master

Disc Building for Sun Workstations®

Optimage Interactive Services Company, L.P.

ACKNOWLEDGEMENTS

I would like to thank Vlad, Randy and the FMV guys for all their assistance in revising this document. I would also like to add my thanks to Dave and Siraj for their helpful comments.

REPRODUCTION NOTICE

The software described in this document is intended to be used on a single computer system. OptImage expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of OptImage and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

For additional copies of this software and/or documentation, or if you have questions concerning the above notice, the documentation, and/or software, please contact the OptImage representative in your area.

DISCLAIMER

The information contained herein is believed to be accurate as of the date of publication; however, OptImage will not be liable for any damages, including indirect or consequential, from the use of OptImage-provided software or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

COPYRIGHT

Publication Editor:	Walden Miller
Contributing Writer:	Tom Dirks
Publication date:	June 1993
DPN:	OPT135

TRADEMARKS

OptImage Interactive Services Company, L.P.
1501 50th Street, Suite 100
West Des Moines, Iowa 50266

Phone numbers:

In the USA: **1 800 CDI-5484**
(515) 225-7000
(515) 225-0252 FAX
In Europe: **+31 (40) 73 6228**
In Japan: **(03) 3 665 9740**

Online Information:

Applelink® D6431
America Online®, Keyword: OptImage
OptImage B.B.S. (515) 225-1933

Table of Contents

Introduction

About this manual

This manual describes the process of generating disc images for CD-I using the **master** utility. The **master** utility reads a script file that controls the creation of realtime files and the placement of CD-I assets in the disc image. From the information contained in the script file, the **master** utility creates the disc image containing the assets and realtime files.

---

*NOTE: The **master** utility version 4.1 replaces the **green** and **master** utilities provided in previous releases of DiscBuild. **master** 4.1 has all of the functionality of previous versions of **green** and **master** and reads previously written (syntactically correct) **green** and **master** scripts without error.*

This manual contains three chapters and eight appendixes.

Chapter 1 describes how to run **master**. This chapter provides a basic overview of the disc building process and the **master** runtime options.

Chapter 2 provides an in-depth look at how **master** operates. This chapter provides the information on disc image layout required to build an efficient script file. Used in conjunction with the script

examples provided in chapter 3, this chapter explains how **master** generates a disc image.

Chapter 3 describes how to build a script file. This chapter contains a complete description of the script syntax. Example scripts with their resulting disc image sector maps provide an inside look at how the script file controls the disc image.

Appendix A provides the script syntax without the detailed explanation found in **Chapter 3**.

Appendix B contains audio/video source mask values.

Appendix C lists the keywords reserved by the **master** utility.

Appendix D describes the Balboa PMM data structures optionally inserted in the disc image by **master**. These data structures can be accessed by the Balboa Play Manager and other runtime programs to control the application's use of realtime data.

Appendix E provides an example realtime file sector map generated by **master**.

Appendix F provides an example table of contents report generated by **master**.

Appendix G describes the entry point list data structures built by **master**.

Appendix H lists the **master** error codes and their descriptions.

Manual conventions

All example pathlists in this manual use the CD-RTOS/Sun pathlist convention of separating pathlist elements with a slash (/). PC pathlists normally use the backslash (\) as a pathlist element separator. Because the backslash character is a reserved escape character, pathlists in PC master scripts must use the Sun conventions. Macintosh pathlists should use the standard colon (:) separator. For example, the following full pathlist is shown in Sun, Macintosh, and PC conventions:

/h0/video/image1	Sun
h0:video:image1	Macintosh
/h0/video/image1	PC

The following typographical conventions are used throughout this manual:

- The names of programs, directories, and files referred to in the text, and command line examples are shown in **bold Helvetica-Narrow** typeface. For example:

  master

  mount /devsd2a /home/emulate

- The names of keys on the keyboard are *italicized* when referred to in the text. For example:

  Enter

  Escape

- In syntax descriptions, the following conventions are used:

  - Variable syntax elements are shown in *Palatino-Italic* typeface. For example:

    filename

- Constant syntax elements, such as keywords, separators, and punctuation, are shown in **bold Helvetica-Narrow** typeface. For example:

  audio

- Syntax elements within italic brackets (*[]*) are optional. For example:

  [real_time]

- Repeated syntax elements are indicated by an ellipsis (...) following the element to be repeated. For example:

  option...

- A syntax bar (|) is the equivalent of a boolean OR; one, but not both, of the syntax elements on each side of the bar may be used. The bar is used in conjunction with italic braces (*{}*) which indicate the limits of a syntax group. For example:

  {at | by} *rsn*

  *{element1 | {***keyword** *element2} | element3}*

The following is a syntax description for the command line of **master:**

master *[option...] [script] [option...]*

where

master	is a keyword and must be typed exactly as it appears in the syntax command line.
option	is a variable replaced by whichever **master** option you find appropriate. Zero or more options can be used before or after the *script* element.
script	is replaced by the name of the actual script file. A maximum of one script file may be specified.

NOTE: The following examples are only show how syntax is interpreted; you do not need to understand the command lines for this demonstration.

The following is an example of the above **master** command line:

master -d -m=maps myscript -o=maps.pmm

where

master	is a keyword
-d	is an *option*
-m=maps	is an *option*
-o=maps.pmm	is an *option*
myscript	is the *script*

The following is a more complex example of a syntax description for a **master** script element:

from *[ident...] asset* **[delta_file** *dfile]* **ep_list** *epspec [{***at** | **by***} rsn]*

where

from, delta_file, ep_list, at, and **by**
are keywords and must be typed exactly as they appear in the above statement.

ident, asset, dfile, epspec, and *rsn*
are variables you replace with the appropriate strings as defined in the text.

ident is an optional variable and can be used more than once in the above statement.

delta_file *dfile* and **at** | **by** *rsn*
are optional specifications that can occur only once each. If using **at** | **by** *rsn*, the possible specifications are **at** *rsn* or **by** *rsn*.

The following are examples of the above syntax.

Example 1:

from "test.pim" delta_file "test.dlt" ep_list off

where

"test.pim"	is an *asset*

"test.dlt" is a *dfile*

off is an *epspec*

Example 2:

from "test.pim" ep_list off by 04:50:30

where

"test.pim" is an *asset*

off is an *epspec*

04:50:30 is an *rsn*

Example 3:

from "test.pim" delta_file "test.dlt" ep_list "link_ep" with index from "test.pep" internal and gap 10 at 04:50:30

where

"test.pim" is an *asset*

"test.dlt" is a *dfile*

"link_ep" with index from "test.pep" internal and gap 10 is an *epspec*

04:50:30 is an *rsn*

Chapter 1 Running master

This chapter provides a basic overview of the **master** *utility and the command line syntax necessary to run* **master**.

Overview

The **master** utility has two main functions: to generate realtime files based on any number of assets and to build disc images. **master** accomplishes both tasks by reading a script command file along with the source files/assets specified within the script. **master** determines the scope of its activity (to either generate a realtime file or a disc image) by the syntax of the script file.

master Input

master input consists of a script file and a set of source files containing various data types, such as:

- CD-DA sound
- ADPCM Audio
- MPEG Video (Full Motion Video)
- images
- data
- programs
- realtime files

master output

master output is determined by the function of the script file and the command line options.

When **master** generates a realtime file, its output consists of:

- **a realtime file**
  This is the main output. You specify the name of the realtime file on the command line or on the options line of the script file. The generated realtime file can be used as input for a disc image at a later time.
- **an optional sector map report**
  The sector map provides a formatted listing of the placement of asset sources making up the realtime file, and a sector-by-sector map of the realtime file. The sector map is useful for examining how efficiently the disc image is laid out. Examples of sector maps are provided in Chapter 3 and Appendix E.
- **an optional BALBOA PMM structure file**
  To access the realtime file with the Balboa Play Manager at the map level, PMM structures must be embedded in the realtime file using the **-o** or **-z** options. These structures may also be written to an external file using the **-o** option. A complete description of the PMM structures is provided in Appendix D.
- **status messages displayed on your output terminal**
  master generates status messages as it reads the script and generates the realtime file. An example of this output is provided in the *Disc Building Installation* manual.

When **master** generates a disc image, its output consists of:

- **a disc image**
  The disc image is the main output. The disc image name is specified in the **master** script file by the volume name specification.
- **a tracks (table of contents) report**
  The tracks report describes all the files in the disc image. The tracks report is required for emulation and disc recording.

- **an optional sector map for each generated realtime file**
  The sector map provides a formatted listing of the placement of asset sources making up each realtime file, and a sector-by-sector map of each realtime file. The sector map is useful for examining how efficiently the disc image is laid out. Examples of sector maps are provided in Chapter 3 and Appendix E.
- **an optional runtime map file**
  To access the realtime file with the Balboa Play Manager at the map level, PMM structures must be embedded in the realtime file using the **-o** or **-z** options. These structures may also be written to an external file using the **-o** option. A complete description of the PMM structures is provided in Appendix D.
- **status messages displayed on your output terminal**
  master generates status messages as it reads the script and generates the disc image. An example of this output is provided in the *Disc Building Installation* manual.

Command line syntax

The **master** command line syntax for building CD-I disc images is:

master *[option...] [script] [option...]*

The **master** command line syntax for building realtime files is:

master *[option...] [script] [option...] [outfile] [option...]*

where

option	is any option compatible with the script function. The options are described below.
script	is the name of the script file to be read.
outfile	is the name of the generated realtime file.

If you specify only one file name on the command line, **master** assumes it is a script file. The disc image is created in the same directory as the script file.

The disc image is created with the name provided by the volume name specification in the **master** script.

Command line options

Options may be specified either on the command line or within the script file. If specified on the command line, they may appear in any order. The only restriction is that options must be compatible with the specified script's function. For example, do not use disc-building options with a realtime file building script.

To specify options from within the script file, use the keyword **options:** followed by the list of options. The options line must be the first non-comment line in the script. All file names specified with an option must be placed within double quotes. For example:

```
options: -d -m="maps.lis"
```

When building realtime files, you can also specify the output file within the script file by specifying the file name within quotes on the option line. For example:

options: "cd_green_file" -m="maps.lis"

General options

NOTE: Options -r, -o, and -z are mutually incompatible.

- **Reads script from standard input**
master reads standard input until a *return, escape, return* character sequence is typed, indicating the end of the script.

-d **Dry run (no output file)**
When this option is selected, **master** produces no disc image output. This option is helpful when used in conjunction with the **-m** option to check whether the sector allocation is as you intended it.

-m[*=map_file*] **Generate allocation map report**
master generates a sector map report for each non-premastered realtime file when you select this option. If you do not specify *map_file*, the map report is named **maps.lis**.

-o[*=PMM_file*] **Generate PMM_RT_MAPS in external file**
This option generates **PMM_RT_MAP** structures in each realtime file that uses labels and writes them to an external file. If you do not specify *PMM_file*, the map structures are created in **RT_file.pmm**. The PMM map structures are used by the OptImage Balboa Play Manager to control an application's use of realtime data. Refer to Appendix D for a full description of the map structures.

-q **Quiet mode**
When this option is selected, **master** generates no progress report messages. Normally, **master**

generates messages as it creates each volume, file, and directory for the disc image.

-r **Build runtime maps**
This option generates "old format" runtime maps in the first sector of each realtime file.

-s **Disable sector scrambling**
This option disables sector scrambling. This option is useful for emulators that do not require scrambled disc images. The time needed to build a disc image is reduced with unscrambled sectors.

-z **Generate embedded PMM_RT_MAPS**
This option generates **PMM_RT_MAP** structures in each realtime file that uses labels. These map structures are used by the OptImage Balboa Play Manager to control an application's use of realtime data. Refer to Appendix D for a full description of the map structures.

-h/-? **Command line help**
Displays command line syntax help.

CD disc image generation options

-a=*char_file* **File for CDROM-XA character set**
This option specifies a file that contains a permissible character set for CDROM-XA file names. The default file name is **XA_chars.txt**.

-c **Convert empty sectors to data sectors**
This option directs **master** to convert empty sectors with events to data sectors in channel 31. This option is used for compatibility with previous versions of **master** (2.5 or earlier).

-e **Generate embedded directories**
This option directs **master** to generate a depth-first directory structure instead of a breadth-first

directory structure. Refer to Chapter 2 for a discussion on these directory structures.

-g **Non-Green mode (relaxed checks)**
When this option is selected, **master** does not check for the presence of an **abstract, copyright, biblio,** or **application** file, or a **message** area. The absence of these files causes a warning instead of an error when this option is selected.

-l*[=N]* **Generate N seconds of leadout**
This option generates the specified number of seconds of leadout for each volume of the album. Leadout may range from 210 to 600 seconds and defaults to 210 seconds.

-nt **Suppress generation of a tracks report**
This option stops **master** from generating a tracks report.

-p **Obey padding requests**
When this option is selected, **master** complies with the padding requests in the script file. Files to be padded are only extended if this option is used. Padding a file allows you to replace it (using **cdedit**) with a file that is slightly larger than the original without rebuilding the entire disc image.

-t=*toc_file* **Generate tracks report**
If this option is selected, **master** writes a tracks (or table of contents) report for each volume in the disc image to the specified file. If you do not specify a file name, **master** creates a tracks report named **tocs.lis**.

-u **Generates a CD-I Ready Image**

-xa **Generates a CDROM-XA Image**

Example command lines

The following command line reads the **script.cd** file and (depending on the script) creates a realtime file or disc image in the current directory. The name of the output file is determined by the disc image's specified volume name or by the output file name specified on the **options:** line within the **script.cd** file.

```
master script.cd
```

The following command line reads the **script.cd** file and creates a tracks report in the current directory with the name **sample.toc**. No other output files are created because the **-d** (dry run) option is specified.

```
master script.cd -d -t=sample.toc
```

The following command line reads the **script.cd** file and creates a disc image in the current directory. The name of the disc image is determined by the specified volume name within the **script.cd** file. Because the **-p** option is specified, **master** will enforce the padding requests contained in the directory specification portion of the **script.cd** file.

```
master script.cd -p
```

Chapter 2

Generating Disc Images Using master

This chapter provides a description of how ***master*** *interprets script files to generate disc images.*

Building a disc image

The disc-building process consists of four steps.

1. Gathering your assets.
2. Writing your **master** script.
3. Generating the disc image using **master**.
4. Emulating the disc image for verification and quality assurance.

Steps one and two are accomplished for most titles using authoring system programs such as **MediaShowcase, Cinergy,** or **MediaMogul**. These programs create titles using intermediate script languages which are then translated into **master** scripts by programs such as **Talk to Disc** or **Script to Disc**.

If you are using an authoring program that generates your **master** script, use the file/asset placement strategy recommended by the specific program.

Writing your script

A **master** script is a text file that describes the assets that will be used to generate your disc image. The **master** script syntax allows you to specify the directory structure of the disc image on the file level and how assets are interleaved in a file on the asset level. The **master** script syntax is described in detail in Chapter 3.

If you are using an authoring system that generates your **master** script, you should be aware of the **master** command line options that provide diagnostic sector and track maps, as well as the dry run option (**-d**) that suppresses the generation of the disc image. These options test the script and image before you invest in the actual disc-building time.

If you are going to use the Balboa Play Manager (PMM) map options (**-o** or **-z**), you must use record- or stream-level labels with your realtime files. If no record labels are specified, no maps will be generated.

If you are writing your **master** scripts without the assistance of an authoring tool, use the **master** script syntax provided in Chapter 3.

Generating a disc image or a realtime file

master can be used to generate realtime files individually or to generate a disc image from many realtime files and other CD-I assets. You do not have to generate the realtime files first as the script to make a realtime file is a true subset of the script to make a disc image.

By building realtime files before you build the disc image, you use more disk space (i.e., space for the realtime file and the disc image). However, it is faster to rebuild a disc image if the realtime files are pre-mastered. The final disc image is the same regardless of the strategy used.

master input assumptions

While **master** basically reads a script file and generates the subsequent disc image or realtime file, it makes the following assumptions about the source files specified in the script:

- For PCM sources, one CD-I sector contains 2352 bytes (1176 samples). CD-DA byte ordering is assumed for implicit form (no byte swap required), while byte swap is performed on IFF PCM samples.

  AIFF AESD chunks are not processed (emphasis).

  BlockSize information from AIFF SSND chunks is not taken into account.

- For ADPCM audio sources, the data portion of one CD-I sector contains 2304 bytes of source (i.e. 18 sound groups + 20 padding bytes).

  AIFF AESD chunks are not processed (emphasis).

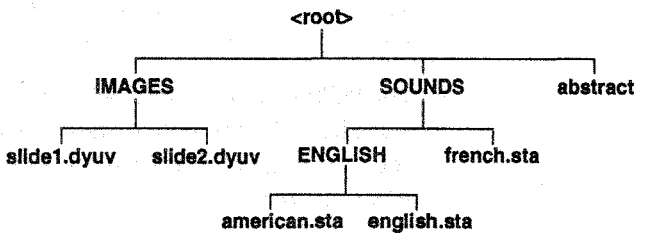
  BlockSize information from AIFF APCM chunks is not taken into account.

- For video sources, the data portion of one CD-I sector contains 2324 bytes of source.
- For data sources, the data portion of one CD-I sector contains 2048 bytes of source.
- Ready sources contain sequences of 2352-byte CD-I sectors.

File placement in the disc image

Files and directories are logically organized as a tree, but must be generated linearly on the disc image. The linear placement of files on the disc image directly affects the seek time between files and may affect the timing of your title.

master uses two file placement strategies. The following file system is used to compare both strategies.



The default strategy uses a breadth-first traversal of the directory tree. That is, each file is generated as each directory level is scanned, and subdirectory generation is delayed.

Starting with the volume's root directory, subdirectories are pushed in a FIFO (first-in, first-out stack) as encountered. The process then loops, popping and scanning subdirectories, generating files, or pushing subdirectories, until the FIFO becomes empty (the same process is used to generate the path table file). This leads to contiguous directory and file placement.

This breadth-first strategy allocates files for the tree example in the following order: **<root>, abstract, IMAGES, slide1.dyuv, slide2.dyuv, SOUNDS, french.sta, ENGLISH, american.sta, english.sta.**

The second strategy uses a depth-first traversal of the directory tree and can be forced by specifying the **-e** option. Each file is generated as each directory level is scanned, but when a subdirectory is encountered, the process recurs generating this subdirectory. Subdirectories and their files are consequently embedded within each other.

This depth-first strategy allocates files for the tree example in the following order: **<root>, IMAGES, slide1.dyuv, slide2.dyuv, SOUNDS, ENGLISH, american.sta, english.sta, french.sta, abstract.**

Emulating the disc image

To test the disc image before pressing, you should always emulate the image. Emulation can provide both a valuable insight into the flow of a title and a quality assurance check on the final title.

To emulate a CD disc image, use the instructions provided with the *Emulation* manual provided with the Disc Building utilities.

Realtime file definitions

The heart of **master** disc building is its interpretation of the realtime file definition; a CD-I disc image is only a sophisticated file system for a CD-I disc. The realtime file script definition specifies the exact internal structure (sequential or interleaved) of the sources from which it will be built. To understand how the realtime file definition is interpreted, it helps to understand how the elements that make up a file definition relate to each other.

A realtime file definition is made up of one or more record specifications. A record is made up of one or more data streams. A data stream is made up of one or more stream assets (sources).

Records

Records are placed in the realtime file in the order they are specified within the realtime file definition. Records cannot be interleaved. Each realtime file must have at least one record definition.

Streams

A record definition is made up of one or more data streams. A stream is defined as a collection of related assets.

Streams may be interleaved or arranged consecutively in the record. If the streams are to be interleaved, the order in which they appear in the record specification helps determine how they are interleaved.

master places the first stream in the record at the first sector unless a sector offset is specified using the stream level **at** *rsn* (*relative sector number*) clause. If the **at** *rsn* clause is given, **master** starts the first stream at the sector number specified by the clause.

While the term relative sector number (*rsn*) implies a "geographic" specification, the *rsn* is actually a time clause that specifies when a stream/asset occurs within the record/stream.

The *rsn* is expressed either as *minutes:seconds:frames* or as an absolute number of frames relative to the beginning of the record.

master allocates subsequent streams in the record according to the free space available in the record and a stream's *rsn* clause.

master places a stream in the channel specified by the stream definition. Streams of the same type cannot be placed in the same channel within a record. For example, you cannot place two audio streams in the same channel, if they are specified in the same record.

Assets

Each stream is made up of one or more assets. For example, a video stream may contain a number of video clips. Stream assets specified for a single stream cannot be allocated to different channels. Assets within a stream are not interleaved. The last sector of an asset always precedes the first sector of the next asset within the stream. To be interleaved, assets must be specified in separate streams (i.e., separate channels).

The exact placement of assets within a stream is ultimately determined by the presence or absence of an *rsn* clause within the asset definition.

There are two types of *rsn* clauses in asset definitions: **at** *rsn* and **by** *rsn*. **at** *rsn* specifies the beginning sector number for the asset. **by** *rsn* specifies the ending sector number for the asset.

If an *rsn* clause is used in an asset definition, the *rsn* specifies where a stream asset is placed. If the *rsn* clause is absent, the asset is placed directly after the preceding asset. If a conflict between asset time clauses occurs, **master** returns an error.

If you specify a stream-level time clause, all asset-level time clauses are offset by the amount specified at the stream level. For example, the specification **at 2** on a stream level is equivalent to adding **+2** to each asset-level time clause.

