master Script Specification

Script file syntax overview

The following sections describe the `master` script syntax in detail. The syntax sections are organized into five main sections: the album definition, the volume definition, the file definition, the PSD 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`.
- **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.

- **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.

- **vcd files**
  These files are MPEG files used in creating VCD discs and Karaoke-compliant VCD discs.

- **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.

The PSD definition section contains subsections covering the two types of psd lists supported by master:

- selection lists
- play item lists

Script file syntax

The `master` script file uses the following syntax when building CD-i disc images:

[`options` *[:] option...]*
album_definition
{volume_definition
file_definition...
[psd_definition]
directory_definition}...

where

option
is a `master` runtime option. These options are defined in the *master and VCDmaster* manual.

album_definition
is the album definition element.

volume_definition
is a volume definition element.

file_definition
is a file definition element.

psd_definition
is the Play Sequence Descriptor (PSD) definition element.

directory_definition
is a directory definition element.

`master` uses the following syntax when building single realtime files:

[`options:` *option]*
record_definition...

where

option
is a `master` runtime option. These options are defined in the *master and VCDmaster* manual.

record_definition
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.

pathlist
: is a quoted string that specifies a file. *pathlist* may be either a full pathlist or an individual file name.

asset
: specifies an IFF file or IFF chunk. *asset* uses the following forms:

  pathlist
  pathlist>*chunkpath*.

rtl is a realtime label.

rsn is a relative sector number.

sn is a sector number.

expression
: is an expression that evaluates to an integer.

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

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_definition)* 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, VCD, 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 volume definition *(volume_definition)* 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. If defining a VCD volume, a sequence of psd definitions is specified before the 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` *volume_name* `in` *pathlist [volume_option...]*

where

volume_name
is a *string* specifying the volume name. *vol_name* is optional in CD-DA scripts.

pathlist
is a pathlist specifying the output file to contain the resulting disc image.

volume_option
is one of the following:

`disc_restrict` *{`0`|`1`|`2`|`3`}*
indicates the level of play restrictions. Implementations of play restrictions are supported on an application-specific basis.

`strt_nxt_l1d1`
indicates that if a PSD is defined for the multi-disc album, then when a new disc has the same album information as the last disc played, then begin playing using selection list 2, instead of list 1. If the album is a single volume, this parameter is ignored.

`strt_nxt_trk2`
: indicates that if no PSD is defined for the multi-disc album, then when a new disc has the same album information as the last disc played, then begin playing using track 2 instead of track 1. If the album is a single volume, this parameter is ignored.

Volume definition examples

```
volume “FMV1” in “fmv_image”
volume “PhotoCD” in “photo_cd”
volume “Build Demo” in “cdimage_xa”
volume “VCD1” in “disc1_vcd”
volume “VCD2” in “disc2_vcd” strt_nxt_l1d1
```

File definitions

A file is defined by a file-type, an identifier used as a file reference in the directory structure definition, and the source (or source list) on which the file depends. There are three types of 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 `red`, `green` (simple), `yellow`, `abstract`, `copyright`, `biblio`, and `application` files also fall into this category.

  There are two special simple file definitions that do not use identifiers and are not specified in the directory structure definition:

  - **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 does not appear as a file in any `CDFM` directory.

- **vcd files**
  These files are MPEG files used in creating VCD discs and Karaoke-compliant VCD discs.

- **realtime green 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.

  Green files pre-mastered by `master` or an equivalent premastering program are interpreted as simple files.

Simple file definitions

Simple files are data files that are dependent on only one source. They are recognized by their file-type specifier (i.e., `red`, `yellow`...). Though the syntax is nearly the same for the different types of simple file, there are some variations. The following discussion provides the overall syntax. Following this section, the specific file types and their definition syntax 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 pathlist          ! 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 pathlist                ! abstract file
application file ident from pathlist          ! application file
biblio file ident from pathlist             ! bibliographic file
copyright file ident from pathlist            ! copyright file
```

where

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

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

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

  pathlist
  pathlist>chunkpath

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

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

Simple green file definition

Simple green files are pre-mastered realtime files. Premastered realtime file definitions use the following syntax:

`green file` *ident* `from` *pathlist*

where

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

pathlist
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:

pathlist
pathlist>chunkpath

pathlist 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* *[*`rounded`*]* `from` *source*

`abstract file` *ident* *[*`rounded`*]* `from` *pathlist*

`biblio file` *ident* *[*`rounded`*]* `from` *pathlist*

`copyright file` *ident* *[*`rounded`*]* `from` *pathlist*

`application file` *ident* *[*`rounded`*]* `from` *pathlist*

where

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

`rounded`
: specifies that the yellow file must be sector aligned (on 2048 byte sectors).

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

  pathlist
  pathlist>*chunkpath*

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

pathlist
: 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.

Only a white-book compliant application can be used for creating VCD discs. It must be the file specified by the application file definition.

Yellow file definition examples

