1Connectable Objects

The COM technology known as Connectable Objects (also called “connection points”) supports a generic ability for any object, called in this context a “connectable” object, to express these capabilities:

- The existence of “outgoing” interfaces,[1] such as event sets
- The ability to enumerate the IIDs of the outgoing interfaces
- The ability to connect and disconnect “sinks” to the object for those outgoing IIDs
- The ability to enumerate the connections that exist to a particular outgoing interface.

Support for these capabilities involves four interfaces: IConnectionPointContainer, IEnumConnectionPoints, IConnectionPoint, and IEnumConnections. A “connectable object” implements IConnectionPointContainer to indicate existence of outgoing interfaces. Through this interface a client can enumerate connection points for each outgoing IID (via an enumerator with IEnumConnectionPoints) and can obtain an IConnectionPoint interface to a connection point for each IID. Through a connection point a client starts or terminates an advisory loop with the connectable object and the client’s own sink. The connection point can also enumerate the connections it knows about through an enumerator with IEnumConnections.

1.1The *IConnectionPoint* Interface

The ability to connect to a single outgoing interface (that is, for a unique IID) is provided by a “connection point” sub-object that is conceptually owned by the connectable object. The object is separate to avoid circular reference counting problems. Through this interface the connection point allows callers to connect a sink to the connectable object, to disconnect a sink, or to enumerate the existing connections.

IDL:

```
[
uuid(B196B286-BAB4-101A-B69C-00AA00341D07)
   , object, pointer_default(unique)
]
interface IConnectionPoint : IUnknown
   {
   HRESULT GetConnectionInterface([out] IID *pIID);
   HRESULT GetConnectionPointContainer([out] IConnectionPointContainer **ppCPC);
   HRESULT Advise([in] IUnknown *pUnk, [out] DWORD *pdwCookie);
   HRESULT Unadvise([in] DWORD dwCookie);
   HRESULT EnumConnections([out] IEnumConnections **ppEnum);
   }
```

A connection point is allowed to stipulate how many connections (one or more) it will allow in its implementation of Advise. A connection point that allows only one interface can return E_NOTIMPL from EnumConnections.

1.1.1IConnectionPoint::GetConnectionInterface

HRESULT IConnectionPoint::GetConnectionInterface([out] IID *pIID);

Returns the IID of the outgoing interface managed by this connection point. This is provided such that a client of IEnumConnectionPoints can determine the IID of each connection point thus enumerated. The IID returned from this method must enable the caller to access this same connection point through IConnectionPointContainer::FindConnectionPoint.

Argument	Type	Description
pIID	IID *	[out] A pointer to the caller’s variable to receive the IID of the outgoing interface managed by this connection point.

[1] An “outgoing” interface is one that an object defines itself but for which the object is itself a client. Another piece of code called the “sink” (generically) implements the outgoing interface such that the object can call the sink.

Return Value	Meaning
S_OK	Success.
E_POINTER	The address in pIID is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.

Comments:

This function must be completely implemented in any connection point; therefore E_NOTIMPL is not an acceptable return code.

1.1.2IConnectionPoint::GetConnectionPointContainer

HRESULT IConnectionPoint::GetConnectionPointContainer([out] IConnectionPointContainer **ppCPC);

Retrieves the IConnectionPointContainer interface pointer to the connectable object that conceptually owns this connection point. The caller becomes responsible for the pointer on a successful return.

Argument	Type	Description
ppCPC	IConnectionPointContainer *	[out] A pointer to the caller's variable in which to return a pointer to the connectable object's IConnectionPointContainer interface. The connection point will call IConnectionPointContainer::AddRef before returning and the caller must call IConnectionPoint::Release when it is done using the pointer.

Return Value	Meaning
S_OK	Success.
E_POINTER	The value in ppCPC is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.

Comments:

E_NOTIMPL is not an allowable return code.

