Full-motion Video for Portable Multimedia Terminals

by
Allan Christian Long, Jr.

B.S. (University of Virginia) 1992

A project report submitted in partial satisfaction of the requirements for the degree of
Master of Science
in
Computer Science
in the
GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Readers:

Professor Lawrence Rowe
Professor Robert Brodersen

1996

Full-motion Video for Portable Multimedia Terminals

Allan Christian Long, Jr.

Abstract

Digital video and mobile computing are two increasingly important technologies, as evidenced by the ubiquity of desktop digital video and the rising popularity of portable computing devices. The InfoPad is a mobile terminal that uniquely combines these two technologies.

This report describes the development of software infrastructure and a user interface to video playback on the InfoPad. It is based on the Berkeley Continuous Media Toolkit.

1. Introduction

Digital video and mobile computing are increasingly important technologies. There are many computer peripherals for playing digital video on the desktop. Similarly, there are a growing number of portable computing devices, ranging from personal digital assistants (e.g., Apple Newton) to portable workstation-class computers (e.g., Intel Pentium and Apple PowerBook laptops).

This report describes software infrastructure that enables video playback on a portable, wireless computer terminal called the *InfoPad*, shown in Figure 1. [Barr94] The InfoPad is a low-power and light weight dumb terminal. Consequently, it does not have enough computing power to run applications, but only enough for primitive input and output. The InfoPad displays applications and allows the user to interact with them, but the applications run on hosts connected to a wired network. The InfoPad communicates with the wired network using a high-speed radio link (1 Mbps).

Instead of the conventional keyboard and mouse input devices, the InfoPad has a pen and a microphone. The InfoPad output devices are a large (10 inch diagonal) black and white LCD panel for text and graphics, a smaller (3 inch) detachable full-color display for video, and an earphone for sound.[1] The resolution of the text/graphics display is 640x480 (horizontal x vertical) pixels and the video display resolution is 128x240. The microphone and earphone are part of a combined headset that plugs into the Pad.

To reduce bandwidth requirements for video, the InfoPad has custom hardware to decode *vector quantized* (VQ) video. Although the InfoPad VQ implementation does not compress video as much as some other encoding algorithms, such as MPEG, it was chosen because it is easy to decode in hardware and it is robust in

1. The next generation InfoPad will integrate the video display into a large color LCD panel.

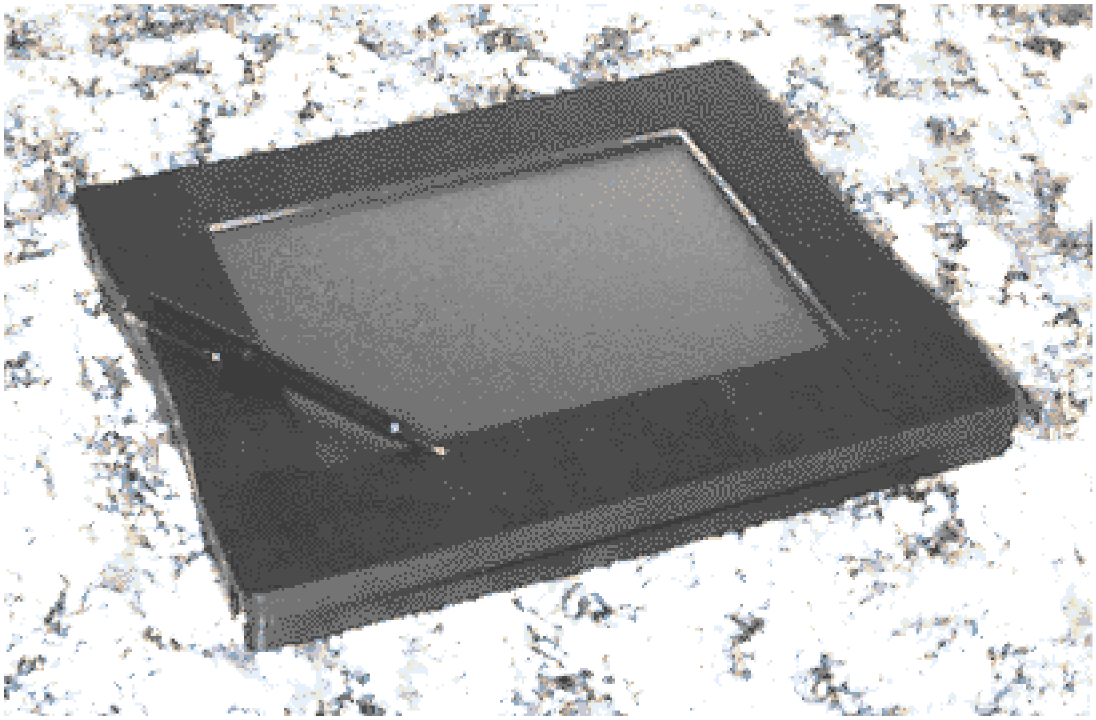

Figure 1 The InfoPad

the presence of bit errors. InfoPad VQ video requires a bandwidth of 675 kbps for a playback rate of 30 frames/second. Other types of data use the following amounts of bandwidth.

Audio input and output require 64 kbps (8 kHz, 8-bit samples) each. The text and graphics bandwidth vary greatly among applications and due to user activity. The most expensive operation is a complete screen refresh, which requires that 300 kbits be transmitted. Currently, the InfoPad uses commercial radios with a maximum bandwidth of about 600 kbps, so it cannot support full-motion, 30 frame/second video. The InfoPad project is developing a custom radio that will have much higher bandwidth (1 Mbit/s) which can simultaneously support audio and 30 frame/second video.

The relevant part of the InfoPad software infrastructure is that dealing with video. Although software for sending video to the InfoPad existed when this project started, it had a primitive interface and limited capabilities. For example, the interface was based on a command line with flags, it could only play video from a single file on a local host at normal speed, and did not synchronize audio with video. Software was needed to remedy these deficiencies. It was decided to take advantage of an existing toolkit designed for writing distributed multimedia applications, the *Berkeley Continuous Media Toolkit* (CMT)[Rowe94], rather than write code from scratch. However, CMT did not support VQ video required by the InfoPad.

