Bilag 3

Microsoft®
Multimedia

Standards Update

New Multimedia Data Types and Data Techniques

June 11, 2004
Revision: 1.0.99

Table of Contents

Overview....5
- Where to Look for Information....5
- Intended Audience....5
- Versions of this Document....6
- Questions?....6

New Chunks....7
- Display Chunk....7
- JUNK (Filler) Chunk....7
- PAD (Filler) Chunk....7
- Wave RIFF form sub-Chunks....9
  - Fact Chunk....9
  - Cue Points Chunk....9
    - Examples of File Position Values....10
  - Playlist Chunk....11
  - Associated Data Chunk....11
    - Label and Note Information....11
    - Text with Data Length Information....13

New Forms....13

New WAVE Types....14
  - Fact Chunk....14
  - EXTWAVEFORMAT....14
- Microsoft ADPCM 15
  - Fact Chunk....15
  - WAVE Format Header....15
  - Block....16
  - Data....16
  - Padding....17
  - ADPCM Algorithm....17
  - Decoding....17
  - Encoding....18
  - Sample C Code....19
- CVSD Wave Type....20
  - Fact Chunk....20
  - WAVE Format Header....20
- CCITT Standard Companded Wave Types....21
  - Fact Chunk....21
  - WAVE Format Header....21
- OKI ADPCM Wave Types....22
  - Fact Chunk....22
  - WAVE Format Header....22
- DVI ADPCM Wave Type....23
  - Fact Chunk....23
  - WAVE Format Header....23
- Digispeech Wave Types....24
  - Fact Chunk....24
  - WAVE Format Header....24
- Unknown Wave Type....25
  - Fact Chunk....25
  - WAVE Format Header....25

DIB File Additions....26
- RGB555 and RGB565 DIB Formats....26
  - BITMAPINFOHEADER Structure for RGB555 and RGB565 DIBs.... 26
  - RGB555 and RGB565 Pixel Encoding....27

RIFF Clipboard Formats....28

CF_RIFF....28
CF_WAVE....28
Registered Clipboard Formats....28
Encoding Language of Text....29
Country Codes....29
Language and Dialect Codes....30

Overview

This standards update presents new and updated information for dealing with multimedia data under Microsoft Windows. This document is also available as part of the Multimedia Developer Registration Kit. The MDRK is used to register multimedia data and ids as well as new MCI command sets.. This document is the result of companies requesting and registering new data types. This document builds on the standard RIFF documentation that is contained in:

1. The Multimedia Development Kit (MDK) 1.0 Programmer's Reference
2. The Windows 3.1 Software Development Kit (SDK)'s Multimedia Programmer's Reference
3. The Multimedia Programmer's Reference book from Microsoft Press

The RIFF file format is a standard published as a joint design document by IBM and Microsoft. This standards document is Multimedia Programming Interface and Data Specifications 1.0 published in August 1991. The first draft of this document was issued in November, 1990. This IBM/Microsoft document is available from the sources listed below.

This standards update assumes that the reader has read the concepts defined in these documents.

New RIFF file forms and chunks are defined in this document. The new RIFF forms and chunks defined here have been registered with Microsoft. If you want to register your own RIFF forms and chunks, please request a Multimedia Developer Registration Kit by call (206) 936-8644 or writing to:

Microsoft
Multimedia Product Management
One Microsoft Way
Redmond, WA 98052-6399
FAX: (206) 93MSFAX

In addition, techniques for dealing with multimedia data in the system, such as clipboard data, are defined in this document.

Where to Look for Information

Current versions of this document as well as other technical update and technical notes and sample code are available from:

1. CompuServe WINSDK forum
2. Microsoft Multimedia BBS at (206) 936-4082 in the *files* library in the *specs* section in the RIFFNEW.ZIP file. Sample code is available in the *samples* section and technical notes are available in the *technote* section. BBS modem settings are 9600 baud, no parity, 8 data bits, 1 stop bit.
3. Via anonymous FTP on `ftp.uu.net` in the *vendors\microsoft\multimedia* directory. Sample code is in the *samples* directory and technical notes are in the *technote* directory.
4. Current versions of any document may be ordered (currently free) by calling (206) 936-8644.

Intended Audience

This document should be read by "multimedia producers" (as defined in the Multimedia Authoring Guide) as well as programmers using all types of tools. You should read this document after reading the base RIFF file format definitions.

Versions of this Document

This document is continually being updated and expanded. Eventually the information presented in this document will be placed in the standard reference for the RIFF and multimedia data standards from Microsoft, such as the Multimedia Programmer's Reference from MS-Press.

