IBM ILOG Views Foundation V5.3 Tutorial

June 2009

Copyright notice

Table of Contents

About This Tutorial

This tutorial explains how to create an ActiveX control from existing IBM® ILOG® Views code and how to add ActiveX controls as graphic objects into an IBM ILOG Views application.

What You Need to Know

This manual assumes that you are familiar with the PC or UNIX® environment in which you are going to use IBM® ILOG® Views, including its particular windowing system. Since IBM ILOG Views is written for C++ developers, the documentation also assumes that you can write C++ code and that you are familiar with your C++ development environment so as to manipulate files and directories, use a text editor, and compile and run C++ programs.

Notation

Typographic Conventions

The following typographic conventions apply throughout this manual:

◆ Code extracts and file names are written in a `"code"` typeface.

- Entries to be made by the user, such as in dialog boxes, are written in a `"code"` typeface.
- Command variables to be supplied by the user are written in *italics*.
- Some words in *italics*, when seen for the first time, may be found in the glossary.

Naming Conventions

Throughout the documentation, the following naming conventions apply to the API.

- The names of types, classes, functions, and macros defined in the IBM ILOG Views library begin with `Ilv`, for example `IlvGraphic`.
- The names of types and macros not specific to IBM ILOG Views begin with `Il`, for example `IlBoolean`.
- The names of classes as well as global functions are written as concatenated words with each initial letter capitalized.

  class `IlvDrawingView`;
- The names of virtual and regular methods begin with a lowercase letter; the names of static methods start with an uppercase letter. For example:

  `virtual IlvClassInfo*` **getClassInfo**`() const;`

  `static IlvClassInfo*` **ClassInfo***`() const;`

C H A P T E R

1

Creating an ActiveX Control from Existing IBM ILOG Views Code

You can create an ActiveX control using IBM® ILOG® Views either with the Microsoft Foundation Classes (MFC) library or with the Active Template Library (ATL).

The following two tutorials demonstrate how to create an ActiveX control from an existing IBM ILOG Views program.

- Creating an ActiveX Control with the ATL Library
- Creating an ActiveX Control with the MFC Library

You will build the ActiveX control from the sample IBM ILOG Views program provided in the following file using the ATL COM AppWizard or the MFC ActiveX ControlWizard, depending on the library you choose: `manager.cpp`

This program displays a map in a manager and allows you to zoom in or out and scroll the map.

You can also create an ActiveX from scratch. The procedure is the same as the one presented in this tutorial.

A knowledge of IBM ILOG Views, C++, Visual C++, COM, and the ATL or the MFC library is a prerequisite to work through these tutorials.

Creating an ActiveX Control with the ATL Library

This tutorial has four steps:

- Step 1: Preparing the Program
- Step 2: Creating an ActiveX Control with ATL
- Step 3: Adding Custom Properties to the ActiveX Control
- Step 4: Adding a Custom Event to The ActiveX Control

The basic ActiveX you will have created by the end of the second step will be enriched to integrate more functionality as you progress in this tutorial.

> ***Note:*** *The following comments in the files generated by the wizard indicate lines of code that have been modified or added specifically for IBM ILOG Views:*
>
> ```
> // Added for IBM ILOG Views
> // End Added for IBM ILOG Views
> ```

Step 1: Preparing the Program

This step prepares the sample application used as a starting point for this tutorial so that it can be transformed to an ActiveX control and used inside another application. Basically what you have to do is separate the data from the code and make it possible to create the main view either as a top view (which is the case in the original program) or as a child view which can become part of a parent window.

Also these modifications will make it easier for you to add new functionality to the ActiveX control. The modified files are in the `step1` directory.

You are going to:

- Separate data from the code and put it into an .ilv file.
- Encapsulate the initialization code in a Ctrl class.
- Create a resource file (.rc) with ilv2data.
- Modify the manager.cpp file.

Separate data from the code and put it into an .ilv file.

Data stored in `.ilv` files is easier to modify than data stored as a string in a `.cpp` file. Also, any dependency problems can be solved with the IBM ILOG Views data files packaging tool, ilv2data.

The result is in the file `step1\data\ctrl.ilv`.

Notice that we have added attachments so that the main view can be resized in a consistent way. Also, we have named objects so that they can be accessed more easily from the program. All of these enhancements were made with IBM ILOG Views Studio.

> ***Note:*** *The separation of data from the code is not really necessary in this tutorial. However it is recommended to facilitate modifications.*

Encapsulate the initialization code in a Ctrl class.

