Structured storage, Monikers, Running Object Table

Structured storage

Structured storage **jest sposobem zorganizowanego zapisywania danych w pliku. Struktura danych ma postać drzewa i obiektów danych podobną do systemu plików i katalogów. Podejście takie pozwala na zapisanie wewnątrz jednego pliku informacji pochodzących od różnych obiektów. Do zapisu i organizacji danych opracowano dwa standardowe interfejsy `IStorage` i `IStream`. `IStorage` jest odpowiednikiem katalogów i pozwala na konstruowanie struktury pliku. `IStream` pozwala na dostęp do wybranych danych umieszczonych w dowolnym miejscu struktury pliku.**

Structured storage
IStorage

```
interface IStorage:IUnknown {
 HRESULT CreateStream([in] OLECHAR *pwcsName, [in] DWORD grfMode,
   [in] DWORD reserved1, [in] DWORD reserved2, [out] IStream **ppstm);
 HRESULT OpenStream([in] OLECHAR *pwcsName, [in] void *reserved1,
   [in] DWORD grfMode, [in] DWORD reserved2, [out] IStream **ppstm);
 HRESULT CreateStorage([in] OLECHAR *pwcsName, [in] DWORD grfMode,
   [in] DWORD reserved1, [in] DWORD reserved2, [out] IStorage **ppstg);
 HRESULT OpenStorage([in] OLECHAR *pwcsName, [in] IStorage *pstgPriority,
   [in] DWORD grfMode, [in] SNB snbExclude, [in] DWORD reserved, [out] IStorage **ppstg);
 HRESULT CopyTo([in] DWORD ciidExclude, [in, size_is(ciidExclude)] IID *rgiidExclude,
   [in] SNB snbExclude, [in] IStorage *pstgDest);
 HRESULT MoveElementTo([in] OLECHAR * pwcsName, [in] IStorage *pstgDest,
   [in] OLECHAR *pwcsNewName, [in] DWORD grfFlags);
 HRESULT Commit([in] DWORD grfCommitFlags);
 HRESULT Revert();
 HRESULT EnumElements([in] DWORD reserved1, [in, size_is(1)] void *reserved2,
   [in] DWORD reserved3, [out] IEnumSTATSTG **ppenum);
 HRESULT DestroyElement([in] OLECHAR *pwcsName);
 HRESULT RenameElement([in] OLECHAR *pwcsOldName, [in] OLECHAR *pwcsNewName);
 HRESULT SetElementTimes([in] OLECHAR *pwcsName, [in] FILETIME *pctime,
   [in] FILETIME *patime, [in] FILETIME *pmtime);
 HRESULT SetClass([in] REFCLSID clsid);
 HRESULT SetStateBits([in] DWORD grfStateBits, [in] DWORD grfMask);
 HRESULT Stat([out] STATSTG *pstatstg, [in] DWORD grfStatFlag);
}
```

Structured storage
IStream

```
interface IStream : ISequentialStream {
 HRESULT Seek([in] LARGE_INTEGER dlibMove, [in] DWORD dwOrigin,
   [out] ULARGE_INTEGER *plibNewPosition);
 HRESULT SetSize([in] ULARGE_INTEGER libNewSize);
 HRESULT CopyTo([in] IStream *pstm, [in] ULARGE_INTEGER cb,
   [out] ULARGE_INTEGER *pcbRead, [out] ULARGE_INTEGER *pcbWritten);
 HRESULT Commit([in] DWORD grfCommitFlags);
 HRESULT Revert();
 HRESULT LockRegion([in] ULARGE_INTEGER libOffset, [in] ULARGE_INTEGER cb,
   [in] DWORD dwLockType);
 HRESULT UnlockRegion([in] ULARGE_INTEGER libOffset, [in] ULARGE_INTEGER cb,
   [in] DWORD dwLockType);
 HRESULT Stat([out] STATSTG *pstatstg, [in] DWORD grfStatFlag);
 HRESULT Clone([out] IStream **ppstm);
}

interface ISequentialStream : IUnknown {
 HRESULT Read([out, size_is(cb), length_is(*pcbRead)] void *pv, [in] ULONG cb,
   [out] ULONG *pcbRead);
 HRESULT Write([in, size_is(cb)] void const *pv, [in] ULONG cb,
   [out] ULONG *pcbWritten);
}
```