When referring to standards defined in this document, please refer to the data and version number printed on the cover page.

Versions of this document with the same version number are just expanded additions of the same information. However, when the version number changes, the information contained in the previous version will be moved to the standard reference locations for RIFF and multimedia data standards.

Questions?

If you have questions, requests, or problems with this technical update, you should send your question to the address above.

New Chunks

These new chunks have been defined for use in any RIFF form.

Display Chunk

Added: 05/01/92
Author: Microsoft

A DISP chunk contains easily rendered and displayable objects associated with an instance of a more complex object in a RIFF form (e.g. sound file, AVI movie).

A DISP chunk is defined as follows:

```
<DISP_ck>    1      DISP( <type> <data> )
```

<type> is a `DWORD` (32 bit unsigned quantity in Intel format) that identifies <data> as one of the standard Windows clipboard formats (`CF_METAFILE, CF_DIB, CF_TEXT`, etc.) as defined in *windows.h*.

The `DISP` chunk should be used as a direct child of the RIFF chunk so that any RIFF aware application can find it. There can be multiple `DISP` chunks with each containing different types of displayable data, but all representative of the same object. The `DISP` chunks should be stored in the file in order of preference (just as in the clipboard).

The `DISP` chunk is especially beneficial when representing OLE data within an application. For example, when pasting a wave file into Excel, the creating application can use the `DISP` chunk to associate an icon and a text description to represent the embedded wave file. This text should be short so that it can be easily displayed in menu bars and under icons.

Note: do not use a `CF_TEXT` for a description of the data. Bibliographic data chunks will be added to support the standard MARC (Machine Readable Cataloging) data.

JUNK (Filler) Chunk

Added: 05/01/92
Author: IBM, Microsoft

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

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

where **<filler>** contains random data.

PAD (Filler) Chunk

Added: 07/15/92
Author: Microsoft

A PAD chunk represents padding. It contains no relevant data; it is a space filler of arbitrary size. When duplicating the file, the copier should maintain the padding of the PAD chunk. Specifically, if the PAD chunk makes the next chunk align on a 2K boundary in the physical file, then this alignment should be preserved even if the size of the PAD chunk must change. The PAD chunk is defined as follows:

```
<PAD chunk> → PAD( <filler> )
```

where **<filler>** contains random data.

Wave RIFF form sub-Chunks

Added: 05/01/92
Author: Microsoft, IBM

Most of the information in this section comes directly from the IBM/Microsoft RIFF standard document.

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

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

The WAVE chunks are described in the following sections.

Fact Chunk

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

```
<fact-ck>   1   fact( <dwSampleLength:DWORD> )
```

<**dwSampleLength**> represents the length of the data in samples. The <**nSamplesPerSec**> field from the wave format header is used in conjunction with the <**dwSampleLength**> field to determine the length of the data in seconds.

The `fact` chunk is required for all new `WAVE` formats. The chunk is not required for the standard WAVE_FORMAT_PCM files.

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

Cue Points Chunk

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

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

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

The <**cue-point**> fields are as follows:

Field	Description
dwName	Specifies the cue point name. Each **<cue-point>** record must have a unique **dwName** field.
dwPosition	Specifies the sample position of the cue point. This is the sequential sample number within the play order. See "Playlist Chunk," later in this document, for a discussion of the play order.
fccChunk	Specifies the name or chunk ID of the chunk containing the cue point.
dwChunkStart	Specifies the position of the start of the data chunk containing the cue point. This should be zero if there is only one chunk containing data (as is currently always the case).
dwBlockStart	Specifies the position of the start of the block containing the position. This is the byte offset from the start of the data section of the chunk, not the chunk's FOURCC.
dwSampleOffset	Specifies the sample offset of the cue point relative to the start of the block.

Examples of File Position Values

The following table describes the **<cue-point>** field values for a WAVE file containing a single '`data`' chunk:

Cue Point Location	Field	Value
Within PCM data	**fccChunk**	FOURCC value '`data`'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the sample (nBlockAlign aligned bytes) relative to the start of the data section of the '`data`' chunk (not the FOURCC).
	dwSampleOffset	Sample position of the cue point relative to the start of the '`data`' chunk.
In all other 'data' chunks	**fccChunk**	FOURCC value '`data`'.
	dwChunkStart	Zero value.
	dwBlockStart	File position of the enclosing block relative to the start of the data section of the '`data`' chunk (not the FOURCC). The software can begin the decompression at this point.
	dwSampleOffset	Sample position of the cue point relative to the start of the block.