Realtime audio assets

If a realtime audio stream contains an asset with an explicit time clause, the first asset in the stream must also use an explicit time clause or **master** returns an error. However, if the first audio asset uses a time clause, subsequent assets are not required to include time clauses and are placed directly after the first asset in the order they are specified. If subsequent audio assets use time clauses, **master** attempts to place the assets according to their time clauses. If an asset's time clause specifies the beginning of the asset before the end of the previous asset, **master** returns an error.

Stream hierarchy

Streams are interleaved according to their allocation priority. The allocation priority is determined by the order of specification, its time clause (*rsn*), and, optionally, any offset information given.

Realtime MPEG and realtime audio streams have the highest allocation priority and must appear in the record definition before any of the lesser priority streams: video, audio, and data. Video, audio, and data streams have equal allocation priority. Streams with equal priority are allocated in the order they are given in the record definition.

Two MPEG streams cannot be declared in the same record. Two MPEG video assets, however, can be declared within the same stream. More than one realtime audio stream may be declared in the same record if each stream is specified with a different channel number.

The stream-level time clause indicates where a stream is allocated within the record. The allocation of the first stream in the record definition is determined by its **at** *rsn* specification. If the time clause is omitted, the first stream starts at the first available sector of the record. The next stream is interleaved into the empty frames of the previous stream unless a time clause is specified. If interleaved, the stream is placed in frames before, amidst, and/or after the previously specified stream data.

master reports

master generates two types of reports that are useful for analyzing the realtime file or disc image created by **master**:

- sector map reports
- tracks reports (disc images only)

Sector map reports

master generates a map report (requested by the **-m** option) that describes each realtime file it creates. For each realtime file, **master** generates a header and a sector map.

The map header describes each record, each stream, and each asset. The asset information includes the following:

- a hexadecimal number (modulo 255) by which the asset is denoted in the sector map section
- file name
- file size, both in bytes and in sectors
- coding byte value and name
- positioning constraints

Each line of the sector map section contains the following:

- the file relative sector number of the first sector of the line in minutes:seconds:frames format and in absolute frame format
- the record relative sector number of the first sector of the line in absolute format
- source numbers of the next 16 sectors (empty sectors are denoted by a period (.)), preceded or replaced by a # if the sector has events (EOR, TRIGGER, EOF)

An example sector map is provided in Appendix E.

Tracks report

master automatically generates a tracks report that describes the tracks contained in the disc image. **master** generates tracks as follows:

- One single green track is generated, padded up to the next second boundary with empty sectors.
- One red track is generated for each red file (sequence is controlled by the file placement policy). Red tracks are preceded by an audio pause of 2 seconds and padded up to the next second boundary with PCM silence.

The table of contents (TOC or tracks) report is designed to be either read by the CD mastering factory to control the generation of the Q channel stream or parsed by an emulator program for the same purpose. The report is output according to the following syntax:

```
toc      := leadin track ... leadout
leadin   := leadin { audio | data }
track    := track { audio | data } index ...
index    := index NUMBER time
time     := NUMBER : NUMBER : NUMBER
leadout  := leadout { audio | data }
```

Comments begin with an (!) and end at the end of the line.

An example tracks report is provided in Appendix F.

Chapter 3

Building master Scripts

This chapter provides an in-depth description of the **master** *script syntax and some example scripts to examine.*

Building a script

To build a script, you must first understand the basic building blocks of a script. A script is made up of four major elements:

- album definition (disc image only)
- volume definition (disc image only)
- file definition
- directory structure definition (disc image only)

The heart of these elements is the file definition. A disc image is basically a directory structure of files assembled and allocated by **master**. A file definition contains specific asset definitions that specify where **master** can find the data necessary to generate the file.

Each of the above elements are created from basic script building blocks, such as filenames, IFF specifications, keywords, etc. The following sections provide an overview of the basic script elements and a description of the basic definition building blocks.

Script file elements

The **master** script file consists of four elements:

- **album definition**
  A CD-I album definition consists of an album name and, optionally, a preparer or publisher name. These names are followed by a sequence of volume definitions to be included in the album. There is only one album definition for each **master** script file.
- **volume definition**
  A CD-I volume definition consists of the volume name and the name of the output file that contains the resulting disc image. These names are followed by a sequence of file and directory definitions. More than one volume can be defined in a single script file.
- **file definition**
  A file definition consists of a file type (color), an identifier, and the source(s) that the file depends on. Each realtime file definition consists of one or more record definitions, which in turn consist of one or more data streams. More than one file can be defined for each volume defined.
- **directory structure definition**
  A directory structure definition consists of a list of all disc image directories and files (and, optionally, their attributes), and their hierarchical placement in the disc image. There is only one directory structure definition for each **master** script file.

The definition of these elements provides **master** with the information necessary to build a CD-I disc image or a realtime file. When you only want to create a realtime file, your script should contain only realtime record specifications. If other elements are present in the script, **master** will assume a disc image is desired and may generate errors if all necessary elements are not present.

Script file building blocks

The **master** script language is built on the following syntactical building blocks:

- include files
- identifiers
- realtime labels
- keywords
- strings
- IFF ID's, wrappers, & chunks
- sector numbers
- numbers
- separators
- whitespace
- comments

The following subsections define the above building blocks used in the script file.

Include files

A script file can embed other partial scripts by enclosing the script's file name within angle brackets (<*partial_script_name*>). The bracketed file is referred to as an include file. When **master** encounters an include file specification, the script input from the current file is suspended and **master** begins reading the new file. When the end of the include file is reached, **master** resumes input from the script file. An include file specification can occur anywhere within the **master** script. Include file specifications may be nested up to eight levels.

For example, the following script segment shows how an include file could be used to specify the required green book files:

```
! ALBUM DEFINITION
define album    "Include File Example"
publisher       "OptImage/Des Moines"
preparer        "Walden Miller"
!
! VOLUME DEFINITION
volume          "Build Demo" in "demo.cdi"
!
! REQUIRED FILE DEFINITIONS
</home/common/required>          ! this is the include file spec
application file cdi_app       from "cdi_application"
.
.
.
```

where **</home/common/required>** contains the following lines:

```
message                        from "/home/common/message.cda"
copyright file  Copyright      from "/home/common/copyright.txt"
abstract file   Abstract       from "/home/common/abstract.txt"
biblio file     Biblio         from "/home/common/bibliographic.txt"
```

Identifiers

Identifiers are alphanumeric strings that refer to files. An identifier can consist of the following characters:

upper case letters (A-Z)

underscore characters (_)

lower case letters (a-z)

decimal digits (0-9)

Identifiers may not begin with a decimal digit and may not be longer than 32 characters. **master** will truncate any identifier longer than 32 characters and issue a warning. Identifiers are case sensitive.

The following are examples of legal identifiers:

file9

Copyright *case sensitivity makes this legal*

Record_Identifier

The following are not legal identifiers:

mono_c *keywords may not be used*

3rd_file *identifiers may not begin with a number*

A_Very_Long_Identifier_Will_Produce_A_Warning

Realtime labels

Realtime labels are special identifiers that refer to realtime records, realtime streams and assets within records. Labels are used by the Balboa Play Manager to access realtime records, streams, and assets. Labels are also used by **master** to place trigger and end-of-record (EOR) bits in realtime files.

Realtime labels use the following format:

identifier[(expression)]

where

identifier is the label name that refers to the record or stream. Label names conform to all the naming conventions of identifiers. All label names must be unique within the script.

expression is the optional value of the label. *expression* has the following formats:

constant	hexadecimal or decimal
(expression)	
expression + expression	
expression - expression	
expression / expression	
*expression * expression*	

If *expression* is not given, the label value defaults to 0.

Lists of realtime labels use two formats. The format used depends on where the label occurs in the realtime record definition:

rtl,rtl,...	*labels for records and streams are separated by commas*
rtl:rtl:...	*labels for assets are separated by colons and the label list must end in a colon*

The following are examples of realtime labels:

record1 (32)	
Record1 (32)	*case sensitivity makes this different*
Asset (32+2):	*assets rtls must end in a colon*
rec1, rec2, rec3 (35)	*record labels are separated by commas*
audio1:audio2:	*asset labels are separated by colons*

Keywords

The following words cannot be used as identifiers because they are reserved by **master** as keywords. All letters in keywords are lower case:

abstract	album	and	application	at
audio	biblio	block	by	channel
clut4	clut7	clut8	copyright	data
define	delta_file	double	dyuv	emphasis
eors	ep_list	even	every	file
from	gap	green	hidden	high
in	index	interleave	internal	length
mask	message	mpeg	mono_a	mono_b
mono_c	normal	odd	off	offset
owner	packed	pre_emphasis	preparer	protection
publisher	qhy	real_time	record	red
rgb555	rgb555l	rgb555u	rl3	rl7
silence	specific	stereo_a	stereo_b	stereo_c
still	swapped	triggers	video	volume
with	yellow			

Strings

Strings consist of up to 128 characters and are specified by enclosing them within double quotation marks ("this is a string"). The following non-graphic characters may be included in a string by using a backslash:

\b backspace

\r return

\f form feed

\t horizontal tab

The newline character (\n) is used to continue strings across lines and is not interpreted as part of the string.

In all other instances, the backslash character informs the interpreter to use the subsequent character in the string. Therefore, backslash characters and double quote characters may also be included by using \\ and \" respectively.

Strings are generally used for specifying file names. **master** makes no assumptions concerning the file name syntax of the host operating system. File name extensions (.ext) do not convey any meaning about the file content to **master**.

NOTE: All example pathlists in this manual use the CD-RTOS/Sun pathlist convention of separating pathlist elements with a slash (/). PC pathlists normally use the backslash (\) as a pathlist element separator. Because the backslash character is a reserved escape character, pathlists in PC master scripts must use the Sun conventions. Macintosh pathlists use the colon (:) separator. For example, the following full pathlist is shown in Sun, Macintosh and PC conventions:

/h0/video/image1	*Sun*
h0:video:image1	*Macintosh*
/h0/video/image1	*PC*

IFF IDs, Wrappers, & Chunks

CD-I IFF is a standard format for sources used in the CD-I production process. CD-I IFF is based on the Interchange File Format developed by Electronics Arts.

NOTE: This section provides a review of the basic concepts of IFF syntax. For a complete explication of the IFF standard, contact Electronics Arts.

IFF files are organized into *data chunks* and *wrapper chunks*. Data chunks are the basic data unit and consist of a chunk ID, a bytecount, and the chunk data itself (padded with a zero byte if the data size is odd). Wrapper chunks contain other chunks in the same fashion that directories contain files. A wrapper chunk consists of a wrapper ID and a chunk ID, followed by a sequence of the chunks it contains.

A data chunk specification consists of a chunk ID and an optional index. A wrapper chunk specification consists of a wrapper ID, a chunk ID, and an optional index. The index is used to individualize chunks with the same ID within a wrapper. The chunk index consists of a decimal number and defaults to 1 when not specified.

A chunkpath specifies an individual chunk within an IFF file. Chunkpaths are similar to pathlists; they specify an individual chunk by a hierarchical sequence of wrapper specifications ending with a chunk specification. Chunkpaths are used in combination with a standard pathlist to specify a chunk within an IFF file within the CD-I directory structure.

A chunk ID is any sequence of up to 4 characters in the graphic character set—from "!" (ASCII 33) to "~" (ASCII 126)—with the following exceptions:

- The # and > characters are reserved as chunkpath separators.

- While the space character is not allowed, IFF IDs shorter than 4 characters are internally padded with spaces to fill out a 4 character ID.
- A sequence of only digits is considered a number, not an ID.

A wrapper ID is either **FORM**, **CAT**, or **LIST**.

NOTE: Advanced IFF features such as property chunks (PROP) semantics are not supported.

The > character separates wrapper and chunk specifications from each other. The # character separates the individual elements of wrapper and chunk specifications. For example:

IHDR#4	*chunk specification with index*
FORM#IMAG#2	*wrapper specification with index*
FORM#IMAG#2>IHDR#4	*chunkpath*
CAT#IMAG>FORM#IMAG>IHDR#4	*chunkpath*

The following are chunkpaths used with pathlists:

"/home/VIDEO/image">CAT#IMAG>FOMR#IMAG>IHDR

"disk:[VIDEO]image">LIST#IMAG>FORM#IMAG#4

"C\\MUSIC\\sound">FORM#AIFF>COMM

Chunkpaths ending in a data chunk are called fully qualified chunkpaths. A fully qualified chunkpath may be optional or required, depending on context. In some instances—for example when only one specific wrapper type is allowed—the last chunkpath component may be omitted and will be automatically added. In such cases, specifying an IFF chunk may be as minimal as using a file name followed by a > character. For example:

"disk:[VIDEO]image">

Sector numbers

A sector number is required in many file definitions. It is expressed either as an absolute number of frames or as minutes, seconds, and frames components separated by colons:

frames	*(12)*
minutes: *seconds*: *frames*	*(0:0:12)*

where

minutes	are decimal numbers ranging from 0 to 99.
seconds	are decimal numbers ranging from 0 to 59.
frames	are decimal numbers ranging from 0 to 74 in the *minutes*:*seconds*:*frames* format and non-negative decimal numbers in the absolute *frames* format.

Numbers

There are two formats for numbers used in the **master** syntax: hexadecimal and decimal. Hexadecimal numbers are prefixed with 0x or 0X and consist of a sequence of hexadecimal digits (0-9, A-F, and a-f). Decimal numbers have no prefix and consist of a sequence of the digits 0-9.

Separators

There are 5 types of separators used in **master** syntax:

commas	,
colons	:
periods	.
brackets	[ ]
braces	{ }

These characters are always interpreted as separators, except when used within IFF IDs. Other single characters occurring in the script cause syntax errors.

Whitespace

master ignores all whitespace separating the definition elements. Consequently, whitespace is normally used to enhance readability by visually separating the various definitions, comments, identifiers, etc. Whitespace is considered any of the following:

space character	(ASCII 32)
horizontal tab	(ASCII 9)
line feed	(ASCII 10)
carriage return	(ASCII 13)
vertical tab	(ASCII 11)
form feed	(ASCII 12)

Comments

Comments are also used to enhance readability and ensure comprehension. Comments begin with an exclamation point (!) and end at the end of the line.

Script file syntax overview

The following sections describe the **master** script syntax in detail. The syntax sections are organized into four main sections: the album definition, the volume definition, the file definition, and the directory structure definition. The file definition section contains a subsection on each type of file supported by the **master** syntax.

- **red files**
  These are audio (CD-DA) files.
- **yellow files**
  These are data or font files. The yellow file definition includes the required green book files: **copyright, biblio, abstract**, and **application**.
- **simple green files**
  These are pre-mastered realtime files. They are created by using **master** to generate a realtime file instead of a disc image. Once a realtime file is created, **master** treats it as a simple file type.
- **realtime green files**
  These files consist of a series of records that can be made up of a number of assets. These assets may be interleaved or consecutively arranged. Full motion video records are specified using the realtime file definition.
- **application use files**
  An application use file is disc space reserved for use by the application use area of the volume descriptor.
- **message area files**
  For syntactic reasons, the message area source for the resulting volume must be declared as a file definition, although it will not appear as a file in any **CDFM** directory.

Script file syntax

The **master** script file uses the following syntax when building CD-I disc images:

*[**options:** option...] album_def volume_def file_def... directory_def [volume_def file_def... directory_def]...*

where

option is a **master** command line option.

album_def is the album definition element.

volume_def is a volume definition element.

file_def is a file definition element.

directory_def is a directory definition element.

master uses the following syntax when building single realtime files:

*[**options:** option] record...*

where

option is a **master** command line option. The command line options are defined in Chapter 1.

record is a record specification.

Each definition element may be separated by white space. The example scripts at the end of this chapter show common spacing conventions.

Script element conventions

The following sections define and explain each of the definition elements. In the syntax descriptions, the following basic script element definitions are used:

string	is a quoted alphanumeric string of up to 128 characters. For example: "/home/file"
ident	is an identifier.
chunkpath	is a CD-I IFF pathlist indicating an IFF chunk.
filename	is a quoted string that specifies a file. *filename* may be either a full pathlist or an individual file name.
rtl	is a realtime label.
rsn	is a relative sector number.
sn	is a sector number.

All of the above are defined in detail in the preceding section.

NOTE: A complete listing of the **master** *script syntax is included in Appendix A.*

Album definition

There are two types of album definitions: CD-I and CD-DA. A CD-I album definition (*album_def*) consists of an album name, and optional publisher and preparer strings. The CD-I album definition is used for the following CD formats: CD-I, CD-ROM XA, and CD-READY. A CD-DA album definition consists solely of a **red** album specification. In both cases, the album definition is followed by a sequence of volume definitions.

The syntax for a CD-I album definition is:

define album *album_name [label]*

where

album_name is a *string* specifying the album name.

label specifies the publisher and/or the preparer using the following syntax:

publisher *string*
preparer *string*

Both labels may be present in an album definition. Whichever labels are specified are included in each volume's label (in their Descriptor Records).

The syntax for a CD-DA album definition is:

red album

Album definition examples

define album "FMV" publisher "OptImage" preparer "OptImage"

define album "Corporate Showcase" publisher "OptImage"

red album

Volume definition

A CD-I volume definition *(volume_def)* consists of the volume name and the name of the output file containing the resulting disc image. The volume definition is followed by a sequence of file definitions and a single directory definition. The file definitions specify the type and structure of each file in the volume, while the directory definition specifies the hierarchical position, name, and attributes of each file in the directory structure of the volume.

The syntax for the volume definition is:

volume *vol_name* **in** *outfile*

where

vol_name is a *string* specifying the volume name.

outfile is a *filename* specifying the output file name (or pathlist) to contain the resulting disc image.

Volume definition examples

```
volume "FMV1" in "fmv_image"

volume "PhotoCD" in "photo_cd"

volume "Build Demo" in "cdimage_xa"
```

File definition

There are four types of CD-I file definition elements:

- **simple or non-realtime files**
  Simple files are data files such as fonts, executable programs, text files, or red (CD-DA) files that depend upon only one source. The **abstract**, **copyright**, **biblio**, and **application** files also fall into this category.
- **green or realtime files**
  Green files consist of a series of records that are made up of a number of streams (from different sources). These streams may be interleaved or consecutively arranged. A green file may be specified in a **master** script by itself or be specified as part of a disc image. Full motion video records are specified using the realtime file definition.

NOTE: Green files pre-mastered by the **green** *utility or an equivalent premastering program are interpreted by* **master** *as simple files.*

- **application use files**
  A data file (**application_use**) can be defined for use by the application use area of the volume descriptor.
- **message areas**
  For syntactic reasons, the message area source for the resulting volume must be declared as a file definition, although it will not appear as a file in any **CDFM** directory.

NOTE: Path table files are generated automatically by **master.**

A file is defined by a file-type, an identifier used as a file reference by the directory definition element, and the source (or source list) on which the file depends.

Simple file definition

Simple files are defined by the file-type specifier. While the definition syntax is generally the same for the different file types, there are some variations on how the syntax is implemented. The following discussion provides the overall syntax. Following this section, the specific file types are described.

NOTE: The **abstract, application, biblio,** *and* **copyright** *files are discussed with the yellow files.*

Simple file definitions use the following syntax:

green file *ident* **from** *filename*	*! pre-mastered realtime file*
red file *ident* **from** *source*	*! CD-DA file*
yellow file *ident* **from** *source*	*! data or font file*
abstract file *ident* **from** *filename*	*! abstract file*
application file *ident* **from** *filename*	*! application file*
biblio file *ident* **from** *filename*	*! bibliographic file*
copyright file *ident* **from** *filename*	*! copyright file*

where

ident is an identifier used to refer to the file in the CD-I directory structure.

filename is a quoted file name or pathlist specifying the file.

source specifies the file's single source using one of the following formats:

filename
filename>*chunkpath*

filename is a quoted file name or pathlist specifying the file.

chunkpath is a CD-I IFF pathlist specifying an IFF chunk.

Premastered realtime file definition

Premastered realtime file definitions use the following syntax:

green file *ident* **from** *filename* *! pre-mastered realtime file*

where

ident is an identifier used to refer to the realtime file in the CD-I directory structure.

filename is a quoted file name or pathlist specifying the file.

Premastered realtime file sectors are copied to the disc image with the following modifications:

- The header time fields are recomputed.
- The subheader submode end-of-file (EOF) bit is cleared in every sector except the last, in which it is set.

The byte length of a realtime file, as recorded into its directory record, is a multiple of 2048. Specifically, the length is 2048 times the number of CD-I sectors written.

Premastered green file definition examples

```
green file   G_File    from "green_file"
green file   gfile     from "stuff/green_file"
```

Red file definition

Red files are CD-DA files or tracks. Red file definitions use the following syntax:

red file *ident* **from** ***[swapped]*** *source* ***[with pre_emphasis]***

where

ident is an identifier used to refer to the file in the CD-I directory structure.

swapped forces **master** to swap bytes in 16-bit words of non-AIFF (raw) source files generated on Motorola-based machines (i.e., Macintosh, CD-I players, etc.). If you do not use the **swapped** specification with raw source files, the audio file does not play as desired.

with pre_emphasis
forces **master** to set the pre-emphasis flag that indicates that pre-emphasis filters should be used during play.

source specifies the file's single source. *source* is assumed to contain stereo PCM samples (low-order byte then high-order byte). *source* is specified using one of the following formats:

filename
filename>*chunkpath*

filename is a quoted file name or pathlist specifying the file.

chunkpath is a CD-I IFF pathlist specifying an IFF chunk.

