# Structured Storage and Compound Files

scorpiosoftware.net/2024/11/09/structured-storage-and-compound-files

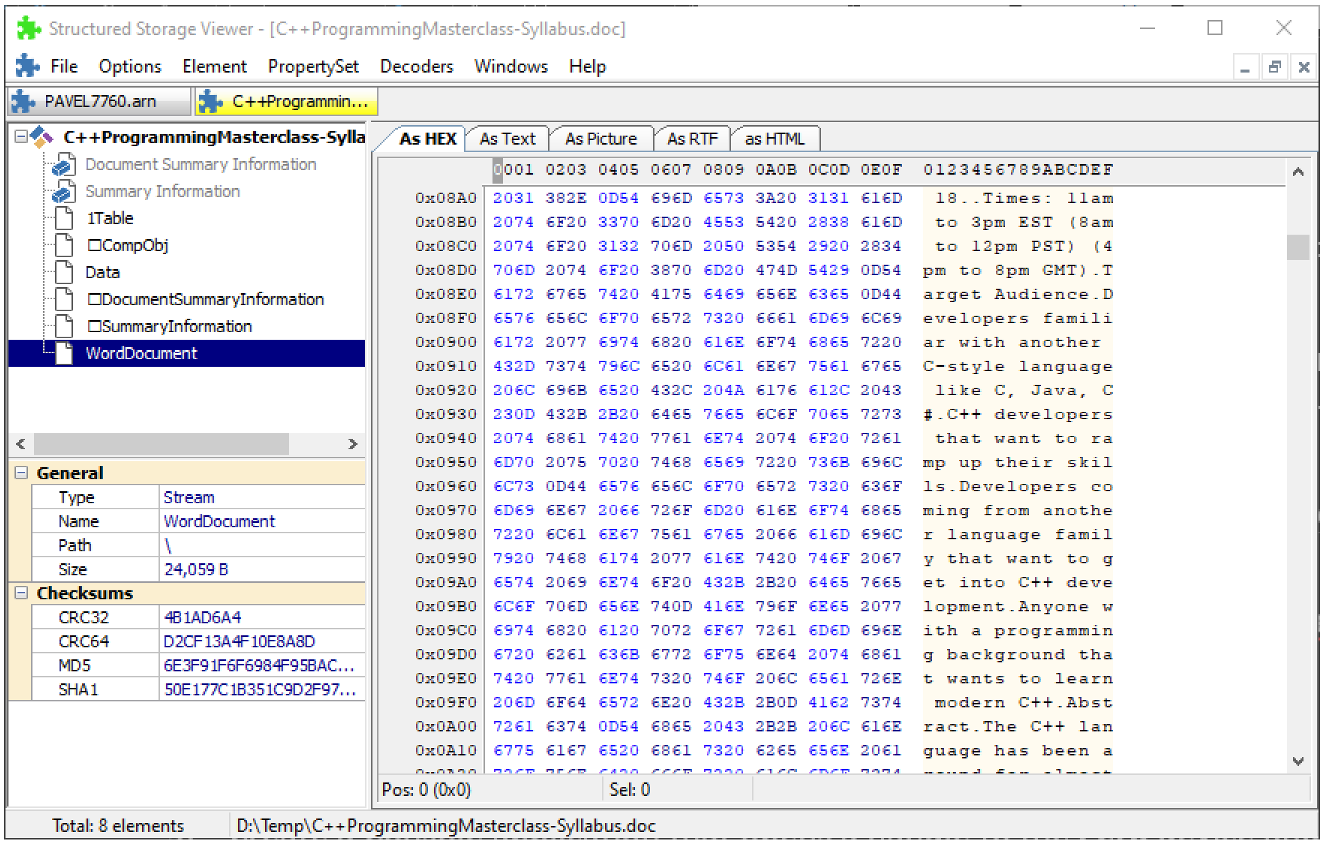
View all posts by Pavel Yosifovich November 9, 2024

Structured Storage is a Windows technology that abstracts the notions of files and directories behind COM interfaces – mainly `IStorage` and `IStream`. Its primary intent is to provide a way to have a file system hierarchy within a single physical file.