```
yellow file      pic1     rounded from “VIDEO/pic1.dyuv”

copyright file   Copyright from “txt/copyright.txt”

abstract file    Abstract  from “txt/ab.txt”

application file Appl     from “Vcd_app/CDI_VCD_APP”
```

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` *pathlist* *! application_use area*

where

pathlist
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

A message area file is a PCM file that is placed at the beginning of a disc image and at the beginning of a realtime file if red files were included in it. The message area file warns the user if she places a CD-i disc in a CD-DA player.

Because of the disc image layout of VCD discs, message area definitions are not allowed in `vcdmaster` scripts.

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:

  pathlist
  pathlist>chunkpath

  pathlist 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"
```

VCD file definitions

There are three types of VCD file definitions:

- audio-video sequence
- segment files
- karaoke files

Audio-video sequences

Audio-video sequences use the following definition:

```
avseq_dat file av_ident from mpeg pathlist
    [scan_list file sc_ident [spaced spacing] [verbose]]
    {[chapter_marks [first_source_tc SMPTE]
    [tag {source_tc | mpeg_tc} SMPTE]... | [ep_list from epfile]}
```

where

av_ident
: is a file identifier used to refer to the MPEG file in the directory structure definition.

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

sc_ident
: is a file identifier used to refer to the scanlist file generated by `vcdmaster`. A directory structure entry that references the identifier must be included in the directory structure definition.

`verbose`
: forces vcdmaster to output the SMPTE timecode, sector number, and byte offset for each entrypoint in the MPEG file.

spacing
: specifies the spacing of the scanpoints. If not included, scanpoints are speced every 2 seconds. *spacing* is given in

seconds and may be specified by any decimal number (e.g., 2.125). `vcdmaster` rounds spacing down to the nearest 1/8 second.

SMPTE
is a SMPTE timecode specifying the timecode of a chaptermark location in the MPEG file.

tag
is a quoted string that references a chaptermark. A *tag* does not have to be unique to the file specification and is not included in any directory structure definition element.

epfile
is a quoted filename or pathlist specifying the entrypoint file generated by the `pink` utility.

VCD files must be at least 4 seconds in duration (excluding the leadin). If the file is too short, it is padded with empty sectors up to four seconds and `vcdmaster` generates a warning.

You can specify up to 98 `avseq_dat` files in a single volume.

VCD 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) and the end-of-record (EOR) bits are cleared in every sector except the last, in which they are set.

The byte length of a VCD 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.

Including entrypoints

A white-book compliant VCD application only recognizes VCD files with at least one entrypoint; consequently, `vcdmaster` generates one entrypoint for each VCD file, by default. `vcdmaster`

includes additional named entrypoints from a specified entrypoint file *epfile* or from the `chapter_mark` specification.

The maximum number of additional named entrypoints allowed per VCD file is 98. The total number of entrypoints for the entire VCD volume allowed by a white-book compliant VCD application is 500.

After processing 98 named entrypoints for a VCD file, `vcdmaster` ignores the remaining named entrypoints.

After processing 500 entrypoints for a VCD volume, `vcdmaster` ignores any remaining named entrypoints.

Warning: If you exceed the 500 named entrypoint limit, `vcdmaster` will process the remaining VCD files and place them in the disc image, but `vcdmaster` will not generate the default entrypoints for them. Therefore, the files without entrypoints are not accessible from the VCD application.

Note: The `chapter_mark` specification and the `eplist` specification are mutually exclusive.

Use the `scan_list` specification to generate a list of scanpoints equally spaced throughout an MPEG file. Scanpoints are used by VCD applications as unnamed entrypoints. They essentially specify the frames displayed during scan forward and scan backward. By default, the scanpoints are placed approximately every 2 seconds.