Playlist Chunk

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

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

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

The <**play-segment**> fields are as follows:

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

Associated Data Chunk

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

```
<assoc-data-list> →      LIST(   'adtl'
                                 <labl-ck>              // Label
                                 <note-ck>              // Note
                                 <ltxt-ck> }            // Text with data
length

<labl-ck> →              labl(   <dwName:DWORD>
                                 <data:ZSTR> )

<note-ck> →              note(   <dwName:DWORD>
                                 <data:ZSTR> )

<ltxt-ck> →              ltxt(   <dwName:DWORD>
                                 <dwSampleLength:DWORD>
                                 <dwPurpose:DWORD>
                                 <wCountry:WORD>
                                 <wLanguage:WORD>
                                 <wDialect:WORD>
                                 <wCodePage:WORD>
                                 <data:BYTE>... )
```

Label and Note Information

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

Field	Description

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

Text with Data Length Information

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

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

New Forms

Currently None

New WAVE Types

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.

EXTWAVEFORMAT

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 */
                    /* SPECIFY TOTAL OR EXTRA */
} 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.
cbExtraSize	The size in bytes of the extra information in the WAVE format header.

```
#define             WAVE_FORMAT_UNKNOWN             (0x0000)
#define             WAVE_FORMAT_ADPCM               (0x0002)
#define             WAVE_FORMAT_IBM_CVSD            (0x0005)
#define             WAVE_FORMAT_ALAW                (0x0006)
#define             WAVE_FORMAT_MULAW               (0x0007)
#define             WAVE_FORMAT_OKI_ADPCM           (0x0010)
#define             WAVE_FORMAT_DIGISTD             (0x0015)
#define             WAVE_FORMAT_DIGIFIX             (0x0016)
```

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 {
      EXTWAVEFORMAT     ewf;
      WORD              nSamplesPerBlock;
      WORD              nNumCoef;
      ADPCMCOEFSET      aCoeff[nNumCoef];
} 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 <nAvgBytesPerSec> value.
nBlockAlign	The block alignment (in bytes) of the data in <data-ck>.

	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.
cbExtraSize	The size in bytes of the entire WAVE format chunk.
	For the standard WAVE_FORMAT_ADPCM 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:
<Byte1> <Byte2>...<ByteN> ...<Byte((nSamplesPerBlock-2)/2)>
where <ByteN> has <High Order Bit ... Low OrderBit> or < (Sample 2N + 2) (Sample 2N + 3)>

```
      <ByteN> =  ((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:
<Byte1> <Byte2>...<ByteN> ...<Byte(nSamplesPerBlock-2)>
where <ByteN> has <High Order Bit ... Low OrderBit> or
< (Left Channel of Sample N + 2) (Right Channel of Sample N + 2)>

```
      <ByteN> =  ((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: 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 <nAvgBytesPerSec> value.
nBlockAlign	Set to 2048 to provide efficient caching of file from CD-ROM.
	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 always 1 for CVSD.
cbExtraSize	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, 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.)
cbExtraSize	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 {
       EXTWAVEFORMAT   ewf;
       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)		
cbExtraSize	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.

DVI ADPCM Wave Type

Added: Pending
Author: Microsoft, Intel

This definition is pending and is not yet final. Do not use this definition.

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 {
       EXTWAVEFORMAT   ewf;
       WORD    wPole;
 } 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	Frequency 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. (DVI is 4)		
cbExtraSize	The size in bytes of the extra information in the extended `WAVE 'fmt'` header. This should be 2.		
wPole	High frequency emphasis value.		

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 <nAvgBytesPerSec> value.
nBlockAlign	Block Alignment of 2 for DIGISTD and 26 for DigiFix.
	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. This should be 2.

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

Digispeech, 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.

