Enhanced Metafiles in Win32

Abstract

With the advent of the Microsoft® Win32™ Application
Programming Interface (API), a new metafile format called
enhanced metafiles has been introduced. The overall design goal
of the enhanced metafile was to describe a picture without any
coding restrictions and to make a metafile easier to use.
Enhanced metafiles have many advantages over the older
Windows metafiles found in Microsoft Windows™ version 3.1
(Win16). Improvements found in the enhanced metafile include an
expanded header, a description string, a metafile palette, and an
increase in the number and type of graphics device interface
(GDI) functions that may be recorded. In addition to these
enhancements, the metafile record and playback code in Win32
has been designed to remove all of the restrictions that applied to
Windows metafiles with respect to scaling, clipping, embedding,
and querying, among others. To top it off, enhanced metafiles
may be played on any device in a device-independent manner.
This article describes the differences between Windows metafiles
and enhanced metafiles. Sample code that illustrates the basic
concepts of creating and playing enhanced metafiles is provided
at the end of this article.

Introduction

The tried-and-true Microsoft® Windows™ metafile has been an
invaluable aid to the development of numerous drawing and
presentation applications for Windows. However, the "vanilla"
Windows metafile did not address issues related to scalability
and device independence. Left on their own, developers
attempted to address this issue in various ways. Some
developers embedded application, location, or scaling comments
in the metafiles. This resulted in extremely nonportable metafiles.
Others added headers to the metafile that provided various
application-specific information. The net result of most of these
efforts was, once again, nonportable metafiles. However, one of
these endeavorsplaceable metafilescaught on. Developed by
Aldus Corporation, placeable metafiles include a 22-byte header
that provides, among other things, mapping and measurement
information that can be used to scale the metafile.

The proliferation of the placeable metafile, other homegrown
formats, and the confusion of many developers regarding the use
of metafiles led to a demand for a metafile format that addressed
all of the development community's needs. Thus the Win32
enhanced metafile was born. Developed by Microsoft, the
enhanced metafile distinguishes itself from the Windows metafile
in that it is device-independent and much easier to use. Easier to
use? You bet! Remember having to code two paths to deal with
drawing? One code path drew to the screen; the second code
path drew to metafiles. The only way to get around this was to
use a subset of graphics device interface (GDI) functions that
used logical coordinates. Although this permitted limited scaling
capabilities, it restricted the use of many helpful GDI functions.

You definitely couldn't query the metafile device context (DC) for information such as window origins and extents. With the advent of the enhanced metafile, those restrictions are unnecessary! A single code path is all that is required to draw to any DC, whether it be a metafile, screen, or printer DC. Furthermore, you no longer need to use a subset of GDI; for example, you can now do the following:

```
DeleteObject(SelectObject(hdcMeta,
hbrOldBrush));
```

Yes, the old object versus TRUE or FALSE is returned by SelectObject when used with a metafile DC. This was not possible with Windows metafiles and is a good indication of the potential for success of the enhanced metafile. Finally, in an enhanced metafile, you can query the current position in the client area.
But what about all of those Windows metafiles? There are thousands of them in the marketplace. It would be a shame to see them go to waste. With this in mind, Win32 functions were written that convert Windows metafiles to enhanced metafiles. However, enhanced metafiles cannot be used in Win16. Figure 1 illustrates the compatibility of the two metafile formats and environments.

Figure 1. Metafile compatibility in the Win16 and Win32 environments
This article discusses the differences between the Windows metafile and the enhanced metafile, the format of the enhanced metafile, its features, and techniques for its use.

Windows Metafiles vs. Enhanced Metafiles

A Windows metafile is used for applications written using the Windows version 3.x application programming interface (API). The format of a Windows metafile consists of a header and an array of metafile records. Windows metafiles are limited in their capabilities and should rarely be used in Win32-based applications. That being said, the Windows metafile functions are supported in Win32 to maintain backward compatibility with applications that use the older Windows metafiles.

An enhanced metafile is used in applications written using the Win32 API. (Win32s, however, does not implement enhanced metafiles.) The enhanced format consists of a header, a table of handles to GDI objects, a private palette, and an array of metafile records. Enhanced metafiles provide true device independence. You can think of the picture stored in an enhanced metafile as a snapshot of the video display taken at a particular moment. This snapshot maintains its dimensions no matter where it appears: on a printer, a plotter, the desktop, or in the client area of any Win32-based application.

