September 3, 1993

Direct Manipulation

The User Interface Class Library provides four types of objects to support direct manipulation:

An event handler (IDMSourceHandler or IDMTargetHandler)

A renderer (IDMSourceRenderer or IDMTargetRenderer)

A drag item (IDMItem)

A drag item provider (IDMItemProvider)

IDMSourceHandler and IDMTargetHandler are derived from IHandler. They handle the Presentation Manager direct manipulation window messages. Objects from these classes pick off the WM_* and DM_* messages for the source and target windows and translate them to virtual function calls to the handler.

In addition to translating messages to virtual function calls, these handlers also manage the second type of objects, the renderers. Renderers transfer the representation of the object being manipulated between the source and target windows. Direct manipulation renderers are objects of classes IDMSourceRenderer and IDMTargetRenderer.

The third type of objects are the drag items, represented by objects of class IDMItem. These objects encapsulate the logic that serves as the bridge between the context-insensitive handlers and renderers and the application-specific behavior of particular source and target windows. Put another way, the drag items provide the application-specific semantics of the direct manipulation operation. For example, the handler and renderer classes can manipulate IString objects, but the entry field items know how to extract the source IString from the source entry field and convert it to a customer object in the target window.

Objects of the IDMItemProvider class allow generic controls like an entry field to generate context-sensitive drag items; for example, an entry field that contains customer names can generate a "customer" item; a bit map can provide an item that can extract the picture from a .bmp file.

User Interface Class Library provides direct manipulation for:

Entry field
Multiple-line edit

Enabling Direct Manipulation

To enable direct manipulation for an entry field or multiple-line edit control, create instances of IDMSourceHandler and IDMTargetHandler and attach each to a control.

The following sample code enables direct manipulation of text between two entry fields in the same process.

```
:
 9 void main()
1  {
11    IFrameWindow
12       frame( "ICLUI Direct Manipulation Sample 1" );
13
14    IEntryField                              //Create window with two
15       client(  , &frame, &frame ),    //entry fields, client and ext.
16       ext      (  , &frame, &frame );
17
18    IDMSourceHandler                          //Define a source handler for
19       srcHandler( &client );              //the entry field client.
2     IDMTargetHandler                          //Define a target handler for
21       tgtHandler( &client );              //the entry field client.
22
23    ext.setItemProvider(client.itemProvider(-));//Add drag item
24                                                        //provider for ext.
25    srcHandler.handleEventsFor(&ext);//Add source handler for ext.
26    tgtHandler.handleEventsFor(&ext);//Add target handler for ext.
27
28    frame
29       .setClient( &client )
3        .addExtension( &ext, IFrameWindow::belowClient,  .5 )
31       .setFocus()
32       .show();
33
34    IApplication::current().run();
35
36 }
```

Lines 18 and 19 create an instance of IDMSourceHandler and attach it to the **client** entry field. IDMSourceHandler creates a drag item provider for the entry field if one has not already been created.

Lines 20 and 21 create an instance of IDMTargetHandler and attach it to the **client** entry field.

Line 23 attaches the drag item provider to the **ext** entry field. The drag item provider was created by the IDMSourceHandler constructor that was invoked on lines 18 and 19.

Lines 25 and 26 attach the source and target handlers that were created on lines 18 through 20 to the **ext** entry field.

Adding Direct Manipulation to an Object

To add direct manipulation to other controls, you must specifically create the handlers, renderers, and item providers that IBM Class Library generates automatically for entry field and multiple-line edit controls. You must:

Add IDMItem as the base class for the current application object and override the member function dropped.

Write a drag item provider for the customized object using IDMItemProvider.

Create the drag item providers, the renderers, and the handlers for the customized object.

The following example adds drop support to a bit-map control. The header file defines two classes, ABitmapItem and ABitmapProvider, and overrides the member functions dropped and provideTargetItemFor. The .cpp file adds the drag item provider, the target handler, and the target renderer.

```
 1 #include <idmprov.hpp>
 2
 3 #include <idmitem.hpp>
 4
 5 class ABitmapItem : public IDMItem {
 6 public:
 7     ABitmapItem ( const IDMItemHandle &item );
 8
 9 virtual Boolean
10      dropped ( IWindow  target, IDMTargetDropEvent & );
11 };
12
13 class ABitmapProvider : public IDMItemProvider {
14 public:
15 virtual IDMItemHandle
16     provideTargetItemFor ( const IDMItemHandle item );
17 };
```

Lines 5 through 11 declare IDMItem as the base class for objects of a specialized class named ABitmapItem. Objects of this class provide bit-map control drop behavior when a source bit-map file is dropped on a bit-map control that is properly configured with a target handler and an ABitmapProvider.

