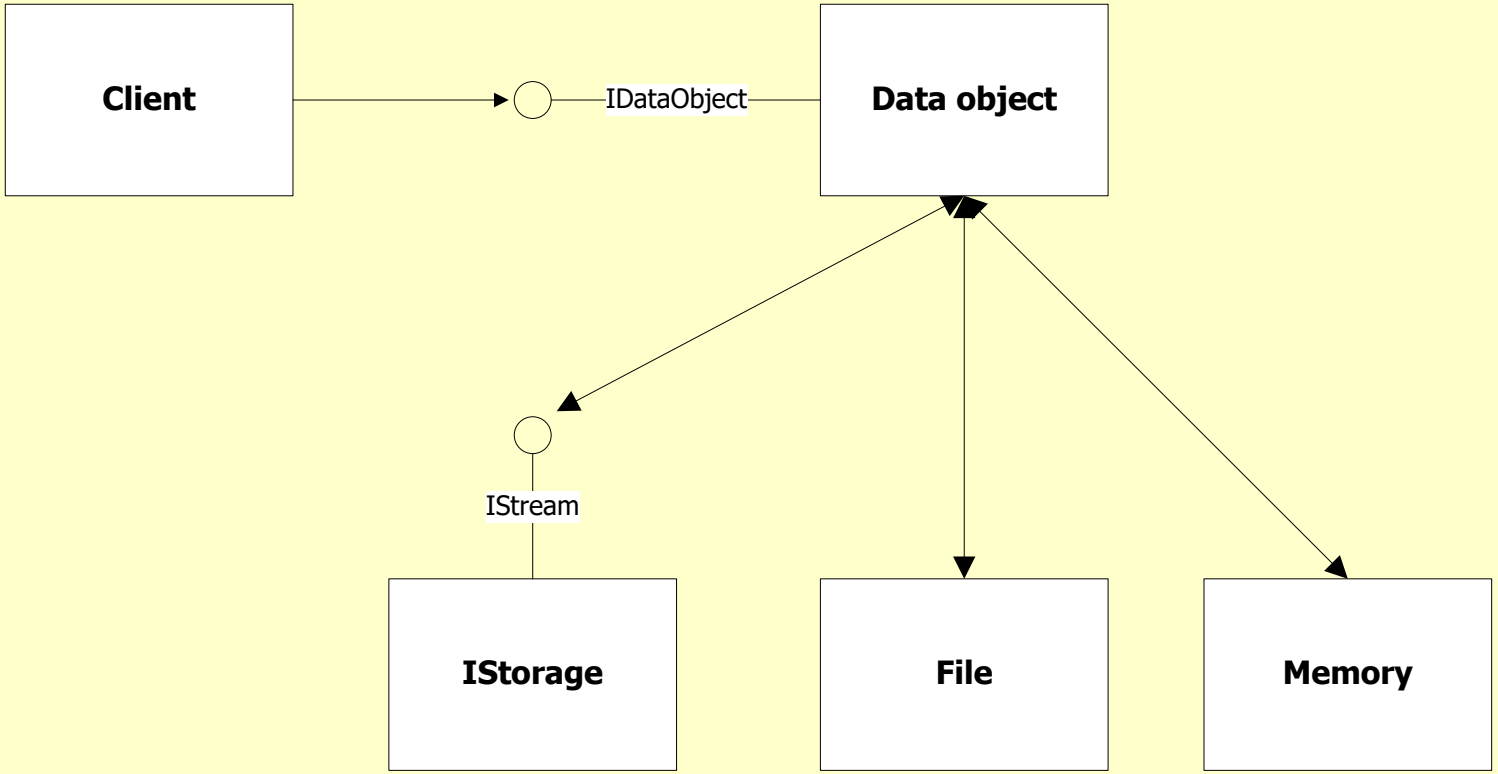
Uniform Data Transfer and Connectable Objects

On the way to (ActiveX) Shangri-La

Uniform Data Transfer

- To provide a standard way of transferring data between components or applications COM defines the IDataObject interface.
- The purpose of the IDataObject interface is to allow any application that knows how to act as a client for this interface to access data provided from any application that supports it, e.g., uniform data transfer.
- IDataObject interface also provides a way to notify clients of recent changes in its component's data.

