Excursions in composition: Adding rewind support to a sequential stream

devblogs.microsoft.com/oldnewthing/20070323-00

March 23, 2007

Raymond Chen

Here's a problem "inspired by actual events":

> I have a sequential stream that is the response to a request I sent to a web site. The format of the stream is rather messy; it comes with a variable-length header that describes what type of data is being returned. I want to read that header and then hand the stream to an appropriate handler. But the handlers expect to be given the stream in its entirety, including the bytes that I have already read. Since this is a sequential stream, I can't change the seek position. How can I "unread" the data and give the handlers what they want?
>
> Right now, I'm just closing the stream I get and issuing a second request. I give that second stream to the handler that I determined by parsing the first stream. Of course, this makes the rather unsafe assumption that the server will send the same data stream back the second time, and I'm issuing twice as many requests as I really need.
>
> I tried reading the entire stream into memory and creating a new stream on that, but the data stream can be very big, and I feel bad allocating all that memory just so I can "unread" a few dozen bytes.

All the customer wants here is to be able to "unread" some bytes from a stream. This is a perfect job for our composite sequential stream. The first stream consists of the bytes we want to push back, and the second stream is the rest of the stream. Here's an explanation in pictures. The pink stream is the original stream returned from the web site, and the green stream remembers the bytes we want to push back.

Initial state:

ABCDEFGHIJK…

After reading a few bytes:

ABCD EFGHIJK…

After reading a few more bytes:

ABCDEFG HIJK…

Finally, we have determined the handler, but the handler expects to start reading from the "A", so we will take these two streams and concatenate them so they look like one stream again:

ABCDEFG HIJK…

Okay, let's do it. All we have to worry about is filling the green stream with the bytes that we read out of the pink stream; we're going to use `CConcatStream` to do the composition at the end.

```
class CRewindStream : public CROSequentialStreamBase
{
public:
 CRewindStream(ISequentialStream *pstm);
 ISequentialStream *Rewind();
 // *** ISequentialStream ***
 STDMETHODIMP Read(void *pv, ULONG cb, ULONG *pcbRead);
protected:
 ~CRewindStream();
 bool m_fRewound;
 IStream *m_pstm1;
 ISequentialStream *m_pstm2;
};
CRewindStream::CRewindStream(ISequentialStream *pstm)
 : m_fRewound(false), m_pstm2(pstm)
{
 CreateStreamOnHGlobal(NULL, TRUE, &m_pstm1);
 m_pstm2->AddRef();
}
CRewindStream::~CRewindStream()
{
 if (m_pstm1) m_pstm1->Release();
 m_pstm2->Release();
}
HRESULT CRewindStream::Read(void *pv, ULONG cb, ULONG *pcbRead)
{
 ULONG cbRead = 0;
 HRESULT hr;
 if (m_fRewound) {
  hr = E_FAIL;
 } else if (!m_pstm1) {
  hr = E_OUTOFMEMORY;
 } else {
  hr = m_pstm2->Read(pv, cb, &cbRead);
  if (SUCCEEDED(hr)) {
   hr = m_pstm1->Write(pv, cbRead, NULL);
  }
 }
 if (pcbRead) *pcbRead = cbRead;
 return hr;
}
ISequentialStream *CRewindStream::Rewind()
{
 if (!m_pstm1 || m_fRewound) return NULL;
 m_fRewound = true;
 const LARGE_INTEGER li0 = { 0, 0 };
 m_pstm1->Seek(li0, STREAM_SEEK_SET, NULL);
 return new CConcatStream(m_pstm1, m_pstm2);
}
```

Our `RewindStream` takes a sequential stream and creates a memory stream via `CreateStreamOnHGlobal` to remember the parts that we read so we can stick them back when it's time to rewind.

When reading from the stream, we read from the sequential stream and append the result to the memory stream before returning it. In this manner, each byte read from the head of the pink stream gets appended to the end of the green stream.

When it's time to rewind, we seek the memory stream back to the beginning and create a concatenated stream out of the memory stream and the unread portion of the sequential stream.

Here's a simple program that illustrates our new rewindable sequential stream:

```
int __cdecl _tmain(int argc, TCHAR **argv)
{
 CoInitialize(NULL);
 IStream *pstmFile;
 if (SUCCEEDED(SHCreateStreamOnFile(argv[1], STGM_READ,
                                    &pstmFile))) {
  CRewindStream *pstmRewind = new CRewindStream(pstmFile);
  if (pstmRewind) {
   char ch;
   ULONG cb;
   while (SUCCEEDED(pstmRewind->Read(&ch, 1, &cb)) && cb) {
    printf("Header: '%c'\n", ch);
    if (ch == ' ') {
     ISequentialStream *pstmRewound = pstmRewind->Rewind();
     if (pstmRewound) {
      PrintStream(pstmRewound);
      pstmRewound->Release();
     }
     break;
    }
   }
   pstmRewind->Release();
  }
  pstmFile->Release();
 }
 CoUninitialize();
 return 0;
}
```

For illustration purposes, let's assume that the header ends when we find a space. We create a stream on the file whose name is passed on the command line and put it inside a `CRewindStream`. We then read from that stream a byte at a time until we find that space. (To prove that we're really rewinding, we also print each byte from the header as we encounter it.) Once we find the space, we ask the `CRewindStream` to create a new stream

that represents the original stream rewound back to the beginning. We pass that stream to the `PrintStream` helper function from last time, to prove that the resulting stream really is the entire file contents starting from the beginning.

Notice that we consumed only as much memory as necessary to remember the parts of the stream that needed to be replayed. Even if the file were a megabyte in size, we only need to remember the bytes that we read up until we decided that we were finished reading the header.

Now, this isn't the most beautiful implementation of a rewindable stream, but it was convenient for expository purposes. I leave you to make as many prettifications as meet your aesthetic requirements.

Exercise: Discuss what would be needed in order to support rewinding more than once and the performance consequences thereof.

Raymond Chen

Follow