The initialization code is encapsulated in the `Ctrl` class that will be used to create the ActiveX control and interact with it.

This class is declared in the file `step1\ctrl.h` and is defined in the file `step1\ctrl.cpp`. This `cpp` file also stores the callback functions.

The `Ctrl` class is basic. It derives from `IlvGadgetContainer`. It is composed of only three constructors and four member functions, of which one is private:

- The first constructor creates the `Ctrl` object as a top view, as in the original version of the file. It is used in the main function in `step1\manager.cpp` and has the following signature:

```
Ctrl(IlvDisplay* display, const IlvRect& size, const char* filename = 0)
```

- The second constructor creates the `Ctrl` object as a child view. It has the following signature:

```
Ctrl(IlvDisplay* display,
     IlvSystemView parent,
     const IlvRect& size,
     const char* filename = 0)
```

- The third constructor creates the `Ctrl` object using the existing system view. It will be used to create the ActiveX control. It has the following signature:

```
Ctrl(IlvDisplay* display, IlvSystemView parent, const char* filename = 0)
```

- The public member functions `getManagerRectangle`, `getManagerView`, and `getManager` provide access to the objects loaded in the control. The private member function `init` is called to initialize the control, that is, read the data file, associate the callbacks with the objects, and so on. Most of the code that was originally in the member function `PopupManager` has been transferred to this class.

Create a resource file (.rc) with ilv2data.

Using `ilv2data`, we create an `.rc` file that contains the files `step1\data\ctrl.ilv` and `step1\data\browse.ilv`. This resource file could be used in this program, but we will use it later to build the ActiveX control.

For more information on `ilv2data`, see *Packaging* IBM ILOG Views *Application*s.

Modify the manager.cpp file.

The file `step1\manager.cpp` contains only the main function, which creates an instance of `IlvDisplay` and an instance of the new `Ctrl` class.

```
int
main(int argc, char* argv[])
{
    IlvDisplay* display =  new IlvDisplay("IlvDemo", "", argc, argv);
    if (!display || display->isBad()) {
        if (display)
            delete display;
        IlvFatalError("Couldn't create display");
        return 1;
    }
    Ctrl* ctrl = new Ctrl(display, IlvRect(0, 0, 400, 400));
    IlvMainLoop();
    return 0;
}
```

Step 2: Creating an ActiveX Control with ATL

This step shows how to create an ActiveX control with the `Ctrl` class defined in *Step 1: Preparing the Program*. The ActiveX control you will obtain once this step is completed has a minimal interface.

You are going to:

- ◆ Create a new project.
- ◆ Add a new ATL object.
- ◆ Add files to the project.
- ◆ Modify the file generated by the wizard.
- ◆ Add a new windows message handler to the CDemoATLCtrl class.
- ◆ Add a private member variable for CDemoATLCtrl.
- ◆ Modify the DemoATLCtrl.h file generated by the wizard.
- ◆ Modify the DemoATL.rc file.
- ◆ Modify the project settings (DemoATL.dsp).
- ◆ Build the ActiveX control.

Create a new project.

1. In Visual C++, choose New from the File menu and select ATL COM AppWizard in the Project page to begin creating a new project.
2. Type `DemoATL` in the Project name text field and click OK.

   The ATL COM AppWizard is displayed.

3. Select the Dynamic Link Library (DLL) option and leave Allow merging of proxy/stub code, Support MFC, and Support of MTS unchecked.

> ***Note:*** *These options are not required for this tutorial. But you might have to select them for your own purposes later, depending on what you want to obtain.*

4. Click Finish and OK.

Add a new ATL object.

1. Click the ClassView tab of the Project Workspace window to activate it, if necessary, and click the DemoATL classes label with the right mouse button.
2. Select New ATL Object from the menu that appears for creating an ATL object.

   An ATL Object Wizard is displayed.
3. Select Controls in Category and double-click Full Controls in Objects.
4. Type `DemoATLCtrl` in the Short Name text field.

   The other fields are automatically filled with labels. Leave these as they are.
5. Click the Attributes tab, select the Apartment Threading Model, the Dual Interface, and select Yes in Aggregation. Also check the Support Connection Points option.
6. Click the Miscellaneous tab and select Opaque, Solid Background, NormalizeDC, Insertable and Windowed Only.

   Leave the other options unchecked. No Stock Properties are declared.

> ***Note:*** *The Attributes and Miscellaneous options may be different, depending on what you want to obtain.*

Add files to the project.