A red source *chunkpath* must specify an **AIFF FORM** wrapper chunk. **master** processes a red source *chunkpath* by checking it for validity and correctness. If necessary, **master** completes the *chunkpath* by adding a final chunk element. Next, the **COMM** and **SSND** chunks are examined in the selected **FORM** wrapper. The **COMM** chunk

must describe a stereo PCM coding model. The **SSND** chunk data size and offset within the chunk are saved for use during disc image generation. Samples are swapped to convert them from IFF byte ordering to CD-DA byte ordering.

Red files are generated in their own track, preceded with an audio pause of at least 2 seconds up to the next second boundary and padded with PCM silence up to the next second boundary.

The byte length of a red file, as recorded into its directory record, is a multiple of 2048. Specifically, the length is 2048 times the number of CD-I sectors written.

Red file definition examples

```
red file    red1      from "/home/audio/pop60.c"
red file    redcd_da  from swapped "audio/pop60.c"
red file    redcd_da  from "/home/optima/wdm/2.cm">CAT#AIFF
red file    redcd_da  from "/home/wdm/2.cm"> with pre_emphasis
```

Yellow file definition

Yellow files are data files and fonts. Yellow files also include the **abstract, copyright, biblio,** and **application** files, which are introduced by their own keywords. Yellow file definitions use the following syntax:

yellow file *ident* **from** *source*

abstract file *ident* **from** *filename*

biblio file *ident* **from** *filename*

copyright file *ident* **from** *filename*

application file *ident* **from** *filename*

where

ident is an identifier used to refer to the yellow file in the CD-I directory structure.

source specifies the single yellow file source. *source* may be specified in two formats:

filename
filename>*chunkpath*

chunkpath is a CD-I IFF pathlist specifying an IFF chunk.

filename is a quoted file name or pathlist specifying the file.

A yellow source *chunkpath* can specify any chunk (data or wrapper, including the whole IFF file). **master** processes the *chunkpath* by analyzing and checking the IFF file chunk structure for validity. Next, **master** searches for the selected chunk. If master finds the chunk, master saves the chunk parameters (offset within IFF files and size) for use during disc image generation.

Yellow files are generated as DATA sectors in CD-I file number 1, channel 1, coding 0 with no trigger or end-of-record (EOR) submode bits. The end-of-file (EOF) submode bit is set in the last sector.

The byte length of a yellow file, as recorded into its directory record, is equal to the byte length of its source.

Yellow file definition examples

```
yellow file     pic1          from "VIDEO/pic1.dyuv"
copyright file  Copyright     from "txt/copyright.txt"
abstract file   Abstract      from "txt/ab.txt"
```

Realtime file definition

Realtime file definitions use the following syntax:

green file *ident* **from** *rec_def...*

where

ident is an identifier used to refer to the realtime file in the CD-I directory structure.

rec_def is information that defines the specific record. The information is specified in one of the following forms:

block *[rtl,...] stream_def...*

record *[rtl,...] stream_def...*

The only difference between the two types of record definitions (**block** and **record**) is the automatic insertion of an end-of-record (EOR) submode bit in the last sector of the record when using the **record** format.

An *rtl* is a realtime label identifying the subsequent stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files.

Record definitions

Record definitions use the following syntax:

block *[rtl,...] stream_def...*

record *[rtl,...] stream_def...*

where

rtl is a label identifying the record. More than one label may be given, but each label refers to the same record. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the record. *ident* must be unique within the script and conforms to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

stream_def is the information which defines the type and source of the actual record to be read. There are six types of stream definitions:

realtime audio
realtime [still] mpeg
audio
video
data
event

Each stream definition is type dependent. The following sections describe each stream type.

Realtime audio stream definitions

Realtime audio stream definitions use the following syntax:

real_time audio *[rtl,...]* **[at** *rsn]* **in channel** *channel*
from *a_source[, a_source...]*

where

rtl is a label identifying the stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the stream. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

rsn specifies the starting sector number of the first sector allocated for this stream.

channel specifies the channel number in which this stream is placed. *channel* must be in the range of 0-15.

a_source specifies an audio source to be included in the stream. The audio source specification is described on the following page.

Audio source specifications

Audio sources are specified using one of the following forms:

[rtl:...] asset [**mask** *pattern] [* **offset** *offset* **length** *length] [{***at** | **by***} rsn]*

or

*[***at** *rsn]* **silence** *sn [a_model] [{***at** | **by***} rsn]*

where

rtl: is a label identifying the asset. More than one label may be given, but each label refers to the same asset. Labels must be used in order to generate the PMM maps for realtime files. An asset-level *rtl* has the following syntax:

ident[(expression)]:

ident is the label name that refers to the asset. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

Asset labels must end in a colon.

asset is a string specifying the location of the source in one of the following forms:

pathlist[a_model]
pathlist>chunkpath

a_model indicates the type of audio and whether emphasis is used. *a_ model* has the following syntax:

a_type [a_modifier]

where

a_type := **mono_a**
| **mono_b**
| **mono_c**
| **stereo_a**
| **stereo_b**
| **stereo_c**

a_modifier := **emphasis**

chunkpath is a legal IFF pathlist as described at the beginning of this chapter.

pattern is a 16-bit quantity mapped to repeated 16 sector groups. Each bit is mapped one-to-one to a sector in the group. The least significant bit maps to the earliest sector in the group. Sectors mapped to zeros in the *pattern* are then filled with data from the *asset*.

NOTE: ***master*** *interleaves realtime audio assets without the* ***mask*** *pattern specification. It is possible to specify a mask pattern that contradicts (and overrides)* ***master****'s interleaving algorithm.*

---

offset specifies the byte offset from the beginning of the *source* at which to begin including data in the stream. If *offset* is greater than the length of the *asset*, an error is returned.

length specifies the byte length of the data to include in the stream. If the combined total of *offset* and *length* is greater than the length of the *asset*, a warning is returned and only the data from *offset* to the end of the *asset* is included.

at *rsn* specifies the sector number (relative to the beginning of the record) at which the first sector of the *source* begins.

by *rsn* specifies the sector number (relative to the beginning of the record) by which the last sector of the *source* is placed.

sn [*a_model*] specifies the number of sectors (*sn*)to be filled with silence at or by the specified *rsn*. The sectors are of the type specified by *a_model*.

If any audio source within a realtime audio stream uses an *rsn* clause, the first source in the stream must also use an *rsn* clause or an error is returned. However, if the first audio source uses an *rsn* clause, subsequent sources are not required to include *rsn* clauses and will be placed directly after the first source in the order they are specified. If subsequent audio sources use *rsn* clauses, **master** attempts to place the sources according to their *rsn* clauses. If a source's *rsn* clause specifies the beginning of the source before the end of the previous source, an error is returned.

If no time clauses are used, **master** places each source in consecutive order, with the first source placed directly after the previous stream.

Realtime MPEG stream definitions

Realtime MPEG stream definitions use the following syntax:

real_time *[***still***]* **mpeg** *[rtl,...] [***at** *rsn]* **in channel** *channel*
from *source[, source...]*

where

still specifies an MPEG stream containing only still video images.

rtl is a label identifying the stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the stream. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

rsn is the starting sector number of the first sector allocated for this stream.

channel specifies the channel number in which this stream is placed. *channel* must be in the range of 0-15.

source defines a source included in the stream. The MPEG source specification is described on the following pages.

MPEG source specifications

MPEG sources are specified using the following form:

[rtl:...] asset [**delta_file** *dfile]* **ep_list** *ep_spec [{***at** | **by***} rsn]*

where

rtl: is a label identifying the asset. More than one label may be given, but each label refers to the same asset. Labels must be used in order to generate the PMM maps for realtime files. An asset-level *rtl* has the following syntax:

ident[(expression)]:

ident is the label name that refers to the asset. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

Asset labels must end in a colon.

asset is a pathlist specifying the location of the source.

dfile specifies the deltafile which describes the sector layout of the MPEG data in the realtime MPEG source. If the **delta_file** specification is omitted, **master** assumes *dfile* has the name of *asset*.**dlt** and is in the same directory as *asset*. *dfile* is generated by the **pink** utility.

NOTE: The **pink** *utility multiplexes MPEG streams in order to ready them for* **master**.

---

ep_spec specifies the use of an entry point list file. Entry point specifications are described below.

at *rsn* specifies the sector number (relative to the beginning of the record) at which the first sector of the source begins.

by *rsn* specifies the sector number (relative to the beginning of the record) by which the last sector of the source is placed.

If no *rsn* is specified, the first source definition is assumed to be **at 0**. If a subsequent source specification also omits a time clause specification, it is placed after a two-second pause that follows the previous source in the record.

If an *rsn* is specified, it must allow for a two-second pause between MPEG sources.

Entry point list specifications

An entry point list specification (*ep_spec*) must be one of the following:

off

internal *[***and gap** *sn] [***with index***]* **from** *ep_infile*

*ident [***with index***]* **from** *ep_infile*

*ep_outfile [***with index***]* **from** *ep_infile*

where

off specifies that no entry point list is used with the MPEG source.

internal generates the entry point data at the beginning of the MPEG data.

and gap *sn* specifies the number of sectors between the end of the entry point data and the beginning of the MPEG data. This gap may not be filled with realtime MPEG data; however, non-MPEG data may be used to fill the gap.

with index generates an alphabetical index of entry point labels.

ep_infile specifies the name of the entry point list file generated by the **pink** utility for this MPEG source.

ident is an identifier for a yellow file in which **master** puts the entry point data. *ident* is only used when generating disc images.

ep_outfile is a quoted file name (pathlist) specifying the file in which **master** puts the entry point data. *ep_outfile* is used only when generating a realtime record.

The structure of entry point data lists is described in Appendix G.

Audio stream definitions

Audio stream definitions use the following syntax:

audio *[***packed***] [rtl,...] [***at** *rsn] **in channel** channel*
from *source[, source...]*

where

packed specifies that each specified source in the stream should begin in the first available byte. If **packed** is not specified, sources begin in the first available sector.

rtl is a label identifying the stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the stream. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

rsn specifies the starting sector number of the first sector allocated for this stream.

channel specifies the channel number in which this stream is placed. *channel* must be in the range of 0-15.

source defines an audio source to be included in the stream. The audio source specification is described on the following pages.

Audio source specifications

Audio sources are specified using one of the following forms:

[rtl:...] asset [**mask** *pattern] [* **offset** *offset* **length** *length] [{***at** | **by***} rsn]*

or

[at *rsn]* **silence** *sn* **[***model***]** *[{***at** | **by***} rsn]*

where

rtl: is a label identifying the asset. More than one label may be given, but each label refers to the same asset. Labels must be used in order to generate the PMM maps for realtime files. An asset-level *rtl* has the following syntax:

ident[(expression)]:

ident is the label name that refers to the asset. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

Asset labels must end in a colon.

asset is a string specifying the location of the source in one of the following forms:

pathlist[a_model]
pathlist>chunkpath

a_model indicates the type of audio and whether emphasis is used. *a_ model* is specified using the following syntax:

a_type [a_modifier]

where

a_type := **mono_a**
| **mono_b**
| **mono_c**
| **stereo_a**
| **stereo_b**
| **stereo_c**

a_modifier := **emphasis**

chunkpath is a legal IFF pathlist as described at the beginning of this chapter.

pattern is a 16-bit quantity mapped to repeated 16 sector groups. Each bit is mapped one-to-one to a sector in the group. The least significant bit maps to the earliest sector in the group. Sectors mapped to zeros in the *pattern* are then filled with data from the *asset*.

offset specifies the byte offset from the beginning of the source at which to begin including data in the stream. If *offset* is greater than the length of the *asset*, an error is returned.

length specifies the byte length of the data to include in the stream. If the combined total of *offset* and *length* is greater than the length of the *asset*, a warning is returned and only the data from *offset* to the end of the *asset* is included.

at *rsn* specifies the sector number (relative to the beginning of the record) at which the first sector of the source begins.

by *rsn* specifies the sector number (relative to the beginning of the record) by which the last sector of the source is placed.

sn [*a_model*] specifies the number of sectors (*sn*)to be filled with silence at or by the specified *rsn*. The sectors are of the type specified by *a_model*.

If any audio source within a realtime stream uses an *rsn* clause, the first source in the stream must also use an *rsn* clause or an error is returned. However, if the first audio source uses an *rsn* clause, subsequent sources are not required to include *rsn* clauses and will be placed directly after the first source in the order they are specified. If subsequent audio sources use *rsn* clauses, **master** attempts to place the sources according to their *rsn* clauses. If a source's *rsn* clause specifies the beginning of the source before the end of the previous source, an error is returned.

If no time clauses are used, **master** places each source in consecutive order, with the first source placed directly after the previous stream.

Video stream definitions

Video stream definitions use the following syntax:

video *[***packed***] [rtl,...] [***at** *rsn]* **in channel** *channel*
from *source[, source...]*

where

packed specifies that each source in the stream should begin in the first available byte. If **packed** is not specified, sources begin in the first available sector.

rtl is a label identifying the stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the stream. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

rsn specifies the starting sector number of the first sector allocated for this stream.

channel specifies the channel number in which this stream is placed. *channel* must be in the range of 0-31.

source defines a source included in the stream. The video source specification is described on the following pages.

Video source specifications

Video sources are specified using the following form:

[rtl:...] asset [**mask** *pattern] [* **offset** *offset* **length** *length] [{***at** | **by***} rsn]*

where

rtl: is a label identifying the asset. More than one label may be given, but each label refers to the same asset. Labels must be used in order to generate the PMM maps for realtime files. An asset-level *rtl* has the following syntax:

ident[(expression)]:

ident is the label name that refers to the asset. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

Asset labels must end in a colon.

asset is a string specifying the location of the source in one of the following forms:

pathlist[v_model]
pathlist>chunkpath

v_model indicates the type of video using the following syntax:

v_type [v_modifier]

where

v_type := **clut4**
| **clut7**

	|	**clut8**
	|	**rl3**
	|	**rl7**
	|	**dyuv**
	|	**rgb555l**
	|	**rgb555u**
	|	**rgb555**
	|	**qhy**
	|	**specific** *mask*
mask	:=	*decimal_number*
	|	*hex_number*
v_modifier	:=	**normal**
	|	**high**
	|	**double**
	|	**even**
	|	**odd**

chunkpath is a legal IFF pathlist as described at the beginning of this chapter.

pattern is a 16-bit quantity mapped to repeated 16 sector groups. Each bit is mapped one-to-one to a sector in the group. The least significant bit maps to the earliest sector in the group. Sectors mapped to zeros in the *pattern* are then filled with data from the *asset*.

offset specifies the byte offset from the beginning of the source at which to begin including data in the stream. If *offset* is greater than the length of the *asset*, an error is returned.

length specifies the byte length of the data to include in the stream. If the combined total of *offset* and *length* is greater than the length of the *asset*, a warning is returned and only the data from *offset* to the end of the *asset* is included.

at *rsn* specifies the sector number (relative to the beginning of the record) at which the first sector of the source begins.

by *rsn* specifies the sector number (relative to the beginning of the record) by which the last sector of the source is placed.

If no *rsn* is specified, the source definition is assumed to be **at 0**; the source is placed directly after any previous source in the record.

Data stream definitions

Data streams are defined by a list of source files to include in a specific channel. Data stream definitions use the following syntax:

data *[***packed***] [rtl,...] [***at** *rsn]* **in channel** *channel*
from *source[, source...]*

where

packed specifies that each source in the stream should begin in the first available byte. If **packed** is not specified, sources begin in the first available sector.

rtl is a label identifying the stream. More than one label may be given, but each label refers to the same stream. Labels must be used in order to generate the PMM maps for realtime files. An *rtl* can be specified using the following syntax:

ident[(expression)]

ident is the label name that refers to the stream. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

rsn is the starting sector number of the first sector allocated for this stream.

channel specifies the channel number in which this stream is placed. *channel* must be in the range of 0-15.

source defines a source included in the stream. The data source specification is described on the following pages.

Data source specifications

Data sources are specified using the following form:

[rtl:...] asset [**mask** *pattern*] [**offset** *offset* **length** *length*] [{**at** | **by**} *rsn*]

where

rtl: is a label identifying the asset. More than one label may be given, but each label refers to the same asset. Labels must be used in order to generate the PMM maps for realtime files. An asset-level *rtl* has the following syntax:

ident[(*expression*)]:

ident is the label name that refers to the asset. *ident* must be unique within the script and conform to the naming conventions of identifiers specified at the beginning of this chapter.

expression is the optional value of the label. *expression* can be any constant or equation that results in an integral number. If *expression* is not given, the label value defaults to 0.

Asset labels must end in a colon.

asset is a string specifying the location of the source in one of the following forms:

pathlist
pathlist>*chunkpath*

chunkpath is a legal IFF pathlist as described at the beginning of this chapter.

pattern is a 16-bit quantity mapped to repeated 16 sector groups. Each bit is mapped one-to-one to a sector in the group. The least significant bit maps to the earliest sector in the group. Sectors mapped to zeros in the *pattern* are then filled with data from the *asset*.

offset specifies the byte offset from the beginning of the source at which to begin including data in the stream. If *offset* is greater than the length of the *asset*, an error is returned.

length specifies the byte length of the data to include in the stream. If the combined total of *offset* and *length* is greater than the length of the *asset*, a warning is returned and only the data from *offset* to the end of the *asset* is included.

at *rsn* specifies the sector number (relative to the beginning of the record) at which the first sector of the source begins.

by *rsn* specifies the sector number (relative to the beginning of the record) by which the last sector of the source is placed.

If no *rsn* is specified, the source definition is assumed to be **at 0**; the source is placed directly after any previous source in the record.

Event stream definitions

Event streams are used to specify trigger or end-of-record (EOR) bits in the realtime file. Events may be defined at specific or periodic positions. More than one event specification may be defined in an event stream definition by separating the event specifications with a comma.

Event stream definitions use the following syntax:

event **at** *rsn* **[every** *sn*][, **at** *rsn* **[every** *sn*]...]

where

event specifies the type of bit set in the specified sector(s) using the following key words:

triggers	sets a trigger bit
eors	sets an EOR bit

at *rsn* specifies the sector in which to set a trigger or EOR bit. *rsn* may be specified in the following formats:

frames	*(12)*
minutes: seconds: frames	*(0:0:12)*
ref_ident.**first**	*(rec1.first)*
ref_ident.**last**	

where

ref_ident is a realtime label previously defined in the script.

every *sn* specifies the sector interval for setting periodic events beginning with the sector specified by *rsn*.

Application_use area definition

Application_use areas are data source files used in the application use area of the volume descriptor. **Application_use** area definitions use the following syntax:

application_use from *filename* *! application_use area*

where

filename is a quoted file name or pathlist specifying the file.

Application_use area definition examples

application_use **from "/home/cdi_apps/app_use"**

application_use **from "/home/cdi_apps/app1/defaults"**

Message area definition

Message areas are PCM source files used as messages for the CD-I title user. Message area definitions use the following syntax:

message from *[**swapped**] source* *! message area*

where

swapped forces **master** to swap bytes in 16-bit words of non-AIFF (raw) source files generated on Motorola-based machines (i.e., Macintosh, CD-I players, etc.). If you do not use the **swapped** specification with raw source files, the audio file does not play as desired.

source specifies the file's single source. *source* is assumed to contain stereo PCM samples (low-order byte then high-order byte). *source* is specified using one of the following formats:

filename
filename>chunkpath

filename is a quoted file name or pathlist specifying the file.

chunkpath is a CD-I IFF pathlist specifying an IFF chunk.

A message source *chunkpath* must specify an **AIFF FORM** wrapper chunk. **master** processes a message source *chunkpath* by checking it for validity and correctness. If necessary, **master** completes the *chunkpath* by adding a final chunk element. Next, the **COMM** and **SSND** chunks are examined in the selected **FORM** wrapper. The **COMM** chunk must describe a stereo PCM coding model. The **SSND** chunk data size and offset within the chunk are saved for use during disc image generation. Samples are swapped to convert them from IFF byte ordering to CD-DA byte ordering.

Message areas are always generated at the start of a disc image, and at the end of a realtime file if red tracks were included within

the realtime file. **message** areas are padded with PCM silence up to 2250 sectors (or 30 seconds) if the source is not long enough.

Message area definition examples

```
message                    from "/home/audio/message.pcm"
red file      redcd_da     from "/home/optima/wdm/2.cm">CAT#AIFF
```

Directory definition

A CD-I directory definition (*directory_def*) specifies the hierarchical file layout of the resulting disc image. Starting from the root directory, the directory definition specifies the names and attributes for each file and directory of the disc.

The definition specifies the owner, protection, and hidden file attributes for each file. These attributes are optional and default to owner 0.0, protection 0 x 555 (read and execute access for all user categories), and not hidden.

master checks that names given to CD-I files are legal CD-RTOS file names, containing only alphanumeric, $, or . characters with at least one alphanumeric. Their length can not exceed 28 characters, and file names must be unique within their directory.

Files are placed in their directory as they occur in the source script file without any reordering. By default, files are generated before subdirectories using the file order specified in the directory definition. If the **-e** command line option is specified, however, subdirectories and their contents are generated as they are encountered.

CD-ROM XA filenames

For CD-ROM XA files, the file names are checked for compliance with the CD-ROM XA standard. Legal CD-ROM XA file names contain only upper-case characters and the numerals 0 to 9. The filenames have the following format:

name.extension;version

where

name	is a 0–8 character string. *name* cannot begin with 0–9.
extension	is a 0–3 character string.
version	is a version number within the range 1–32767.