This report describes the VQ extensions added to CMT to support the InfoPad. CMT provides objects to read and write files and encode and decode media streams with many different formats (e.g., MJPEG, MPEG, and H.261). Objects were needed to read files stored in VQ format and to receive a VQ stream and send it to the InfoPad. CMT already supplied objects to transmit and receive media streams over a TCP/IP network using a protocol named *Cyclic UDP*, implemented on the user datagram interface to IP [Smith] or the *RTP* real-time protocol implemented on IP-multicast [Schu96]. Figure 2 shows the general CMT playback architecture. CM data is read from disk by the CMT source process. The source

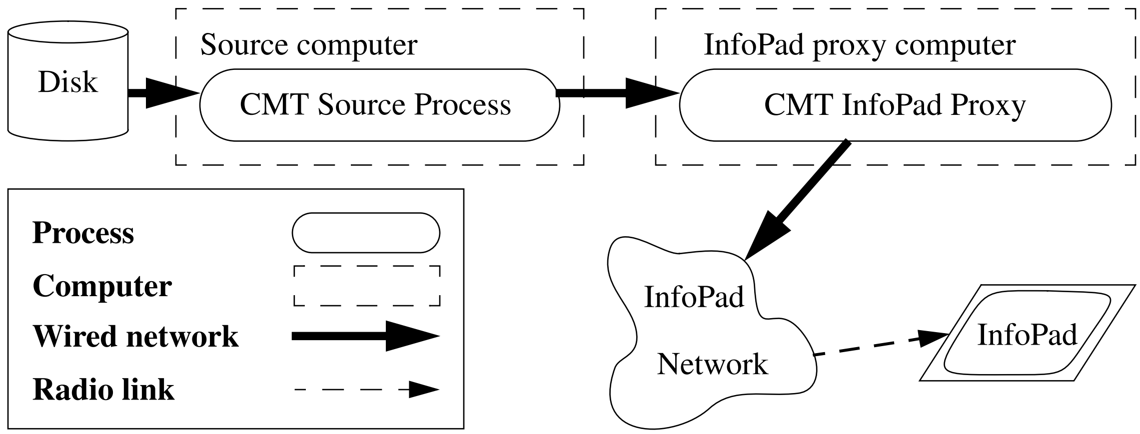

Figure 2 Overview of audio and video playback through CMT on InfoPad.

process sends the CM data to the CMT InfoPad proxy process, which schedules the data for play at the appropriate time. When the time comes to play the data, the proxy forwards the data to the InfoPad network, which delivers the data to the InfoPad via the radio link. The proxy computer is assumed to be a small number of network hops away from the host with a radio interface communicating directly with the InfoPad. The source computer may be an arbitrary number of network hops from the proxy computer.

Since the available video source material was not in VQ format, a transcoder program also had to be written to create VQ files. Transcoding to the format required by CMT applications entailed both changing the format of the data and adding indexing information to the file.

This report is organized as follows. Section 2 describes the overall CMT architecture. Section 3 describes the relevant file formats and CMT objects. Section 4 describes the results of performance experiments on video playback on the InfoPad and experiences using and developing the code. The last section suggests possible future research.

2. CMT Architecture

This section describes the architecture of CMT. CMT defines object-oriented abstractions for audio and video devices (e.g., audio boards, video CODECS, and video capture and overlay boards), time systems, CM sources and sinks (e.g., cameras, microphones, files, displays, and speakers), and network transport protocols (e.g., cyclic-UDP and RTP).

CMT is implemented in *Tcl/Tk*, *Tcl-DP*, and C. Tcl/Tk is a scripting language and graphical user interface toolkit [Oust94]. The interface to CMT objects is similar to the interface to Tk widgets. Tcl-DP provides blocking and non-blocking remote procedure calls, TCP/IP connection management, and a simple distributed object system. Using Tcl-DP, it is easy to write applications composed of multiple processes on different hosts exchanging data and commands.

The *CMPlayer* application, written using CMT, plays synchronized audio and video across a TCP/IP network. It supports full-function VCR commands including pause, fast-forward and rewind, plus additional features like random access. The CMPlayer can play movies composed of several streams with different media types and formats. It uses hardware to decode data if present; otherwise, software decoding is used. Figure 3 shows the CMPlayer control panel.

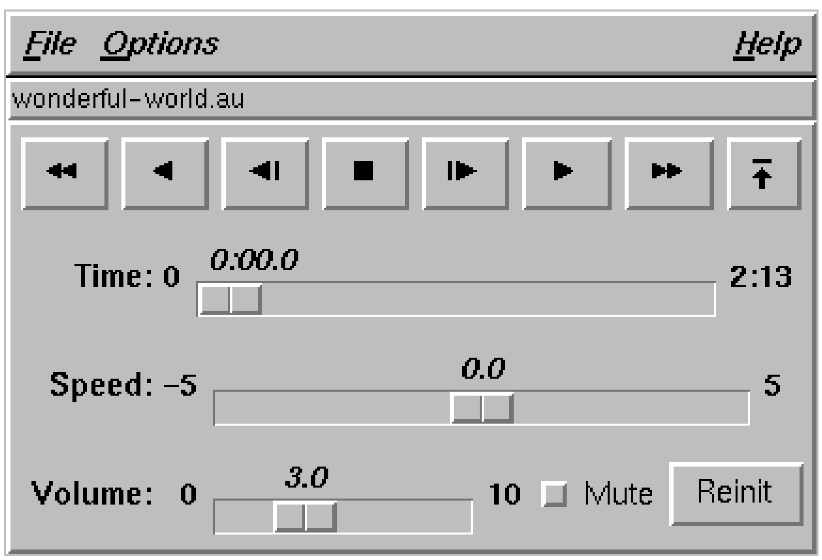

Figure 3 CMPlayer control panel

A CMT application is composed of multiple communicating objects. There are file objects that read CM data from files, packet source and destination objects that send and receive data to/from the network, and play objects that decode streams and display them on output devices. Figure 4 shows the software architecture of a typical, non-InfoPad, playback application. It expands on Figure 2 by showing some of the objects contained in the CMT processes. CM data is read from disk by

