How to receive data dragged from other applications (part 4 of 6)

Querying and getting data from a data object

Querying the Data Object

We often need to query an object to find out about the formats it supports. There are two distinct ways to do this. The first is to enquire whether the data object supports a specified format and the second is to enumerate all the supported formats. We will consider each of these in turn.

Checking for a specified data format

To check for a specific format we call the data object's *QueryGetData* method, passing it a *TFormatEtc* structure that describes the required data format. *Listing 4* defines a helper routine that finds this information for a given clipboard format and storage medium.

```
function HasFormat(const DataObj: IDataObject;
  const Fmt: TClipFormat; const Tymed: Integer): Boolean;
var
  FmtEtc: TFormatEtc;
begin
  FmtEtc.cfFormat := Fmt;
  FmtEtc.ptd := nil;                    // device independent
  FmtEtc.dwAspect := DVASPECT_CONTENT;  // full detail
  FmtEtc.lindex := -1;                  // all the data
  FmtEtc.tymed := Tymed;
  Result := DataObj.QueryGetData(FmtEtc) = S_OK;
end;
```

Listing 4

Here we pass a *IDataObject* instance to the helper function along with the required clipboard format and storage medium type. We record the clipboard format and storage medium type in a *TFormatEtc* structure and set its other fields to default values. Once we have set up the *TFormatEtc* structure we pass it to the data object's *QueryGetData* method and check the return value. A return of *S_OK* indicates the format is supported.

Enumerating all data formats

IDataObject can provide a standard Windows enumerator that iterates through all data formats supported by a data object. The skeleton code in *Listing 5* show how to use the enumeration.

```
procedure EnumDataFormats(const DataObj: IDataObject);
var
  Enum: IEnumFormatEtc;
  FormatEtc: TFormatEtc;
begin
  OleCheck(DataObj.EnumFormatEtc(DATADIR_GET, Enum));
  while Enum.Next(1, FormatEtc, nil) = S_OK do
  begin
    // Do something with FormatEtc here
    ...
  end;
end;
```

Listing 5

We first call the *EnumFormatEtc* method of the data object to get the required enumerator. We pass the *DATADIR_GET* flag to the method to indicate that we want to know about data formats we can read from the

data object (rather the formats we could write to the object). If all goes well the method passes the enumerator out via the *Enum* parameter. If the call fails the *OleCheck* traps the error return and raises an exception.

Once we have the enumerator we repeatedly call its *Next* method passing `1` as the first parameter to indicate we want one *TFormatEtc* structure at each iteration. For as long as there is more information available the method sets *FormatEtc* and returns *S_OK*. When the data formats are exhausted the method returns *S_FALSE* and the loop ends. Within the **while** loop we can do something with data formats, such as list them in a dialog box.

Example code

One of the examples in this article's *Demo Code* uses this technique to list data formats in a list view.

Getting Data from the Data Object

Once we know the data format we want, the easiest way to get data from the data object is via its *GetData* method. The skeleton code presented in *Listing 6* shows how to approach this.

```
procedure GetDataFromObj(const DataObj: IDataObject;
  const Fmt: TClipFormat; const Tymed: Integer);
var
  FmtEtc: TFormatEtc;
  Medium: TStgMedium;
begin
  FmtEtc.cfFormat := Fmt;
  FmtEtc.ptd := nil;
  FmtEtc.dwAspect := DVASPECT_CONTENT;
  FmtEtc.lindex := -1;
  FmtEtc.tymed := Tymed;
  OleCheck(DataObj.GetData(FmtEtc, Medium));
  try
    Assert(Medium.tymed = FmtEtc.tymed);
    // Do something with the data from Medium
    ...
  finally
    ReleaseStgMedium(Medium);
  end;
end;
```

Listing 6

Just like in *Listing 4* we must fill in the *TFormatEtc* structure with details of the required data format as specified by the *Fmt* and *Tymed* parameters. Next we call the data object's *GetData* method passing in the data format structure. If the method succeeds it returns S_OK and sets *Medium* to reference the required storage medium. *Medium* should have the same value in its *tymed* field as *FmtEtc* so we use an assertion to check this. If *GetData* fails it causes *OleCheck* to raise an exception.

Now we have the storage medium we can then do something with it, for example display the information it contains. Once we have finished with the storage medium we must release its data structures. How this is done varies depending on the medium type. Fortunately Windows provides the *ReleaseStgMedium* method that knows how to free the required data structures, so we simply call that routine.

Now it's all very well saying "do something with the data from Medium", but what exactly is that "something"? Well, the answer depends on both the data format and the media type. So next we'll look at a few examples of working with different data formats and data types.

