorite media to a Portable Media Center, Pocket PC, music player, or other storage devices, such as hard disk drives.

■ Remote Control



The Media Center remote control keeps all of the entertainment at your computer within easy reach, complementing the keyboard.

■ Movie Finder Service

The new Movie Finder² makes it easy to find movies you love. Get the most value out of your content package by easily discovering and recording movies already available on your lineup, or learn how to find movies or old favorites with online movie services.

■ Over-the-Air Digital TV/HDTV Local TV Recording (US only)

Enjoy true digital and high-definition local TV programming and personal video recording. Windows Media Center Edition 2005 now supports full live TV and personal video recording support for ATSC Digital TV (720p/1080i) local network TV programming delivered via UHF antenna.^{3,4}

■ My Radio

My Radio offers instant access to your Internet or FM radio services and keeps them all in one place. FM radio listeners have simple access to radio presets just like a car stereo and can even pause minutes of live radio.

■ More Programs

More Programs offers simple organization and access to applications and services designed for Media Center. More Programs are installed and available on your Media Center PC. The More Programs menu option organizes applications and services designed for Media Center in one place.

New to Update Rollup 2 for Windows XP Media Center Edition 2005

Support for Windows Media Center Extender for Xbox 360

Windows Media Center Extender technology built into every Xbox 360 lets you access entertainment from your Media Center PC—live and recorded TV and movies, music, photos and Online Spotlight—on a second screen.

■ Download Update Rollup 2 for Windows XP Media Center Edition 2005

1. To enjoy all the benefits of Windows XP, Internet access is recommended; for some features, such as the Electronic Programming Guide, Movie Finder, and Online Spotlight, it is required.
2. This feature available only in the U.S. and Canada.
3. DTV/HDTV quality via over-the-air only. Digital Satellite and Cable HD programming can be viewed in HD TV quality via set-top box pass-thru.

4. Supports up two (2) analog, one (1) DTV/HDTV tuner for local channels via over-the-air signal, other supported regions can support up to two (2) tuners per computer.



 [Printer-Friendly Version](#)  [Send This Page](#)  [Add to Favorites](#)

[Manage Your Profile](#) | [Contact Us](#) | [Free Newsletter](#) | [RSS Feeds](#)

© 2007 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)



TEXT OF THE FOURTH OFFICE ACTION

The examination is conducted on the basis of the observations and amended application documents submitted by the applicant on April 16, 2014. After perusing the aforesaid documents, the examiner continued the examination on the present application and set forth the following opinions:

1. Claim 1 claims a method of encoding a media file for playing back. The cited document CN1169229A (published on December 31, 1997, hereinafter referred to as D1) discloses an encoding method for digital image signal, and in particular makes public the following technical features (see line 5 on page 4 to line 6 on page 8 and line 8 on page 12 to line 6 on page 15 of the description in D1): the television transport stream is demultiplexed to obtain the PES stream corresponding to the television video information; the PES stream is decoded to extract from the packets, the video stream payload; for each TP speed, low-resolution, intra-coded frames are constructed from a received transport stream; since an intra coded I frame is to be constructed only intra-slices are extracted; intra-slices contain only intra-coded macroblocks; the reconstructed TP I frame is of lower spatial resolution than the original stream; the low resolution TP I frame utilizes a subset of the macroblocks from the original frame with the remaining non-selected macroblocks being discarded; the television transport stream is buffered by buffer 15, which generates signal 10, a transport stream for normal play speed processing; normal play transport stream 10 is coupled to multiplexor; the TP I frame packets, 5x signal 121, 18x signal 131 and 35x signal 141, are coupled to multiplexor which inserts the I-frame packets for each TP speed into the normal play transport stream. As can be seen, the technical solution claimed by claim 1 differs from the contents disclosed in D1 in that: a trick-play track to enable the visual-search of the video track, and the key frame corresponds to a frame in the encoded content that is not a key frame. Based on the above distinguishing technical feature, the technical problem to be actually solved by claim 1 is the search of the video frame.

The cited document US5887110A (published on March 23, 1999, hereinafter referred to as D2) discloses a method for producing fast playback video data, and in particular makes public the following technical features (see line 5 on column 1 to line 18 on column 2 and line 10 to line 56 on column 8 of the description and Figs.2-10 of D2): playback video data includes a normal playback mode and a fast playback (visual search) mode; normal playback coded video data and fast playback coded video data are stored in advance in the video server, and the foregoing encoded data to be used for producing the video data at a normal speed; the fast playback coded video data are encoded by using the intra-frame coding that enables random access at any desired frame (equivalent to the visual-search of the video track); the decoded video data are encoded at the second encoder by using the I-frame, the key frames such as I1' and I3' are produced by using the frames such as f5' and f9', to obtain the fast playback coded video data. It can be seen that the aforementioned distinguishing technical feature has been disclosed in D2, and it plays the same role in D2 as it does in claim 1, i.e. to search the video

frame. As can be seen, it would be obvious for those skilled in the art to reach the technical solution of claim 1 by comparing D1 with D2. Therefore, the technical solution claimed by claim 1 neither possesses prominent substantive features nor represents any notable progress, and thus lacks inventiveness as prescribed in Article 22, paragraph 3 of the Chinese Patent Law.

2. The additional technical feature of claim 2 has been disclosed in D1 (see line 32 on page 4 to line 13 on page 6 of the description in D1): the reconstructed TP I-frame is of lower spatial resolution than the original stream; the low resolution TP I frame utilizes a subset of the macroblocks from the original frame with the remaining non-selected macroblocks being discarded." Therefore, given that claim 1 to which claim 2 refers lacks inventiveness, dependent claim 2 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

3. D1 discloses the following content (see line 32 on page 4 to line 13 on page 6 of the description in D1): the reconstructed TP I-frame is of lower spatial resolution than the original stream. And it is a common sense in the art to reduce the resolution of the media content by a particular factor, e.g. multiplying by a particular factor, to generate the video for playing back. Therefore, given that claim 2 to which claim 3 refers lacks inventiveness, the claim 3 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

4. D1 discloses the following contents (see lines 18 to 29 on page 12 of the description in D1): "speed-dependent temporal subsampling is implemented on signal; at N times trick-play speed, the frame rate is reduced from 30 Hz to $30/2N$ Hz." And it is common sense in the art to reduce the rate of the frame by a particular factor, e.g. multiplying by a particular factor, to generate the video for playing back. Therefore, given that claim 2 to which claim 4 refers lacks inventiveness, claim 4 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

5. The additional technical feature of claim 5 has been disclosed in D1 (see line 25 on page 7 to line 6 on page 8 of the description in D1): normal play transport stream 10 is coupled to multiplexor; the TP I-frame packets, 5x signal 121, 18x signal 131 and 35x signal 141, are coupled to multiplexor which inserts the I-frame packets for each TP speed into the normal play transport stream. Therefore, given that claim 4 to which claim 5 refers lacks inventiveness, claim 5 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

6. It is common sense in the art to realize corresponding play speed by embedding the play track into the normally played media file. Therefore, given that claim 4 to which claim 6 refers lacks inventiveness, claim 6 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

7. It is common sense in the art to contain only video data in the play track so as to improve

the speed of play. Therefore, given that claim 1 to which claim 7 refers lacks inventiveness, claim 7 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

8. The additional technical feature of claim 8 has been disclosed in D1 (see lines 16 to 24 on page 7 of the description in D1): at 18x speed third or fourth I-frame is used, and at 35x speed seventh I-frame is used. Therefore, given that claim 1 to which claim 8 refers lacks inventiveness, claim 8 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

9. It is common sense in the art to re-encode the content to create a compressed video track processing the same resolution and frame, and the play track file is smaller than the original file so as to reduce the video file. Therefore, given that claim 1 to which claim 8 refers lacks inventiveness, claim 9 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

10. Compressing the original media file content is a customary technical means in encoding audio and video files. Therefore, given that claim 1 to which claim 8 refers lacks inventiveness, claim 10 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

11. Claim 11 claims a method of decoding a media file for playing back. D1 discloses a reproduction method for digital image signal, and in particular makes public the following technical features (see line 5 on page 4 to line 6 on page 8 and line 8 on page 12 to line 6 on page 15 of the description in D1): the television transport stream is demultiplexed to obtain the PES stream corresponding to the television video information; the PES stream is decoded to extract from the packets, the video stream payload; for each TP speed, low-resolution, intra-coded frames are constructed from a received transport stream; since an intra coded I frame is to be constructed only intra-slices are extracted; intra-slices contain only intra-coded macroblocks; the reconstructed TP I frame is of lower spatial resolution than the original stream; the low resolution TP I frame utilizes a subset of the macroblocks from the original frame with the remaining non-selected macroblocks being discarded; the television transport stream is buffered by buffer 15, which generates signal 10, a transport stream for normal play speed processing; normal play transport stream 10 is coupled to multiplexor; the TP I frame packets, 5x signal 121, 18x signal 131 and 35x signal 141, are coupled to multiplexor which inserts the I-frame packets for each TP speed into the normal play transport stream." As can be seen, the technical solution claimed by claim 11 differs from the contents disclosed in D1 in that: the trick-play track using a frame in the compressed content that is not a key frame to create key frame, the key frame corresponds to a frame in the compressed content that is not a key frame; playing back the compressed content, and decoding frames of the trick-play track at a rate proportional to a visual-search speed and from a location determined by the portion of the compressed content most recently played back. Based on the above distinguishing technical feature, the technical problem to be actually solved by claim 11 is the play of the video frame.

D2 discloses a method for producing fast playback video data, and in particular makes public the following technical features (see line 5 on column 1 to line 18 on column 2 and line 10 to line 56 on column 8 of the description and Figs.2-10 of D2): playback video data includes a normal playback mode and a fast playback (visual search) mode; normal playback coded video data and fast playback coded video data are stored in advance in the video server, and the foregoing encoded data to be used for producing the video data at a normal speed; the fast playback coded video data are encoded by using the intra-frame coding that enables random access at any desired frame (equivalent to the visual-search of the video track); the decoded video data are encoded at the second encoder by using the I-frame, the key frames such as I1' and I3' are produced by using the frames such as f5' and f9', to obtain the fast playback coded video data. It is common sense in the art to play compressed content, decode and play video file from a selected location according to actual design requirement. Therefore, it would be obvious for those skilled in the art to reach the technical solution of claim 11 by combining D1 with D2 and on basis of common sense. Therefore, the technical solution claimed by claim 11 neither possesses prominent substantive features nor represents any notable progress, and thus lacks inventiveness as prescribed in Article 22, paragraph 3 of the Chinese Patent Law.

12. The additional technical feature of claim 12 has been disclosed in the cited document WO0235832A2 (published on May 2, 2002, hereinafter referred to as D3) (see line 10 on page 5 to line 35 on page 6 of the description in D3): in a trick-mode display, only specific frames are displayed to the viewer, and the frames specific for trick-mode display are typically I-frames. Therefore, given that claim 11 to which claim 12 refers lacks inventiveness, claim 12 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

13. The additional technical feature of claim 13 has been disclosed in D3 (see lines 15-35 on page 6 of the description in D3): the I-frames contain differing amounts of data and therefore require different delivery intervals before being available for display; the amount of data it reads at the time of playing may encompass not only an I-frame but portions of neighboring P-frames or B-frames. Therefore, given that claim 11 to which claim 13 refers lacks inventiveness, claim 13 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

14. It is common sense in the art to only decode key-frames or ignore relevant amount of delta frames and key-frames so as to realize specific play speed. Therefore, given that claim 11 to which claims 14 and 15 refer lacks inventiveness, claims 14 and 15 also involve no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

15. The cited document US5717816A (published on February 10, 1998, hereinafter referred to as D4) discloses the following contents (see lines 52 to 65 on column 4 and lines 35-65 on column 5 of the description in D4): performing variable length encoding on received I-frames to generate reduced resolution I-frames. It is common sense that a video file has partial images at a reduced frame rate so as to compress the video file. Therefore, given that claim 11

to which claim 16 refers lacks inventiveness, claim 16 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

16. Claim 17 claims a system for playback of a media file. D1 discloses an encoding method for digital image signal, and in particular makes public the following technical features (see line 5 on page 4 to line 6 on page 8 and line 8 on page 12 to line 6 on page 15 of the description in D1): the television transport stream is demultiplexed to obtain the PES stream corresponding to the television video information; the PES stream is decoded to extract from the packets, the video stream payload; for each TP speed, low-resolution, intra-coded frames are constructed from a received transport stream; since an intra coded I frame is to be constructed only intra-slices are extracted; intra-slices contain only intra-coded macroblocks; the reconstructed TP I frame is of lower spatial resolution than the original stream; the low resolution TP I frame utilizes a subset of the macroblocks from the original frame with the remaining non-selected macroblocks being discarded; the television transport stream is buffered by buffer 15, which generates signal 10, a transport stream for normal play speed processing; normal play transport stream 10 is coupled to multiplexor; the TP I frame packets, 5x signal 121, 18x signal 131 and 35x signal 141, are coupled to multiplexor which inserts the I-frame packets for each TP speed into the normal play transport stream; a carrier, modulated responsive to a video signal, is received by receiver; the modulated carrier may be sourced from a cable; receiver demodulates and processes the received carrier to produce transport stream. The technical solution claimed by claim 17 differs from the contents disclosed in D1 in that: the trick-play track having at least one frame in common with the original media file, and being smaller in file size than the original media file; the trick-play track is to perform the visual-search of video track; the key frame corresponds to a frame in the encoded content that is not a key frame; a client processor in network connection with the media server and configured to send requests for the original media file to the media server. Based on the above distinguishing technical feature, the technical problem to be actually solved by claim 17 is the search and play of the video frame.

D2 discloses a method for producing fast playback video data, and in particular makes public the following technical features (see line 5 on column 1 to line 18 on column 2 and line 10 to line 56 on column 8 of the description and Figs.2-10 of D2): playback video data includes a normal playback mode and a fast playback (visual search) mode; normal playback coded video data and fast playback coded video data are stored in advance in the video server, and the foregoing encoded data to be used for producing the video data at a normal speed; the fast playback coded video data are encoded by using the intra-frame coding that enables random access at any desired frame (equivalent to the visual-search of the video track); the decoded video data are encoded at the second encoder by using the I-frame, the key frames such as I1' and I3' are produced by using the frames such as f5' and f9', to obtain the fast playback coded video data. It is common sense in the art that, according to actual design requirement, the trick-play track has the frame in common with the original media file, and is smaller in file size than the original media file so as to reduce the space of video file. It is also common sense in the art that a user side in network connection with a server. Therefore, it would be obvious for those skilled in the art to reach the technical solution of claim 17 by combining D1 with D2

and on basis of common sense. Therefore, the technical solution claimed by claim 17 neither possesses prominent substantive features nor represents any notable progress, and thus lacks inventiveness as prescribed in Article 22, paragraph 3 of the Chinese Patent Law.

17. It is common sense in the art to decode the video file at a user side and to play the corresponding content in comply with user instructions. Therefore, given that claim 17 to which claim 18 refers lacks inventiveness, claim 18 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

18. The additional technical feature of claim 19 has been disclosed in D3 (see line 10 on page 5 to line 35 on page 6 of the description in D3): in a trick-mode display, only selected frames are displayed to the viewer, and the frames selected for trick-mode display are typically I-frames. Therefore, given that claim 18 to which claim 19 refers lacks inventiveness, claim 19 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

19. Claim 20 claims a media player. D1 discloses an encoding method for digital image signal, and in particular makes public the following technical features (see line 5 on page 4 to line 6 on page 8 and line 8 on page 12 to line 6 on page 15 of the description in D1): the television transport stream is demultiplexed to obtain the PES stream corresponding to the television video information; the PES stream is decoded to extract from the packets, the video stream payload; for each TP speed, low-resolution, intra-coded frames are constructed from a received transport stream; since an intra coded I frame is to be constructed only intra-slices are extracted; intra-slices contain only intra-coded macroblocks; the reconstructed TP I frame is of lower spatial resolution than the original stream; the low resolution TP I frame utilizes a subset of the macroblocks from the original frame with the remaining non-selected macroblocks being discarded; the television transport stream is buffered by buffer 15, which generates signal 10, a transport stream for normal play speed processing; normal play transport stream 10 is coupled to multiplexor; the TP I frame packets, 5x signal 121, 18x signal 131 and 35x signal 141, are coupled to multiplexor which inserts the I-frame packets for each TP speed into the normal play transport stream; an user installation can realize the output of digital signal and control the video-on-demand service; a carrier, modulated responsive to a video signal, is received by receiver; the modulated carrier may be sourced from a cable; a receiver demodulates and processes the received carrier to produce transport stream. The technical solution claimed by claim 20 differs from the contents disclosed in D1 in that: a trick-play track includes at least one key frame created using a frame in the content that is not a key frame, the at least one key frame corresponds to a frame in the content that is not a key frame; the playback engine is to playback in response to a user instruction the portion of the content most recently played back by the playback engine. Based on the above distinguishing technical feature, the technical problem to be actually solved by claim 20 is the search and play of video frame.

D2 discloses a method for producing fast playback video data, and in particular makes public the following technical features (see line 5 on column 1 to line 18 on column 2 and line 10 to

line 56 on column 8 of the description and Figs.2-10 of D2); playback video data includes a normal playback mode and a fast playback (visual search) mode; normal playback coded video data and fast playback coded video data are stored in advance in the video server, and the foregoing encoded data to be used for producing the video data at a normal speed; the fast playback coded video data are encoded by using the intra-frame coding that enables random access at any desired frame (equivalent to the visual-search of the video track); the decoded video data are encoded at the second encoder by using the I-frame, the key frames such as I1' and I3' are produced by using the frames such as f5' and f9', to obtain the fast playback coded video data. It is common sense in the art that, according to actual design requirement, playing the selected content in response to a user instruction as well as the installation to realize corresponding function. Therefore, it would be obvious for those skilled in the art to reach the technical solution of claim 20 by combining D1 with D2 and on basis of common sense. Therefore, the technical solution claimed by claim 20 neither possesses prominent substantive features nor represents any notable progress, and thus lacks inventiveness as prescribed in Article 22, paragraph 3 of the Chinese Patent Law.

20. It is a customary technical means in playing back a media file to play the selected content in response to a user instruction. Therefore, given that claim 21 to which claim 22 refers lacks inventiveness, claim 22 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

21. D1 discloses the following content (see line 1 to line 5 on page 8 of the description in D1): the TP I frame packets are coupled to multiplexor which inserts the I-frame packets for TP speed into the normal play transport stream. And it is a common sense in the art to locate the play frame by using timestamps. Therefore, given that claim 21 to which claim 22 refers lacks inventiveness, the claim 22 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

22. The additional technical feature of claim 23 has been disclosed in D1 (see line 32 on page 4 to line 30 on page 6 of the description in D1): the reconstructed TP I-frame is of lower spatial resolution than the original stream. Therefore, given that claim 22 to which claim 23 refers lacks inventiveness, the claim 23 also involves no inventiveness as provided for in Article 22, paragraph 3 of the Chinese Patent Law.

For the above reasons, the independent claims and dependent claims of the present application lack inventiveness. And no other substantive content that can be granted a patent right is mentioned in the description. Therefore, even if the applicant recombines the claims or further defines the claims according to the contents mentioned in the description, the application does not have any prospects of being patented. If the applicant fails to present sufficient reasons to show that the application has novelty and an inventive step within the time limit prescribed in the office action, the application will be rejected.