1.1.3IConnectionPoint::Advise

HRESULT IConnectionPoint::Advise([in] IUnknown *pUnk, [out] DWORD *pdwCookie);

Establishes an advisory connection between the connection point and the caller's sink object identified with pUnk. The connection point must call pUnk->QueryInterface(iid, ...) on this pointer in order to obtain the correct outgoing interface pointer to call when events occur, where *iid* is the inherent outgoing interface IID managed by the connection point (that is, the that when passed to IConnectionPointContainer::FindConnectionPoint would return an interface pointer to this same connection point).

Upone successful return, the connection point provides a unique "cookie" in *pdwCookie that must be later passed to IConnectionPoint::Unadvise to terminate the connection.

Argument	Type	Description
pUnk	IUnknown *	[in] The IUnknown pointer to the client's sink that wishes to receive calls for the outgoing interface managed by this connection point. The connection point must query this pointer for the correct outgoing interface. If this query fails, this member returns CONNECT_E_CANNOTCONNECT.
pdwCookie	DWORD *	[out] A pointer to the caller's variable that is to receive the connection "cookie" when connection is successful. This cookie must be unique for each connection to any given *instance* of a connection point.

Return Value	**Meaning**
S_OK	The connection has been established and *pdwCookie has the connection key.
E_POINTER	The value of pUnk or pdwCookie is not valid (NULL cannot be passed for either argument)
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There was not enough memory to complete the operation, such as if the connection point failed to allocate memory in which to store the sink's interface pointer.
CONNECT_E_ADVISELIMIT	The connection point has already reached its limit of connections and cannot accept any more.
CONNECT_E_CANNOTCONNECT	The sink does not support the interface required by this connection point.

1.1.4IConnectionPoint::Unadvise

HRESULT IConnectionPoint::Unadvise([in] DWORD dwCookie);

Terminates an advisory connection previously established through IConnectionPoint::Advise. The dwCookie argument identifies the connection to terminate.

Argument	Type	Description
dwCookie	DWORD	[in] The connection "cookie" previously returned from IConnectionPoint::Advise.

Return Value	**Meaning**
S_OK	The connection was successfully terminated.
E_UNEXPECTED	An unknown error occurred.
CONNECT_E_NOCONNECTION	dwCookie does not represent a value connection to this connection point.

1.1.5IConnectionPoint::EnumConnections

HRESULT IConnectionPoint::EnumConnections([out] IEnumConnections **ppEnum);

Creates an enumerator object for iteration through the connections that exist to this connection point.

Argument	Type	Description
ppEnum	IEnumConnections *	[out] A pointer to the caller's variable to receive the interface pointer of the newly created enumerator. The caller is responsible for releasing this pointer when it is no longer needed.

Return Value	**Meaning**
S_OK	Success.
E_POINTER	The address in *ppEnum* is not valid (such as NULL)
E_NOTIMPL	The connection point does not support enumeration.
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There was not enough memory to create the enumerator.

1.2The *IConnectionPointContainer* Interface

When implemented on an object, makes the object "connectable" and expresses the existence of outgoing interfaces on the object. Through this interface a client may either locate a specific "connection point" for one IID or it can enumerate the connections points that exist.

IDL:

```
[
uuid(B196B284-BAB4-101A-B69C-00AA00341D07)
   , object, pointer_default(unique)
]
interface IConnectionPointContainer : IUnknown
   {
   HRESULT EnumConnectionPoints([out] IEnumConnectionPoints **ppEnum);
   HRESULT FindConnectionPoint([in] REFIID riid
      , [out] IConnectionPoint **ppCP);
   }
```

1.2.1IConnectionPointContainer::EnumConnectionPoints

HRESULT IConnectionPointContainer::EnumConnectionPoints([out] IEnumConnectionPoints **ppEnum);

Creates an enumerator of all the connection points supported in the connectable object, one connection point per IID. Since IEnumConnectionPoints enumerates IConnectionPoint* types, the caller must use IConnectionPoint::GetConnectionInterface to determine the actual IID that the connection point supports.