Uniform Data Transfer



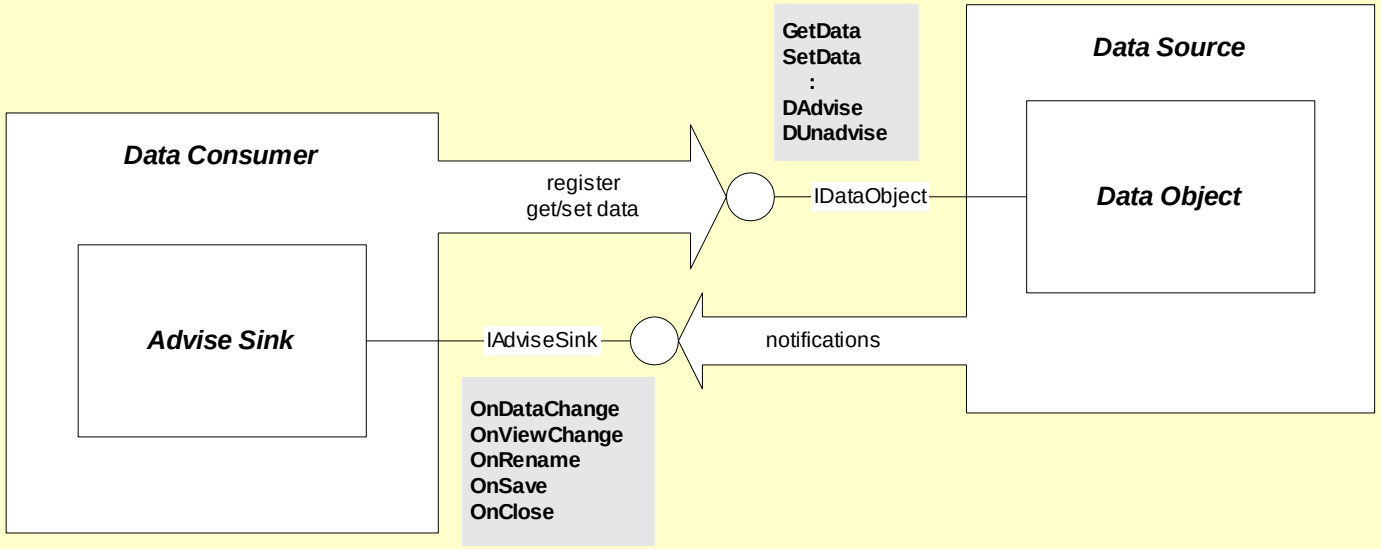
IDataObject

- The IDataObject interface uses two structures to support the transfer of data. The first is:
  - FORMATETC describes the data being transferred:
    - data format - could be the standard clipboard formats or custom formats defined by the data object and known by the client
    - target device - details about the intended destination, e.g., printer with a given resolution
    - role for which the data is designed - icon, thumbnail image, full screen bitmap
    - how the data should be transferred - global memory, disk file, storage objects

IDataObject

- The IDataObject interface uses two structures to support the transfer of data. The second is:
  - STGMEDIUM describes where the data is stored:
    - tag which tells a marshaler how the data is stored
    - a union which indicates that the data is stored in:
      - bitmap
      - metafile
      - global memory
      - file
      - stream
      - storage

Data Object


GetData
SetData
:
DAdvise
DUnadvise
Data Source
Data Consumer
register
get/set data
IDataObject
Data Object
Advise Sink
IAdviseSink
notifications
OnDataChange
OnViewChange
OnRename
OnSave
OnClose

IDataObject Methods

GetData
Renders the data described in a **FORMATETC** structure and transfers it through the **STGMEDIUM** structure.

GetDataHere
Renders the data described in a **FORMATETC** structure and transfers it through the **STGMEDIUM** structure allocated by the caller.

QueryGetData
Determines whether data object is capable of rendering data described in the **FORMATETC** structure.

GetCanonicalFormatEtc
Provides a potentially different but logically equivalent **FORMATETC** structure.

SetData
Provides source data object with data described by a **FORMATETC** structure and **STGMEDIUM** structure.

IDataObject Methods (continued)

EnumFormatEtc
Creates and returns a pointer to an object to enumerate the **FORMATETC** supported by the data object.

DAdvise
Creates a connection between a data object and an advise sink so the advise sink can receive notifications of changes in the data object.

DUnadvise
Destroys a notification previously set up with the **DAdvise** method.

EnumDAdvise
Creates and returns a pointer to an object to enumerate the current advisory connections.