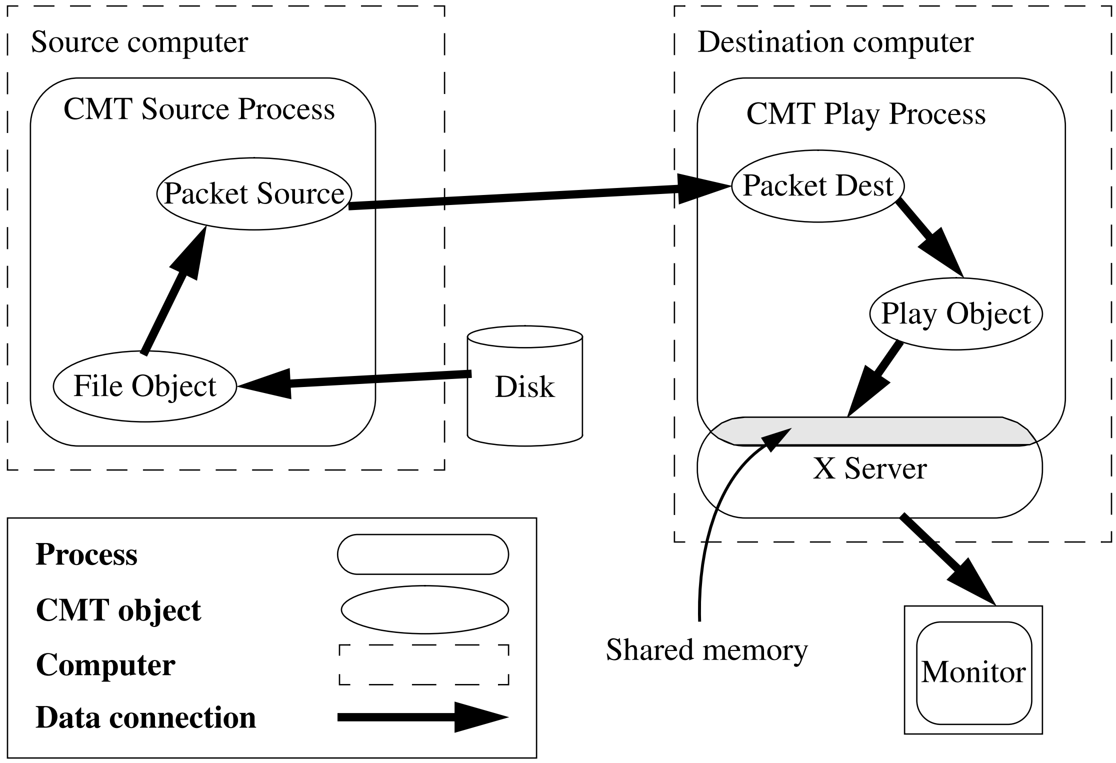

Figure 4 Typical CMT playback.

a file object of the appropriate type. The file object passes the data to the packet source object, which prepares the data for transmission across the network and transmits it. The data is received by the packet destination object, which passes it to a play object of the appropriate type. The play object sends the data to the X server, which displays it on the monitor. The CMT play process shares memory with the X server because this obviates the need for the X server to copy each video frame which significantly improves performance.

The rest of this section describes the CMT abstractions in more detail. The first subsection discusses how CMT manages time. The next subsection describes the CM storage representation. The last subsection discusses the *scatter buffer*, which is an abstraction for media stream data.

2.1. Time

Because CM data is time-sensitive, CMT has a time management abstraction called a *Logical Time System* (LTS) object. LTS has two properties that are to be controlled by an application: *value* and *speed*. Value specifies the logical time, and speed specifies the rate and direction at which logical time moves with respect to real time (positive speeds being forward and negative ones being backwards). If $clock_0$ is the value of the real-time clock at logical time zero, *speed* is the speed of the LTS, and *clock* is the current real-time clock, the current LTS value $value_c$ is given by the formula

$$\text{value}_c = \text{speed} \times (\text{clock} - \text{clock}_0)$$

For example, a speed of one results in normal forward playback while negative five results in fast reverse.

CMT applications typically are composed of multiple processes running on multiple hosts. A CMT application running across multiple hosts has a master LTS object and one or more slave LTS objects. To provide proper synchronization of different CM streams across hosts, LTS objects are synchronized as follows. When the application changes the master LTS object, the master LTS object tells the slaves. To account for clock drift, each slave periodically queries the master and resets itself if it is too far off.

CMT file and play objects use LTS objects to determine when to perform their functions. That is, LTS tells a file object how long it will be in system time until the next frame needs to be sent across the network. The file object queues itself to wake up slightly before then so it can read the frame from the disk and send it on time. Similarly, a play object uses an LTS to determine when the play object should schedule itself to play each frame. If the speed of the LTS is changed (e.g., when play starts or stops) or its value is explicitly changed (e.g., when the user randomly jumps to a new position in the media sequence), the LTS notifies the play and file objects that depend upon it. Those dependent objects then reset their timers to values appropriate for the new LTS speed and value.

2.2. CM Storage Representation

Many standard file formats exist for storing audio and video data. However, it is difficult to perform certain operations on these files, such as jump quickly from one part of a video sequence to another. To simplify these operations and to enable reuse of video sequences without copying them, CMT defines a file format, called a *clip file*, that includes CM data and indexes into that data.

A clip file stores a single clip, which is a contiguous sequence of source material stored in media-specific units (e.g., video frames or audio blocks). A clip file contains media in a single format (e.g., MPEG or MJPEG). It is organized internally as a sequence of chunks, each of which is typed. A chunk may contain a clip file header, media data, or indexes into the media data. An example clip file is shown in Figure 5. All clip files have a generic header chunk that has information about the file as a whole: media type (e.g., video or audio), format (e.g., MPEG, VQ), version number, device (e.g., Parallax, InfoPad), number of media units, and temporal length. In addition, most types also have a media-specific header that contains information specific to that type. Video clip files contain index chunks to support random access.

One might want to play a single video or audio clip. However, it is more likely that a user wants to play a synchronized sequence of audio and video clips. To allow this operation, CMT defines a *movie* file type, identified by the file extension `cmt`.