The caller of this member must call (*ppEnum)->Release when the enumerator object is no longer needed.

Argument	Type	Description
ppEnum	IEnumConnectionPoints *	[out] A pointer to the caller's variable that is to receive the interface pointer to the enumerator. The caller is responsible for releasing this pointer after this function returns successfully.

Return Value	**Meaning**
S_OK	The enumerator was created successfully.
E_UNEXPECTED	An unknown error occurred.
E_POINTER	The value passed in ppEnum is not valid (such as NULL).
E_OUTOFMEMORY	There was not enough memory to create the enumerator object.

Comments:

E_NOTIMPL is specifically disallowed because outside of type information there would be no other means through which a caller could find the IIDs of the outgoing interfaces.

1.2.2IConnectionPointContainer::FindConnectionPoint

HRESULT FindConnectionPoint([in] REFIID riid , [out] IConnectionPoint **ppCP);

Asks the "connectable object" if it has a connection point for a particular IID, and if so, returns the IConnectionPoint interface pointer to that connection point. Upon successful return, the caller must call IConnectionPoint::Release when that connection point is no longer needed.

Note that this function is the QueryInterface equivalent for an object's outgoing interfaces, where the outgoing interface is specified with riid and where the interface pointer returned is always that of a connection point.

Argument	Type	Description
riid	REFIID	[in] A reference to the outgoing interface IID whose connection point is being requested.
ppCP	IConnectionPoint **	[out] The address of the caller's variable that is to receive the IConnectionPoint interface pointer to the connection point that manages the outgoing interface identified with riid. This is set to NULL on failure of the call; otherwise the caller must call IConnectionPoint::Release when the connection point is no longer needed.

Return Value	**Meaning**
S_OK	The call succeeded and *ppCP has a valid interface pointer.
E_POINTER	The address passed in ppCP is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There was not enough memory to carry out the operation, such as not being able to create a new connection point object.
CONNECT_E_NOCONNECTION	This connectable object does not support the outgoing interface specified by riid.

Comments:

E_NOTIMPL is not allowed as a return code for this member. Any implementation of *IConnectionPointContainer* must implement this method.

1.3The *IEnumConnectionPoints* Interface

A connectable object can be asked to enumerate its supported connection points–in essence, it's outgoing interfaces–through IConnectionPointContainer::EnumConnectionPoints. The resulting enumerator returned from this member implements the interface IEnumConnectionPoints through which a client can access all the individual connection point sub-objects supported within the connectable object itself, where each connection point, of course, implements IConnectionPoint.

Therefore IEnumConnectionPoints is a standard enumerator interface typed for IConnectionPoint*.

IDL:

```
[
uuid(B196B285-BAB4-101A-B69C-00AA00341D07)
   , object, pointer_default(unique)
]
interface IEnumConnectionPoints : IUnknown
   {
   HRESULT Next([in] ULONG cConnections
      , [out, max_is(cConnections)] IConnectionPoint **rgpcn
```

```
    , [out] ULONG *pcFetched);

HRESULT Skip([in] ULONG cConnections);
HRESULT Reset(void);
HRESULT Clone([out] IEnumConnectionPoints **ppEnum);
}
```

1.3.1IEnumConnectionPoints::Next

HRESULT IEnumConnectionPoints::Next([in] ULONG cConnections , [out, max_is(cConnections)]
IConnectionPoint **rgpcn, [out] ULONG *pcFetched);

Enumerates the next cConnections elements in the enumerator's list, returning them in rgpcn along with the actual number of enumerated elements in pcFetched. The caller is responsible for calling IConnectionPoint::Release through each pointer returned in rgpcn.