Structured Storage has been around for many years, where its most famous usage was in Microsoft Office files (*.doc, *.ppt, *.xls, etc.) – before Office moved to the extended file formats (*.docx, *.pptx, etc.). Of course, the old formats are still very much supported.

The Structured Storage interfaces (`IStorage` representing a directory, and `IStream` representing a file) are just that – interfaces. To actually use them, some implementation must be available. Windows provided an implementation of Structured Storage called **Compound Files**. These terms are sometime used interchangeably, but the distinction is important: Compound Files is just one implementation of Structured Storage – there could be others. Compound Files does not implement everything that could be implemented based on the defined Structured Storage interfaces, but it implements a lot, definitely enough to make it useful.

You can download an old tool (but still works well) called **SSView**, which can be used to graphically view the contents of physical files that were created by using the Compound File implementation. Here is a screenshot of **SSView**, looking at some DOC file:


Structured Storage Viewer - [C++ProgrammingMasterclass-Syllabus.doc]
File Options Element PropertySet Decoders Windows Help
PAVEL7760.arn
C++Programmin...
C++ProgrammingMasterclass-Sylla
Document Summary Information
Summary Information
1Table
□CompObj
Data
□DocumentSummaryInformation
□SummaryInformation
WordDocument
As HEX
As Text
As Picture
As RTF
as HTML

