Compound Files
Structured Storage, Persistence, and Monikers

Jim Fawcett
CSE775 – Distributed Objects
Spring 2006

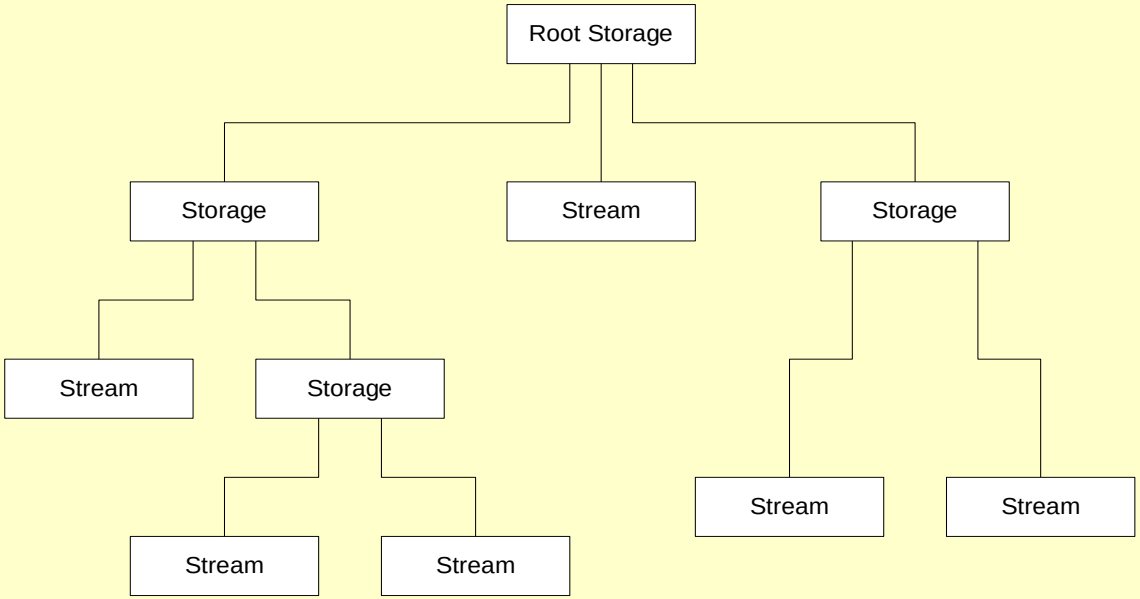
COM components + persistent storage = objects

Compound Files

- A compound file is a file capable of containing, as separate unique identities, the states of more than one object.
- Compound files contain storages and streams:
  - A storage is analogous to a directory. It is a container for other storages and streams.
  - A stream is analogous to a file. It consists of a stream of bytes. No further division or structure is present in a stream.
- Each compound file has a root storage, below which other storages and streams exist.

Structured Storage

- Structured storage allows several objects to store their persistent data in a single file



Ownership

- Each software component that shares a compound file can be assigned its own stream to store its persistent data.
- If a component needs to store complex data it can be assigned a storage, allowing it to create its own substorages and streams as appropriate for its own activities.
- You assign a stream or storage to an object by associating it with the object's CLSID.

Structured Storage API Functions

- StgCreateDocFile
  - creates a new compound file and returns a pointer to the IStorage interface of the new file's root storage.
- StgOpenStorage
  - opens an existing compound file.
- StgIsStorageFile
  - indicates if a named file is a compound file.

IStorage

- Storages support the IStorage interface:
  - **CreateStream** creates a stream below the storage object
  - **OpenStream** opens a stream below the storage object
  - **CreateStorage** creates a new storage below the storage object
  - **OpenStorage** opens a storage below the storage object
  - **DestroyElement** destroys a stream or storage below the storage object
  - **RenameElement** renames a stream or storage below the storage object
  - **CopyTo** copies contents of one storage into another
  - **MoveElement** copies then deletes a storage

IStorage (continued)

- IStorage:
  - **EnumElements** returns list of elements contained in storage object
  - **SetElementTimes** sets creation, access, and modification times of a storage or stream
  - **SetClass** persistently stores a CLSID in its own stream immediately below the storage object. This allows persistent storage of a COM object. Its data is stored in one or more streams and its methods are identified by the CLSID.
  - **Stat** returns information about the storage object including a CLSID stored by invoking SetClass.
  - **Commit** commits changes made since the last commit request for a storage object opened in transacted mode.
  - **Revert** discards all changes made since the last commit request.

IStream

- Streams support the IStream interface:
  - **Read** reads a specified number of bytes from the stream object
  - **Write** writes a specified number of bytes to the stream object
  - **Seek** moves to a specified number of bytes from the beginning, end, or relative to current location
  - **CopyTo** copies a range of bytes from one stream to another
  - **LockRegion** lock a range of bytes in a stream
  - **UnlockRegion** unlocks a range of bytes in a stream
  - **Commit** - not currently supported for streams
  - **Revert** - not currently supported for streams
  - **Stat** retrieves STATSTG structure for this stream
  - **Clone** creates a new stream object referring to same bytes.