Argument	Type	Description
cConnections	ULONG	[in] Specfies the number of IConnectionPoint * values to return in the array pointed to by rgpcn. This argument must be 1 if pcFetched is NULL.
rgpcn	IConnectionPoint **	[out] A pointer to a caller-allocated IConnectionPoint * array of size cConnections in which to return the enumerated connection points. The caller is responsible for calling IConnectionPoint::Release through each pointer enumerated into the array once this method returns successfully. If cConnections is greater than one the caller must also pass a non-NULL pointer passed to pcFetched to know how many pointers to release.
pcFetched	ULONG	[out] A pointer to the variable to receive the actual number of connection points enumerated in *rgpcn*. This argument can be NULL in which case the *cConnections* argument must be 1.

Return Value	**Meaning**
S_OK	The requested number of elements has been returned and *pcFetched (if non-NULL) is set to cConnections if
S_FALSE	The enumerator returned fewer elements than cConnections because there were not that many elements left in the list.. In this case, unused elements in rgpcn in the enumeration are not set to NULL and *pcFetched holds the number of valid entries, even if zero is returned.
E_POINTER	The address in rgpcn is not valid (such as NULL)
E_INVALIDARG	The value of *cConnections* is not 1 when pcFetched is NULL; or the value of cConnections is zero.
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to enumerate the elements.

Comments:

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.

1.3.2IEnumConnectionPoints::Skip

HRESULT IEnumConnectionPoints::Skip([in] ULONG cConnections);

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

Argument	Type	Description
cConnections	ULONG	[in] Specifies the number of elements to skip in the enumeration.

Return Value	**Meaning**
S_OK	The number of elements skipped is cConnections.
S_FALSE	The enumerator skipped fewer than cConnections because there were not that many left in the list. The enumerator will, at this point, be positioned at the end of the list such that subsequent calls to Next (without an intervening Reset) will return zero elements.
E_INVALIDARG	The value of *cConnections* is zero, which is not valid.
E_UNEXPECTED	An unknown error occurred.

1.3.3IEnumConnectionPoints::Reset

HRESULT IEnumConnectionPoints::Reset(void);

Instructs the enumerator to position itself back to the beginning of the list of elements.

Argument	Type	Description
none		

Return Value	**Meaning**
S_OK	The enumerator was successfully reset to the beginning of the list.
S_FALSE	The enumerator was not reset to the beginning of the list.
E_UNEXPECTED	An unknown error occurred.

Comments:

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 this condition.

1.3.4IEnumConnectionPoints::Clone

HRESULT IEnumConnectionPoints::Clone([out] IEnumConnectionPoints **ppEnum);

Creates another connection point enumerator with the same state as the current enumerator, which iterates over the same list. This makes it possible to record a point in the enumeration sequence in order to return to that point at a later time.

Argument	Type	Description
ppEnum	IEnumConnectionPoints**	[out] The address of the variable to receive the IEnumConnectionPoints interface pointer to the newly created enumerator. The caller must release this new enumerator separately from the first enumerator.

Return Value	Meaning
S_OK	Clone creation succeeded.
E_NOTIMPL	Cloning is not supported for this enumerator.
E_POINTER	The address in ppEnum is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to create the clone enumerator.

1.4The *IEnumConnections* Interface

Any individual connection point can support enumeration of its known connections through IConnectionPoint::EnumConnections. The enumerator created by this function implements the interface IEnumConnections which deals with the type CONNECTDATA. Each CONNECTDATA structure contains the the IUnknown * of a connected sink and the dwCookie that was returned by IConnectionPoint::Advise when that sink was connected. When enumerating connections through IEnumConnections, the enumerator is responsible for calling IUnknown::AddRef through the pointer in each enumerated structure, and the caller is responsible to later call IUnknown::Release when those pointers are no longer needed.