Structured storage
IStorage

Do tworzenia obiektu `IStorage` skojarzonego z plikiem służą funkcje:

```
StgCreateStorageEx()
StgOpenStorageEx()
StgCreateDocfileOnILockBytes()
```

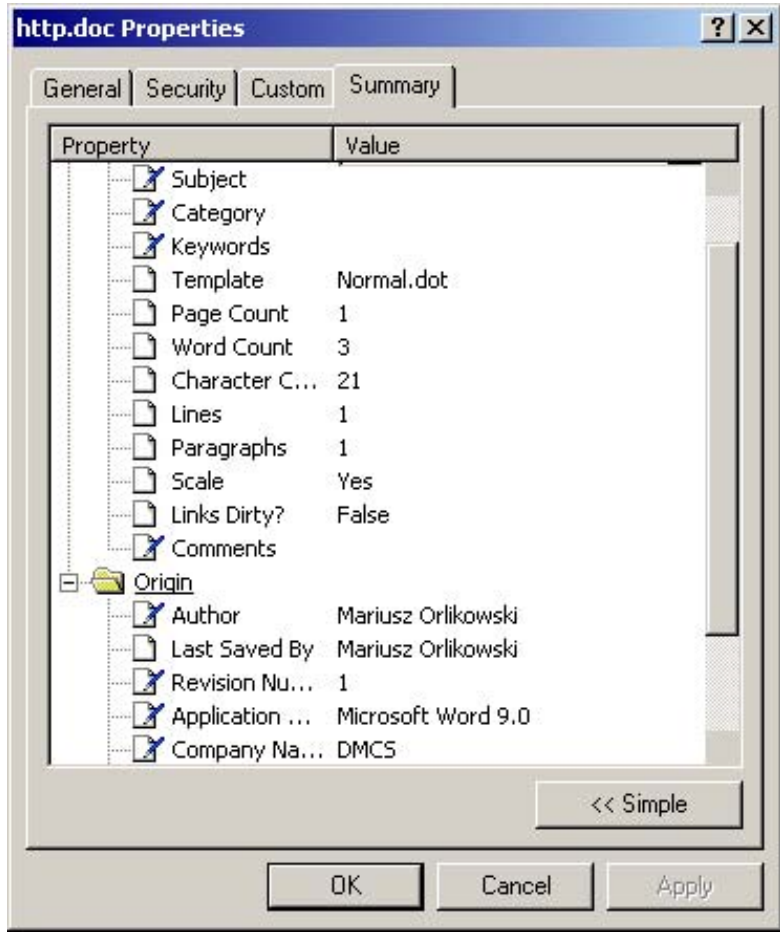
Przykład:

```
IStorage *pStorage;
hr = StgCreateStorageEx( L"C:\Plik.stg",
STGM_CREATE|STGM_SHARE_EXCLUSIVE|STGM_READWRITE,
STGFMT_DOCFILE, 0, NULL, NULL, IID_IStorage,
(void**)&pStorage);
...
pStorage->Release();
```

Structured storage

IPropertySetStorage, IPropertyStorage

Zawartość poszczególnych strumieni danych `IStream` może mieć dowolny format i jest najczęściej zrozumiała jedynie dla obiektu, który go zapisał. Aby system operacyjny mógł odczytać pewne podstawowe informacje o pliku, opracowano dwa standardowe interfejsy zapisujące informacje w formie zrozumiałej dla systemu. Przedstawiane są one jako kolejne zakładki podczas wyświetlania właściwości pliku.
Do zarządzania informacjami o właściwościach pliku opracowano dwa interfejsy `IPropertyStorage` i `IPropertySetStorage`.



