Reading and Writing Metadata

Article • 06/30/2021

Some image files contain metadata that you can read to determine features of the image. For example, a digital photograph might contain metadata that you can read to determine the make and model of the camera used to capture the image. With Windows GDI+, you can read existing metadata, and you can also write new metadata to image files.

GDI+ provides a uniform way of storing and retrieving metadata from image files in various formats. In GDI+, a piece of metadata is called a *property item*. You can store and retrieve metadata by calling the **SetPropertyItem** and **GetPropertyItem** methods of the **Image** class, and you don't have to be concerned about the details of how a particular file format stores that metadata.

GDI+ currently supports metadata for the TIFF, JPEG, Exif, and PNG file formats. The Exif format, which specifies how to store images captured by digital still cameras, is built on top of the TIFF and JPEG formats. Exif uses the TIFF format for uncompressed pixel data and the JPEG format for compressed pixel data.

GDI+ defines a set of property tags that identify property items. Certain tags are general-purpose; that is, they are supported by all of the file formats mentioned in the preceding paragraph. Other tags are special-purpose and apply only to certain formats. If you try to save a property item to a file that does not support that property item, GDI+ ignores the request. More specifically, the **Image::SetPropertyItem** method returns PropertyNotSupported.

You can determine the property items that are stored in an image file by calling **Image::GetPropertyIdList**. If you try to retrieve a property item that is not in the file, GDI+ ignores the request. More specifically, the **Image::GetPropertyItem** method returns PropertyNotFound.

Reading Metadata from a File

The following console application calls the **GetPropertySize** method of an **Image** object to determine how many pieces of metadata are in the file FakePhoto.jpg.

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;
INT main()
{
   // Initialize <tla rid="tla_gdiplus"/>.
   GdiplusStartupInput gdiplusStartupInput;
   ULONG_PTR gdiplusToken;
   GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
   UINT    size = 0;
   UINT    count = 0;
   Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");
   bitmap->GetPropertySize(&size, &count);
   printf("There are %d pieces of metadata in the file.\n", count);
   printf("The total size of the metadata is %d bytes.\n", size);

   delete bitmap;
   GdiplusShutdown(gdiplusToken);
   return 0;
}
```

The preceding code, along with a particular file, FakePhoto.jpg, produced the following output:

```
There are 7 pieces of metadata in the file.
The total size of the metadata is 436 bytes.
```

GDI+ stores an individual piece of metadata in a **PropertyItem** object. You can call the **GetAllPropertyItems** method of the **Image** class to retrieve all the metadata from a file. The **GetAllPropertyItems** method returns an array of **PropertyItem** objects. Before you call **GetAllPropertyItems**, you must allocate a buffer large enough to receive that array. You can call the **GetPropertySize** method of the **Image** class to get the size (in bytes) of the required buffer.

A **PropertyItem** object has the following four public members:

	Description
id	A tag that identifies the metadata item. The values that can be assigned to **id** (PropertyTagImageTitle, PropertyTagEquipMake, PropertyTagExifExposureTime, and the like) are defined in Gdiplusimaging.h.

	Description
length	The length, in bytes, of the array of values pointed to by the **value** data member. Note that if the **type** data member is set to PropertyTagTypeASCII, then the length data member is the **length** of a null-terminated character string, including the NULL terminator.
type	The data type of the values in the array pointed to by the value data member. Constants (PropertyTagTypeByte, PropertyTagTypeASCII, and the like) that represent various data types are described in Image Property Tag Type Constants.
value	A pointer to an array of values.

The following console application reads and displays the seven pieces of metadata in the file FakePhoto.jpg. The main function relies on the helper function PropertyTypeFromWORD, which is shown following the main function.

```
#include <windows.h>
#include <gdiplus.h>
#include <strsafe.h>
using namespace Gdiplus;

INT main()
{
   // Initialize GDI+
   GdiplusStartupInput gdiplusStartupInput;
   ULONG_PTR gdiplusToken;
   GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

   UINT  size = 0;
   UINT  count = 0;

   #define MAX_PROPTYPE_SIZE 30
   WCHAR strPropertyType[MAX_PROPTYPE_SIZE] = L"";

   Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");

   bitmap->GetPropertySize(&size, &count);
   printf("There are %d pieces of metadata in the file.\n\n", count);

   // GetAllPropertyItems returns an array of PropertyItem objects.
   // Allocate a buffer large enough to receive that array.
   PropertyItem* pPropBuffer =(PropertyItem*)malloc(size);

   // Get the array of PropertyItem objects.
```

```
    bitmap->GetAllPropertyItems(size, count, pPropBuffer);

    // For each PropertyItem in the array, display the id, type, and length.
    for(UINT j = 0; j < count; ++j)
    {
       // Convert the property type from a WORD to a string.
       PropertyTypeFromWORD(
          pPropBuffer[j].type, strPropertyType, MAX_PROPTYPE_SIZE);

       printf("Property Item %d\n", j);
       printf("  id: 0x%x\n", pPropBuffer[j].id);
       wprintf(L"  type: %s\n", strPropertyType);
       printf("  length: %d bytes\n\n", pPropBuffer[j].length);
    }

    free(pPropBuffer);
    delete bitmap;
    GdiplusShutdown(gdiplusToken);
    return 0;
} // main