Lines 12 through 17 define a drag item provider for a bit-map control and override provideTargetItemFor so it returns an ABitmapItem to replace the argument item.

```
:
19 void main()
2    {
21    IFrameWindow
22        frame ( "C Set ++ Direct Manipulation - Sample 2" );
23
24    IBitmapControl                  // Create an empty bit-map control.
25        bmpControl (  , &frame, &frame );
26
27    IDMTargetHandler                // Create target handler
28        handler;                    // for the bit-map control.
29
3     handler.handleEventsFor( &bmpControl );// Add target handler on bit-map control.
31
32    ABitmapProvider                 // Create a bit-map drag item provider.
33        itemProvider;
34
35   bmpControl.setDragItemProvider( &itemProvider );// Attach provider to the bit-map control.
36
37   IDMTargetRenderer               //Create renderer to render items.
38       renderer;
39
4     renderer.setSupportedRMFs( "<DRM_OS2FILE,DRF_TEXT>" )// Set up renderer
41                .setSupportedTypes( "Bitmap" );                    // to accept .bmp files.
42
43    handler.addRenderer( &renderer );
44
45    bmpControl.setText( "Drop .bmp files here." );// Set the bit-map control
46    frame.setClient( &bmpControl )                  // as the frame's client and
47           .showModally();                            // display the frame.
48    }
```

First, the .cpp file creates an empty bit-map control object called **bmpControl** and then creates and attaches the handler, provider, and renderer.

Lines 24 and 25 create the bit-map control object.

Lines 27 and 28 construct a target handler without passing a parameter. The constructor for IDMTargetHandler accepts one parameter, a pointer to an instance of IEntryField or IMultiLineEdit or NULL. The default is NULL.

Line 30 attaches the target handler created on line 27 to **bmpControl**.

Lines 32 and 33 construct a drag item provider named **itemprovider**.

Line 35 attaches the drag item provider to **bmpControl**.

Line 37 constructs a renderer.

Lines 40 and 41 define the rendering mechanisms and formats (RMFs) and the type that the renderer will support.

Line 43 attaches the renderer to the target handler.

```
49
5  ABitmapItem :: ABitmapItem ( const IDMItemHandle &item )
51     : IDMItem(  item )
52     {
53     }
54
55 Boolean ABitmapItem :: dropped ( IWindow  target, IDMTargetDropEvent & )
56     {
57     IBitmapControl                                          // Get pointer to target bitmap control.
58        bmpControl = (IBitmapControl )target;
59
6      IString                                                 // Construct dropped .bmp file name from this item.
61        fname = this->containerName() + "\\" + this->sourceName();
62
63     struct stat                                             // Get file size.
64        buf;
65     stat( fname, &buf);
66
67     FILE                                                    // Open and read the file.
68        fileptr = fopen( fname, "rb" );
69     char
7         buffer = new char[ buf.st_size ];
71     fread( buffer, sizeof(char), buf.st_size, fileptr );
72
73     BITMAPARRAYFILEHEADER2                                  // Construct the bitmap from the file.
74        array = ( BITMAPARRAYFILEHEADER2   )buffer;
75     BITMAPFILEHEADER2
76        header;
77
78     if ( array->usType == BFT_BITMAPARRAY ) {       // First, see if file holds array of bitmaps.
79        header = &array->bfh2;                       // It is, point to first file header in array.
8      } else {
81        header = ( BITMAPFILEHEADER2   )buffer;      // It isn't, point to file header at start of file.
82     }
83     if ( header->usType == BFT_BMAP) {              // Now check to see if this is a bitmap.
84
85        IPresSpaceHandle                             // We can proceed, first get a presentation space.
86           hps = bmpControl -> presSpace();
87
88        if ( hps ) {
89           IBitmapHandle                             // Now create the bit map from the file contents.
9             hbm = GpiCreateBitmap( hps,
91                                    &header->bmp2,
92                                    CBM_INIT,
93                                    buffer + header->offBits,
94                                    (BITMAPINFO2 )&header->bmp2 );
95           if ( hbm ) {
96              IBitmapHandle                          // Get previously dropped bit map.
97                 old = bmpControl -> bitmap();
98
99              bmpControl -> setBitmap( hbm );      // Set new one.
1
```

```
1 1              GpiDeleteBitmap( old );                    // Destroy old because we no longer need it.
1 2
1 3              bmpControl -> setText( fname );     // Indicate name of dropped file.
1 4            } else {
1 5              bmpControl -> setText( "Couldn't create bit map!" );
1 6            }
1 7            bmpControl -> releasePresSpace(hps);// Release the presentation space.
1 8          } else {
1 9            bmpControl -> setText ( "Couldn't get PS!" );
11           }
111        } else {
112          bmpControl -> setText( fname + " isn't a bit map!" );
113      }
114
115       delete [] buffer;                                  // Free buffer.
116
117      return  true;
118      }
119
12  IDMItemHandle ABitmapProvider :: provideTargetItemFor ( const IDMItemHandle item )
121      {
122       return new ABitmapItem ( item );
123      }
```

The rest of the .cpp file defines the overridden member functions, dropped and provideTargetItemFor, for the classes that were declared in the .hpp file.

Lines 50 through 118 define dropped. This member function gets the dropped file, creates the bitmap, and displays the bitmap in the new space.

Lines 120 through 122 use provideTargetItemFor to replace the generic IDMItem that was created during the initial target enter event (IDMTargetEnterEvent) with the derived class. This function is called when a drop event (IDMTargetDropEvent) occurs on a target window.