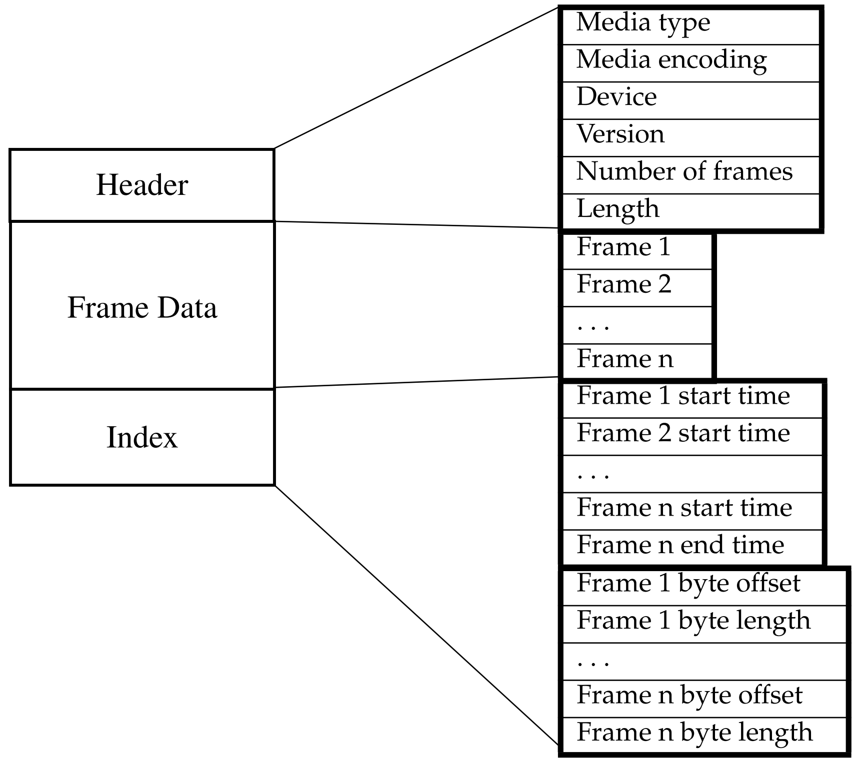

Figure 5 Example generic clipfile layout.

A movie is composed of one or more type-specific *streams*. A stream is a sequence of format-specific clips. A movie is represented by a *script*, which is stored in a separate file that contains references to media sequences (i.e. subsequences of clip files). The logical structure of a movie is shown in Figure 6.

A script file is composed of a header that lists the number of streams and the media type of each stream followed by descriptions of each stream. A stream may be composed of any number of clips that do not overlap in time. For each clip, the script file may specify the start and end times of the clip in the source material (i.e., clip file) and the logical start and end times where the clip should be positioned in the movie script. The source start and end times control which parts of the clip file are used when playing the script, so a subsequence of a clip file may be included in a script. The logical start and end times specify when in the LTS the clip should play (see §2.1. Time). By setting the logical start and end times appropriately, the

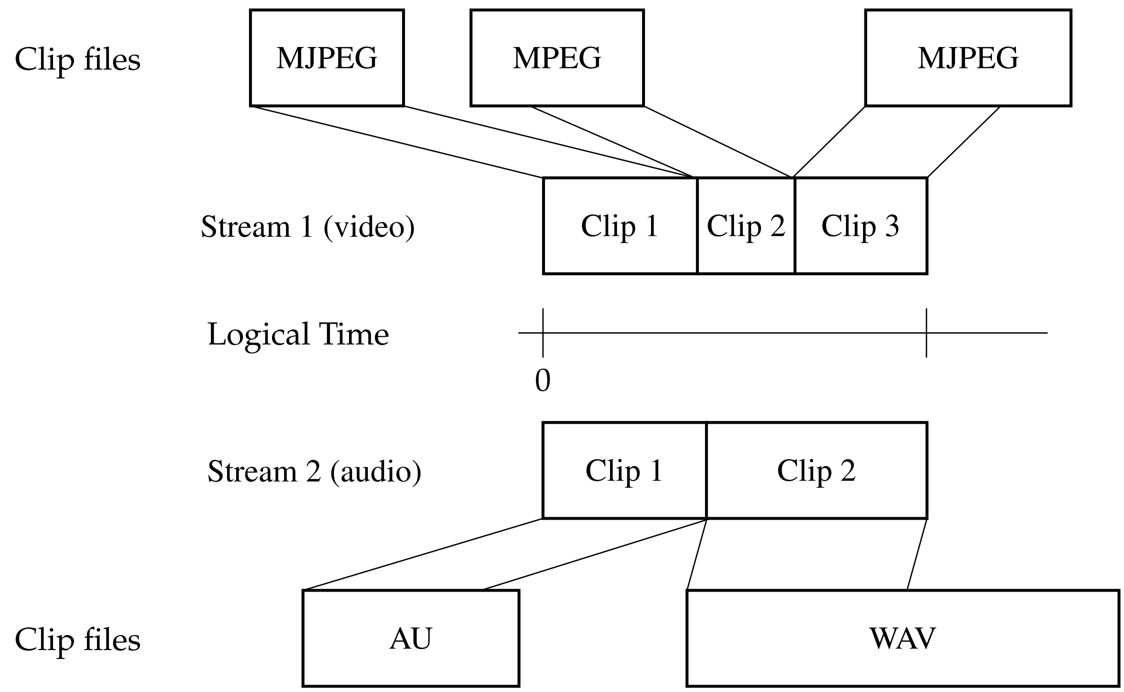

Figure 6 Logical structure of an example script file.

play time for a clip may be squeezed or stretched. Below is an example script file, with annotations (in italics).

Version number
```
# CMT-Movie-1.0
```

Specify tracks and title (tracks numbered from 0 based on order here)
```
movie video audio -tags {{title {Creatures}}}
```

Specify path of a clip for the first (video) track
Format is "`clip` *trackNumber clipFileName"*
```
clip 0 /vid6/creatures.mpg
```
Specify paths of two clips for the second (audio) track. The `-le` *and* `-ls` *options are for logical end logical start. In this case, they cause the audio to play backwards.*
```
clip 1 /vid6/wonderful-world.au
clip 1 /vid6/wonderful-world.au -le 0 -ls 38
```

The clip files can be referenced as above (with a standard Unix absolute path) if they are local. Remote files can also be referenced using a Universal Resource Locator (URL) syntax.

2.3. CMT Scatter Buffer

A CM application manipulates blocks of CM data that may require data copying. For example, an application might copy CM data blocks to concatenate them. Or, headers and trailers might be prepended and appended to a data block before network transmission. To make copying more efficient, CMT contains two abstractions: buffers and scatter buffers. A buffer is a contiguous region of memory. A scatter buffer is an array of pointers to buffers. Since a scatter buffer is an array of pointers instead of actual data, adding and removing buffers to/from scatter buffers takes little time. Data is normally passed between CMT objects within a process by passing the address of a scatter buffer.