Metafile Structure

At first glance, Windows metafiles and enhanced metafiles share the same overall structure. They are an array of variable-length structures called metafile records. The first records in the metafile specify general information such as the resolution of the device on which the picture was created, the dimensions of the picture, and so on. The remaining records, which constitute the bulk of any metafile, correspond to the GDI functions required to draw the picture.

A closer inspection reveals a number of differences between them, as shown in Figure 2. Unlike the Windows metafile, the enhanced metafile has a different header and may include a description string and an optional palette stored in a special end-of-file record. The enhanced metafile also provides support for additional types of records.

Figure 2. Structure of a Windows metafile and an enhanced metafile

Enhanced metafile header

The major difference between the Windows metafile header and the enhanced metafile header is that the Windows metafile header contains only size and version information, whereas the enhanced metafile header contains dimension and resolution information, as well as size and version information. The Windows metafile header has the following form:

```
typedef struct tagMETAHEADER {
    WORD  mtType;
    WORD  mtHeaderSize;
    WORD  mtVersion;
    DWORD mtSize;
    WORD  mtNoObjects;
    DWORD mtMaxRecord;
    WORD  mtNoParameters;
} METAHEADER;
```

In contrast, notice the added dimension and resolution information in the code for an enhanced metafile header below. This information is used by the metafile playback functions to achieve device independence:

```
typedef struct tagENHMETAHEADER
{
  DWORD  iType;         // Record type
EMR_HEADER.
  DWORD  nSize;         // Record size in
bytes. This may be greater
                // than the
sizeof(ENHMETAHEADER).
  RECTL   rclBounds;      //
Inclusive-inclusive bounds in device units.
  RECTL   rclFrame;       //
Inclusive-inclusive Picture Frame of
```

```
                        // metafile in .01
mm units.
  DWORD  dSignature;     // Signature.
Must be ENHMETA_SIGNATURE.

  DWORD  nVersion;       // Version number.
  DWORD  nBytes;         // Size of the
metafile in bytes.
  DWORD  nRecords;       // Number of
records in the metafile.
  WORD   nHandles;       // Number of
handles in the handle table.
                        // Handle index
zero is reserved.
  WORD   sReserved;      // Reserved.  Must
be zero.
  DWORD  nDescription;   // Number of chars
in the unicode description string.
                        // This is 0 if
there is no description string.

  DWORD  offDescription; // Offset to the
metafile description record.
                        // This is 0 if
there is no description string.
  DWORD  nPalEntries;    // Number of
entries in the metafile palette.
  SIZEL  szlDevice;      // Size of the
reference device in pixels.
  SIZEL  szlMillimeters; // Size of the
reference device in millimeters.
} ENHMETAHEADER;
```

A good question at this point might be, "How much of the header information in the enhanced metafile do I need to provide?" When you call CreateEnhMetafile, you provide a long pointer to a rectangle that specifies the picture frame, and you provide a DC that serves as a reference. The members szlDevice and szlMillimeters are derived from the reference DC. You can also provide a long pointer to a string that describes the metafile.

rclFrame

The picture frame is stored in the metafile header structure member rclFrame. When you create the metafile using CreateEnhMetaFile, a pointer to a RECT structure (lpRect) is among the parameters. It is this rectangle that specifies the picture frame. The left and top members of the RECT structure must be values less than the right and bottom members, respectively. Points along the edges of the rectangle are included in the picture. If lpRect is NULL, GDI computes the dimensions of the smallest rectangle that surrounds the picture. The lpRect parameter should be provided whenever possible.

nDescription and offDescription

Information regarding the description string is maintained in two structure members of the metafile header, nDescription and offDescription. This string is also specified when you create the metafile using CreateEnhMetaFile. The lpDescription parameter contains the address of the description string. When the metafile is created, the length of the description string is stored in the metafile header. When the metafile is closed, GDI writes the string to the metafile and updates offDescription in the header.

nPalEntries

As palettes are created and selected into a metafile DC, GDI accumulates the palette entries and places them in a "metafile palette." This palette is located in the EMR_EOF record. An application can store the palette in an enhanced metafile by calling either the CreatePalette or SetPaletteEntries function and the SelectPalette function before creating the picture. nPalEntries is updated as the palette is collected in the metafile palette.