Notifications

- It may be important for the data object to notify a client of changes in its data content or views.
- The IAdvise interface was designed to support notification of the client by data objects in a general way (more general than the IDataObject interface):
  - notify and get new data to the client
  - notify the client of changes in view or data source

IAdviseSink Methods

OnDataChange
Advises that data has changed. Uses FORMATETC and STGMEDIUM to pass data to client.

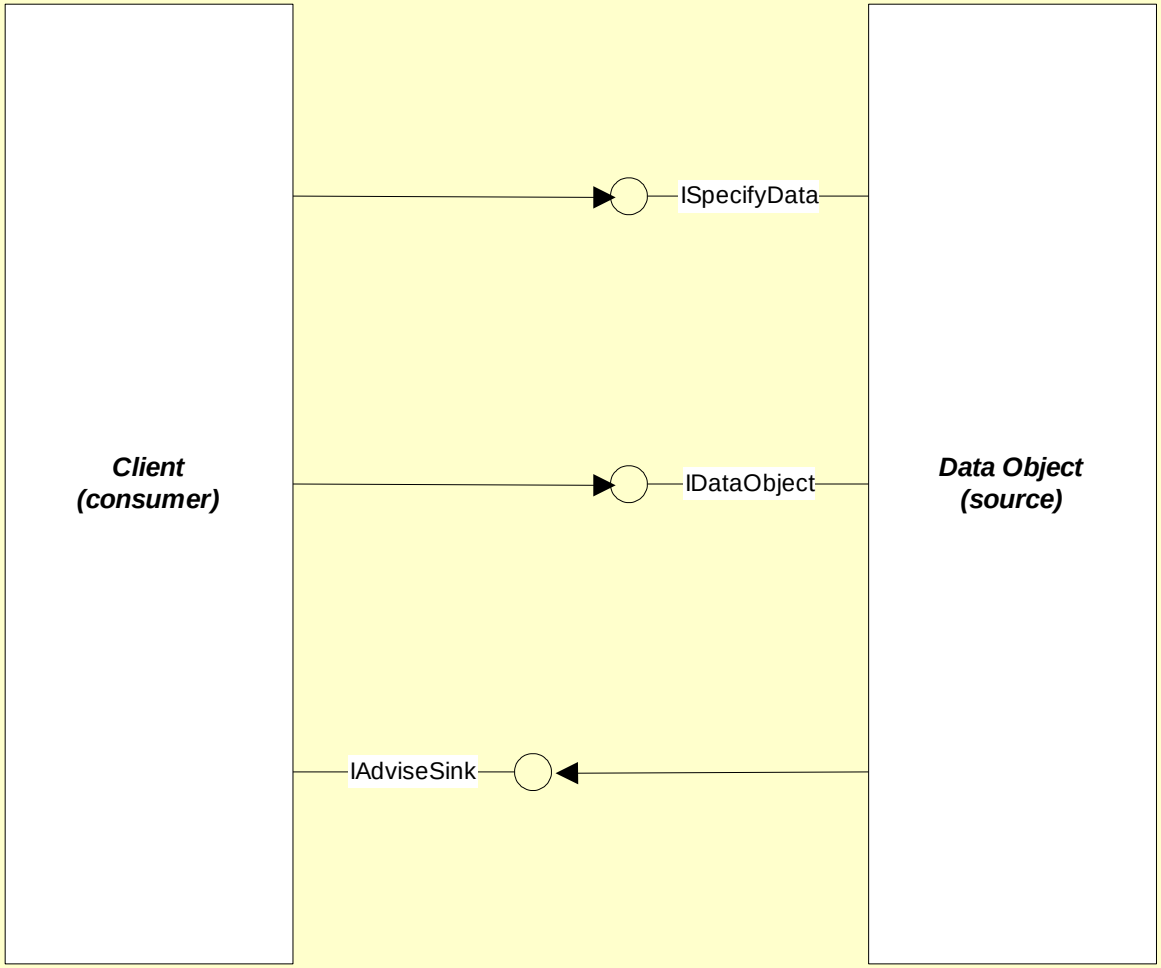
OnViewChange
Advises that view of object has changed. Uses a DVASPEC enumeration to define the view, e.g., content, thumbnail, icon, ...

OnRename
Advises that name of object has changed.

OnSave
Advises that object has been saved to disk.

OnClose
Advises that object has been closed.