3. CMT Objects

This section describes the new VQ objects and how they fit into the CMT architecture. For details on CMT object creation and usage, consult the CMT reference book [Pate96].

Figure 7 expands the CMT source and CMT InfoPad proxy processes shown in Figure 2 to show the CMT objects that read, format, transmit, and play VQ data.

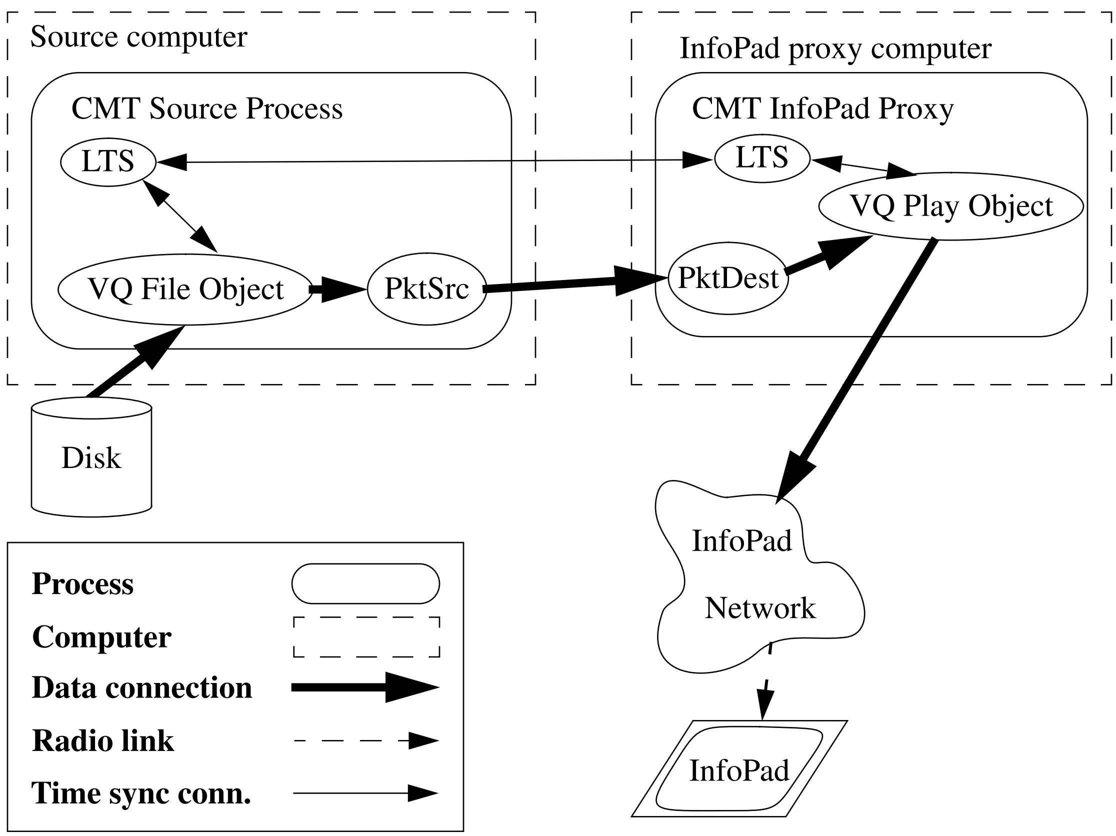

Figure 7 Software modules and dataflow for audio and video playback through CMT on InfoPad.

Figure 7 is similar to the generic playback system shown in Figure 4 except that it shows the LTS objects and it shows InfoPad specific playback. The playback operation begins when VQ data is read from the disk by the VQ file object and sent to the packet source object (PktSrc) for transmission across the network. The packet destination object (PktDest) receives the data and passes it to the VQ play object. The play object formats the data for display and sends it to the InfoPad, via the InfoPad network. In the figure, video data paths are shown by dark arrows. Synchronization messages between LTSs are represented by the light arrows.

The remainder of this section describes the CMT objects in more detail. The first subsection describes vector quantization. The subsections that follow describe the format of VQ clip files, the VQ-to-clipfile transcoder, the VQ file object, and the VQ play object.

3.1. Vector Quantization

The idea behind VQ is very similar to the idea behind color mapping on traditional 8- and 16-bit color bitmap displays. That is, table lookup is used to achieve greater flexibility in what can be displayed than could be had with a simple static mapping of pixel values to color. They are different in that whereas an index for a colormap is used to look up the color of a single pixel, a single element in a VQ frame is used to look up the color of multiple pixels. VQ compression arises from the same small-to-large colorspace mapping used in traditional colormaps and the spatial one-to-many mapping. Since the dimensions of both mappings are static, VQ gives fixed-rate compression. VQ lookup tables (i.e. the analog of colormaps) are called *codebooks*.[Chan]

Here is an example of how a simple codebook scheme works. Suppose that VQ frames, codebooks, and screen images are one-dimensional and that each codebook entry has three pixel values. Then, each frame value corresponds to three pixel values. Figure 8 shows how the frame data corresponds to pixel values for a very small image. Logically, each frame data value expands into three pixel values.

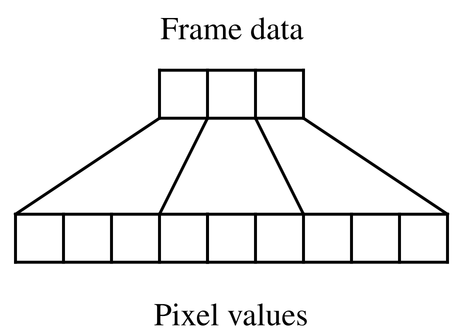

Figure 8 Frame data to pixel value mapping

The pixel values come from the codebook, which is indexed by the frame data. Each codebook entry contains values for three pixels. Figure 9 elaborates on Figure 8 by showing the codebook.