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

This format type should be used during development of a new type. This type should only be used internally at a company during development before the new WAVE type is registered.

```
#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 for 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.
cbExtraSize	The size in bytes of the extra information in the extended `WAVE 'fmt'` header.

DIB File Additions

These are new biCompression types for the DIB and RDIB file formats.

These new DIB data formats can be passed to any Windows device driver by passing the correct BITMAPINFOHEADER structure when using RGB555 and RGB565 formats.

RGB555 and RGB565 DIB Formats

Efficient utilization of the new video modes provided by new video cards requires a new format to accommodate 16-bit RGB DIBs. Standard 8-bit DIBs (256 colors) use a color table to encode the color information. The new 16-bit RGB DIBs do not have a color table, but encode the color information directly into the 16 bits representing each pixel. There are two types of 16-bit RGB DIBs:

1 RGB555 — 32K colors using five bits each for red, green, and blue.
1 RGB565 — 64K colors using five bits each for red and blue, and six bits for green.

BITMAPINFOHEADER Structure for RGB555 and RGB565 DIBs

The following table shows how to set up the fields of the BITMAPINFOHEADER structure for RGB555 and RGB565 DIBs:

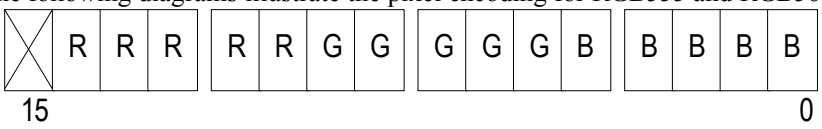
Field	Description
biSize	Size in bytes of the BITMAPINFOHEADER structure.
biWidth	Width of the bitmap in pixels.
biHeight	Height of the bitmap in pixels.
biPlanes	Set to 1.
biBitCount	Set to 16.
biCompression	For RGB555, set to 0 (BI_RGB). For RGB565, set to the four-character code ‘R565’.
biSizeImage	Size in bytes of the image.
biXPelsPerMeter	Horizontal resolution in pixels per meter.
biYPelsPerMeter	Vertical resolution in pixels per meter.
biClrUsed	Set to 0.
biClrImportant	Set to 0.

The following code fragment shows how to create the four-character code required in the **biCompression** field for RGB565 DIBs:

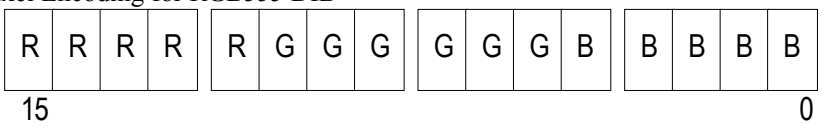
```
#include <mmsystem.h>
...

bmih.biCompression = mmioFOURCC('R', '5', '6', '5');
```

RGB555 and RGB565 Pixel Encoding

The following diagrams illustrate the pixel encoding for RGB555 and RGB565 DIBs:



Pixel Encoding for RGB555 DIB



Pixel Encoding for RGB565 DIB

RIFF Clipboard Formats

CF_RIFF

Windows 3.1 defines a new clipboard format, CF_RIFF, that allows any RIFF form to be encoded into the clipboard.

CF_WAVE

Windows 3.1 defines a new clipboard format, CF_WAVE, that allows any RIFF form of type WAVE to be encoded into the clipboard.

Registered Clipboard Formats

Because the only way to tell the form of RIFF clipboard data is to read it, an application cannot know if it wants to read the CF_RIFF format or not without getting the data and parsing it. Usually it just wants to look at the form type to determine if it is interested in the data that it contained in the clipboard.

In addition, encoding multiple forms involves a complicated compound RIFF file.

To overcome these problems, Microsoft has defined a standard way to register RIFF clipboard formats. The application should call the Windows API *RegisterClipboardFormat* with a string that specifies the RIFF form of the type that the application is interested. The string should be constructed as follows:

```
RIFF <FORM>[[' '| u | l][' '| u | l][' '| u | l][' '| u | l]]
```

where <**form**> is the FOURCC of the form, including spaces. The registration is case insensitive, so form types that have different cases must be uniquely registered. This is accomplished by adding designations of the case of the FOURCC when the <**form**> is not all upper-case.

If any of the characters in the <**form**> are lower-case, then the entire <**form**> must be represented by case designations. Case is designated by appending four characters that represent the case of each character in the <**form**>. The designations are '`u`' for uppercase, '`l`' for lower-case, and '` `' for space. All non-alphabetics should be represented as spaces.

For example, the form '`Isp `' would be registered as "`RIFF Isp ull `". The first character is upper case and therefore the designation character is 'u'. The next two characters are lower-case and therefore the designation characters are both 'l'. The last character is a non-alpha and the designation is therefore a space. As another example, '`L245`' would be registered as "`RIFF L245 U   `"

The CF_RIFF and CF_WAVE formats should still be created in the clipboard in addition to any registered clipboard formats.

Encoding Language of Text

The following fields and values should be used when the encoding of text's language is important.

Country Codes

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

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

Language and Dialect Codes

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

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