The following are legal CD-ROM XA file names:

FILE1.EXT;3200

FILE1.;3201

For CD-ROM XA disc images, **master** places all files within a directory in alphabetical order in the disc image.

Directory definition Syntax

The syntax for a directory definition is:

{[file_entry...] [dir_name {file_entry...}]}

where

file_entry specifies the file name, file attributes, and the identifier associated with the file. *file_entry* has the following format:

file_name [attribute...] **from** *ident*

file_name is a legal file name string.

attribute is one of the following:

owner *group.user*
protection *permissions*
hidden
pad *sectors*

group.user is a CD-RTOS group.user specification (for example, **0.0**).

permissions is a three digit hexadecimal number specifying the CD-RTOS file access permissions (for example, **0x555**).

hidden defines the file as a hidden file.

pad *sectors* defines the number of sectors with which to pad the file.

ident is the identifier used in the file definition.

dir_name specifies a legal directory name.

Example directory structure definition

The following example specifies the root directory and two subdirectories (**CMDS** and **RTF**):

```
{
   "copyright" protection 0x111 from Copyright
   "abstract" protection 0x111 from Abstract
   "bibliographic" protection 0x111 from Biblio
   "CMDS"
   {
      "play_demo" from appl
   }
   "RTF"
   {
      "demo.rtf" from demo
   }
}
```

Example scripts

There are ten scripts examined in this section:

- a realtime file using audio and DYUV video
- a CD-I image with a realtime file using audio and DYUV video
- a CD-I image with a pre-mastered realtime file
- a CD-DA disc image
- a CD-I image with CD-DA tracks
- a CD-I image with no realtime files
- a CD-I image with a realtime file using audio, DYUV video, and EORs
- a CD-I image with a realtime file using interleaved audio, DYUV video, and EORs
- a CD-I image with FMV assets
- a CD-I image with FMV assets using an entry point specification

---

NOTE: All example pathlists in this manual use the CD-RTOS/Sun pathlist convention of separating pathlist elements with a slash (/). PC pathlists normally use the backslash (\) as a pathlist element separator. Because the backslash character is a reserved escape character, pathlists in PC ***master*** *scripts must use the Sun conventsions. Macintosh pathlists use the colon (:) separator. For example, the following full pathlist is shown in Sun, Macintosh and PC conventions:*

/h0/video/image1	*Sun*
h0:video:image1	*Macintosh*
/h0/video/image1	*PC*

Example 1: Realtime file

This script builds a single realtime file from two records. The first record consists of a single video asset. The second record consists of a realtime audio asset and a video asset built from 10 images. This realtime file is used in the demo disc image supplied with this disc building software package.

```
! master realtime file scripts contain only record definitions
!
! MASTER OPTIONS
options: "demo.grn"
!
! RECORD DEFINITIONS
record
   video in channel 0 from
      "/emu/Resources/VIDEO/group.d">CAT#IMAG

record
   real_time audio in channel 0 from
      "/emu/Resources/AUDIO/pop60.c">CAT#AIFF

   video in channel 0 from
      "/emu/Resources/VIDEO/bob.d">CAT#IMAG at 00:10:00,
      "/emu/Resources/VIDEO/steve.d">CAT#IMAG at 00:15:00,
      "/emu/Resources/VIDEO/pam.d">CAT#IMAG at 00:20:00,
      "/emu/Resources/VIDEO/randy.d">CAT#IMAG at 00:25:00,
      "/emu/Resources/VIDEO/tom.d">CAT#IMAG at 00:30:00,
      "/emu/Resources/VIDEO/vicki.d">CAT#IMAG at 00:35:00,
      "/emu/Resources/VIDEO/sean.d">CAT#IMAG at 00:40:00,
      "/emu/Resources/VIDEO/ken.d">CAT#IMAG at 00:45:00,
      "/emu/Resources/VIDEO/john.d">CAT#IMAG at 00:50:00,
      "/emu/Resources/VIDEO/group.d">CAT#IMAG at 00:55:00
```

Example 2: CD-I image with realtime file

This script builds a disc image from the realtime file shown in the first example script. This master script builds the demo disc image supplied with this disc building software package.

```
! master disc image scripts contain 4 elements: an album definition, a
! volume definition, file definitions, and a directory structure definition
!
! MASTER OPTIONS
options: -t="demo.cdi.toc"
!
! ALBUM DEFINITION
define album        "Disc Building Demo"
publisher           "OptImage/Des Moines"
preparer            "Patrick A. Murphy"
!
! VOLUME DEFINITION
volume "Build Demo" in "demo.cdi.cd"
!
! REQUIRED FILE DEFINITIONS
message                          from  "/emu/Resources/AUDIO/message.cda"
copyright file      Copyright    from  "/emu/Resources/TXT/copyright.txt"
abstract file       Abstract     from  "/emu/Resources/TXT/abstract.txt"
biblio file         Biblio       from  "/emu/Resources/TXT/bibliographic.txt"
application file    appl         from  "/emu/Resources/CMDS/play_demo"
!
! REALTIME FILE DEFINITION
green file demo from
   record
      video in channel 0 from
         "/emu/Resources/VIDEO/group.d">CAT#IMAG

   record
      real_time audio in channel 0 from
         "/emu/Resources/AUDIO/pop60.c">CAT#AIFF
      video in channel 0 from
         "/emu/Resources/VIDEO/bob.d">CAT#IMAG at 00:10:00,
         "/emu/Resources/VIDEO/steve.d">CAT#IMAG at 00:15:00,
         "/emu/Resources/VIDEO/pam.d">CAT#IMAG at 00:20:00,
         "/emu/Resources/VIDEO/randy.d">CAT#IMAG at 00:25:00,
         "/emu/Resources/VIDEO/tom.d">CAT#IMAG at 00:30:00,
         "/emu/Resources/VIDEO/vicki.d">CAT#IMAG at 00:35:00,
         "/emu/Resources/VIDEO/sean.d">CAT#IMAG at 00:40:00,
         "/emu/Resources/VIDEO/ken.d">CAT#IMAG at 00:45:00,
```

```
        "/emu/Resources/VIDEO/john.d">CAT#IMAG at 00:50:00,
        "/emu/Resources/VIDEO/group.d">CAT#IMAG at 00:55:00
!
! DIRECTORY STRUCTURE DEFINITION
{
   "copyright"      protection 0x111 from Copyright
   "abstract"       protection 0x111 from Abstract
   "bibliographic"  protection 0x111 from Biblio
   "CMDS"
   {
      "play_demo"   from appl
   }
   "RTF"
   {
      "demo.rtf"    from demo
   }
}
!
! >>> End of master Script <<<
```

Example 3: CD-I image with premastered realtime file

This script builds the same disc image as the previous example script using the premastered realtime file from the first example.

```
! master disc image scripts contain 4 elements: an album definition, a
! volume definition, file definitions, and a directory structure definition
!
! ALBUM DEFINITION
define album        "Disc Building Demo"
publisher           "OptImage/Des Moines"
preparer            "Patrick A. Murphy"
!
! VOLUME DEFINITION
volume "Build Demo" in "demo.cdi"
!
! REQUIRED FILE DEFINITIONS
message                          from "/emu/Resources/AUDIO/message.cda"
copyright file      Copyright    from "/emu/Resources/TXT/copyright.txt"
abstract file       Abstract     from "/emu/Resources/TXT/abstract.txt"
biblio file         Biblio       from "/emu/Resources/TXT/biblio.txt"
application file    appl         from "/emu/Resources/CMDS/play_demo"
!
! PREMASTERED REALTIME FILE DEFINITION
green file          demo         from "./demo.grn"
!
! DIRECTORY STRUCTURE DEFINITION
{
   "copyright"      protection 0x111 from Copyright
   "abstract"       protection 0x111 from Abstract
   "bibliographic"  protection 0x111 from Biblio
   "CMDS"
   {
      "play_demo"   from appl
   }
   "RTF"
   {
      "demo.rtf"    from demo
   }
}
!
! >>> End of master Script <<<
```

Example 4: CD-DA disc image

This script builds a CD-DA disc image containing four red tracks. Note the following differences between a CD-DA disc image script and a CD-I disc image script:

- The album definition uses the **red album** syntax.
- There are no required files (i.e., **copyright, message, abstract, biblio,** and **application**).
- The directory structure definition is empty.

```
! CD-DA disc image scripts contain only a red album definition, a volume
! definition, red track file definitions, and an empty directory structure
! definition.
!
! MASTER OPTIONS
options: -t="demo.red.toc" -v
! ALBUM DEFINITION
red album
!
! VOLUME DEFINITION
volume in "demo.red.cd"
!
! RED FILE DEFINITIONS
red file            ready00    from "AUDIO/track1.p4s">
red file            ready01    from "AUDIO/track2.raw"
red file            ready02    from "AUDIO/track3.p4s">
red file            ready03    from "AUDIO/track4.p4s">
!
! EMPTY DIRECTORY STRUCTURE DEFINITION
{}
!
! >>> End of master Script <<<
```

Example 5: CD-I image with CD-DA tracks

This script builds a CD-I disc image containing one realtime file and three CD-DA files.

NOTE: This script can also be used to build a CD-I Ready disc image by using the -u command line option.

```
! master disc image scripts contain 4 elements: an album definition, a
! volume definition, file definitions, and a directory structure definition
!
! MASTER OPTIONS
options: -t="emu/demo/demo.rdy.toc"
! ALBUM DEFINITION
define album        "Disc Building Demo"
publisher           "OptImage/Des Moines"
preparer            "Patrick A. Murphy"
!
! VOLUME DEFINITION
volume "Build Demo" in "demo.rdy"
!
! REQUIRED FILE DEFINITIONS
message                          from  "/emu/Resources/AUDIO/message.cda"
copyright file      Copyright    from  "/emu/Resources/TXT/copyright.txt"
abstract file       Abstract     from  "/emu/Resources/TXT/abstract.txt"
biblio file         Biblio       from  "/emu/Resources/TXT/biblio.txt"
application file    appl         from  "/emu/Resources/CMDS/play_demo"
!
! REALTIME FILE DEFINITION
green file demo from
   record
      video in channel 0 from
         "/emu/Resources/VIDEO/group.d">CAT#IMAG
   record
      real_time audio in channel 0 from
         "/emu/Resources/AUDIO/pop60.c">CAT#AIFF
      video in channel 0 from
         "/emu/Resources/VIDEO/bob.d">CAT#IMAG at 00:10:00,
         "/emu/Resources/VIDEO/steve.d">CAT#IMAG at 00:15:00,
         "/emu/Resources/VIDEO/pam.d">CAT#IMAG at 00:20:00,
         "/emu/Resources/VIDEO/randy.d">CAT#IMAG at 00:25:00,
         "/emu/Resources/VIDEO/tom.d">CAT#IMAG at 00:30:00,
         "/emu/Resources/VIDEO/vicki.d">CAT#IMAG at 00:35:00,
         "/emu/Resources/VIDEO/sean.d">CAT#IMAG at 00:40:00,
         "/emu/Resources/VIDEO/ken.d">CAT#IMAG at 00:45:00,
```

```
        "/emu/Resources/VIDEO/john.d">CAT#IMAG at 00:50:00,
        "/emu/Resources/VIDEO/group.d">CAT#IMAG at 00:55:00
!
! RED FILE DEFINITIONS (CD-DA FILES)
red file            Red_01      from "/emu/Resources/AUDIO/track1.p4s"
red file            Red_02      from "/emu/Resources/AUDIO/track2.p4s">
red file            Red_03      from "/emu/Resources/AUDIO/track3.p4s">
!
! DIRECTORY STRUCTURE DEFINITION
{
   "copyright"      protection 0x111 from Copyright
   "abstract"       protection 0x111 from Abstract
   "bibliographic"  protection 0x111 from Biblio
   "CMDS"
   {
      "play_demo"   from appl
   }
   "RTF"
   {
      "demo.rtf"    from demo
   }
   "RED"
   {
      "KMusic.raw"  from Red_01
      "ABAND.p4s"   from Red_02
      "BKMAGIC.p4s" from Red_03
   }
}
!
! >>> End of master Script <<<
```

Example 6: CD-I image with no realtime files

This script builds a CD-I disc image containing two yellow files. The **message.cda** file is an IFF file with 16 bit 44.1 kHz audio that is to be heard when the CD-I disc is inserted into an audio CD-DA player, the **cdi_application** file is the OS9 executable CD-I application boot program, and the **copyright.txt, abstract.txt,** and **bibliographic.txt** are text files.

It is recommended to place the **abstract, copyright,** and **biblio** files in the disc image at the root level of the compact disc file system; the location of the application boot program isn't necessarily at the root level. The compact disc file name of the application boot program begins with **cdi_** to conform to the Green Book recommendation that the module names of all OS9 modules (e.g., executable programs) begin with **cdi_**. Everything in this example is required by **master** except for the **VIDEO** subdirectory and its contents.

Following this script is the **master** output showing the disc image directory structure.

```
! ALBUM DEFINITION
define album        "Album Name"
publisher           "I. M. Publisher"
preparer            "M. Preparer"
!
! VOLUME DEFINITION
volume "Volume Name" in "./cdi_discimage.cd"
!
! REQUIRED FILE DESCRIPTIONS
application file    cdi_app     from "cdi_application"
message                         from "message.cda"
copyright file      Copyright   from "copyright.txt"
abstract file       Abstract    from "abstract.txt"
biblio file         Biblio      from "bibliographic.txt"
!
! YELLOW FILE DESCRIPTIONS
yellow file         image0_cl8 from "image0.cl8"
yellow file         image1_cl8 from "image1.cl8"
!
! DIRECTORY STRUCTURE DEFINITION
{
   "abstract"       protection 0x555 from Abstract
   "bibliographic"  protection 0x555 from Biblio
   "copyright"      protection 0x555 from Copyright
   "cdi_app"        protection 0x555 from cdi_app
   "VIDEO"          ! subdirectory
   {
      "image0.cl8"  protection 0x555 from image0_cl8
```

```
      "image1.cl8"  protection 0x555 from image1_cl8
   } ! end of VIDEO subdirectory
}
```

master generates the following output for the preceding script:

```
Start generation of album "Album Name"
Start generation of volume "Volume Name" [about 00:34]
Start generation of MESSAGE area
Start generation of ABSTRACT file abstract
Start generation of BIBLIOGRAPHIC file bibliographic
Start generation of COPYRIGHT file copyright
Start generation of APPLICATION file cdi_app
Start generation of YELLOW file VIDEO/image0.cl8
Start generation of YELLOW file VIDEO/image1.cl8
Start generation of DIRECTORY /
Start generation of DIRECTORY VIDEO
Start generation of PATH table file
Start generation of volume "Volume Name"'s label
End   generation of volume "Volume Name" [00:34 - space efficiency 99%]

                Directory of

 Owner    Last modified Attributes Sector Bytecount Name
-------   ------------- ---------- ------ --------- ----
  0.0     93/01/08 1543 -e-re-re-r    8de         2 abstract
  0.0     93/01/08 1543 -e-re-re-r    8df         2 bibliographic
  0.0     93/01/08 1543 -e-re-re-r    8e0         2 copyright
  0.0     93/02/26 1500 -e-re-re-r    8e1     14564 cdi_app
  0.0     93/02/26 1501 d--r--r--r    8e9       196 VIDEO
  0.0     93/02/26 1501 ---r--r--r    8dc        24 path_tbl

                Directory of VIDEO

 Owner    Last modified Attributes Sector Bytecount Name
-------   ------------- ---------- ------ --------- ----
  0.0     93/01/11 1134 -e-re-re-r    8ea    108384 image0.cl8
  0.0     93/01/11 1140 -e-re-re-r    91f    108384 image1.cl8

End   generation of album "Album Name"
```

Example 7: CD-I image with realtime file

This script builds a CD-I disc image containing one realtime file in the **RTF** subdirectory. Every sector in the realtime file has the realtime and form bits set in the submode byte of its subheader. The first 47 sectors are sectors with the video type bit set in their submode bytes and comprise the data chunk of the IFF image file **/home/optima/dsl/1.d.**

NOTE: The information contained in other IFF chunks is NOT included in the realtime file, and consequently would have to be stored elsewhere. For example, the image dimensions and YUV-start values for these DYUV images could be hard-coded into the application program which plays this realtime file. This illustrates the general principle that the structure of a realtime file and the logic of the application which plays it are closely related.

The last realtime file video sector has its end-of-record (EOR) bit set in its submode byte because the keyword **record** is used instead of **block**. The 48th sector has the audio bit set in its subheader and is the first sector from the data chunk of the 141-second IFF audio file **/home/optima/dsl/1.cm**. Thereafter, every 16th sector of the realtime file is taken successively from this audio file (i.e., the 64th sector, 80th, etc.) until the audio data chunk from the audio file is exhausted. The last audio sector has its EOR bit set.

At 5 seconds after the start of the audio, 47 sectors from the data chunk of the IFF video file **/home/optima/dsl/2.d** are backfilled as specified by the time code specification **by 00:05:00**. These 47 sectors are interleaved with the audio sectors from **/home/optima/dsl/1.cm** which are placed every 16 sectors. The last sector of the interleaved video sectors has the trigger bit set in its submode byte. The trigger bit is used to signal the application program that all the video data has been loaded into memory. The sectors from the video file **/home/optima/dsl/3.d** are similarly placed.

Because the structure of a realtime file and the logic of the application that plays it are integrally related, the logic of the program that could play this realtime file is worth discussing. Before calling **ss_play()** for this realtime file, a play control block (pcb) is created with:

- its **PCB_Chan** field set to 0x3 (since data from channels 0 and 1 are to be processed)
- its **AChan** field set to 0x1 (since channel 0 contains real-time audio)
- its video play-control-list (pcl) array element indexed by channel 1 and set to the address of the memory buffer where the data from **/home/optima/dsl/1.d** will be loaded

The **PCB_Sig** field contains a signal number handled by the application's signal handler function, which is invoked whenever a sector is read with the EOR or trigger bits set. The

PCB_Rec field is set to 1 so that the play initiated by **ss_play()** stops when one record has been read (i.e., when the last sector of **/home/optima/dsl/1.d** has been read).

When the application's signal handler function is called, the application displays this image (assuming that the information about the image dimensions and YUV-start values is already available, e.g., hard-coded into the application program itself). The video pcl array element indexed by 1 is then set to the address of the memory buffer where the data from **/home/optima/dsl/2.d** is to be loaded, the **PCB_Rec** field is again set to 1, and at some later time, another **ss_play()** call initiates another play of the disc, when audio will then begin to sound.

The next time the application's signal handler is called, the last sector from **/home/optima/dsl/2.d** is read (since its trigger bit is set), 5 seconds after audio begins to sound. The application displays this image (again assuming that all necessary display control information is available), and sets the video pcl array element indexed by 1 to the address of the buffer where the data from **/home/optima/dsl/3.d** is to be loaded. Next, the signal handler is invoked when the last sector of **/home/optima/dsl/3.d** is read (10 seconds after audio begins to sound). The application will then (or at some later time) display this image. The signal handler is invoked a final time when the last sector of the realtime file is processed because that sector has its EOR bit set.

In the above application example, the first picture (**/home/optima/dsl/1.d**) is displayed and then the audio and other pictures are begun at some later time with a second **ss_play()** call. If the **PCB_Rec** field were set to 2 before the first **ss_play()**, then 2 records would be played (i.e., the disc would play until 2 sectors with EOR bits are processed), and there would be no need for a second **ss_play()** call (and no delay from stopping and restarting the play of the disc); the entire realtime file would be played and the video buffer logic would be handled as above.

```
! ALBUM DEFINITION
define album        "Album Name"
publisher           "I. M. Publisher"
preparer            "M. Preparer"
!
! VOLUME DEFINITION
volume "Volume Name" in "cdi_discimage.cd"
!
! REQUIRED FILE DEFINITIONS
application file    cdi_app     from "cdi_application"
message             from
                    "/home/MM_1.1/CDI/DISC/AUDIO/MESSAGES/message.long.iff"
```

```
copyright file      Copyright  from "copyright.txt"
abstract  file      Abstract   from "abstract.txt"
biblio    file      Biblio     from "bibliographic.txt"
!
! YELLOW FILE DEFINITIONS
yellow file         image0_cl8 from "image0.cl8"
yellow file         image1_cl8 from "image1.cl8"
!
! REALTIME FILE DEFINITION
green file rtf1 from
   record
      video in channel 1 from
         "/home/optima/dsl/1.d">CAT#IMAG
   record
      real_time audio in channel 0 from
         "/home/optima/dsl/1.cm">CAT#AIFF
      video in channel 1 from
         "/home/optima/dsl/2.d">CAT#IMAG by 00:05:00,
         "/home/optima/dsl/3.d">CAT#IMAG by 00:10:00
      triggers at 00:05:00, 00:10:00
!
! DIRECTORY STRUCTURE DEFINITION
{
   "abstract"       protection 0x555 from Abstract
   "bibliographic"  protection 0x555 from Biblio
   "copyright"      protection 0x555 from Copyright
   "cdi_app"        protection 0x555 from cdi_app
   "RTF" ! subdirectory
   {
      "rtf1.rtf"    from rtf1
   }                           ! end of RTF subdirectory
   "VIDEO"                     ! subdirectory
  {
      "image0.cl8"  protection 0x555 from image0_cl8
      "image1.cl8"  protection 0x555 from image1_cl8
  }                            ! end of VIDEO subdirectory
}
```

Example 8: CD-I disc image with interleaved audio

This script expands the previous example by interleaving two realtime audio tracks within the realtime file, and interleaving the IFF **IHDR** and **PLTE** chunks within the realtime file. Interleaving the audio tracks allows you to select them at runtime. The IFF chunks contain display control information such as image dimensions, YUV-start values, and CLUT palette. Consequently, this information doesn't need to be hard-coded into the application program as in the last example.