We can also describe the relationships among the frame data, codebook, and pixel values with equations. Continuing the example above, let F be the array of frame data, C be the array of codebook entries, and P be the array of pixel values (all with base index zero). C is indexed first by entry number and then by pixel value within

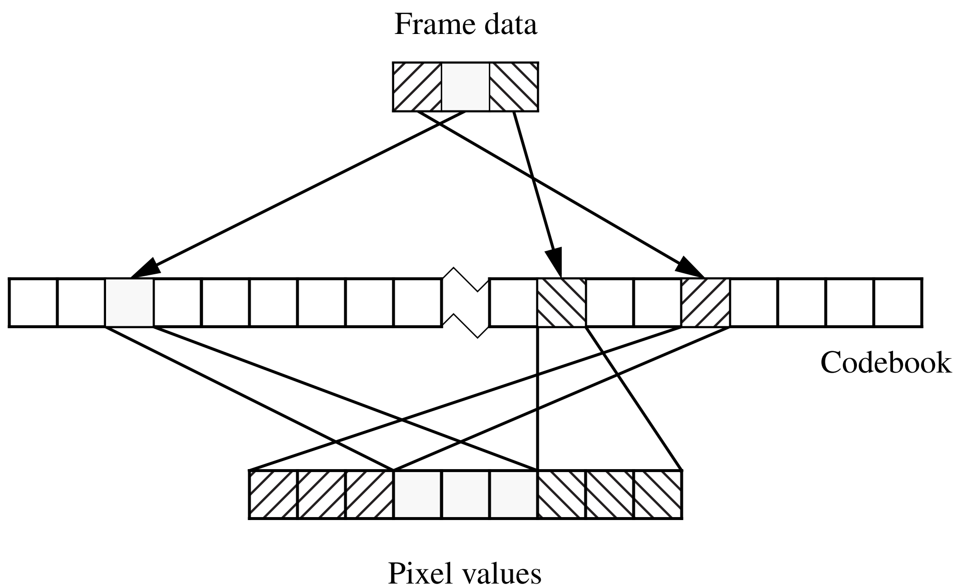

Figure 9 Frame data, codebook, and pixel value relationships, shown by shading. (Shading does not imply anything about the actual color of the pixels, only their relationship to codebook entries and frame data.)

entry, so $C[n][m]$ is the value of the mth pixel in the nth codebook entry. The value of the ith pixel on the screen is then given by

$$P[i] = C[F[\lfloor i/3 \rfloor]][i \bmod 3]$$

The InfoPad video display is two-dimensional, using a straightforward extensions of the above method.

Color spaces are typically described with three independent numbers, frequently red, green, and blue. The InfoPad also separates VQ color data into three channels: one luminance (Y) and two chrominance (I and Q). YIQ represents the same information as RGB, YUV, or CMY, but using different color axes.

The Y, I, and Q planes are stored separately for each frame and each plane has its own codebook. Each Y value (which is 8 bits) corresponds to a 4x4 block of pixels and each I and Q value (each 8 bits) to an 8x8 block. All three codebooks contain 256 entries, which have sixteen values each. For Y, each codebook entry corresponds directly to a 4x4 pixel block. For I and Q, the 4x4 value stored in each codebook entry is scaled to fill an 8x8 block.

3.2. VQ Clip File Format and Transcoder

VQ clip files include the chunk types of other video formats, as well as three specific to VQ. Figure 10 shows the layout of a VQ clip file.

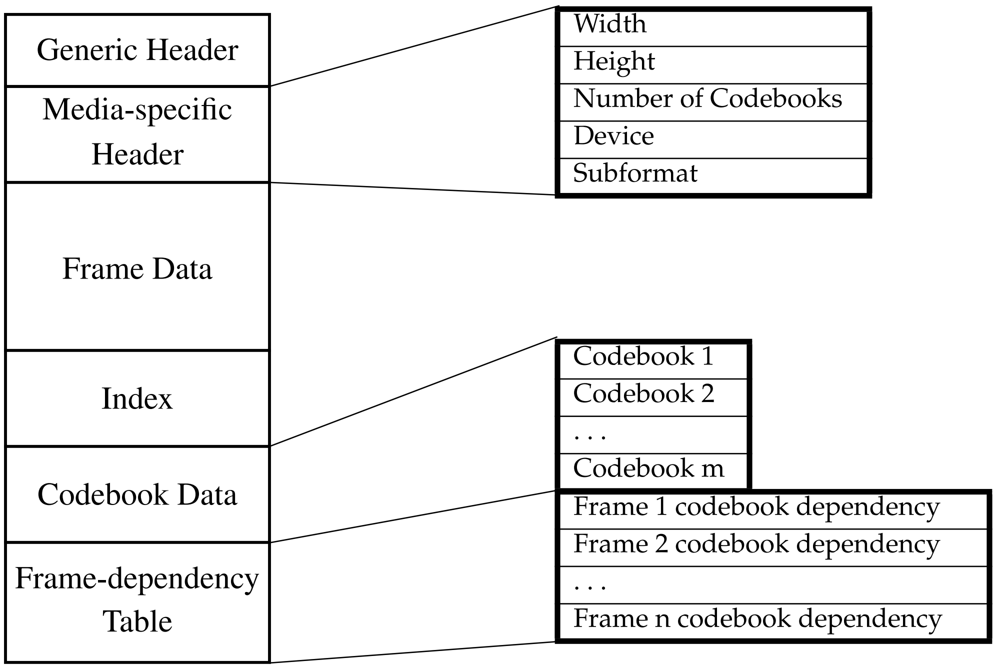

Figure 10 Example clipfile layout for a VQ clipfile. Only VQ-specific chunks are expanded.

The chunks that VQ clipfiles share with other clipfiles are the generic header, frame data, and index. For simplicity, the frame format stored in the clip file is identical to the format the InfoPad expects.

One VQ-specific chunk is the codebook chunk that contains VQ codebooks. Another specialized chunk is the *frame dependency table* (FDT), which specifies the codebook for each frame. The FDT allows for more efficient random access. Codebooks can be used by an arbitrary number of frames, as shown in Figure 11.

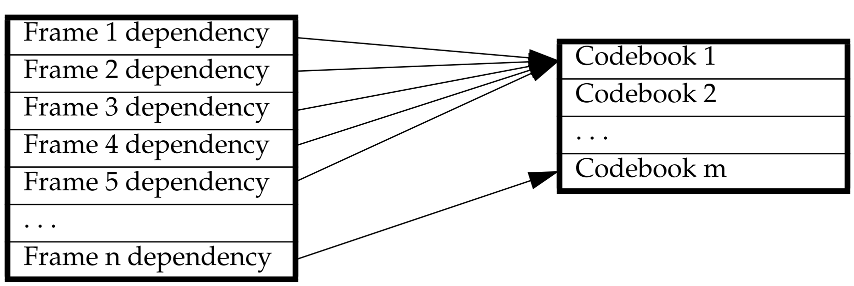

Figure 11 Example Frame Dependency Table (FDT).