// Helper function
HRESULT PropertyTypeFromWORD(WORD index, WCHAR* string, UINT maxChars)
{
    HRESULT hr = E_FAIL;

    WCHAR* propertyTypes[] = {
       L"Nothing",                   // 0
       L"PropertyTagTypeByte",       // 1
       L"PropertyTagTypeASCII",      // 2
       L"PropertyTagTypeShort",      // 3
       L"PropertyTagTypeLong",       // 4
       L"PropertyTagTypeRational",   // 5
       L"Nothing",                   // 6
       L"PropertyTagTypeUndefined",  // 7
       L"Nothing",                   // 8
       L"PropertyTagTypeSLONG",      // 9
       L"PropertyTagTypeSRational"}; // 10

    hr = StringCchCopyW(string, maxChars, propertyTypes[index]);
    return hr;
}
```

The preceding console application produces the following output:

```
Property Item 0
  id: 0x320
  type: PropertyTagTypeASCII
```

```
    length: 16 bytes
  Property Item 1
    id: 0x10f
    type: PropertyTagTypeASCII
    length: 17 bytes
  Property Item 2
    id: 0x110
    type: PropertyTagTypeASCII
    length: 7 bytes
  Property Item 3
    id: 0x9003
    type: PropertyTagTypeASCII
    length: 20 bytes
  Property Item 4
    id: 0x829a
    type: PropertyTagTypeRational
    length: 8 bytes
  Property Item 5
    id: 0x5090
    type: PropertyTagTypeShort
    length: 128 bytes
  Property Item 6
    id: 0x5091
    type: PropertyTagTypeShort
    length: 128 bytes
```

The preceding output shows a hexadecimal ID number for each property item. You can look up those ID numbers in Image Property Tag Constants and find out that they represent the following property tags.

Hexadecimal value	Property tag
0x0320 0x010f	PropertyTagImageTitle PropertyTagEquipMake
0x0110	PropertyTagEquipModel
0x9003	PropertyTagExifDTOriginal
0x829a	PropertyTagExifExposureTime
0x5090	PropertyTagLuminanceTable
0x5091	PropertyTagChrominanceTable

The second (index 1) property item in the list has **id** PropertyTagEquipMake and **type** PropertyTagTypeASCII. The following example, which is a continuation of the previous console application, displays the value of that property item:

```
printf("The equipment make is %s.\n", pPropBuffer[1].value);
```

The preceding line of code produces the following output:

```

The equipment make is Northwind Traders.
```

The fifth (index 4) property item in the list has **id** PropertyTagExifExposureTime and **type** PropertyTagTypeRational. That data type (PropertyTagTypeRational) is a pair of **LONG**s. The following example, which is a continuation of the previous console application, displays those two **LONG** values as a fraction. That fraction, 1/125, is the exposure time measured in seconds.

```

long* ptrLong = (long*)(pPropBuffer[4].value);
printf("The exposure time is %d/%d.\n", ptrLong[0], ptrLong[1]);
```

The preceding code produces the following output:

```

The exposure time is 1/125.
```

Writing Metadata to a File

To write an item of metadata to an **Image** object, initialize a **PropertyItem** object and then pass the address of that **PropertyItem** object to the **SetPropertyItem** method of the **Image** object.

The following console application writes one item (the image title) of metadata to an **Image** object and then saves the image in the disk file FakePhoto2.jpg. The main function relies on the helper function GetEncoderClsid, which is shown in the topic Retrieving the Class Identifier for an Encoder.

```

```

```
#include <windows.h>
#include <gdiplus.h>
#include <stdio.h>
using namespace Gdiplus;
INT main()
{
   // Initialize <tla rid="tla_gdiplus"/>.
   GdiplusStartupInput gdiplusStartupInput;
   ULONG_PTR gdiplusToken;
   GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
   Status stat;
   CLSID  clsid;
   char   propertyValue[] = "Fake Photograph";
   Bitmap* bitmap = new Bitmap(L"FakePhoto.jpg");
   PropertyItem* propertyItem = new PropertyItem;
   // Get the CLSID of the JPEG encoder.
   GetEncoderClsid(L"image/jpeg", &clsid);
   propertyItem->id = PropertyTagImageTitle;
   propertyItem->length = 16;  // string length including NULL terminator
   propertyItem->type = PropertyTagTypeASCII;
   propertyItem->value = propertyValue;
   bitmap->SetPropertyItem(propertyItem);
   stat = bitmap->Save(L"FakePhoto2.jpg", &clsid, NULL);
   if(stat == Ok)
      printf("FakePhoto2.jpg saved successfully.\n");

   delete propertyItem;
   delete bitmap;
   GdiplusShutdown(gdiplusToken);
   return 0;
}
```

Feedback

Was this page helpful? Yes No