The first sector in the realtime file is a data sector with the data type bit set in its submode byte and neither the form bit nor realtime bit set. This sector contains the IFF **IHDR** structure (14 bytes) and **PLTE** information (388 bytes = 4 bytes for **IFF_PLTE** structure plus 384 bytes of RGB palette information) for the file **/home/optima/dsl/1.cl7** (which has **plte_offset**=0 and **plte_count**=128). Following the data sector are 47 sectors from the data chunk of this file, with the last sector having its end-of-record (EOR) bit set.

Next, the interleaved realtime audio tracks are laid out, and the video sectors and display control data sectors from the other image files are backfilled from 5 seconds and 10 seconds after the start of the audio tracks.

Following this script is the tracks report and sector map report generated by ***master***.

```
! MASTER OPTIONS
options: -m="cdi_discimage.map" -t="cdi_discimage.toc"
!
! ALBUM DEFINITION
define album        "Album Name"
publisher           "I. M. Publisher"
preparer            "M. Preparer"
!
! VOLUME DEFINITION
volume "Volume Name" in "cdi_discimage.cd"
!
! REQUIRED FILE DEFINITIONS
application file    cdi_app     from "cdi_application"
message             from
                    "/home/MM_1.1/CDI/DISC/AUDIO/MESSAGES/message.long.iff"
copyright file      Copyright   from "copyright.txt"
abstract file       Abstract    from "abstract.txt"
biblio file         Biblio      from "bibliographic.txt"
!
! YELLOW FILE DEFINITIONS
yellow file         image0_cl8 from "image0.cl8"
```

```
yellow file        image1_c18 from "image1.c18"
!
! REALTIME FILE DEFINITION
green file rtf1 from
   record
      data packed in channel 2 from
         "/home/optima/dsl/1.c17">CAT#IMAG>FORM#IMAG>IHDR,
         "/home/optima/dsl/1.c17">CAT#IMAG>FORM#IMAG>PLTE
      video in channel 3 from
         "/home/optima/dsl/1.d">CAT#IMAG
   record
      real_time audio at 00:00:00 in channel 0 from
         "/home/optima/dsl/1.cm">CAT#AIFF
      real_time audio at 00:00:00 in channel 1 from
         "/home/optima/dsl/2.cm">CAT#AIFF
      video in channel 3 from
         "/home/optima/dsl/2.d">CAT#IMAG by 00:05:00,
         "/home/optima/dsl/3.d">CAT#IMAG by 00:10:00
      data packed in channel 2 from
         "/home/optima/dsl/2.d">CAT#IMAG>FORM#IMAG>IHDR by 00:05:00,
         "/home/optima/dsl/3.d">CAT#IMAG>FORM#IMAG>IHDR by 00:10:00
      triggers at 00:05:00, 00:10:00
!
! DIRECTORY STRUCTURE DEFINITION
{
   "abstract"       protection 0x555 from Abstract
   "bibliographic"  protection 0x555 from Biblio
   "copyright"      protection 0x555 from Copyright
   "cdi_appn"       protection 0x555 from cdi_app
   "RTF"                       ! subdirectory
   {
      "rtf1.rtf"    from rtf1
   }
   "VIDEO"                     ! subdirectory
   {
      "image0.c18"  protection 0x555 from image0_c18
      "image1.c18"  protection 0x555 from image1_c18
   }                           ! end of VIDEO subdirectory
}
```

master generated the following tracks report for the preceding script:

```
!
! @(#)Disc Building Utility for Sun 4 - Version 4.1
!
!       Album             Album Name
!       Publisher         I. M. Publisher
!       Preparer          M. Preparer
!
!       Volume            Volume Name
!       Disc name         cdi_discimage.cd
!
!  Track 00 (Data Mode 2 Form 2)
!
        leadin data
!
!  Track 01 (Data Mode 2 Form 1/2, scrambled)
!
        track data
        index 00          00:00:00 ! 00:01:74 (150 sectors)
        index 01          00:02:00 ! 01:04:74 (4725 sectors)
!
!  Track AA (Data Mode 2 Form 2, scrambled)
!
        leadout data
        index 01          01:05:00
```

master generated the following sector map report for the preceding script:

NOTE: Page breaks have been omitted to decrease the size of the output.

```
File RTF/rft1.rtf                                                    Page 1
Map for GREEN file RTF/rtf1.rtf


record 1 at 00:00:00 (0)

Data in channel 2

Source  Description
(hexa)

01      /home/optima/ds1/1.c17>CAT#IMAG>FORM#IMAG>IHDR
        Size      1 Sectors              (14 bytes)
        Coding    data                   (0x00)
        Channel   2
        Starting  00:00:00 (0)           (no request)
        Ending    00:00:00 (0)           (no request)

02      /home/optima/ds1/1.c17>CAT#IMAG>FORM#IMAG>PLTE
        Size      1 Sectors              (388 bytes)
        Coding    data                   (0x00)
        Channel   2
        Starting  00:00:00 (0)           (no request)
        Ending    00:00:00 (0)           (no request)

Video in channel 3

Source  Description
(hexa)
03      /home/optima/ds1/1.d>CAT#IMAG
        Size      47 Sectors             (107520 bytes)
        Coding    dyuv                   (0x05)
        Channel   3
        Starting  00:00:01 (1)           (no request)
        Ending    00:00:47 (47)          (no request)

record 2 at 00:00:48 (48)

Real_time audio in channel 0
```

```
Source  Description
(hexa)
01      /home/optima/dsl/1.cm>CAT#AIFF
        Size      141 Sectors           (327684 bytes)
        Coding    mono_c                (0x04)
        Channel   0
        Starting  00:00:00 (0)          (no request)
        Ending    00:29:65 (2240)       (no request)

Real_time audio in channel 1

Source  Description
(hexa)

02      /home/optima/dsl/2.cm>CAT#AIFF
        Size      141 Sectors           (327684 bytes)
        Coding    mono_c                (0x04)
        Channel   1
        Starting  00:00:08 (8)          (no request)
        Ending    00:29:73 (2248)       (no request)

Video in channel 3

Source  Description
(hexa)

03      /home/optima/dsl/2.d>CAT#IMAG
        Size      47 Sectors            (107520 bytes)
        Coding    dyuv                  (0x05)
        Channel   3
        Starting  00:04:23 (323)        (no request)
        Ending    00:05:00 (375)        (00:05:00 requested)

04      /home/optima/dsl/3.d>CAT#IMAG
        Size      47 Sectors            (107520 bytes)
        Coding    dyuv                  (0x05)
        Channel   3
        Starting  00:09:23 (698)        (no request)
        Ending    00:10:00 (750)        (00:10:00 requested)

Data in channel 2

Source  Description
(hexa)
```

```
05      /home/optima/dsl/2.d>CAT#IMAG>FORM#IMAG>IHDR
        Size      1 Sectors              (14 bytes)
        Coding    data                   (0x00)
        Channel   2
        Startng   00:04:22 (322)         (no request)
        Ending    00:04:22 (322)         (00:05:00 requested)

06      /home/optima/dsl/3.d>CAT#IMAG>FORM#IMAG>IHDR
        Size      1 Sectors              (14 bytes)
        Coding    data                   (0x00)
        Channel   2
        Starting  00:09:22 (697)         (no request)
        Ending    00:09:22 (697)         (00:10:00 requested)
```

Meaning of symbols Page 2

EVENTS			SYMBOL
EOF	TRG	EOR	
0	0	0	.
0	0	1	>
0	1	0	@
0	1	1	&
1	0	0	#
1	0	1	$
1	1	0	%
1	1	1	*

File RTF/rtf1.rtf Page 3

```
rel.time    vsn       rsn                    Sectors
-------------------------------------------------------------------------------
00:00:00  0000000  0000000     01 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
00:00:16  0000016  0000016     03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
00:00:32  0000032  0000032     03 03 03 03 03 03 03 03 03 03 03 03 03 03 03>03
00:00:48  0000048  0000000      01 . . . . . . . 02 . . . . . . . .
00:00:64  0000064  0000016      01 . . . . . . . 02 . . . . . . . .
00:01:05  0000080  0000032      01 . . . . . . . 02 . . . . . . . .
00:01:21  0000096  0000048      01 . . . . . . . 02 . . . . . . . .
00:01:37  0000112  0000064      01 . . . . . . . 02 . . . . . . . .
00:01:53  0000128  0000080      01 . . . . . . . 02 . . . . . . . .
00:01:69  0000144  0000096      01 . . . . . . . 02 . . . . . . . .
00:02:10  0000160  0000112      01 . . . . . . . 02 . . . . . . . .
00:02:26  0000176  0000128      01 . . . . . . . 02 . . . . . . . .
00:02:42  0000192  0000144      01 . . . . . . . 02 . . . . . . . .
00:02:58  0000208  0000160      01 . . . . . . . 02 . . . . . . . .
```

```
00:02:74  0000224  0000176     01 . . . . . . . 02 . . . . . . .
00:03:15  0000240  0000192     01 . . . . . . . 02 . . . . . . .
00:03:31  0000256  0000208     01 . . . . . . . 02 . . . . . . .
00:03:47  0000272  0000224     01 . . . . . . . 02 . . . . . . .
00:03:63  0000288  0000240     01 . . . . . . . 02 . . . . . . .
00:04:04  0000304  0000256     01 . . . . . . . 02 . . . . . . .
00:04:20  0000320  0000272     01 . . . . . . . 02 . . . . . . .
00:04:36  0000336  0000288     01 . . . . . . . 02 . . . . . . .
00:04:52  0000352  0000304     01 . . . . . . . 02 . . . . . . .
00:04:68  0000368  0000320     01 . 05 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:09  0000384  0000336     01 03 03 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:25  0000400  0000352     01 03 03 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:41  0000416  0000368     01 03 03 03 03 03 03@03 02 . . . . . . .
00:05:57  0000432  0000384     01 . . . . . . . 02 . . . . . . .
00:05:73  0000448  0000400     01 . . . . . . . 02 . . . . . . .
00:06:14  0000464  0000416     01 . . . . . . . 02 . . . . . . .
00:06:30  0000480  0000432     01 . . . . . . . 02 . . . . . . .
00:06:46  0000496  0000448     01 . . . . . . . 02 . . . . . . .
00:06:62  0000512  0000464     01 . . . . . . . 02 . . . . . . .
00:07:03  0000528  0000480     01 . . . . . . . 02 . . . . . . .
00:07:19  0000544  0000496     01 . . . . . . . 02 . . . . . . .
00:07:35  0000560  0000512     01 . . . . . . . 02 . . . . . . .
00:07:51  0000576  0000528     01 . . . . . . . 02 . . . . . . .
00:07:67  0000592  0000544     01 . . . . . . . 02 . . . . . . .
00:08:08  0000608  0000560     01 . . . . . . . 02 . . . . . . .
00:08:24  0000624  0000576     01 . . . . . . . 02 . . . . . . .
00:08:40  0000640  0000592     01 . . . . . . . 02 . . . . . . .
00:08:56  0000656  0000608     01 . . . . . . . 02 . . . . . . .
00:08:72  0000672  0000624     01 . . . . . . . 02 . . . . . . .
00:09:13  0000688  0000640     01 . . . . . . . 02 . . . . . . .
00:09:29  0000704  0000656     01 . . . . . . . 02 . . . . . . .
00:09:45  0000720  0000672     01 . . . . . . . 02 . . . . . . .
00:09:61  0000736  0000688     01 . . . . . . . 02 06 04 04 04 04 04 04
00:10:02  0000752  0000704     01 04 04 04 04 04 04 04 02 04 04 04 04 04 04 04
00:10:18  0000768  0000720     01 04 04 04 04 04 04 04 02 04 04 04 04 04 04 04
00:10:34  0000784  0000736     01 04 04 04 04 04 04 04 02 04 04 04 04 04@04 .
00:10:50  0000800  0000752     01 . . . . . . . 02 . . . . . . .
00:10:66  0000816  0000768     01 . . . . . . . 02 . . . . . . .
00:11:07  0000832  0000784     01 . . . . . . . 02 . . . . . . .
00:11:23  0000848  0000800     01 . . . . . . . 02 . . . . . . .
00:11:39  0000864  0000816     01 . . . . . . . 02 . . . . . . .
00:11:55  0000880  0000832     01 . . . . . . . 02 . . . . . . .
00:11:71  0000896  0000848     01 . . . . . . . 02 . . . . . . .
00:12:12  0000912  0000864     01 . . . . . . . 02 . . . . . . .
00:12:28  0000928  0000880     01 . . . . . . . 02 . . . . . . .
```

```
00:12:44  0000944  0000896      01 . . . . . 02 . . . . . .

File RTF/rtf1.rtf                                          Page    4

rel.time    vsn       rsn                    Sectors
-------------------------------------------------------------------------
00:12:60  0000960  0000912      01 . . . . . 02 . . . . . .
00:13:01  0000976  0000928      01 . . . . . 02 . . . . . .
00:13:17  0000992  0000944      01 . . . . . 02 . . . . . .
00:13:33  0001008  0000960      01 . . . . . 02 . . . . . .
00:13:49  0001024  0000976      01 . . . . . 02 . . . . . .
00:13:65  0001040  0000992      01 . . . . . 02 . . . . . .
00:14:06  0001056  0001008      01 . . . . . 02 . . . . . .
00:14:22  0001072  0001024      01 . . . . . 02 . . . . . .
00:14:38  0001088  0001040      01 . . . . . 02 . . . . . .
00:14:54  0001104  0001056      01 . . . . . 02 . . . . . .
00:14:70  0001120  0001072      01 . . . . . 02 . . . . . .
00:15:11  0001136  0001088      01 . . . . . 02 . . . . . .
00:15:27  0001152  0001104      01 . . . . . 02 . . . . . .
00:15:43  0001168  0001120      01 . . . . . 02 . . . . . .
00:15:59  0001184  0001136      01 . . . . . 02 . . . . . .
00:16:00  0001200  0001152      01 . . . . . 02 . . . . . .
00:16:16  0001216  0001168      01 . . . . . 02 . . . . . .
00:16:32  0001232  0001184      01 . . . . . 02 . . . . . .
00:16:48  0001248  0001200      01 . . . . . 02 . . . . . .
00:16:64  0001264  0001216      01 . . . . . 02 . . . . . .
00:17:05  0001280  0001232      01 . . . . . 02 . . . . . .
00:17:21  0001296  0001248      01 . . . . . 02 . . . . . .
00:17:37  0001312  0001264      01 . . . . . 02 . . . . . .
00:17:53  0001328  0001280      01 . . . . . 02 . . . . . .
00:17:69  0001344  0001296      01 . . . . . 02 . . . . . .
00:18:10  0001360  0001312      01 . . . . . 02 . . . . . .
00:18:26  0001376  0001328      01 . . . . . 02 . . . . . .
00:18:42  0001392  0001344      01 . . . . . 02 . . . . . .
00:18:58  0001408  0001360      01 . . . . . 02 . . . . . .
00:18:74  0001424  0001376      01 . . . . . 02 . . . . . .
00:19:15  0001440  0001392      01 . . . . . 02 . . . . . .
00:19:31  0001456  0001408      01 . . . . . 02 . . . . . .
00:19:47  0001472  0001424      01 . . . . . 02 . . . . . .
00:19:63  0001488  0001440      01 . . . . . 02 . . . . . .
00:20:04  0001504  0001456      01 . . . . . 02 . . . . . .
00:20:20  0001520  0001472      01 . . . . . 02 . . . . . .
00:20:36  0001536  0001488      01 . . . . . 02 . . . . . .
00:20:52  0001552  0001504      01 . . . . . 02 . . . . . .
00:20:68  0001568  0001520      01 . . . . . 02 . . . . . .
```

```
00:21:09  0001584  0001536      01 . . . . . 02 . . . . . .
00:21:25  0001600  0001552      01 . . . . . 02 . . . . . .
00:21:41  0001616  0001568      01 . . . . . 02 . . . . . .
00:21:57  0001632  0001584      01 . . . . . 02 . . . . . .
00:21:73  0001648  0001600      01 . . . . . 02 . . . . . .
00:22:14  0001664  0001616      01 . . . . . 02 . . . . . .
00:22:30  0001680  0001632      01 . . . . . 02 . . . . . .
00:22:46  0001696  0001648      01 . . . . . 02 . . . . . .
00:22:62  0001712  0001664      01 . . . . . 02 . . . . . .
00:23:03  0001728  0001680      01 . . . . . 02 . . . . . .
00:23:19  0001744  0001696      01 . . . . . 02 . . . . . .
00:23:35  0001760  0001712      01 . . . . . 02 . . . . . .
00:23:51  0001776  0001728      01 . . . . . 02 . . . . . .
00:23:67  0001792  0001744      01 . . . . . 02 . . . . . .
00:24:08  0001808  0001760      01 . . . . . 02 . . . . . .
00:24:24  0001824  0001776      01 . . . . . 02 . . . . . .
00:24:40  0001840  0001792      01 . . . . . 02 . . . . . .
00:24:56  0001856  0001808      01 . . . . . 02 . . . . . .
00:24:72  0001872  0001824      01 . . . . . 02 . . . . . .
00:25:13  0001888  0001840      01 . . . . . 02 . . . . . .
00:25:29  0001904  0001856      01 . . . . . 02 . . . . . .
00:25:45  0001920  0001872      01 . . . . . 02 . . . . . .
00:25:61  0001936  0001888      01 . . . . . 02 . . . . . .
00:26:02  0001952  0001904      01 . . . . . 02 . . . . . .
```

File RTF/rtf1.rtf Page 5

```
rel.time    vsn       rsn                    Sectors
-------------------------------------------------------------------------------
00:26:18  0001968  0001920      01 . . . . . 02 . . . . . .
00:26:34  0001984  0001936      01 . . . . . 02 . . . . . .
00:26:50  0002000  0001952      01 . . . . . 02 . . . . . .
00:26:66  0002016  0001968      01 . . . . . 02 . . . . . .
00:27:07  0002032  0001984      01 . . . . . 02 . . . . . .
00:27:23  0002048  0002000      01 . . . . . 02 . . . . . .
00:27:39  0002064  0002016      01 . . . . . 02 . . . . . .
00:27:55  0002080  0002032      01 . . . . . 02 . . . . . .
00:27:71  0002096  0002048      01 . . . . . 02 . . . . . .
00:28:12  0002112  0002064      01 . . . . . 02 . . . . . .
00:28:28  0002128  0002080      01 . . . . . 02 . . . . . .
00:28:44  0002144  0002096      01 . . . . . 02 . . . . . .
00:28:60  0002160  0002112      01 . . . . . 02 . . . . . .
00:29:01  0002176  0002128      01 . . . . . 02 . . . . . .
00:29:17  0002192  0002144      01 . . . . . 02 . . . . . .
00:29:33  0002208  0002160      01 . . . . . 02 . . . . . .
```

```
00:29:49  0002224  0002176      01 . . . . . . . 02 . . . . . . .
00:29:65  0002240  0002192      01 . . . . . . . 02 . . . . . . .
00:30:06  0002256  0002208      01 . . . . . . . 02 . . . . . . .
00:30:22  0002272  0002224      01 . . . . . . . 02 . . . . . . .
00:30:38  0002288  0002240      01 . . . . . . . 02 . . . . . . .
00:30:54  0002304  0002256       . . . . . . . $

Space efficiency 18%
```

Example 9: CD-I image with FMV assets

This example uses three scripts to build a disc image containing FMV assets. The first two scripts build realtime files from FMV assets, and the last script creates the disc image from the premastered realtime files and some additional yellow files.

