MacBinary

Description

MacBinary is a standard format for binary transfer of arbitrary Macintosh documents via a telecommunication link. It is intended for use both between Macintoshes and for use in uploading arbitrary Macintosh documents to remote systems (where it is presumed that they will be stored as an exact image of the data transmitted). It does this by combing both the resource and data forks (as well as the "Finder Info") of a standard Macintosh file into a single data fork only file that can be stored on non-Macintosh machines.

The original version of MacBinary was first proposed on March 13, 1985 by Dennis Brothers to other developers on MAUG (Micro-networked Apple User's Group) on Compuserve. Over the course of about 2 months the working group met and a few minor revisions what we now refer to as MacBinary I was born. The full specification as originally written by Dennis is available here.

In 1987, a group of software developers met on the Compuserve, Delphi and Bix networks (the three top online services of the time) and made a series of additions to the original MacBinary specification leading to MacBinary II. MacBinary II addressed changes in the MacOS including the (then) new HFS volume format and provided for future extensions via a "secondary header" (a feature which has never been put into use in any non-experimental form). You can find the MacBinary II specification, as originally written by that working group, can be found here.

MacBinary II had served the Macintosh community well for almost 10 years when it was discovered that it was time for an update. MacOS8 introduced some features to the MacOS that are incompatible, so I recruited a group of the key Internet software developers update the MacBinary standard. We produced the new MacBinary III (available here) which adds supports for all MacOS & HFS features.

Sample Code

MacBinary III - a Pascal application written originally by Peter Lewis with MacBinary III support by Leonard Rosenthol

DropMacBinary III - a C application written by Chris Evans

Applications

MacBinary III - by Peter Lewis (MacBinary III support by Leonard Rosenthol)

DropMacBinary III - by Chris Evans

Data File Format

A Mascot data file is a plain text (ASCII) file.

For a Peptide Mass Fingerprint, the file should contain a list of peptide mass values, one per line. Mascot will treat the first numeric value on each line as a mass value, and ignore everything else.

The peak list export formats of a wide range of instrument data systems are directly compatible with these requirements. In addition, Mascot will automatically recognise the PE Biosystems .PKM format and the Bruker Analysis Data Report from AutoXecute.

For an MS/MS Ions Search, the data file must contain one or more sets of MS/MS data. In the Mascot generic file format, each MS/MS dataset is a list of pairs of mass and intensity values. In addition, certain proprietary formats are supported for MS/MS datasets:

- Finnigan (.ASC)
- Micromass (.PKL)
- Sequest (.DTA)
- PerSeptive (.PKS)
- Sciex API III

A data file may include embedded search parameters. Most embedded parameters can only appear once, at the head of the data file. In a Mascot generic format file, a few parameters can appear within an MS/MS dataset (PEPMASS, TITLE, CHARGE, TOL, TOLU, SEQ, COMP).

If there is a conflict between the values of the embedded parameters and values entered into search form fields, the embedded parameters always take precedence.

The following paragraphs illustrate the data file format by means of examples. The rules which Mascot follows when parsing a data file provide an alternative, more rigorous description of what is and is not acceptable.

Mascot Generic Format

The Mascot generic format for a data file submitted to Mascot is (square brackets indicate optional elements, they should not be included in an actual data file):

```
[Embedded Parameter(s)]
 Query 1
[Query 2]
 .
 .
 .
[Query N]
```

Blank lines can be used to improve readability, and comment lines which begin with one of the symbols #;!/ can be included, but only outside of MS/MS datasets.

Peptide Mass Fingerprint

In the case of a Peptide Mass Fingerprint, each query is just a single peptide mass value. For example:

```
764.2
1231.0
```

```
1284
1944.8
2020.2
2100.35
```

If your MS data system outputs peak lists which include additional values, such as peak intensity, these will be ignored. Only the first value on each line is used for a Peptide Mass Fingerprint.

There are two ways to change default search parameters. One way is using the search form fields. The other is to place embedded parameters at the beginning of the data file. For example:

```
COM=Digest #A6345
CLE=Lys-C
CHARGE=1+
PFA=1
764.2
1231.0
1284
1944.8
2020.2
2100.35
```

The embedded parameters (COM, CLE, CHARGE, PFA) over-ride the entries in the corresponding form fields, if any. All of the other search parameters default to the search form settings.

A peptide mass fingerprint data file can only contain peptide mass fingerprint queries. Sequence queries or MS/MS datasets are not permitted.

MS/MS Ions Search

For an MS/MS Ions Search, each query is an MS/MS dataset which represents a complete MS/MS spectrum, and is delimited by a pair of tags: BEGIN IONS and END IONS.

As in the case of a peptide mass fingerprint, the search form defaults can be over-ridden by including embedded parameters at the beginning of the data file. Within each MS/MS dataset, the mass of the precursor peptide *must* be specified using the PEPMASS parameter. Additional parameters within each dataset are optional, and can be used to specify:

- TITLE for spectrum identification
- CHARGE state of the precursor peptide
- TOL peptide tolerance
- TOLU peptide tolerance units
- SEQ a sequence qualifier (multiple SEQ qualifier are allowed)
- COMP a composition qualifier (only one COMP qualifier is allowed)

Parameters within an MS/MS dataset only apply locally, to that dataset. In the case of the CHARGE parameter, this means that you can have a global CHARGE setting, either from the search form or from a parameter at the head of the data file, as well as a local setting in one or more of the MS/MS datasets. This can be useful if the mass spectrometer data system cannot always determine precursor charge state correctly. For example, the global setting could be 2+ and 3+. When an unambiguous charge state can be determined, the correct charge is written to the local CHARGE parameter.

Parameters within an MS/MS dataset must always be at the beginning, immediately following the BEGIN IONS tag. They cannot appear within or following the fragment ion list. For example:

```
COM=10 pmol digest of Sample X15
ITOL=1
ITOLU=Da
MODS=Carbamidomethyl (C)
IT_MODS=Oxidation (M)
MASS=Monoisotopic
USERNAME=Lou Scene
USEREMAIL=leu@altered-state.edu
CHARGE=2+ and 3+
```

```
BEGIN IONS
TITLE=Peak 1
PEPMASS=983.6
846.60 73
846.80 44
847.60 67
.
.
.
1640.10 291
1640.60 54
1895.50 49
END IONS

BEGIN IONS
TITLE=Peak 2
PEPMASS=1084.9
345.10 237
370.20 128
460.20 108
.
.
.
1673.30 1007
1674.00 974
1675.30 79
END IONS

BEGIN IONS
TITLE=Peak 3
PEPMASS=1244.7
.
.
.
```

Most MS data systems have some form of ASCII peak list function which will generate a file requiring only minor modification with a text editor to conform to this format.

Fragment ion intensity information is very important. Mascot will iteratively select sub-sets of the most intense peaks, looking for the group which most clearly discriminates the score of the top matched protein.

N.B. There is an upper limit of 10,000 peaks per individual MS/MS spectrum. If you see an error message reporting that this limit has been exceeded, it almost certainly means that your data are profile data, and not peak lists. Mascot has a peptide mass limit of 16 kDa or 255 residues. This makes it *very* unlikely that a single MS/MS spectrum could ever contain more than 1000 true peaks.

It is possible for an MS/MS ions search data file in the Mascot generic format to include sequence queries and peptide mass fingerprint queries. This is not allowed if the file contains proprietary format MS/MS data, and neither is mixing proprietary formats.

Here is a rather baroque example:

```
# following lines define parameters.
# NB no spaces allowed on either side of the = symbol
COM=My favourite protein has been eaten by an enzyme
CLE=Trypsin
CHARGE=2+
# following line is a peptide mass fingerprint query.
# NB can be no other values or text on the line
1024.6
# following line is a sequence query, which must
# conform precisely to sequence query syntax rules
2321 seq(n-ACTL) comp(2[C])
# so is this
1896 ions(345.6:24.7,347.8:45.4, ... ,1024.7:18.7)
# An MS/MS ions query is delimited by the tags
# BEGIN IONS and END IONS. Space(s)
# are used to separate mass and intensity values
BEGIN IONS
TITLE=The first peptide - dodgy peak detection, so extra wide tolerance
PEPMASS=896.05
CHARGE=3+
```

```
TOL=3
TOLU=Da
SEQ=n-AC[DHK]
COMP=2[H]0[M]3[DE]*[K]
240.1 3
242.1 12
245.2 32
.
.
.
1623.7 55
1624.7 23
END IONS
```

Embedded Search Parameters

Search parameters can be embedded into the data file or the query window using the following tags. In the absence of an embedded parameter, the default value is the setting of the corresponding search form field at the point of submission:

Name	Description	Choices/Range	Notes
ACCESSION	List of accession strings for error tolerant search		
CHARGE	Peptide charge	Mr	
		1+	MH+ in PMF
		2+	Not PMF
		2+ and 3+	Not PMF
		3+	Not PMF
		etc., up to 8+	Not PMF
CLE	Enzyme	Trypsin	Default
		etc., as defined	
COM	Search title		
DB	Database	MSDB	
		etc., as defined	
ERRORTOLERANT	Error tolerant	0 (false)	Default
		1 (true)	
FORMAT	MS/MS dataset	Mascot generic	Default
		Sequest (.DTA)	
		Finnigan (.ASC)	
		Micromass (.PKL)	
		PerSeptive (.PKS)	
		Sciex API III	
ICAT	ICAT	Off	Default
		On	Not PMF
INSTRUMENT	MS/MS ion series	Default	Default
		ESI-QUAD-TOF	
		etc., as defined	
ITOL	Fragment ion tol.	Unit dependent	
ITOLU	Units for ITOL	Da	
		mmu	

MASS	Mono. or average	Monoisotopic	
		Average	
MODS	Fixed Mods	As specified in mod_file, or choice shown on search form	
IT_MODS	Variable Mods		
OVERVIEW	Report overview	Off	default
		On	
PEPMASS	Peptide mass	>100	
PFA	Partials	integer, 0 to 9	default 1
PRECURSOR	Precursor m/z	>100	
REPORT	Maximum hits	AUTO or integer	
REPTYPE	Type of report	Protein	All searches
		Concise	PMF only
		Peptide	MIS only
SEARCH	Type of search	PMF	
		MIS	
SEG	Protein mass	Empty or >0	
TAXONOMY	Taxonomy	choice shown on search form	
TITLE	Query identifier		
TOL	Peptide mass tol.	Unit dependent	
TOLU	Units for TOL	%	
		ppm	
		mmu	
		Da	
USER00 to USER12		Uncommitted variables, for use in custom reports	
USEREMAIL	User email		
USERNAME	User name		

Most of these parameters can appear once only, at the head of a data file. The exceptions, which in Mascot generic format (only) can be defined with local scope in an MS/MS dataset are:

- PEPMASS: The mass or observed m/z for the precursor peptide, (mandatory).
- CHARGE: The charge state for the precursor peptide, (optional).
- TITLE: An identifying title for an individual query, (optional).
- TOL: Precursor peptide tolerance
- TOLU: Precursor peptide tolerance units
- SEQ: Sequence qualifier (multiple SEQ qualifier are allowed)
- COMP: Composition qualifier (only one COMP qualifier is allowed)

N.B. TITLE identifies the individual dataset, COM identifies the entire search.

The FORMAT parameter is used to identify proprietary MS/MS dataset formats. It can appear once only, at the start of the file. If there is no FORMAT parameter, the default is Mascot generic format.

Proprietary MS/MS Peak List Formats

Finnigan (ASC) Files

Files in this format are created by the LIST command on the ICIS data system. The header block for each MS/MS dataset begins with a "LIST:" field. The text in this field is used by Mascot to identify the query, equivalent to an embedded TITLE parameter.

The ASC file header does not specify a charge state for the precursor peptide. This can be specified (globally) on the search form, or by an embedded CHARGE parameter at the head of the data file.

The precursor peptide m/z value is parsed from the "Mode:" field. Mascot uses the prevailing CHARGE value to calculate M_r from the observed m/z.

A blank line to delimit MS/MS datasets is optional.