Data Transfer Using IAdvise Interface



Uniform Data Transfer using Notification

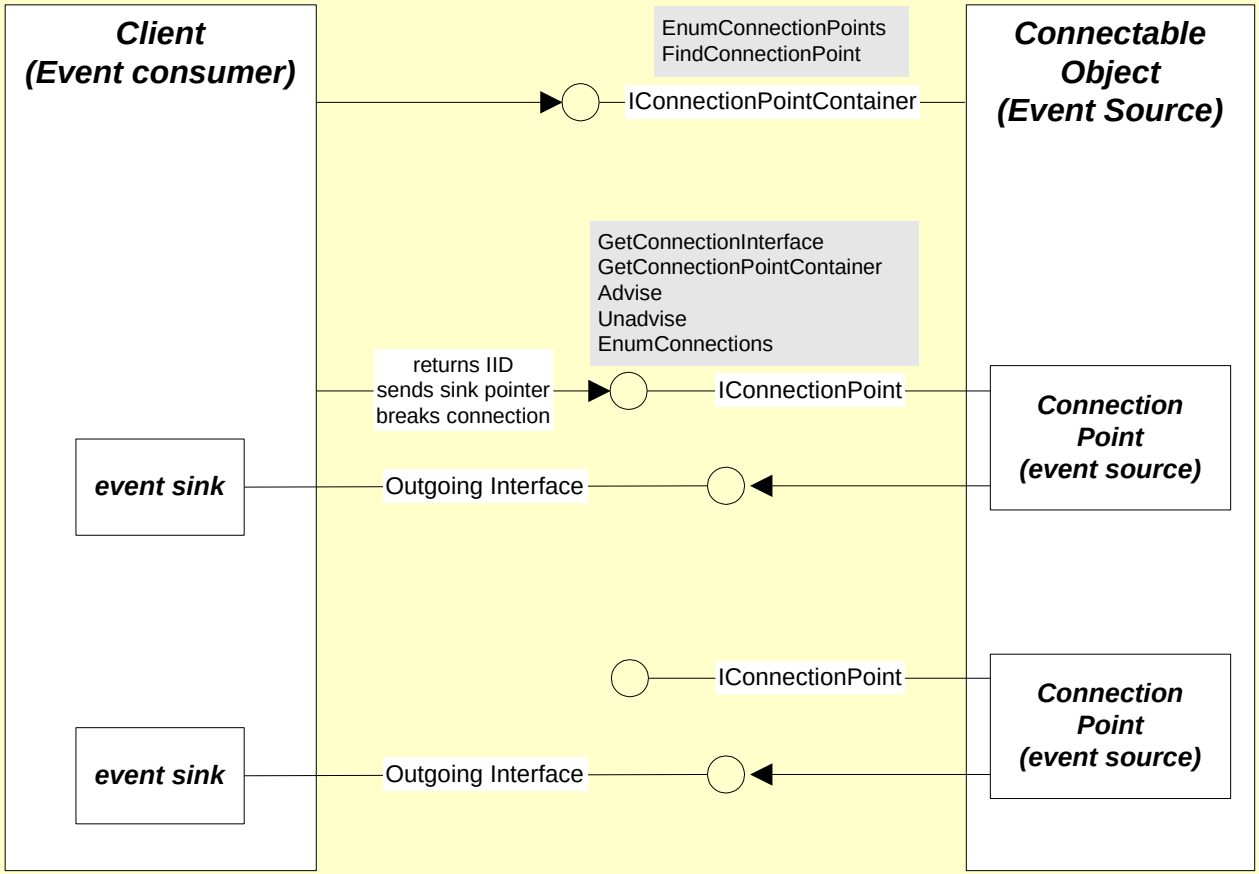
- Referring to the diagram on the preceding page:
  - The client specifies to the data object which data it is interested in using a custom interface called ISpecifyData.
  - The client passes a pointer to its IAdviseSink interface to the data object.
  - Data object notifies client of a change using

    IAdviseSink::OnDataChange

Connectable Objects

- Connectable objects support interfaces to allow two-way communication with one or more clients.
- Clients talk to the connectable object in the usual way, e.g., by getting a pointer to one of its interfaces using CoCreateInstance and QueryInterface.
- Each connectable object provides connection points for specific interfaces, defined by the connectable object at design time.
- The connectable object's clients implement the interfaces used by connectable objects to talk back.