The following script generates a realtime file named **fmv1.rtf**:

```
! MASTER OPTIONS
options: "fmv1.rtf"
!
! REALTIME FILE DEFINITION
record
   real_time mpeg in channel 0
      from "system.mmd"
      delta_file "system.dlt1"
      ep_list off
```

The following script generates a realtime file named **fmv2.rtf**:

```
! MASTER OPTIONS
options: "fmv2.rtf"
!
! REALTIME FILE DEFINITION
record
   real_time mpeg in channel 0
      from "system.mmd"
      delta_file "system.dlt2"
      ep_list off
```

The following script generates a disc image named **app.cd** using the two realtime files generated in the previous scripts:

```
! ALBUM DEFINITION
define album        "cdispot"
publisher           "OptImage Des Moines"
preparer            "Ivan Autour"
!
! VOLUME DEFINITION
volume "cdispot" in "app.cd"
!
! REQUIRED FILE DEFINITIONS
 message                        from "message.cda"
 copyright file     Copyright   from "copyright.txt"
```

```
 abstract file      Abstract    from "abstract.txt"
 biblio file        Biblio      from "bibliographic.txt"
 application file   Appl        from "cdi_fmv_demo"
!
! YELLOW FILE DEFINITIONS
yellow file         menu        from "fmv11.dyuv"
yellow file         dec_down    from "dec_down.bin"
yellow file         module      from "module"
yellow file         clipinfo    from "ClipInfo.dat"
yellow file         load        from "load"
yellow file         shell       from "shell"
yellow file         xmode       from "xmode"
!
! PREMASTERED REALTIME FILE DEFINITIONS
green file          bj          from "../bj/fmv.rtf"
green file          money       from "../money/fmv.rtf"
!
! DIRECTORY STRUCTURE DEFINITION
{
   "copyright"      protection 0x111 from Copyright
   "abstract"       protection 0x111 from Abstract
   "biblio"         protection 0x111 from Biblio
   "cdi_fmv_demo"   from Appl
   "module"         protection 0x555 from module
   "CMDS"
   {
      "load"        protection 0x555 from load
      "shell"       protection 0x555 from shell
      "xmode"       protection 0x555 from xmode
   }
   "DATA"
   {
      "dec_down.bin"            from dec_down
      "fmv11.dyuv"              from menu
      "ClipInfo.dat"            from clipinfo
      "bjCCpink.mpg"            from bj_cc_pink
      "moneyCCpink.mpg"         from money_cc_pink
   }
   "FMV"
   {
      "bj.rtf"      from bj
      "money.rtf"   from money
   }
}
```

Example 10: CD-I image with FMV assets using entry points

This example builds a disc image from two realtime files: **Fmv** and **Fmv1**. **Fmv** contains a record built from a realtime MPEG file and a video file. **Fmv2** contains two similar records. Each realtime MPEG file directs **master** to generate an entry point list for the MPEG data. **Fmv** requests an external (yellow) entry point file identified by **Link_epl**. The other records request internal entry point lists. Note that the external entry point file must be specified in the directory structure definition at the end of the script.

```
! ALBUM DEFINITION
define album        "FMV"
publisher           "ME"
preparer            "AGAIN ME"
!
! VOLUME DEFINITION
volume "FMV1" in "fmv_image"
!
! REQUIRED FILE DEFINITIONS
message                          from "AUDIO/message.cda"
copyright file      Copyright    from "TXT/copyright.txt"
abstract file       Abstract     from "TXT/abstract.txt"
biblio file         Biblio       from "TXT/bibliographic.txt"
application file    appl         from "CMDS/play_demo"
!
! REALTIME FILE DEFINITIONS
green file Fmv from
   record
      real_time mpeg at 2 in channel 0
         from "test.pim"
         delta_file "test_1.dlt"
         ep_list Link_epl with index from "test.pep"

      video in channel 0
         from "VIDEO/pic1.dyuv">CAT#IMAG

green file Fmv1 from
   record
      real_time mpeg in channel 0
         from "test.pim"
         delta_file "test_1.dlt"
         ep_list internal and gap 5 with index from "test.pep"

      video in channel 0
         from "VIDEO/pic1.dyuv">CAT#IMAG
```

```
   record
      real_time mpeg in channel 0
         from "test.pim"
         delta_file "test_1.dlt"
         ep_list internal and gap 5 with index from "test.pep"

      video in channel 0
         from "VIDEO/pic1.dyuv">CAT#IMAG by 148
!
! DIRECTORY STRUCTURE DEFINITION
{
   "copyright" from Copyright
   "abstract" from Abstract
   "biblio" from Biblio
   "CMDS"
   {
      "exec" from appl
   }
   "RTF"
   {
      "YELLOW"
      {
         "ep_yellow1" from Link_ep1
      }
      "gren" from Fmv
      "gren1" from Fmv1
   }

}
```

Appendix A

master Script Syntax Summary

The **master** script uses the following syntax to build disc images:

[**options:** *option...] album_def volume_def file_def... directory_def [volume_def file_def... directory_def]...*

The **master** script uses the following syntax to build realtime files:

[**options:** *option...] rec_def...*

The **master** script syntax elements are defined below:

option := **-**
| **-a**=*char_file*
| **-b**
| **-c**
| **-d**
| **-e**
| **-f**
| **-g**
| **-l**[=*leadout*]
| **-m**[=*map_file*]
| **-nt**
| **-o**[=*PMM_file*]
| **-p**
| **-q**
| **-r**
| **-s**

```
                | -t=toc_file
                | -u
                | -xa
                | -z
char_file       := pathlist
leadout         := decimal_number
map_file        := pathlist
PMM_file        := pathlist
toc_file        := pathlist

album_def       := define album album_name [label]
                | red album
album_name
                := string
label           := publisher string preparer string
                | publisher string
                | preparer string

volume_def      := volume vol_name in outfile
vol_name        := string
outfile         := pathlist

file_def        := green file ident from pathlist
                | red file ident from [swapped] source [with pre_emphasis]
                | yellow file ident from source
                | application file ident from pathlist
                | biblio file ident from pathlist
                | copyright file ident from pathlist
                | abstract file ident from pathlist
                | application_use from pathlist
                | message from [swapped] source
                | green file ident from rec_def
ident           := identifier
```

source := *pathlist*
| *pathlist*>*chunkpath*

rec_def := **block** *[rtl,...] stream_def...*
| **record** *[rtl,...] stream_def...*

stream_def := *rtaud_str*
| *mpeg_str*
| *audio_str*
| *video_str*
| *data_str*
| *event_str*

rtaud_str := **realtime audio** *[rtl,...]* [**at** *rsn]* **in channel** *channel*
from *a_source[, a_source...]*

a_source := *[rtl:...] a_asset* [**mask** *pattern]* [**offset** *offset* **length** *length]*
*[{***at** | **by***} rsn]*
| [**at** *rsn]* **silence** *sn* [*a_model]* *[{***at** | **by***} rsn]*

a_asset := *pathlist*[*a_model*]
| *pathlist*>*chunkpath*

a_model := *a_type [a_modifier]*

a_type := **mono_a**
| **mono_b**
| **mono_c**
| **stereo_a**
| **stereo_b**
| **stereo_c**

a_modifier := **emphasis**

pattern := *16bit_number*

offset := *number_of_bytes*

length := *number_of_bytes*

sn := *number_of_sectors*
| *hours:minutes:sectors*

mpeg_str := **realtime** [**still**] **mpeg** *[rtl,...]* [**at** *rsn]* **in channel** *channel*
from *mp_source[, mp_source...]*

mp_source := *[rtl:...] mp_asset* [**delta_file** *dfile]* **ep_list** *ep_spec*
*[{***at** | **by***} rsn]*

mp_asset := *pathlist*

dfile := *pathlist*

ep_spec := **off**
| **internal** *[***and gap** *rsn]* *[***with index***]* **from** *ep_infile*
| *directory_entry_ident* *[***with index***]* **from** *ep_infile*
| *ep_outfile* *[***with index***]* **from** *ep_infile*

ep_infile := *pathlist*

directory_entry_ident
:= *ident*

ep_outfile := *pathlist*

audio_str := **audio** *[***packed***]* *[rtl,...]* *[***at** *rsn]* **in channel** *channel*
from *a_source[, a_source...]*

video_str := **video** *[***packed***]* *[rtl,...]* *[***at** *rsn]* **in channel** *channel*
from *v_source[, v_source...]*

v_source := *[rtl:...] v_asset* *[***mask** *pattern]* *[{***at** | **by***} rsn]*

v_asset := *pathlist[v_mask]*
| *pathlist chunkpath*

pattern := *16bit_number*

v_mask := *v_type [v_modifier]*

v_type := **clut4**
| **clut7**
| **clut8**
| **rl3**
| **rl7**
| **dyuv**
| **rgb555l**
| **rgb555u**
| **rgb555**
| **qhy**
| **specific** *mask*

mask := *decimal_number*
| *hex_number*

v_modifier := **normal**
| **high**
| **double**
| **even**
| **odd**

data_str := **data** *[***packed***] [rtl,...] [***at** *rsn]* **in channel** *channel*
from *d_source[, d_source...]*

d_source := *[rtl:...] d_asset [***mask** *pattern] [***offset** *offset*
length *length] [{***at** | **by***} rsn]*

d_asset := *pathlist*
| *pathlist chunkpath*

event_str := *{***triggers** | **eors***}* **at** *event[,* **at** *event...]*

event := *[rtl:...] rsn [***every** *sn]*

hours := *decimal_number*

minutes := *decimal_number*

sectors := *decimal_number*

directory_def
:= **{***[file_entry...] [dir_name* **{***file_entry...***}***]***}**

file_entry := *file_name [attribute...]* **from** *ident*

file_name := *string*

dir_name := *string*

attribute := **owner** *group.user*
| **protection** *permissions*
| **hidden**
| **pad**

pathlist := *[string/...]string*

chunkpath := *[>wrapper...][>chunk]*

wrapper := *{***FORM** | **LIST** | **CAT***} #chunk*

chunk := *IFFID[#index]*

index := *decimal_number*

```
channel  := 1..15

rtl      := ident
          | ident(expr)
expr     := hex_number
          | decimal_number
          | (expr)
          | expr + expr
          | expr - expr
          | expr / expr
          | expr * expr
```

Appendix B

Audio/Video Source Mask Values

Audio and video sources for a realtime record are specified using pathlists and optional coding masks. A coding mask consists of a main keyword indicating the source's audio or video type and one or more optional coding modifiers.

The following audio coding keywords specify bitfield 0–4:

mono_a (value 0x10)
mono_b (value 0x00)
mono_c (value 0x04)
stereo_a (value 0x11)
stereo_b (value 0x01)
stereo_c (value 0x05)

The following audio coding modifier specifies bitfield 6:

emphasis (bitfield 6, Value 0x40)

The following video coding keywords specify bitfield 0–3:

clut4 (bitfield 0–3, value 0x00)
clut7 (bitfield 0–3, value 0x01)
clut8 (bitfield 0–3, value 0x02)
r13 (bitfield 0–3, value 0x03)
r17 (bitfield 0–3, value 0x04)
dyuv (bitfield 0–3, value 0x05)
rgb555l (bitfield 0–3, value 0x06)

rgb555u **(bitfield 0–3, value 0x07)**
qhy **(bitfield 0–3, value 0x08)**
rgb555 **(bitfield 0–3, value 0x09)**

rgb555 is a pseudo video coding keyword indicating that **rgb555** sources are duplicated into their upper and lower planes.

The following video coding modifiers specify bitfields 4–6:

normal **(bitfield 4–5, value 0x00, default)**
double **(bitfield 4–5, value 0x10)**
high **(bitfield 4–5, value 0x30)**
even **(bitfield 6, value 0x00, default)**
odd **(bitfield 6, value 0x40)**

You can specify the video mask directly using the **specific** keyword and a mask value. The mask value may be specified as a hexadecimal or decimal number, but must be in the range of 0 to 255 (0x00–0xFF).

specific **(bitfield 0–7, value 0x00 to 0xFF)**

Appendix C

master Script Keywords

The following words cannot be used as identifiers because they are reserved by **master** as keywords. All letters in keywords are lower case:

abstract	**album**	**and**	**application**	**at**
audio	**biblio**	**block**	**by**	**channel**
clut4	**clut7**	**clut8**	**copyright**	**data**
define	**delta_file**	**double**	**dyuv**	**emphasis**
eors	**ep_list**	**even**	**every**	**file**
from	**gap**	**green**	**hidden**	**high**
in	**index**	**interleave**	**internal**	**length**
mask	**message**	**mpeg**	**mono_a**	**mono_b**
mono_c	**normal**	**odd**	**off**	**offset**
owner	**packed**	**pre_emphasis**	**preparer**	**protection**
publisher	**qhy**	**real_time**	**record**	**red**
rgb555	**rgb555l**	**rgb555u**	**rl3**	**rl7**
silence	**specific**	**stereo_a**	**stereo_b**	**stereo_c**
still	**swapped**	**triggers**	**video**	**volume**
with	**yellow**			

Appendix D

Balboa Play Manager PMM Structures

The Balboa Play Manager

The Balboa Play Manager provides two levels of interaction with CD-I hardware and firmware. On a low-level, the Play Manager deals with disk events, assets, etc. on a detailed, event-specific basis. On a high-level, the Play Manager provides the application with symbolic references for disk events, assets, etc. The Play Manager can only access realtime files with embedded PMM structures.

The high-level map functions use special PMM structures placed in your disc image to provide you with symbolic references for events and assets. The functions allow you to access the PMM structures and find out various things about the realtime file. For example, the **pmm_count_eors** function returns the exact number of end-of-record (EOR) bits set between two specified locations on the disc. Likewise, you can use the **pmm_find_trigger** or **pmm_find_eor** functions to find the next trigger or EOR bit.

For the map functions to work, you must use the **-o** or **-z master** command line options when building a realtime file. These options build the necessary structures into the disc image. The **-o** option also generates an external file containing the PMM structures.

BALBOA realtime file and realtime record maps

All realtime file maps and realtime record maps require the following structure:

- All the "longs" must be 4 bytes and aligned on a half-word boundary.
- The size of a "struct" must be a multiple of 2 and aligned on a half-word boundary.
- The "longs" have to have the following byte ordering:

  byte 3 byte 2 byte 1 byte 0

The diagram in Figure 1 shows how the Balboa PMM structures fit into the sectors of a realtime file. Each sector containing a PMM structure is in channel 0.

The **PMM_RTF_REC** structure is placed at the beginning of the realtime file. A **PMM_RTR_REC** structure is placed at the beginning and the end of each realtime record in the realtime file.

NOTE: All maps can be read in memory as a block for OS-9. All data is aligned on 2 byte boundaries.

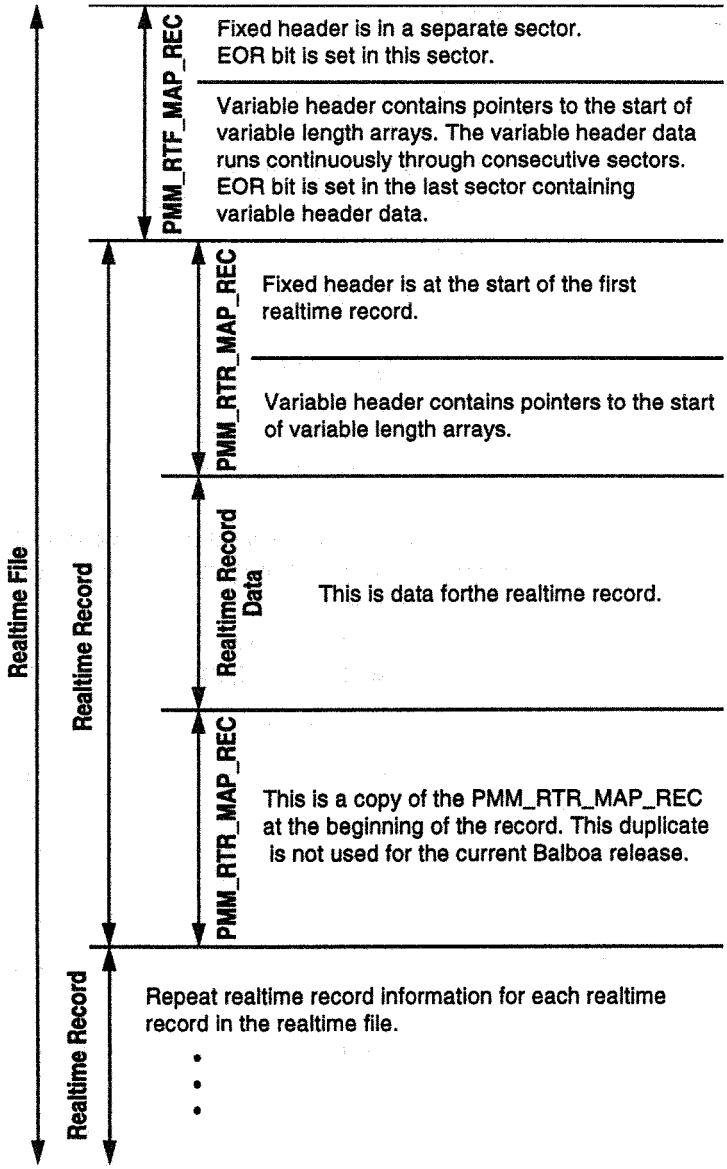


Figure D-1: PMM structure placement in realtime files

PMM_RTF_MAP_REC

PMM_RTF_MAP_REC is the heart of the realtime file's realtime record index. This structure is opened by **pmm_open_rtf()** and is interrogated by **pmm_open_rtr()** to determine the placement of the realtime record maps in the realtime file.

PMM_RTF_MAP_REC is defined as follows:

```
typedef struct
{
    long application_id;
    long rtr_count;
    long label_space_size;
    long user_space_size;

/* the following are pointers to the start of the variable length arrays*/
    PMM_RTR_INDEX_REC *rtr_index_space;
    char *label_space;
    char *user_space;
}  PMM_RTF_MAP_REC;
```

where

application_id	contains an ID that identifies either the disc builder program that made this realtime file or the runtime engine for which this realtime file is intended. This ID is assigned by Philips.
rtr_count	contains the number of realtime records in the file.
label_space_size	contains the size of the realtime file label space.
user_space_size	contains the size of the user-defined data block that may be added to the end of the realtime file's realtime record index.

rtr_index_space contains a pointer to the start of the realtime record index array.

label_space contains a pointer to the start of the label space.

user_space contains a pointer to the start of the user-defined data block. **user_space** contains **NULL** if there is no user-defined data block.

PMM_RTR_INDEX_REC

PMM_RTR_INDEX_REC is a component of **RTF_MAP_REC** and describes the position and size of a realtime record map. The realtime record index consists of an array of **PMM_RTR_INDEX_REC** structures.

PMM_RTR_INDEX_REC is defined as follows:

```
typedef struct
{
    long label_start_offset;
    long map_rtr_sector;
    long dup_map_rtr_sector;
    long rtr_map_size;
    long map_rtr_size;
} PMM_RTR_INDEX_REC;
```

where

label_start_offset contains the byte offset from the start of the realtime file's label space to the label of the realtime record associated with this record.

map_rtr_sector contains the sector offset from the start of the realtime file to the realtime record map.

dup_map_rtr_sector contains the sector offset from the start of the realtime file to the duplicate realtime record map.

rtr_map_size contains the sector size of the realtime record map.

map_rtr_size contains the byte size of the realtime record map.

PMM_RTR_MAP_REC

PMM_RTR_MAP_REC reflects the structure of the map data. It is used to access the symbolic data regarding a realtime record that is contained in a realtime file map.

PMM_RTR_MAP_REC is defined as follows:

```
typedef struct
{
    long audio_channel;
    long video_channel;
    long data_channel;
    long channel_count;
    long asset_label_count;
    long trigger_count;
    long eor_count;
    long asset_count;
    long data_rtr_position;
    long dup_data_rtr_position;
    long user_data_size;
    long label_space_size;

/* the following are pointers to the start of the variable length arrays*/
    char *label_space;
    PMM_CHAN_NUM_REC *channel_space;
    PMM_ASSET_LABEL_REC *asset_label_space;
    PMM_TRIGGER_REC *trigger_space;
    PMM_EOR_REC *eor_space;
    PMM_ASSET_REC *asset_space;
    char *user_space;
} PMM_RTR_MAP_REC;
```

where

audio_channel contains a mask of the legal audio channels in this realtime record.

video_channel contains a mask of the legal video channels in this realtime record.

data_channel contains a mask of the legal data channels in this realtime record.

channel_count contains the number of labeled channels in this realtime record.

asset_label_count contains the number of labeled assets in this realtime record.

trigger_count contains the number of triggers in this realtime record.

eor_count contains the number of EORs in this realtime record.

asset_count contains the number of assets (labeled and unlabeled) in this realtime record.

dup_rtr_position contains the sector position of the duplicate map realtime record.

dup_data_rtr_position contains the sector position of the duplicate data realtime record.

user_data_size contains the size of a user-defined data block.

label_space_size contains the size of the realtime record label space.

label_space contains a pointer to the start of the realtime record label space.

channel_space contains a pointer to an array of **PMM_CHANNEL_REC** structures.

asset_label_space contains a pointer to an array of **PMM_ASSET_LABEL_REC** structures.

trigger_space contains a pointer to an array of **PMM_TRIGGER_REC** structures.

eor_space	contains a pointer to an array of **PMM_EOR_REC** structures.
asset_space	contains a pointer to an array of **PMM_ASSET_REC** structures.
user_space	contains a pointer to a user-defined data block.

PMM_CHAN_NUM_REC

PMM_CHAN_NUM_REC is a component of **PMM_RTR_MAP_REC** containing the information about the bindings of channel names to channel numbers. This information is particularly useful when you are working with systems that perform their own channel assignments when building a disc (for example, DBL). **PMM_RTR_MAP_REC** contains an array of **PMM_CHAN_NUM_REC** structures, one for each EOR in the realtime record.

PMM_CHAN_NUM_REC is defined as follows:

```
typedef struct
{
    long label_start_offet;
    char channel;
}  PMM_EOR_REC;
```

where

label_start_offset	contains the byte offset from the beginning of the realtime record label space of the first character in the name of this channel.
channel	contains the channel number bound to this label.

PMM_EOR_REC

PMM_EOR_REC is a component of **PMM_RTR_MAP_REC** containing the position, label, and channel/type information necessary to find an EOR by its label. **PMM_RTR_MAP_REC** contains a pointer to an array of **PMM_EOR_REC** structures, one for each EOR in the realtime record.