Example of Finnigan ASC format:

```
LIST: dp210198b                      21-Jan-98 DERIVED SPECTRUM    #9
Samp: Spot 6483 from Gel 29A44                 Start : 18:37:54   100
Mode: ESI +DAU 808.3 @ 25eV UP LR
Oper: Administrator                            Inlet :
Base: 798.9             Inten : 25525          Masses: 225 > 2000
Norm: 798.9             RIC   : 181489         #peaks: 586
Peak: 1000.00 mmu
Data: +/1>99
                           0
  No.        Mass   Intensity     %RA    %RIC  Flags
    1       229.3           8    0.03    0.00   #
    2       230.3           9    0.04    0.00   #
    3       259.9           8    0.03    0.00   #
{{{{{{{{{{{{{{{{{{
{ data edited out {
{{{{{{{{{{{{{{{{{{
  583      1831.0           5    0.02    0.00   #
  584      1878.3           5    0.02    0.00   #
  585      1881.8           8    0.03    0.00   #

LIST: dp210198a                      21-Jan-98 DERIVED SPECTRUM    #9
Samp: Spot 6483 from Gel 29A44                 Start : 18:27:30    95
Mode: ESI +DAU 973.9 @ 25eV AVER UP LR
Oper: Administrator                            Inlet :
Base: 974.5             Inten : 191564         Masses: 270 > 1800
Norm: 974.5             RIC   : 341387         #peaks: 593
Peak: 1000.00 mmu
Data: +/1>95
                           0
  No.        Mass   Intensity     %RA    %RIC  Flags
    1       297.9          10    0.01    0.00   #
    2       326.7           8    0.00    0.00   #
    3       345.1         237    0.12    0.07   #
{{{{{{{{
{ Etc. {
{{{{{{{{
```

Sequest (DTA) Files

Sequest users can create these files from Finnigan LCQ data using the LCQ_DTA utility.

The DTA format is very simple. The first line contains the singly protonated peptide mass (MH^+) and the peptide charge state as a pair of space separated values. Subsequent lines contain space separated pairs of fragment ion m/z and intensity values.

N.B. In a DTA file, the precursor peptide mass is an MH^+ value independent of the charge state. In Mascot generic format, the precursor peptide mass is an observed m/z value, from which M_r or MH_n^{n+} is calculated using the prevailing charge state. For example, in Mascot:

PEPMASS=1000
CHARGE=2+

... means that the relative molecular mass M_r is 1998. This is equivalent to a DTA file which starts:

1999 2

The DTA format uses the file name to identify the dataset. An example of a file name would be "Myoglobin_digest.0012.0015.3.dta". This corresponds to scans 12 to 15 of an LC-MS run, averaged together, and a peptide charge state of 3^+.

While it is perfectly possible to submit a native DTA file to Mascot, each file contains only a single MS/MS data set. If you have a series of related datasets, such as from an LC-MS experiment, it is much better to concatenate the DTA files into a single data file so that the queries can be scored and reported collectively.

Remember to include at least one blank line between each MS/MS dataset. A delimiter between datasets is essential because the DTA format is relatively unstructured. Without a delimiter, the first line of a new dataset (peptide mass, charge) might be just another line from the previous dataset (fragment ion mass, intensity).

Utilities to concatenate DTA files automatically can be downloaded from the Xcalibur help page.

Micromass (PKL) Files

QTof users can export peak list data in either DTA or PKL format using the Micromass ProteinLynx package.

The PKL format is similar to the DTA file format, but supports multiple MS/MS datasets in a single file. The first line of a PKL dataset contains the observed m/z, intensity, and charge state of the precursor peptide as a triplet of space separated values. Subsequent lines contain space separated pairs of fragment ion m/z and intensity values.

Multiple MS/MS datasets are delimited by at least one blank line.

PerSeptive (.PKS)

PSD peak lists exported from Grams as .PKS files contain data from a single PSD spectrum. Since the .PKS format does not include details of the precursor peptide m/z, this information must be entered manually into the PRECURSOR and CHARGE form fields. This limitation also means that multiple spectra cannot be merged into a single data file.

Example of the .PKS file format:

```
"Peak Table"
OP=0
Center X   Peak Y   Left X   Right X   Time X  Mass Difference  Name
STD.Misc   Height   Left Y   Right Y   %Height,Width,%Area,%Quan,H/A
818.39992      4265.0000 818.39992 818.39992 81554.550 0          818.3999
C  0.?     0         4265.0000 4265.0000
820.42154      3765.0000 820.42154 820.42154 81616.547 0          820.4215
C  0.?     0         3765.0000 3765.0000
842.38252      2571.0000 842.10681 842.62999 82290.021 0          842.3825
C  0.?     0         1800.0000 1800.0000
{{{{{{{{
{ Etc. {
{{{{{{{{
```

Sciex API III

Peak lists exported from PE Sciex API III contain data from a single MS/MS spectrum. Since the file format does not include details of the precursor peptide m/z, this information must be entered manually into the PRECURSOR and CHARGE form fields. This limitation also means that multiple spectra cannot be merged into a single data file.

Example of PE Sciex peak list format:

```
287.50 650 287.5
301.00 1150 301.0
305.00 1150 305.0
315.00 6550 315.0
321.00 16,000 321.0
333.00 3050 333.0
333.50 1800 333.5
370.00 1550 370.0
{{{{{{{{
{ Etc. {
{{{{{{{{
```

The Rules

1. Filename extensions are not significant.
2. Parameters at the head of the data file apply to the entire search and over-ride the default settings provided by the search form fields.
3. Parameter labels are not case sensitive. Case is preserved for parameter values which are free text strings. There must be no spaces either side of the = symbol
4. Most embedded parameters can only appear at the head of the file, prior to any query data. The exceptions are PEPMASS and TITLE, which can only be used between the BEGIN IONS and END IONS tags in a Mascot generic format MS/MS data file, and CHARGE, TOL, TOLU which can appear in either place. SEQ and COMP can appear between the BEGIN IONS and END IONS tags or as qualifiers to a mass value using the Sequence Query syntax.
5. Parameters between BEGIN IONS and END IONS tag pairs only apply to the local MS/MS dataset. If present, they must appear immediately following the BEGIN IONS tag.
6. Outside of an MS/MS dataset, blank lines or lines which start with the symbols # ; ! / are ignored. This means that blank lines can be used to improve readability and comments can be included by starting the line with one of these symbols.
7. A SEARCH type must be defined, (peptide mass fingerprint or MS/MS ions search). The default is determined by the search form used to upload the file. Like any other parameter, this can be over-ridden by including a SEARCH parameter in the file header.
8. A peptide mass fingerprint (PMF) search can only contain PMF queries. This allows for a relaxed syntax in which any line starting with a number is assumed to be a query. This first number is parsed as a peptide mass and the rest of the line is ignored.
9. In the absence of a FORMAT parameter, the default is Mascot generic.
10. Mascot generic format permits an MS/MS ions search file to include peptide mass fingerprint queries and sequence queries.
11. MS/MS ions searches can contain MS/MS datasets in proprietary formats only if this is declared with a FORMAT parameter. Mixing proprietary formats, or including non-MS/MS dataset queries in a proprietary format file, is not allowed.
12. Within an MS/MS dataset, intensity values must be supplied. Fragment ion masses must be positive, non-zero values. Intensities must be positive values. Peptide m/z values must be equivalent to 100 < = Mr <= 16000.

New WAVE Types

The necessary type, structure and constant defintions are in *mmreg.h*.

All newly defined WAVE types must contain both a fact chunk and an extended wave format description within the 'fmt' chunk. RIFF WAVE files of type WAVE_FORMAT_PCM need not have the extra chunk nor the extended wave format description.

Fact Chunk

This chunk stores file dependent information about the contents of the WAVE file. It currently specifies the length of the file in samples.

WAVEFORMATEX

The extended wave format structure is used to defined all non-PCM format wave data, and is described as follows in the include file *mmreg.h*:

```
/* general extended waveform format structure */

/* Use this for all NON PCM formats */

/* (information common to all formats) */

typedef struct waveformat_extended_tag {

WORD wFormatTag; /* format type */

WORD nChannels; /* number of channels (i.e. mono, stereo...) */

DWORD nSamplesPerSec; /* sample rate */

DWORD nAvgBytesPerSec; /* for buffer estimation */

WORD nBlockAlign; /* block size of data */

WORD wBitsPerSample; /* Number of bits per sample of mono data */

WORD cbSize; /* The count in bytes of the extra size */} WAVEFORMATEX;
```

wFormatTag	Defines the type of WAVE file.
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 11025, 22050, or 44100. Other sample rates are allowed, but not encouraged. This rate is also used by the sample size entry in the fact chunk to determine the length in time of the data.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	The block alignment (in bytes) of the data in <**data-ck**>. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample per channel data. Each channel is assumed to have the same sample resolution. If this field is not needed, then it should be set to zero.img
cbSize	The size in bytes of the extra information in the WAVE format header not including the size of the WAVEFORMATEX structure.. As an example, in the IMA ADPCM format cbSize is calculated as sizeof(IMAADPCMWAVEFORMAT) - sizeof(WAVEFORMATEX) which yeilds two.

Defined wFormatTags

Expr1	WAVE form Registration No - Hex	Expr2
#define WAVE_FORMAT_G723_ADPCM	0x0014	/* Antex Electronics Corporation */
#define WAVE_FORMAT_ANTEX_ADPCME	0x0033	/* Antex Electronics Corporation */
#define WAVE_FORMAT_G721_ADPCM	0x0040	/* Antex Electronics Corporation */
#define WAVE_FORMAT_APTX	0x0025	/* Audio Processing Technology */
#define WAVE_FORMAT_AUDIOFILE_AF36	0x0024	/* Audiofile, Inc. */
#define WAVE_FORMAT_AUDIOFILE_AF10	0x0026	/* Audiofile, Inc. */
#define WAVE_FORMAT_CONTROL_RES_VQLPC	0x0034	/* Control Resources Limited */
#define WAVE_FORMAT_CONTROL_RES_CR10	0x0037	/* Control Resources Limited */
#define WAVE_FORMAT_CREATIVE_ADPCM	0x0200	/* Creative Labs, Inc */
#define WAVE_FORMAT_DOLBY_AC2	0x0030	/* Dolby Laboratories */
#define WAVE_FORMAT_DSPGROUP_TRUESPEECH	0x0022	/* DSP Group, Inc */
#define WAVE_FORMAT_DIGISTD	0x0015	/* DSP Solutions, Inc. */
#define WAVE_FORMAT_DIGIFIX	0x0016	/* DSP Solutions, Inc. */
#define WAVE_FORMAT_DIGIREAL	0x0035	/* DSP Solutions, Inc. */
#define WAVE_FORMAT_DIGIADPCM	0x0036	/* DSP Solutions, Inc. */
#define WAVE_FORMAT_ECHOSC1	0x0023	/* Echo Speech Corporation */
#define WAVE_FORMAT_FM_TOWNS_SND	0x0300	/* Fujitsu Corp. */
#define WAVE_FORMAT_IBM_CVSD	0x0005	/* IBM Corporation */
#define WAVE_FORMAT_OLIGSM	0x1000	/* Ing C. Olivetti & C., S.p.A. */
#define WAVE_FORMAT_OLIADPCM	0x1001	/* Ing C. Olivetti & C., S.p.A. */
#define WAVE_FORMAT_OLICELP	0x1002	/* Ing C. Olivetti & C., S.p.A. */
#define WAVE_FORMAT_OLISBC	0x1003	/* Ing C. Olivetti & C., S.p.A. */
#define WAVE_FORMAT_OLIOPR	0x1004	/* Ing C. Olivetti & C., S.p.A. */
#define WAVE_FORMAT_IMA_ADPCM	(WAVE_FORM_DVI_ADPCM)	/* Intel Corporation */
#define WAVE_FORMAT_DVI_ADPCM	0x0011	/* Intel Corporation */
#define WAVE_FORMAT_UNKNOWN	0x0000	/* Microsoft Corporation */
#define WAVE_FORMAT_PCM	0x0001	/* Microsoft Corporation */
#define WAVE_FORMAT_ADPCM	0x0002	/* Microsoft Corporation */
#define WAVE_FORMAT_ALAW	0x0006	/* Microsoft Corporation */
#define WAVE_FORMAT_MULAW	0x0007	/* Microsoft Corporation */
#define WAVE_FORMAT_GSM610	0x0031	/* Microsoft Corporation */
#define WAVE_FORMAT_MPEG	0x0050	/* Microsoft Corporation */
#define WAVE_FORMAT_NMS_VBXADPCM	0x0038	/* Natural MicroSystems */
#define WAVE_FORMAT_OKI_ADPCM	0x0010	/* OKI */
#define WAVE_FORMAT_SIERRA_ADPCM	0x0013	/* Sierra Semiconductor Corp */
#define WAVE_FORMAT_SONARC	0x0021	/* Speech Compression */
#define WAVE_FORMAT_MEDIASPACE_ADPCM	0x0012	/* Videologic */
#define WAVE_FORMAT_YAMAHA_ADPCM	0x0020	/* Yamaha Corporation of America */

Unknown Wave Type

Added: 05/01/92
Author: Microsoft

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

Changed as of September 5, 1993: This wave format will not be defined. For development purposes, DO NOT USE 0x0000. Instead, USE 0xffff until an ID has been obtained.

