Programmatically editing the metadata of an audio file

devblogs.microsoft.com/oldnewthing/20130909-00

September 9, 2013

Raymond Chen

Today’s Little Program edits the metadata of an audio file, ostensibly to correct a spelling error, but really just to show how it’s done.

Today’s smart pointer class library is… (rolls dice)… CComPtr!

We open with two helper functions which encapsulate the patterns

- Get property from property store
  1. Call `IPropertyStore::GetValue`
  2. Convert `PROPVARIANT` into desired final type
  3. Destroy the `PROPVARIANT`
- Set property in property store
  1. Create a `PROPVARIANT`
  2. Call `IPropertyStore::SetValue`
  3. Destroy the `PROPVARIANT`

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <shlobj.h>
#include <propkey.h>
#include <propvarutil.h>
#include <atlbase.h>
#include <atlalloc.h>
template<typename TLambda>
HRESULT GetPropertyAsLambda(IPropertyStore *pps, REFPROPERTYKEY key,
                             TLambda lambda)
{
  PROPVARIANT pvar;
  HRESULT hr = pps->GetValue(key, &pvar);
  if (SUCCEEDED(hr)) {
    hr = lambda(pvar);
    PropVariantClear(&pvar);
  }
  return hr;
}
template<typename TLambda>
HRESULT SetPropertyAsLambda(IPropertyStore *pps, REFPROPERTYKEY key,
                            TLambda lambda)
{
  PROPVARIANT pvar;
  HRESULT hr = lambda(&pvar);
  if (SUCCEEDED(hr)) {
    hr = pps->SetValue(key, pvar);
    PropVariantClear(&pvar);
  }
  return hr;
}
```

Both functions use a lambda to do the type-specific work.

Here are some functions that will use the helpers:

```
HRESULT GetPropertyAsString(
    IPropertyStore *pps, REFPROPERTYKEY key, PWSTR *ppszValue)
{
  *ppszValue = nullptr;
  return GetPropertyAsLambda(pps, key, [=](REFPROPVARIANT pvar) {
    return PropVariantToStringAlloc(pvar, ppszValue);
  });
}
HRESULT SetPropertyAsString(
    IPropertyStore *pps, REFPROPERTYKEY key, PCWSTR pszValue)
{
  return SetPropertyAsLambda(pps, key, [=](PROPVARIANT *ppvar) {
    return InitPropVariantFromString(pszValue, ppvar);
  });
}
HRESULT GetPropertyAsStringVector(
    IPropertyStore *pps, REFPROPERTYKEY key,
    PWSTR **pprgsz, ULONG *pcElem)
{
  *pprgsz = nullptr;
  *pcElem = 0;
  return GetPropertyAsLambda(pps, key, [=](REFPROPVARIANT pvar) {
    return PropVariantToStringVectorAlloc(pvar, pprgsz, pcElem);
  });
}
HRESULT SetPropertyAsStringVector(
    IPropertyStore *pps, REFPROPERTYKEY key,
    PCWSTR *prgsz, ULONG cElems)
{
  return SetPropertyAsLambda(pps, key, [=](PROPVARIANT *ppvar) {
    return InitPropVariantFromStringVector(prgsz, cElems, ppvar);
  });
}
```

The `PropVariantToStringVectorAlloc` function returns an array of pointers to memory allocated via `CoTaskMemAlloc`, and the array itself was also allocated by the same function. Here's a helper function to free the memory and the array:

```
template<typename T>
void CoTaskMemFreeArray(T **prgElem, ULONG cElem)
{
    for (ULONG i = 0; i < cElem; i++) {
        CoTaskMemFree(prgElem[i]);
    }
    CoTaskMemFree(prgElem);
}
```

Okay, we're ready to write our main program. Remember, Little Programs do little to no error checking. In a real program, you would check that your function calls succeeded.

```
int __cdecl wmain(int argc, wchar_t **argv)
{
  CCoInitialize init;
  CComPtr<IPropertyStore> spps;
  SHGetPropertyStoreFromParsingName(argv[1], nullptr,
    GPS_READWRITE, IID_PPV_ARGS(&spps));
  // Get the existing composers
  PWSTR *rgpszComposers;
  ULONG cComposers;
  GetPropertyAsStringVector(spps, PKEY_Music_Composer,
    &rgpszComposers, &cComposers);
  // Look for "Dvorak, Antonin" and add diacritics
  for (ULONG ulPos = 0; ulPos < cComposers; ulPos++) {
    if (wcscmp(rgpszComposers[ulPos], L"Dvorak, Antonin") == 0) {
      // Swap in the new name
      PWSTR pszOld = rgpszComposers[ulPos];
      rgpszComposers[ulPos] = L"Dvo\x0159\x00E1k, Anton\x00EDn";
      // Write out the new list of composers
      SetPropertyAsStringVector(spps, PKEY_Music_Composer, (PCWSTR *)rgpszComposers,
cComposers);
      // Swap it back so we can free it
      rgpszComposers[ulPos] = pszOld;
      // Add a little graffiti just because
      SetPropertyAsString(spps, PKEY_Comment, L"Kilroy was here");
      spps->Commit();
      break;
    }
  }
  CoTaskMemFreeArray(rgpszComposers, cComposers);
  return 0;
}
```

Okay, what just happened here?

First, we took the file whose name was passed on the command line (fully-qualified path, please) and obtained its property store.

Next, we queried the property store for the `System.Music.Composer` property. This property is typed as a multiple-valued string, so we read and write the value in the form of a string vector. You could also read and write it as a single string: The `PropVariantToStringAlloc` function represents string arrays by joining the strings together, separating them with `"; "` (semicolon and space). However, we access it as an array because that makes it easier to insert and remove individual entries.

Once we get the list of composers, we look for one that says `"Dvorak, Antonin"`. If we find it, then we change that entry to `"Dvořák, Antonín"` and write out the new vector.

And then just to show that I know how to write out a string property too, I'll put some graffiti in the Comment field.

Commit the changes and break the loop now that we found what we’re looking for. (This assumes that the song was not a collaboration between Antonín Dvořák and himself!)

So there you have it, a little program that modifies metadata. Obviously, this program is not particularly useful by itself, but it illustrates what you need to do to do something more useful in general.

Raymond Chen

Follow