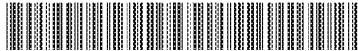
中华人民共和国国家知识产权局

100191

北京市海淀区北四环中路238号柏彦大厦1701B 北京金恩港知识产权
代理有限公司
邵毓琴

发文日:

2014年05月06日



申请号或专利号: 200880127596.4

发文序号: 2014043001038390

申请人或专利权人: 索尼克 IP 股份有限公司

发明创造名称: 用于编码供包含在媒体文件中的应用加强轨道的系统和方法

第四次审查意见通知书

1. ☒ 审查员已经收到申请人于 2014 年 4 月 16 日提交的意见陈述书, 在此基础上审查员对上述专利申请继续进行实质审查。

☐ 根据国家知识产权局专利复审委员会于_____年_____月_____日作出的复审决定, 审查员对上述专利申请继续进行实质审查。

☐ _____

2. ☐ 经审查, 申请人于_____提交的修改文件, 不符合专利法实施细则第 51 条第 3 款的规定, 不予接受。

3. 继续审查是针对下列申请文件进行的:

☐ 上述意见陈述书中所附的经修改的申请文件。

☒ 前次审查意见通知书所针对的申请文件以及上述意见陈述书中所附的经修改的申请文件替换文件。

☐ 前次审查意见通知书所针对的申请文件。

☐ 上述复审决定所确定的申请文件。

☐ _____

4. ☐ 本通知书未引用新的对比文件。

☒ 本通知书引用下列对比文件(其编号续前, 并在今后的审查过程中继续沿用):

编号	文 件 号 或 名 称	公开日期 (或授权公告的日期)
1	CN 1169229A	19971231
2	US 5887110A	19990323
3	WO 0235832A2	20020502
4	US 5717816A	19980210

5. 审查的结论性意见:

关于说明书:

☐ 申请的内容属于专利法第 5 条规定的不授予专利权的范围。

☐ 说明书不符合专利法第 26 条第 3 款的规定。

210403
2010.2

纸件申请, 回函请寄: 100088 北京市海淀区蓟门桥西土城路 6 号 国家知识产权局专利局受理处收
电子申请, 应当通过电子专利申请系统以电子文件形式提交相关文件。除另有规定外, 以纸件等其他形式提交的文件视为未提交。



中华人民共和国国家知识产权局

- ☐ 说明书的修改不符合专利法第 33 条的规定。
☐ 说明书的撰写不符合专利法实施细则第 17 条的规定。
☐ _____

关于权利要求书:

- ☐ 权利要求_____不符合专利法第 2 条第 2 款的规定。
☐ 权利要求_____不符合专利法第 9 条第 1 款的规定。
☐ 权利要求_____不具备专利法第 22 条第 2 款规定的新颖性。
☒ 权利要求 1-23 不具备专利法第 22 条第 3 款规定的创造性。
☐ 权利要求_____不具备专利法第 22 条第 4 款规定的实用性。
☐ 权利要求_____属于专利法第 25 条规定的不授予专利权的范围。
☐ 权利要求_____不符合专利法第 26 条第 4 款的规定。
☐ 权利要求_____不符合专利法第 31 条第 1 款的规定。
☐ 权利要求_____的修改不符合专利法第 33 条的规定。
☐ 权利要求_____不符合专利法实施细则第 19 条的规定。
☐ 权利要求_____不符合专利法实施细则第 20 条的规定。
☐ 权利要求_____不符合专利法实施细则第 21 条的规定。
☐ 权利要求_____不符合专利法实施细则第 22 条的规定。
☐ _____
☐ 申请不符合专利法第 26 条第 5 款或者实施细则第 26 条的规定。
☐ 申请不符合专利法第 20 条第 1 款的规定。
☐ 分案申请不符合专利法实施细则第 43 条第 1 款的规定。
上述结论性意见的具体分析见本通知书的正文部分。
6. 基于上述结论性意见, 审查员认为:
☐ 申请人应当按照通知书正文部分提出的要求, 对申请文件进行修改。
☐ 申请人应当在意见陈述书中论述其专利申请可以被授予专利权的理由, 并对通知书正文部分中指出的不符合规定之处进行修改, 否则将不能授予专利权。
☒ 专利申请中没有可以被授予专利权的实质性内容, 如果申请人没有陈述理由或者陈述理由不充分, 其申请将被驳回。
☐ _____

7. 申请人应注意下列事项:

(1) 根据专利法第 37 条的规定, 申请人应当在收到本通知书之日起的 2 个月内陈述意见, 如果申请人无正当理由逾期不答复, 其申请将被视为撤回。

(2) 申请人对其申请的修改应当符合专利法第 33 条的规定, 不得超出原说明书和权利要求书记载的范围, 同时申请人对专利申请文件进行的修改应当符合专利法实施细则第 51 条第 3 款的规定, 按照本通知书的要求进行修改。

(3) 申请人的意见陈述书和/或修改文本应当邮寄或递交国家知识产权局专利局受理处, 凡未邮寄或递交给受理处的文件不具备法律效力。

(4) 未经预约, 申请人和/或代理人不得前来国家知识产权局与审查员举行会晤。

8. 本通知书正文部分共有 4 页, 并附有下列附件:

- ☐ 引用的对比文件的复印件共_____份_____页。
☐ _____

审查员: 王博

联系电话: 62089145

审查部门: 通信发明审查部

210403
2010.2

纸件申请, 回函请寄: 100088 北京市海淀区蓟门桥西土城路 6 号 国家知识产权局专利局受理处收
电子申请, 应当通过电子专利申请系统以电子文件形式提交相关文件。除另有规定外, 以纸件等其他形式提交的文件视为未提交。



第四次审查意见通知书

申请号:2008801275964

申请人于2014年4月16日提交了意见陈述书和经过修改的申请文件,审查员在阅读了上述文件后,对本案继续进行审查,再次提出如下审查意见。

1、权利要求1要求保护一种用于播放的媒体文件进行编码的方法,对比文件1公开了一种数字图像信号的编码方法,并公开了以下技术特征(参见说明书第4页第5行-第8页第6行、第12页第8行-第15页第6行):电视传输数据流被解多路,以便获取对应于电视视频信息的PES数据流;PES数据流被解码,以便从数据包中提取视频数据流负载;针对每一个TP重放速度,从已接收的传送数据流中产生出低分辨率的帧内编码的各帧;由于将被重建的是一个编码内的I帧,所以只提取一些内部数据片,内部数据片只包括内编码的宏数据块;重建的TP I帧具有比原始的数据流要低的空间分辨率;低分辨率的TP I帧使用的是来自原始帧的宏数据块的一个子集,将其余的没有被选用的宏数据块放弃;缓存器15对于电视传输数据流进行缓存,产生用于正常放像速度处理的一个传送数据流,信号10;正常重放传送数据流10被耦合到多路复用器;TP I帧数据包5X信号121,18X信号131和35X信号141,被耦合到多路复用器,它将每一个TP速度的I帧数据包加入到正常的重放传送数据流中。权利要求1的技术方案与对比文件1的区别是:特技播放轨道进行视频轨道的图像搜索,关键帧对应于编码后的内容中不是关键帧的帧。基于上述区别技术特征,权利要求1要解决的技术问题是搜索视频帧。

对比文件2公开了一种生成快速重放视频数据的方法,并公开了以下技术特征(参见说明书第1栏第5行-第2栏第18行,第8栏第10-56行,附图2-10):重放视频数据包括正常重放模式和快速重放(可视化搜索)模式;正常重放编码视频数据和快速重放编码视频数据预先存储在视频服务器中,上述编码后的数据用于产生正常播放速率的视频数据;以帧内编码对快速重放编码视频数据进行编码,从而可以存取视频内容中期望的帧(相当于对视频轨道进行图像搜索);在第二编码器中使用I帧对已解码的视频数据进行编码,使用f5*、f9*等帧生成I1*、I13*等关键帧,得到快速重放编码视频数据。由此可知,区别特征已被对比文件2公开,并且上述特征在对比文件2中的作用与区别特征在权利要求1中的作用都是搜索视频帧。因此,在对比文件1、2的基础上得到权利要求1的技术方案对本领域技术人员是显而易见的,权利要求1不具有突出的实质性特点和显著的进步,不具备专利法第二十二条第三款规定的创造性。

2、权利要求2的附加技术特征已在对比文件1中公开(参见说明书第4页第32行-第6页第13行):重建的TP I帧具有比原始的数据流要低的空间分辨率;低分辨率的TP I帧使用的是来自原始帧的宏数据块的子集,将没有被选用的宏数据块放弃。因此,在引用的权利要求不具备创造性的情况下,权利要求2也不具备专利法第二十二条第三款规定的创造性。

3、对比文件1公开了(参见说明书第4页第32行-第6页第30行):重建的TP I帧具有比原始的数据流要低的空间分辨率。使用特定的因数,如乘以特定因子,来降低媒体内容的分辨率,以产生用于播放的视频,是本领域的公知常识。因此,在引用的权利要求不具备创造性的情况下,权利要求3也不具备专利法第二十二条第三款规定的创造性。

4、对比文件1公开了(参见说明书第12页第18-29行):对信号进行速度相关的时间亚取样;在N倍特技放像速度时,帧速率从30Hz降低到30/2NHz。使用特定的因数,如乘以特定因子,来实现对帧速率的减小,以产生播放的视频,是本领域的公知常识。因此,在引用的权利要求不具备创造性的情况下,权利要求4也不具备专利法第二十二条第三款规定的创造性。

5、权利要求5的附加技术特征已在对比文件1中公开(参见说明书第7页第25行-第8页第6行):正常重放传送数据流10被耦合到多路复用器;TP I帧数据包5X信号121、18X信号131和35X信号141,被耦合到多路复用器,它将TP速度的I帧数据包加入到正常的重放传送数据流中。因此,在引用的权利要求不具备创造性的情况下,权利要求5也不具备专利法第二十二条第三款规定的创造性。

6、将播放轨道集成到正常播放的文件块中,实现相应的播放速度,是本领域的公知常识。因此,在引用的权利要求不具备创造性的情况下,权利要求6也不具备专利法第二十二条第三款规定的创造性。

7、播放轨道只包括视频数据,以提高播放速度,是本领域的公知常识。因此,在引用的权利要求不具备创造性的情况下,权利要求7也不具备专利法第二十二条第三款规定的创造性。

210403
2010.2

纸件申请,回函请寄:100088 北京市海淀区蓟门桥西土城路6号 国家知识产权局专利局受理处收
电子申请,应当通过电子专利申请系统以电子文件形式提交相关文件。除另有规定外,以纸件等其他形式提交的文件视为未提交。



8、权利要求8的附加技术特征已在对比文件1中公开(参见说明书第7页16-24行)；在18X的速度

的情况下，权利要求8也不具备专利法第二十二条第三款规定的创造性。

9、重新对文件内容进行编码，产生相同分辨率和帧速率的压缩视频轨道，以及为减小视频文件的大小、

使播放轨道文件小于原始文件，是本领域的公知常识。因此，在引用的权利要求不具备创造性的情况下，权

利要求9也不具备专利法第二十二条第三款规定的创造性。

10、对原始文件进行压缩，是编码音视频文件时常用的技术手段。因此，在引用的权利要求不具备创造

性的情况下，权利要求10也不具备专利法第二十二条第三款规定的创造性。

11、权利要求11要求保护一种用于播放的媒体文件进行解码的方法，对比文件1公开了一种数字图像

信号的再生方法，并公开了以下技术特征(参见说明书第4页第5行-第8页第6行、第12页第8行-第15

页第6行)：电视传输数据流被解码多路，以便获取对应于电视视频信息的PES数据流；PES数据流被解码，

以便从数据包中提取视频数据流负载；针对每一个TP重放速度，从已接收的传输数据流中产生出低分辨率的

帧内编码的各帧；由于将被重建的是一个编码内的1帧，所以只提取一些内部数据片，内部数据片只包括内

编码的宏数据块；重建的TP1帧具有比原始的数据流要低的空间分辨率；低分辨率的TP1帧使用的是来自

原始帧的宏数据块的一个子集，将其余的没有被选用的宏数据块放弃；缓存器15对于电视传输数据流进行缓

存，产生用于正常播放速度处理的一个传输数据流，信号10；正常重放传输数据流10被耦合到多路复用器；

TP1帧数据包5X信号12L18X信号13L1和35X信号14L，被耦合到多路复用器，它将每一个TP速度的1帧

数据包加入到正常的重放传输数据流中。权利要求11的技术方案与对比文件1的区别是：特技播放轨道使用

经压缩的内容中的不是关键帧的帧创建关键帧，关键帧对应于压缩后的内容中的不是关键帧的帧；播放压缩

的内容，以与图像搜索速度成比例的速率、从压缩的内容最近播放的部分确定的位置对特技播放轨道的帧解

码。基于上述区别技术特征，权利要求11要解决的技术问题是播放视频帧。

对比文件2公开了一种生成快速重放视频数据的方法，并公开了以下技术特征(参见说明书第1栏第5

行-第2栏第18行，第8栏第10-56行，附图2-10)：重放视频数据包括正常重放模式和快速重放(可视化

搜索)模式；正常重放编码视频数据和快速重放编码视频数据预先存储在视频服务器中，上述编码后的数据

用于产生正常播放速率的视频数据；以帧内编码对快速重放编码视频数据进行编码，从而可以存取视频内容

中期望的帧(相当于对视频轨道进行图像搜索)；在第二编码器中使用1帧对已解码的视频数据进行编码，

使用f5、f9等帧生成I1、I13等关键帧，得到快速重放编码视频数据。对压缩后的内容进行播放，以及根

据实际需要，从指定部分开始解码和播放视频文件，是本领域的公知常识。因此，在对比文件1、2和公

知常识的基础上得到权利要求11的技术方案对本领域技术人员是显而易见的，权利要求11不具有突出的实

质性特点和显著的进步，不具备专利法第二十二条第三款规定的创造性。

12、权利要求12的附加技术特征已在对比文件3中公开(参见说明书第5页第10行-第6页第35行)；

在特技模式播放时，只有特定帧被播放给用户，特定帧典型的是1帧。因此，在引用的权利要求不具备创造

性的情况下，权利要求12也不具备专利法第二十二条第三款规定的创造性。

13、权利要求13的附加技术特征已在对比文件3中公开(参见说明书第6页13-35行)；1帧具有不

同的数据，因此在准备播放前需要不同的传送间隔；播放时读取的数据不仅有1帧还有部分相邻的P帧或B

帧。因此，在引用的权利要求不具备创造性的情况下，权利要求13也不具备专利法第二十二条第三款规定的

创造性。

14、根据实际需要，仅对关键帧解码或者忽略相关数量的delta帧和关键帧，以实现特定的播放速

度，是本领域的公知常识。因此，在引用的权利要求不具备创造性的情况下，权利要求14、15也不具备专利

法第二十二条第三款规定的创造性。

15、对比文件4公开了(参见说明书第4栏第52-65行，第5栏第35-65行)：对视频数据进行可变长

度编码以产生减小分辨率的1帧图像。视频文件具有部分减小帧速率的图像，以压缩视频文件的大小，是本

领域的公知常识。因此，在引用的权利要求不具备创造性的情况下，权利要求16也不具备专利法第二十二

条第三款规定的创造性。

16、权利要求17要求保护一种用于播放媒体文件的系统，对比文件1公开了一种数字图像信号的编码方



法，并公开了以下技术特征（参见说明书第4页第5行-第8页第6行、第12页第8行-第15页第6行）：电视传输数据流被解多路，以便获取对应于电视视频信息的 PES 数据流；PES 数据流被解码，以便从数据包中提取视频数据流负载；针对每一个 TP 重放速度，从已接收的传送数据流中产生出低分辨率的帧内编码的各帧；由于将被重建的是一个编码内的 I 帧，所以只提取一些内部数据片，内部数据片只包括内编码的宏数据块；重建的 TP I 帧具有比原始的数据流要低的空间分辨率；低分辨率的 TP I 帧使用的是来自原始帧的宏数据块的一个子集，将其余的没有被选用的宏数据块放弃；缓存器 15 对于电视传输数据流进行缓存，产生用于正常放像速度处理的一个传送数据流，信号 10；正常重放传送数据流 10 被耦合到多路复用器；TP I 帧数据包 5X 信号 121, 18X 信号 131 和 35X 信号 141，被耦合到多路复用器，它将每一个 TP 速度的 I 帧数据包加入到正常的重放传送数据流中；接收机接收对应于视频信号而调制的载波，该调制的载波可以是来自电缆的信号源；接收机解调并处理所接收的载波以便产生传送数据流。权利要求 17 的技术方案与对比文件 1 的区别是：特技播放轨道和原始媒体文件共同具有至少一个帧并且文件大小小于原始媒体文件；特技播放轨道进行视频轨道的图像搜索；关键帧对应于编码后的内容中不是关键帧的帧；客户处理器与媒体服务器网络连接，将针对原始媒体文件的请求发送给媒体服务器。基于上述区别技术特征，权利要求 17 要解决的技术问题是搜索和播放视频帧。

对比文件 2 公开了一种生成快速重放视频数据的系统，并公开了以下技术特征（参见说明书第 1 栏第 5 行-第 2 栏第 18 行，第 8 栏第 10-56 行，附图 2-10）：重放视频数据包括正常重放模式和快速重放（可视化搜索）模式；正常重放编码视频数据和快速重放编码视频数据预先存储在视频服务器中，上述编码后的数据用于产生正常播放速率的视频数据；以帧内编码对快速重放编码视频数据进行编码，从而可以存取视频内容中期望的帧（相当于对视频轨道进行图像搜索）；在第二编码器中使用 I 帧对已解码的视频数据进行编码，使用 f5'、f9'等帧生成 I1'、I13'等关键帧，得到快速重放编码视频数据。根据实际设计需要，使特技播放轨道和媒体文件具有相同帧，并且特技播放轨道文件小于原始媒体文件，以减小视频文件占用的空间，对本领域技术人员是公知的。用户端通过网络与服务器端连接，也是本领域的公知常识。因此，在对比文件 1、2 和公知常识的基础上得到权利要求 17 的技术方案对本领域技术人员是显而易见的，权利要求 17 不具有突出的实质性特点和显著的进步，不具备专利法第二十二条第三款规定的创造性。

17、用户端可以对视频文件进行解码，并按照用户指令播放相应内容，是本领域的公知常识。因此，在引用的权利要求不具备创造性的情况下，权利要求 18 也不具备专利法第二十二条第三款规定的创造性。

18、权利要求 19 的附加技术特征已在对比文件 3 中公开（参见说明书第 5 页第 10 行-第 6 页第 35 行）：在特技模式播放时，只有特定帧被播放给用户，特定帧典型的是 I 帧。因此，在引用的权利要求不具备创造性的情况下，权利要求 19 也不具备专利法第二十二条第三款规定的创造性。

19、权利要求 20 要求保护一种媒体播放器，对比文件 1 公开了一种数字图像信号的编码方法，并公开了以下技术特征（参见说明书第 4 页第 5 行-第 8 页第 6 行、第 12 页第 8 行-第 15 页第 6 行）：电视传输数据流被解多路，以便获取对应于电视视频信息的 PES 数据流；PES 数据流被解码，以便从数据包中提取视频数据流负载；针对每一个 TP 重放速度，从已接收的传送数据流中产生出低分辨率的帧内编码的各帧；由于将被重建的是一个编码内的 I 帧，所以只提取一些内部数据片，内部数据片只包括内编码的宏数据块；重建的 TP I 帧具有比原始的数据流要低的空间分辨率；低分辨率的 TP I 帧使用的是来自原始帧的宏数据块的一个子集，将其余的没有被选用的宏数据块放弃；缓存器 15 对于电视传输数据流进行缓存，产生用于正常放像速度处理的一个传送数据流，信号 10；正常重放传送数据流 10 被耦合到多路复用器；TP I 帧数据包 5X 信号 121, 18X 信号 131 和 35X 信号 141，被耦合到多路复用器，它将每一个 TP 速度的 I 帧数据包加入到正常的重放传送数据流中；用户装置可以实现数字信号的导出，控制视频点播服务；接收机接收对应于视频信号而调制的载波，该调制的载波可以是来自电缆的信号源；接收机解调并处理所接收的载波以便产生传送数据流。权利要求 20 的技术方案与对比文件 1 的区别是：特技播放轨道包括使用内容中不是关键帧的帧创建至少一个关键帧，关键帧对应于内容中不是关键帧的帧；播放搜索引擎响应于用户指令，选择播放引擎最近播放的内容的部分。基于上述区别技术特征，权利要求 20 要解决的技术问题是搜索和播放视频帧。

对比文件 2 公开了一种生成快速重放视频数据的系统，并公开了以下技术特征（参见说明书第 1 栏第 5 行-第 2 栏第 18 行，第 8 栏第 10-56 行，附图 2-10）：重放视频数据包括正常重放模式和快速重放（可视化



中华人民共和国国家知识产权局

搜索)模式:正常重放编码视频数据和快速重放编码视频数据预先存储在视频服务器中,上述编码后的数据被用于产生正常播放速率的视频数据;以帧内编码对快速重放编码视频数据进行编码,从而可以存取视频内容中期望的帧(相当于对视频轨道进行图像搜索);在第二编码器中使用I帧对已解码的视频数据进行编码,使用I5、I9等帧生成I1、I13等关键帧,得到快速重放编码视频数据。根据实际设计需要,响应用户指令播放特定部分的内容,以及实现相应功能的装置,是本领域的公知常识。因此,在对比文件1、2和公知常识的基础上得到权利要求20的技术方案对本领域技术人员是显而易见的,权利要求20不具有突出的实质性特点和显著的进步,不具备专利法第二十二条第三款规定的创造性。