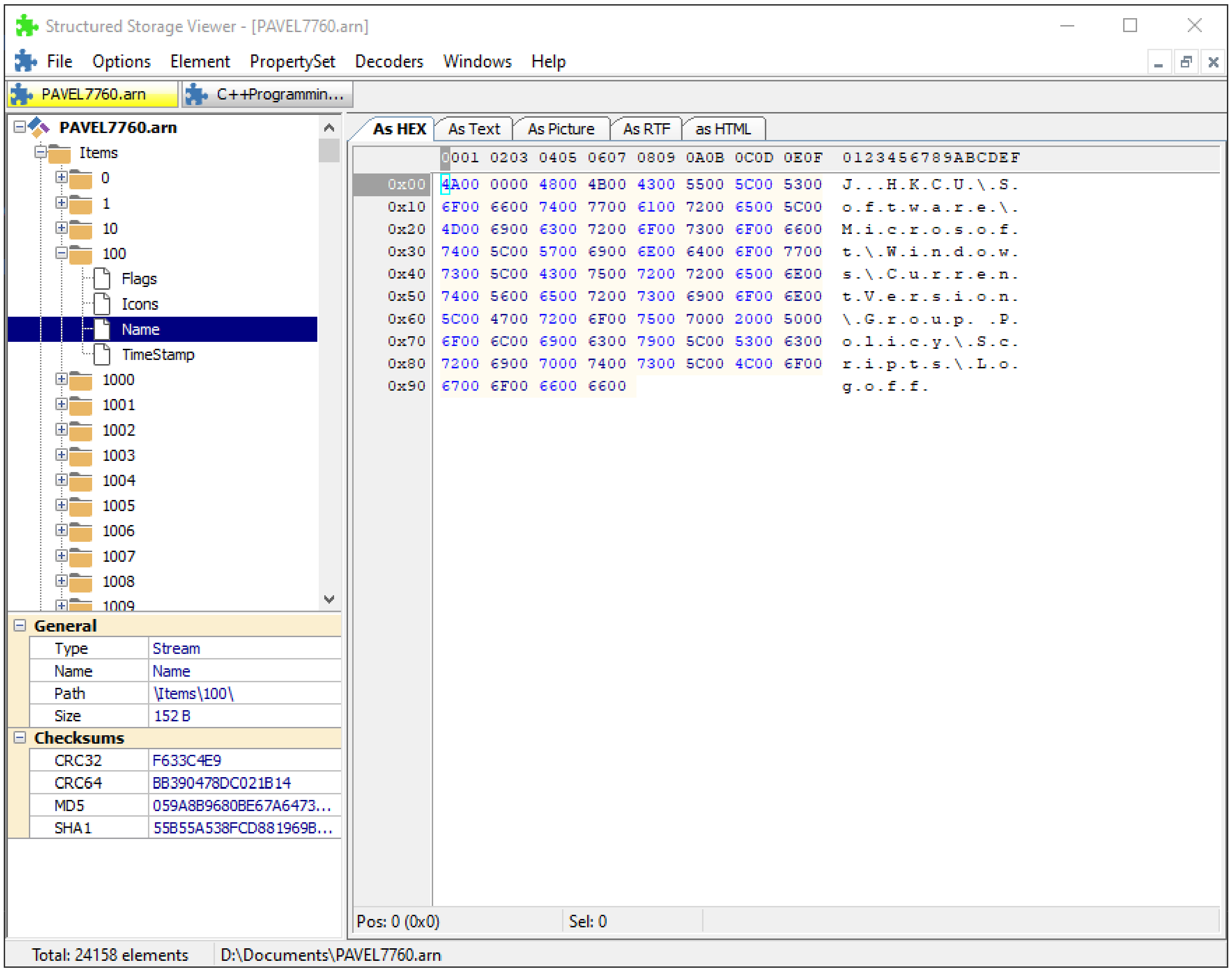
| General | |
|---|---|
| Type | Stream |
| Name | WordDocument |
| Path | \ |
| Size | 24,059 B |
| **Checksums** | |
| CRC32 | 4B1AD6A4 |
| CRC64 | D2CF13A4F10E8A8D |
| MD5 | 6E3F91F6F6984F95BAC... |
| SHA1 | 50E177C1B351C9D2F97... |

```
        0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0x08A0  2031 382E 0D54 696D 6573 3A20 3131 616D   18..Times: 11am
0x08B0  2074 6F20 3370 6D20 4553 5420 2838 616D   to 3pm EST (8am
0x08C0  2074 6F20 3132 706D 2050 5354 2920 2834   to 12pm PST) (4
0x08D0  706D 2074 6F20 3870 6D20 474D 5429 0D54  pm to 8pm GMT).T
0x08E0  6172 6765 7420 4175 6469 656E 6365 0D44  arget Audience.D
0x08F0  6576 656C 6F70 6572 7320 6661 6D69 6C69  evelopers famili
0x0900  6172 2077 6974 6820 616E 6F74 6865 7220  ar with another
0x0910  432D 7374 796C 6520 6C61 6E67 7561 6765  C-style language
0x0920  206C 696B 6520 432C 204A 6176 612C 2043   like C, Java, C
0x0930  230D 432B 2B20 6465 7665 6C6F 7065 7273  #.C++ developers
0x0940  2074 6861 7420 7761 6E74 2074 6F20 7261   that want to ra
0x0950  6D70 2075 7020 7468 6569 7220 736B 696C  mp up their skil
0x0960  6C73 0D44 6576 656C 6F70 6572 7320 636F  ls.Developers co
0x0970  6D69 6E67 2066 726F 6D20 616E 6F74 6865  ming from anothe
0x0980  7220 6C61 6E67 7561 6765 2066 616D 696C  r language famil
0x0990  7920 7468 6174 2077 616E 7420 746F 2067  y that want to g
0x09A0  6574 2069 6E74 6F20 432B 2B20 6465 7665  et into C++ deve
0x09B0  6C6F 706D 656E 740D 416E 796F 6E65 2077  lopment.Anyone w
0x09C0  6974 6820 6120 7072 6F67 7261 6D6D 696E  ith a programmin
0x09D0  6720 6261 636B 6772 6F75 6E64 2074 6861  g background tha
0x09E0  7420 7761 6E74 7320 746F 206C 6561 726E  t wants to learn
0x09F0  206D 6F64 6572 6E20 432B 2B0D 4162 7374   modern C++.Abst
0x0A00  7261 6374 0D54 6865 2043 2B2B 206C 616E  ract.The C++ lan
0x0A10  6775 6167 6520 6861 7320 6265 656E 2061  guage has been a
```