Enhanced metafile records

The Windows metafile record and the enhanced metafile record are similar in structure and size (see code below). However, before breathing a sigh of relief, take a closer look at the record structure. The size and type members are reversed. This could be a potential pitfall when porting existing 16-bit Windows-based applications to Win32. Take note that the array of parameters is now an array of DWORD values to accommodate the 32-bit girth of GDI.

```
typedef struct tagMETARECORD
  {
    DWORD     rdSize;      // Record size
in bytes
    WORD      rdFunction;   // Record type
META_XXX
    WORD      rdParm[1];    // WORD array
of parameters
  } METARECORD;
```

```
typedef struct tagENHMETARECORD
{
    DWORD   iType;          // Record
type EMR_XXX
    DWORD   nSize;          // Record
size in bytes
    DWORD   dParm[1];       // DWORD
Array of parameters
} ENHMETARECORD;
```

Several new metafile records have been added to an already extensive list of records. Table 1 lists the records found in enhanced metafile records with their corresponding iType values,

which can be found in WINGDI.H. Some of the records seem to be similar, for example, EMR_EXTTEXTOUTA and EMR_EXTTEXTOUTW. The A specifies that the text is based on ANSI and the W indicates that it is based on UNICODE. Another similar pair is EMR_POLYLINE and EMR_POLYLINE16. The 16 indicates that GDI has converted the points for PolyLine to 16 bits for the purpose of saving space in the metafile.

Table 1. Enhanced Metafile Records

Record	Value	Record	Value
EMR_ABORTPATH	68		
EMR_POLYLINE	4		
EMR_ANGLEARC	41		
EMR_POLYLINE16	87		
EMR_ARC	45		
EMR_POLYLINETO	6		
EMR_ARCTO	55		
EMR_POLYLINETO16	89		
EMR_BEGINPATH	59		
EMR_POLYPOLYGON	8		
EMR_BITBLT	76		
EMR_POLYPOLYGON16	91		
EMR_CHORD	46		
EMR_POLYPOLYLINE	7		
EMR_CLOSEFIGURE	61		
EMR_POLYPOLYLINE16	90		
EMR_CREATEBRUSHINDIRECT	39		
EMR_POLYTEXTOUTA	96		
EMR_CREATEDIBPATTERNBRUSHPT	94		
EMR_POLYTEXTOUTW	97		
EMR_CREATEMONOBRUSH	93		
EMR_REALIZEPALETTE	52		
EMR_CREATEPALETTE	49		
EMR_RECTANGLE	43		
EMR_CREATEPEN	38		
EMR_RESIZEPALETTE	51		
EMR_DELETEOBJECT	40		
EMR_RESTOREDC	34		
EMR_ELLIPSE	42		
EMR_ROUNDRECT	44		
EMR_ENDPATH	60		
EMR_SAVEDC	33		
EMR_EOF	14		
EMR_SCALEVIEWPORTEXTEX	31		
EMR_EXCLUDECLIPRECT	29		
EMR_SCALEWINDOWEXTEX	32		
EMR_EXTCREATEFONTINDIRECTW	82		
EMR_SELECTCLIPPATH	67		
EMR_EXTCREATEPEN	95		
EMR_SELECTOBJECT	37		
EMR_EXTFLOODFILL	53		
EMR_SELECTPALETTE	48		

EMR_EXTSELECTCLIPRGN   75
EMR_SETARCDIRECTION   57
EMR_EXTTEXTOUTA 83
EMR_SETBKCOLOR   25

EMR_EXTTEXTOUTW 84
EMR_SETBKMODE   18
EMR_FILLPATH   62
EMR_SETBRUSHORGEX   13
EMR_FILLRGN 71
EMR_SETCOLORADJUSTMENT   23
EMR_FLATTENPATH   65
EMR_SETDIBITSTODEVICE   80
EMR_FRAMERGN   72
EMR_SETMAPMODE   17
EMR_GDICOMMENT   70
EMR_SETMAPPERFLAGS   16
EMR_HEADER 1
EMR_SETMETARGN   28
EMR_INTERSECTCLIPRECT   30
EMR_SETMITERLIMIT 58
EMR_INVERTRGN   73
EMR_SETPALETTEENTRIES   50
EMR_LINETO   54
EMR_SETPIXELV   15
EMR_MASKBLT   78
EMR_SETPOLYFILLMODE   19
EMR_MODIFYWORLDTRANSFORM   36
EMR_SETROP2 20