20、响应用户指令播放特定部分的内容,是播放音视频文件时常用的技术手段。因此,在引用的权利要求不具备创造性的情况下,权利要求21也不具备专利法第二十二条第三款规定的创造性。

21、对比文件1公开了(参见说明书第8页第1-5行):TP I帧数据被耦合到多路复用器,它将TP速度的I帧数据加入到正常的重放传送数据流中。使用时间戳定位播放帧,是本领域的公知常识。因此,在引用的权利要求不具备创造性的情况下,权利要求22也不具备专利法第二十二条第三款规定的创造性。

22、权利要求23的附加技术特征已在对比文件1中公开(参见说明书第4页第32行-第6页第30行):重建的TP I帧具有比原始的数据流要低的空间分辨率。因此,在引用的权利要求不具备创造性的情况下,权利要求23也不具备专利法第二十二条第三款规定的创造性。

基于上述理由,本申请的独立权利要求以及从属权利要求都不具备创造性,同时说明书中也没有记载其他任何可以授予专利权的实质性内容,因而即使申请人对权利要求进行重新组合或根据说明书记载的内容作进一步的限定,本申请也不具备被授予专利权的前景。如果申请人不能在本通知书规定的答复期限内提出表明本申请具有新颖性和创造性的充分理由,本申请将被驳回。

审查员姓名:王博

审查员代码:130932

210403
2010.2

纸件申请,回函请寄:100088 北京市海淀区蓟门桥西土城路6号 国家知识产权局专利局受理处收
电子申请,应当通过电子专利申请系统以电子文件形式提交相关文件。除另有规定外,以纸件等其他形式提交的文件视为未提交。

检索报告

申请号：2008801275964		申请日：20081222		补充检索	
申请人：索尼克 IP 股份有限公司		最早的优先权日：20080102			
权利要求项数：23		说明书段数：80+20			
审查员确定的 IPC 分类号：H04N 7/12					
检索记录信息：CNABS,CNKI：视频，图像，压缩，播放，帧；VEN:video,compress+,resolution,cod+,frame					
相 关 专 利 文 献					
类型	国别以及代码[11] 给出的文献号	代码[43]或[45] 给出的日期	IPC 分类号	相关的段落 和 / 或图号	涉及的权 利要求
Y	CN1169229A	19971231	H04N5/76	说明书第 4 页第 5 行-第 8 页第 6 行， 第 12 页第 8 行-第 15 页 第 6 行	1-23
Y	US5887110A	19990323	H04N5/783	说明书第 1 栏第 5 行-第 2 栏第 18 行， 第 8 栏第 10-56 行，附 图 2-10	1-23
Y	WO0235832A2	20020502	H04N5/76	说明书第 5 页第 10 行- 第 6 页第 15-35 行	12-13,19
Y	US5717816A	19980210	H04N5/917	说明书第 4 栏第 52-65 行，第 5 栏 第 35-65 行	16



中华人民共和国国家知识产权局

相 关 非 专 利 文 献					
类型	书名(包括版本号和卷号)	出版日期	作者姓名和出版者名称	相关页数	涉及的权利要求
类型	期刊或文摘名称 (包括卷号和期号)	发行日期	作者姓名和文章标题	相关页数	涉及的权利要求

表格填写说明事项:

1. 审查员实际检索领域的 IPC 分类号应当填写到大组和 / 或小组所在的分类位置。
2. 期刊或其它定期出版物的名称可以使用符合一般公认的国际惯例的缩写名称。
3. 相关文件的类型说明:
 - X: 一篇文件影响新颖性或创造性;
 - Y: 与本报告中的另外的 Y 类文件组合而影响创造性;
 - A: 背景技术文件;
 - R: 任何单位或个人在申请日向专利局提交的、属于同样的发明创造的专利或专利申请文件。
 - P: 中间文件, 其公开日在申请的申请日与所要求的优先权日之间的文件;
 - E: 抵触申请。

审 查 员: 王博
2014 年 04 月 23 日

审查部门: 通信发明审查部



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
13/223,210	08/31/2011	Jason Branes	DI:02357B	6884

71897	7590	04/03/2015
KPPB LLP		
2400 EAST KATELLA AVENUE		
SUITE 1050		
ANAHEIM, CA 92806		

EXAMINER
RAHMAN, MOHAMMAD J

ART UNIT	PAPER NUMBER
2487	

NOTIFICATION DATE	DELIVERY MODE
04/03/2015	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

docketing@cpaglobal.com
PAIR@KPPB.COM

Office Action Summary	Application No. 13/223,210	Applicant(s) BRANESS ET AL.	
	Examiner MOHAMMAD J. RAHMAN	Art Unit 2487	AIA (First Inventor to File) Status No

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTHS FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) ☒ Responsive to communication(s) filed on 02/13/2015.
☐ A declaration(s)/affidavit(s) under **37 CFR 1.130(b)** was/were filed on _____.

2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.

3) ☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims*

5) ☒ Claim(s) 21-40 is/are pending in the application.
 5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6) ☐ Claim(s) _____ is/are allowed.

7) ☒ Claim(s) 21-40 is/are rejected.

8) ☐ Claim(s) _____ is/are objected to.

9) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

* If any claims have been determined allowable, you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information, please see http://www.uspto.gov/patents/init_events/pph/index.jsp or send an inquiry to PPHfeedback@uspto.gov.

Application Papers

10) ☐ The specification is objected to by the Examiner.

11) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
 Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
 Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

Priority under 35 U.S.C. § 119

12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

Certified copies:

a) ☐ All b) ☐ Some** c) ☐ None of the:

1) ☐ Certified copies of the priority documents have been received.

2) ☐ Certified copies of the priority documents have been received in Application No. _____.

3) ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

** See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☒ Information Disclosure Statement(s) (PTO/SB/08a and/or PTO/SB/08b)
 Paper No(s)/Mail Date 02/13/2015 and 02/26/2015

3) ☐ Interview Summary (PTO-413)
 Paper No(s)/Mail Date. _____

4) ☐ Other: _____

Art Unit: 2487

1. The present application is being examined under the pre-AIA first to invent provisions.

DETAILED ACTION

Response to Amendment

2. Applicant's argument, filed on 02/13/2015 has been entered and carefully considered. Claims 1-20 are canceled and claims 21-40 are added. Claims 21-40 are pending.

Response to Arguments

3. Applicant's arguments with respect to claims 21 and 34 have been considered but are moot in view of the new ground(s) of rejection.

Information Disclosure Statement

4. This information disclosure statement (IDS) submitted on 02/13/2015 and 02/26/2015. The submission is in compliance with the provisions of 37 CFR 1.97 and 37 CFR 1.98. Accordingly, the information disclosure statement is being considered by the examiner.

Claim Rejections - 35 USC § 103

5. The following is a quotation of pre-AIA 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

6. Claims 21-27 and 34-38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Eerenberg et al. (US 20030231863 A1), hereinafter Eerenberg, in view of Soroushian et al. (US 20090132599 A1), hereinafter Soroushian, further in view of De Lange et al. (US20080008455 A1), hereinafter De Lange. Soroushian is cited by applicant in IDS filed on September 12, 2013.

Regarding claim 21, Eerenberg discloses a method of performing visual search of encoded media using a trick play stream using a playback device, where the media is encoded as a plurality of alternative streams used during normal playback and a trick play stream used during visual search **(Abstract, [0004])**, the method comprising: receiving at least one user instruction directing the playback device to perform a visual search of the media **([0004])**, wherein the received at least one user instruction indicates a desired rate for the visual search **([0008], enable a trick play reproduction)**.

Eerenberg discloses all the elements of claim 1 but Eerenberg does not appear to explicitly disclose wherein: the alternative streams and the trick play stream are stored in separate Extensible Binary Markup Language (EBML) container files, each of the EBML container files comprises a plurality of Cluster elements, where each Cluster element contains a portion of encoded video, the portion of encoded video in each of the Cluster elements for the alternative streams and the trick play stream commences with an intra-frame, and the intra frame in each of the Cluster elements for the trick play stream has a time code corresponding to at least one Cluster element of the alternative streams, wherein the time codes indicate start times of the encoded video within the Cluster elements; wherein the timecodes are manipulated in order to achieve a rate of playback based on the desired rate for the visual search.

However, Soroushian from the same or similar endeavor teaches wherein: the alternative streams and the trick play stream are stored in separate Extensible Binary Markup Language (EBML) container files **([0029], line 1-5, distributed in clusters in the file; [0042], line 12)**, each of the EBML container files comprises a plurality of Cluster elements, where each Cluster element contains a portion of

Art Unit: 2487

encoded video, the portion of encoded video in each of the Cluster elements for the alternative streams and the trick play stream commences with an intra-frame, and the intra frame in each of the Cluster elements for the trick play stream has a time code corresponding to at least one Cluster element of the alternative streams, wherein the time codes indicate start times of the encoded video within the Cluster elements **([0041], line 8-13, index used to locate various portions of the media and also used for trick play functions; Fig 3, element 31e, flags can have Timecode, [0032], video frames can be independently decoded using reduced index, reduced index has location of access and time stamp and it can be used to point to any other type of key-frame for other streams stored in the multimedia file)**; wherein the timecodes are manipulated in order to achieve a rate of playback based on the desired rate for the visual search **([0033], timestamps for desired speed)**.

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Eerenberg to incorporate the teachings of Soroushian to decode enhanced content without significantly increasing the file size of the multimedia file **(Soroushian, Abstract, line 4-6)**.

Eerenberg in view of Soroushian discloses all the elements of claim 1 but they do not appear to explicitly disclose requesting and buffering portions of video from at least one of a plurality of alternative streams for normal playback using a playback device ; decoding the buffered portions of video using a decoder on the playback device; manipulating, using the playback device, the timecodes assigned to the buffered frames of the trick play stream prior to providing the buffered frames of the trick play stream to a decoder on the playback device, wherein the timecodes are manipulated in order to achieve a rate of playback based on the desired rate for the visual search; providing, using the playback device, the buffered frames of the trick play stream to the decoder on the playback device; and decoding the buffered frames of the trick play stream using the decoder on the playback device; requesting and buffering portions of video from the trick play stream using the playback device.

However, De Lange from the same or similar endeavor teaches requesting and buffering portions of video from at least one of a plurality of alternative streams for normal playback using a playback device **(Fig, 2, element 201 and 205)**; decoding the buffered portions of video using a decoder on the playback device **([0011], decoder 112 and buffer 113)**; manipulating, using the playback device, the timecodes

Art Unit: 2487

assigned to the buffered frames of the trick play stream prior to providing the buffered frames of the trick play stream to a decoder on the playback device (**[[0003], timestamp associated to facilitate jumping]**), providing, using the playback device, the buffered frames of the trick play stream to the decoder on the playback device; and decoding the buffered frames of the trick play stream using the decoder on the playback device (**Fig. 2, element 205**); requesting and buffering portions of video from the trick play stream using the playback device (**[[0021], buffer is retrieved using metadata]**).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Eerenberg in view of Soroushian to incorporate the teachings of De Lange to improve trickplay operations during playback of the multimedia content (**De Lange, [0001]**).

Regarding claim 22, Eerenberg in view of Soroushian further in view of De Lange discloses comprising measuring the current streaming conditions by measuring the time taken to receive requested portions of a stream from the time at which the portions were requested using the playback device (**Eerenberg, [0199], synchronize time base**).

Regarding claim 23, Eerenberg in view of Soroushian further in view of De Lange wherein requesting and buffering portions of video from at least one of the alternative streams using a playback device further comprises requesting and buffering portions of video from at least one of the alternative streams based upon the bitrates of the alternative streams and the measured streaming conditions using the playback device (**Soroushian, [0030], line 4-11, cluster with timestamp**).

Regarding claim 24, Eerenberg in view of Soroushian further in view of De Lange wherein: the at least one user instruction additionally specifies a direction of the visual search; and requesting and buffering portions of video from the trick play stream using the playback device further comprises requesting frames from the trick play stream based upon the desired rate and direction of the visual search specified by the at least one user instruction (**Eerenberg, Fig. 13; Soroushian, [0044], direction**).

Regarding claim 25, Eerenberg in view of Soroushian further in view of De Lange wherein requesting portions of video from one of a plurality of alternative streams for normal playback further comprises requesting portions of files from remote servers via Hypertext Transfer Protocol (HTTP) byte range requests using the playback device (**Soroushian, [0041]**).

Regarding claim 26, Eerenberg in view of Soroushian further in view of De Lange wherein requesting and buffering portions of video from the trick play stream using the playback device further comprises requesting and buffering only individual frames from the trick play stream based upon the desired rate for the visual search (**Soroushian, [0044], speed and direction**).

Regarding claim 27, Eerenberg in view of Soroushian further in view of De Lange wherein requesting and buffering only individual frames from the trick play stream based upon the desired rate for the visual search further comprises omitting downloading of certain frames from the trick play stream according to a multiple of the desired rate for the visual search (**Eerenberg, [0008], higher speed**).

Regarding claim 34, Eerenberg discloses a playback device configured to perform visual search of encoded media using a trick play stream, where the media is encoded as a plurality of alternative streams used during normal playback and a trick play stream used during visual search (**Abstract, [0004]**), the playback device comprising: a processor configured to communicate with memory (**[0221]**), the processor configured by a playback client application stored in memory (**[0178]**), wherein the playback client application directs the processor to: receive at least one user instruction directing the playback device to perform a visual search of the media, wherein the received at least one user instruction indicates a desired rate for the visual search (**[0008], enable a trick play reproduction**).

Eerenberg discloses all the elements of claim 34 but Eerenberg does not appear to explicitly disclose wherein: the alternative streams and the trick play stream are stored in separate Extensible Binary Markup Language (EBML) container files, each of the EBML container files comprises a plurality of

Art Unit: 2487

Cluster elements, where each Cluster element contains a portion of encoded video, the portion of encoded video in each of the Cluster elements for the alternative streams and the trick play stream commences with an intra-frame, and the intra frame in each of the Cluster elements for the trick play stream has a time code corresponding to at least one Cluster element of the alternative streams, wherein the time codes indicate start times of the encoded video within the Cluster elements; manipulate the timecodes assigned to the buffered frames of the trick play stream prior to providing the buffered frames of the trick play stream to a decoder on the playback device, wherein the timecodes are manipulated in order to achieve a rate of playback based on the desired rate for the visual search; provide the buffered frames of the trick play stream to the decoder on the playback device; and decode the buffered frames of the trick play stream using the decoder on the playback device.

However, Soroushian from the same or similar endeavor teaches wherein: the alternative streams and the trick play stream are stored in separate Extensible Binary Markup Language (EBML) container files ([0029], line 1-5, **distributed in clusters in the file; [0042], line 12**), each of the EBML container files comprises a plurality of Cluster elements, where each Cluster element contains a portion of encoded video, the portion of encoded video in each of the Cluster elements for the alternative streams and the trick play stream commences with an intra-frame, and the intra frame in each of the Cluster elements for the trick play stream has a time code corresponding to at least one Cluster element of the alternative streams, wherein the time codes indicate start times of the encoded video within the Cluster elements ([0041], line 8-13, **index used to locate various portions of the media and also used for trick play functions; Fig 3, element 31e, flags can have Timecode, [0032], video frames can be independently decoded using reduced index, reduced index has location of access and time stamp and it can be used to point to any other type of key-frame for other streams stored in the multimedia file**); wherein the timecodes are manipulated in order to achieve a rate of playback based on the desired rate for the visual search ([0033], **timestamps for desired speed**).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Eerenberg to incorporate the teachings of Soroushian to decode enhanced content without significantly increasing the file size of the multimedia file (**Soroushian, Abstract, line 4-6**).

Eerenberg in view of Soroushian discloses all the elements of claim 34 but they do not appear to explicitly disclose request and buffer portions of video from at least one of a plurality of alternative streams for normal playback; decode the buffered portions of video using a decoder on the playback device; request and buffer portions of video from the trick play stream; decoding the buffered portions of video using a decoder on the playback device; manipulating, using the playback device, the timecodes assigned to the buffered frames of the trick play stream prior to providing the buffered frames of the trick play stream to a decoder on the playback device; providing, using the playback device, the buffered frames of the trick play stream to the decoder on the playback device; and decoding the buffered frames of the trick play stream using the decoder on the playback device; requesting and buffering portions of video from the trick play stream using the playback device.

However, De Lange from the same or similar endeavor teaches requesting and buffering portions of video from at least one of a plurality of alternative streams for normal playback using a playback device (**Fig. 2, element 201 and 205**); decoding the buffered portions of video using a decoder on the playback device (**[0011], decoder 112 and buffer 113**); manipulating, using the playback device, the timecodes assigned to the buffered frames of the trick play stream prior to providing the buffered frames of the trick play stream to a decoder on the playback device (**[0003], timestamp associated to facilitate jumping**), providing, using the playback device, the buffered frames of the trick play stream to the decoder on the playback device; and decoding the buffered frames of the trick play stream using the decoder on the playback device (**Fig. 2, element 205**); requesting and buffering portions of video from the trick play stream using the playback device (**[0021], buffer is retrieved using metadata**).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Eerenberg in view of Soroushian to incorporate the teachings of De Lange to improve trickplay operations during playback of the multimedia content (**De Lange, [0001]**).

Regarding claim 35, Eerenberg in view of Soroushian further in view of De Lange discloses wherein the playback client application further directs the processor to measure the current streaming

Art Unit: 2487

conditions by measuring the time taken to receive requested portions of a stream from the time at which the portions were requested (**Eerenberg, [0199], synchronize time base**).

Regarding claim 36, Eerenberg in view of Soroushian further in view of De Lange discloses wherein the playback client application further directs the processor to request and buffer portions of video from at least one of the alternative streams based upon the bitrates of the alternative streams and the measured streaming conditions (**Soroushian, [0030], line 4-11, cluster with timestamp**).

Regarding claim 37, Eerenberg in view of Soroushian further in view of De Lange discloses wherein: the at least one user instruction additionally specifies a direction of the visual search; and wherein the playback client application further directs the processor to request frames from the trick play stream based upon the desired rate and direction of the visual search specified by the at least one user instruction (**Eerenberg, Fig. 13; Soroushian, [0044], direction**).

Regarding claim 38, Eerenberg in view of Soroushian further in view of De Lange discloses wherein the playback client application further directs the processor to request portions of files from remote servers via Hypertext Transfer Protocol (HTTP) byte range requests (**Soroushian, [0041]**).

7. Claims 28-33 and 39-40 are rejected under 35 U.S.C. 103(a) as being unpatentable over Eerenberg in view of Soroushian further in view of De Lange further in in view of Priyadarshi et al. (US 20090169181 A1), hereinafter Priyadarshi.

Regarding claim 28, Eerenberg in view of Soroushian further in view of De Lange discloses all the elements of claim 28 but they not appear to explicitly disclose wherein the portions of encoded video in each of the Cluster elements of the alternative streams have the same duration.

However, Priyadarshi from the same or similar endeavor teaches wherein the portions of encoded video in each of the Cluster elements of the alternative streams have the same duration **([0061], same properties, duration; Fig. 10)**.

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify Eerenberg in view of Soroushian further in view of De Lange to incorporate the teachings of Priyadarshi to provide enhanced digital media playback **(Priyadarshi, Abstract, line 1-3)**.

Regarding claim 29, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses wherein each encoded frame of the portion of encoded video contained within the Cluster element is contained within a separate element **(Soroushian, [0034])**.

Regarding claim 30, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses wherein the first element containing a frame in the Cluster element contains the intra frame **(Priyadarshi, [0059])**.

Regarding claim 31, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses wherein the EBML container file containing the trick play stream includes at least one modified Cues element containing an index that references Clusters elements containing portions of encoded video within the EBML container file and the elements containing frames within the referenced Clusters elements **(Soroushian, [0042])**.

Regarding claim 32, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses further comprising: retrieving the modified Cues element using the playback device; and identifying frames to retrieve from the EBML container file based upon a visual search rate using the Cues element, wherein requesting and buffering portions of video from the trick play stream using the playback device further comprises requesting and buffering elements that contain frames from

Art Unit: 2487

the EBML container file containing the trick play stream using the playback device (**Soroushian, [0040], [0042]**).