Pos: 0 (0x0)    Sel: 0

Total: 8 elements    D:\Temp\C++ProgrammingMasterclass-Syllabus.doc

Here is a more interesting example – information persisted using Sysinternals *Autoruns* tool (discussed later):



```
PAVEL7760.arn
  Items
    0
    1
    10
    100
      Flags
      Icons
      Name
      TimeStamp
    1000
    1001
    1002
    1003
    1004
    1005
    1006
    1007
    1008
    1009
```

**General**

| | |
|---|---|
| Type | Stream |
| Name | Name |
| Path | \Items\100\ |
| Size | 152 B |

**Checksums**

| | |
|---|---|
| CRC32 | F633C4E9 |
| CRC64 | BB390478DC021B14 |
| MD5 | 059A8B9680BE67A6473... |
| SHA1 | 55B55A538FCD881969B... |

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0x00  4A00 0000 4800 4B00 4300 5500 5C00 5300  J...H.K.C.U.\.S.
0x10  6F00 6600 7400 7700 6100 7200 6500 5C00  o.f.t.w.a.r.e.\.
0x20  4D00 6900 6300 7200 6F00 7300 6F00 6600  M.i.c.r.o.s.o.f.
0x30  7400 5C00 5700 6900 6E00 6400 6F00 7700  t.\.W.i.n.d.o.w.
0x40  7300 5C00 4300 7500 7200 7200 6500 6E00  s.\.C.u.r.r.e.n.
0x50  7400 5600 6500 7200 7300 6900 6F00 6E00  t.V.e.r.s.i.o.n.
0x60  5C00 4700 7200 6F00 7500 7000 2000 5000  \.G.r.o.u.p.  .P.
0x70  6F00 6C00 6900 6300 7900 5C00 5300 6300  o.l.i.c.y.\.S.c.
0x80  7200 6900 7000 7400 7300 5C00 4C00 6F00  r.i.p.t.s.\.L.o.
0x90  6700 6F00 6600 6600                      g.o.f.f.
```

Pos: 0 (0x0)    Sel: 0

Total: 24158 elements    D:\Documents\PAVEL7760.arn

A more interesting hierarchy is clearly visible – although it's all in a single file!

## The Main Interfaces

The `IStorage` interface represents a "directory", that can contain other directories and "files", represented as `IStream` interface implementations. To get started a physical file can be created with `StgCreateStorageEx` or an existing file opened with `StgOpenStorageEx`. Both return an `IStorage` pointer on success. From there, methods on `IStorage` can be called to create or open other directories (storages) and/or files (streams).

The most useful methods on `IStorage` are `CreateStorage`, `CreateStream`, `OpenStorage` and `OpenStream`. Enumeration of storages/streams is possible with `EnumElements`. Here is an example for opening a compound file for read access (filename is from a command line argument):

```
CComPtr<IStorage> spStg;
auto hr = ::StgOpenStorageEx(argv[1], STGM_READ | STGM_SHARE_EXCLUSIVE,
STGFMT_STORAGE, 0, nullptr, nullptr, __uuidof(IStorage),
reinterpret_cast<void**>(&spStg));
if (FAILED(hr)) {
printf("Failed to open file (0x%X)\n", hr);
return hr;
}
```

The following demonstrates enumerating the hierarchy of a given storage, recursively:

```
void EnumItems(IStorage* stg, int indent = 0) {
CComPtr<IEnumSTATSTG> spEnum;
stg->EnumElements(0, nullptr, 0, &spEnum);
if (spEnum == nullptr)
return;
STATSTG stat;
while (S_OK == spEnum->Next(1, &stat, nullptr)) {
if (indent)
printf(std::string(indent, ' ').c_str());
printf("%ws", stat.pwcsName);
if (stat.type == STGTY_STORAGE) {
printf(" [DIR]\n");
CComPtr<IStorage> spSubStg;
stg->OpenStorage(stat.pwcsName, nullptr,
STGM_READ | STGM_SHARE_EXCLUSIVE, 0, 0, &spSubStg);
if (spSubStg)
EnumItems(spSubStg, indent + 1);
}
else
printf(" (%u bytes)\n", stat.cbSize.LowPart);
::CoTaskMemFree(stat.pwcsName);
}
}
```

Each item has a name, but streams ("files") can have data. The `cbSize` member of `STATSTG` returns that size. A stream is just an abstraction over a bunch of bytes. To actually read/write from/to a stream, it needs to be opened with `IStorage::OpenStream` before accessing the data with `IStream::Read`, `IStream::Write` and similar methods.

## More on Streams

The `IStream` interface is used in various places within the Windows API, not just part of Structured Storage. It represents an abstraction over a buffer, that in theory could be anywhere – that's the nice thing about an abstraction. Given an `IStream` pointer, you can read, wrote, seek, copy to another stream, clone, and even commit/revert a transaction, if supported by the implementation. Compound Files, by the way, doesn't support transactions on streams.

Outside of Structured Storage, streams can be obtained in several ways.

The `CreateStreamOnHGlobal` API creates a memory buffer over an optional `HGLOBAL` (can be `NULL` to allocate a new one) and returns an `IStream` pointer to that memory buffer. This is useful when dealing with the clipboard for example, as it requires an `HGLOBAL`, which may not be convenient to work with. By getting an `IStream` pointer, the code can work with it (maybe reading it from another stream, or manually populating it with data), and then calling `GetHGlobalFromStream` to get the underlying `HGLOBAL` before passing it to the clipboard (e.g. `SetClipboardData`).

A stream can also be obtained for a file directly by calling `SHCreateStreamOnFile`, providing a convenient access to file data, abstracted as `IStream`.

Another case where `IStream` makes an appearance is in ActiveX controls persistence.

Yet another example of using `IStream` is as a way to "package" information for COM object's state that would allow creating a proxy to that object from a different apartment by calling `CoMarshalInterThreadInterfaceInStream` (probably the longest-name COM API), that captures the state required (as an `IStream`) to pass to another apartment, where the corresponding `CoGetInterfaceAndReleaseStream` can be called to generate a proxy to the original object, if needed.

## Case Study: Autoruns

Back in 2021 when I was working for the *Sysinternals* team, one of my tasks was to modernize Autoruns from a GUI perspective. I thought I would take this opportunity to do a significant rewrite, so that it would be easier to maintain the tool and improve it as needed. Fortunately, Mark Russinovich was onboard with that, although my time estimate for this project was way off 🙂 But I digress.

One of the features of Autoruns is the ability to save the information provided by the tool so it can be loaded later, possibly on a different machine. This is non-trivial, as some of the information is not easy to persist, such as icons. I don't recall if the old Autoruns persisted them, but I definitely wanted to do so.

The old Autoruns format was sequential in nature, storing structures of data linearly in the file. Any new properties that needed to be added would require offset changes, which forced changing the format "version" and make the correct decisions when reading a file in an various "old" formats.

I wanted to make persistence more flexible, so I decided to change the format completely to be a compound file. With this scheme, adding new properties would not cause any issues – a new stream may be added, but other streams are not disturbed. The code could just ignore properties (could be storages and/or streams) it wasn't aware of. This made the format extensible by definition, immune to any offset changes, and very easy to view using tools, like *SSView*. The above screenshot is from an Autoruns-persisted file.

Persisting icons, by the way, becomes pretty easy, because **ImageList** objects, used by Autoruns to hold collection of icons can be persisted to a stream with a single function call: `ImageList_Write`; very convenient!

## Conclusion

The Structured Storage idea is a powerful one, and the Compound File implementation provided by Windows is pretty good and flexible. One of the reasons Microsoft moved Office to a new format was the need to make files smaller, so the new extended formats are ZIP compressed. Their internal format changed as well, and is not using Compound Files for the most part. A Structured Storage file could be compressed, saving disk space, while still maintaining convenient access using storages and streams.