 # define WAVE_FORMAT_UNKNOWN (0x0000)

wFormatTag	This must be set to WAVE_FORMAT_UNKNOWN.
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

Microsoft ADPCM

Added 05/01/92
Author: Microsoft

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_ADPCM (0x0002)

```
typedef struct adpcmcoef_tag {

int iCoef1;

int iCoef2;

} ADPCMCOEFSET;


typedef struct adpcmwaveformat_tag {

WAVEFORMATEX wfxx;

WORD wSamplesPerBlock;

WORD wNumCoef;

ADPCMCOEFSET aCoeff[wNumCoef];

} ADPCMWAVEFORMAT;
```

---

wFormatTag	This must be set to WAVE_FORMAT_ADPCM.		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.		
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 11025, 22050, or 44100. Other sample rates are allowed, but not encouraged.		
nAvgBytesPerSec	Average data rate. ((nSamplesperSec / nSamplesPerBlock) * nBlockAlign). Playback software can estimate the buffer size using the value.		
nBlockAlign	The block alignment (in bytes) of the data in .		
	nSamplesPerSec x Channels	nBlockAlign	
	8k	256	
	11k	256	
	22k	512	
	44k	1024	
	Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.		
wBitsPerSample	This is the number of bits per sample of ADPCM. Currently only 4 bits per sample is defined. Other values are reserved.		
cbSize	The size in bytes of the extended information after the WAVEFORMATEX structure. For the standard WAVE_FORMAT_ADPCM using the standard seven coefficient pairs, this is 32. If extra coefficients are added, then this value will increase.		
nSamplesPerBlock	Count of number of samples per block. (((nBlockAlign - (7 * nChannels)) * 8) / (wBitsPerSample * nChannels)) + 2.		
nNumCoef	Count of the number of coefficient sets defined in aCoef.		
aCoeff	These are the coefficients used by the wave to play. They may be interpreted as fixed point 8.8 signed values. Currently there are 7 preset coefficient sets. They must appear in the following order.		
	Coef Set	Coef1	Coef2
	0	256	0
	1	512	-256
	2	0	0
	3	192	64
	4	240	0
	5	460	-208
	6	392	-232
	Note that if even only 1 coefficient set was used to encode the file then all coefficient sets are still included. More coefficients may be added by the encoding software, but the first 7 must always be the same.		

Note: 8.8 signed values can be divided by 256 to obtain the integer portion of the value.

Block

The block has three parts, the header, data, and padding. The three together are <**nBlockAlign**> bytes.

typedef struct adpcmblockheader_tag {

BYTE bPredictor[nChannels];

int iDelta[nChannels];

int iSamp1[nChannels];

int iSamp2[nChannels];

} ADPCMBLOCKHEADER;

Field	Description
bPredictor	Index into the aCoef array to define the predictor used to encode this block.
iDelta	Initial Delta value to use.
iSamp1	The second sample value of the block. When decoding this will be used as the previous sample to start decoding with.
iSamp2	The first sample value of the block. When decoding this will be used as the previous' previous sample to start decoding with.

Data

The data is a bit string parsed in groups of (wBitsPerSample * nChannels).

For the case of Mono Voice ADPCM (wBitsPerSample = 4, nChannels = 1) we have:

... ...

where has or < (Sample 2N + 2) (Sample 2N + 3)>

= ((4 bit error delta for sample (2 * N) + 2) << 4) | (4 bit error delta for sample (2 * N) + 3)

For the case of Stereo Voice ADPCM (wBitsPerSample = 4, nChannels = 2) we have:

... ...

where has or

< (Left Channel of Sample N + 2) (Right Channel of Sample N + 2)>

= ((4 bit error delta for left channel of sample N + 2) << 4) | (4 bit error delta for right channel of sample N + 2)

Padding

Bit Padding is used to round off the block to an exact byte length.

The size of the padding (in bits):

((nBlockAlign - (7 * nChannels)) * 8) -

(((nSamplesPerBlock - 2) * nChannels) * wBitsPerSample)

The padding does not store any data and should be made zero.

ADPCM Algorithm

Each channel of the ADPCM file can be encoded/decoded independently. However this should not destroy phase and amplitude information since each channel will track the original. Since the channels are encoded/decoded independently, this document is written as if only one channel is being decoded. Since the channels are interleaved, multiple channels may be encoded/decoded in parallel using independent local storage and temporaries.

Note that the process for encoding/decoding one block is independent from the process for the next block. Therefore the process is described for one block only, and may be repeated for other blocks. While some optimizations may relate the process for one block to another, in theory they are still independent.

Note that in the description below the number designation appended to iSamp (i.e. iSamp1 and iSamp2) refers to the placement of the sample in relation to the current one being decoded. Thus when you are decoding sample N, iSamp1 would be sample N - 1 and iSamp2 would be sample N - 2. Coef1 is the coefficient for iSamp1 and Coef2 is the coefficient for iSamp2. This numbering is identical to that used in the block and format descriptions above.

A sample application will be provided to convert a RIFF waveform file to and from ADPCM and PCM formats.

Decoding

First the predictor coefficients are determined by using the bPredictor field of block header. This value is an index into the aCoef array in the file header.

bPredictor = GETBYTE

The initial iDelta is also taken from the block header.

iDelta = GETWORD

Then the first two samples are taken from block header. (They are stored as 16 bit PCM data as iSamp1 and iSamp2. iSamp2 is the first sample of the block, iSamp1 is the second sample.)

iSamp1= GETINT

iSamp2 = GETINT

After taking this initial data from the block header, the process of decoding the rest of the block may begin. It can be done in the following manner:

While there are more samples in the block to decode:

Predict the next sample from the previous two samples.

lPredSamp = ((iSamp1 * iCoef1) + (iSamp2 *iCoef2)) / FIXED_POINT_COEF_BASE

Get the 4 bit signed error delta.

(iErrorDelta = GETNIBBLE)

Add the 'error in prediction' to the predicted next sample and prevent over/underflow errors.

(lNewSamp = lPredSample + (iDelta * iErrorDelta)

if lNewSample too large, make it the maximum allowable size.

if lNewSample too small, make it the minimum allowable size.

Output the new sample.

OUTPUT(lNewSamp)

Adjust the quantization step size used to calculate the 'error in prediction'.

iDelta = iDelta * AdaptionTable[iErrorDelta] / FIXED_POINT_ADAPTION_BASE

if iDelta too small, make it the minimum allowable size.

Update the record of previous samples.

iSamp2 = iSamp1;

iSamp1 = lNewSample.

Encoding

For each block, the encoding process can be done through the following steps. (for each channel)

Determine the predictor to use for the block.

Determine the initial iDelta for the block.

Write out the block header.

Encode and write out the data.

The predictor to use for each block can be determined in many ways.

1. A static predictor for all files.

2. The block can be encoded with each possible predictor. Then the predictor that gave the least error can be chosen. The least error can be determined from:

1. Sum of squares of differences. (from compressed/decompressed to original data)

2. The least average absolute difference.

3. The least average iDelta

3. The predictor that has the smallest initial iDelta can be chosen. (This is an approximation of method 2.3)

4. Statistics from either the previous or current block. (e.g. a linear combination of the first 5 samples of a block that corresponds to the average predicted error.)

The starting iDelta for each block can also be determined in a couple of ways.

1. One way is to always start off with the same initial iDelta.

2. Another way is to use the iDelta from the end of the previous block. (Note that for the first block an initial value must then be chosen.)

3. The initial iDelta may also be determined from the first few samples of the block. (iDelta generally fluctuates around the value that makes the absolute value of the encoded output about half maximum absolute value of the encoded output. (for 4 bit error deltas the maximum absolute value is 8. This means the initial iDelta should be set so that the first output is around 4.)

4. Finally the initial iDelta for this block may be determined from the last few samples of the last block. (Note that for the first block an initial value must then be chosen.)

Note that different choices for predictor and initial iDelta will result in different audio quality.

Once the predictor and starting quantization values are chosen, the block header may be written out.

First the choice of predictor is written out. (For each channel.)

Then the initial iDelta (quantization scale) is written out. (For each channel.)

Then the 16 bit PCM value of the second sample is written out. (iSamp1) (For each channel.)

Finally the 16 bit PCM value of the first sample is written out. (iSamp2) (For each channel.)

Then the rest of the block may be encoded. (Note that the first encoded value will be for the 3rd sample in the block since the first two are contained in the header.)

While there are more samples in the block to decode:

Predict the next sample from the previous two samples.

lPredSamp = ((iSamp1 * iCoef1) + (iSamp2 *iCoef2))

/ FIXED_POINT_COEF_BASE

The 4 bit signed error delta is produced and overflow/underflow is prevented..

iErrorDelta = (Sample(n) - lPredSamp) / iDelta

if iErrorDelta is too large, make it the maximum allowable size.

if iErrorDelta is too small, make it the minimum allowable size.

Then the nibble iErrorDelta is written out.

PutNIBBLE(iErrorDelta)

Add the 'error in prediction' to the predicted next sample and prevent over/underflow errors.

(lNewSamp = lPredSample + (iDelta * iErrorDelta)

if lNewSample too large, make it the maximum allowable size.

if lNewSample too small, make it the minimum allowable size.

Adjust the quantization step size used to calculate the 'error in prediction'.

iDelta = iDelta * AdaptionTable[iErrorDelta] / FIXED_POINT_ADAPTION_BASE

if iDelta too small, make it the minimum allowable size.

Update the record of previous samples.

iSamp2 = iSamp1;

iSamp1 = lNewSample.

Sample C Code

Sample C Code is contained in the file msadpcm.c, which is available with this document in electronic form and separately. See the Overview section for how to obtain this sample code.

CVSD Wave Type

Added 07/21/92
Author: DSP Solutions, formerly Digispeech

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_IBM_CVSD (0x0005)

wFormatTag	This must be set to WAVE_FORMAT_IBM_CVSD
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo...
nSamplesPerSec	Frequency the source was sampled at. See chart below.
nAvgBytesPerSec	Average data rate. See chart below. (One of 1800, 2400, 3000, 3600, 4200, or 4800) Playback software can estimate the buffer size using the value.
nBlockAlign	Set to 2048 to provide efficient caching of file from CD-ROM. Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. This is always 1 for CVSD.
cbSize	The size in bytes of the rest of the wave format header. This is zero for CVSD.

The Digispeech CVSD compression format is compatible with the IBM PS/2 Speech Adapter, which uses a Motorola MC3418 for CVSD modulation. The Motorola chip uses only one algorithm which can work at variable sampling clock rates. The CVSD algorithm compresses each input audio sample to 1 bit. An acceptable quality of sound is achieved using high sampling rates. The Digispeech DS201 adapter supports six CVSD sampling frequencies, which are being used by most software using the IBM PS/2 Speech Adapter:

Sample Rate	**Bytes/Second**
14,400Hz	1800 Bytes
19,200Hz	2400 Bytes
24,000Hz	3000 Bytes
28,800Hz	3600 Bytes
33,600Hz	4200 Bytes
38,400Hz	4800 Bytes

The CVSD format is a compression scheme which has been used by IBM and is supported by the IBM PS/2 Speech Adapter card. Digispeech also has a card that uses this compression scheme. It is not Digispeech's policy to disclose any of these algorithms to any third party vendor.

CCITT Standard Companded Wave Types

Added: 05/22/92
Author: Microsoft, DSP Solutions formerly Digispeech, Vocaltec, Artisoft

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_ALAW (0x0006)

#define WAVE_FORMAT_MULAW (0x0007)

wFormatTag	This must be set to one of WAVE_FORMAT_ALAW, WAVE_FORMAT_MULAW
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo...
nSamplesPerSec	Frequency of the wave file. (8000, 11025, 22050, 44100).
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Size of the blocks in bytes. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. (This is 8 for all the companded formats.)
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be zero.

See the CCITT G.711 specification for details of the data format.