Use the `spaced` keyword to change the spacing to your title requirements. `vcdmaster` generates a scanlist that places the scanpoints as close as possible to the *spacing* parameter. Because I-pictures are not always regurlarly spaced due to forced scene breaks, etc., commonly the spacing of scanpoints vary up to 1/4 of a second.

The scanlist file generated by `vcdmaster` must be included in the directory structure definition. Refer to the *Directory Structure Definition* section of this chapter.

Use the `verbose` keyword to force `vcdmaster` to output the SMPTE timecodes, sector number and byte offsets of each I-frame in the MPEG file. This is useful if you do not know the specific SMPTE timecodes for chaptermarks.

Use the `chapter_mark` specification to specify the named entrypoints to be included in the `ENTRIES.VCD` file. The `chapter_mark` specification allows you to specify chaptermark locations by source timecodes or MPEG timecodes. Depending on how the MPEG file was encoded, the source and MPEG timecodes may be identical or vary by a specific time unit. If the MPEG file uses timecodes that begin at 00:00:00:00 and your source timecodes do not, you can specify a first time code that is substracted from all subsequent source time codes in the list. If you do not specify a `first_source_tc` timecode, `vcdmaster` interprets `source_tc` timecodes and `mpeg_tc` timecodes identically.

The only restrictions for chaptermarks are that the timecodes must be specified in ascending order and that the adjusted timecodes are within the timecodes of the MPEG file.

If you specify timecodes in non-ascending order. vcdmaster places the offending entrypoint at the next available I-picture after the previous entrypoint. If you specify a timecode that is not within the MPEG file, the offending entrypoint and all subsequent entrypoints are not included.

Use the `eplist` specification to read the entrypoint file *epfile* generated by `pink`. `vcdmaster` will only include named entrypoints from the *epfile*. If your *epfile* does not include named entrypoints, you must either edit the *epfile* to include tags for the required entrypoints or use the `chapter_marks` specification.

◆

Note: Each named entry point included by `vcdmaster` is placed in an `INDEX` entry in the TOC file generated by `vcdmaster`. You do not have to include any special directory structure definition element for the `vcdmaster`-generated `ENTRIES.VCD` file.

VCD Segment file definitions

The VCD White book 2.0 specification allows files to be included on the volume in the segment play item area. Segment files are not specified in

Use the following syntax for a segment file definition:

`seg_play_item file` *ident* `from` *segment_type pathlist*
[`autopause` *{*`at` *rsn [*`every` *sn]}...]*

where

ident
is an identifier specifying the segment file.

segment_type
is the VCD file type, specified by one of the following:

`wb_still`
is a PAL or NTSC dimensioned MPEG still.

`wb_still_seq`
is a series of MPEG still pictures multiplexed according to White Book standards. An MPEG still sequence may contain audio.

`wb_a`
is an audio-only MPEG stream multiplexed according to White Book standards.

`wb_v`
is a video-only MPEG stream multiplexed according to White Book standards.

`wb_av`
is a MPEG stream consisting of one audio and video stream multiplexed according to White Book standards.

For complete information on allowable picture size, bitrates, and modes, refer to the White Book 2.0 V4.0.

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

`autopause at` *rsn*
: sets a trigger bit at the sector specified by *rsn*. When the trigger bit is encountered, the playback of the segment file pauses.

`every` *sn*
: is the sector interval for setting periodic trigger bits beginning with the sector specified by *rsn*.

Karaoke-compliant VCD discs

If you want to make a Karaoke-compliant VCD disc, you must also include the following file specification:

`karinfo_data from` *pathlist*

where

pathlist
: is a quoted file name or pathlist specifying the `karinfo_data` file generated by the `texcon` utility.

The `karinfo_data` file specification may be defined before or after you define the `avseq_dat` files. You can only define one `karinfo_data` file per volume.

For more information about creating Karaoke-compliant VCD discs, refer to the *VideoCD Toolkit* manual and the *Using Texcon* manual.