Structured storage
IPropertySetStorage, IPropertyStorage

```
IPropertySetStorage : IUnknown {
 HRESULT Create([in] REFFMTID rfmtid, [in] CLSID *pclsid, [in] DWORD grfFlags,
  [in] DWORD grfMode, [out] IPropertyStorage **ppprstg);
 HRESULT Open([in] REFFMTID rfmtid, [in] DWORD grfMode,
  [out] IPropertyStorage **ppprstg);
 HRESULT Delete([in] REFFMTID rfmtid);
 HRESULT Enum([out] IEnumSTATPROPSETSTG **ppenum);
}

interface IPropertyStorage : IUnknown {
 HRESULT ReadMultiple([in] ULONG cpsp,[in,size_is(cpsp)] PROPSPEC rgpsp[],
   [out,size_is(cpspec)] PROPVARIANT rgpropvar[]);
 HRESULT WriteMultiple([in] ULONG cpsp, [in,size_is(cpsp)] PROPSPEC rgpsp[],
   [in,size_is(cpsp)] PROPVARIANT rgpropvar[], [in] PROPID propidNameFirst);
 HRESULT DeleteMultiple([in] ULONG cpsp, [in,size_is(cpsp)] PROPSPEC rgpsp[]);
 HRESULT ReadPropertyNames([in] ULONG cpid, [in,size_is(cpid)] PROPID rgpid[],
   [out,size_is(cpid)] LPOLESTR rglpwstrName[]);
 HRESULT WritePropertyNames([in] ULONG cpid, [in,size_is(cpid)] PROPID rgpid[],
   [in, size_is(cpid)] LPOLESTR rglpwstrName[]);
 HRESULT DeletePropertyNames([in] ULONG cpid, [in,size_is(cpid)] PROPID rgpid[]);
 HRESULT Commit([in] DWORD grfCommitFlags);
 HRESULT Revert();
 HRESULT Enum([out] IEnumSTATPROPSTG **ppenum);
 HRESULT SetTimes([in] FILETIME *pct, [in] FILETIME *pat, [in] FILETIME *pmt);
 HRESULT SetClass([in] REFCLSID clsid);
 HRESULT Stat([out] STATPROPSETSTG *pstatpsstg);
}
```

Structured storage
IPropertySetStorage, IPropertyStorage

Z obiektem `IStorage` może być skojarzonych wiele obiektów `IPropertySetStorage`. Każdy z nich identyfikowany jest przez GUID, który określa obiekt, który wizualizuje zestaw. Powszechnie stosowanymi rodzajami zestawów właściwości są:

`FMTID_SummaryInformation`
`FMTID_DocSummaryInformation`

Możliwe jest zdefiniowanie własnych typów zestawów i opracowanie obiektów, które będą je we właściwy sposób wizualizowały.

```
IProperySetStorage* pPSS;
IProperyStorage* pPS;

pStorage->QueryInterface(IID_ IProperySetStorage, (void**)pPSS);
pPSS->Create(FMTID_SummaryInformation, NULL, PROPSETFLAG_ANSI,
  STGM_CREATE|STGM_READWRITE|STGM_SHARE_EXCLUSIVE,&pPS));
...
pPS-_Release();
pPSS->Release();
```

Moniker

Moniker jest komponentem typu `IMoniker` służącym do aktywacji lub dostępu do zarejestrowanych obiektów poprzez jego nazwę. Moniker może obsługiwać różnego typu obiekty w systemie, takie jak pliki, wskaźniki, URL, a także obiekty COM.
Dziedziczy on od interfejsu IPersistStream, który z kolei dziedziczy od IPersist. Interfejsy te pozwalają na aktywację obiektów i odtworzenie ich stanu.

```
interface IPersist : IUnknown
{
  HRESULT GetClassID([out] CLSID *pClassID);
}

interface IPersistStream : IPersist
{
  HRESULT IsDirty(void);
  HRESULT Load([in] IStream *pStm);
  HRESULT Save([in] IStream *pStm,[in] BOOL fClearDirty);
  HRESULT GetSizeMax([out] ULARGE_INTEGER *pcbSize);
}
```

Moniker
IPersist, IPersistStream

`IPersist::GetClasID()` zwraca CLSID komponentu. Podczas zapisywania stanu obiektu, zapamiętywany jest również CLSID w celu uniknięcia konfliktów typu podczas odczytu.

`IPersistStream::IsDirty()` informuje czy obiekt został zmieniony o ostatniego zapisu.
`IPersistStream::Load()` odczytuje stan obiektu ze strumienia.
`IPersistStream::Save()` zapisuje stan obiektu do strumienia.
`IPersistStream::GetSizeMax()` identyfikuje rozmiar w bajtach potrzebny do zapisania obiektu.