1. Add the files `ctrl.cpp` and `ctrl.h` to the project.
2. Create the `DemoCtrl.h` file and add it to the project.

   This file should contain the declaration of the `DemoCtrl` class that follows. This class initializes the control. It also provides an interface between the code generated by the wizard and the IBM ILOG Views code and/or the `Ctrl` class.

```
class DemoCtrl : public Ctrl
{
public:
    DemoCtrl(HWND parent);
    ~DemoCtrl();
    static IlvDisplay* getDisplay()throw() { return _Display; }
    static bool InitDisplay(HINSTANCE hInstance) throw();
    static void CleanDisplay();
private:
    DemoCtrl(const DemoCtrl&); // Not defined to avoid
                               // default copy constructor.
    static IlvDisplay* _Display;
};
```

3. Create the `DemoCtrl.cpp` file and add it to the project.

This file defines the member functions of the `DemoCtrl` class that are not declared inline and the static class variable `_Display`.

- The constructor `DemoCtrl(HWND parent)` calls the corresponding constructor of the `Ctrl` class:

```
DemoCtrl::DemoCtrl(HWND hWnd)
: Ctrl(_Display, reinterpret_cast<IlvSystemView>(hWnd))
{
}
```

- The static member function `InitDisplay` creates an instance of `IlvDisplay` and stores it into `_Display`.

```
bool
DemoCtrl::InitDisplay(HINSTANCE hInstance) throw ()
{
    try {
        _Display = new IlvDisplay(hInstance, "ATL Views Sample");
        if (!_Display || _Display->isBad()) {
            IlvFatalError("Can’t initialize IlvDisplay.");
            if (_Display) {
                delete _Display;
                _Display = 0;
           }
        }
    }
    catch(...) {
        CleanDisplay();
    }
     return _Display ? true : false;
}
```

- The static function `CleanDisplay` releases the instance of `IlvDisplay` stored in `_Display`.

```
void
DemoCtrl::CleanDisplay()
{
    if (_Display) {
        delete _Display;
        _Display = 0;
    }
}
```

Modify the file generated by the wizard.

Modify the `DemoATL.cpp` file generated by the wizard to create and then delete an instance of the `IlvDisplay` class. Use the static member functions of the `DemoCtrl` class as follows:

1. Include the header file `DemoCtrl.h`.
2. In the `DllMain` function, add the following line at the end of the case `DLL_PROCESS_ATTACH`:

```
if (!DemoCtrl::InitDisplay(hInstance))
           return FALSE;
```

3. In the `DllMain` function, add the following lines at the beginning of the case `DLL_PROCESS_DETACH`:

```
DemoCtrl::CleanDisplay();
```

Add a new windows message handler to the CDemoATLCtrl class.

1. In the ClassView tab of the Project workspace window, expand the project folder DemoATL classes.
2. Click `CDemoATLCtrl` with the right mouse button.
3. Choose Add Windows Message Handler from the menu that appears and double-click `WM_CREATE`.

Add a private member variable for CDemoATLCtrl.

1. Expand DemoATL classes and select `CDemoATLCtrl`.
2. Press the right mouse button and choose Add Member Variable.
3. Enter `DemoCtrl*` in the Variable Type field.
4. Enter `m_Ctrl` in Variable Name field and click OK.

Modify the DemoATLCtrl.h file generated by the wizard.

Modify the file `DemoATLCtrl.h` generated by the wizard to create the instance of the class `DemoCtrl` used by the application as follows:

1. Add a forward declaration for the class `DemoCtrl` to the file `DemoATLCtrl.h`.
2. Include the header file `DemoCtrl.h` in the file `DemoATLCtrl.h`.
3. Initialize the variable `m_Ctrl` to `0` in the constructor (file `DemoATLCtrl.h`):

```
CDemoATLCtrl()
// Added for IBM ILOG Views.
: m_Ctrl(0)
// End Added for IBM ILOG Views.
{
        m_bWindowOnly = TRUE;
}
```

4. Add the following lines to the member function `CDemoATLCtrl::OnCreate` (file `DemoATLCtrl.h`):

```
try {
    m_Ctrl = new DemoCtrl(m_hWnd);
} catch (...) {
    if (m_Ctrl) {
        delete m_Ctrl;
        m_Ctrl = 0;
    }
}
```

Modify the DemoATL.rc file.