VCD file definition examples

```
karinfo_data from “KARINFO.kia”

seg_play_item file nasa_one
    from wb_av "nasa_kar1.mmd" auto_pause at 00:03:15

seg_play_item file nasa_two
    from wb_av "nasa_kar2.mmd" auto_pause at 75 every 300

seg_play_item file oibump
    from wb_av "oibump.mmd"
    auto_pause at 00:01:00 at 00:03:00 at 00:07:00

avseq_dat file   music02  from “MPEG/Project1/clip02.mmd”
    ep_list from “clip2.pkl”

avseq_dat file   music03  from “MPEG/Project1/clip03.mmd”
    scan_list file scn1 verbose

avseq_dat file   music03  from “MPEG/Project1/clip03.mmd”
    scan_list file scn1 spacing 2.125
    chapter_marks
    “mick”       mpeg_tc 00:10:15:00
    “keith”      mpeg_tc 00:15:24:12
    “bill”       mpeg_tc 00:24:48:00

avseq_dat file     music03 from “MPEG/Project1/clip03.mmd”
    scan_list file scn1 spacing 2.125
    chapter_marks  first_source_tc 6:00:00:00
    “mick”         source_tc 06:10:15:00
    “keith”        source_tc 06:15:24:12
    “bill”         source_tc 06:24:48:00
```

Realtime file definition

A realtime file definition consists of an exact specification of the how the sources that make up the realtime file are organized. Realtime sources are either sequentially laid out or interleaved together.

A realtime file definition consists of three layers of specification: the realtime record, the streams that make up the record, and the sources (assets) that make up each stream. Additionally, you may specify the placement of trigger bits and end-of-record bits in the realtime file using event definitions.

Each realtime file definition must have at least one record that is specified by at least one stream, made up of at least one source. Event definitions are optional.

Realtime file definitions use the following syntax:

`green file` *ident* `from` *record_definition...*

where

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

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

`block` *[rtl[, rtl...]] stream_definition...*
`record` *[rtl[, rtl...]] stream_definition...*

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

stream_definition
is a stream definition. Stream definitions are explained in the following sections.

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.

There are five types of stream definitions: `audio`, `mpeg`, `video`, `data`, and `event`. Their are two sub-types of `audio` and `mpeg` streams: `real_time` and `non_real_time`. Any realtime record may be made up of multiple streams of different types.

Each stream, in turn, is made up of one or more sources. There are four types of source definitions that correspond to the stream definitions: `audio`, `mpeg`, `video`, and `data`. A source definition must correspond to its stream type.

Each of the stream and source types are defined in the following sections.

MPEG stream definitions

An mpeg stream definition uses the following syntax:

{`non_real_time | real_time`*}* *[*`still`*]* `mpeg` *[rtl[, rtl...]]* *[*`at` *rsn]*
`in channel` *mpeg_channel* `from` *source [, source...]*

where

`real_time`
specifies that the MPEG data should be gapped to meet the demands of the MPEG decoder.

`non_real_time`
specifies that the MPEG data should be packed into a minimum number of sectors.

`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.

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.

mpeg_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 below.

MPEG source specifications

MPEG sources are specified using the following form:

[rtl:*...] asset [{*`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 multiplexed mpeg file. *asset* is specified as a quoted pathlist.

`at | by` *rsn*
specifies how to place *asset* in the stream. *rsn* indicates the time, relative to the beginning of the stream, `at` which to place the first sector of *asset* or `by` which to place the last sector of *asset*.

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.

Audio stream definitions

An audio stream definition uses the following syntax:

`real_time audio` *[rtl[, rtl...]]* *[*`at` *rsn]*
`in channel` *audio_channel* `from` *source [, source...]*

or

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

where

`real_time`
indicates that the audio data should be gapped to meet the demands of the audio decoder.

`non_real_time`
indicates that the audio data should be packed into a minimum number of sectors.

`packed`
indicated that the audio data of the stream should begin with the first available byte. If `packed` is not specified, data begins with 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.

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. *rsn* is given as a time-clause relative to the beginning of the record.

`at` *rsn*
is a relative sector number that specifies the starting sector number of the first sector allocated for the stream. *rsn* is given as a time-clause relative to the beginning of the record.

audio_channel
is the channel number to place the audio stream in. *audio_channel* must be in the range of 0..15. Each *audio_channel* must be unique within the realtime record.

source
is the source definition for the stream and is defined on the following pages.