EMR_MOVETOEX   27
EMR_SETSTRETCHBLTMODE 21
EMR_OFFSETCLIPRGN 26
EMR_SETTEXTALIGN 22
EMR_PAINTRGN   74
EMR_SETTEXTCOLOR 24
EMR_PIE   47
EMR_SETVIEWPORTEXTEX   11
EMR_PLGBLT 79
EMR_SETVIEWPORTORGEX   12
EMR_POLYBEZIER   2
EMR_SETWINDOWEXTEX   9
EMR_POLYBEZIER16 85
EMR_SETWINDOWORGEX   10
EMR_POLYBEZIERTO 5
EMR_SETWORLDTRANSFORM 35
EMR_POLYBEZIERTO16   88
EMR_STRETCHBLT   77
EMR_POLYDRAW   56
EMR_STRETCHDIBITS 81
EMR_POLYDRAW16   92
EMR_STROKEANDFILLPATH   63

EMR_POLYGON   3
EMR_STROKEPATH   64
EMR_POLYGON16   86

Of the records listed in Table 1, two are present in every enhanced metafile. The first record in any enhanced metafile is the metafile header. The value of this record is EMR_HEADER (1). The last record of an enhanced metafile is always the end-of-file record. The value of this record is EMR_EOF (14). In addition to the enhanced metafile header and metafile records, two additional pieces of data may be found in an enhanced metafile. The optional description string follows the enhanced metafile header. An optional color palette, if it exists, is contained in a special enhanced metafile record, the EMR_EOF record. The EMR_EOF is present even when a palette is not available.

Description string

Have you ever just wanted to know what was in a given metafile without having to decipher a cryptic filename or play back the entire metafile? The enhanced metafile provides an optional description string that provides exactly this information. In addition to a descriptive name, the string specifies the name of the application that created the picture. The string must contain a null character between the application name and the picture name. It must terminate with two null characters; for example, "ACME Inc.\0Rocket Skates\0\0", where \0 represents the null character. If lpDescription is NULL, there is no corresponding entry in the header of the enhanced metafile. If the description string is present, it is found offDescription bytes from the beginning of the ENHMETAHEADER structure. The array found at that offset contains nDescription characters. A convenient way to obtain the description string is to use the function GetEnhMetaFileDescription.

Color palette

When a palette was required in a Windows metafile, you recorded a CreatePalette, SelectPalette, and RealizePalette sequence. When the Windows metafile was played back, the palette was selected as a foreground palette. The realization of the foreground palette typically resulted in odd screen behavior as the other palettes went to the background. With the enhanced metafiles, palette sequences may still be recorded, but they are never selected and realized as foreground palettes when they are subsequently played back. These palette functions serve only to build the metafile palette. Enhanced metafiles place this optional palette in the metafile end-of-file record (EMR_EOF). Although the palette is optional, there are advantages to using it. One palette may be generated and used for the duration of the playback, thus avoiding the problems associated with foreground and background palette changes. The optional palette also makes it easier for a palette-oriented application to examine the metafile colors and merge them with an existing palette. The easiest way to get the palette is to call GetEnhMetaFilePaletteEntries. However, you can locate the palette yourself if you wish. First, determine whether there is a

palette. This is done by examining nPalEntries in the enhanced metafile header or in the last record of the metafile, the EMR_EOF record (see code below).

```
typedef struct tagEMREOF
{
    EMR    emr;          // Base enhanced
metafile record.
    DWORD   nPalEntries;   // Number of
palette entries.
    DWORD   offPalEntries; // Offset to the
palette entries.
    DWORD   nSizeLast;     // Same as
emr.nSize and must be the
                   // last DWORD of
the record.  The palette
                   // entries, if
they exist, precede this field.
} EMREOF;
```

If this value is greater than zero, a palette is present. The nSizeLast member of the EMR_EOF record indicates how many bytes to seek back to find the beginning of the EMR_EOF record. Seek forward from this point by offPalEntries and bingo! you have a palette location. After having used either method to obtain the palette, you can simply select the palette into the destination DC, realize it, and then play back the metafile.