This is a CCITT (International Telegraph and Telephone Consultative Committee) specification. Their address is:

Palais des Nations
CH-1211 Geneva 10, Switzerland
Phone: 22 7305111

OKI ADPCM Wave Types

Added: 05/22/92
Author: DigiSpeech, Vocaltec, Wang

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_OKI_ADPCM (0x0010)

```
typedef struct oki_adpcmwaveformat_tag {
WAVEFORMATEX wfx;
WORD wPole;
} OKIADPCMWAVEFORMAT;
```

wFormatTag	This must be set to WAVE_FORMAT_OKI_ADPCM		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.		
nSamplesPerSec	Frequency the sample rate of the wave file. (8000, 11025, 22050, 44100).		
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.		
nBlockAlign	This is dependent upon the number of bits per sample.		
	wBitsPerSample	nChannels	nBlockAlign
	3	1	3
	3	2	6
	4	1	1
	4	2	1
	Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.		
wBitsPerSample	This is the number of bits per sample of data. (OKI can be 3 or 4)		
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.		
wPole	High frequency emphasis value		

This format is created and read by the OKI APDCM chip set. This chip set is used by a number of card manufacturers.

IMA ADPCM Wave Type

The IMA ADPCM and the DVI ADPCM are identical. Please see the following section on the DVI ADPCM Wave Type for a full description.

 # define WAVE_FORMAT_IMA_ADPCM (0x0011)

DVI ADPCM Wave Type

Added: 12/16/92
Author: Intel

Please note that DVI ADPCM Wave Type is Identical to IMA ADPCM Wave Type.

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_DVI_ADPCM (0x0011)

```
typedef struct dvi_adpcmwaveformat_tag {
WAVEFORMATEX wfx;

WORD wSamplesPerBlock;
} DVIADPCMWAVEFORMAT;
```

---

wFormatTag	This must be set to WAVE_FORMAT_DVI_ADPCM.	
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo...	
nSamplesPerSec	Sample rate of the WAVE file. This should be 8000, 11025, 22050 or 44100. Other sample rates are allowed.	
nAvgBytesPerSec	Total average data rate. Playback software can estimate the buffer size for a selected amount of time by using the <**nAvgBytesPerSec**> value.	
nBlockAlign	This is dependent upon the number of bits per sample.	
	wBitsPerSample	nBlockAlign
	3	((N * 3) + 1) * 4 * nChannels
	4	(N + 1) * 4 * nChannels
		Where N = 0, 1, 2, 3 . . .
	The recommended block size for coding is 256 * <**nChannels**> bytes* min(1, (/ 11 kHz)) Smaller values cause the block header to become a more significant storage overhead. But, it is up to the implementation of the coding portion of the algorithm to decide the optimal value for <**nBlockAlign**> within the given constraints (see above). The decoding portion of the algorithm must be able to handle any valid block size. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so the value of <**nBlockAlign**> can be used for allocating buffers.	
wBitsPerSample	This is the number of bits per sample of data. DVI ADPCM supports 3 or 4 bits per sample.	
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.	
wSamplesPerBlock	Count of the number of samples per channel per Block.	

Block

The block is defined to be <**nBlockAlign**> bytes in length. For DVI ADPCM this must be a multiple of 4 bytes since all information in the block is divided on 32 bit word boundaries.

The block has two parts, the header and the data. The two together are <**nBlockAlign**> bytes in length. The following diagram shows the Header and Data parts of one block.

Where:

M =

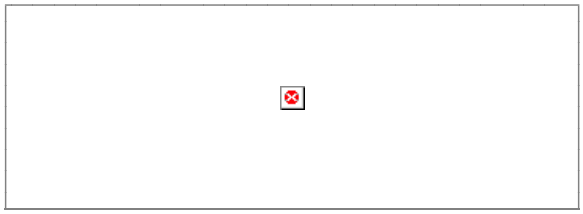
Header

This is a C structure that defines the DVI ADPCM block header.

```
typedef struct dvi_adpcmblockheader_tag {
int iSamp0;
BYTE bStepTableIndex;
BYTE bReserved;
} DVI_ADPCMBLOCKHEADER;
```

Field	Description
iSamp0	The first sample value of the block. When decoding, this will be used as the previous sample to start decoding with.
bStepTableIndex	The current index into the step table array. (0 - 88)
bReserved	This byte is reserved for future use.

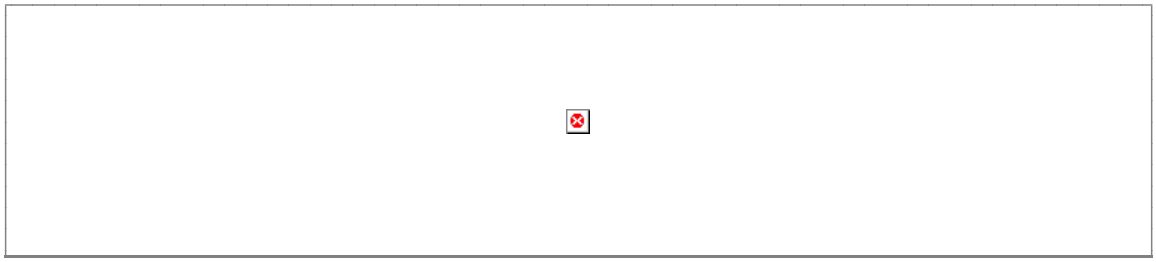
A block contains an array of <**nChannels**> header structures as defined above. This diagram gives a byte level description of the contents of each header word.



Data

The data words are interpreted differently depending on the number of bits per sample selected.

For 4 bit DVI ADPCM (where <**wBitsPerSample**> is equal to four) each data word contains eight sample codes as shown in the following diagram.



Where:

N = A data word for a given channel, in the range of 0 to

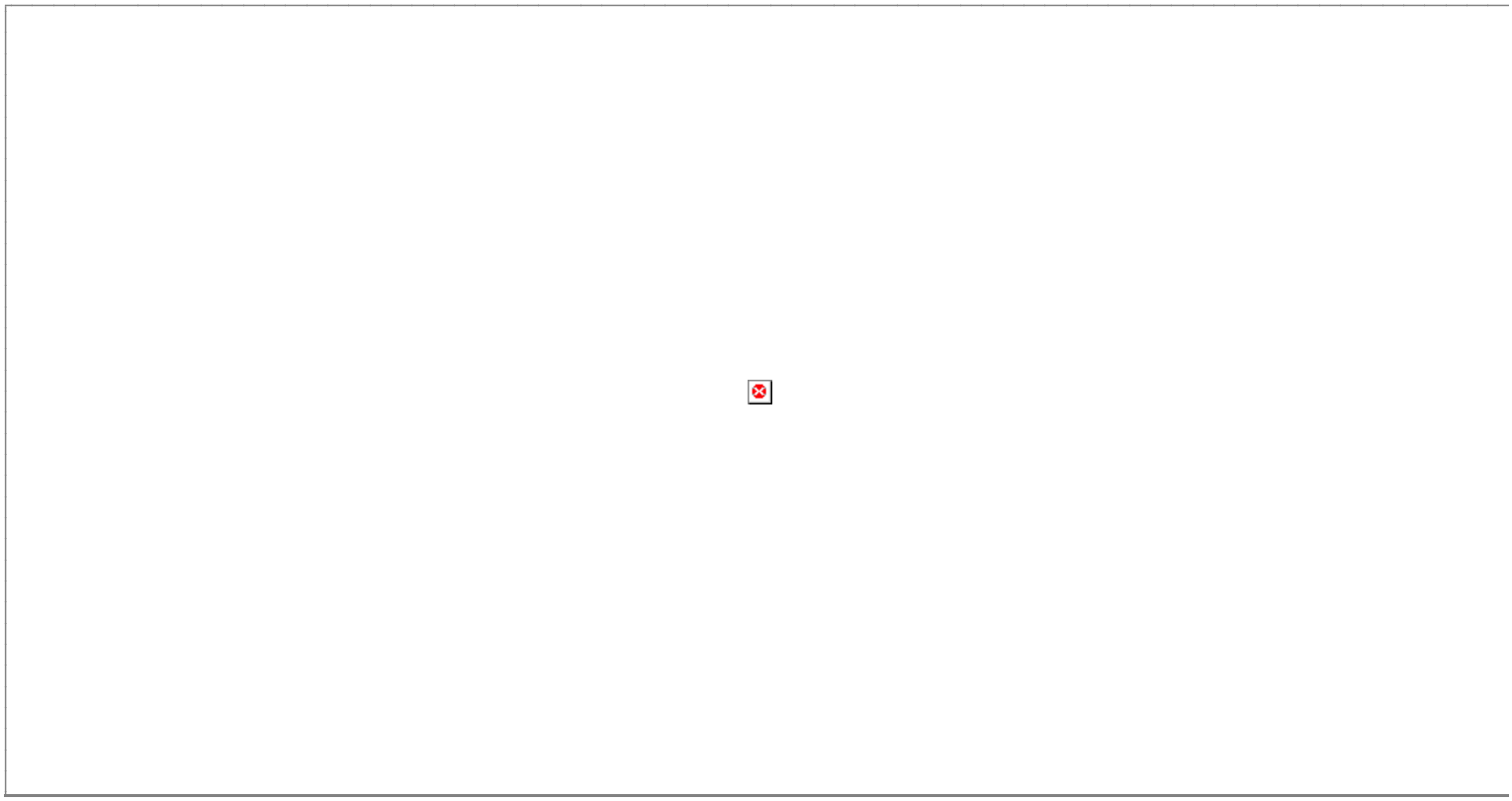
<nBlockAlign> / (4 * <nChannels>) - <nChannels> - 1

P = (N * 8) + 1

Sample 0 is always included in the block header for the channel.

Each Sample is 4 bits in length. Each block contains a total of <**wSamplesPerBlock**> samples for each channel.

For 3 bit DVI ADPCM (where <**wBitsPerSample**> is equal to three) each data word contains 10.667 sample codes. It takes three words to hold an integral number of sample codes at 3 bits per code. So for 3 bit DVI ADPCM, the number of data words is required to be a multiple of three words (12 bytes). These three words contain 32 sample codes as shown in the following diagram.



Where:

M = One of the channels, in the range of 1 to <nChannels>

N = The first data word in a group of three data words for channelM, in the

range of 0 to <nBlockAlign> / (4 * <nChannels>) - <nChannels> - 1

P = ((N / 3) * 32) + 1

Sample 0 is always included in the block header for the channel.

Each Sample is 3 bits in length. Each block contains a total of **<wSamplesPerBlock>** samples for each channel.

DVI ADPCM Algorithm

Each channel of the DVI ADPCM file can be encoded/decoded independently. Since the channels are encoded/decoded independently, this document is written as if only one channel is being decoded. Since the channels are interleaved, multiple channels may be encoded/decoded in parallel using independent local storage and temporaries.

Note that the process for encoding/decoding one block is independent from the process for the next block. Therefore the process is described for one block only, and may be repeated for other blocks.

The processes for encoding and decoding is discussed below.

Tables

The DVI ADPCM algorithm relies on two tables to encode and decode audio samples. These are the step table and the index table. The contents of these tables are fixed for this algorithm. The 3 and 4 bit versions of the DVI ADPCM algorithm use the same step table, which is:

```
const int StepTab[ 89 ] = {

7, 8, 9, 10, 11, 12, 13, 14,

16, 17, 19, 21, 23, 25, 28, 31,

34, 37, 41, 45, 50, 55, 60, 66,

73, 80, 88, 97, 107, 118, 130, 143,

157, 173, 190, 209, 230, 253, 279, 307,

337, 371, 408, 449, 494, 544, 598, 658,

724, 796, 876, 963, 1060, 1166, 1282, 1411,

1552, 1707, 1878, 2066, 2272, 2499, 2749, 3024,

3327, 3660, 4026, 4428, 4871, 5358, 5894, 6484,

7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,

15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794,

32767 }
```

But, the index table is different for the different bit rates. For the 4 bit DVI ADPCM the contents of index table is:

```
const int IndexTab[ 16 ] = { -1, -1, -1, -1, 2, 4, 6, 8,

-1, -1, -1, -1, 2, 4, 6, 8 };
```

For 3 bit DVI ADPCM the contents of the index table is:

```
const int IndexTab[ 8 ] = { -1, -1, 1, 2,
-1, -1, 1, 2 };
```

Decoding

This section describes the algorithm used for decoding the 4 bit DVI ADPCM. This procedure must be followed for each block for each channel.