Audio source specifications

Audio sources are specified using one of the following forms:

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

or

[`at` *rsn]* `silence` *sn* `[`*type[*`emphasis`*]*`]` *[{*`at`|`by`*} rsn]*

where

rtl is an optional 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.

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 the location and type of audio file. *asset* uses one the following formats:

pathlist`[`*type* `[emphasis]]`
filename`>`*chunkpath*

where

pathlist

is a quoted string that specifies the source file.

type

indicates the type of audio contained in the file and is one of the following:

```
mono_a
mono_b
mono_c
stereo_a
stereo_b
stereo_c
```

`emphasis`

indicates whether emphasis is used.

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
: is the byte offset from the beginning of *asset* at which to begin including data. If *offset* is greater than the length of the *asset*, `master` returns an error.

len
: is the byte length of the data to include starting from *offset*. If *offset* and *length* are greater than the length of the *asset*, `master` issues a warning and only includes the data from *offset* to the end of *asset*.

`at` | `by` *rsn*
: specifies how to place *asset* in the stream. *rsn* indicates the time, relative to the beginning of the stream, `at` which to place the first sector of *asset* or `by` which to place the last sector of *asset*.

`at` *rsn*
: is the sector number, relative to the beginning of the record, at which to place the first sector of silence. *rsn* is given as a time-clause relative to the beginning of the record.

`silence`
: indicates that empty audio sectors (silence) are allocated for the stream instead of audio data.

sn [*type*]
: is the number of sectors (*sn*) to be filled with silence. The sector-type is specified by *type* (as defined above).

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.

Video stream definitions

Video stream definitions use the following syntax:

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

where

`packed`
indicates that the video data of the stream should begin with the first available byte. If `packed` is not specified, data begins with 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.

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.

`at` *rsn*
is a relative sector number that specifies the starting sector number of the first sector allocated for the stream. *rsn* is given as a time-clause relative to the beginning of the record.

data_channel
is the channel number to place the video stream in. *data_channel* must be in the range of 0..31. Each *data_channel* must be unique within the realtime record.

source
is the source definition for the stream and is defined on the following pages.

Video source specifications

Video sources are specified using the following form:

[rtl:*...] asset [*`mask` *pattern] [*`offset` *offset* `length` *len] [{*`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.

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 the location and type of video file. *asset* uses one the following formats:

pathlist[*video_type*]
pathlist>*chunkpath*

where

pathlist is a quoted string that specifies the source file.

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

video_type indicates the type of video contained in the file. *video_type* can be specified using pre-defined keywords or by using a hexadecimal or decimal number.

The following syntax is used to define *video_type* with keywords:

type[resolution][scan_lines]

where

type is one of the following and maps to bits 0-3 of *video_type*:

`clut4`	(bit value: 0000)
`clut7`	(bit value: 0001)
`clut8`	(bit value: 0010)
`rl3`	(bit value: 0011)
`rl7`	(bit value: 0100)
`dyuv`	(bit value: 0101)
`rgb555`	(bit value: 0110)
`rbg555u`	(bit value: 0111)
`rgb555`	(bit value: 1000)
`qhy`	(bit value: 1001)

resolution indicates the resolution of the video and maps to bits 4-5 of *video_type*:

`normal`	(bit value: 00)
`double`	(bit value: 10)
`high`	(bit value: 11)

scan_lines indicates whether the beginning scan line of the video is odd or even and maps to bit 6 of *video_type*:

`even`	(bit value: 0)
`odd`	(bit value: 1)

The following syntax is used to define *video_type* directly:

`specific` *mask*

where

mask is either a decimal or hexadecimal number in the range of 0..255 (0x0..0xFF). *mask* is interpreted using the bit settings defined above.

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` | `by` *rsn*

specifies how to place *asset* in the stream. *rsn* indicates the time, relative to the beginning of the stream, `at` which to place the first sector of *asset* or `by` which to place the last sector of *asset*.

If no *rsn* clause 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[, rtl...]] [*`at` *rsn]*
`in channel` *data_channel* `from` *source [, source...]*

where

`packed`
indicates that the data of the stream should begin with the first available byte. If `packed` is not specified, the data begins with 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.

`at` *rsn*
is a relative sector number that specifies the starting sector number of the first sector allocated for the stream. *rsn* is given as a time-clause relative to the beginning of the record.

data_channel
is the channel number to place the data stream in. *data_channel* must be in the range of 0..31. Each *data_channel* must be unique within the realtime record.

source
is the source definition for the stream and is defined on the following pages.