Add the following line at the end of the file `DemoATL.rc`.

```
#include "viewsdata.rc"
```

Modify the project settings (DemoATL.dsp).

1. Choose Settings from the Project menu.
2. In the Project Settings dialog box, select Win32Debug and click the C/C++ tab.
3. Select the General or Preprocessor option in the Category drop-down list and replace the preprocessor definition `_DEBUG` by `NDEBUG`.
4. Select All Configurations and click the C/C++ tab if necessary.
5. Select the General or Preprocessor option in the Category drop-down list and add `ILVSTD` in the “Preprocessor definitions” text field.
6. Select C++ language in the Category drop-down list and select the options Enable Exception handling and Enable Run-Time Type Information (RTTI).
7. Select Code Generation in the Category drop-down list and Multithreaded in “Use run-time library.”
8. Select Preprocessor in the Category drop-down list and type the following line in the “Additional include directories” text field:

   ```
   "., c:\ilog\views\include"
   ```

   Here we assume that IBM ILOG Views is installed in `C:\ilog\views\`.
9. Select “Precompiled headers” in the Category drop-down list and select the option “Not using precompiled headers” for the files `ctrl.cpp` and `DemoCtrl.cpp`.

   Because of the initialization mechanism of IBM ILOG Views, most of the IBM ILOG Views headers cannot be precompiled.
10. In the Link page, select Input in the Category drop-down list and add the path to the IBM ILOG Views libraries to the “Additional library path” text field.

By default, the path is `c:\ilog\lib\msvc6_mta`.

11. Select General or Input in the Category drop-down list and add the IBM ILOG Views libraries (that is, `winviews.lib`, `views.lib`, and `ilvgadgt.lib`) and also the `wsock32.lib` and `imm32.lib` libraries.

12. Select all the Release configurations and remove `_ATL_MIN_CRT` from the "Preprocessor definitions" text field.

    This option must be removed because the standard C++ runtime library is used.

Build the ActiveX control.

Now you can build the ActiveX control and test it. The wizard generates an html file, `DemoATLCtrl.htm`, that you can use to load the ActiveX control in Internet Explorer.

Since the default size of the ActiveX control is small, you can modify its width and height as follows:

```
<OBJECT WIDTH=400 HEIGHT=400 ID="DemoATLCtrl"  CLASSID="CLSID:3B10417D-3E6C-
11D3-B74F-00C04F68A89D"></OBJECT>
```

***Note:** Since the class id is generated, the class id in your file can be different from the one in this example.*

Step 3: Adding Custom Properties to the ActiveX Control

This step shows how to add a file name and a background color to the ActiveX control. The file name is that of the file read by the manager. By default, this file is `data/browse.ilv`, which we included in the `.rc` file in Step 2. The background color is that of the manager view.

You are going to:

- Add the FileName property.
- Add the Background property.
- Add new private member variables.
- Modify the CDemoAtlCtrl constructor.
- Add the setFileName member function.
- Add the setBackground member function.
- Modify the member functions generated by the wizard.
- Add lines to the property map macro.
- Modify the CDemoATLCtrl::OnCreate member function.

- Test the properties.

Add the FileName property.

1. In the ClassView tab of the Project Workspace window, click `IDemoAtlCtrl` with the right mouse button and select Add Property.
2. Enter `FileName` in the Property Name field.
3. Choose `BSTR` from the Property Type drop-down list.
4. Be sure the Parameters field is empty.
5. If necessary, select Get Function, Put Function, and PropPut.

   Leave PropPutRef unchecked. The attributes are `id` and `helpstring`.
6. Click OK.

Add the Background property.

1. In the ClassView tab of the Project Workspace window, click `IDemoAtlCtrl` with the right mouse button and select Add Property.
2. Enter `Background` in the Property Name field.
3. Choose `OLE_COLOR` in the Property Type drop-down list.
4. Leave the Parameters field empty.
5. If necessary, select Get Function, Put Function, and PropPut.

   Leave PropPutRef unckecked. The attributes are `id` (value 2) and `helpstring`.
6. Click OK.

Add new private member variables.

1. In the ClassView tab of the Project Workspace window, click `CDemoATLCtrl` with the right mouse button.
2. Choose Add Member Variable from the menu that appears.
3. Enter `CComBSTR` in Variable Type and `m_FileName` in Variable Name and click OK.
4. Choose Add Member Variable again.
5. Enter `OLE_COLOR` in Variable Type and `m_Background` in Variable Name and click OK.

Modify the CDemoAtlCtrl constructor.