Regarding claim 33, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses further comprising retrieving a top level index file using the playback device that identifies the alternative streams used during normal playback of the encoded video and identifies the trick play stream used during visual search of the encoded video (**Soroushian, [0040]-[0042]**).

Regarding claim 39, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses wherein the EBML container file containing the trick play stream includes at least one modified Cues element containing an index that references Clusters elements containing portions of encoded video within the EBML container file and the elements containing frames within the referenced Clusters elements (**Soroushian, [0042]**).

Regarding claim 40, Eerenberg in view of Soroushian further in view of De Lange further in view of Priyadarshi discloses wherein the playback client application further directs the processor to: retrieve the modified Cues element; identify frames to retrieve from the EBML container file based upon a visual search rate using the Cues element; and request and buffer elements that contain frames from the EBML container file containing the trick play stream (**Soroushian, [0040], [0042]**).

Conclusion

8. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date

Art Unit: 2487

of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to MOHAMMAD J. RAHMAN whose telephone number is (571)270-7190. The examiner can normally be reached on Monday-Thursday 8:30am-7:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Dave Czekaj can be reached on 571 272-7327. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/MOHAMMAD J RAHMAN/
Examiner, Art Unit 2487

/Dave Czekaj/
Supervisory Patent Examiner, Art Unit 2487

Notice of References Cited	Application/Control No. 13/223,210	Applicant(s)/Patent Under Reexamination BRANESS ET AL.	
	Examiner MOHAMMAD J. RAHMAN	Art Unit 2487	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-2005/0183120 A1	08-2005	Jain et al.	725/046
*	B	US-2008/0008455 A1	01-2008	De Lange et al.	386/125
*	C	US-2010/0319017 A1	12-2010	Cook, Michael	725/31
*	D	US-8,401,900 B2	03-2013	Cansler et al.	705/14.49
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1400
Alexandria, Virginia 22304-1400
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
14/564,003	12/08/2014	Michael George Kuefer	D1-02359A.CON	5114
71897 KPPB LLP 2400 EAST KATELLA AVENUE SUITE 1050 ANAHEIM, CA 92806			EXAMINER GREYCHOW, ABY	
			ART UNIT 2434	PAPER NUMBER
			NOTIFICATION DATE 04/17/2015	DELIVERY MODE ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

docketing@cpaglobal.com
PAIR@KPPB.COM

Office Action Summary	Application No. 14/564,003	Applicant(s) KIEFER ET AL.	
	Examiner ABIY GETACHEW	Art Unit 2434	AIA (First Inventor to File) Status No

– The MAILING DATE of this communication appears on the cover sheet with the correspondence address –

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTHS FROM THE MAILING DATE OF THIS COMMUNICATION.

Extensions of time may be available under the provisions of 37 CFR 1.136(s). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
 If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
 Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
 Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any awarded patent term adjustment. See 37 CFR 1.704(b).

Status

1) ☒ Responsive to communication(s) filed on 12/08/2014.
☐ A declaration(s)/affidavit(s) under 37 CFR 1.130(b) was/were filed on _____.

2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.

3) ☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims*

5) ☒ Claim(s) 1-20 is/are pending in the application.
 5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6) ☐ Claim(s) _____ is/are allowed.

7) ☒ Claim(s) 1-20 is/are rejected.

8) ☐ Claim(s) _____ is/are objected to.

9) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

* If any claims have been determined allowable, you may be eligible to benefit from the **Patent Prosecution Highway** program at a participating intellectual property office for the corresponding application. For more information, please see http://www.uspto.gov/patents/init_events/pph/index.jsp or send an inquiry to PPHfeedback@uspto.gov.

Application Papers

10) ☐ The specification is objected to by the Examiner.

11) ☒ The drawing(s) filed on 12/08/2014 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
 Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
 Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

Priority under 35 U.S.C. § 119

12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

Certified copies:

a) ☐ All b) ☐ Some** c) ☐ None of the:

1) ☐ Certified copies of the priority documents have been received.

2) ☐ Certified copies of the priority documents have been received in Application No. _____.

3) ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

** See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Information Disclosure Statement(s) (PTO/SB/08a and/or PTO/SB/08b).
 Paper No(s)/Mail Date _____.

3) ☐ Interview Summary (PTO-413)
 Paper No(s)/Mail Date _____.

4) ☐ Other: _____.

DETAILED ACTION

1. The present application is being examined under the pre-AIA first to invent provisions.
2. This is a Non-Final Office Action in response to the communication filed on December 08, 2014.
 - Claims 1-20 have been examined.

Drawings

3. The drawings filed on September 08, 2014 are acceptable for examination proceedings.

Specification

4. The specification filed on September 08, 2014 is also accepted.

Double Patenting

5. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and, *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).
A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).
Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).
6. Claims 1-20 is rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 1 of U.S. Patent No. 8,909,922

(hereinafter referred to as '922 Patent) Although the conflicting claims are not identical, they are not patentably distinct from each other.

The following is referring to the independent claim

• As per independent claim 1.

Independent claim 1 of the instant application and claim 1 of the '922 patent recite similar limitation. The above independent claims, namely claim 1 of the instant/present application would have been obvious over independent claim 1 of the '922 patent because each and every element of the above independent claim 1 of the present application is anticipated by the corresponding independent claim 1 of the '922 patent.

As per independent claim 1.

Independent claim 18 of the instant application and claim 18 of the '922 patent recite similar limitation. The above independent claims, namely claim 18 of the instant/present application would have been obvious over independent claim 15 of the '922 patent because each and every element of the above independent claim 1 of the present application is anticipated by the corresponding independent claim 15 of the '922 patent.

The following is referring to the dependent claims

• Referring to dependent claims 2-17 and 19-20.

Claims 2-17 and 19-20 of the instant application is also anticipated by claims 2,3,4,5,6,7,8,9,10,11,12,13,14,15 and 17 of the '922 US patent since the corresponding claims further recite similar/same limitation of the same subject matter.

Instant Application	Copending Patent No. 8,909,922
1. A playback device, comprising: a processor; memory containing a client application; wherein the client application configures the processor to: request a top level index file identifying a plurality of alternative streams of protected content, where each of the alternative streams of protected content are encrypted using common cryptographic information; obtain the common cryptographic information; request portions of content from at least the plurality of alternative streams of protected content; access the protected content using the common cryptographic information; and playback the content.	1. A playback device, comprising: a processor; memory containing a client application; wherein the client application configures the processor to: obtain a top level index file identifying a plurality of separate container files each containing one of a plurality of alternative streams of protected video, wherein each of the alternative streams of protected video includes video frames that are encrypted using a common set of frame keys comprising a plurality of different frame keys, wherein each of the container files containing the alternative streams of protected video comprises (i) an encrypted copy of the common set of frame keys and (ii) a set of DRM information that identifies encrypted portions of the video frames and frame keys to utilize in accessing the encrypted portions of the video frames, and wherein the set of DRM information in each of the plurality of alternative streams of protected video references the same common set of frame keys; obtain an encrypted copy of the common set of frame keys from a given container file and decrypt the encrypted copy of the common set of frame keys using a content key; obtain a set of DRM information from the given container file; obtain portions of protected video from the given container file containing one of the plurality of alternative streams of protected video; decrypt encrypted video frames from the obtained portions of video using the decrypted copy of the common set of frame keys according to the obtained set of DRM information; and playback the obtained portions of video.
2. The playback device of claim 1, wherein: the alternative streams of	2. The playback device of claim 1, wherein: the alternative streams of

protected content are alternative streams of protected video; each of the alternative streams of protected video includes DRM information indicating portions of frames of video that are encrypted and identifying at least a portion of the common cryptographic information used to access the encrypted portions of the frames of video; and the client application configures the processor to: obtain a protected frame of video from one of the alternative streams of protected video; obtain DRM information indicating a portion of the protected frame of video that is encrypted; identify at least a portion of the common cryptographic information that is used to access the encrypted portion of the protected frame using the DRM information; decrypt the encrypted portion of the protected frame of video using the identified portion of the common cryptographic information; and decode the frame of video.	protected video are encoded at different maximum bitrates using different sets of according parameters and the client application <u>further configures the processor to decode the encoded frames of video prior to playback of the obtained portions of video.</u>
3. The playback device of claim 1, wherein: each of the alternative streams of protected content is contained within a separate container file; and the top level index file identifies a container file containing each of the alternative streams.	3. The playback device of claim 1, wherein: the container files containing each of the alternative streams of <u>protected video comprise at least one header that includes information that describes the encoding of the stream of protected video; and the top level index file identifies the location of the at least one header within each container file.</u>
4. The playback device of claim 3, wherein: the container files containing each of the alternative streams of protected content comprise at least one header including the common cryptographic information and information that describes the encoding of the stream of protected content; and the top level index file identifies the location of the at least one header within each container file.	3. The playback device of claim 1, wherein: the container files containing each of the alternative streams of protected video comprise at least one header that includes information that <u>describes the encoding of the stream of protected video; and the top level index file identifies the location of the at least one header within each container file.</u>
5. The playback device of claim 4, wherein	4. The playback device of claim 1, wherein

the common cryptographic information is a set of encryption keys.	the common set of frame keys <u>is part of a frame key table that is common to each of the plurality of alternative streams.</u>
6. The playback device of claim 5, wherein: the common set of encryption keys is encrypted; and the client application configures the processor to securely receive cryptographic information used to unencrypt the common set of encryption keys.	5. The playback device of claim 1, wherein: the client application further configures the processor to securely receive <u>the content key.</u>
7. The playback device of claim 4, wherein the client application configures the processor to obtain the common cryptographic information by requesting at least one header of a container file identified in the top level index file.	18. The method of claim 17, further comprising obtaining <u>an encrypted copy of the common set of frame keys</u> by requesting at least one header of a container file identified in the top level index file <u>using the playback device.</u>
8. The playback device of claim 7, wherein: each container file containing one of the alternative streams of protected content further comprises an index; and the client application configures the processor to request a portion of content from one of the container files by configuring the processor to: request at least a portion of an index from the container file; identify the portion of the container file containing a specific portion of content using the index; and request the identified portion of the container file.	6. The playback device of claim 1, wherein: each container file containing one of the alternative streams of protected video further comprises an index; and the client application further configures the processor to request at least a portion of an index from the given container file; identify the portion of the given container file containing a specific portion of <u>video using the index;</u> and request the identified portion of the given container file.
9. The playback device of claim 4, wherein the container files are Matroska container files.	7. The playback device of claim 1, wherein the container files are Matroska container files.
10. The playback device of claim 9, wherein the common cryptographic information is stored in a common DRM Header element contained within a Tracks element within each of the Matroska container files containing the alternative streams of protected content.	8. The playback device of claim 9, wherein the <u>encrypted copy of the common set of frame keys</u> is stored in a common DRM Header element contained within a Tracks element within each of the Matroska container files containing the alternative <u>streams of protected video.</u>

11. The playback device of claim 10, wherein: the common cryptographic information is a set of encryption keys; and the set of encryption keys is encrypted and stored within the DRM Header element.	8. The playback device of claim 7, wherein the encrypted copy of the common set of frame keys is stored in a common DRM Header element <u>contained within a Tracks element within each of the Matroska container files containing the alternative streams of protected video.</u>
12. The playback device of claim 11, wherein the client application configures the processor to: obtain cryptographic information that can be used to unencrypt the set of encryption keys stored within the DRM Header element; and unencrypt the set of encryption keys stored within the DRM Header element using the obtained cryptographic information.	9. The playback device of claim 10, wherein the client application <u>further</u> configures the processor to: obtain <u>the content key</u> that can be used to <u>decrypt the encrypted copy of the set of frame keys</u> stored within the <u>common</u> DRM Header element; and <u>decrypt the encrypted copy of the set of frame keys</u> stored within the <u>common</u> DRM Header element using the obtained content key.
13. The playback device of claim 10, wherein the client application configures the processor to obtain the DRM Header element by requesting a Tracks element from one of the Matroska container files identified in the top level index file.	10. The playback device of claim 10, wherein the client application configures the processor to obtain the <u>common DRM</u> Header element by requesting a Tracks element from one of the Matroska container files identified in the top level index file.
14. The playback device of claim 13, wherein: each Matroska container file containing an alternative stream of protected content further comprises an index contained within a Cues element; and the client application configures the processor to request a portion of content from one of the Matroska container files by configuring the processor to: request at least a portion of a Cues element from a Matroska container file containing one of the alternative streams of protected content; identify a portion of the Matroska container file containing a specific portion of content using at least one CueTrackPosition element within the Cues element; and request the portion of the	11. The playback device of claim 13, wherein: each Matroska container file containing an alternative stream of protected video further comprises an index contained within a Cues element; and the client application <u>further configures the processor to request a portion of video from one of the Matroska container files by configuring the processor to: request at least a portion of a Cues element from a Matroska container file containing one of the alternative streams of protected video; identify a portion of the Matroska container file containing a specific portion of video using at least one CueTrackPosition element within the Cues element; and request the portion of the</u>

Matroska container file identified by the at least one CueTrackPosition element.	Matroska container file identified by the at least one CueTrackPosition element.
15. The playback device of claim 14, wherein the index comprises a plurality of CueTrackPosition elements that identify the location of a plurality of Cluster elements within the Matroska container file.	12. The playback device of claim 14, wherein the index comprises a plurality of CueTrackPosition elements that identify the location of a plurality of Cluster elements within the Matroska container file.
16. The playback device of claim 15, wherein: each Cluster element contains a plurality of BlockGroup elements including encoded portions of content; a CueTrack Position element indexes each Cluster element; and each CueTrackPosition element includes the size of the Cluster element that it indexes.	14. The playback device of claim 12, wherein: each Cluster element contains a plurality of BlockGroup elements including encoded portions of video; a CueTrack Position element indexes each Cluster element; and each CueTrackPosition element includes the size of the Cluster element that it indexes.
17. The playback device of claim 14, wherein the alternative streams of protected content are alternative streams of protected video; each of the Matroska container files containing an alternative stream of protected video includes a DRMinfo element in a BlockGroup element containing a protected frame of video, where the DRMinfo element indicates at least a portion of the protected frames of video that is encrypted and identifies at least a portion of the common cryptographic information used to access the at least one encrypted portion of the protected frame of video; and the client application configures the processor to access video frames in a stream of protected video using the common cryptographic information by configuring the processor to: obtain a BlockGroup containing a protected frame of video; parse the BlockGroup element to obtain a DRMinfo element indicating at least a	13. The playback device of claim 14, wherein each of the Matroska container files containing an alternative stream of protected video includes <u>a set of BlockGroup elements that each contain a DRMinfo element and an encrypted frame of video, where the DRMinfo element is a part of the set of DRM information for each of the Matroska container files</u> and the client application <u>further configures the processor to decrypt encrypted video frames of the obtained portions of video using the decrypted copy of the common set of frame keys</u> by configuring the processor to: obtain a BlockGroup containing <u>an encrypted frame of video</u> ; parse the BlockGroup element to obtain a DRMinfo element indicating at least a portion of the <u>encrypted frame of video</u> that is encrypted using a frame key from the <u>common set of frame keys</u> ; decrypt the at least one encrypted portion of the <u>encrypted frame of video</u> using the

portion of the protected frame of video that is encrypted using an encryption key from the set of encryption keys; decrypt the at least one encrypted portion of the protected frame of video using the identified encryption key; and decode the frame of video.	identified frame key; and decode the frame of video.
18. A method of performing adaptive bitrate streaming using a plurality of alternative streams of protected content, where each of the streams of protected content is protected using common cryptographic information, the method comprising: requesting a top level index file identifying a plurality of alternative streams of protected content using a playback device, where each of the alternative streams of protected content are encrypted using common cryptographic information; obtaining the common cryptographic information using the playback device; requesting portions of content from at least the plurality of alternative streams of protected content using the playback device; accessing the protected content using the playback device and the common cryptographic information; and playing back the content using the playback device.	15. A method of performing adaptive bitrate streaming using a plurality of alternative streams of protected video the method comprising: <u>obtaining a top level index file identifying a plurality of separate container files each containing one of a plurality of alternative streams of protected using a playback device, where wherein each of the alternative streams of protected include video frames that are encrypted using a common set of frame keys comprising a plurality of different frame keys and wherein each of the container files containing the alternative streams of protected video comprises (i) an encrypted copy of the common set of frame keys and (ii) a set of DRM information that identifies encrypted portions of the video frames and frame keys to utilize in accessing the encrypted portions of the video frames, and wherein the set of DRM information in each of the plurality of alternative streams of protected video references the same common set of frame keys; obtaining an encrypted copy of the common set of frame keys from a given container file and decrypting the encrypted copy of the common set of frame keys using a content key using the playback device; obtaining a set of DRM information from the given container file using the playback device; obtaining portions of video from the given container file containing one of the plurality of alternative streams of protected using the playback device; decrypting encrypted video frames from the obtained portions of</u>

	video using the playback device and the <u>decrypted copy of the common set of frame keys according to the obtained set of DRM information and playing back the obtained portions of using the playback device.</u>
19. The method of claim 18, wherein: the alternative streams of protected content are alternative streams of protected video; each of the alternative streams of protected video includes DRM information indicating portions of frames of video that are encrypted and identifying at least a portion of the common cryptographic information used to access the encrypted portions of the frames of video; and the method further comprises: obtaining a protected frame of video from one of the alternative streams of protected video using the playback device; obtaining DRM information indicating a portion of the protected frame of video that is encrypted using the playback device; identify at least a portion of the common cryptographic information that is used to access the encrypted portion of the protected frame using the playback device and the DRM information; decrypting the encrypted portion of the protected frame of video using the playback device and the identified portion of the common cryptographic information; and decoding the frame of video using the playback device.	15. A method of performing adaptive bitrate streaming using a plurality of alternative streams of protected video, the method comprising: obtaining a top level index file identifying a plurality of separate container files each containing one of a plurality of alternative streams of protected video using a playback device, wherein each of the alternative streams of protected video include video frames that are encrypted using a common set of frame keys comprising a plurality of different frame keys, and wherein each of the container files containing the alternative streams of protected video comprises (i) an encrypted copy of the common set of frame keys and (ii) a set of DRM information that identifies encrypted portions of the video frames and frame keys to utilize in accessing the encrypted portions of the video frames, and wherein the set of DRM information in each of the plurality of alternative streams of protected video references the same common set of frame keys; obtaining an encrypted copy of the common set of frame keys from a given container file and decrypting the encrypted copy of the common set of frame keys using a content key using the playback device; obtaining a set of DRM information from the given container file using the playback device; obtaining portions of video from the given container file containing one of the plurality of alternative streams of protected video using the playback device; <u>decrypting encrypted video frames from the obtained portions of video using the playback</u>

	<u>device and the decrypted copy of the common set of frame keys according to the obtained set of DRM information; and playing back the obtained portions of video using the playback device.</u>
20. The method of claim 18, wherein: each of the alternative streams of protected content is contained within a separate container file; and the top level index file identifies a container file containing each of the alternative streams.	17. The method of claim 15, wherein: the container files containing each of the alternative streams of protected video comprise at least one header that includes information and information that describes the encoding of the stream of protected video; and the top level index file <u>identifies the location of the at least one header within each container file.</u>

Claim Rejections - 35 USC § 103

7. The following is a quotation of pre-AIA 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

8. The text of those sections of Title 35, U.S. Code not included in this action can be found in a prior Office action.

9. Claims 1,3 and 18 are rejected under pre-AIA 35 U.S.C. 103(a) as being unpatentable over Amini et al. (US 2004/0202320 A1) hereafter refers as to Amini in view of Thorwirth (WO 2011/103364 A1).

Regarding claim 1, Amini discloses a device (***fig. 2 a multi-encoder platform 220 i.e. "a device"***), comprising: a memory containing a client application, wherein the client application configures the processor (***Fig. 11 processor 1110 and furthermore***

Para 0068, 0069) a plurality of alternative streams of protected content (fig. 2 a multimedia stream source 210 i.e. "a plurality of alternative"), where each of the alternative streams of protected content are encrypted using common cryptographic information; obtain the common cryptographic information (fig. 2 encryptor 240 discloses same key is used for different streams i.e. "common cryptographic information" para 0054 discloses "...a single key per stream, then all alternate streams must use the same key"); request portions of content from at least the plurality of alternative streams of protected content (fig. 2 and para 0031 discloses multi-encoder 220 generates multiple encoded access units and gives alternate or multiple streams); access the protected content using the common cryptographic information; and playback the content. (Para 0032 discloses encryptor 240 where the streams are encrypted and fig. 2 and para 0054 discloses same key is used for different streams i.e. "common cryptographic information").