Data source specifications

Data sources are specified using the following form:

[rtl:*...] asset [*`mask` *pattern] [*`offset` *offset* `length` *len] [{*`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

where

pathlist is a quoted string that specifies the source file.

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

pattern
is the 16-bit mask pattern that is mapped to repeated 16 sector groups. Each bit in *pattern* is mapped one-to-one to a sector in the group. The least significant bit of *pattern* is mapped to the first sector in the group. Sectors mapped to zeros in *pattern* are filled with data from the asset.

offset
is the byte offset from the beginning of *asset* at which to begin including data. If *offset* is greater than the length of the *asset*, `master` returns an error.

len is the byte length of the data to include starting from *offset*. If *offset* and *length* are greater than the length of the *asset*, `master` issues a warning and only includes the data from *offset* to the end of *asset*.

`at` | `by` *rsn*
specifies how to place *asset* in the stream. *rsn* indicates the time, relative to the beginning of the stream, `at` which to place the first sector of *asset* or `by` which to place the last sector of *asset*.

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

Event definitions

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

An event definition uses the following syntax:

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

where

event
is the type of bit to set. *event* must be one of the following:

`triggers`	sets a trigger bit
`eors`	sets an EOR bit

rsn specifies the sector in which to set a trigger or EOR bit. *rsn* may be specified in the following formats and is relative to the beginning of the stream:

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

where

ref_ident
is a realtime label previously defined in the script. The `.first` and `.last` specification indicate that the event bit should be set in the first or last sector of the stream or source specified by *ref_ident*.

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

PSD definition

All White Book 2.0 volumes must have a Play Sequence Descriptor (PSD) defined. The PSD definition (psd_definition) consists of a series of list definitions. There are two types of list definitions:

- Play item lists
- Play sequence lists

All file definitions must precede the list definitions. Both types of lists reference VCD file identifiers defined in the file specification portion of the master script. PSD definitions do not reference files directly.

Play item list definition

A play item list uses the following syntax:

```
play_list ident
    [next {n_ident [hotspot] | off}]
    [previous p_ident [hotspot]]
    [return r_ident [hotspot]]
    [playing_time {all | time}]
    [after_play_wait {rsn | forever | none}]
    [auto_pause_wait {rsn | forever | none}]
    {play_item pi_ident}...
```

where

`playlist` *ident*
: is an identifier specifying the playlist.

`next` *n_ident*
: is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the next function of the play list.

`next off`
specifies that no next play item is used for the play item list.

`previous` *p_ident*
is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the previous function of the play list.

`return` *r_ident*
is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the return function of the play list.

hotspot
are the upper-left and lower-right screen coordinates for the active area. The coordinates are given in White Book coordinate units (0..255), with the origin in the upper-left corner of the image. The coordinates are scaled to the source image size. The format for *hotspot* is:

x1, y1, x2, y2 upper-left, lower-right corners

`playtime all`
specifies that each play item in the list is played in its entirety.

`playtime` *time*
specifies that each play item in the list is played for the time specified by *time*. *time* is given is 1/15 second units.

`after_play_wait` *rsn*
specifies the wait time (*rsn*) after playing a play item from the list before playing the next play item.

`after_play_wait forever`
indicates to wait after playing a play item from the list until some user interaction specifies to play another list.

`after_play_wait none`
indicates there is no wait time after playing a play item from the list before playing the next play item.

`auto_pause_wait` *rsn*
: specifies the wait time (*rsn*) when encountering a pause trigger in the current play item before continuing the playback.

`auto_pause_wait forever`
: indicates to stop until a user action specifies to begin play when encountering a pause trigger in the current play item.

`auto_pause_wait none`
: indicates to not puase when encountering a pause trigger in the current play item before continuing the playback.

`play_item` *pi_ident*
: specifies an audio-video sequence, segment, or red file that has been previously defined in the master script.