Moniker
IMoniker

```
interface IMoniker:IPersistStream {
 HRESULT BindToObject([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [in] REFIID riidResult, [out] void **ppvResult);
 HRESULT BindToStorage([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [in] REFIID riid, [out] void **ppvObj);
 HRESULT Reduce([in] IBindCtx *pbc, [in] DWORD dwReduceHowFar,
  [in, out] IMoniker **ppmkToLeft, [out] IMoniker **ppmkReduced);
 HRESULT ComposeWith([in] IMoniker *pmkRight, [in] BOOL fOnlyIfNotGeneric,
  [out] IMoniker **ppmkComposite);
 HRESULT Enum([in] BOOL fForward, [out] IEnumMoniker **ppenMoniker);
 HRESULT IsEqual([in] IMoniker *pmkOtherMoniker);
 HRESULT Hash([out] DWORD *pdwHash);
 HRESULT IsRunning([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [in] IMoniker *pmkNewlyRunning);
 HRESULT GetTimeOfLastChange([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [out] FILETIME *pFileTime);
 HRESULT Inverse([out] IMoniker **ppmk);
 HRESULT CommonPrefixWith([in] IMoniker *pmkOther, [out] IMoniker **ppmkPrefix);
 HRESULT RelativePathTo([in] IMoniker *pmkOther, [out] IMoniker **ppmkRelPath);
 HRESULT GetDisplayName([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [out] LPOLESTR *ppszDisplayName);
 HRESULT ParseDisplayName([in] IBindCtx *pbc, [in] IMoniker *pmkToLeft,
  [in] LPOLESTR pszDisplayName, [out] ULONG *pchEaten, [out] IMoniker **ppmkOut);
 HRESULT IsSystemMoniker([out] DWORD *pdwMs);
}
```

Moniker
IMoniker

`ParseDisplayName()` analizuje łańcuch tekstowy i przypisuje monikerowi nazwę obiektu. Dla złożonych nazw analizowany jest początek łańcucha do pierwszego niezrozumiałego wyrażenia. Pozostała część łańcucha pozostawiana jest do analizy przez inny moniker, który zostanie zgrupowany z bieżącym. Metoda ta najczęściej nie jest wywoływana bezpośrednio ale przez funkcję systemową `MkParseDisplayName`, która wywołuje kolejne monikery, interpretując kolejne fragmenty, a następnie zwraca moniker całej grupy.

```
ULONG eaten;
IBindCtx *pbc;
IMoniker* pMoniker;
OLECHAR str[] = L"clsid:10000013-0000-0000-0000-000000000001";

CreateBindCtx( 0, &pbc );
MkParseDisplayName(pbc, str, &eaten, pMoniker);
```

Moniker
IMoniker

`BindToObject()` łączy moniker z obiektem o zdefiniowanej nazwie i określonym IID (znajduje istniejący lub aktywuje nowy). Przekazywany parametr IBindCtx informuje min. czy przy aktywacji obiektu należy załadować z pliku stan obiektu (obiekt typu *persistent*) i czy też maksymalny czas przeznaczony na łączenie się z obiektem.

```
IStopwatch *pStopwatch;

pMoniker->BindToObject( pbc, NULL, IID_IStopwatch, &pStopwatch );
pbc->Release(); // pStopwatch now points to the object

// do some job with pStopwatch

pStopwatch->Release();
```

`IsRunning()` informuje, czy obiekt identyfikowany przez moniker jest załadowany i uruchomiony.

Moniker
IMoniker

`GetDisplayName()` zwraca łańcuch znakowy reprezentujący nazwę obiektu lub grupy.

`ComposeWith()` łączy monikery w grupy (*composition*) zwracając jeden wspólny moniker. Funkcja ta może być używana do tworzenia pełnej ścieżki do pliku na bazie niekompletnej i relatywnej ścieżki.

`Enum()` pozwala na przeglądanie składowych grupy monikerów utworzonych przez `ComposeWith()`.

`IsEqual()` porównuje moniker z innym i zwraca informację czy wskazują ten sam obiekt.

`GetTimeOfLastChange()` zwraca czas ostatniej modyfikacji obiektu.