Amini does not explicitly request a top level index file.

However, Thorwirth discloses a top level index file (para 0009 discloses top level index file pointing to a number of alternate streams that contain actual video and audio data).

Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate the teachings of Thorwirth with the teachings of Amini in order for providing a playlist with a list of URLs pointing to TS container files that contain alternative chunks for the sections of a piece of content (Thorwirth, para 0010).

Regarding claim 3 Amini as modified by Thorwirth discloses wherein: each of the alternative streams of protected content is contained within a separate container file; and the top level index file identifies a container file containing each of the alternative streams (*Para 0004 of Thorwirth discloses sections of files in a container file and para 0009 and 0010 of Thorwirth discloses different media containers for adaptive streaming*).

Regarding claim 18, Amini discloses a method of performing adaptive bitrate streaming using a plurality of alternative streams of protected content, where each of the streams of protected content is protected using common cryptographic information, the method comprising: identifying a plurality of alternative streams of protected content using a playback device (*Fig. 2 a multimedia stream source 210 i.e. "a plurality of alternative"*) where each of the alternative streams of protected content are encrypted using common cryptographic information (*Fig. 2 a multimedia stream source 210 i.e. "a plurality of alternative"*) obtaining the common cryptographic information using the playback device; requesting portions of content from at least the plurality of alternative streams of protected content using the playback device; accessing the protected content using the playback device and the common cryptographic information; and playing back the content using the playback device. (*Fig. 2 encryptor 240 discloses same key is used for different streams i.e. "common cryptographic information" para 0054 discloses "...a single key per stream, then all alternate streams must use the same key"*);

Amini does not explicitly request a top level index file.

However, Thorwirth discloses a top level index file (*para 0009 discloses top level index file pointing to a number of alternate streams that contain actual video and audio data*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate the teachings of Thorwirth with the teachings of Amini in order for providing a playlist with a list of URLs pointing to TS container files that contain alternative chunks for the sections of a piece of content (Thorwirth, para 0010).

10. Claims 2, 4-8, 19-20 are rejected under pre-AIA 35 U.S.C. 103(a) as being unpatentable over Amini et al. (US 2004/0202320 A1) hereafter refers as to Amini in view of Thorwirth (WO 2011/103364 A1) as applied to claim 1 above, and further in view of Schnell (US 2009/0328228 A1).

Regarding claim 2 Amini as modified by Thorwirth discloses wherein: the alternative streams of protected content are alternative streams of protected video (*Para 0052 of Thorwirth discloses alternative streams of protected content*); and identifying at least a portion of the common cryptographic information used to access the encrypted portions of the frames of video (*para 0054 of Amini discloses a stream source 210 offers unencoded 13ccess units (i.e. frames) to the encoder, pars 0031 t see fig. 2; encryptor 240 where the streams are encrypted, para 0032, see fig. 2; same keys used for different streams (i.e. "common cryptographic Information")*); and the client application configures the processor to: obtain a protected frame of video from one of the alternative streams of protected video (*figs 4 and 5 of Thorwirth*

discloses Encryption is a sort of a content protecting method of preventing illegal copy redistribution, editing and the like of contents provided by a content provider (CP) in a manner of transforming data using a specific algorithm);

decrypt the encrypted portion of the protected frame of video using the identified portion of the common cryptographic information; and decode the frame of video. *(Para 0059 of Thorwirth discloses decryption of streamed file utilizes the content keys based on the sections).*

Though Amini as modified by Thorwirth teaches a stream is encrypted with an indexed stream (para 0060 of Amini, for example) However, Amini as modified by Thorwirth does not explicitly disclose each of the alternative streams of protected video includes DRM information indicating portions of frames of video that are encrypted and obtain DRM information indicating a portion of the protected frame of video that is encrypted; identify at least a portion of the common cryptographic information that is used to access the encrypted portion of the protected frame using the DRM information;

Schnell discloses each of the alternative streams of protected video includes DRM information indicating portions of frames of video that are encrypted (para 0034 OF discloses media device 202 also includes DRM platform 21 to receive the ORM license) and obtain DRM information indicating a portion of the protected frame of video that is encrypted (para 0002 discloses digital rights management (DRM) system, various devices that obtain or download digital media content can perform

actions on the protected media content as permitted by a DRM license and furthermore para); identify at least a portion of the common cryptographic information that is used to access the encrypted portion of the protected frame using the DRM information (para 0028-0031 discloses content server 122 and a DRM system 118 distribute protected media content 120 to various wireless media devices 106).

It would have been obvious to one of ordinary skill in the art at the time of the invention to use the teachings of Schnell to incorporate the DRM system with the teachings of Amini as modified by Thorwirth for the purpose of providing DRM associated features such as licenses that correspond to the media segments of the protected media (Schnell para 0004).

Regarding claim 4 Amini as modified by Thorwirth further in view of Schnell discloses wherein: the container files containing each of the alternative streams of protected content (*Para 0004 of Thorwirth discloses sections of files in a container file para 0009 and 0010 of Thorwirth discloses different media containers for adaptive streaming*) comprise at least one header including the common cryptographic information and information that describes the encoding of the stream of protected content; and the top level index file identifies the location of the at least one header within each container file. (*fig. 2 and para 0037 of Schnell discloses file header object 218 [DRM header element] has the ORM licenses for the protected media content, para 0036, written in a file segment 160, which has track objects 222 associated with the tracks or video tracks*).

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 5 Amini as modified by Thorwirth further in view of Schnell discloses wherein the common cryptographic information is a set of encryption keys. *(Para 0054 of Amini discloses encryptor 240 where the streams are encrypted and para 0032. Also see. fig. 2; same key is used for different streams i.e. ("common cryptographic information").*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 6 Amini as modified by Thorwirth further in view of Schnell discloses wherein: the common set of encryption keys is encrypted; and the client application configures the processor to securely receive cryptographic information used to unencrypt the common set of encryption keys. *(Para 0058 of Thorwirth discloses the content keys used to encrypt the content are encrypted using a key unique to a specific user or client device However, Thorwirth lacks the teaching of common cryptographic information. However, Amini teaches a content distribution with common cryptographic information for the alternate streams of content (para 0054, 0044 of Amini discloses multi-encoder 220 generates multiple encoded access units and gives alternate or multiple streams, para 0031, see fig. 2; same key is used for different streams, (protecting the content)).*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 7 Amini as modified by Thorwirth further in view of Schnell discloses wherein the client application configures the processor to obtain the common cryptographic information by requesting at least one header of a container file identified in the top level index file. ***(Para 0004 of Thorwirth discloses sections of files in a container file and para 0004 of Thorwirth, different media containers for adaptive streaming).***

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 8 Amini as modified by Thorwirth further in view of Schnell discloses wherein: each container file containing one of the alternative streams of protected content further comprises an index; and the client application configures the processor to request a portion of content from one of the container files by configuring the processor to: request at least a portion of an index from the container file; identify the portion of the container file containing a specific portion of content using the index; and request the identified portion of the container file. ***(Para 0053 Thorwirth server provides access to the client based on the selection or chunks requested).***

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 19 Amini as modified by Thorwirth further in view of Schnell discloses wherein: the alternative streams of protected content are alternative streams of protected video; each of the alternative streams of protected video ***(Para 0052 of Thorwirth discloses alternative streams of protected content)***; includes DRM

Art Unit: 2434

information indicating portions of frames of video that are encrypted and identifying at least a portion of the common cryptographic information used to access the encrypted portions of the frames of video; and the method further comprises: obtaining a protected frame of video from one of the alternative streams of protected video using the playback device *figs 4 and 5 of Thorwirth discloses Encryption is a sort of a content protecting method of preventing illegal copy redistribution, editing and the like of contents provided by a content provider (CP) in a manner of transforming data using a specific algorithm)* obtaining DRM information indicating a portion of the protected frame of video that is encrypted using the playback device; identify at least a portion of the common cryptographic information that is used to access the encrypted portion of the protected frame using the playback device and the DRM information; decrypting the encrypted portion of the protected frame of video using the playback device and the identified portion of the common cryptographic information; and decoding the frame of video using the playback device. *(Para 0028-0031 of Schnell discloses content server 122 and a ORM system 118 distribute protected media content 120 to various wireless media devices 106).*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

Regarding claim 20 Amini as modified by Thorwirth further in view of Schnell discloses wherein: each of the alternative streams of protected content is contained within a separate container file; and the top level index file identifies a container file containing each of the alternative streams. *(Para 0004 of Thorwirth discloses*

sections of files in a container file and para 0009 and 0010 of Thorwirth discloses different media containers for adaptive streaming).

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell as in claim 2 which is set forth above.

11. Claims 9-17 are rejected under pre-AIA 35 U.S.C. 103(a) as being unpatentable over Amini et al. (US 2004/0202320 A1) hereafter refers as to Amini in view of Thorwirth (WO 2011/103364 A1) and further in view of Schnell (US 2009/0328228 A1) as applied above in claim 4 and further in view of Soroushian et al. (US 2009/0132721 A1) herein after refer as to Soroushian.

Regarding claim 9 Amini as modified by Thorwirth further in view of Schnell discloses all claimed limitations except for wherein the container files are Matroska container files.

However, Soroushian discloses wherein the container files are Matroska container files. ***(see fig, 1c and para 0031, Matroska is MKV, discloses Soroushian is in the field of playback and distribution systems (abstract) and further teaches wherein the content is stored as chunks or elements in a Matroska container (MKV container composed of lists, chunks and/or elements).***

Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to use the teachings of Soroushian to incorporate with the teaching of Amini as modified by Thorwirth modified Schnell in order for the purpose of providing a MKV container for distributing high-definition downloadable content on the internet and peer-to-peer networks (Soroushian, para 0003).

Regarding claim 10 Amini as modified by Thorwirth and further modified Soroushian discloses wherein the common cryptographic information is stored in a common DRM Header element contained within a Tracks element within each of the Matroska container files containing the alternative streams of protected content. ***(Para 0052 of Thorwirth discloses the chunks are protected using different watermark payloads, i.e. "alternative streams of protected content")***

Regarding claim 11 Amini as modified by Thorwirth and further modified Soroushian discloses wherein: the common cryptographic information is a set of encryption keys; and the set of encryption keys is encrypted and stored within the DRM Header element. ***(para 0036 0037 of Schnell file header object 218 (DRM header element) has the DRM licenses for the protected media content,; tracks objects 22 which has the track attributes and segment data to identify the content).***

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 12 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein the client application configures the processor to: obtain cryptographic information that can be used to unencrypt the set of encryption keys stored within the DRM Header element; and unencrypt the set of encryption keys stored within the DRM Header element using the obtained cryptographic information. ***(Para 0036 of Schnell file header object 218 (DRM header element) has the DRM licenses for the protected media content).***

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 13 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein the client application configures the processor to obtain the DRM Header element by requesting a Tracks element from one of the Matroska container files identified in the top level index file. *(See para 0004 of Thorwirth, different media containers for adaptive streaming, para 0009 of Thorwirth [background]) and a top level index file identifies a container file containing each of the alternative streams and furthermore para 0009 of Thorwirth top level index file pointing to a number of alternate streams that contain actual video and audio data,).*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 14 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein: each Matroska container file containing an alternative stream of protected content further comprises an index contained within a Cues element ; and the client application configures the processor to request a portion of content from one of the Matroska container files by configuring the processor to: request at least a portion of a Cues element from a Matroska container file containing one of the alternative streams of protected content; identify a portion of the Matroska

container file containing a specific portion of content using at least one CueTrackPosition element within the Cues element; and request the portion of the Matroska container file identified by the at least one CueTrackPosition element.

(Schnell teaches file index 158 for each me and the corresponding segments, para 0031, see fig. 2; and (track objects 222 has the attributes and information included in a track ("Cues element"), para 0036, 0037 see fig. 2 and includes size, type, mode 10, Index, 'Algorithm 10 with index. para 0041 and table under it).

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 15 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein the index comprises a plurality of CueTrackPosition elements that identify the location of a plurality of Cluster elements within the Matroska container file. *(See fig. 1c and Para 0031, discloses (Matroska is MKV) and furthermore Para 0037 of Soroushian table with hierarchical index Identifiers with CUEPOS_ID, table above).*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 16 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein: each Cluster element contains a plurality of BlockGroup elements including encoded portions of content; a CueTrack Position

element indexes each Cluster element; and each CueTrackPosition element includes the size of the Cluster element that it indexes. *(fig. 2 para 0031 of Schnell file index 158 for each file, and the corresponding segments (segments can be interpreted as track elements or comprising of cluster element and furthermore fig 2 and para 0036, 0037 and para 9041 of Schnell teaches size, type, mode ID, index, Algorithm ID with index).*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Regarding claim 17 Amini as modified by Thorwirth modified Schnell and further modified Soroushian discloses wherein the alternative streams of protected content are alternative streams of protected video *(para 0052 of Thorwirth discloses CDN stores six alternative chunks [video] created for each section with different encoding parameters and two different watermark payloads [alternative streams of protected content]);* each of the Matroska container files containing an alternative stream of protected video includes a DRMInfo element in a BlockGroup element containing a protected frame of video *(para 0052 of Thorwirth discloses the chunks are protected using different watermark payloads),* where the DRMInfo element indicates at least a portion of the protected frames of video that is encrypted and identifies at least a portion of the common cryptographic information used to access the at least one encrypted portion of the protected frame of video *(para 0036 of Schnell file header object 218 i.e. "DRM header Element" has the DRM licenses for the*

protected media content); and the client application configures the processor to access video frames in a stream of protected video using the common cryptographic information by configuring the processor to: obtain a BlockGroup containing a protected frame of video; parse the BlockGroup element to obtain a DRMInfo element indicating at least a portion of the protected frame of video that is encrypted using an encryption key from the set of encryption keys; decrypt the at least one encrypted portion of the protected frame of video using the identified encryption key; and decode the frame of video. *(figs 4 and 5 of Thorwirth discloses Encryption is a sort of a content protecting method of preventing illegal copy redistribution, editing and the like of contents provided by a content provider (CP) in a manner of transforming data using a specific algorithm);*

The examiner supplies the same rationale for the combination of Amini as modified by Thorwirth further in view of Schnell and further in view of Soroushian as in claim 9 which is set forth above.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to ABIY GETACHEW whose telephone number is (571)272-6932. The examiner can normally be reached on Monday to Friday 8Am to 4:30Pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kambiz Zand can be reached on (571)272-3811. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Art Unit: 2434

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/DANT SHAFER HARRIMAN/
Primary Examiner, Art Unit 2434
04/06/2015

/ABIY GETACHEW/
Examiner, Art Unit 2434

A.G.
April 3, 2015

Notice of References Cited	Application/Control No. 14/564,003	Applicant(s)/Patent Under Reexamination KIEFER ET AL.	
	Examiner ABIY GETACHEW	Art Unit 2434	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-2004/0202320 A1	10-2004	Amini et al.	380/036
*	B	US-2009/0328228 A1	12-2009	Schnell, Patrik	725/26
*	C	US-2009/0192721 A1	05-2009	Soroushian et al.	703/231
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	THORWIRTH NIELS et al. August 25, 2011 WO 2011/03364 A1
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.02(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

OpenDML AVI File Format Extensions

Version 1.02

OpenDML AVI M-JPEG File Format Subcommittee
February 28, 1996

Introduction

The OpenDML File Format Subcommittee is defining an AVI *compatible* file format that addresses the particular needs of professional video. As such this document relates specific proposals to these needs.

Further work by OpenDML will elaborate on extensions to the Codec and Vidcap specifications of Video for Windows to meet the needs of professional video.

Scope

This document describes the proposed format of the OpenDML compliant AVI extensions, specifically for the Motion JPEG DIB AVI file. This format is an extension to the AVI M-JPEG DIB as defined in the *JPEG DIB FORMAT* technical note from Microsoft. The format is based on the ISO 10918 defined JPEG format.

It is assumed that the reader is familiar with JPEG as defined in the ISO 10918 document. For additional information on JPEG, see the *ISO 10918 Information technology -- Digital compression and coding of continuous-tone still images: Requirements and guidelines*.

For additional information about the JPEG DIB format, see Microsoft Multimedia Technical Note *JPEG DIB FORMAT*.

For additional information about RIFF files see the Microsoft Windows Software Development Kit *Multimedia Programmer's Guide* and *Multimedia Programmer's Reference*.

For additional information on installable compressors and decompressors, see the *Video Compression/Decompression Drivers* technical note from Microsoft.

Portions of the above Microsoft documents have been reprinted with permission of Microsoft Corp.

OpenDML Sub-Committee on File Format: High Level Goals

- Interoperability and AVI Compatibility
- Large File Support
- Frame and Field Indexing
- Interleaving Audio with Video
- Source and Header Information Storage
- OpenDML File Format Certification Code
- Microsoft Certification and Communication

Version History

Version 1.0	November 13, 1995	First Release
Version 1.1	December 18, 1995	Includes small changes from Microsoft

1.0 Overview of Profession Video Requirements

Goal

Provide interoperability between different hardware and software vendors' Motion JPEG codecs. Support Video for Windows AVI file format for MJPG video, while improving the performance of these systems.

Issues

AVI: Support Current AVI Format

The goal is to support interoperability by standardizing on the AVI file format for Motion-JPEG video. This implies a minimum compatibility to support the standard AVI file access interface, as well as the vidcap and codecs that are part of Video for Windows. This means support of the MJPG DIB specification, supporting the RIFF file format as well as the standard AVI stream headers, allowing for optional audio interleaving, and supporting one storage of one frame of video per ##dc data chunk in the LIST 'movi' chunk of the AVI file.

60 Fields Per Second vs. 30 Frames Per Second; or 24 Frames Per Second For Film

Professional Video Applications require the ability to sequence individual fields of the video data for certain playback rates and video effects. For example, in order to do a slow motion effect, it is not sufficient to just repeat the frames; the fields must be individually repeated. For example, as sequence of fields in a file numbered 123456... should be played at half speed as 11223344... rather than 12123434... . As such, the individual fields should be accessible. AVI requires that frames are stored per data chunk. Storing individual fields in each chunk would break AVI applications.

The extended AVI format will allow for access to individual fields. In addition, the format will allow for storage of 24 frame based files, such that file readers can properly convert between 24 frame based data storage and 25 or 30 frame PAL and NTSC playback.

Improved Tombstone and Header Information

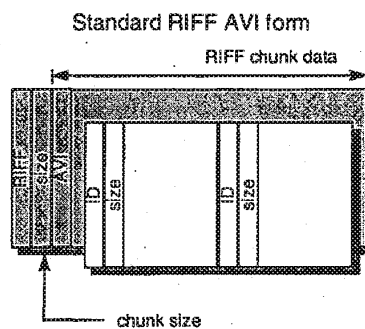
Professional video applications need extended information such as starting timecode and reel ID to be contained inside the AVI file. More advanced information such as the location of timecode discontinuities, etc. are also desirable.

2.0 Increased AVI File Size

Introduction: RIFF Chunk Format

The AVI file begins as a standard RIFF chunk (note that this is a non-standard representation that shows the 4-byte size):

RIFF (size) 'AVI'



There is only one RIFF chunk per file. RIFF sub-chunks may be either LIST chunks or regular sub-chunks. The LIST chunk obeys the same structure and may have regular or LIST sub-chunks; all other RIFF sub-chunks have just an ID identifier and a size. These regular sub-chunks may not have sub-chunks.

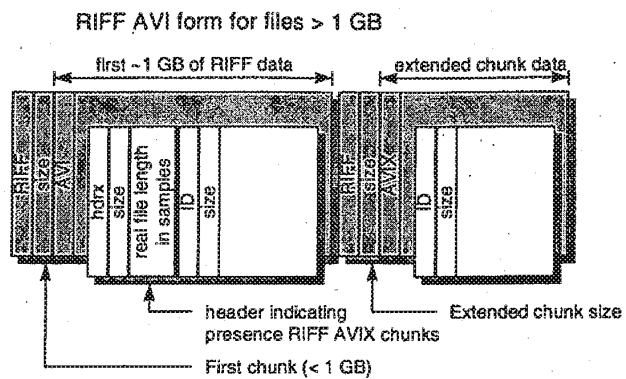
Increased File Size Limits (>> 1 GB)

The current RIFF file format implies a maximum chunk size of 4 GB because the size is stored as a 32-bit value. However, limitations to the RIFF parser code and MCI/AVI limits the file to only 1 GB. At data rates of 10 MB/s, a 1 GB file will last less than 2 minutes.