Sequence list definition

A sequence list uses the following syntax:

`sel_list` *sl_ident* `play_item` *pi_ident*
[repeat]
[default]
[`next` *{n_ident [hotspot]* | `off`*}]*
[`previous` *p_ident [hotspot]]*
[`return` *r_ident [hotspot]]*
{`selection` *selection_number pi_ident [hotspot]}...*

where

sl_ident
: is an identifier specifying the selection list.

pi_ident
: is an identifier specifying an audio-video sequence, segment, or red file previously defined in the master script.

repeat
: specifies the number of times to repeat playing the

selection list play item and the action to take after the specified number of repeats. *repeat* has two forms:

```
repeat forever [finish]
repeat num [finish] [thenwait {rsn | forever | none}]
    then {random_sel | play_item pi_ident}
```

where

`forever`
: indicates that the play item should be repeated until a selection is made.

`finish`
: indicates that when a selection is made the play item should finish playing before beginning to play the selection.

num
: specifies the number of times to repeat the play item.

`wait` *rsn*
: specifies the wait time (*rsn*) after repeating the play item *num* times before the playing a random selection or the specified selection..

`wait forever`
: indicates to wait for a selection after repeating the play item *num* times.

`wait none`
: indicates to immediately play a random selection of the specified selection after repeating the play item *num* times.

`random_sel`
: indicates to play a random play item from the selection list after repeating the play item *num* times and, if specified, waiting the specified time.

default
: specifies the default play item to play when the `wait`

period specified in the `repeat` clause is over. *default* has three forms:

```
default off
default pi_ident [hotspot]
default multi_default [md_hotspot] [no_numeric_sel]
```

where

`default off`
: indicates there is no default selection.

`default` *pi_ident*
: specifies the default play item.

hotspot
: specifies the hotspot coordinates of the default play item.

`default multi_default`
: indicates to use multi-default mode when playing selections from the list.

md_hotspot
: specifies the coordinates of the multi-default hotspot. Clicking on this hotspot is equivalent of selecting the default action during selection play.

`no_numeric_sel`
: indicates that no numeric selections are available for this selection list. The only access to the play items in the list is through the multi-default selections.

`next` *n_ident*
: is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the next function of the selection list.

`next off`
: specifies that no next play item is used for the selection list.

`previous` *p_ident*
: is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the previous function of the selection list.

`return` *r_ident*
: is an identifier specifying a segment, audio-video sequence, or red file. This file becomes associated with the return function of the selection list.

hotspot
: are the upper-left and lower-right screen coordinates for the active area. The coordinates are given in White Book coordinate units (0..255), with the origin in the upper-left corner of the image. The coordinates are scaled to the source image size. The format for *hotspot* is:

  x1, y1, x2, y2 upper-left, lower-right corners

selection_number
: specifies the selection number of the play item in the list. The selection numbers must be sequential within the range of 1..99. If multi-default mode is specified, the first selection number of the list must be 1.

Example PSD definitions

```
play_list ExpertLesson3
    next ExpertMenuStill
    return ExpertMenuStill
    after_play_wait 2
    auto_pause_wait 20
    play_item still_img0004
    play_item still_img0006
    play_item still_img0024
    play_item still_img0031
    play_item jup1
    play_item ngc2
    play_item ori3
    play_item car
```

```
sel_list EnglishLessonSelList play_item shenandoah
    repeat 2 then wait 3 then playWarDrums
    selection 1 ExpertMenuStill  10, 108, 180, 120
    selection 2 nasaFirst5       10, 123, 180, 138
    selection 3 nasaFirst30      10, 141, 180, 153
    selection 4 nasaComplete     10, 156, 180, 171
    selection 5 justCDDA         10, 174, 180, 186

sel_list MultDefaultSel play_item mult_def
    default multi_default 50, 50, 100, 100
    next ExpertMenuStill   1, 1, 1, 1
    previous ExpertMenuStill 1, 1, 1, 1
    return ExpertMenuStill 1, 1, 1, 1
    selection 1 Md_one    0, 0, 0, 1
    selection 2 Md_two    0, 0, 0, 1
    selection 3 Md_three  0, 0, 0, 1
```

Directory definition

A CD-i directory definition (*directory_defintion*) 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 . (period) characters. File names must contain at least one alphanumeric. Their length can not exceed 28 characters, and file names must be unique within their directory.

VCD directory structure definitions

A VCD/Karaoke directory definition must contain the `CDI` and `MPEGAV` directories. The `CDI` directory contains the VCD application and all the application-related files. The `MPEGAV` directory contains all the audio-video sequence files in the order they appear on the disc. Segment files are not specified in the directory definition.

