WMF - *ein unterschätztes Graphikformat*

Robert P. Michelic DSK-515:\WMF

Im Zeitalter der GIF- und JPEG-Dateien, wo wir megabyteweise aufwendig Bilder durch das Internet pumpen, ist es vielleicht etwas altmodisch, über das WMF-Format (Windows-Meta-File-Format) zu schreiben. Da aber meines Erachtens dieses Format für viele Windows-Anwendungen nicht zu unterschätzende Vorteile hat, lohnt es sich doch, sich ein bißchen genauer damit zu beschäftigen.

Zunächst ein paar allgemeine Informationen: Das GDI (Graphical Device Interface) nimmt dem Programmierer unter Windows viel Arbeit ab. Ein Hauptvorteil von Windowsprogrammen ist ja, daß zu einzelnen Programmen keine Hardwarespezifischen Treiber mehr entwickelt werden müssen, weil zum Ansteuern von Druckern und Bildschirmen eine standardisierte Schnittstelle zur Verfügung steht. Wer sich vor Augen führt, mit wieviel Treibern etwa die letzten DOS-Word Versionen geliefert wurden, weiß wovon ich schreibe.

Wenn also eine standardisierte Schnittstelle, etwa zur Ausgabe einer Graphik, verwendet wird, liegt es nahe, die standardisierten Anweisungen, die ja die Graphik ergeben, zusammenzufassen und als eigenes Graphikformat zu verwenden.

Genau das ist eine WMF-Datei. Für (nahezu) jede GDI-Funktion gibt es einen Code, und mit den notwendigen Daten wird der gesamte Funktionsaufruf in einem Record gespeichert. In Pascal-Syntax schaut das so aus:

```
TMetaRecord=record
    rdSize: Longint;     {Information über die Länge der Structur in Word}
    rdFunction: Word;    {Funktionscode}
    rdParam: array[0..0] of Word         {Parameter}
end;
```

Naheliegender Vorteil: Das Abarbeiten einer solchen Datei, d.h. das Anzeigen oder Drucken einer in diesem Format gespeicherten Graphik, ist ohne zusätzliche Interpretationsarbeit, somit also schnell möglich.

Weiterer Vorteil: WMF-Dateien sind relativ kompakt, da Sie ja von der Idee her schon die Graphikinformation in komprimierter Form enthalten.

Nachteil: WMF-Dateien lassen sich nur dort sinnvollerweise verwenden, wo Graphiken „vektorisierbar" sind, d.h. wo Graphiken aus Linien oder Flächen bestehen, die durch GDI-Anweisungen schnell dargestellt werden können - etwa die typischen Word-Cliparts. Eine pixelweise aufgebaute Graphik (z.B. ein Foto), kann man zwar grundsätzlich als WMF-Datei speichern, braucht aber mehr Platz dafür als für das entsprechende Bitmap, das hat also keinen Sinn.

Ein weiterer Vorteil: Wie jede Vektorgraphik lassen sich WMF-Graphiken beliebig und ohne Qualitätsverluste vergrößern und verkleinern.

Was aber für den Programmierer das allerschönste ist: Wenn in einem Programm irgendetwas gezeichnet wird, dann kann man die entsprechenden Anweisungen an das GDI ganz einfach in einen Metafile „umleiten", muß sich weder über die einzelnen Funktionscodes oder über sonst viel den Kopf zerbrechen und hat die Graphik im WMF-Format vorliegen. Und in die andere Richtung geht es ähnlich einfach: Habe ich eine WMF-Datei vorliegen, so genügt im wesentlichen eine Anweisung an Windows, um mir den Inhalt dieser WMF-Datei „vorspielen" zu lassen.

Nebenbei: Die heute mehr und mehr verbreiteten sogenannten "GDI-Drucker", machen sich ein ähnliches Vorgehen zunutze: Zwischen PC und Drucker werden keine Pixeldaten übertragen, sondern GDI-Anweisungen, die erst im Drukker aufbereitet werden, was den PC entlastet und die zu übertragende Datenmenge drastisch verringert.

Der Vollständigkeit halber wollen wir hier noch die Struktur eines WMF-Files beschreiben: Das Metafile besteht aus zwei Teilen: einem Header und den Records mit den einzelnen GDI-Funktionen.

Der Header hat die Form (wieder in Pascal-Syntax):

```
TMetaHeader=record
    mtType: Word;
    mtHeaderSize: Word;
    mtVersion: Word
    mtSize: Longint;
    mtNoObjects: Word;
    mtMaxRecord: Longint;
    mtNoParameters: Word;
end;
```

Dazu einige Erläuterungen:

`mtType:`	0 ... Metafile ist im Speicher abgelegt 1 ... Metafile ist in einer Datei abgelegt
`mtHeaderSize:`	Die Größe des Headers in Word (9 Words)
`mtVersion:`	Windows-Versionnummer, $300 für Windows 3.0 und höher
`mtSize:`	Größe der Datei in Word
`mtNoObject:`	Maximale Anzahl von Objekten, die gleichzeitig im Metafile existieren können.
`mtMaxRecord:`	Größe des längsten Records im Metafile (in Word).
`MtNoParameters`	wird nicht verwendet.

