How to receive data dragged from other applications (part 2 of 6)

The IDropTarget and IDataObject interfaces

About *IDropTarget*

IDropTarget descends directly from *IUnknown* and implements four additional methods. The definition of the interface is found in Delphi's `ActiveX` unit and is reproduced in *Listing 1*.

```
type
  IDropTarget = interface(IUnknown)
    ['{00000122-0000-0000-C000-000000000046}']
    function DragEnter(const dataObj: IDataObject; grfKeyState: Longint;
      pt: TPoint; var dwEffect: Longint): HResult;
      stdcall;
    function DragOver(grfKeyState: Longint; pt: TPoint;
      var dwEffect: Longint): HResult;
      stdcall;
    function DragLeave: HResult;
      stdcall;
    function Drop(const dataObj: IDataObject; grfKeyState: Longint;
      pt: TPoint; var dwEffect: Longint): HResult;
      stdcall;
  end;
```

Listing 1

Table 1 provides a description of the methods and how we use them. Further information can be found in Delphi's Windows API help file.

IDropTarget methods	
DragEnter	Called when the user first drags an object over the window that is registered for drag and drop. Parameters are *dataObj* Describes the object being dragged. We will examine this object in detail later. *grfKeyState* Bit mask describing which mouse buttons and "shift" (modifier) keys are pressed. We may want to use this information to decide how to handle the operation and to help decide the value assigned to *dwEffect* (see below). *pt* The mouse location. *dwEffect* Describes the cursor effect used for the operation. When the method is called it contains a bit mask detailing the permitted effects and must be set to a single value decribing the desired effect within the method. Effects are: the "no entry" cursor (meaning we can't accept a drop), the copy cursor, the move cursor or the shortcut cursor.
DragOver	Called repeatedly as the user moves the mouse over our window. The parameters are the same as for *DragEnter* except that *dataObj* is not provided here. We use this method to modify the the cursor according to the mouse position and any change in the mouse button or modifier keys.
DragLeave	Called to inform us that the user has dragged the object off our window without dropping the object. We use this to tidy up if necessary.
Drop	Called if the user drops the object over our window. The parameters are exactly the same as for *DragEnter*. When this method is called we process the dropped object and tidy up. Note that

	DragLeave is not called if the object is dropped. *Drop* is not called if the "no entry" cursor has been requested.

Table 1

When implementing *IDropTarget* we may need to examine the data object, the permitted drop effects and the modifier keys to answer the following questions:

- Can we accept the object?
- How should we handle the object if dropped?

We will answer these questions later in the article when we look at implementing *IDropTarget*.

About *IDataObject*

Again the declaration of *IDataObject* is provided in the `ActiveX` unit. *Listing 2* replicates the definition.

```
type
  IDataObject = interface(IUnknown)
    ['{0000010E-0000-0000-C000-000000000046}']
    function GetData(const formatetcIn: TFormatEtc;
      out medium: TStgMedium): HResult;
      stdcall;
    function GetDataHere(const formatetc: TFormatEtc;
      out medium: TStgMedium): HResult;
      stdcall;
    function QueryGetData(const formatetc: TFormatEtc): HResult;
      stdcall;
    function GetCanonicalFormatEtc(const formatetc: TFormatEtc;
      out formatetcOut: TFormatEtc): HResult;
      stdcall;
    function SetData(const formatetc: TFormatEtc;
      var medium: TStgMedium; fRelease: BOOL): HResult;
      stdcall;
    function EnumFormatEtc(dwDirection: Longint; out enumFormatEtc:
      IEnumFormatEtc): HResult;
      stdcall;
    function DAdvise(const formatetc: TFormatEtc;
      advf: Longint; const advSink: IAdviseSink;
      out dwConnection: Longint): HResult;
      stdcall;
    function DUnadvise(dwConnection: Longint): HResult;
      stdcall;
    function EnumDAdvise(out enumAdvise: IEnumStatData): HResult;
      stdcall;
  end;
```

Listing 2

IDataObject is supported by data objects dragged and dropped over our window. We don't need to implement this interface – Windows provides an instance to us via the methods of *IDropTarget*. Not all the methods are required when handling drag and drop. The only ones of interest to us in this article are:

Required IDropTarget Methods	
GetData	Unsurprisingly this method retrieves data from the drop object. We specify a desired data format, medium and target device in the method's *formatetcIn* parameter and, if the object supports it, the method returns the required data encapsulated in a storage medium via its *medium* parameter. We will discuss storage media in detail later.
QueryGetData	Checks whether the data object can provide data in the required format.
EnumFormatEtc	Returns an enumerator that can iterate through the various data formats, media and target devices supported by the data object.

Table 2

The *GetDataHere* and *GetCanonicalFormatEtc* methods could also be of use but are not considered in this article.

As has been observed, data objects can store different versions of the same data in various formats. It is important that we have a reasonable understanding of them. In the *next section* we examine data formats in more detail.