Connectable Objects



Establishing Two-way Connections

- The IConnectionPointContainer interface must be supported by every connectable object.
- Its purpose is to allow clients to discover what outgoing (client) interfaces the connectable object supports and get pointers to them.
- Once the client has a pointer to a connection point object for its interface that the connectable object will use it must then send a pointer to that interface to the connection point object. It does that using the IConnectionPoint interface.

IConnectionPointContainer

- Provided so clients can learn which outgoing (client) interfaces the object supports.
- Each of these supported client interfaces is represented with the connectable object by a separate connection point object.
- Connection point objects handle only one type of outgoing interface (they know how to call that interface's methods) and also must provide an interface called IConnectionPoint.
- The connectable object's clients must provide a sink object that implements the outgoing interface.

IConnectionPointContainer Interface

EnumConnectionPoints
Returns an object to enumerate all the connection points supported in the connectable object.

FindConnectionPoint
Returns a pointer to the **IConnectionPoint** interface for a specified connection point.

IEnumConnectionPoints Interface

IEnumConnectionPoints::Next
Enumerates the next *cConnections* elements (**IConnectionPoint** pointers) in the enumerator's list, returning them in *rgpcn* along with the actual number of enumerated elements in *pcFetched*. The enumerator calls **IConnectionPoint::AddRef** for each returned pointer in *rgpcn*, and the caller is responsible for calling **IConnectionPoint::Release** through each pointer when those pointers are no longer needed. E_NOTIMPL is not allowed as a return value. If an error value is returned, no entries in the *rgpcn* array are valid on exit and require no release.

IEnumConnectionPoints::Skip
Instructs the enumerator to skip the next *cConnections* elements in the enumeration so that the next call to **IEnumConnectionPoints::Next** will not return those elements.

IEnumConnectionPoints::Reset
Instructs the enumerator to position itself at the beginning of the list of elements. There is no guarantee that the same set of elements will be enumerated on each pass through the list, nor will the elements necessarily be enumerated in the same order. The exact behavior depends on the collection being enumerated. It is too expensive for some collections, such as files in a directory, to maintain a specific state.
IEnumConnectionPoints::Clone
Creates another connection point enumerator with the same state as the current enumerator to iterate over the same list. This method makes it possible to record a point in the enumeration sequence in order to return to that point at a later time. The caller must release this new enumerator separately from the first enumerator.

IConnectionPoint

- With the IConnectionPoint interface a client starts, or terminates, an advisory loop with the connectable object and the client's own sink.
- The client can also use this interface to get an enumerator object with the IEnumConnections interface to enumerate the connections that it knows about.

IConnectionPoint Interface

GetConnectionInterface
Returns the IID of the outgoing interface managed by this connection point.

GetConnectionPointContainer
Returns the parent (connectable) object's **IConnectionPointContainer** interface pointer.

Advise
Creates a connection between a connection point and a client's sink, where the sink implements the outgoing interface supported by this connection point.

Unadvise
Terminates a notification previously set up with **Advise**.

EnumConnections
Returns an object to enumerate the current advisory connections for this connection point.

IEnumConnections

IEnumConnections::Next
Enumerates the next *cConnections* elements (i.e., **CONNECTDATA** structures) in the enumerator's list, returning them in *rgpcd* along with the actual number of enumerated elements in *pcFetched*. The caller is responsible for calling CONNECTDATA.pUnk->Release
for each element in the array once this method returns successfully. If *cConnections* is greater than one, the caller must also pass a non-NULL pointer to *pcFetched* to get the number of pointers it has to release. E_NOTIMPL is not allowed as a return value. If an error value is returned, no entries in the *rgpcd* array are valid on exit and require no release.

IEnumConnections::Skip
Instructs the enumerator to skip the next *cConnections* elements in the enumeration so that the next call to **IEnumConnections::Next** will not return those elements.

IEnumConnections::Reset
Instructs the enumerator to position itself at the beginning of the list of elements. There is no guarantee that the same set of elements will be enumerated on each pass through the list. It depends on the collection being enumerated. It is too expensive for some collections, such as files in a directory, to maintain a specific state.

IEnumConnections::Clone
Creates another connection point enumerator with the same state as the current enumerator to iterate over the same list. This method makes it possible to record a point in the enumeration sequence in order to return to that point at a later time. The caller must release this new enumerator separately from the first enumerator.