PMM_EOR_REC is defined as follows:

```
typedef struct
{
    long label_start_offet;
    long position;
    char channel;
    char type;
}  PMM_EOR_REC;
```

where

label_start_offset contains the byte offset from the beginning of the realtime record label space of the first character in this EOR label. **label_start_offset** contains a value of -1 if the EOR is not labeled.

position contains the sector offset from the beginning of the realtime record of the sector that contains the EOR.

channel contains the channel number associated with the EOR.

type contains the data stream type associated with this EOR:

0	audio
1	video
2	data
3	direct audio
99	any type

PMM_TRIGGER_REC

PMM_TRIGGER_REC is a component of **PMM_RTR_MAP_REC** containing the location and label (if any) of a trigger. **PMM_RTR_MAP_REC** contains an array of **PMM_TRIGGER_REC** structures, one for each trigger in the realtime record.

PMM_TRIGGER_REC is defined as follows:

```
typedef struct
{
   long label_start_offet;
   long position;
} PMM_TRIGGER_REC;
```

where

label_start_offset contains the byte offset from the beginning of the realtime record label space of the first character in this trigger's label. **label_start_offset** contains a value of -1 if the trigger is not labeled.

position contains the sector offset from the start of the realtime record of the sector that contains the trigger.

PMM_ASSET_REC

PMM_ASSET_REC is a component of **PMM_RTR_MAP_REC** containing all the information necessary to create and add buffers to buffer lists prior to play. **PMM_RTR_MAP_REC** contains an array of **PMM_ASSET_REC** structures, one for each labeled asset in the play sequence.

PMM_ASSET_REC is defined as follows:

```
typedef struct
{
    char channel;
    char type;
    long size;
}  PMM_ASSET_REC;
```

where

channel contains the number of the channel that contains the asset.

type contains the data stream type associated with the asset:

0	audio
1	video
2	data

size contains the sector size of the asset.

PMM_ASSET_LABEL_REC

PMM_ASSET_LABEL_REC is a component of **PMM_RTR_MAP_REC** containing all the information necessary to begin or end play of a particular asset. **PMM_RTR_MAP_REC** contains an array of **PMM_ASSET_LABEL_REC** structures, one for each labeled asset in the play sequence.

PMM_ASSET_LABEL_REC is defined as follows:

```
typedef struct
{
    long label_start_offset;
    long start_sector_position;
    long end_sector_position;
    long asset_index;
} PMM_ASSET_LABEL_REC;
```

where

label_start_offset contains the byte offset from the start of the play sequence label space of the first character in the name of this asset.

start_sector_position contains the sector offset from the beginning of the play sequence of the first sector of this asset.

end_sector_position contains the sector offset from the beginning of the play sequence of the last sector of this asset.

asset_index contains an index of the asset table for this play sequence of the asset associated with this label.

Appendix E Sector Map Format

The map report (requested by the **-m** option) describes each real-time file. For each realtime file, a header and a sector map are output.

The map header describes each record, each stream, and each source. The source information includes:

- a hexadecimal number (modulo 255) by which the source is denoted in the sector map section
- file name
- file size, both in bytes and in sectors
- coding byte value and name
- positioning constraints

Each line of the sector map section contains:

- the file relative sector number (vsn) of the first sector of the line in minutes:seconds:frame format and in absolute frame format
- the record relative sector number of the first sector of the line in absolute format
- source numbers of the next 16 sectors (empty sectors being denoted by a dot (.)), preceded or replaced by a # if the sector has events (EOR, TRIGGER, EOF)

Example sector map report

```
File RTF/rtf1.rtf                                                    Page    1
Map for GREEN file RTF/rtf1.rtf

record 1 at 00:00:00 (0)

Data in channel 2

Source          Description
(hexa)

01              /home/optima/dsl/1.c17>CAT#IMAG>FORM#IMAG>IHDR
                Size     1 Sectors       (14 bytes)
                Coding   data            (0x00)
                Channel  2
                Starting 00:00:00 (0)      (no request)
                Ending   00:00:00 (0)      (no request)

02              /home/optima/dsl/1.c17>CAT#IMAG>FORM#IMAG>PLTE
                Size     1 Sectors       (388 bytes)
                Coding   data            (0x00)
                Channel  2
                Starting 00:00:00 (0)      (no request)
                Ending   00:00:00 (0)      (no request)

Video in channel 3

Source          Description
(hexa)

03              /home/optima/dsl/1.d>CAT#IMAG
                Size     47 Sectors      (107520 bytes)
                Coding   dyuv            (0x05)
                Channel  3
                Starting 00:00:01 (1)      (no request)
                Ending   00:00:47 (47)     (no request)

record 2 at 00:00:48 (48)

Real_time audio in channel 0

Source          Description
(hexa)
```

```
01              /home/optima/dsl/1.cm>CAT#AIFF
                Size     141 Sectors      (327684 bytes)
                Coding   mono_c           (0x04)
                Channel  0
                Starting 00:00:00 (0)       (no request)
                Ending   00:29:65 (2240)    (no request)

Real_time audio in channel 1

Source          Description
(hexa)

02              /home/optima/dsl/2.cm>CAT#AIFF
                Size     141 Sectors      (327684 bytes)
                Coding   mono_c           (0x04)
                Channel  1
                Starting 00:00:08 (8)       (no request)
                Ending   00:29:73 (2248)    (no request)

Video in channel 3

Source          Description
(hexa)

03              /home/optima/dsl/2.d>CAT#IMAG
                Size     47 Sectors       (107520 bytes)
                Coding   dyuv             (0x05)
                Channel  3
                Starting 00:04:23 (323)     (no request)
                Ending   00:05:00 (375)     (00:05:00 requested)

04              /home/optima/dsl/3.d>CAT#IMAG
                Size     47 Sectors       (107520 bytes)
                Coding   dyuv             (0x05)
                Channel  3
                Starting 00:09:23 (698)     (no request)
                Ending   00:10:00 (750)     (00:10:00 requested)

Data in channel 2


Source          Description
(hexa)
```

```
05              /home/optima/dsl/2.d>CAT#IMAG>FORM#IMAG>IHDR
                Size 1 Sectors(14 bytes)
                Coding   data            (0x00)
                Channel  2
                Starting 00:04:22 (322)     (no request)
                Ending   00:04:22 (322)     (00:05:00 requested)

06              /home/optima/dsl/3.d>CAT#IMAG>FORM#IMAG>IHDR
                Size     1 Sectors       (14 bytes)
                Coding   data            (0x00)
                Channel  2
                Starting 00:09:22 (697)     (no request)
                Ending   00:09:22 (697)     (00:10:00 requested)
```

Meaning of symbols Page 2

EVENTS			SYMBOL
EOF	TRG	EOR	
0	0	0	.
0	0	1	>
0	1	0	@
0	1	1	&
1	0	0	#
1	0	1	$
1	1	0	%
1	1	1	*

File RTF/rtf1.rtf Page 3

```
rel.time    vsn      rsn         Sectors
--------------------------------------------------------------------------------

00:00:00  0000000  0000000      01 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
00:00:16  0000016  0000016      03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
00:00:32  0000032  0000032      03 03 03 03 03 03 03 03 03 03 03 03 03 03 03>03

00:00:48  0000048  0000000      01 . . . . . . . 02 . . . . . . .
00:00:64  0000064  0000016      01 . . . . . . . 02 . . . . . . .
00:01:05  0000080  0000032      01 . . . . . . . 02 . . . . . . .
00:01:21  0000096  0000048      01 . . . . . . . 02 . . . . . . .
00:01:37  0000112  0000064      01 . . . . . . . 02 . . . . . . .
00:01:53  0000128  0000080      01 . . . . . . . 02 . . . . . . .
```

```
00:01:69  0000144  0000096     01 . . . . . . . 02 . . . . . . .
00:02:10  0000160  0000112     01 . . . . . . . 02 . . . . . . .
00:02:26  0000176  0000128     01 . . . . . . . 02 . . . . . . .
00:02:42  0000192  0000144     01 . . . . . . . 02 . . . . . . .
00:02:58  0000208  0000160     01 . . . . . . . 02 . . . . . . .
00:02:74  0000224  0000176     01 . . . . . . . 02 . . . . . . .
00:03:15  0000240  0000192     01 . . . . . . . 02 . . . . . . .
00:03:31  0000256  0000208     01 . . . . . . . 02 . . . . . . .
00:03:47  0000272  0000224     01 . . . . . . . 02 . . . . . . .
00:03:63  0000288  0000240     01 . . . . . . . 02 . . . . . . .
00:04:04  0000304  0000256     01 . . . . . . . 02 . . . . . . .
00:04:20  0000320  0000272     01 . . . . . . . 02 . . . . . . .
00:04:36  0000336  0000288     01 . . . . . . . 02 . . . . . . .
00:04:52  0000352  0000304     01 . . . . . . . 02 . . . . . . .
00:04:68  0000368  0000320     01 . 05 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:09  0000384  0000336     01 03 03 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:25  0000400  0000352     01 03 03 03 03 03 03 03 02 03 03 03 03 03 03 03
00:05:41  0000416  0000368     01 03 03 03 03 03 03@03 02 . . . . . . .
00:05:57  0000432  0000384     01 . . . . . . . 02 . . . . . . .
00:05:73  0000448  0000400     01 . . . . . . . 02 . . . . . . .
00:06:14  0000464  0000416     01 . . . . . . . 02 . . . . . . .
00:06:30  0000480  0000432     01 . . . . . . . 02 . . . . . . .
00:06:46  0000496  0000448     01 . . . . . . . 02 . . . . . . .
00:06:62  0000512  0000464     01 . . . . . . . 02 . . . . . . .
00:07:03  0000528  0000480     01 . . . . . . . 02 . . . . . . .
00:07:19  0000544  0000496     01 . . . . . . . 02 . . . . . . .
00:07:35  0000560  0000512     01 . . . . . . . 02 . . . . . . .
00:07:51  0000576  0000528     01 . . . . . . . 02 . . . . . . .
00:07:67  0000592  0000544     01 . . . . . . . 02 . . . . . . .
00:08:08  0000608  0000560     01 . . . . . . . 02 . . . . . . .
00:08:24  0000624  0000576     01 . . . . . . . 02 . . . . . . .
00:08:40  0000640  0000592     01 . . . . . . . 02 . . . . . . .
00:08:56  0000656  0000608     01 . . . . . . . 02 . . . . . . .
00:08:72  0000672  0000624     01 . . . . . . . 02 . . . . . . .
00:09:13  0000688  0000640     01 . . . . . . . 02 . . . . . . .
00:09:29  0000704  0000656     01 . . . . . . . 02 . . . . . . .
00:09:45  0000720  0000672     01 . . . . . . . 02 . . . . . . .
00:09:61  0000736  0000688     01 . . . . . . . 02 06 04 04 04 04 04 04
00:10:02  0000752  0000704     01 04 04 04 04 04 04 04 02 04 04 04 04 04 04 04
00:10:18  0000768  0000720     01 04 04 04 04 04 04 04 02 04 04 04 04 04 04 04
00:10:34  0000784  0000736     01 04 04 04 04 04 04 04 02 04 04 04 04 04@04 .
00:10:50  0000800  0000752     01 . . . . . . . 02 . . . . . . .
00:10:66  0000816  0000768     01 . . . . . . . 02 . . . . . . .
00:11:07  0000832  0000784     01 . . . . . . . 02 . . . . . . .
00:11:23  0000848  0000800     01 . . . . . . . 02 . . . . . . .
```

```
00:11:39  0000864  0000816      01 . . . . . 02 . . . . . .
00:11:55  0000880  0000832      01 . . . . . 02 . . . . . .
00:11:71  0000896  0000848      01 . . . . . 02 . . . . . .
00:12:12  0000912  0000864      01 . . . . . 02 . . . . . .
00:12:28  0000928  0000880      01 . . . . . 02 . . . . . .
00:12:44  0000944  0000896      01 . . . . . 02 . . . . . .
File RTF/rtf1.rtf                                    Page    4

rel.time    vsn       rsn         Sectors
-------------------------------------------------------------------------
00:12:60  0000960  0000912      01 . . . . . 02 . . . . . .
00:13:01  0000976  0000928      01 . . . . . 02 . . . . . .
00:13:17  0000992  0000944      01 . . . . . 02 . . . . . .
00:13:33  0001008  0000960      01 . . . . . 02 . . . . . .
00:13:49  0001024  0000976      01 . . . . . 02 . . . . . .
00:13:65  0001040  0000992      01 . . . . . 02 . . . . . .
00:14:06  0001056  0001008      01 . . . . . 02 . . . . . .
00:14:22  0001072  0001024      01 . . . . . 02 . . . . . .
00:14:38  0001088  0001040      01 . . . . . 02 . . . . . .
00:14:54  0001104  0001056      01 . . . . . 02 . . . . . .
00:14:70  0001120  0001072      01 . . . . . 02 . . . . . .
00:15:11  0001136  0001088      01 . . . . . 02 . . . . . .
00:15:27  0001152  0001104      01 . . . . . 02 . . . . . .
00:15:43  0001168  0001120      01 . . . . . 02 . . . . . .
00:15:59  0001184  0001136      01 . . . . . 02 . . . . . .
00:16:00  0001200  0001152      01 . . . . . 02 . . . . . .
00:16:16  0001216  0001168      01 . . . . . 02 . . . . . .
00:16:32  0001232  0001184      01 . . . . . 02 . . . . . .
00:16:48  0001248  0001200      01 . . . . . 02 . . . . . .
00:16:64  0001264  0001216      01 . . . . . 02 . . . . . .
00:17:05  0001280  0001232      01 . . . . . 02 . . . . . .
00:17:21  0001296  0001248      01 . . . . . 02 . . . . . .
00:17:37  0001312  0001264      01 . . . . . 02 . . . . . .
00:17:53  0001328  0001280      01 . . . . . 02 . . . . . .
00:17:69  0001344  0001296      01 . . . . . 02 . . . . . .
00:18:10  0001360  0001312      01 . . . . . 02 . . . . . .
00:18:26  0001376  0001328      01 . . . . . 02 . . . . . .
00:18:42  0001392  0001344      01 . . . . . 02 . . . . . .
00:18:58  0001408  0001360      01 . . . . . 02 . . . . . .
00:18:74  0001424  0001376      01 . . . . . 02 . . . . . .
00:19:15  0001440  0001392      01 . . . . . 02 . . . . . .
00:19:31  0001456  0001408      01 . . . . . 02 . . . . . .
00:19:47  0001472  0001424      01 . . . . . 02 . . . . . .
00:19:63  0001488  0001440      01 . . . . . 02 . . . . . .
```

```
00:20:04  0001504  0001456      01 . . . . . . . 02 . . . . . . .
00:20:20  0001520  0001472      01 . . . . . . . 02 . . . . . . .
00:20:36  0001536  0001488      01 . . . . . . . 02 . . . . . . .
00:20:52  0001552  0001504      01 . . . . . . . 02 . . . . . . .
00:20:68  0001568  0001520      01 . . . . . . . 02 . . . . . . .
00:21:09  0001584  0001536      01 . . . . . . . 02 . . . . . . .
00:21:25  0001600  0001552      01 . . . . . . . 02 . . . . . . .
00:21:41  0001616  0001568      01 . . . . . . . 02 . . . . . . .
00:21:57  0001632  0001584      01 . . . . . . . 02 . . . . . . .
00:21:73  0001648  0001600      01 . . . . . . . 02 . . . . . . .
00:22:14  0001664  0001616      01 . . . . . . . 02 . . . . . . .
00:22:30  0001680  0001632      01 . . . . . . . 02 . . . . . . .
00:22:46  0001696  0001648      01 . . . . . . . 02 . . . . . . .
00:22:62  0001712  0001664      01 . . . . . . . 02 . . . . . . .
00:23:03  0001728  0001680      01 . . . . . . . 02 . . . . . . .
00:23:19  0001744  0001696      01 . . . . . . . 02 . . . . . . .
00:23:35  0001760  0001712      01 . . . . . . . 02 . . . . . . .
00:23:51  0001776  0001728      01 . . . . . . . 02 . . . . . . .
00:23:67  0001792  0001744      01 . . . . . . . 02 . . . . . . .
00:24:08  0001808  0001760      01 . . . . . . . 02 . . . . . . .
00:24:24  0001824  0001776      01 . . . . . . . 02 . . . . . . .
00:24:40  0001840  0001792      01 . . . . . . . 02 . . . . . . .
00:24:56  0001856  0001808      01 . . . . . . . 02 . . . . . . .
00:24:72  0001872  0001824      01 . . . . . . . 02 . . . . . . .
00:25:13  0001888  0001840      01 . . . . . . . 02 . . . . . . .
00:25:29  0001904  0001856      01 . . . . . . . 02 . . . . . . .
00:25:45  0001920  0001872      01 . . . . . . . 02 . . . . . . .
00:25:61  0001936  0001888      01 . . . . . . . 02 . . . . . . .
00:26:02  0001952  0001904      01 . . . . . . . 02 . . . . . . .
File RTF/rtf1.rtf                                        Page    5
```

```
rel.time    vsn       rsn         Sectors
-------------------------------------------------------------------------------
00:26:18  0001968  0001920      01 . . . . . . . 02 . . . . . . .
00:26:34  0001984  0001936      01 . . . . . . . 02 . . . . . . .
00:26:50  0002000  0001952      01 . . . . . . . 02 . . . . . . .
00:26:66  0002016  0001968      01 . . . . . . . 02 . . . . . . .
00:27:07  0002032  0001984      01 . . . . . . . 02 . . . . . . .
00:27:23  0002048  0002000      01 . . . . . . . 02 . . . . . . .
00:27:39  0002064  0002016      01 . . . . . . . 02 . . . . . . .
00:27:55  0002080  0002032      01 . . . . . . . 02 . . . . . . .
00:27:71  0002096  0002048      01 . . . . . . . 02 . . . . . . .
00:28:12  0002112  0002064      01 . . . . . . . 02 . . . . . . .
00:28:28  0002128  0002080      01 . . . . . . . 02 . . . . . . .
```

```
00:28:44  0002144  0002096      01 . . . . . . . 02 . . . . . . . .
00:28:60  0002160  0002112      01 . . . . . . . 02 . . . . . . . .
00:29:01  0002176  0002128      01 . . . . . . . 02 . . . . . . . .
00:29:17  0002192  0002144      01 . . . . . . . 02 . . . . . . . .
00:29:33  0002208  0002160      01 . . . . . . . 02 . . . . . . . .
00:29:49  0002224  0002176      01 . . . . . . . 02 . . . . . . . .
00:29:65  0002240  0002192      01 . . . . . . . 02 . . . . . . . .
00:30:06  0002256  0002208      01 . . . . . . . 02 . . . . . . . .
00:30:22  0002272  0002224      01 . . . . . . . 02 . . . . . . . .
00:30:38  0002288  0002240      01 . . . . . . . 02 . . . . . . . .
00:30:54  0002304  0002256       . . . . . . . $

Space efficiency 18%
```

Appendix F Tracks Report Format

master generates tracks as follows:

- One single green track is generated, padded up to the next second boundary with empty sectors.
- One red track is generated for each red file (sequence is controlled by the file placement policy). Red tracks are preceded by an audio pause of 2 seconds and padded up to the next second boundary with PCM silence.

The tracks (or table of contents) report is designed to be either read by the CD mastering factory to control the generation of the Q channel stream or parsed by an emulator program for the same purpose. It is output using the following syntax:

```
toc      := leadin track ... leadout
leadin   := leadin { audio | data }
track    := track { audio | data } index ...
index    := index NUMBER time
time     := NUMBER : NUMBER : NUMBER
leadout  := leadout { audio | data }
```

Comments begin with an (!) and end at the end of the line.

The following is an example tracks report generated by **master**:

```
!
! @(#)Disc Building Utility for Sun 4 - Version 4.1

!
!            Album        Album Name
!            Publisher    I. M. Publisher
!            Preparer     M. Preparer
!
!            Volume       Volume Name
!            Disc name    cdi_discimage.cd
!
!  Track 00 (Data Mode 2 Form 2)
!
             leadin data
!
!  Track 01 (Data Mode 2 Form 1/2, scrambled)
!
             track data
             index 00     00:00:00 ! 00:01:74 (150 sectors)
             index 01     00:02:00 ! 01:04:74 (4725 sectors)
!
!  Track AA (Data Mode 2 Form 2, scrambled)
!
             leadout data
             index 01     01:05:00
```

Appendix G

Entry Point Lists

Overview

Entry points are only used in realtime MPEG assets. Essentially, entry points contain information necessary to access specific points in the MPEG stream. Entry points are created during the process of generating MPEG assets. Consequently, to understand how entry points are created, labeled, and indexed, it is necessary to understand the process of creating MPEG assets.

The following sections discuss:

- the process of creating MPEG assets
- the definition of entry points
- the structure of **pink** entry point files
- **master** entry point lists
- **master** entry point data structures

Creating full motion video assets

Producing MPEG assets for CD-I titles involves five major steps:

1. **creating an Encoding Control List (ECL)**
   The ECL is a script file that controls the processes of grabbing and encoding audio and video assets.
2. **grabbing/preprocessing**
   Audio and video assets must be converted into a digital format that can be read by the MPEG encoder.
3. **encoding**
   The grabbed files must be converted into files that conform to the MPEG standard for full motion video and audio.
4. **multiplexing**
   The encoded audio and video files must be combined into a single interleaved file ready to be included in a CD-I disc image.
5. **premastering**
   The multiplexed files must be combined with other CD-I realtime files to build a disc image.