Although the figure does not emphasize it, there will be many frames that depend on each codebook in a typical video. The third VQ-specific chunk type is a header that contains the width, height, number of codebooks, device, and subformat.

Since this is a customized format, a program to convert from other video formats to VQ clip files was needed. Independently from CMT integration, an MPEG to VQ transcoder was written. However, the output of this program was not intended for CMT, and as such did not contain any metadata. (That is, it was equivalent to the data and codebook chunks of a clip file.) It would be possible to modify the VQ file object to read "raw" VQ files, as other CMT file objects do, but for simplicity the file object only reads chunk files. Instead, a second transcoder was written to compute the metadata and produce a VQ clipfile from a raw VQ file.

3.3. VQ File Object

The VQ video format has no interframe dependencies, which makes it similar to MJPEG. Therefore, the VQ file object was based on the MJPEG file object.

MJPEG does not have anything analogous to VQ codebooks, so some modifications were required. Separate network channels are used for the frame data and the codebook data, because the two types of data have different quality of service characteristics. The only practical benefit of having two channels is that codebooks will not back up behind frames. One can imagine future network software that allows quality of service specification for different channels, in which case this architecture would be significantly better than a single channel.

The VQ file object can play all of a clipfile or one or more contiguous parts of it, which are called segments. The object keeps a list of all segments to be played. The user may add segments to the list, delete segments from the list, and get information about segments on the list. Similar to other CMT objects, the file object has many user-settable parameters. Appendix A lists the commands available for the VQ file object.

3.4. VQ Play Object

As with the file object, the VQ play object is very similar to the MJPEG play object. It contains all the information about the InfoPad itself. Since the file format was chosen to be close to what the InfoPad expects, the play object does not have to reformat the frame data before sending it to the InfoPad. The play object does reformat the codebook data before sending it. The InfoPad has exactly one codebook in memory at any given time, so the play object has to ensure that the current codebook in the InfoPad matches the frames it sends. The play object relies on the file object to send each codebook before it is time to play the first frame that requires it. If the required codebook does not arrive by the time its dependent frame should be played, the frame is dropped. Since codebooks change infrequently and are significantly larger than frames, the play object caches them so the file object does not have to resend them.

The greatest challenge for the play object is timing the audio and video so they are synchronized on the InfoPad. The audio must arrive no more than 15 ms before or 40 ms after the corresponding video in order for the audio and video to be perceptually synchronized [Stein]. The VQ play object synchronizes audio and video the same way other CMT play objects do, as follows. When a frame arrives at the destination, it is put in a display queue that is prioritized by display time. An event is scheduled to occur when it is time for the frame to be displayed. When the display event occurs, the frame is sent over the network and wireless link to the InfoPad. The audio object similarly queues and plays audio.

There is some delay from the time the play object sends a frame until the time it is played, but since the play object runs close to the Pad, in terms of the network, this delay is small. More importantly, network measurements show that the jitter in the latency is very small [Burg]. Thus, it is possible to tune the play object so that it sends corresponding audio and video at any relative time offset. Therefore, by measuring synchronization on a network, we should be able to calibrate the play object once to obtain acceptable synchronization. Over time this latency may change so we will have to monitor it and feedback the skew to the play object.

The VQ play object has commands similar to other play objects. The VQ play object commands are listed in Appendix B.

4. Experiences

This section describes various experiences we had developing this code and conducting a performance experiment.

4.1. Frame Dropping

A test video consisting of 90 unique frames was generated to determine how many frames were being dropped by the network. To test the software and hardware performance, this video was played on the InfoPad. The video was played on an InfoPad communicating over a wired serial link and on an InfoPad communicating over a wireless radio link. For each type of link, the video was played back at normal speed, half-speed, and one-tenth speed.

Each of these six playbacks on the InfoPad was recorded on a video camcorder. This video was analyzed by hand with a frame-accurate tape deck to determine which, if any, frames had been dropped and to see if the overall playing time of the video was correct.

The raw data from the experiment is shown in Table 1. In wired mode, no frames

Playback Speed	Frames played		Percent dropped for wireless
	Wired link	Wireless link	
Normal	89	76	15
Half	89	87	2
One Tenth	89	89	0

Table 1 Frames played and dropped on actual InfoPad

were dropped at any of the three speeds, so we know that the hardware functions properly. The only difference between the wired and wireless tests is that in the former a serial link is used to connect the InfoPad to the wired network, and in the latter a radio link is used. Therefore it seems likely that the frame dropping in the wireless case is due to radio link errors. Also, the running times were very close to the expected times, which means that the software is sending frames at the right time.

There are two possible explanations of why the wireless link drops frames. One is that the bit-clock training sequence is too short, making the radios unsynchronized. Another is that an error is occurring in the packet header, which could cause several problems. For example, if the packet length were corrupted and made much too long, it would tie up the receiver for a long time. [Tru] Work is being done to make the radio link more robust, which should improve video performance.

4.2. Playback Quality

Currently there are problems with the hardware that limit the playback quality. The most significant problem is that the hardware is not able to play audio. Also, the radio link does not yet have any error correction, so the display frequently shows radio noise.

A systems problem is that of load balancing. One InfoPad is supported by many processes. Since interactive computer usage is bursty, it is better for many Pads to share many computers than for each Pad to have a dedicated computer. In the future the processes for each Pad will be spread across multiple hosts, but now they are consolidated. Additionally, many of the computers that support the Pads sit on graduate students desks and are often being used for other purposes. For these reasons the system is often bursty in video playback and interactive performance.

At times when the system is lightly loaded and the radio channel is not very noisy, the video playback is very good. Video frames come through intact and properly timed, (at least to the naked eye).

4.3. Development Environment

The hardest part about developing the VQ objects was the difficulty of usefully monitoring the state of CMT objects and distributed applications in general. This problem was exacerbated because the most interesting data is the VQ frame and codebook data, which cannot be usefully viewed with the usual debugging tools (e.g. debuggers, text editors).

A test of the complete system had many steps, and problems could occur at each. The first step in this project was adding metadata to raw VQ files to produce clip files. The correctness of the added metadata was difficult to test in isolation, since the only program that could play the output was the CMT VQ objects. The VQ file object reads VQ clip files and sends frames and codebooks over the network to the VQ play object, which forwards it to the InfoPad. Either of these objects could be buggy, and since the output from the file object can only be correctly interpreted by the play object and the play object can only receive data from the file object, it was difficult to test them separately. In the end, debugging code had to be added to both objects to print out frame and codebook indices and raw data, which were examined by hand.