Device Independence

Achieving device independence was very difficult, if not impossible, with Windows metafiles. The placeable variant of the Windows metafile provided the best shot at this. The additional header provided in the placeable metafile (see code below) provided an opportunity for an application to render these metafiles in a device-relative way.

```
typedef struct tagPLACEABLEMETAFILEHEADER {
    DWORD   key;
    HANDLE  hmf;
    RECT    bbox;
    WORD    inch;
    DWORD   reserved;
    WORD    checksum;
} PLACEABLEMETAFILEHEADER;
```

Device independence was typically achieved by setting the mapping mode to anisotropic, setting the viewport extents to the physical dimensions of the device, and finally setting the windows extents to the product of the device's physical dimensions (in inches) and the metafile units per inch (contained in the inch member of the header structure). The biggest problem with this approach was that variants of the placeable Windows metafile began surfacing. Often the mapping mode and viewport extents were included in the metafile as records. This necessitated enumerating the metafile as a method of filtering

out undesirable records. Unfortunately, the bounding box and metafile units per inch often did not match the environment being set by the undesirable metafile records! This led to the situation in which even the placeable metafiles were, once again, application-specific.

Device independence is a key feature of enhanced metafiles. The Microsoft Win32 Software Development Kit (SDK) for Windows NT Programmer's Reference: Overviews states that "...when an application creates a picture measuring 2 inches by 4 inches on a VGA display and stores that picture in a metafile, it (the picture) will maintain those original dimensions when it is printed on a 300 dpi laser printer or copied over a network and displayed in another application that is running on an 8514/A video display." So, just how is this done? The key to achieving this device independence is the use of a reference device context, that is, the context of the device on which the picture was created. When a metafile is created, information regarding the reference DC is placed in the enhanced metafile header. More specifically, GDI calls GetDeviceCaps and assigns the HORZSIZE and VERTSIZE return values to szlMillimeters and assigns the HORZRES and VERTRES return values to szlDevice. The rclFrame member is assigned the bounding rectangle specified in the lpRect parameter of CreateEnhMetaFile. If lpRect is NULL, GDI determines the bounding rectangle and assigns it to rclFrame. This information is sufficient to enable the playback functions to provide device independence. When a metafile is played back, the picture undergoes a series of transformations that scale and translate the picture to the output rectangle that was specified in the call to the PlayEnhMetaFile or EnumEnhMetaFile playback functions. These transformations rely on the dimensions of the picture frame (rclFrame), the dimensions of the device upon which the metafile was created (szlMillimeters and szlDevice), and the world-to-page transform values currently set in the destination DC.

Compatibility

Although it is not recommended, Windows metafiles can be used with Win32-based applications. Unfortunately, enhanced metafiles cannot be used in Windows version 3.x. The Win32 API provides these familiar-sounding functions that manipulate Windows metafiles:

CloseMetaFile   Closes a Windows metafile DC.
CopyMetaFile   Copies a Windows metafile.
CreateMetaFile   Creates a Windows metafile DC.
DeleteMetaFile   Invalidates Windows metafile handle.
EnumMetaFile   Returns GDI calls within a Windows metafile.
EnumMetaFileProc   Processes metafile data.
GetMetaFile   Creates a Windows metafile.
GetMetaFileBitsEx   Copies Windows metafile bits to a buffer.
PlayMetaFile   Plays a Windows metafile to a

DC.

PlayMetaFileRecord   Plays a Windows metafile record.
SetMetaFileBitsEx   Creates a memory-based
Windows metafile from data.
GetMetaFileBits Obsolete; use
GetMetaFileBitsEx.
SetMetaFileBits   Obsolete; use
SetMetaFileBitsEx.

In addition to providing functions that enable the use of Windows
metafiles, the Win32 API also provides functions to convert
Windows metafiles into enhanced metafiles. These include the
following functions:

GetWinMetaFileBits   Retrieves enhanced metafile
contents in Windows format.
SetWinMetaFileBits   Creates enhanced metafile from
Windows metafile data.

Pulling It All Together

At this point, the differences between a Windows metafile and an
enhanced metafile should be clear:

        The enhanced metafile header is larger and more complete
than a Windows metafile.
        The enhanced metafile may contain a description string or a
palette.
        The enhanced metafile achieves device independence by
means of a reference DC and special transformations in the
playback functions.