IDL:

```
[
uuid(B196B287-BAB4-101A-B69C-00AA00341D07)
    , object, pointer_default(unique)
]
interface IEnumConnections : IUnknown
    {
    typedef struct tagCONNECTDATA
        {
        IUnknown   *pUnk;
        DWORD      dwCookie;
        } CONNECTDATA;

    typedef struct tagCONNECTDATA *PCONNECTDATA;
    typedef struct tagCONNECTDATA *LPCONNECTDATA;

    HRESULT Next([in] ULONG cConnections
        , [out, max_is(cConnections)] CONNECTDATA *rgpcd
        , [out] ULONG *pcFetched);

    HRESULT Skip([in] ULONG cConnections);
    HRESULT Reset(void);
    HRESULT Clone([out] IEnumConnections **ppEnum);
    }
```

1.4.1IEnumConnections::Next

HRESULT IEnumConnections::Next([in] ULONG cConnections ,
                [out, max_is(cConnections)] CONNECTDATA *rgpcd,
                [out] ULONG *pcFetched);

Enumerates the next cConnections elements 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 IUnknown::Release through each pUnk pointer returned in the structure elements of rgpcd.

Argument	Type	Description
cConnections	ULONG	[in] Specfies the number of CONNECTDATA structures to return in the array pointed to by rgpcd. This argument must be 1 if pcFetched is NULL.
rgpcd	CONNECTDATA *	[out] A pointer to a caller-allocated CONNECTDATA array of size cConnections in which to return the enumerated connections. 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 passed to pcFetched to know how many pointers to release.
pcFetched	ULONG	[out] A pointer to the variable to receive the actual number of connections enumerated in rgpcd. This argument can be NULL in which case the cConnections argument must be 1.

Return Value	**Meaning**
S_OK	The requested number of elements has been returned and *pcFetched (if non-NULL) is set to *cConnections* if
S_FALSE	The enumerator returned fewer elements than cConnections because there were not that many elements left in the list.. In this case, unused elements in *rgpcd* in the enumeration are not set to NULL and *pcFetched holds the number of valid entries, even if zero is returned.
E_POINTER	The address in rgpcd is not valid (such as NULL).
E_INVALIDARG	The value of cConnections is not 1 when pcFetched is NULL; or the value of cConnections is zero.
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to enumerate the elements.

Comments:

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.

1.4.2IEnumConnections::Skip

HRESULT IEnumConnections::Skip([in] ULONG cConnections);

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

Argument	Type	Description
cConnections	ULONG	[in] Specifies the number of elements to skip in the enumeration.

Return Value	**Meaning**
S_OK	The number of elements skipped is cConnections.

S_FALSE	The enumerator skipped fewer than cConnections because there were not that many left in the list. The enumerator will, at this point, be positioned at the end of the list such that subsequent calls to Next (without an intervening Reset) will return zero elements.
E_INVALIDARG	The value in cConnections is zero which is not valid.
E_UNEXPECTED	An unknown error occurred.

1.4.3IEnumConnections::Reset

HRESULT IEnumConnections::Reset(void);

Instructs the enumerator to position itself back to the beginning of the list of elements.

Argument	Type	Description
none		

Return Value	**Meaning**
S_OK	The enumerator was successfully reset to the beginning of the list.
S_FALSE	The enumerator was not reset to the beginning of the list.
E_UNEXPECTED	An unknown error occurred.

Comments:

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 this condition.

1.4.4IEnumConnections::Clone

HRESULT IEnumConnections::Clone([out] IEnumConnections **ppEnum);

Creates another connections enumerator with the same state as the current enumerator, which iterates over the same list. This makes it possible to record a point in the enumeration sequence in order to return to that point at a later time.

Argument	Type	Description
ppEnum	IEnumConnections**	[out] The address of the variable to receive the IEnumConnections interface pointer to the newly created enumerator. The caller must release this new enumerator separately from the first enumerator.

Return Value	**Meaning**
S_OK	Clone creation succeeded.
E_NOTIMPL	Cloning is not supported for this enumerator.
E_POINTER	The address in ppEnum is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to create the clone enumerator.