Another challenge was that it required detailed knowledge of two complex systems, CMT and InfoNet, which is the network software infrastructure that supports the InfoPad. At the time this project began, limited CMT documentation was available. CMT has high level abstractions, which are helpful for application writers. At the same time, the richness and complexity of CMT increases the learning curve for programmers wanting to add to the toolkit.

The system design had some good points, also. Both CMT and InfoNet were developed under UNIX and both were written in C. InfoNet treats all network traffic as simple bytes, so sending information in any format is easy. The InfoPad architecture is designed to make it easy to integrate network servers for different data types.

5. Future Directions

There are both high- and low-level issues that could be addressed in future work. A useful low-level project would be to add more video and audio source objects and a transcoder object to CMT. This would enable CMT to capture live video and audio and sent it to the InfoPad, and live audio to captured on the InfoPad and sent back. This would enable 1.5-way video conferencing. Software has already been written that enables similar conferencing, but it is less flexible because it is centered on the MBONE.

Another low level enhancement would be to extend the file object to read raw VQ files. Currently, the file object can only read VQ chunk files, that is, files that have been preprocessed to have index information. There is no reason in principle why the file object could not construct the index information on the fly. This would remove one step from the conversion of stored video in another format to something that can play on the InfoPad and might make a transcoder object simpler.

More and more video is being stored across the network and in various formats, none of it VQ. Also, in live video broadcasts, such as videoconferencing, even if the source is capable of transmitting VQ, it may not desire to if most of the recipients want a different format. For CMT to provide end-to-end delivery, it needs to be able to transcode among video formats.

A higher level issue is whether or not VQ is a good choice for a device like the InfoPad. VQ is expensive to encode, which is a big disadvantage for transcoding. Advantages of VQ are low cost to decode and tolerance of bit errors. It is possible that a better format might be one based on DCTs, which would enable easier scaling, and which had no interframe dependencies.

A still open question is whether or not the current strategy for audio/video synchronization is sufficient. Under the current system, the audio and video streams are synchronized by the play object just before they are sent to the InfoNet. If the streams travel a small number of network hops before being transmitted over the wireless link to the InfoPad, it is possible that little latency jitter will be introduced and thus the streams will remain synchronized. However, it is also possible that these hops will introduce too much latency jitter. If that is the case, it could be remedied by synchronizing at the machine that transmits the data to the InfoPad. It could also be solved by synchronizing on the InfoPad itself, but for power reasons it would be more desirable to synchronize on the gateway.

References

[Barr94] B. Barringer, T. Burd, F. Burghardt, A. Burstein, A. Chandrakasan, R. Doering, S. Narayanaswamy, T. Pering, B. Richards, T. Truman, J. Rabaey, R. Brodersen; “Infopad: A System Design for Portable Multimedia Access”, Calgary Wireless 94 Conference, July 1994.

[Burg] F. Burghart, “InfoPad Network (IPN) Performance,” `http://infopad.EECS.Berkeley.EDU/research/infonet/perform.papers/report1/report_ipn1.fm.html`.

[Chan] A. P. Chandrakasan. A. Burstein, and R. W. Brodersen, “A Low-power Chipset for a Portable Multimedia I/O Terminal,” IEEE Journal of Solid-State Circuits, p. 1415-1428 (December 1994).

[Oust94] J. Ousterhout, “Tcl and the Tk Toolkit.” Addison-Wesley, 1994.

[Pate96] K. Patel, “Introduction to the CM Toolkit,” `http://bmrc.berkeley.edu/projects/cmt/cmtdoc/CMT_Intro.html`.

[Rowe94] L.A. Rowe, “Continuous Media Applications,” Presented at Multipoint Workshop held in conjunction with ACM Multimedia ‘94, San Francisco, CA, Nov. 1994.

[Schu96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, 01/25/1996.

[Smith] B.C. Smith, “Implementation Techniques for Continuous Media Systems and Applications,” Ph.D. Dissertation, Computer Science Division - EECS, U.C. Berkeley, December 1994, `ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/papers/bsmith-thesis.ps.gz`.

[Stein] R. Steinmetz and C. Engler, “Human Perception of Media Synchronization,” Technical Report 43.9310, IBM European Networking Center, Heidelberg, Germany, 1993.

[Tru] T. Truman. Personal communications, April 1996.

Appendix A : VQ File Object Commands

Command	Option(s)	Description
configure		Set/get object parameters
	-lts name	Logical Time System object to use
	-sendAhead seconds	Number of seconds worth of video to send ahead
	-bandwidth bps	Maximum bandwidth to provide
	-cycleTime	How often to wake up and send data
	-fileName filename	Clip file to read data from
	-outCmd	Command to invoke to send data
	-codebookOutCmd	Command to invoke to send codebooks
	-maxFrameRate	Maximum number of frames/second to send
destroy		Delete object
addSegment clipStart clipEnd		Add a new segment to the segment list. Returns segmentId.
	-start	Specify logical start time for segment (defaults to 0)
	-end	Specify logical end time for segment (defaults to length of source)
clearSegments		Clear the segment list
segmentInfo segmentId		Return data about a segment
ready		Make the object ready to send data
unready		Make the object unready to send data
info		General data about the object

Appendix B : VQ Play Object Commands

Command	Option(s)	Description
configure		Get/set object parameters
	-width pixels	Width of image (center image if -1 and squeeze off)
	-height pixels	Height of image (center image if -1 and squeeze off)
	-xpos pixels	x offset of video within window
	-ypos	y offset of video within window
	-squeeze n	If nonzero, resize video to fit the window
	-xid	X window id or InfoPad ID
	-device deviceName	Specify decoding device (only “default” supported)
	-lts	logical time system to use
	-maxFrameRate n	Maximum number of frames/second to play. Not yet implemented.
	-ctrlCallback procName	Procedure to call for flow control. Not yet implemented.
	-ctrlCycleTime seconds	How often to do flow control. Not yet implemented.
destroy		Delete object
accept ScatterBufferList		Put ScatterBufferList of data in display queue.
acceptCodebook ScatterBufferList		Set new codebook
ready		Ready the object
unready		Unready the object
stats		Get decode/playback statistics
refresh		Refresh display with current frame
reinit		Reset object and device to default state