In the file `DemoATLCtrl.h`, modify the `CDemoATLCtrl` constructor as follows to initialize the member variables `m_FileName` and `m_Background`:

```
CDemoATLCtrl()
// Added for IBM ILOG Views.
```

```
: m_Ctrl(0), m_FileName(), m_Background(-1)
// End Added for IBM ILOG Views.
{
    m_bWindowOnly = TRUE;
}
```

Add the setFileName member function.

This member function must be added to the `DemoCtrl` class. It has one parameter of type `const char*` that represents the new file name. Its purpose is to load the corresponding file.

1. Add the declaration of `DemoCtrl::setFileName` to the file `DemoCtrl.h`:

```
void setFileName(const char*);
```

2. Add the definition `DemoCtrl::setFileName` to the file `DemoCtrl.cpp`:

```
void
DemoCtrl::setFileName(const char* filename)
{
    if (!filename || !*filename)
        return;
    IlvManager* manager = getManager();
    if (!manager)
        return;
    manager->initReDraws();
    manager->deleteAll(IlvTrue, IlvFalse);
    manager->read(filename);
    manager->fitTransformerToContents(getManagerView());
    manager->reDrawViews();
}
```

Add the setBackground member function.

This member function must be added to the `DemoCtrl` class. It has one parameter of type `OLE_COLOR` that represents the new color. Its purpose is to convert the `OLE_COLOR` into an `IlvColor*` that IBM ILOG Views can interpret and to redraw the view.

1. Add the declaration of `DemoCtrl::setBackground` to the file `DemoCtrl.h`:

```
void setBackground(OLE_COLOR);
```

2. Add the definition of `DemoCtrl::setBackground` to the file `DemoCtrl.cpp`:

```
void
DemoCtrl::setBackground(OLE_COLOR oleColor)
{
    if (oleColor == -1)
        return;
    IlvManager* manager = getManager();
    if (!manager)
        return;
    COLORREF colRef;
    HPALETTE hPal = NULL;
    if (getDisplay()->screenDepth() <= 8)
        hPal = HPALETTE(getDisplay()->getPaletteHandle());
     if (S_OK != OleTranslateColor(oleColor, hPal, &colRef))
```

```
            return;
        IlvIntensity red, blue, green;
        getDisplay()->pixelToRGB((unsigned long)(colRef), red, blue, green);
        IlvColor* color = getDisplay()->getColor(red, blue, green);
         manager->initReDraws();
         manager->setBackground(getManagerView(), color);
         manager->reDraw();
         manager->reDrawViews();
}
```

Modify the member functions generated by the wizard.

1. Modify the member functions `CDemoATLCtrl::put_FileName` and `CDemoATLCtrl::get_FileName` generated by the wizard in the file `DemoATLCtrl.cpp` as follows:

```
STDMETHODIMP CDemoATLCtrl::put_FileName(BSTR newVal)
{
    // TODO: Add your implementation code here
    // Added for IBM ILOG Views.
    if (!newVal)
        return S_OK;
    if (m_FileName != newVal) {
        m_FileName = newVal;
        if (!m_FileName.Length())
            m_FileName.Empty();
        if (m_Ctrl && !!m_FileName)
            m_Ctrl->setFileName(m_FileName);
    }
     // End Added for IBM ILOG Views.
    return S_OK;
}



STDMETHODIMP CDemoATLCtrl::get_FileName(BSTR *pVal)
{
    // TODO: Add your implementation code here
     // Added for IBM ILOG Views.
    return m_FileName.CopyTo(pVal);
    // End Added for IBM ILOG Views.
    return S_OK;
}
```

2. Modify the member functions `CDemoATLCtrl::put_Background` and `CDemoATLCtrl::get_Background` generated by the wizard in the file `DemoATLCtrl.cpp`:

```
STDMETHODIMP CDemoATLCtrl::put_Background(OLE_COLOR newVal)
{
    // TODO: Add your implementation code here
    // Added for IBM ILOG Views.
    if (m_Background != newVal) {
        m_Background = newVal;
        if (m_Ctrl && (m_Background != -1))
            m_Ctrl->setBackground(m_Background);
   }
    // End Added for IBM ILOG Views.
    return S_OK;
}



STDMETHODIMP CDemoATLCtrl::get_Background(OLE_COLOR *pVal)
{
    // TODO: Add your implementation code here
    // Added for IBM ILOG Views.
    *pVal = m_Background;
    // End Added for IBM ILOG Views.
    return S_OK;
}
```

Add lines to the property map macro.