The utilities controlling this process include **ecl_tool, grb_tool, enc_tool, rtae, pink,** and **master**. The specific actions of each program are controlled by a script file that you create (or that is generated by an authoring tool). With the exceptions of **grb_tool** and **rtae**, each utility also reads one or more files generated by the previous program in the MPEG asset development process (shown in Figure 1).

NOTE: With the exception of **master** *and* **rtae**, *all of these utilities exist only on the Sun platform at the time of this printing.* **master** *runs on Macintosh, PC, CD-RTOS, and Sun platforms.* **rtae** *runs only on the PC.*

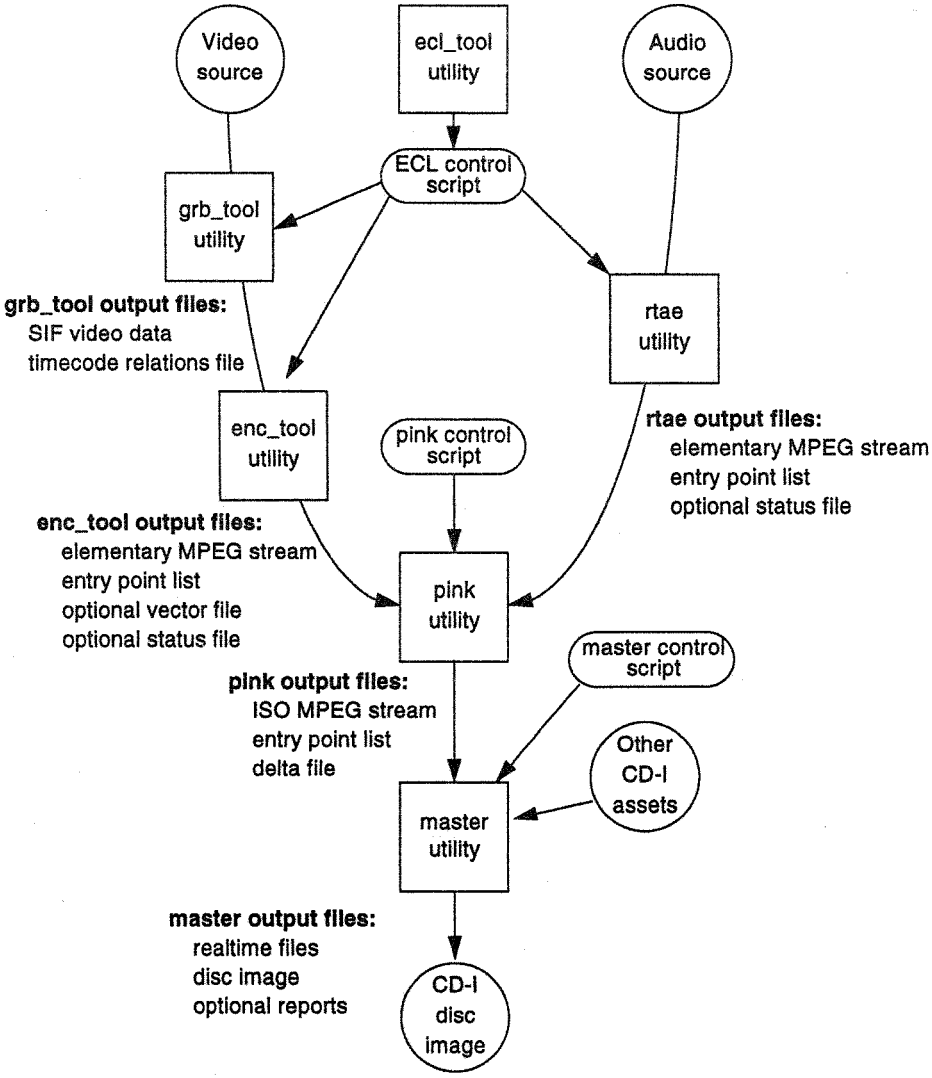


Figure G-1: Mpeg Asset Development Process

Grabbing/Preprocessing

Grabbing is the process of digitizing video sources into a format that can be processed by a computer system. Preprocessing is the process of converting grabbed data to Source Input Format (SIF) data (as described by the MPEG specification).

grb_tool grabs and preprocesses video data by reading an encoding control list (ECL). An ECL can be created using **ecl_tool** or the ECL can be written manually. The ECL contains instructions on how and when to grab video data. Included in this information are the labels for the entry point list.

grb_tool output consists of a SIF video file to be read by **enc_tool** and a timecode relations file used by **enc_tool** to translate VTR timecodes into IFF timecodes.

The realtime audio encoder and decoder (**rtae** and **rtad**) read standard ECLs generated by **ecl_tool** or written manually. **rtae** output can be transferred to the Sun platform and multiplexed with video data to create an audio and video MPEG stream.

Encoding

Encoding is the process of converting SIF files into elementary MPEG streams, and creating entry points in the streams.

The entry points define positions in the audio/video stream where decoding may begin. Decoding is the process of converting MPEG streams into playable data.

enc_tool reads the SIF video file, the encoding control list (ECL) for instructions on how to encode the SIF file, and the timecode relations file generated by **grb_tool**. The **enc_tool** utility generates an MPEG elementary stream file, an entry point list file, and optionally, a statistics file.

NOTE: If entry points are specified in the video and audio ECL, only the video entry points are used by the **pink** *utility. Audio entry points are generated to correspond to the video entry points.*

Multiplexing

Multiplexing is the process of converting elementary MPEG streams into a single ISO11172 MPEG data stream. The **pink** utility does this by adding a system layer to each type of elementary stream (i.e., audio or video) and interleaving the input streams according to their presentation time stamps into a single type of ISO11172 MPEG stream.

A multiplexed FMV data asset may contain two ISO11172 MPEG streams, one for audio and one for video. Each multiplexed stream can consist of up to 16 video and 32 audio elementary streams.

pink reads the elementary streams, entry point lists, and a control script that specifies how to process the input streams. **pink** generates an entry point list and a delta file for the resultant multiplexed MPEG stream.

Premastering

The premastering process merges CD-I assets into a single CD-I disc image file that can be played by a CD-I emulator or sent to a CD mastering facility.

Premastering is accomplished by the **master** utility. **master** formats input into CD-I sectors (according to Green Book specifications) and interleaves the sectors of the various input files into CD-I files. On an administrative level, **master** computes and adds the CD-I file retrieval structure sectors for disc labels, directories, and path table files, and divides the disc volume into tracks.

master accepts the multiplexed streams created by the **pink** utility along with other CD-I assets built through other authoring tools, and merges them according to the instructions in its control script file. **master** output consists of either a realtime file or a disc image. Optionally, **master** generates various reports and maps of the structures built into realtime files or disc images.

Entry points

An entry point specifies a position in an MPEG stream where decoding of video or audio information can begin. For example, a video offset within an entry point specifies an I-picture in the MPEG stream.

An entry point can contain information for up to 16 video multiplexed video streams and up to 32 multiplexed audio streams (one per channel in both cases). The number of streams that can be multiplexed, however, is constrained by the bitrate.

Each entry point can be labeled or unlabeled. Labels are specified in the encoding control list and used by an encoding utility to generate an entry point file that is read by the **pink** utility. Entry point labels can be used by a CD-I application to seek to a specific position in the MPEG stream. **pink** automatically generates unlabeled entry points between labeled entry points. Unlabeled entry points can also be accessed by applications, but the application must reference the unlabeled entry point by the previous labeled entry point + an index value.

pink entry point files

To build an entry point list for an MPEG stream in a realtime file, **master** reads an entry point list created by the **pink** utility. **pink** creates an entry point file for each multiplexed stream it generates. The entry point file consists of a list of entry point offset groups. Each group consists of video and audio offsets that specify the byte offset from the beginning of the corresponding MPEG stream to the entry point.

For example, the following is the beginning of a **pink** entry point list:

```
1 4 54 108
V 0:2356 A 0:1928 SEQ1;
V 0:286514 A 0:334855;
V 0:575694 A 0:617067;
V 0:876217 A 0:910921;
V 0:1170320 A 0:1207099;
V 0:1453629 A 0:1489310;
V 0:1737756 A 0:1773218;
V 0:2038569 A 0:2071720;
V 0:2321347 A 0:2365574;
V 0:2613332 A 0:2650109;
```

The first line is information for **master** to process the entry point list. The remainder of the file consists of groups of offsets. A group of offsets is considered all offsets listed between two semicolons. In the above example, each group contains two offsets: one video (V) and one audio (A). Each offset specifies the elementary stream type and an elementary stream number followed by a byte offset measured from the beginning of the **pink** output stream. Each group of offsets may contain up to 16 video offsets and 32 audio offsets.

A label for an offset group is placed at the end of the group directly preceding the semicolon. In the above example, the first group is given the label **SEQ1**. The entry point groups that follow the first group are unlabeled entry points generated by **pink**.

master entry point lists

master reads the entry point information generated by **pink**, recalculates the offsets, and creates an entry point list that can be written to an external yellow file or included within the first sectors of the realtime file. The entry point information is written in binary format and must be accessed via the data structures provided in the following section. However, a formatted listing of the entry point data generated by **master** can be obtained using the **cdedit** command **infoeps**.

For each group of offsets, **master** calculates the first sector in the file that contains an entry point offset. This is referred to as the start sector for that group of offsets. For each offset in the group, **master** calculates a byte offset from the beginning of the start sector to the beginning of the video or audio data associated with the entry point offset. The byte offset is specific to audio or video bytes. For example, given the following information, **master** calculates the following entry point data:

*data from the **pink** entry point file*

```
V 0:2356 A 0:1928 SEQ1;
```

*is recalculated to the **master** entry point list (formatted by **cdedit**)*

```
SEQ1
--------------------
  index       0  link_to_name       0
    start sector    6849  elem. streams  2
      V  0 offset      52, A  0 offset   1928,
```

The first line of data specifies the entry point label **SEQ1**.

The second line specifies the sequential index number of the offset group (first = 0, second = 1, etc.) and the number of links to the name structure in the name segment of the entry point structure (full details of the entry point structure are given at the end of this appendix). The sequential index starts over with each labeled entry point.

The third line specifies the start sector and the number of elementary streams in the multiplexed stream. In this example, the start sector is 6849 and there are two elementary streams: one video and one audio.

The fourth line specifies the video and audio offsets. In this example, the video data begins 52 video bytes after the beginning of the start sector and the audio data begins 1928 audio bytes after the beginning of the start sector.

NOTE: The above example assumes the entry point information is contained within the realtime file. Start sector values will be different if the entry point data is written to an external file.

Placing entry point data

master does not create entry point data by default; you must specify the **pink** entry point file for **master** to read in the realtime MPEG file definition in the **master** script.

Entry point data structures

master optionally places entry point information in a yellow file in the CD-I disc image or at the beginning of an MPEG realtime file. This information is binary entry point data for a single MPEG multiplexed stream in a realtime file. The entry point information is provided in three or four segments within the file.

Header Segment	Header Structure (1 per entry point list)
Name Segment	Entry Point Name Structure (minimum of 1)
Index Segment (optional)	Index Structure (1 per entry point)
Data Segment	Data Structure (1 per entry point)

Entry point structure definitions

The following structure definitions are used in the following C structures:

```
/* general definitions */
#define EP_VARIABLE_SIZE                    1
#define EP_ODD_SIZE                         1

/* header definition */
#define EP_CURRENT_VERSION                  0x0001

/* name segment definitions */
#define EP_NAME_SEG_START_OFFSET            256
#define MAX_EP_NAME_LENGTH                  32
#define MAX_EP_NAME_SEG_STRUCT_SIZE         44

/* Data Structure Definitions */
#define EP_AV_STREAM_MASK                   0x1f
#define EP_AV_TYPE_MASK                     0x80
#define EP_AV_TYPE_AUDIO                    0x00
#define EP_AV_TYPE_VIDEO                    0x80

#define EP_MAX_VIDEO_STREAMS                16
#define EP_MAX_AUDIO_STREAMS                32

#define MAX_EP_DATA_STRUCT_SIZE             302
```

Entry point header segment

The entry point header contains one 256-byte structure. Both a graphic representation and the C structure for the entry point header are provided below.

```
struct ep_header_seg_struct
{
    unsigned char    version;
    char             HEADER_CODE=0x69;
    unsigned long    name_seg_size;
    unsigned long    index_size;
    unsigned long    data_seg_size;
    unsigned long    ep_size;
    unsigned long    ep_mpeg_gap;
    long             av_timebase_offset;
    unsigned char    RESERVED_B[230];
    /* RESERVED: Will Be 0x00 */
};
```

Byte Offset

0	**Entry point file version**	1 byte
1	**Header code** = 0x69	1 byte
2	**Name segment size**	4 bytes
6	**Index segment size**: zero if absent	4 bytes
10	**Data segment size**	4 bytes
14	**Length of entry point data**: zero, if yellow file	4 bytes
18	**Size of gap** (in sectors):	4 bytes
22	**Audio-Video timebase offset** = 0x00000000	4 bytes
26	**RESERVED**: contains 0x00 values	230 bytes
256		

Entry point group name segment

The entry point group name segment contains a series of name structures (minimum one) containing the byte offset to the entry point group data (relative to the start of the data segment), the audio and video stream numbers containing the entry points, and the length of the name (and the name) of the entry points group. Both a graphic representation and the C structure for the entry point name structure are provided below.

```
struct ep_name_seg_struct
{
    long            group_first_data_offset;
    /* MPEG Entry Point Information */
    unsigned long   audio_streams;      /* Bit Fields */
    unsigned short  video_streams;      /* Bit Fields */
    char            group_name_size;
    char            group_name[ group_name_size ];
};
```

Byte Offset

0	**Byte offset to data for entry point group**	4 bytes
4	**Audio stream used by this group: bit 0 = stream 0, bit 1 = stream 1,...**	4 bytes
8	**Video streams used by this group: bit 0 = stream 0, bit 1 = stream 1, ...**	2 bytes
10	**NAM_SIZ: Byte length of entry point name + '\000' + null byte padding**	1 bytes
11	**Entry point name + '\000' + null byte padding**	NAM_SIZ
NAM_SIZ + 11		

Entry point index segment

The entry point index segment contains an array of access pointers. Each access pointer is a byte offset from the start of the name segment to the corresponding name structure. The array is sorted in alphabetical order of the labels of the corresponding name structures. The C structure for the entry points group index segment is provided below.

```
long            name_index[ ]
```

NOTE: If you specify **with index** *in the entry point list specification in the* **master** *script,* **master** *builds an index of the entry point names. If you do not specify* **with index**, *no index is built and* **index_size** *= 0 in the header segment structure.*

Entry point data segment

The entry point data segment contains a series of entry point group data structures containing offsets to the entry point group name, the previous entry point group, the next entry point group, the number of entry point offsets in the group, the starting sector for the entry point group, and the audio/video offset itself (stream type, stream number and byte offset within the stream). Both a graphic representation and the C structures for the entry point group data segment are provided below.

```
struct ep_av_offset_struct
{
    char            stream_id;
    char            RESERVED;
    unsigned long   offset;
};

/* Combined Entry Point Data */
struct ep_data_seg_struct
{
    long            group_name_offset;
    short           prev_rel_offset;
    short           next_rel_offset;
    char            num_mpeg_offsets;
    char            RESERVED;
/* MPEG Entry Point Information */
    long            start_sector;
    struct          ep_av_offset_struct offset[ EP_VARIABLE_SIZE ];
};
```

Entry point group data structure

Byte Offset		
0	**Byte offset to entry point group name**	4 bytes
4	**Relative Byte Offset To Previous Entry Point Data Structure (negative or zero, zero = first)**	1 byte
6	**Relative Byte Offset To Next Entry Point Data Structure**	4 bytes
8	**NUM_OFST: Number of elementary streams with offsets for entry point group**	1 byte
9	**RESERVED:** contains 0x00	1 byte
10	**Starting disc sector for entry point group**	4 bytes
14	**Offsets to video and audio streams**	NUM_OFST*6 bytes
	⋮	
NUM_OFST*6+14		

Entry point stream offset structure

Byte Offset		
0	**Stream ID** **bit 7 = stream number (0=Audio, 1=Video)** **bit 5 and 6 = Reserved** **bit 0-4 = Stream number (audio: 0-31; video: 0-15)**	1 byte
1	**RESERVED:** contains 0x00	1 byte
2	**Byte offset in stream-type bytes to entry point from starting disc sector for this entry point**	4 bytes
6		

Appendix H master Error Messages

too many parameters [%s]

%c is an invalid option

no script file given

no ready file given

error opening "%s" as input

error opening "%s" as output

error seeking in file "%s"

error reading from file "%s"

error writing to file "%s"

error closing "%s"

dynamic memory shortage

ident too long - truncated to %s...

Identifier length is limited to 32 characters.

string too long - truncated to %s...

String length is limited to 128 characters.

closing %c missing in %s

include files too deeply nested (max %d)

syntax error

near [%s], expecting %s...

The given lexical token has caused a syntax error. The token list included in this error message specifies acceptable tokens at the position in the script source file causing the error. The compiler, using an elementary recovery scheme, skips some tokens until parsing can resume.

%s multiply defined

A special simple file such as **abstract, copyright, biblio,** or **application** has been defined more than once in a given volume.

unknown label type

internal error

unknown file attribute type

internal error

%s file has not been defined

One of the special simple files such as **abstract, copyright, biblio,** or **application** has not been defined in a given volume.

%s file has been defined more than once

One of the special simple files such as **abstract, copyright, biblio,** or **application** has been defined more than once in a given volume.

file ident %s multiply defined

The given identifier has already been used to describe another file.

file ident %s never defined

No file has been described using the given identifier.

file ident %s already entered in a directory

The file whose identifier is given has already been declared in the directory section.

file ident %s was not entered in any directory

The file whose identifier is given has not been used in the directory section.

%d is an invalid CDI channel number

An invalid channel number was used. The CD-I channel range for realtime audio streams is 0 to 15. For other streams, the channel range is 0-31.

invalid minute [%d] in lsn - should be < 99

invalid second [%d] in lsn - should be < 60

invalid frame [%d] in lsn - should be < 75

duplicate file name "%s"

The given file name is already entered in the same directory level.

%s track will be less than 4 seconds

more than 99 tracks in volume "%s"

More than 99 red files were declared in a given volume.

volume "%s" would exceed CD maximum capacity (99:59:74)

channel %d already contains a %s stream

More than one stream of the given type has been defined in the given channel.

sources must be given in increasing order of starting/ending time

%s file lost !!!...

internal error

realtime audio stream sources must have same bandwidth requirements

insufficient CD bandwidth for file %s

Realtime audio streams saturate the CD bandwidth in the given file.

unable to satisfy "%s" time requirement in file %s

source type does not match stream type

internal error

invalid file name ["%s]

An invalid file name was used. The file name must use characters from the legal characters set.

file name ["%s] too long - should be <= 28

An invalid file name was used. The file name length must be less than 29 characters.

invalid file protection [0x%1x]

Write access was requested.

source "%s" would start before previous source in file %s

coding invalid for "%s" source

An inappropriate coding was used for a source of a given stream type.

events would remain in GREEN file %s

Triggers or EORs were requested after the end of the given realtime file.

no source for MESSAGE areas of volume "%s"

at clause is incompatible with realtime audio stream

cannot specify both starting and ending time

Only one of the starting or ending positioning clauses can be specified for given source specification.

no index allowed on root wrapper

There is only one main or root wrapper chunk in an IFF file; consequently, the only allowed index value is 1.

IFF id [%s] too long (should be less or equal 4)

null IFF chunk index (should be > 0)

source "%s" must be specified implicit

Implicit is the only source specification form allowed in this context.

IFF source "%s"%s must be fully qualified

source "%s" is not an IFF file

An inconsistency has been found while analyzing the structure of an IFF file (ID characters outside the graphic characters set, file size, and chunk size inconsistency).

inconsistent chunk length in IFF source "%s"

A chunk has been found that could not fit into the enclosing chunk or file.

error looking up "%s"%s chunk

The given chunk was not found in the given file.

"%s"%s is not a valid VIDEO source specification

"%s"%s is not a valid AUDIO source specification

IFF source "%s" is not a stereo PCM source

samples are not coded on 16 bits in IFF PCM source "%s"

sample rate is not 44.1 KHz in IFF PCM source "%s"

SSND offset is past end of SSND chunk in IFF PCM source "%s"

more than 2 channels in "%s"%s ADPCM source

invalid sample size in "%s"%s ADPCM source

invalid sample rate in "%s"%s ADPCM source

%s source "%s" is incompletely specified

no coding allowed for DATA sources

incompatible field coding keywords type

conflicting field coding keywords

main coding field keyword missing

invalid IMAG IHDR chunk size [%d] in IFF source "%s" (should be >= 14)

invalid IMAG IHDR model [%d] in IFF source "%s"

The IHDR model field of an IMAG IFF file has been found outside the range 0-10.

IHDR image size parameters inconsistent with IDAT chunk size in IMAG source "%s"

invalid AIFF COMM chunk size [%d] in IFF source "%s" (should be >= 18)

invalid AIFF AESD chunk size [%d] in IFF source "%s" (should be >= 24)

AIFF AESD chunk ignored in IFF source "%s"

AIFF BlockSize ignored in IFF source "%s"

unsupported coding model in IMAG IFF source "%s"

The IHDR model field of an IMAG IFF file was either IFF_MDL_RGB888, IFF_MDL_CLUT3, or IFF_MDL_PLTE.

invalid specific coding value [%d]

INDEX

F

Q

R

S

T

V

W

Y