Moniker
Running Object Table (ROT)

ROT jest tablicą przechowującą zarejestrowane obiekty. Pozwala ona na dostęp do jednej instancji obiektu z różnych procesów. Komponent lub aplikacja go aktywująca może zarejestrować informację o nim, dzięki czemu inny proces może się do tego odwołać. Ponieważ sam obiekt nie przechowuje informacji o swoim CLSID i IID, musi być on zarejestrowany wraz z monikerem go opisującym. Dostęp do ROT odbywa się poprzez obiekt `IRunningObjectTable`. `IMoniker` podczas wywołania `BindToObject()` najpierw przeszukuje ROT czy żądany obiekt już jest zarejestrowany i zwraca jego referencję.
Dostęp do ROT uzyskuje się przez funkcję `GetRunningObjectTable()`:

```
IRunningObjectTable* pROT;
GetRunningObjectTable(NULL, &pROT);
```

Moniker
Running Object Table (ROT)

```
interface IRunningObjectTable : IUnknown {
  HRESULT Register (
    [in] DWORD grfFlags,
    [in] IUnknown *punkObject,
    [in] IMoniker *pmkObjectName,
    [out] DWORD *pdwRegister );
  HRESULT Revoke ( [in] DWORD dwRegister );
  HRESULT IsRunning ( [in] IMoniker *pmkObjectName );
  HRESULT GetObject (
    [in, unique] IMoniker *pmkObjectName,
    [out] IUnknown **ppunkObject );
  HRESULT NoteChangeTime (
    [in] DWORD dwRegister,
    [in] FILETIME *pfiletime );
  HRESULT GetTimeOfLastChange (
    [in] IMoniker *pmkObjectName,
    [out] FILETIME *pfiletime );
  HRESULT EnumRunning ( [out] IEnumMoniker **ppenumMoniker );
}
```

Moniker
Running Object Table (ROT)

```
HRESULT Register( DWORD grfFlags,
  IUnknown *punkObject,
  IMoniker *pmkObjectName,
  DWORD *pdwRegister );
```

`Register()` **rejestruje moniker wraz z obiektem w ROT:**

- `grfFlags`:
  - `ROTFLAGS_REGISTRATIONKEEPSALIVE` **- ROT wywołuje metodę** `AddRef()` **przy rejestracji obiektu. Gdy flaga ta nie jest ustawiona wywoływane jest** `Release()` **i obiekt jest usuwany z ROT gdy ostatnie połączenie z obiektem jest zwolnione. W przeciwnym wypadku** `Release()` **jest wywoływane dopiero przy jawnym wywołaniu** `Revoke()`.
  - `ROTFLAGS_ALLOWANYCLIENT` **- gdy ustawiona pozwala na dostęp do obiektu każdemu użytkownikowi. Gdy flaga ta nie jest ustawiona, dostęp do obiektu ma jedynie użytkownik rejestrujący obiekt.**

Moniker
Running Object Table (ROT)

```
HRESULT Revoke( DWORD dwRegister );
```

`Revoke()` **usuwa zarejestrowany obiekt z ROT.**

```
HRESULT IsRunning( IMoniker *pmkObjectName);
```

`IsRunning()` **sprawdza czy obiekt identyfikowany przez moniker zarejestrowany jest w ROT.**

```
HRESULT GetObject(
   IMoniker *pmkObjectName,
   IUnknown **ppunkObject);
```

`GetObject()` **pobiera interfejs obiektu zarejestrowanego w ROT identyfikowanego przez moniker.**

Moniker
Running Object Table (ROT)

```
HRESULT NoteChangeTime(
  DWORD dwRegister,
  FILETIME *pfiletime );
```

`NoteChangeTime()` **aktualizuje czas ostatniej modyfikacji obiektu.**

```
HRESULT GetTimeOfLastChange(
  IMoniker *pmkObjectName,
  FILETIME *pfiletime );
```

`GetTimeOfLastChange()` **odczytuje czas ostatniej modyfikacji obiektu identyfikowanego przez moniker.**

```
HRESULT EnumRunning( IEnumMoniker **ppenmMoniker );
```

`EnumRunning()` **pozwala na przegląd obiektów zarejestrowanych w ROT.**