A quick look at the enhanced metafile functions and some
example code should help clarify the features of enhanced
metafiles.

Enhanced Metafile Functions

The following functions are very similar to the functions used for
Windows metafiles. The differences that exist do so to
accommodate new features of enhanced metafiles.

CloseEnhMetaFile   Closes an enhanced
metafile DC.
CopyEnhMetaFile   Copies an enhanced
metafile.
CreateEnhMetaFile   Creates an enhanced
metafile DC.
DeleteEnhMetaFile   Invalidates enhanced
metafile handle.
EnhMetaFileProc Processes enhanced
metafile data.
EnumEnhMetaFile   Returns GDI calls within
an enhanced metafile.
GdiComment   Adds a comment to an

enhanced metafile.
GetEnhMetaFile Creates an enhanced
metafile.

GetEnhMetaFileBits   Copies enhanced metafile
bits to a buffer.
GetEnhMetaFileDescription   Returns creator and title
for enhanced metafile.
GetEnhMetaFileHeader   Returns enhanced
metafile header.
GetEnhMetaFilePaletteEntries   Returns enhanced
metafile palette entries.
PlayEnhMetaFile Plays an enhanced
metafile to a DC.
PlayEnhMetaFileRecord   Plays an enhanced
metafile record.
SetEnhMetaFileBits   Creates a memory-based
enhanced metafile from data.

GdiComment

GdiComment deserves a little elaboration. When a comment
was needed in a Windows metafile, the MFCOMMENT printer
escape was used. These comments were restricted to private
data only. The MFCOMMENT printer escape cannot be used in
enhanced metafiles. Escapes, in general, cannot be used in
enhanced metafiles because they would introduce device
dependence, which is in direct opposition to the goal of device
independence. Realizing that there is still a place for private data
in metafiles, the architects of the Win32 API made GdiComment
 available for embedding private information in enhanced
metafiles. But GdiComment is more than simply an alternative
to MFCOMMENT. It was designed to enable public comments as
well. The currently supported public comments include:

        GDICOMMENT_WINDOWS_METAFILE
        GDICOMMENT_BEGINGROUP
        GDICOMMENT_ENDGROUP
        GDICOMMENT_MULTIFORMATS

The use of public comment permits embedding of other metafiles
and encapsulated PostScript (EPS) files within the metafile. The
multiformat public comment is the most exciting of the
comments. (I must be losing touch with reality!) If an EPS file is
embedded in an enhanced metafile and subsequently played
back, GDI will select the best format for the device!
Transparently! When I first heard about the multiformat comment,
I was sure that I was going to be expending a great deal of effort
writing code for rendering EPS files. I was relieved to find out how
wrong I was!

Coding Examples

The example code in the following sections demonstrates the
creation and playback of an enhanced metafile, illustrating how
some of these functions are used. (These examples are

pared-down versions of examples in the Win32 documentation.)

Creating an enhanced metafile

Creating an enhanced metafile is similar to creating a Windows metafile. The code that follows demonstrates the creation of an enhanced metafile that is stored on a disk. The example uses a device context for the application window as the reference DC. The dimensions of the application's client area are used to define the dimensions of the picture frame. Using the rectangle dimensions returned by the GetClientRect function, the example converts the device units to .01-millimeter units and passes the converted values to the CreateEnhMetaFile function. The example also embeds a text description of the picture in the header of the enhanced metafile.

```
// Obtain a handle to a reference DC.

hdcRef = GetDC(hWnd);

// Determine the picture frame dimensions.
// iWidthMM is the display width in
millimeters.
// iHeightMM is the display height in
millimeters.
// iWidthPels is the display width in pixels.
// iHeightPels is the display height in
pixels.

iWidthMM = GetDeviceCaps(hdcRef, HORZSIZE);
iHeightMM = GetDeviceCaps(hdcRef, VERTSIZE);
iWidthPels = GetDeviceCaps(hdcRef, HORZRES);
iHeightPels = GetDeviceCaps(hdcRef, VERTRES);


// Use iWidthMM, iWidthPels, iHeightMM, and
iHeightPels to determine the
// number of .01-millimeter units per pixel
in the x and y directions.

iMMPerPelX = (iWidthMM * 100)/iWidthPels;
iMMPerPelY = (iHeightMM * 100)/iHeightPels;

// Retrieve the coordinates of the client
rectangle in pixels.

GetClientRect(hWnd, &rect);

// Convert client coordinates to .01-mm
units.

rect.left = rect.left * iMMPerPelX;
rect.top = rect.top * iMMPerPelY;
rect.right = rect.right * iMMPerPelX;

rect.bottom = rect.bottom * iMMPerPelY;
```

```
// Create the metafile DC.

hdcMeta = CreateEnhMetaFile(hdcRef,
(LPTSTR)"MYFILE.EMF", &rect,
                (LPSTR)"ACME
Inc.\0Rocket Skates\0\0");

if (!hdcMeta)
    errhandler("CreateEnhMetaFile", hWnd);

// Release the reference DC.

ReleaseDC(hWnd, hdcRef);
```