COM Object Persistence

- Persistent objects usually use structured storage through one of the following interfaces:
  - IPersistStream used by clients to ask an object to load its persistent data from and save to a stream.
  - IPersistStreamInit adds a method to IPersistStream to tell an object it's being initialized for the first time.
  - IPersistStorage used by clients to ask an object to load its persistent data from and save to a storage.
  - IPersistFile used by clients to tell an object to load and save persistent data to a flat file.
  - IPersistPropertyBag allows a client to ask an object to load and save property values, each of which is a string.
  - IPersistMemory like IPersistStreamInit except it uses memory.

IPersistStream

- IPersistStream supports the methods:
  - **Load** instructs an object to load its persistent data from a stream
  - **Save** instructs an object to save its persistent data to a stream
  - **IsDirty** allows a client to determine if an objects persistent data has been modified
  - **GetSizeMax** returns maximum size stream needed to save the current persistent data

- IPersistStreamInit supports these plus the method:
  - **InitNew** allows client to instruct the component that it is being initialized for the first time.

IPersistStorage

- The IPersistStorage interface supports the methods:
  - **InitNew** allows a client to pass a storage object pointer for sub-sequent use.
  - **Load** instructs the object to retrieve its persistent data
  - **Save** instructs the object to save its persistent data. After save the object cannot again write to the storage until it receives a SaveCompleted message.
  - **SaveCompleted** indicates that the object can again write to its storage.
  - **HandsOffStorage** causes object to release any pointers to streams or substorages it has opened below its storage, allowing the client to safely copy the storage. The client calls SaveCompleted to allow the object to write to storage again.

Persistence and Monikers

- Monikers are COM objects supporting the IMoniker interface which derives from the IPersistStream interface.
- A moniker is a name for a specific object instance, e.g., a particular combination of a CLSID and persistent data.
- Each moniker identifies only one object instance.
- Clients can use monikers to create a COM object and initialize it in one operation:
  - IPersistMoniker lets a client ask an object to load and retrieve its persistent data using a moniker. This allows the client to surrender control to the object regarding how it retrieves and stores its persistent data.

Moniker Types

- File moniker identifies a file-based object and is created with the CreateFileMoniker function. File monikers are used to instantiate a class associated with some specific file and initialize the class with that data, e.g., Word and *.docs.

- Item moniker is based on a string that identifies an object in a container. Item monikers can be used to identify objects smaller than a file like embedded objects in a compound file and non-objects like a range of spreadsheet cells. They are created with the CreateItemMoniker function.

- Generic composite monikers consist of two or more monikers of arbitrary type that have been composed together. They are created with the function CreateGenericComposite.

Moniker Types (continued)

- Anti-monikers are used to construct relative monikers, analogous to a relative path. They specify a location of an object relative to the location of another object. They are created with the function CreateAntiMoniker.

- Pointer monikers are non-persistent and wrap a pointer to an object loaded in memory. Pointer monikers identify objects that cannot be saved to persistent memory. Pointer monikers are created with CreatePointerMoniker.

Moniker Types (continued)

- Class Monikers act as wrappers for the CLSID of a COM class.
  - One common use is to call MkParseDisplayName with the string form of a CLSID, e.g., clsid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx and a binding context, getting back a class moniker.
  - The object can then be activated by calling BindToObject like this:

```
pMoniker->BindToObject(
  pClassBindCtx,
  NULL,
  IID_IUnknown,
  (void**)&pUnknown
);
```

- URL Monikers manage Uniform Resource Locators

IMoniker

- Some of the methods IMoniker supports are:
  - BindToObject instantiates the object the moniker refers to and returns a pointer to a specific interface on the object.
  - BindToStorage returns a pointer to an object's stream or storage instead of one of its interfaces. It is possible that the object is not instantiated by this call.
  - ComposeWith returns a new moniker that is a composite of two exisiting monikers supplied as arguments.
  - IsRunning indicates whether the object is currently running.

- Two kinds of objects call the IMoniker methods:
  - A component that contains objects to be identified with a moniker and provides the moniker to other objects.
  - A client object that needs to bind to the object identified by the moniker.

Running Object Table

- When a client wants to bind to an object using a moniker it is possible that the object is already running. To allow a moniker to bind to a currently active object COM supports the Running Object Table (ROT). This object can be accessed via the IRunningObjectTable interface which supports the methods:
  - Register registers a running object in the table
  - Revoke removes an object from the table
  - GetObject indicates if an object with a particular moniker is currently running. If so, the call returns a pointer to the object.
- You access it by calling the global GetRunningObjectTable function.