Diese Erläuterungen sind hier nur der Vollständigkeit halber angeführt, wenn man Windows die Arbeit des Aufzeichnens und Wiedergebens des Metafiles überläßt, braucht man sich um diese Details nicht kümmern.

Wichtiger ist noch folgende Information: Neben den oben beschriebenen Standardmetafiles gibt es noch positionierbare Metafiles, die einen zusätzlichen 22-Byte-Header haben müssen:

```
TMFHeader=record
    Key: Longint;
    hMF: Thandle;
    bbox: Trect;
    inch: Word;
    reserved: Longint;
    checksum: Word
end;
```

Da man diesen Record u.U. selber mit Daten anfüllen muß, folgende Information:

`Key`	Ein Schlüssel, muß den Wert $9AC6CDD7 haben.
`hMF`	nicht verwendet, muß 0 sein;
`bbox`	Koordinaten des kleinsten umgebenden Rechtecks, Maßeinheit ist die im Feld inch gegebene Metafile-Einheit.
`inch`	Anzahl der Metafile-Einheiten pro Zoll. Üblicherweise verwendet man 576 oder 1000 - man muß ein bißchen aufpassen, um keine Überläufe zu provozieren!
`reserved`	nicht verwendet, 0.
`checksum`	Die Prüfsumme wird aus den ersten 10 Word-Werten des Headers unter Verwendung des Operators XOR gebildet.

Nach diesem Header folgt der Standard-Metafile, wie oben beschrieben.

Einige weitere Hinweise für die Verwendung von Metafiles: Eine der ersten Anweisungen sollte einen Abbildungsmodus setzen, zu beachten wäre, daß viele Anwendungen ausschließlich den Modus `mm_Anisotropic` akzeptieren! Speziell gilt dies für OLE-Anwendungen. Rufen Sie auch `SetWindowOrg` und `SetWindowExt` auf.

Verwenden Sie `SetViewportExt` und `SetViewportOrg` mit Vorsicht - der Anwender kann die Abmessungen nachträglich nicht mehr verändern, wenn diese Funktionen im Metafile enthalten sind.

Verwenden Sie keine Funktionen, die Daten abrufen, achten Sie insbesondere darauf, daß Sie nicht zuerst Daten abrufen und dann im nächsten Aufruf verwenden. Z. B. führt

```
TextXY:=GetTextExtent(MyDC,'Irgendein Text',14);
{Berechnungen mit TextXY, die X und Y ergeben}
TextOut(MyDC,X,Y,'Irgendein Text',14);
```

zu falschen Ergebnissen, wenn diese Anweisungen als Metafile aufgezeichnet werden!

Verwenden Sie auch keine geräteabhängigen Funktionen, z.B. Region-Funktionen.

Wie einfach das Abspielen eines Metafiles ist, wollen wir in einer kleinen Musteranwendung zeigen. Es soll ein Dialogfenster erstellt werden, das im wesentlichen dem üblichen Dateidialog entspricht, aber nur WMF-Dateien anzeigt - diese dafür mit Vorschau, also ähnlich dem Dialog, mit dem man in Word Cliparts suchen geht, oder ähnlich dem CorelMOSAIC-Rollup.

In diesem Beispiel geht es nicht darum, einen möglichst ausgeklügelten Dateidialog zu entwickeln - daher haben wir uns in dieser Hinsicht auf das Allernotwendigste beschränkt. In unserem Zusammenhang von Bedeutung ist die Prozedur `TPrevWindow.SetMetaFile`, die zeigt, wie man auf einfachste Weise eine positionierbare Metadatei einlesen kann, sie von dem überflüssigen Header befreit und sich ein Handle auf diese Metadatei holt.

Anschließend wird dieses Handle in der Prozedur `TPrevWindow.Paint` verwendet, um die Metadatei abzuspielen (`PlayMetaFile`) - im wesentlichen ein Befehl.

Neben dem letztlich geringen Programmieraufwand erkennt man, wenn man dieses Programmfragment testet, wie schnell die Metadateien abgespielt werden können. Außerdem sieht man auch schön, wie sich die verschieden großen Bilder alle problemlos in dasselbe Rechteck quetschen lassen.

Wenn man aus eigenen Graphiken Metafiles erstellen will, ist die Vorgangsweise ähnlich einfach, wie schon weiter oben erwähnt, muß man nur den Header mit den geeigneten Daten anfüllen und vor die Standardmetadatei hängen. □