Playing an enhanced metafile

Playing an enhanced metafile is also similar to the method used
to play Windows metafiles. The following example demonstrates
how to open an enhanced metafile stored on disk, and displays
the associated picture in the client area. The example passes
the handle returned by the GetEnhMetaFile function to the
PlayEnhMetaFile function in order to display the picture. Before
diving into the code, consider the following advice about
enumeration of the metafile and some tips on how to maximize
the advanced features of the GDI metafile player.

Using EnumEnhMetaFile

It's common practice to enumerate Windows metafiles, rather
than simply to play them back, to achieve better control over
positioning, scaling, getting access to application-specific
comments, or manipulating the palette records. However, the
improvements to enhanced metafiles reduce the need for
enumeration of the metafile. In Win32, most applications need to
use only PlayEnhMetaFile unless they need to edit the
enhanced metafile by adding, deleting or modifying records, in
which case they'd use EnumEnhMetaFile.

Advanced features

Three advanced features of enhanced metafiles require action by
the application before playing the metafile to the destination DC:

        Advanced palette functionality
        Advanced clipping capabilities
        World-to-page transform values

The advanced palette functionality provides a means of
examining the palette before playing the metafile. This is useful if
the palette is to be merged with another or optimized before the
enhanced metafile is played or enumerated. If the metafile palette
is to be used, it must be retrieved (
GetEnhMetaFilePaletteEntries), manipulated as desired,
created, selected, and realized in the destination DC.
The advanced clipping capabilities permit the enhanced metafile

to be clipped to a predetermined clipping region. To accomplish this, the metafile player determines if a clipping region exists in the destination DC. If a clipping region exists, the region is applied to the metafile contents as they are played. To use the clipping feature, create and select any clipping regions into the destination DC prior to playing the metafile.

Finally, the metafile player applies world-to-page transform values set in the destination DC to the contents of the enhanced metafile. If any scaling, rotation, reflection, or shearing is desired, set the world-to-page transform value in the destination DC before playing the metafile:

```
hemf =
GetEnhMetaFile((LPSTR)"MYFILE.EMF");
// Open the metafile.

hDC = GetDC(hWnd);               //
Retrieve a handle to a window DC.

GetClientRect(hWnd, &rect);  // Retrieve the
client rectangle dimensions.

PlayEnhMetaFile(hDC, hemf,
&rect);                 // Draw the picture.

DeleteEnhMetaFile(hemf);              //
Release the metafile handle.

ReleaseDC(hWnd,
hDC);                    // Release
the window DC.
```

Summary

Enhanced metafiles are a giant step beyond the Windows metafile. An expanded metafile header, a description string, a palette, device independence, and ease of porting from the Windows metafile format make enhanced metafiles an offer you can't refuse! It is expected that the advanced features of enhanced metafiles will make the use of metafiles more acceptable than Windows metafiles. With the features listed below, it is easy to understand why enhanced metafiles will become an invaluable tool for Win32-based applications:

          Full transformation support (removes scaling restrictions found in Windows metafiles)
          Unrestricted clipping capabilities
          Improved palette support
          Query support (as in GetViewportExtent and GetCurrentPositionEx)
          Advanced embedding features (metafiles and EPS files)

Probably the most important point to make about enhanced metafiles is that there is very little reason to enumerate the metafile. The playback code is smarter and does much of what

developers have had to do themselves by means of enumeration
for years.