Get the first sample, **Samp0**, from the block header

Set the initial step table index, **Index**, from the block header

Output the first sample, **Samp0**

Set the previous Sample value:

SampX-1 = Samp0

While there are still samples to decode

Get the next sample code, **SampX Code**

Calculate the new sample:

Calculate the difference:

```
Diff = 0
if ( SampX Code & 4 )
Diff = Diff + StepTab[ Index ]
if ( SampX Code & 2 )
Diff = Diff + ( StepTab[ Index ] >> 1 )
if ( SampX Code & 1 )
Diff = Diff + ( StepTab[ Index ] >> 2 )
Diff = Diff + ( StepTab[ Index ] >> 3 )
```

Check the sign bit:

```
if ( SampX Code & 8 )
Diff = -Diff
SampX = SampX-1 + Diff
```

Check for overflow and underflow errors:

if **SampX** too large, make it the maximum allowable size (32767)

if **SampX** too small, make it the minimum allowable size (-32768)

Output the new sample, **SampX**

Adjust the step table index:

Index = Index + IndexTab[SampX Code]

Check for step table index overflow and underflow:

if **Index** too large, make it the maximum allowable size (88)

if **Index** too small, make it the minimum allowable size (0)

Save the previous Sample value:

SampX-1 = SampX

This section describes the algorithm used for decoding the 3 bit DVI ADPCM. This procedure must be followed for each block for each channel.

Get the first sample, **Samp0**, from the block header

Set the initial step table index, **Index**, from the block header

Output the first sample, **Samp0**

Set the previous Sample value:

SampX-1 = Samp0

While there are still samples to decode

Get the next sample code, **SampX Code**

Calculate the new sample:

Calculate the difference:

```
Diff = 0
if ( SampX Code & 2 )
Diff = Diff + StepTab[ Index ]
if ( SampX Code & 1 )
Diff = Diff + ( StepTab[ Index ] >> 1 )
Diff = Diff + ( StepTab[ Index ] >> 2 )
```

Check the sign bit:

```
if ( SampX Code & 4 )
Diff = -Diff
```

SampX = **SampX-1** + **Diff**

Check for overflow and underflow errors:

if **SampX** too large, make it the maximum allowable size (32767)

if **SampX** too small, make it the minimum allowable size (-32768)

Output the new sample, **SampX**

Adjust the step table index:

Index = Index + IndexTab[SampX Code]

Check for step table index overflow and underflow:

if **Index** too large, make it the maximum allowable size (88)

if **Index** too small, make it the minimum allowable size (0)

Save the previous Sample value:

SampX-1 = SampX

Encoding

This section describes the algorithm used for encoding the 4 bit DVI ADPCM. This procedure must be followed for each block for each channel.

For the first block only, clear the initial step table index:

Index = 0

Get the first sample, **Samp0**

Create the block header:

Write the first sample, **Samp0**, to the header

Write the initial step table index, **Index**, to the header

Set the previously predicted sample value:

PredSamp = Samp0

While there are still samples to encode, and we're not at the end of the block

Get the next sample to encode, **SampX**

Calculate the new sample code:

Diff = SampX - PredSamp

Set the sign bit:

```
if ( Diff < 0 )
SampX Code = 8
Diff = -Diff
else
SampX Code = 0
```

Set the rest of the code:

```
if ( Diff >= StepTab[ Index ] )
SampX Code = SampX Code | 4
Diff = Diff - StepTab[ Index ]
if ( Diff >= ( StepTab[ Index ] >> 1 )
SampX Code = SampX Code | 2
Diff = Diff - ( StepTab[ Index ] >> 1 )
if ( Diff >= ( StepTab[ Index ] >> 2 )
SampX Code = SampX Code | 1
```

Save the sample code, **SampX Code** in the block

Predict the current sample based on the sample code:

Calculate the difference:

```
Diff = 0
if ( SampX Code & 4 )
Diff = Diff + StepTab[ Index ]
if ( SampX Code & 2 )
Diff = Diff + ( StepTab[ Index ] >> 1 )
if ( SampX Code & 1 )
Diff = Diff + ( StepTab[ Index ] >> 2 )
Diff = Diff + ( StepTab[ Index ] >> 3 )
```

Check the sign bit:

```
if ( SampX Code & 8 )
Diff = -Diff
```

SampX = **SampX-1** + **Diff**

Check for overflow and underflow errors:

if **PredSamp** too large, make it the maximum allowable size (32767)

if **PredSamp** too small, make it the minimum allowable size (-32768)

Adjust the step table index:

Index = Index + IndexTab[SampX Code]

Check for step table index overflow and underflow:

if **Index** too large, make it the maximum allowable size (88)

if **Index** too small, make it the minimum allowable size (0)

This section describes the algorithm used for encoding the 3 bit DVI ADPCM. This procedure must be followed for each block for each channel.

For the first block only, clear the initial step table index:

Index = 0

Get the first sample, **Samp0**

Create the block header:

Write the first sample, **Samp0**, to the header

Write the initial step table index, **Index**, to the header

Set the previously predicted sample value:

PredSamp = Samp0

While there are still samples to encode, and we're not at the end of the block

Get the next sample to encode, **SampX**

Calculate the new sample code:

Diff = SampX - PredSamp

Set the sign bit:

if (**Diff** < 0)

SampX Code = 4

Diff = -**Diff**

else

SampX Code = 0

Set the rest of the code:

if (**Diff** >= **StepTab**[**Index**])

SampX Code = **SampX Code** | 2

Diff = **Diff** - **StepTab**[**Index**]

if (**Diff** >= (**StepTab**[**Index**] >> 1)

SampX Code = **SampX Code** | 1

Save the sample code, **SampX Code** in the block

Predict the current sample based on the sample code:

Calculate the difference:

Diff = 0

if (**SampX Code** & 2)

Diff = **Diff** + **StepTab**[**Index**]

if (**SampX Code** & 1)

Diff = **Diff** + (**StepTab**[**Index**] >> 1)

Diff = **Diff** + (**StepTab**[**Index**] >> 2)

Check the sign bit:

if (**SampX Code** & 4)

Diff = **-Diff**

SampX = **SampX-1** + **Diff**

Check for overflow and underflow errors:

if **PredSamp** too large, make it the maximum allowable size (32767)

if **PredSamp** too small, make it the minimum allowable size (-32768)

Adjust the step table index:

Index = Index + IndexTab[SampX Code]

Check for step table index overflow and underflow:

if **Index** too large, make it the maximum allowable size (88)

if **Index** too small, make it the minimum allowable size (0)

DSP Solutions formerly Digispeech Wave Types

Added: 05/22/92
Author: Digispeech

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_DIGISTD (0x0015)

 # define WAVE_FORMAT_DIGIFIX (0x0016)

wFormatTag	This must be set to either WAVE_FORMAT_DIGISTD or WAVE_FORMAT_DIGIFIX.
nChannels	Number of channels in the wave. (1 for mono)
nSamplesPerSec	Frequency the sample rate of the wave file. (8000). This value is also used by the fact chunk to determine the length in time units of the date.
nAvgBytesPerSec	Average data rate. (1100 for DIGISTD or 1625 for DigiFix) Playback software can estimate the buffer size using the value.
nBlockAlign	Block Alignment of 2 for DIGISTD and 26 for DigiFix. Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be zero.

The definition of the data contained in the Digistd and DigiFix formats are considered proprietary information of Digispeech. They can be contacted at:

DSP Solutions, Inc.
2464 Embarcadero Way
Palo Alto, CA 94303

The DIGISTD is a format used in a compression technique developed by Digispeech, Inc. DIGISTD format provides good speech quality with average rate of about 1100 bytes/second. The blocks (or buffers) in this format cannot be cyclically repeated.

The DigiFix is a format used in a compression technique developed by Digispeech, Inc. DigiFix format provides good speech quality (similar to DIGISTD) with average rate of exactly 1625 bytes/second. This format uses blocks of 26 bytes long.

Yamaha ADPCM

Added 09/25/92
Author: Yamaha

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_YAMAHA_ADPCM (0x0020)

wFormatTag	This must be set to WAVE_FORMAT_YAMAHA_ADPCM.	
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.	
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 5125, 7350, 9600, 11025, 22050, or 44100 Hz. Other sample rates are not allowed.	
nAvgBytesPerSec	Average data rate.. Playback software can estimate the buffer size using the value.	
nBlockAlign	This is dependent upon the number of bits per sample.	
	wBitsPerSample	nBlockAlign
	4	1
	4	1
wBitsPerSample	This is the number of bits per sample of YADPCM. Currently only 4 bits per sample is defined. Other values are reserved.	
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be zero.	

This format is created and read by Yamaha chip included in the Gold Sound Standard (GSS) that is implemented in a number of manufacturers boards. The algorithm and conversion routines are published in the source code provided in YADPCM.C with this technote.

Sonarc™ Compression

Added 10/21/92
Author: Sound Compression

Sound Compression has developed a new compression algorithm which, unlike ADPCM, is capable of lossless compression of digitized audio files to a degree far greater (50-60%) than that achievable with the other compressors, PKZIP and LHarc. "Lossy" compression is possible with even higher ratios. Information about the algorithm is available form the address below.

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

```
typedef struct sonarcwaveformat_tag {

WAVEFORMATEX wfx;

WORD wCompType;

} SONARCWAVEFORMAT
```

 # define WAVE_FORMAT_SONARC (0x0021)

wFormatTag	This must be set to WAVE_FORMAT_SONARC.
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 11025, 22050, or 44100 Hz. Other sample rates are not allowed.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the value.
nBlockAlign	The valid values have not been defined. Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of SONARC.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.
wCompType	This value is not yet defined..

"Sonarc" is a trademark of Speech Compression.

To get information on this format please contact:

Speech Compression
1682 Langley Ave.
Irvine, CA 92714
Telephone: 714-660-7727 Fax: 714-660-7155

Creative Labs ADPCM

Added 10/01/92
Author: Creative Labs

Createive has defined a new ADPCM compression scheme, and this new scheme will be implemented on their H/W and will be able to support compression and decompression real-time. They do not provide a description of this algorithm. Information about the algorithm is available form the address below.

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

```
typedef struct creative_adpcmwaveformat_tag {

WAVEFORMATEX wfx;

WORD wRevision;

} CREATIVEADPCMWAVEFORMAT
```

 # define WAVE_FORMAT_CREATIVE_ADPCM (0x0200)

wFormatTag	This must be set to WAVE_FORMAT_CREATIVE_ADPCM.		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.		
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 8000, 11025, 22050, or 44100 Hz. Other sample rates are not allowed.		
nAvgBytesPerSec	Average data rate.. Playback software can estimate the buffer size using the value.		
nBlockAlign	This is dependent upon the number of bits per sample.		
	wBitsPerSample	nChannels	nBlockAlign
	4	1	1
	4	2	1
	Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.		
wBitsPerSample	This is the number of bits per sample of CADPCM.		
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.		
wRevision	Revision of algorithm. This should be one for the current definition.		

To get information on this format please contact:

Creative Developer Support
1901, McCarthy Blvd, Milpitas, CA 95035.
Tel : 408-428 6644 Fax : 408-428 6655

DSP Group Wave Type

Added: 01/04/93
Author: Paul Beard, DSP Group

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_DSPGROUP_TRUESPEECH (0x0022)

wFormatTag	This must be set to WAVE_FORMAT_DSPGROUP_TRUESPEECH.
nChannels	Number of channels in the wave, 1 for mono.
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 8000
nAvgBytesPerSec	Average data rate.. (1067) Playback software can estimate the buffer size using the value.
nBlockAlign	This is the block alignment of the data in bytes. (32). Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of TRUESPEECH. Not used; set to zero.
cbExtraSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 32.
wRevision	Revision no (1,...)
nSamplesPerBlock	Number of samples per block. 240

```
= / * )
```

The definition of the data contained in the TRUESPEECH format is considered proprietary information of DSP Group Inc. They can be contacted at:

DSP Group Inc.,
4050 Moorpark Ave.,
San Jose CA. 95117
(408) 985 0722

TRUESPEECH is a format used in a compression technique developed by DSP Group Inc. TRUESPEECH format provides high quality telephony bandwidth voice vocoding with a rate of 1067 bytes per second. This format uses blocks of 32 bytes long.

Echo Speech Wave Type

Added: 01/21/93
Author: Echo Speech Corporation

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_ECHOSC1 (0x0023)

wFormatTag	This must be set to WAVE_FORMAT_ECHOSC1.
nChannels	Number of channels in the wave, always 1 for mono.
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 11025
nAvgBytesPerSec	Average data rate.. (450) Playback software can estimate the buffer size using the value.
nBlockAlign	This is the block alignment of the data in bytes. (6). Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample. Not used; set to zero.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 0.