Extension for File Size > 1 GB

An AVI file can be extended beyond 1 GB by placing more than one RIFF chunk in the same file (which is legal). Each RIFF chunk can have a maximum size of 1 GB. Standard AVI applications will see the first RIFF chunk (RIFF 'AVI') as a standard AVI file, and this chunk should be complete. Information that the file is extended over 1 GB should be placed in the Extended AVI header (see below). Subsequent RIFF chunks will be identified by the AVIX (for AVI extended) chunk id. These RIFF 'AVIX' chunks do not need to contain further data. It is expected that the AVIX chunks only contain LIST 'movi' data. All other AVI information should be stored in the first (RIFF 'AVI') chunk so that it is accessible by all applications.

BEST AVAILABLE COPY

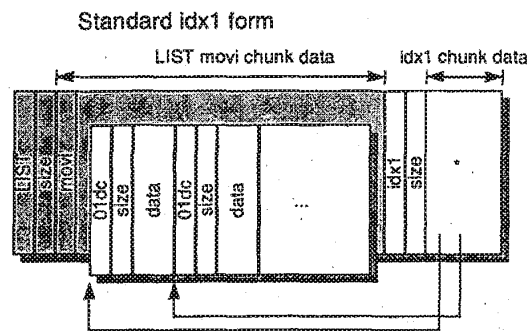


BEST AVAILABLE COPY

3.0 Frame and Field Indexing

Introduction: 'idx1' Index Chunk

The AVI file format specifies an optional 'idx1' chunk which contains a list of the offsets and chunk types of every chunk inside the LIST 'movi' chunk. This list is used to make AVI playback and seeks more efficient because the location of each frame of video can be found without scanning through each sub-chunk of the LIST 'movi' data.



* data of idx1 chunk is an index to each movi sub-chunk, includes offset and length of each chunk

Issues

Allow Field Indexing

In order to support field accurate indexing while still retaining AVI compatibility, the video data must be stored one frame per 'movi' chunk, while an index must be present that gives the locations of both fields (if they exist) within the frame. Applications could then use field based codecs to perform field based effects (e.g., slow motion, etc.).

Allow Incremental Growth of Files

The 'idx1' chunk follows the LIST 'movi' chunk in the file. As such, it must be moved in order to insert new data into the file. An interleaved frame index would be more efficient for the purposes of file growth.

Minimize Disk Seeks

In playback, there are several causes of inefficiency. One is that the index is at the end of the file, requiring a large seek when opening the file. The other is that the index itself contains all the chunks inside the 'movi' data, not just those for one AVI stream. As such, the index must be preprocessed so that the location of each frame in the index can be found.

Proposed Index List

Overview

Instead of using a index chunk like 'idx1', a new index structure is proposed. The index has the following main characteristics:

- The can either be a single index chunk, or a two tiered index, with a super index pointing to interleaved index segments in the 'movi' data.
- The index can include locations of the fields within a frame.
- The index has no restrictions to 4 GB file size.
- There is an index per data stream (if necessary).

The advantages of the index are as follows:

- The index can contain the locations of the individual fields in each chunk as well as the chunk information as a whole. The 'idx1' chunk only contains indices of the frames.
- The index segments are interleaved within the 'movi' data, meaning that the file can be grown. The 'idx1' chunk is present at the end of the file. It must be moved for the file to grow.
- The index segment is smaller than an 'idx1' chunk, so the amount of data needed to be read in order to access a particular frame is less. Also, if the index is interleaved before its data (i.e., less efficient in write), then playback will be more efficient than the 'idx1' chunk.
- The index lists are present per stream number (##). As such, the list can be accesses like an array, unlike the 'idx1' chunk which contains entries for all 'movi' chunks from all streams.

The index is implemented as an index of indexes (optional) and an index of chunks. Both are based on a base index form.

Base Index Form 'indx'

Thus the actual implementation is based on a base index form 'indx':

```
struct _aviindex_chunk {
    FOURCC fcc;
    DWORD cb;
    WORD wLongsPerEntry; // size of each entry in aIndex array
    BYTE bIndexSubType; // future use. must be 0
    BYTE bIndexType; // one of AVI_INDEX_* codes
    DWORD nEntriesInUse; // index of first unused member in aIndex array
    DWORD dwChunkId; // fcc of what is indexed
    DWORD dwReserved[3]; // meaning differs for each index
                        // type/subtype. 0 if unused

    struct _aviindex_entry {
        DWORD adw[wLongsPerEntry];
    } aIndex[ ];
};
```

The actual size of the entries of the aIndex array are entered into wLongsPerEntry. Between the cb field and wLongsPerEntry the actual size of the array is known. The field nEntriesInUse allows for a chunk to be allocated longer than the actual number of used elements in the array.

The types defined for the bIndexType and bIndexSubtype field are the following:

```
// bIndexType codes
//
#define AVI_INDEX_OF_INDEXES 0x00 // when each entry in aIndex
```

```

// array points to an index chunk
#define AVI_INDEX_OF_CHUNKS 0x01 // when each entry in aIndex
// array points to a chunk in the file
#define AVI_INDEX_IS_DATA 0x80 // when each entry is aIndex is
// really the data

// bIndexSubtype codes for INDEX_OF_CHUNKS
//
#define AVI_INDEX_2FIELD 0x01 // when fields within frames
// are also indexed

```

AVI Standard Index Chunk

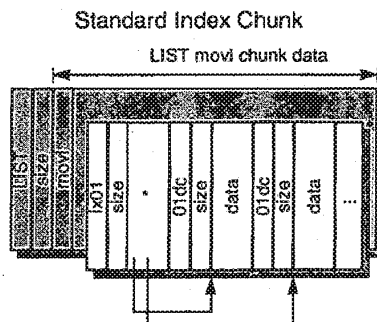
The AVI Standard Index chunk contains information which indexes AVI frames.

```

typedef struct _avistdindex_chunk {
    FOURCC    fcc;           // 'ix##'
    DWORD     cb;
    WORD      wLongsPerEntry; // must be sizeof(aIndex[0])/sizeof(DWORD)
    BYTE      bIndexSubType; // must be 0
    BYTE      bIndexType;    // must be AVI_INDEX_OF_CHUNKS
    DWORD     nEntriesInUse; //
    DWORD     dwChunkId;     // '##dc' or '##db' or '##wb' etc..
    QWORD     qwBaseOffset;  // all dwOffsets in aIndex array are
                           // relative to this
    DWORD     dwReserved3;   // must be 0
    struct _avistdindex_entry {
        DWORD    dwOffset;    // qwBaseOffset + this is absolute file offset
        DWORD    dwSize;      // bit 31 is set if this is NOT a keyframe
    } aIndex[ ];
} AVISTDINDEX, * PAVISTDINDEX;

```

A single standard index chunk can only index data within a 4 GB region. The dwOffset field points to the start of the data itself, and not to the start of the RIFF chunk for that field.

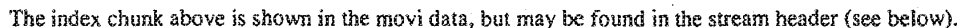


* data of Standard AVI index chunk is an index to each movi chunk data, includes offset and length of chunk.

The index chunk above is shown in the 'movi' data, but may be found in the stream header (see below).

The AVI Field Index Chunk is the same as the Standard Index Chunk except that it contains the locations of each field in the frame.

The field `wLongsPerEntry` is set to 3 because of the addition of the `dwOffsetField2` field in the array. This is indicated by setting the sub type to `AVI_INDEX_2FIELD`.



The Super Index Chunk is an index of indexes and is always found in the 'indx' chunk of an AVI file. It is defined as follows:

BEST AVAILABLE COPY

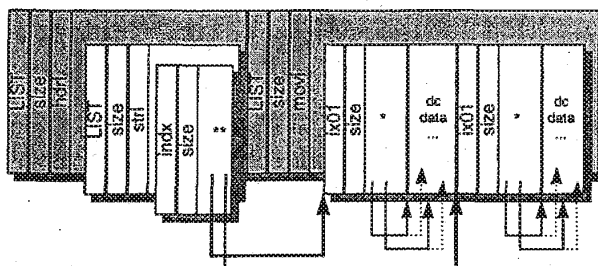
```

WORD    wLongsPerEntry; // must be 4 (size of each entry in aIndex array)
BYTE    bIndexSubType;  // must be 0 or AVI_INDEX_2FIELD
BYTE    bIndexType;     // must be AVI_INDEX_OF_INDEXES
DWORD   nEntriesInUse;  // number of entries in aIndex array that
                        // are used
DWORD   dwChunkId;      // '##dc' or '##db' or '##wb', etc
DWORD   dwReserved[3];  // must be 0
struct _avisuperindex_entry {
    QUADWORD qwOffset; // absolute file offset, offset 0 is
                       // unused entry??
    DWORD    dwSize;   // size of index chunk at this offset
    DWORD    dwDuration; // time span in stream ticks
} aIndex[ ];
) AVISUPERINDEX, * PAVISUPERINDEX;

```

The `bIndexSubType` is set to the type of index that the Super Index points to. If the index chunks are Standard Index Chunks, then the value should be 0. If the index chunks are AVI Field Index Chunks, then the value should be `AVI_INDEX_2FIELD`. This implies that a stream cannot mix Field and Standard Index Chunks.

Super Index Chunk



* data of ##ix chunk is an index to each movi sub-chunk data, includes offset and length of chunk, and possibly the offset to the second field.
 ** data of Super Index indx chunk is an index to each ##ix movi sub-chunk.

If we use the `##pc` chunks, then they must be indexed by their own Index List: Palette changes should not appear in the same Index List as `##dc` chunks, since they eliminate the ability to access an element (frame) directly without having to scan the list.

Index Locations in RIFF File

Unlike the `'idx1'` chunk, a single index is stored per *stream* in the AVI file. An `'indx'` chunk follows the `'strf'` chunk in the LIST `'strf'` chunk of an AVI header. This `'indx'` chunk may either be an index of indexes (super index), or may be an index to the chunks directly. In the case of video, this means that the chunk is either a `AVISUPERINDEX` or an `AVIFIELDINDEX/AVISTDINDEX`.

If the 'indx' chunk is a standard or field index chunk (i.e., not an index of indexes) then the stream has only one index chunk and there is none in the 'movi' data.

If the 'indx' chunk is a Super Index, then the corresponding index chunks are marked with 'ix##' in the 'movi' data. The ## is the stream number, the same as for the ##dc or ##wb. The index chunks can be either standard or field index chunks.

A file can be easily grown if it has a standard index in the 'indx' chunk position. The chunk can be moved to a new 'ix##' chunk, and a new super index can be inserted into the stream header ('indx' position). New 'ix##' chunks can be added to grow the file.

Note the reversal of the '##ix' to 'ix##'. This is for AVI backward compatibility. Note that INDEX_IS_DATA streams remain as '##ix'.

Here is a sample RIFF file in the format shown by the RIFFWALK utility:

```
RIFF (7F038718) 'AVI'
LIST (0001084C) 'hdlr'
  avih (00000038)
  LIST (000044E0) 'strl'
    strh (00000038)
    strf (00000428)
    indx (00003FF8)      <- video superindex
    vprp (00000064)      <- video property header
  LIST (0000405E) 'strl'
    strh (00000038)
    strf (00000012)
    indx (00003FF8)      <- audio superindex
  LIST (0000404C) 'strl'
    strh (00000038)      <- timecode stream...
    strf (00000000)
    indx (00003FF8)      <- timecode superindex
  LIST (00000060) 'odml'
    dmlh (00000054)      <- size still temp.
LIST (7F000000) 'movi'
  01wb (00002dF0)
  00db (00030004)
  01wb (00002dF0)
  00db (00030004)
  ... time passes
  02ix (00003FF8)        <- data chunk for timecode stream
  ... time passes
  ix00 (00003FF8)        <- partial video index
  ix01 (00003FF8)        <- partial audio index
  .. etc ..
  idx1 (00030000)        <- original index (optional)
RIFF (7F038718) 'AVIX'
```

4.0 Interleaving Audio with Video

Issues

Interleaving Rate

Files for Editing

Files for Playback

Single or Multiple Files for Video and Audio

Multiple Tracks of Audio Storage

Mono and Stereo Audio

Support of multiple streams of data exists in AVI. A proposal to recommend that audio be stored as two mono streams should be examined. Not all applications would be able to play this back in stereo, however.

Audio / Video Time Stamping (Sync Point List)

It is necessary to add time stamping information to account (and correct) for drift between audio and video when using hardware that is not able to maintain a perfect synchronization between audio and video (via a hardware connection) and when there exists a drift in the crystal frequency on audio hardware.

This can be accomplished by adding a sync point list into the file. A sync point list identifies two samples in different streams that are to be played at the same time. Thus drift can be corrected by identifying sync points and if necessary, correcting for the drift during playback.

Definition of Sync Point List is based on the base index list chunk and is forthcoming...

Notes on Interleaving Audio and Video

With the exception of AVI files playing off CD-ROM drives, it is not necessary to interleave the audio and video into REC chunks. In fact, with higher video and audio data rates, it is probably more efficient to store the audio in large consecutive blocks. As such, the audio can be interleaved in large blocks (not necessarily done on a frame basis).

Audio data chunks may or may not be in integral frames, or multiple of frames. The other option would be a constant number of bytes. Also, audio may be interleaved before or after the corresponding video. This is just a tradeoff between record and playback performance. Typically, saving audio chunks after video in record will give better record performance, which is important because hard drives tend to be less performant during write operations. However, during playback, multiple streams and cuts may be played, reducing the drive performance.

Do we still want this section, or do we want to drop it?

5.0 Source and Header Information Storage

Goal

The goal is to implement a standard way to store certain header and tombstone information in an AVI file so that other applications can interpret and use this data. In many cases, the header data is not uniformly set between different AVI files.

Required Information

Main AVI Header (avih)

Total Frames

The `dwTotalFrames` field indicates the size (number of frames) within the first RIFF 'AVI' chunk.

AVI Stream Header (strh)

Quality Information

AVI defines the quality of a clip by a DWORD value from 0 to 10000. This value seems arbitrary. For MJPG files, a standard quality should be defined. The quality can be defined by either of these three modes:

- constant average data rate
- constant Q-table
- lossless

Lossless is not part of the Baseline process. As such it requires the definition of a new Process. Under baseline, lossless can be achieved by setting the Q-tables to unity¹.

Basically, under constant data rate, the desired data rate for the file should be specified (in kB/s). This, plus the number of fields per frame, should be able to determine the Q-tables required to compress the image.

Under constant Q-table, there are two possibilities. Either the absolute Q-tables are given (8x8 matrices) per component (in our case, 2), or a factor which is used to generate any Q-table of a standard one. The constant Q-table file would of course have the same tables repeated per field, as per the MJPG DIB spec. The advantage of a Q-factor is the reduction of header information. However, tweaked Q-tables would no longer be possible.

¹ While a lossless (or pseudo lossless) process can be defined by setting the Q-tables to unity, the ISO specification of lossless JPEG compression is a different process. Lossless encoding as defined by ISO does not use DCT transformed data. For a sample to be encoded, a predictor is formed from the reconstructed values of up to three neighborhood samples. This gives a prediction of the of the sample to be encoded. The prediction is subtracted from the actual value of the sample and the difference is losslessly entropy coded using either Huffman or arithmetic coding. As far as forming a type of pseudo lossless compression from a baseline JPEG process with unity Q tables, there is the potential of some slight loss of data during the DCT process due to arithmetic precision. Practically speaking, there is no real image quality loss, but the word *lossless* implies a guarantee of reconstructing the source image on a bit by bit basis.

It seems that on this point, there was no clarification possible. Do we drop the section?

Scale / Rate

The scale and rate parameters will define the correct ratio for different video standards. Known ratio values are (rate/scale):

NTSC	30000/1001
PAL	25/1

rcFrame

The on-disk RECT coordinates are 16-bit values. The RECT structure in NT is a 32-bit (LONG) for each coordinate, but the on-disk ones remain 16-bit.

AVI Stream Format (strf)

Height

The MJPG DIB spec lists that heights less than 288 are for single frames (one chunk) and greater than 288 are for interleaved fields. However, there is no real connection between frame height and interleaving. The **biHeight** parameter refers to the raw height of the frame (interleaved or not). OpenDML codecs that operate in field by field modes will be passed this value divided by 2 to get the field height.

Extended AVI Header (dmlh)

```
typedef struct {
    DWORD dwTotalFrames;
} ODMLExtendedAVIHeader;
```

Total Frames

The **dwTotalFrames** field indicates the real size of the AVI file. Since the same field in the Main AVI Header 'avih' indicates the size within the first RIFF 'AVI' chunk.

Video Properties Header (vprp)

The video properties header identifies video signal properties associated with a digital video stream in an AVI file. This header attempts to address two main video properties:

- The type of video signal (PAL, NTSC, etc., as well as the resolution of the video signal).
- The framing of the compression within a video signal.

The parameters can be used to uniquely describe a video signal.

```
typedef struct {
```

```

DWORD      VideoFormatToken;
DWORD      VideoStandard;
DWORD      dwVerticalRefreshRate;
DWORD      dwHTotalInT;
DWORD      dwVTotalInLines;
DWORD      dwFrameAspectRatio;
DWORD      dwFrameWidthInPixels;
DWORD      dwFrameHeightInLines;
DWORD      nbFieldPerFrame;
VIDEO_FIELD_DESC FieldInfo[nbFieldPerFrame];
) VideoPropHeader;

```

Video Format Token

```

enum { FORMAT_UNKNOWN, FORMAT_PAL_SQUARE, FORMAT_PAL_CCIR_601,
      FORMAT_NTSC_SQUARE, FORMAT_NTSC_CCIR_601, ... } VIDEO_FORMAT;

```

The format token indicates that a known standard is defined for the following data fields. Those fields must be filled, but their value can be expected to be the defined standard. If the format is defined as FORMAT_UNKNOWN then the fields may contain special values. Known tokens are defined in the table below:

Token	Standard	Refresh Rate	H-Total	V-Total	Frame Aspect Ratio	Frame Width	Frame Height	Pixel Aspect Ratio (derived)
NTSC	NTSC	60	858	525	0x00040003 (4:3)	720	485	2160:1940
CCIR 601	NTSC	60	780	525	0x00040003 (4:3)	640	480	1:1
NTSC SQUARE	PAL	50	864	625	0x00040003 (4:3)	720	576	2160:2304
PAL	PAL	50	944	625	0x00040003 (4:3)	768	576	1:1
CCIR 601	PAL	50	944	625	0x00040003 (4:3)	768	576	1:1
PAL SQUARE	PAL	50	944	625	0x00040003 (4:3)	768	576	1:1

Video Standard

```

enum { STANDARD_UNKNOWN, STANDARD_PAL, STANDARD_NTSC, STANDARD_SECAM }
      VIDEO_STANDARD;

```

Defines standards such as NTSC, PAL etc. Implicitly defines vertical refresh rate.

Vertical Refresh Rate

Used when an unknown standard is specified. Normally, 60 for NTSC, and 50 for PAL.

H-Total in T

Defines the horizontal total, in T (one luminance sample: pixel)

V-Total in Lines

Defines the vertical total, in lines.

Active Frame Aspect Ratio

The aspect ratio is stored as a DWORD value with a word each storing the x:y ratio. For example, 1 to 1 is 0x00010001. Standard values for television is 4:3 or 16:9. This value can be used with the frame width and height to calculate the pixel aspect ratio.

Active Frame Width in Pixels

Defines the active frame width in pixels. The bitmap might digitize a region that is smaller or bigger than the active video width.

Active Frame Height in Lines

Defines the frame height in lines. The bitmap might digitize a region that is smaller or bigger than the active video height.