Note: `vcdmaster` automatically creates a `VCD` and a `SEGMENTS` directory specification. `vcdmaster` returns an error if you explicitly create a `VCD` or `SEGMENTS` directory.

The MPEG files specified in the `MPEGAV` directory must use the following name convention: `MUSICXX.DAT` or `AVSEQXX.DAT` where `XX` is 01..98, sequentially. The `MUSICXX.DAT` and `AVSEQXX.DAT` naming conventions cannot be mixed.

For example, the following specifications show legal and illegal naming conventions:

```
"MPEGAV"                                ! illegal specification
    {
      "MUSIC01.DAT;1" from music01
      "AVSEQ02.DAT;1" from music02      ! mixed names
      "MUSIC04.DAT;1" from music03      ! nonsequential names
      "Music03.DAT;1" from music04      ! non-CD-ROMXA name
    }
"MPEGAV"                                ! legal specification
    {
      "MUSIC01.DAT;1" from music01
      "MUSIC02.DAT;1" from music02
      "MUSIC03.DAT;1" from music03
      "MUSIC04.DAT;1" from music04
    }
```

If you use scanlists with your MPEG files, the scanlist files generated by `vcdmaster` must be specified in the directory structure definition and must use the exact same filename as the corresponding specification in the `MPEGAV` directory. The only difference between the two file specifications is the identifier used in the scanlist specification.

Note: The Philips VCD application requires the scanlist file specifications to be placed in the `CDI` directory.

For example, the following directory specification segment shows the VCD application files, three scanlist specifications and their corresponding MPEG file specifications:

```
"CDI" {                                     ! scanlist files
      "CDI_VCD.APP;1"  protection 0x555     from Appl
      "CDI_ALL.RTF;1"  from All
      "CDI_BUM.DAT;1"  from Bumper
      "CDI_FONT.FNT;1" from Font
      "MUSIC01.DAT;1"  from scan01
```

```
      “MUSIC02.DAT;1”  from scan02
      “MUSIC03.DAT;1”  from scan03
    }
“MPEGAV”{                                   ! mpeg files
      “MUSIC01.DAT;1”  from music01
      “MUSIC02.DAT;1”  from music02
      “MUSIC03.DAT;1”  from music03
    }
```

In all other respects, the VCD naming convention must follow the CD-ROM XA filename requirements that follow.

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

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

where

file_name is a quoted file name string.

attribute is one of the following:

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

where

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
is a quoted 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
    }
}
```

The following example specifies a VCD directory definition:

```
{
    “COPYRIGH.E;1” protection 0x111 from Copyright
    “ABSTRACT.E;1” protection 0x111 from Abstract
    “BIBLIOGR.E;1” protection 0x111 from Biblio
    “CDI”
    {
        “CDI_VCD.APP;1” protection 0x555 from Appl
        “CDI_ALL.RTF;1”  from All
        “CDI_BUM.DAT;1”  from Bumper
        “CDI_FONT.FNT;1”  from Font
        “MUSIC01;1” from scan01
        “MUSIC02;1” from scan02
        “MUSIC03;1” from scan03
    }
    “MPEGAV”
    {
        “MUSIC01;1” from music01
        “MUSIC02;1” from music02
        “MUSIC03;1” from music03
    }
}
```

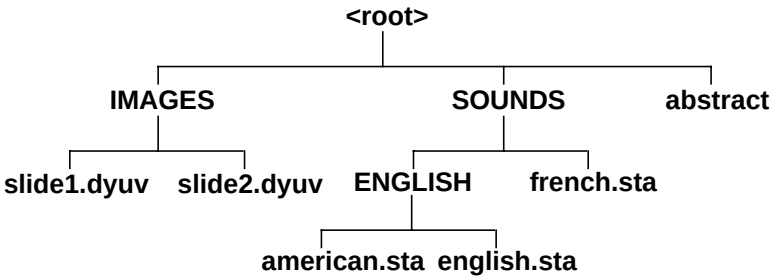
File placement in the disc image

Files are placed in their directory on the disc image 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` runtime option is specified, however, subdirectories and their contents are generated as they are encountered.

This section explains how master actually generates the directory structure in the disc image from the directory structure script definition.

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`.