The definition of the data contained in the ECHO SC-1 format is considered proprietary information of Echo Speech Corporation. They can be contacted at:

Echo Speech Corporation
6460 Via Real
Carpinteria, CA. 93013
805 684-4593

ECHO SC-1 is a format used in a compression technique developed by Echo Speech Corporation. ECHO SC-1 format provides excellent speech quality with an average data rate of exactly 450 bytes/second. This format uses blocks 6 bytes long.

ECHO is a registered trademark of Echo Speech Corporation.

AUDIOFILE Wave Type AF36

Added: April 29, 1993
Author: AudioFile

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_AUDIOFILE_AF36 (0x0024)

wFormatTag	This must be set to WAVE_FORMAT_AUDIOFILE_AF36
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

Audio File AF36 format provides very high compression for speech -based waveform audio. (Relative to 11 kHz, 16-bit PCM, a compression ratio of 36-to-1 is achieved with AF36.

For more information on AF36 and other AudioFile host-based and DSP based compression software contact: :

AudioFile, Inc.
Four Militia Drive
Lexington, MA, 02173
(617) 861-2996

Comment

Trademark info.

Audio Processing Technology Wave Type

Added: 06/22/93
Author: Calypso Software Limited

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_APTX (0x0025)

wFormatTag	This must be set to WAVE_FORMAT_APTX.
nChannels	Number of channels in the wave, always 1 for mono, 2 for stereo.
nSamplesPerSec	Frequency of the sample rate of the wave file. (8000, 11025, 22050, 44100, 48000)
nAvgBytesPerSec	Average data rate..= nChannels * nSamplesPerSec/2. (16bit audio) Playback software can estimate the buffer size using the value.
nBlockAlign	Should be set to 2 (bytes) for mono data or 4 (bytes) for stereo. For mono data 4 sixteen bit samples will be compressed into 1 sixteen bit word For stereo data 4 sizteen bit left channel samples will be compressed into the first 16bit word and 4 sixteen bit right channel samples will be cmpressed into the next 16 bit word. Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample. Not used; set to four.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 0.(zero)

The definition of the data contained in the APTX format is considered proprietary information of Audio Processing Technology Limited. They can be contacted at:

Audio Processing Technology Limited
Edgewater Road
Belfast, Northern Ireland, BT3 9QJ
Tel 44 232 371110
Fax 44 232 371137

This format is proprietary audio format using 4:1 compression i.c. 16 bits of audio are compressed to 4 bits. It is only encoded/decoded by dedicated hardware from MM_APT

AUDIOFILE Wave Type AF10

Added: June 22, 1993
Author: AudioFile

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_AUDIOFILE_AF10 (0x0026)

---

wFormatTag	This must be set to WAVE_FORMAT_AUDIOFILE_AF10
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

For more information on AF36 and other AudioFile host-based and DSP based compression software contact: :

AudioFile, Inc.
Four Militia Drive
Lexington, MA, 02173
(617) 861-2996

Dolby Labs AC-2 Wave Type

Added: 06/24/93
Author: Dolby Laboratories, Inc.

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the length of the data in samples.

WAVE Format Header

define WAVE_FORMAT_DOLBY_AC2 (0x0030)

wFormatTag	This must be set to WAVE_FORMAT_DOLBY_AC2
nChannels	Number of channels, 1 for mono, 2 for stereo
nSamplesPerSec	Three sample rates allowed: 48000, 44100, 32000 samples per second
nAvgBytesPerSec	Average data rate. ((nSamplesperSec*nBlockAlign)/512
nBlockAlign	The block alignment (in bytes) of the dat in . Given in table
nSamplesPerSec	nBlockAlign
48000	nChannels*168
44100	nChannels*184
32000	nChannels*190
wBitsPerSample	Approximately 3 bits per sample
cbExtraSize	2 extra bytes of information in format header
nAuxBitsCode	Auxiliary bits code indicating number of Aux. bits per block. The amount of audio data bits is reduced by this number in the decoder, such that the overall block size remains constant.
nAuxBitsCode	Number of Aux bits in block
0	0
1	8
2	16
3	32

specific structure of the chunk is proprietary, and may be obtained from Dolby Laboratories. Also contact Dolby for methods of including chunks.

Dolby Laboratories
100 Potrero Avenue
San Francisco, CA 94103-4813
Tel 415-558-0200

```
/* Dolby's AC-2 wave format structure definition */

typedef struct dolbyac2waveformat_tag {

WAVEFORMATEX wfx;

WORD nAuxBitsCode;

} DOLBYAC2WAVEFORMAT;
```

Sierra ADPCM

Added 07/26/93
Author: Sierra Semiconductor Corp.

Sierra Semiconductor has developed a compression scheme similar to the standard CCITT ADPCM. This scheme has been implemented in AriaÔ -based sound boards and is capable of supporting compression and decompression in real-time. A description of the algorithm is not available at this time.

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

```
typedef struct sierra_adpcmwaveformat_tag {

EXTWAVEFORMAT ewf;

WORD wRevision;

} SIERRAADPCMWAVEFORMAT;
```

 # define WAVE_FORMAT_SIERRA_ADPCM (0x0013)

wFormatTag	This must be set to WAVE_FORMAT_SIERRA_ADPCM.		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.		
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 22050 Hz. Other sample rates are not currently allowed.		
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the value.		
nBlockAlign	This is dependent upon the number of bits per sample.		
	wBitsPerSample	nChannels	nBlockAlign
	4	1	1
	4	2	1
	Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.		
wBitsPerSample	This is the number of bits per sample of Sierra ADPCM. Currently, only 4 bits per sample is defined. Other values are reserved.		
cbExtraSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.		
wRevision	Revision of algorithm. This should be 0x0100 for the current definition.		

VideoLogic Wave Types

Added: 07/13/93
Author: VideoLogic

Fact Chunck

Wave Format Header

 # define WAVE_FORMAT_MEDIASPACE_ADPCM (0x0012)

```
//

// VideoLogic's MediaSpace ADPCM structure definitions

//

// for WAVE_FORMAT_MEDIASPACE_ADPCM (0x0012)

//

//


typedef struct mediaspace_adpcmwaveformat_tag {

WAVEFORMATEX wfx;

WORD wRevision;

} MEDIASPACEADPCMWAVEFORMAT;

typedef MEDIASPACEADPCMWAVEFORMAT *PMEDIASPACEADPCMWAVEFORMAT;

typedef MEDIASPACEADPCMWAVEFORMAT NEAR *NPMEDIASPACEADPCMWAVEFORMAT;

typedef MEDIASPACEADPCMWAVEFORMAT FAR *LPMEDIASPACEADPCMWAVEFORMAT;
```

CCITT G.723 ADPCM

Added: 08/25/93
Author: Antex Electronics Corp.

The algorithm for G.721 header format is essentially the same as G723.

Fact Chunk

WAVE Format Header

 # define WAVE_FORMAT_G723_ADPCM (0x0014)

wFormatTag	This must be set to WAVE_FORMAT_G.723_ADPCM		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo		
nSamplesPerSec	Frequency the sample rate of the wave file. (8000, 11025, 22050, 44100)		
nAvgBytesPerSec	Average data rate Playback software can estimate the buffer size using the value.		
nBlockAlign	This is dependent upon the number of bits per sample.		
	wBitsPerSample	nChannels	nBlockAlign
	3	1	48 + nAuxBlockSize
	3	2	96 + nAuxBlockSize
	5	1	80 + nAuxBlockSize
	5	2	160 + nAuxBlockSize
	Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.		
wBitsPerSample	This is the number of bits per sample of data. (G.723 can be 3 or 5)		
cbExtraSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.		
nAuxBlockSize	This is the size in bytes of auxiliary data that is stored at the beginning of each data block. In most instances this should be set to 0.		

See the G.723 specification for algorithm details.

Data Format

Mono, 3 bits per sample

Grouped into 3 byte sub-blocks containing 8 mono samples. The bit ordering for samples labeled A through H is:

```
Byte 1 Byte 2 Byte 3
```

;where A2 is the MSB and A0 is the LSB of the first sample.

Stereo, 3 bits per sample

Grouped into 6 byte sub-blocks containing 8 stereo samples. The bit ordering for samples labeled A through H is:

```
Byte 1 Byte 2


Byte 3 Byte 4


Byte 5 Byte 6
```

;where AL2 is the MSB and AL0 is the LSB of the first left sample, and AR2 is the MSB and AR0 is the LSB of the first right sample

Mono, 5 bits per sample

Grouped into 5 byte sub-blocks containing 8 mono samples. The bit ordering for samples labeled A through H is:

```
Byte 1 Byte 2 Byte 3



Byte 4 Byte 5
```

;where A4 is the MSB and A0 is the LSB of the first sample.

Stereo, 5 bits per sample

Grouped into 10 byte sub-blocks containing 8 stereo samples. The bit ordering for samples labeled A through H is:

```
Byte 1 Byte 2


Byte 3 Byte 4


Byte 5 Byte 6


Byte 7 Byte 8
```

Byte 9 Byte 10

;where AL4 is the MSB and AL0 is the LSB of the first left sample, and AR4 is the MSB and AR0 is the LSB of the first right sample

Dialogic OKI ADPCM

Added: 04/07/94
Author: Dialogic

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_DIALOGIC_OKI_ADPCM (0x0203)

wFormatTag	This must be set to WAVE_FORMAT_DIALOGIC_OKI_ADPCM.
nChannels	Number of channels in the wave. 1
nSamplesPerSec	Frequency the of the sample rate of wave file. 6000, 8000,
nAvgBytesPerSec	Average data rate. 3000, 4000 Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of for the data. 1 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. 4
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. 0

This format can be created and read by either OKI ADPCM chip set of by a firmware program.

Control Resources Limited VQLPC

Added: 04/05/94
Author: Control Resources Limited

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_CONTROL_RES_VQLPC (0x0034)

wFormatTag	This must be set to WAVE_FORMAT_CONTROL_RES_VQLPC
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file. 8000
nAvgBytesPerSec	Average data rate.394 Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data in Bytes. 18 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. 4
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. 2
wCompType	This value is reserved and should be set to 1

VQLPC is trademarked of Control Resources Ltd.

Control Resources Limited CR10

Added: 04/05/94
Author: Control Resources Limited

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_CONTROL_RES_CR10 (0x0037)

---

wFormatTag	This must be set to WAVE_FORMAT_CONTROL_RES_CR10.
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

data not available at time of printing.

G.721 WAVE Format Header

Added: 08/25/93
Author: Antex Electronics Corp.

The algorithm for G.721 header format is essentially the same as G723.

Fact Chunk

WAVE Format Header

 # define WAVE_FORMAT_G721_ADPCM (0x0040)

wFormatTag	This must be set to WAVE_FORMAT_G721_ADPCM.	
nChannels	Number of channels in the wave.(1 for mono, 2 for stereo)	
nSamplesPerSec	Frequency the of the sample rate of wave file.	
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the value.	
nBlockAlign	Block Alignment of the data.	
	nChannels	nBlockAlign
	1	64+nAuxBlockSize
	2	128+nAuxBlockSize
	Playback software needs to process a multiple of bytes of data at a time, so that the value of can be used for buffer alignment.	
wBitsPerSample	This is the number of bits per sample of data. This should be 4.	
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 2.	
nAuxBlockSize	This is the size in bytes of auxiliary data that is stored at the beginning of each data block. In most instances this should be set to 0.	

See the G.721 specification for algorithm details.

This is a CCITT (International Telegraph and Telephone Consultative Committee) specification. Their address is:

Palais des Nations
CH-1211 Geneva 10, Switzerland
Phone: 22 7305111

Data Format

Mono, 4 bits per sample

Grouped into 1 byte sub-blocks containing 2 mono samples. The bit ordering for samples labeled A and B is:

;where A3 is the MSB and A0 is the LSB of the first sample and B3 is the MSB and B0 is the LSB of the second sample.

Stereo, 4 bits per sample

Grouped into 1 byte sub-blocks containing 1 stereo sample. The bit ordering for one stereo sample is:

;where L3 is the MSB and L0 is the LSB of the left sample, and R3 is the MSB and R0 is the LSB of the right sample

ADPCME WAVE Format Header

Added: 10/23/93
Author: Antex Electronics Corp.

Fact Chunk

WAVE Format Header

 # define WAVE_FORMAT_ADPCME (0x0033)

wFormatTag	This must be set to WAVE_FORMAT_ADPCME.
nChannels	Number of channels in the wave.(1 for mono, 2 for stereo)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data, 1 for Mono, 2 for Stereo.
wBitsPerSample	This is the number of bits per sample of data. This should be 4.
cbExtraSize	0

Data Format

Mono nibbles are labelled M and left and right samples labelled L and R.

```
Mono ADPCME

< M0|M1>

byte 0 byte 1 byte 2 byte 3

Stereo ADPCME

< L1|R1>

byte 0 byte 1 byte 2 byte 3
```

Note: Stereo nibble ordering is delibrately different from the mono order.