Example 1: Text stored in global memory

Several data formats provide their data as ANSI text stored in global memory. Examples are:

- *CF_TEXT* – a standard format.
- *CF_FILENAMEA* – a format defined by the shell for passing filenames.
- Rich Text Format
- HTML Format

Listing 7 shows an example of how to retrieve plain text data from global memory. The function can be passed any clipboard format that uses plain text data and will return the text as a string.

```
function GetTextFromDataObj(const DataObj: IDataObject;
  const Fmt: TClipFormat): string;
var
  FormatEtc: TFormatEtc;
  Medium: TStgMedium;
  PText: PChar;
begin
  FormatEtc.cfFormat := Fmt;
  FormatEtc.ptd := nil;
  FormatEtc.dwAspect := DVASPECT_CONTENT;
  FormatEtc.lindex := -1;
  FormatEtc.tymed := TYMED_HGLOBAL;
  OleCheck(DataObj.GetData(FormatEtc, Medium));
  try
    Assert(Medium.tymed = TYMED_HGLOBAL);
    PText := GlobalLock(Medium.hGlobal);
    try
      Result := PText;
    finally
      GlobalUnlock(Medium.hGlobal);
    end;
  finally
    ReleaseStgMedium(Medium);
  end;
end;
```

Listing 7

The routine starts the same as *Listing 6* except that we hard wire the *TYMED_HGLOBAL* media type. The code that retrieves the text begins after the *Assert* statement. We lock the global memory handle (accessed via a field of the *TStgMedium* structure returned by *GetData* in *Medium*) and store the resulting pointer in *PText*. *PText* now points to the start of a zero termimated string of ANSI characters, so we simply set the function result to that string. Finally we unlock the memory handle and then call *ReleaseStgMedium*, which in turn frees the storage medium's memory.

Example 2: File list stored in global memory

In *article #11* we noted that the list of files dropped on a window was accessed via a *HDROP* handle and we used the *DragQueryXXX* API functions to get at the list of files. Well, the same principle applies for OLE drag and drop. The related clipboard format is *CF_HDROP* and the medium type is *TYMED_HGLOBAL*. The drop handle is stored in the *TStgMedium.hGlobal* field. *Listing 8* is based on *article #11's listing 7*, but it gets its drop handle from the data object.

```
procedure GetFileListFromObj(const DataObj: IDataObject;
  const FileList: TStrings);
var
  FmtEtc: TFormatEtc;         // specifies required data format
  Medium: TStgMedium;         // storage medium containing file list
  DroppedFileCount: Integer;  // number of dropped files
  I: Integer;                 // loops thru dropped files
  FileNameLength: Integer;    // length of a dropped file name
  FileName: string;           // name of a dropped file
begin
  // Get required storage medium from data object
  FmtEtc.cfFormat := CF_HDROP;
  FmtEtc.ptd := nil;
  FmtEtc.dwAspect := DVASPECT_CONTENT;
  FmtEtc.lindex := -1;
  FmtEtc.tymed := TYMED_HGLOBAL;
  OleCheck(DataObj.GetData(FmtEtc, Medium));
  try
    try
      // Get count of files dropped
      DroppedFileCount := DragQueryFile(
        Medium.hGlobal, $FFFFFFFF, nil, 0
      );
      // Get name of each file dropped and process it
      for I := 0 to Pred(DroppedFileCount) do
      begin
        // get length of file name, then name itself
```

```
        FileNameLength := DragQueryFile(Medium.hGlobal, I, nil, 0);
        SetLength(FileName, FileNameLength);
        DragQueryFile(
          Medium.hGlobal, I, PChar(FileName), FileNameLength + 1
        );
        // add file name to list
        FileList.Add(FileName);
      end;
    finally
      // Tidy up - release the drop handle
      // don't use DropH again after this
      DragFinish(Medium.hGlobal);
    end;
  finally
    ReleaseStgMedium(Medium);
  end;
end;
```

Listing 8

This routine gets a list of dropped files from the the provided data object and stores the file names in the routine's *FileList* string list parameter. It begins, as usual by populating a *TFormatEtc* with the required information (this time requesting the *CF_HDROP* format and *TYMED_HGLOBAL* storage medium). It then gets the data from the data object.

We pass the value of the storage medium's *hGlobal* field to *DragQueryFile* to get the number of dropped files. Next we loop through all the dropped files getting the name of each file name by making two more calls to the ever so flexible *DragQueryFile* function. The first call gets the length of the filename, to enable us to allocate a string of the required length. The second call reads the file name into the string. The file name is then added to the list. When the loop finishes a call to *DragFinish* finalises the drop handle. Finally, we release the storage medium's memory by means of the now familiar call to *ReleaseStgMedium*.

Example 3: Data stored in a stream

Most data objects you will encounter when handling drag and drop will make their data available through a memory handle accessed via *TStgMedium*'s *hGlobal* field. However you will occasionally come across the other storage media types. We will take a very brief look at just one more here – data streams accessed via the *IStream* interface.

Get the the required medium in the usual way, but specify *TYMED_ISTREAM* in *TFormatEtc*'s *tymed* field. Obtain the medium as normal by calling the data object's *GetData* method. Once you have the storage medium cast it's *pstm* field to *IStream* and use the methods of *IStream* to manipulate the data.

Now we have an understanding of how to manipulate data objects we are at long last in a position to look at how to implement the *IDropTarget* interface. We do this in the *next section*.