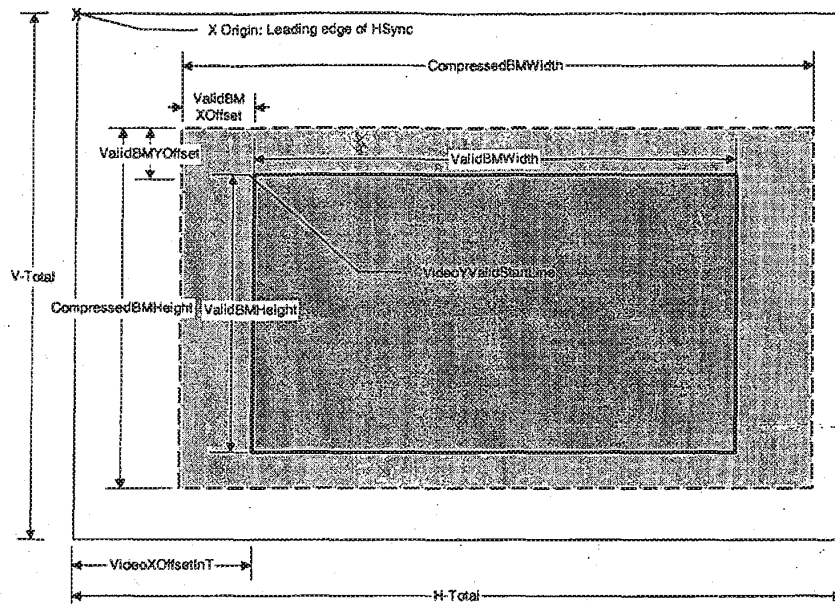
Number of Fields Per Frame

One or two, depending on whether the video is interlaced or progressive.

Field Framing Information

The field framing information defines where the compressed image is with respect to the video signal. The data is present for each field (and may be different).

```
typedef struct {  
    DWORD    CompressedBMHeight;  
    DWORD    CompressedBMWidth;  
    DWORD    ValidBMHeight;  
    DWORD    ValidBMWidth;  
    DWORD    ValidBMXOffset;  
    DWORD    ValidBMYOffset;  
    DWORD    VideoXOffsetInT;  
    DWORD    VideoYValidStartLine;  
} VIDEO_FIELD_DESC;
```



Compressed Bitmap Height and Width

The compressed bitmap height and width represent the size of the compressed image. For JPEG, these values are multiples of 8.

Valid Bitmap Height and Width, X and Y Offset

The valid bitmap height, width and x and y offsets represent the size of the valid data within the compressed bitmap. Because padding may be required when compressing, it is not guaranteed that all the data within the compressed image is valid. Note that compressing blanking is still valid. In the case where all the compressed bitmap comes from the video signal, then the valid height and width are equal to the compressed height and width, and the offsets are 0.

Valid X-Offset In T

The VideoXOffsetInT is used to locate the x position of the start of the valid bitmap with reference to the video signal. The value is a measurement in units of T, which is one luminance sampling clock, from the leading edge of the horizontal sync pulse (CCIR 624-3).

Valid Y Start Line

The VideoYValidStartLine field is used to locate the line that the valid bitmap starts on. This value will be different for each field. (CCIR 624-3).

A typical value for the Framing Information for NTSC CCIR 601 would be:

Video Type	Compressed BM Height	Compressed BM Width	Valid BM Height	Valid BM Width	Valid BM X Offset	Valid BM Y Offset	Video X Offset In T	Video Y Valid Start Line
Field 1	248	720	248	720	0	0	122	20
Field 1	248	720	248	720	0	0	122	283

Source and Timecode Information

The following information is used to describe timecode and source information inside an AVI file.

Base Timecode Structure

All SMPTE timecode information is stored in the following format:

```
typedef union _timecode {
    struct {
        WORD    wFrameRate; // 0 is 30 drop. do we need other drop
                        // frame rates?
        WORD    wFrameFract; // fractional frame. full scale is always 0x10000
        LONG    lFrame;
    };
    __int64    qwAll;
} TIMECODE, *PTIMECODE;
```

This format allows for a frame, and a frame fractional value to be specified, with the frame rate stored in the low word of the value. As the values are stored, two timecodes can be compared and subtracted. For drop frame code, no lFrame values are skipped, so a drop frame timecode of 1:00:00;00 would be a lFrame value of 107892.

Timecode Discontinuity Table (Stream) 'tcdl'

The following structure defines a timecode discontinuity table. This would be stored as a stream itself. The table would either be a table alone in the 'indx' chunk of that stream, or if the table is large, an index to the table could be stored with the table itself stored as a ##ix in the 'movi' data.

```
#define FILM_SEQUENCE_NONE 0
#define FILM_SEQUENCE_AABBBCCDDD 1
#define FILM_SEQUENCE_AAABBBCCDDD 2

// structure of a timecode discontinuity table. note that this fits
// within the definition of the structure of an index chunk
//
typedef struct _timecode_dl_chunk {
    FOURCC    fcc; // '##ix'
    DWORD     cb; // sizeof this structure less 8
    WORD      wLongsPerEntry; // must be 4 or more
    BYTE      bIndexSubType; // must be 0
    BYTE      bIndexType; // AVI_INDEX_IS_DATA
    DWORD     nEntriesInUse; // index of first unused entry in aIndex array
    DWORD     dwChunkId; // 'tcdl' (timecode discontinuity list)
    DWORD     dwReserved[3]; // future, must be 0
    struct _timecode_dl_entry {
        QUADWORD    qwTick; // time in terms of this stream's tick rate
        TIMECODE     timecode; // timecode
        DWORD        dwUser; // timecode user data
    } struct {
```

```

        DWORD ColorFrame : 4;        // Which frame in color sequence
        DWORD ColorSequence : 4;      // Duration in frames of complete sequence
        DWORD FilmFrame : 5;          // Offset into pull-down sequence
        DWORD FilmSequenceType : 3;   // One of FILM_SEQUENCE_XXX defines
        DWORD Reserved : 16;          // Future use - set to 0
    } Flags;
    TCHAR szReelId[32]; // source id
} aIndex[ ];
);

```

The Flags structure can optionally specify color framing, and film pull-down information. If all fields are 0 then no such optional information is present. Notice that there is no explicit flag for drop frame NTSC since that information is already stored in the TIMECODE structure.

If ColorSequence is non-zero then ColorFrame contains a frame number from 0 to ColorSequence-1 which specifies the color framing of the frame specified in the timecode structure.

If FilmSequenceType is not FILM_SEQUENCE_NONE, then FilmFrame contains the offset into the pull-down sequence corresponding to the frame specified in the timecode structure. For example, if FilmFrame is 1 and FilmSequenceType is FILM_SEQUENCE_AAABBBCCDDDD then the referenced frame is a BB frame. Again, if FilmFrame is 1 and FilmSequenceType is FILM_SEQUENCE_AAABBBCCDD then the referenced frame is an AB frame.

Timecode Stream 'time'

This structure defines a timecode value for every tick of the timecode stream

```

// structure of a timecode stream data chunk. for storing every
// event in a timecode stream rather than just the discontinuities
//
typedef struct __timecode_stream_chunk {
    FOURCC fcc; // '##ix'
    DWORD cb; // sizeof this structure less 8
    WORD wLongsPerEntry; // must be 3
    BYTE bIndexSubType; // must be 0
    BYTE bIndexType; // AVI_INDEX_IS_DATA
    DWORD nEntriesInUse; // index of first unused entry in aIndex array
    DWORD dwChunkId; // 'time' (timecode stream)
    DWORD dwReserved[3]; // future, must be 0
    struct __timecode_stream_entry {
        TIMECODE timecode; // timecode
        DWORD dwUser; // timecode user data
        struct {
            DWORD ColorFrame : 4; // Which frame in color sequence
            DWORD ColorSequence : 4; // Duration in frames of complete sequence
            DWORD FilmFrame : 5; // Offset into sequence
            DWORD FilmSequenceType : 3; // One of FILM_SEQUENCE_XXX defines
            DWORD Reserved : 16; // Future use - set to 0
        } Flags;
    } aIndex[ ];
};

```

Film Transfer Log Information

```

#define FILM_LOG_EVERTZ 1
#define FILM_LOG_FLEX 2

// structure of a film transfer log. Note that this fits
// within the definition of the structure of an index chunk
//

```

```

typedef struct _film_transfer_chunk {
    FOURCC   fcc;           // '##ix'
    DWORD    cb;            // sizeof this structure less 8
    WORD     wLongsPerEntry; // must be 4 or more
    BYTE     bIndexSubType;  // must be 0
    BYTE     bIndexType;     // AVI_INDEX_IS_DATA
    DWORD    nEntriesInUse;   // index of first unused entry in aIndex array
    DWORD    dwChunkId;       // 'film' (film transfer log)
    DWORD    dwReserved[3];   // future, must be 0
    struct _film_transfer_header {
        DWORD    dwLogType;    // Which type of FILM_LOG_XXX file format
        DWORD    dwHeaderSize; // Size of this _film_transfer_header structure
        DWORD    dwReserved[4]; // Future use - set to 0
        // DWORD    dwUserDefined[]; // Optional user data
    } FilmHeader;
    // BYTE  bTransferLog[];    // The actual transfer log begins here
};

```

The size of the FilmHeader is given by the dwHeaderSize. This allows for user defined additions to the header, which appear at the end of the FilmHeader. The ASCII film transfer log begins immediately after the FilmHeader structure. The size of the ASCII data can be determined from the dwHeaderSize of the header and the cb size of the entire structure. The ASCII data should be padded with NULLs to the nearest DWORD structure size.

The contents of the bTransferLog array is byte for byte the contents of the ASCII transfer log file of the type given in the dwLogType field. For now, two types are defined:

FILM_LOG_EVERTZ: the detailed .FTL file format can be obtained from
 Evertz Microsystems, Ltd.
 3465 Mainway
 Burlington, Ontario L7M 1A9, Canada
 905-335-3700

FILM_LOG_FLEX: the detailed FLEX file format can be obtained from
 Time Logic, Inc.
 11992 Challenger Ct.
 Moorpark, CA 93021
 805-529-1155

Microsoft Defined Tombstone Data

A suggestion to replace the entire RIFF INFO section by a tagged text field bibliographic information section requires standardized text fields to be defined.

The floor is open for proposals.

The following tombstone data has already been defined by Microsoft:

Additional AVI information following the 'strl' chunk in the AVI header LIST 'hdrl' chunk. (from AVI RIFF form, Video for Windows SDK):

Additional Header Data

After the 'strl' chunk, some AVI files might have additional header data. Additional information can include timecode chunks that apply to the whole file. The following timecode four-character codes are useful in AVI files:²

Four-character code	Description
ISMP	Indicates the chunk contains the SMPTE timecode of the digitization starting point. The time is expressed as a zero-terminated text string of the form HH:MM:SS.FF. If performing MCI capture in AVICAP, this chunk is automatically set from the MCI start time.
IDIT	Indicates the chunk specifies the time and date digitizing commenced. This time is contained in an ASCII string of exactly 26 characters and has the format "Wed Jan 02 02:03:55 1990\n\0". The <code>ctime</code> and <code>asctime</code> functions can be used to create strings in this format.

The 'ISMP' timecode chunk usually corresponds to the starting timecode copied from the first sample of a captured sequence. Use this chunk to index the captured sample to the original sample. For example, if you need to recapture a video sequence, you can use the timecode to reposition the original sequence and restart capture from that point.

The 'IDIT' timecode chunk usually corresponds to the time the sample was captured. This timecode provides a reference to the age of a sample and creates a history if a series of samples is captured.

The weakness with the above section is that the ISMP information does not include any reel ID, so the effective timecode is meaningless.

Additional information, global to RIFF files (from *Multimedia File Formats, Windows 3.1 SDK*):

The INFO List Chunk

The "INFO" list is a registered global form type that can store information that helps identify the contents of the chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments. An "INFO" list is a "LIST" chunk with list type "INFO." The following example shows a sample "INFO" list chunk:

```
LIST('INFO' INAM('Two Trees'Z)
      ICMT('A picture for the opening screen'Z) )
```

An "INFO" list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an "INFO" list. Each chunk contains a ZSTR, or null-terminated text string.

Chunk ID	Description
IARL	<i>Archival Location.</i> Indicates where the subject of the file is archived.
IART	<i>Artist.</i> Lists the artist of the original subject of the file; for example, "Michaelangelo."
ICMS	<i>Commissioned.</i> Lists the name of the person or organization that commissioned the subject of the file; for example, "Pope Julian II."

² Note that these chunks are lxxx chunks, and as such, should be found in an INFO chunk (see comment below)?

ICMT	<i>Comments.</i> Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do not include newline characters.
ICOP	<i>Copyright.</i> Records the copyright information for the file; for example, "Copyright Encyclopedia International 1991." If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	<i>Creation date.</i> Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left; for example, "1553-05-03" for May 3, 1553.
ICRP	<i>Cropped.</i> Describes whether an image has been cropped and, if so, how it was cropped; for example, "lower-right corner."
IDIM	<i>Dimensions.</i> Specifies the size of the original subject of the file; for example, "8.5 in h, 11 in w."
IDPI	<i>Dots Per Inch.</i> Stores dots per inch setting of the digitizer used to produce the file, such as "300."
IENG	<i>Engineer.</i> Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank; for example, "Smith, John; Adams, Joe."
IGNR	<i>Genre.</i> Describes the original work, such as "landscape," "portrait," "still life," etc.
IKEY	<i>Keywords.</i> Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank; for example, "Seattle; aerial view; scenery."
ILGT	<i>Lightness.</i> Describes the changes in lightness settings on the digitizer required to produce the file. Note that the format of this information depends on hardware used.
IMED	<i>Medium.</i> Describes the original subject of the file, such as "computer image," "drawing," "lithograph," and so on.
INAM	<i>Name.</i> Stores the title of the subject of the file, such as "Seattle From Above."
IPLT	<i>Palette Setting.</i> Specifies the number of colors requested when digitizing an image, such as "256."
IPRD	<i>Product.</i> Specifies the name of the title the file was originally intended for, such as "Encyclopedia of Pacific Northwest Geography."
ISBJ	<i>Subject.</i> Describes the contents of the file, such as "Aerial view of Seattle."
ISFT	<i>Software.</i> Identifies the name of the software package used to create the file, such as "Microsoft WaveEdit."
ISHP	<i>Sharpness.</i> Identifies the changes in sharpness for the digitizer required to produce the file (the format depends on the hardware used).
ISRC	<i>Source.</i> Identifies the name of the person or organization who supplied the original subject of the file; for example, "Trey Research."
ISRF	<i>Source Form.</i> Identifies the original form of the material that was digitized, such as "slide," "paper," "map," and so on. This is not necessarily the same as IMED.
ITCH	<i>Technician.</i> Identifies the technician who digitized the subject file; for example, "Smith, John."

Other chunks can be added to the INFO section (or to a new section?)

THIS PAGE BLANK (USPTO)

6.0 Other File Issues

Issues

Continuation Over Multiple Files

With restrictions to the maximum file format and limitations on the maximum size of disk partitions, it may be desirable to implement a mechanism that specifies a linkage across multiple files. It will certainly be desired that the data of one stream of video can be captured across multiple files. For the moment, this will be external to the AVI file.

Minimized Time to Open Files

tbd

Simultaneous Reading and Writing / Multiple Access

tbd

7.0 MJPG Motion JPEG DIB Extensions

Issues

ISO vs. Non ISO Formats

For interoperability between various hardware and software codecs, it is not sufficient that the file format be standardized; the contents of the video data chunks must be standardized. In order to comply with industry standards, the AVI MJPG specification and OMF 2.0 file format specifications, the standard ISO Motion JPEG specification is proposed, with certain additions to support efficient identification of the interleaved fields in the frame.

Non ISO formats could be used provided that they provide the same outward appearance as the ISO data chunk. This would require different codecs to decompress and compress the data.

8-bit vs. 10-bit YUV

This document currently addresses 8-bit 4:2:2 YUV as per the Microsoft specification.

Video vs. Bitmap

Issues between the fact that a bitmap is a pure rectangle, while video starts with half lines; Motion JPEG compresses on multiple of 8 lines, so what is the frame of video that is compressed vs. the real frame.

See the section of the Video Properties Header in Section 5.0.

Proposed Data Chunk Format

The data chunk format is defined as per the Microsoft *JPEG DIB FORMAT* Technical Note, Type 2: Motion JPEG DIB.

From the DIB spec:

Type 2: Motion JPEG

Motion JPEG DIBs shall accommodate interchange formats which satisfy the "General sequential and progressive syntax" (ISO 10918 Part 1, Annex B, Para. B.2). A set of images of this type with compatible parameters can be placed in an AVI file to describe a motion sequence. Frame headers for these DIBs shall be limited to those specified in Para B.2.2 of the cited Annex B. These types are SOF0, SOF1, SOF2, SOF3, SOF9, SOF10 and SOF11. Of the types accommodated, this specification provides implementation only for the Baseline Sequential DCT (SOF0).

...

This DIB type contains incomplete JPEG data (Abbreviated Format per ISO 10918) and is not intended for stand-alone single image frame disk files. It may be used within RIFF files and other contexts where it is appropriate to:

- a. Decode an image without supplying the associated JPEG Huffman tables. This presumes the codec has been properly pre-initialized prior to image decode.
- b. Request encoder output of compressed image data absent embedded Huffman Tables.

All motion JPEG data will use YCrCb encoding.

In an AVI sequence all JPEG frames will be key frames as this ensures that within the AVI and Video for Windows architecture all frames will be directly and independently addressable.

For optimal size and speed during playback of an AVI file the Huffman data used by motion JPEG will be fixed and defined by this document. This will make the individual frames of every motion sequence smaller and more efficient to play-back. Also as all sequences of motion images use the same Huffman data and colour space it is much more likely that motion data can be directly exchanged without re-compression. A definition of the Huffman data will be provided in MMREG.H (which is listed at the end of this document) as a byte string which can be concatenated onto the start of a motion JPEG image to form a valid still JPEG image-

```
MJPGDHTSeg = { X'FF', DHT, length, JPEG Huffman table parameters }
```

Q-table data is present and may vary in every frame of a motion sequence to permit control over the bandwidth of sequences which contain bursts of frames of varying levels of complexity. The restart interval used during the compression process may also vary for every frame.

Only the interleaved form of YCrCb images is supported for motion JPEG data. This implies that only one SOS segment will be present in any particular motion JPEG image.

The following applies (again from MS DIB spec):

As in the JPEG DIB format the JPEG stream syntax is used for the image data with the following constraints. The following JPEG marker codes SOI, DRI, DQT, SOF0, SOS and EOI are expected (mandatory) in the image data chunk, and the constrained values shown in the example below are mandatory for the image data within the AVI stream.

Any parameters in the SOF0 (frame) and SOS (start of scan) headers that are duplicated in the BITMAPINFOHEADER for JPEG must be the same. This would include Sample Precision, subsampling, number of components (as implied by JPEGColorSpaceID), etc. The number of lines and samples per lines in the SOF0 segment and the width and height defined in the format chunk must match the main AVI header width and height values. All of these values are expected to remain the same for every image data chunk in the AVI sequence.

Within the image data chunk two JPEG segments beginning with the SOI marker and ending with the EOI marker are allowed to accommodate field interleaved streams. There is an APP0 marker immediately following the SOI marker that contains information about the video image. Specifically, this allows the identification of the two fields which comprise an interleaved frame. This APP0 marker is expected to have the first 4 bytes following the length bytes set to the characters 'A', 'V', 'I', '1'. The next byte indicates which field the JPEG data was compressed from and has an expected value of 0x01 for the FIRST JPEG data segment and 0x02 for the SECOND segment, indicating the FIRST and SECOND fields respectively. If the stream is not field interleaved then this value will be 0x00 and there will only be one JPEG segment. The remaining seven bytes are expected to be set to 0 and will be ignored by the codec.

If a codec cannot handle the interleaved fields, the codec will use only the FIRST field and will replicate the lines as necessary to provide an image that conforms to the image size defined in the main AVI header. Conversely if a capture system only accesses a single field of each source frame only a

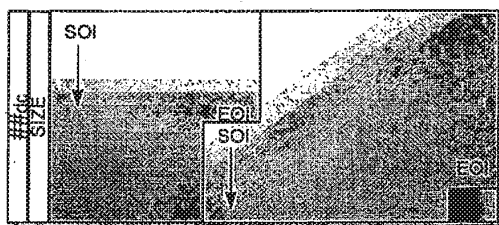
single field image may be present in a JPEG stream. This implies that the single field data should be used as the source of both fields by a decompressor that wishes to process full interlaced data.

And a new proposal:

The APP0 marker will have an added use beside the field polarity and will also specify the size of the current field (entirely, from SOI to EOI), as well as any padding required. This is to allow applications and codecs to avoid being forced to scan for the second SOI-EOI pair. The size information in the APP0 marker does not include any information about a second field in the ##dc chunk. This way, fields can be copied as integral units without having to have the contained data modified for the presence and size of neighbouring fields.