GSM610 Wave Type

Added: 09/05/93
Author: Microsoft

Fact Chunk

WAVE Format Header

typedef struct gsm610waveformat_tag {

WAVEFORMATEX wfx;
WORD wSamplesPerBlock;

} GSM610WAVEFORMAT;

typedef GSM610WAVEFORMAT *PGSM610WAVEFORMAT;

typedef GSM610WAVEFORMAT NEAR *NPGSM610WAVEFORMAT;

typedef GSM610WAVEFORMAT FAR *LPGSM610WAVEFORMAT;

#define WAVE_FORMAT_GSM610 (0x0031)

wFormatTag	This must be set to WAVE_FORMAT_GSM610
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

DSP Solutions REAL Wave Type

Added 02/03/94
Author: DSP Solutions (formerly Digispeech)

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE FORMAT HEADER

The extended wave format structure is used to defined all non-PCM format wave data, and is described as follows in the include file *mmreg.h*:

```
/* general extended waveform format structure */

/* Use this for all NON PCM formats */

/* (information common to all formats) */

typedef struct waveformat_extended_tag {

WORD wFormatTag; /* format type */

WORD nChannels; /* number of channels (i.e. mono, stereo...) */

DWORD nSamplesPerSec; /* sample rate */
```

```
DWORD nAvgBytesPerSec; /* for buffer estimation */

WORD nBlockAlign; /* block size of data */

WORD wBitsPerSample; /* Number of bits per sample of mono data */

WORD cbSize; /* The count in bytes of the extra size */} WAVEFORMATEX;
```

#define WAVE_FORMAT_DIGIREAL (0x0035)

wFormatTag	Must be set WAVE_FORMAT_DIGIREAL
nChannels	Number of channels in the wave, 1 for mono.
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 8000. Other sample rates are allowed, but not encouraged. This rate is also used by the sample size entry in the fact chunk to determine the length in time of the data.
nAvgBytesPerSec	Average data rate (1650). Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	The block alignment (in bytes) of the data in <**data-ck**> (13). Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample per sample of data. (2). Each channel is assumed to have the same sample resolution. If this field is not needed, then it should be set to zero.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. This should be 0.

DSP Solutions ADPCM Wave Type

Added 02/03/94
Author: DSP Solutions (formerly Digispeech)

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVEFORMATEX

The extended wave format structure is used to defined all non-PCM format wave data, and is described as follows in the include file *mmreg.h*:

```
/* general extended waveform format structure */

/* Use this for all NON PCM formats */

/* (information common to all formats) */

typedef struct waveformat_extended_tag {

WORD wFormatTag; /* format type */

WORD nChannels; /* number of channels (i.e. mono, stereo...) */

DWORD nSamplesPerSec; /* sample rate */

DWORD nAvgBytesPerSec; /* for buffer estimation */

WORD nBlockAlign; /* block size of data */

WORD wBitsPerSample; /* Number of bits per sample of mono data */

WORD cbSize; /* The count in bytes of the extra size */} WAVEFORMATEX;
```

#define WAVE_FORMAT_DIGIADPCM (0x0036)

wFormatTag	Must be set to WAVE_FORMAT_DIGIADPCM		
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo		
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 11025, 22050, or 44100. Other sample rates are allowed.		
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.		
nBlockAlign	The block alignment (in bytes) of the data in <**data-ck**>. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.		
	wBitsPerSample	nChannels	nBlockAlign
	3	1	3
	3	2	6
wBitsPerSample	This is the number of bits per sample per channel data. (3)		
cbSize	The size in bytes of the extra information in the WAVE format. Should be 0.		

MPEG-1 Audio (Audio-only)

Added 18/01/93
Author: Microsoft

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

 # define WAVE_FORMAT_MPEG (0x0050)

```
typedef struct mpeg1waveformat_tag {
WAVEFORMATEX wfx;
WORD fwHeadLayer;
DWORD dwHeadBitrate;
WORD fwHeadMode;
WORD fwHeadModeExt;
WORD wHeadEmphasis;
WORD fwHeadFlags;
DWORD dwPTSLow;
DWORD dwPTSHigh;
} MPEG1WAVEFORMAT;
```

wFormatTag	This must be set to WAVE_FORMAT_MPEG.
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.
nSamplesPerSec	Sampling frequency (in Hz) of the wave file: 32000, 44100, or 48000. Note, however, that if the sampling frequency of the data is variable, then this field should be set to zero. It is *strongly* recommended that a fixed sampling frequency be used for desktop applications.
nAvgBytesPerSec	Average data rate; this might not be a legal MPEG bit rate if variable bit rate coding under layer 3 is used.
nBlockAlign	The block alignment (in bytes) of the data in . For audio streams which have a fixed audio frame length, the block alignment is equal to the length of the frame. For streams in which the frame length varies, nBlockAlign should be set to 1.
	With a sampling frequency of 32 or 48 kHz, the size of an MPEG audio frame is a function of the bit rate. If an audio stream uses a constant bit rate, the size of the audio frames does not vary. Therefore, the following formulas apply: Layer 1: nBlockAlign = 4*(int)(12*BitRate/SamplingFreq) Layers 2 and 3: nBlockAlign = (int)(144*BitRate/SamplingFreq) Example 1: For layer 1, with a sampling frequency of 32000 Hz and a bit rate of 256 kbits/s, nBlockAlign = 384 bytes.
	If an audio stream contains frames with different bit rates, then the length of the frames varies within the stream. Variable frame lengths also occur when using a sampling frequency of 44.1 kHz: in order to maintain the data rate at the nominal value, the size of an MPEG audio frame is periodically increased by one "slot" (4 bytes in layer 1, 1 byte in layers 2 and 3) as compared to the formulas given above. In these two cases, the concept of block alignment is invalid. The value of nBlockAlign must therefore be set to 1, so that MPEG-aware applications can tell whether the data is block-aligned or not. Note that it is possible to construct an audio stream which has constant-length audio frames at 44.1 kHz by setting the *padding_bit* in each audio frame header to the same value (either 0 or 1). Note, however, that bit rate of the resulting stream will not correspond exactly to the nominal value in the frame header, and therefore some decoders may not be capable of decoding the stream correctly. In the interested of standardization and compatibility, this approach is discouraged.
wBitsPerSample	Not used; set to zero.
cbSize	The size in bytes of the extended information after the WAVEFORMATEX structure. For the standard WAVE_FORMAT_MPEG format, this is 22. If extra fields are added, this value will increase.
fwHeadLayer	The MPEG audio layer, as defined by the following flags: ACM_MPEG_LAYER1 - layer 1. ACM_MPEG_LAYER2 - layer 2. ACM_MPEG_LAYER3 - layer 3. Some legal MPEG streams may contain frames of different layers. In this case, the above flags should be ORed together so that a driver may tell which layers are present in the stream.
dwHeadBitrate	The bit rate of the data, in bits per second. This value must be a standard bit rate according to the MPEG specification; not all bit rates are valid for all modes and layers. See Tables 1 and 2, below. Note that this field records the actual bit rate, not MPEG frame header code. If the bitrate is variable, or if it is a non-standard bit rate, then this field should be set to zero. It is recommended that variable bit rate coding be avoided where possible.
fwHeadMode	Stream mode, as defined by the following flags: ACM_MPEG_STEREO - stereo. ACM_MPEG_JOINTSTEREO - joint-stereo. ACM_MPEG_DUALCHANNEL - dual-channel (for example, a bilingual stream). ACM_MPEG_SINGLECHANNEL - single channel. Some legal MPEG streams may contain frames of different modes. In this case, the above flags should be ORed together so that a driver may tell which modes are present in the stream. This situation is particularly likely with joint-stereo encoding, as encoders may find it useful to switch dynamically between stereo and joint-stereo according to the characteristics of the signal. In this case, both the ACM_MPEG_STEREO and the ACM_MPEG_JOINTSTEREO flags should be set.
	Contains extra parameters for joint-stereo coding; not used for other modes. See Table 3, below. Some legal MPEG streams may contain frames of different mode extensions. In this case, the values in Table 3 may be ORed together. Note that fwHeadModeExt is

fwHeadModeExt	only used for joint-stereo coding; for other modes (single channel, dual channel, or stereo), it should be set to zero. In general, encoders will dynamically switch between the various possible *mode_extension* values according to the characteristics of the signal. Therefore, for normal joint-stereo encoding, this field should be set to 0x000f. However, if it is desirable to limit the encoder to a particular type of joint-stereo coding, this field may be used to specify the allowable types.
wHeadEmphasis	Describes the de-emphasis required by the decoder; this implies the emphasis performed on the stream prior to encoding. See Table 4, below.
fwHeadFlags	Sets the corresponding flags in the audio frame header: ACM_MPEG_PRIVATEBIT - set the private bit. ACM_MPEG_COPYRIGHT - set the copyright bit. ACM_MPEG_ORIGINALHOME - sets the original/home bit. ACM_MPEG_PROTECTIONBIT - sets the protection bit, and inserts a 16-bit error protection code into each frame. ACM_MPEG_ID_MPEG1 - sets the ID bit to 1, defining the stream as an MPEG-1 audio stream. *This flag must always be set explicitly to maintain compatibility with future MPEG audio extensions (i.e. MPEG-2).* An encoder will use the value of these flags to set the corresponding bits in the header of each MPEG audio frame. When describing an encoded data stream, these flags represent a logical OR of the flags set in each frame header. That is, if the copyright bit is set in one or more frame headers in the stream, then the ACM_MPEG_COPYRIGHT flag will be set. Therefore, the value of these flags is not necessarily valid for every audio frame.
dwPTSLow	This field (together with the following field) consists of the presentation time stamp (PTS) of the first frame of the audio stream, as taken from the MPEG system layer. dwPTSLow contains the 32 LSBs of the 33-bit PTS. The PTS may be used to aid in the re-integration of an audio stream with an associated video stream. If the audio stream is not associated with a system layer, then this field should be set to zero.
dwPTSHigh	This field (together with the previous field) consists of the presentation time stamp (PTS) of the first frame of the audio stream, as taken from the MPEG system layer. The LSB of dwPTSHigh contains the MSB of the 33-bit PTS. The PTS may be used to aid in the re-integration of an audio stream with an associated video stream. If the audio stream is not associated with a system layer, then this field should be set to zero.
	Note: The previous two fields can be treated as a single 64-bit integer; optionally, the dwPTSHigh field can be tested as a flag to determine whether the MSB is set or cleared.

Table 1: Allowable Bit Rates (bits/s)

MPEG frame header code	Layer 1	Layer 2	Layer 3
'0000'	free format	free format	free format
'0001'	32000	32000	32000
'0010'	64000	48000	40000
'0011'	96000	56000	48000
'0100'	128000	64000	56000
'0101'	160000	80000	64000
'0110'	192000	96000	80000
'0111'	224000	112000	96000
'1000'	256000	128000	112000
'1001'	288000	160000	128000
'1010'	320000	192000	160000
'1011'	352000	224000	192000
'1100'	384000	256000	224000
'1101'	416000	320000	256000
'1110'	448000	384000	320000
'1111'	forbidden	forbidden	forbidden

Table 2: Allowable mode-bitrate combinations for Layer 2.

Bit rate (bits/sec)	Allowable modes
32000	single channel
48000	single channel
56000	single channel
64000	all modes
80000	single channel
96000	all modes
112000	all modes
128000	all modes
160000	all modes
192000	all modes
224000	stereo, intensity stereo, dual channel
256000	stereo, intensity stereo, dual channel
320000	stereo, intensity stereo, dual channel
384000	stereo, intensity stereo, dual channel

Table 3: Mode Extension

fwHeadModeExt	MPEG frame header code	Layers 1 and 2	Layers 3
0x0001	'00'	subbands 4-31 in intensity stereo	no intensity or ms-stereo coding
0x0002	'01'	subbands 8-31 in intensity stereo	intensity stereo
0x0004	'10'	subbands 12-31 in intensity stereo	ms-stereo
0x0008	'11'	subbands 16-31 in intensity stereo	both intensity and ms-stereo coding

Table 4: Emphasis Field

wHeadEmphasis	MPEG frame header code	De-emphasis required
1	'00'	no emphasis
2	'01'	50/15 ms emphasis
3	'10'	reserved
4	'11'	CCITT J.17

Flags

The following flags are defined for the **fwHeadLayer** field. For encoding, one of these flags should be set so that the encoder knows what layer to use. For decoding, the driver can check these flags to determine whether it is capable of decoding the stream. Note that a legal MPEG stream may use different layers in different frames within a single stream. Therefore, more than one of these flags may be set.

```
#define ACM_MPEG_LAYER1 (0x0001)

#define ACM_MPEG_LAYER2 (0x0002)

#define ACM_MPEG_LAYER3 (0x0004)
```

The following flags are defined for the **fwHeadMode** field. For encoding, one of these flags should be set so that the encoder knows what layer to use; for joint-stereo encoding, typically the ACM_MPEG_STEREO and ACM_MPEG_JOINTSTEREO flags will both be set so that the encoder can use joint-stereo coding only when it is more efficient than stereo. For decoding, the driver can check these flags to determine whether it is capable of decoding the stream. Note that a legal MPEG stream may use different layers in different frames within a single stream. Therefore, more than one of these flags may be set.

```
#define ACM_MPEG_STEREO (0x0001)

#define ACM_MPEG_JOINTSTEREO (0x0002)

#define ACM_MPEG_DUALCHANNEL (0x0004)

#define ACM_MPEG_SINGLECHANNEL (0x0008)
```

Table 3 defines flags for the **fwHeadModeExt** field. This field is only used for joint-stereo coding; for other encoding modes, this field should be set to zero. For joint-stereo encoding, these flags indicate the types of joint-stereo encoding which an encoder is permitted to use. Normally, an encoder will dynamically select the mode extension which is most appropriate for the input signal; therefore, an application would typically set this field to 0x000f so that the encoder may select between all possibilities; however, it is possible to limit the encoder by clearing some of the flags. For an encoded stream, this field indicates the values of the MPEG *mode_extension* field which are present in the stream.

The following flags are defined for the **fwHeadFlags** field. These flags should be set before encoding so that the appropriate bits are set in the MPEG frame header. When describing an encoded MPEG audio stream, these flags represent a logical OR of the corresponding bits in the header of each audio frame. That is, if the bit is set in any of the frames, it is set in the **fwHeadFlags** field. If an application wraps a RIFF WAVE header around a pre-encoded MPEG audio bit stream, it is responsible for parsing the bit stream and setting the flags in this field.

```
#define ACM_MPEG_PRIVATEBIT (0x0001)

#define ACM_MPEG_COPYRIGHT (0x0002)

#define ACM_MPEG_ORIGINALHOME (0x0004)

#define ACM_MPEG_PROTECTIONBIT (0x0008)

#define ACM_MPEG_ID_MPEG1 (0x0010)
```

Data

The data chunk consists of an MPEG-1 audio sequence as defined by the ISO 11172 specification, part 3 (audio). This sequence consists of a bit stream, which is stored in the data chunk as an array of bytes. Within a byte, the MSB is the first bit of the stream, and the LSB is the last bit. The data is *not* byte-reversed. For example, the following data consists of the first 16 bits (from left to right) of a typical audio frame header:

Syncword ID Layer ProtectionBit ...

111111111111 1 10 1 ...

This data would be stored in bytes in the following order:

Byte0 Byte1 ...

FF FD ...

MPEG Audio Frames

An MPEG audio sequence consists of a series of audio frames, each of which begins with a frame header. Most of the fields within this frame header correspond to fields in the MPEG1WAVEFORMAT structure defined above. For encoding, these fields can be set in the MPEG1WAVEFORMAT structure, and the driver can use this information to set the appropriate bits in the frame header when it encodes. For decoding, a driver can check these fields to determine whether it is capable of decoding the stream.

Encoding

A driver which encodes an MPEG audio stream should read the header fields in the MPEG1WAVEFORMAT structure and set the corresponding bits in the MPEG frame header. If there is any other information which a driver requires, it must get this information either from a configuration dialog box, or through a driver callback function. For more information, see the Ancillary Data section, below.

If a pre-encoded MPEG audio stream is wrapped with a RIFF header, it is the responsibility of the application to parse the bit stream and set the fields in the MPEG1WAVEFORMAT structure. If the sampling frequency or the bitrate index is not constant throughout the data stream, the driver should set the corresponding MPEG1WAVEFORMAT fields (**nSamplesPerSec** and **dwHeadBitrate**) to zero, as described above. If the stream contains frames of more than one layer, it should set the flags in **fwHeadLayer** for all layers which are present in the stream. Since fields such as **fwHeadFlags** can vary from frame to frame, caution must be used in setting and testing these flags; in general, an application should not rely on them to be valid for every frame. When setting these flags, adhere to the following guidelines:

• ACM_MPEG_COPYRIGHT should be set if any of the frames in the stream have the copyright bit set.

• ACM_MPEG_PROTECTIONBIT should be set if any of the frames in the stream have the protection bit set.

• ACM_MPEG_ORIGINALHOME should be set if any of the frames in the stream have the original/home bit set. This bit may be cleared if a copy of the stream is made.

• ACM_MPEG_PRIVATEBIT should be set if any of the frames in the stream have the private bit set.

• ACM_MPEG_ID_MPEG1 should be set if any of the frames in the stream have the ID bit set. For MPEG-1 streams, the ID bit should always be set; however, future extensions of MPEG (such as the MPEG-2 multi-channel format) may have the ID bit cleared.

If the MPEG audio stream was taken from a system-layer MPEG stream, or if the stream is intended to be integrated into the system layer, then the presentation time stamp (PTS) fields may be used. The PTS is a field in the MPEG system layer which is used for synchronization of the various fields. The MPEG PTS field is 33 bits, and therefore the RIFF WAVE format header stores the value in two fields: **dwPTSLow** contains the 32 LSBs of the PTS, and **dwPTSHigh** contains the MSB. These two fields may be taken together as a 64-bit integer; optionally, the **dwPTSHigh** field may be tested as a flag to determine whether the MSB is set or cleared. When extracting an audio stream from a system layer, a driver should set the PTS fields to the PTS of the first frame of the audio data. This may later be used to re-integrate the stream into the system layer. *The PTS fields should not be used for any other purpose.* If the audio stream is not associated with the MPEG system layer, then the PTS fields should be set to zero.

Decoding

A driver may test the fields in the MPEG1WAVEFORMAT structure to determine whether it is capable of decoding the stream. However, the driver must be aware that some fields, such as the **fwHeadFlags** field, may not be consistent for every frame in the bit stream. A driver should never use the fields of the MPEG1WAVEFORMAT structure to perform the actual decoding. The decoding parameters should be taken entirely from the MPEG data stream.

A driver may check the **nSamplesPerSec** field to determine whether it supports the sampling frequency specified. If the MPEG stream contains data with a variable sampling rate, then the **nSamplesPerSec** field will be set to zero. If the driver cannot handle this type of data stream, then it should not attempt to decode the data, but should fail immediately.

Ancillary Data

The audio data in an MPEG audio frame may not fill the entire frame. Any remaining data is called *ancillary data*. This data may have any format desired, and may be used to pass additional information of any kind. If a driver wishes to support the ancillary data, it must have a facility for passing the data to and from the calling application. The driver may use a callback function for this purpose. Basically, the driver may call a specified callback function whenever it has ancillary data to pass to the application (i.e. on decode) or whenever it requires more ancillary data (on encode).

Drivers should be aware that not all applications will want to process the ancillary data. Therefore, a driver should only provide this service when explicitly requested by the application. The driver may define a custom message which enables and disables the callback facility. Separate messages could be defined for the encoding and decoding operations for more flexibility.

If the callback facility is enabled, then the application is responsible for creating a callback function which is capable of processing the ancillary data. Typically, the application already has a callback defined in order to feed data blocks to the wave device as they are needed; this callback processes the WOM_CLOSE, WOM_DONE, and WOM_OPEN messages, and/or the WIM_CLOSE, WIM_DATA, and WIM_OPEN messages. The address of the callback function (or a window handle) is passed to the driver by the waveOutOpen or the waveInOpen calls in the **dwCallback** parameter. Two additional messages must defined by the driver and supported by the callback: one to pass ancillary data back to the application (i.e. WOM_ANCDATA_OUT), and one to request ancillary data from the application (i.e. WIM_ANCDATA_IN).

As message parameters, the WOM_ANCDATA_OUT could pass a pointer to a data buffer, and a size parameter indicating the number of bits (or bytes) of data in the buffer. The buffer would be allocated by the driver, and freed after the message has been processed by the callback. The driver could pass back the ancillary data frame by frame as it is received, or it could process an entire block of data and pass back the ancillary data in a single large chunk. The method is up to the driver, or could be configurable either through a configuration dialog or as a parameter passed when the ancillary data functions are enabled by the application.

To request ancillary data, the WIM_ANCDATA_IN message could pass a pointer to an empty data buffer, which the callback function would fill with ancillary data. If the amount of ancillary data varies from frame to frame, the first few bytes of the buffer could be defined to be the number of bits (or bytes) of data. This buffer would be allocated and freed by the driver; in order to ensure that there is enough space to hold all the data, the buffer size could be configurable using either a configuration dialog or by passing the value to the driver as a parameter when the ancillary data functions are enabled by the application.

Note that this method may not be appropriate for all drivers or all applications; it is included only as an illustration of how ancillary data may be supported. For more information, consult the Windows 3.1 Software Development Kit, "Multimedia Programmers Reference," and the Windows 3.1 Device Driver Kit, "Multimedia Device Adaptation Guide."

Standards

It is recommended that applications use the 44.1 kHz sampling rate whenever possible, to maintain compatibility with current computer standards. It is also recommended that encoders avoid the use of variable bitrate coding, and it is *strongly* recommended that all bit streams use a constant sampling frequency. Streams which have a variable sampling frequency cannot be decoded to PCM for manipulation by other audio services.

References

ISO/IEC JTC1/SC29/WG11 MPEG, April 1992. ISO/IEC Draft International Standard: "Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s."

Creative Labs, Inc. FastSpeech 8 & 10

Added: 03/2/94

Author: Creative Labs

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_CREATIVE_FASTSPEECH8 (0x0202)

#define WAVE_FORMAT_CREATIVE_FASTSPEECH10 (0x0203)

wFormatTag	This must be set to WAVE_FORMAT_CREATIVE_FASTSPEECH8 or 10
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file. 8000 or 11025
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of for the data. 32 for FASTSPEECH8 and 26 for FASTSPEECH10 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbExtraSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. 2.
wRevision	Revision of the Algorithm. This should be 1 for the current definition.

Fujitsu FM Towns SND Wave Type

Added: 02/15/94
Author: Fujitsu

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_FM_TOWNS_SND (0x0300)

wFormatTag	This must be set to WAVE_FORMAT_FM_TOWNS_SND
nChannels	Number of channels in the wave. 1
nSamplesPerSec	Frequency the of the sample rate of wave file. 0-20833
nAvgBytesPerSec	Average data rate. Same as sampling rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of for the data. Always 1 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. Always 8.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

Olivetti GSM

Added: 01/20/94
Author: Olivetti

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_OLIGSM (0x1000)

wFormatTag	This must be set to WAVE_FORMAT_OLIGSM
nChannels	Number of channels in the wave.(1 for mono), 2
nSamplesPerSec	Frequency the of the sample rate of wave file. 8000
nAvgBytesPerSec	Average data rate. 1633
	Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. 196 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. 2
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. 0

Olivetti ADPCM

Added: 01/20/94
Author: Olivetti

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_OLIADPCM (0x1001)

wFormatTag	This must be set to WAVE_FORMAT_OLIADPCM.
nChannels	Number of channels in the wave. (1, 2)
nSamplesPerSec	Frequency the of the sample rate of wave file. 8000
nAvgBytesPerSec	Average data rate. 4000 Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. 480 Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data. 4
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header. 0

Olivetti CELP

Added: 01/20/94
Author: Olivetti

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_OLISBC (0x1003)

wFormatTag	This must be set to WAVE_FORMAT_OLISBC.
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.

Olivetti OPR

Added: 01/20/94
Author: Olivetti

Fact Chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

WAVE Format Header

#define WAVE_FORMAT_OLIOPR (0x1004)

more data not available at time of printing.

wFormatTag	This must be set to WAVE_FORMAT_OLIOPR.
nChannels	Number of channels in the wave.(1 for mono)
nSamplesPerSec	Frequency the of the sample rate of wave file.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	Block Alignment of the data. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample of data.
cbSize	The size in bytes of the extra information in the extended WAVE 'fmt' header.