The byte following the polarity will be reserved for future expansion. The remaining 8 bytes of this marker will be used to store two sizes values. The first value specifies the length in bytes of the field (entirely, from SOI to EOI) including any padding after the EOI (which should be small) and the second value specifies the size of the same field excluding any padding after the EOI. Both size values will be stored in the usual ISO JPEG fashion where bytes are stored in decreasing order with the most significant byte being first. The length of this marker was previously 14 and is now set to 16.

MJPEG DIB RIFF chunk (##dc)



An example MJPEG frame with two interleaved fields

-  First field (SOI EOI pair)
-  Second field (SOI EOI pair)
-  Padding

MJPEG Baseline DCT - YCbCr DIB Map

A Baseline DCT - YCbCr will now therefore have the following look :

```

SOI  (xFFD8)
APP0 (xFFE0),
    Length (16),
    "AVI1", Polarity, 0, FieldSize, FieldSizeLessPadding.
DRI  (xFFDD),
    Length (4),
    Restart interval.
```

```

DQT  (x FFDE),
    Length (132),
    Precision (0), Table ID (0),
    DQT data [64] for table 0,
    Precision (0), Table ID (1),
    DQT data [64] for table 1.
SOF0 (x FFC0),
    Length (17),
    Sample Precision (8),
    Number of lines (biHeight : multiple of 8),
    Sample per line (biWidth : multiple of 16),
    Number of components (3),
    ID of 1st component (1),
    Sampling ratio of 1st component (H,V),
    Q-table ID of 1st component (0),
    ID of 2nd component (2),
    Sampling ratio of 2nd component (H,V),
    Q table ID of 2nd component (1),
    ID of 3rd component (3),
    Sampling ratio of 3rd component (H,V),
    Q table ID of 3rd component (1).
SOS  (x FFDA),
    Length (12),
    Number of components (3),
    ID of 1st component (1),
    DC and AC Huffman ID for 1st component (0,0),
    ID of 2nd component (2),
    DC and AC Huffman ID for 2nd component (1,1),
    ID of 3rd component (3),
    DC and AC Huffman ID for 3rd component (1,1),
    Start of spectral (0),
    End of spectral (63),
    Successive approximation bit position, high and low (0,0).

```

***** IMAGE DATA *****

There could be some Restart markers.
 Values are from xFFD0 to xFFD7.

EOI (x FFD9).

Be sure to have a single DQT as the previous ones will be overridden.
 The order of the markers may affect some codecs.
 IDs for the 3 component as recommended by the Microsoft DIB specification are 1,2 and 3.

Sector Alignment and Padding

The ##dc chunks may be sector aligned if necessary. As such, a padding chunk would be inserted following the ##dc to preserve the RIFF format. Padding between fields should be discouraged; however the ISO specification allows padding if the data contains 0xFF for each padded byte.

8.0 OpenDML File Format Certification Procedures

Concurrently Develop a Test Suite to measure compatibility

Test older Codecs with new AVI files.

Test old AVI files with new Codecs.

Circulate all files through industry (i.e., everybody tests everybody else's files).

File checker routine to verify AVI format. Program may grow to include test decompressions using software Codec.

Test using MCI/AVI with standard codecs.

Go to MS porting labs for specific video file format / Codec test week.



Search

[Browse by Topic](#)[Join Us](#)[Sign in](#)[Home](#) [News](#) [Hardware Reviews](#) [Software Reviews](#) [How-To](#) [Blogs](#) [Videos](#) [Downloads](#) [Shc](#)

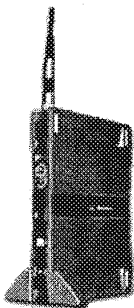
Future Gear: PC on the HiFi, and the TV

A profusion of little boxes help beam movies, music, and photos from your PC into the living room.

Sean Captain

Tuesday, January 21, 2003 5:00 PM PST

Today's high-power PCs can serve as complete entertainment centers—playing CDs, DVDs, and compressed digital music; doubling as TVs and video recorders; and blasting out high-end video game graphics. But they also run spreadsheets, e-mail clients, and database programs. While this versatility is what makes PCs so wonderful, people shopping for entertainment devices don't want to mix business with pleasure. They want to banish the work to the home office while they watch movies in the family room; and they don't want calendar reminders popping up on screen while they are watching movies.



Enter wireless networks, which promise to bring all your digital entertainment into the TV room while keeping the PC out. Just as the furnace and hot water heater pump their content through the house via ducts and pipes, the PC can, in theory, pump its goodies through the air (an idea I considered in [last August's column](#)). This notion appeared to be a newfound religion—with plenty of converts—at this month's Consumer Electronics Show in Las Vegas. By my count, eight companies introduced some type of digital media receiver that grabs content from a PC and channels it to a TV or stereo system; and more companies are likely to join them. Prices range from about \$100 to \$300. Most products should go on sale in the next six months, at which time we'll see how many people have the patience to make them work.

The Air Isn't Clear

Almost all of the new receivers include Ethernet ports for connection to wired networks. But no one cares about that, of course. If we did, many more of us would have already dragged cables through the house or bought home phone line or power line network adapters. The fact is, wires are not cool—but wireless networking has its problems.

One problem is bandwidth: The currently dominant PC wireless standard, 802.11b, with a maximum throughput of 11 megabits per second (and often 5 mbps or less), is not reliable enough to transmit high-quality video. Although some companies, such as Prismig, market their 802.11b receivers for lower-bit-rate video like video CDs, the first wireless devices mostly grab audio from PCs, and some also display photos.

Displaying photos is a prominent feature on the [Digital Media Receiver 5000](#) from Hewlett-Packard, which also makes digital cameras and photo printers. It displays .jpg, .gif, .bmp, and .png files, and allows you to rotate them, arrange them in a slide show, or send them to a printer. HP's black-and-silver set-top box comes with an infrared remote control and displays menu options on your TV for accessing photos and music.

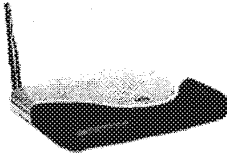


But video, not music or photos, appears to be HP's real target. "These are just baby steps," Ameer Karim of HP's home products division told me when he demonstrated the receiver. He's already planning for the next-generation product,

Related Content

which will probably use the emerging 20+ mbps 802.11g wireless standard to stream video, possibly even of DVD quality. (Companies such as Actiontec and D-Link are now offering 802.11g wireless cards and access points, and Apple has integrated the technology on its new laptops and offers a new 802.11g base station called AirPort Extreme.)

But HP may be dreaming of video to the detriment of audio. While it handles MP3 and .wma music files, the 5000 doesn't support the high-quality .wav format found on music CDs. And it provides basic left and right analog outputs only, not the digital outputs that feed high-end stereo systems.



In contrast, a new company called Cd3o is focusing solely on music with its Network MP3 Players. Despite the name, the little silver boxes also support .wma and .wav formats, and the \$249 model includes optical and coaxial SP/DIF outputs. The Cd3o receiver has an infrared remote control, but it doesn't hook up to a TV to display its menus. Instead, it talks you through them with a synthesized voice that, on my prerelease version, sounded like Max Headroom sucking Helium. The company says it will sound better in the shipping version.

Tangled Up Without Wires

For the most part, the receivers worked smoothly in the controlled environments in which I saw them demonstrated. Each vendor brought its own preconfigured wireless gateway and notebook PC, with the receiver already installed via the Universal Plug and Play feature in Windows XP. Transmission conditions were ideal, with the gateway, PC, and TV or stereo in the same room and no microwave ovens or 2.4-GHz cordless phones to interfere.

Wireless networking gets messier, however, when you try to set it up yourself. We already have an 802.11b access point at *PC World* headquarters, but when I tried to set up one of the media receivers I still had Microsoft software to contend with. Windows XP provides an illogically complex series of dialog boxes for configuring a wireless connection on a PC, and there are plenty of places to get lost--especially if you want to get on a network with Wired Equivalency Protocol security enabled. Without too much trouble, I found the box for entering my WEP key, but I had to dig around for a long time to specify that I wanted 128-bit encryption using a hexadecimal key. And I can't remember where I went to specify the wireless channel--a requirement for some access points.

I'm not an expert in wireless networking, but neither are many of the consumers that companies are targeting with these media receivers. At least I had the benefit of asking for help from some of our tech-savvy editors and IT staff, and even they had trouble navigating the Windows setup.

Talk of Windows reminds me of another problem with this new home entertainment vision: It requires PCs that function smoothly. Because PCs are so complex, they inevitably have more problems than devices like TVs or stereos. What happens when we wed the two types of products? Will people tolerate a TV that crashes, or a stereo that won't boot?

HP's Karim was right when he talked about baby steps. This first generation of media receivers is an interesting experiment in a system that may some day redefine our entertainment experience. But at first these devices may appeal only to hobbyists who have a PC-hardened sense of patience.

Sean Captain is a senior associate editor for PCWorld.com.

SonicWall is wrapping a reduced-price package of security services around some of its branch-office firewall/VPN hardware.
03-May-2007

[More news](#)

Qtv

- [Home](#)
- [Language](#)
 - [Deutsch](#)
 - [English](#)
-
- [Product Tour](#)
- [Content](#)
- [Business Development](#)
- [About](#)
 - [Overview](#)
 - [News](#)
 - [Team](#)
 - [Contact](#)
- [Sign Up Now for a Free QTV Trial](#)
-

Overview

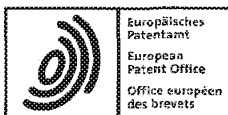
BroadQ is revolutionizing television entertainment with innovative software and service solutions that enable the convergence of digital entertainment within the digital living room. Our Qtv Internet TV media player fundamentally changes television entertainment by expanding the consumer's entertainment choices to include both personal and Internet content/services from a potentially unlimited number of providers.

BroadQ was founded in 2002 in Austin, Texas, by Stacy Cook and Rob Lipman. We launched our first product in October, 2002 - QCast for the PlayStation 2, which allowed users to access their PC-based photos, music and videos from the TV. In 2003, we released the second version of the QCast software through Mad Catz, Inc. under the product name GameShark Media Player. In 2004, we launched a multi-language PAL version of the QCast software in Europe through trendExpress under the product name Medio.

BroadQ is uniquely positioned to take advantage of the emerging Internet TV market. Our team is highly experienced with expertise in IPTV, PS2 software development, intellectual property, advertising and content licensing.

Copyright © 2009 BroadQ, LLC

- [Home](#)
- [Business Development](#)
- [Privacy and Legal](#)



**SUPPLEMENTARY
PARTIAL EUROPEAN SEARCH REPORT**

Application Number

under Rule 62a and/or 63 of the European Patent Convention.
This report shall be considered, for the purposes of
subsequent proceedings, as the European search report

EP 04 81 3918

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
X,P	WO 2004/097811 A1 (LG ELECTRONICS INC [KR]; KIM BYUNG JIN [KR]; KIM HYUNG SUN [KR]; YANG) 11 November 2004 (2004-11-11) * the whole document *	54-95	INV. G06F15/00
X	WO 02/01880 A1 (THOMSON LICENSING SA [FR]; STROTHMAN JAMES ALAN [US]; BARRON STEVEN AN) 3 January 2002 (2002-01-03) * the whole document *	54-95	
X	US 6 046 778 A (NONOMURA TOMOYUKI [JP] ET AL) 4 April 2000 (2000-04-04) * abstract *	54-95	
X	EP 0 757 484 A2 (SONY CORP [JP]) 5 February 1997 (1997-02-05) * abstract *	54-95	
X	WO 01/50732 A2 (VM LABS INC [US]) 12 July 2001 (2001-07-12) * abstract *	54-95	
			TECHNICAL FIELDS SEARCHED (IPC)
			G11B H04N G06F
The supplementary search report has been based on the last set of claims valid and available at the start of the search.			
INCOMPLETE SEARCH			
The Search Division considers that the present application, or one or more of its claims, does/do not comply with the EPC so that only a partial search (R.62a, 63) has been carried out.			
Claims searched completely :			
Claims searched incompletely :			
Claims not searched :			
Reason for the limitation of the search: see sheet C			
Place of search The Hague		Date of completion of the search 19 December 2012	Examiner Giannotti, Pantaleo
CATEGORY OF CITED DOCUMENTS			
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document			

3
EPO FORM 1503 03.82 (P04E20)



INCOMPLETE SEARCH
SHEET C

Application Number
EP 04 81 3918

Claim(s) completely searchable:
54-95

Claim(s) not searched:
1-53, 96-169

Reason for the limitation of the search:

The present application contains 169 claims, of which 20 are independent. There are so many claims and they are drafted in such a way that the claims as a whole do not comply with the provisions of clarity and conciseness in Article 84 EPC, as it is unduly burdensome for a skilled person to establish the subject-matter for which protection is sought. In particular, contrary to Article 84 EPC, the claims appear to be directed to a format whose features are not clearly defined. Non-compliance with the substantive provisions is such that a meaningful search of the whole claimed subject-matter can not be carried out. Under Rule 63 (1) EPC, the Applicant was therefore invited to file a statement indicating the subject-matter to be searched, with explicit reference to the specific claims on file. The Applicant indicated Claims 54-95. The search has therefore been directed to the subject matter of said claims.

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 04 81 3918

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

19-12-2012

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2004097811 A1	11-11-2004	BR P10409807 A	09-05-2006
		CN 1781143 A	31-05-2006
		EP 1623413 A1	08-02-2006
		JP 4790599 B2	12-10-2011
		JP 2006525742 A	09-11-2006
		KR 20060006952 A	20-01-2006
		TW 1289298 B	01-11-2007
		US 2004218907 A1	04-11-2004
		WO 2004097811 A1	11-11-2004
WO 0201880 A1	03-01-2002	AU 6467401 A	08-01-2002
		US 8091106 B1	03-01-2012
		WO 0201880 A1	03-01-2002
US 6046778 A	04-04-2000	NONE	
EP 0757484 A2	05-02-1997	AT 232345 T	15-02-2003
		AU 702694 B2	04-03-1999
		BR 9603263 A	28-04-1998
		CA 2181864 A1	03-02-1997
		CN 1153977 A	09-07-1997
		CN 1444222 A	24-09-2003
		DE 69626072 D1	13-03-2003
		DE 69626072 T2	27-11-2003
		EP 0757484 A2	05-02-1997
		ES 2191740 T3	16-09-2003
		JP 3326670 B2	24-09-2002
		JP 9051502 A	18-02-1997
		MY 116664 A	31-03-2004
		US 5848217 A	08-12-1998
		US 6424792 B1	23-07-2002
WO 0150732 A2	12-07-2001	AU 2543001 A	16-07-2001
		US 6871008 B1	22-03-2005
		WO 0150732 A2	12-07-2001

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82



SUPPLEMENTARY EUROPEAN SEARCH REPORT

Application Number
EP 10 72 9513

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
X	US 2005/108320 A1 (LORD ROBERT D [US] ET AL) 19 May 2005 (2005-05-19) * abstract * * figures 1,7 * * paragraphs [0015] - [0020] * * paragraphs [0032] - [0034] *	1-15	INV. G06F15/16 H04N21/231 H04N21/278
A	WO 02/08948 A2 (VIVCOM INC [US]; SULL SANGHOON [KR]; KIM HYEOKMAN [KR]; CHOI HYUNGSEOK) 31 January 2002 (2002-01-31) * abstract * * figures 2,3 * * page 30, line 2 - page 35, line 5 *	1-15	
			TECHNICAL FIELDS SEARCHED (IPC)
			H04N G06F
The supplementary search report has been based on the last set of claims valid and available at the start of the search.			
Place of search The Hague		Date of completion of the search 9 December 2013	Examiner Naci, Suphi Umut
CATEGORY OF CITED DOCUMENTS X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document		T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document	

1
EPO FORM 1503 03-02 (P04C04)

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 10 72 9513

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

09-12-2013

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005108320 A1	19-05-2005	US 2005108320 A1	19-05-2005
		US 2007192445 A1	16-08-2007
		US 2010185729 A1	22-07-2010
		WO 2005050373 A2	02-06-2005
WO 0208948 A2	31-01-2002	AU 8300401 A	05-02-2002
		KR 20050002681 A	10-01-2005
		KR 20070103728 A	24-10-2007
		US 2002069218 A1	06-06-2002
		US 2007033170 A1	08-02-2007
		US 2007033292 A1	08-02-2007
		US 2007033515 A1	08-02-2007
		US 2007033521 A1	08-02-2007
		US 2007033533 A1	08-02-2007
		US 2007038612 A1	15-02-2007
		US 2007044010 A1	22-02-2007
		US 2011093492 A1	21-04-2011
		WO 0208948 A2	31-01-2002

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

The examination is being carried out on the **following application documents**

Description, Pages

1-10 as published

Claims, Numbers

1-15 received on 05-03-2012 with letter of 01-03-2012

Drawings, Sheets

1/4-4/4 as published

Reference is made to the following document; the numbering will be adhered to in the rest of the procedure.

D1 US 2005/108320 A1 (LORD ROBERT D [US] ET AL) 19 May 2005
(2005-05-19)

1 The present application does not meet the requirements of Article 52(1) EPC because the subject-matter of claim 1 is not new within the meaning of Article 54(1) and (2) EPC.

1.1 D1 discloses:

*A content guide system (abstract) comprising:
a content server configured to provide access to content (§15 and 16);
a tag server including a searchable database of tags (fig.1, 34 - database and §21); the content guide system characterized by:
a tagging device in network communication with the tag server and the content server, where the tagging device is configured to access content via the content server referencing the accessed content that is provided to the tag server for inclusion into the searchable database of tags (§25 and 26); and*

a playback device in network communication with the tag server and the content server, where the playback device is configured to search the database, retrieve from the database a tag that references content accessible via the content server and access the content referenced by the retrieved tag via the content server (§33).

1.2 The subject-matter of claim 1 is therefore not new (Article 54(1) and (2) EPC).

- 2 Dependent claims 2-15 do not appear to contain any additional features which, in combination with the features of any claim to which they refer, meet the requirements of the EPC with respect to novelty and/or inventive step:

Claims 2-4 and 15 introduce content and location identifiers like URLs, which is also disclosed in D1 (§27).

Claims 5 and 6 specify merely well-known database operations like add, delete, etc...

Claims 7 and 8 further introduce search and recommendation operations, which is also disclosed in D1 (§33).

Claims 9-14 introduce various design options like caching or not the parts of the content and tags locally, which are well-known in the art.

3 FURTHER PROSECUTION OF THE APPLICATION

- 3.1 It is not at present apparent which part of the application could serve as a basis for a new, allowable claim. Should the applicant nevertheless regard some particular matter as patentable, an independent claim should be filed taking account of Rule 43(1) EPC. The applicant should also indicate how the subject-matter of the new claim differs from the state of the art and the significance thereof.
- 3.2 To meet the requirements of Rule 42(1)(b) EPC, D1 should be identified in the description and the relevant background art disclosed therein should be briefly discussed.
- 3.3 In order to facilitate the examination of the conformity of the amended application with the requirements of Article 123(2) EPC, the applicant should clearly identify the amendments made, irrespective of whether they concern amendments by addition, replacement or deletion, and indicate the passages of the application as filed on which these amendments are based (see Guidelines E-II, 1).



**SUPPLEMENTARY
EUROPEAN SEARCH REPORT**

Application Number
EP 11 77 4529

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (IPC)
X	HEINZ H BAUSCHKE ET AL: "Recompression of JPEG Images by Requantization", IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE SERVICE CENTER, PISCATAWAY, NJ, US, vol. 12, no. 7, 1 July 2003 (2003-07-01), XP011074411, ISSN: 1057-7149 * the whole document *	1-15	INV. G06K9/36 G06K9/46 H04N19/124 H04N19/154 H04N19/176 H04N19/192 H04N19/196 H04N19/40 H04N19/60
X	US 6 850 650 B1 (BAUSCHKE HEINZ [CA] ET AL) 1 February 2005 (2005-02-01) * column 6, line 35 - column 8, line 49; figures 4-7 *	1-15	
			TECHNICAL FIELDS SEARCHED (IPC)
			H04N
The supplementary search report has been based on the last set of claims valid and available at the start of the search.			
Place of search The Hague		Date of completion of the search 31 January 2014	Examiner: Georgiou, Georgia
CATEGORY OF CITED DOCUMENTS			
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document		T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document	

EPO FORM 1503 03 02 (P04C04)

EP 11 77 4529

31-01-2014

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6850650 B1	01-02-2005	US 6850650 B1	01-02-2005
		US 2005111747 A1	26-05-2005

SEP FORM PQ459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82