Add the following lines to the `BEGIN_PROP_MAP/END_PROP_MAP` macro declaration block in the `DemoATLCtrl.h` file.

```
// Added for IBM ILOG Views.
       PROP_ENTRY("FileName", 1, CLSID_NULL)
       PROP_ENTRY("Background", 2, CLSID_NULL)
// End Added for IBM ILOG Views.
```

Modify the CDemoATLCtrl::OnCreate member function.

Modify the member function `CDemoATLCtrl::OnCreate` in the file `DemoATLCtrl.h` to set the file name and/or the background color when the corresponding properties are defined:

```
LRESULT OnCreate(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
{
    // TODO : Add Code for message handler. Call DefWindowProc if necessary.
    // Added for IBM ILOG Views.
    try {
        m_Ctrl = new DemoCtrl(m_hWnd);
        // Added for the FileName property.
        m_Ctrl->setFileName(m_FileName);
        // Added for the Background property.
        m_Ctrl->setBackground(m_Background);
    } catch (...) {
        if (m_Ctrl) {
            delete m_Ctrl;
            m_Ctrl = 0;
    }
}
    // End Added for IBM ILOG Views.
    return 0;
}
```

Test the properties.

To test the properties, you can use either Internet Explorer or Visual Basic.

With Internet Explorer, you can modify the file `DemoATLCtrl.html` and use the JavaScript language. For an example with VBScript, see the section Test the properties. in *Step 4: Adding a Custom Event to The ActiveX Control*.

To test the `FileName` property, we use a combo box that has three predefined choices.

***Note:** This example assumes that the IBM ILOG Views data files are installed in the* `ILVHOME` *or* `ILVPATH` *environment variable paths.*

```
<HEAD>
<TITLE>ATL 3.0 test page for object DemoATLCtrl</TITLE>
</HEAD>

<SCRIPT LANGUAGE = JavaScript>

function load(value)
{
    DemoATLCtrl.FileName = value;
}

</SCRIPT>
<BODY>

<SELECT NAME=Files onchange = "load(value)" >
<OPTION Value="bunny.ilv">bunny
<OPTION Selected Value="data/browse.ilv">browser
<OPTION Value="buttrfly.ilv">butterfly
</SELECT>
<P>

<OBJECT WIDTH=400 HEIGHT=400 ID="DemoATLCtrl" CLASSID="CLSID:3B10417D-3E6C-
11D3-B74F-00C04F68A89D"></OBJECT>
</BODY>
</HTML>
```

***Note:** Since the class id is generated, the class id in your file may be different from the one in this example.*

See the project `VB/testatl.vbp` for an example of how to run a test via Visual Basic. Visual Basic lets you modify the `FileName` and `Background` properties dynamically, either by editing them in the Properties tab or by running a small program.

Step 4: Adding a Custom Event to The ActiveX Control

This step demonstrates how to add a custom event (`PointerPosition`) to the ActiveX control. Each time the user moves the mouse pointer, the member function `handleEvent` of the view interactor will fire this event and send it to the container. This event has three parameters:

- A `BSTR` parameter, which represents the name of the object underneath the mouse, if any.
- `OLE_XPOS_PIXELS` and `OLE_YPOS_PIXELS` parameters, which represent the position of the mouse relative to the `IlvManagerView`.

You are going to:

- Define the new custom event.
- Compile the file DemoATL.idl or rebuild your project.
- Implement a connection point interface.
- Check the parameter of the CONNECTION_POINT_ENTRY macro.
- Modify the constructor of the class DemoCtrl.
- Modify the member function CDemoATLCtrl::onCreate.
- Define the class TrackInteractor.
- Add the private m_Inter variable to the DemoCtrl class.
- Initialize the m_Inter to the DemoCtrl constructor and attach it to the view through the manager.
- Modify the destructor to delete the interactor stored in m_Inter.
- Test the properties.

Define the new custom event.

1. In the ClassView tab of the Project Workspace window, click the `_IDemoATLCtrlEvents` interface with the right mouse button.
2. Choose Add Method from the menu that appears.
3. In the dialog box, enter the following values: `HRESULT` as the Return Type, `PointerPosition` as the Method Name, and `BSTR name`, `OLE_XPOS_PIXELS x`, `OLE_YPOS_PIXELS y` as the parameters.

   No other attributes than the id and the helpstring are defined.

Compile the file DemoATL.idl or rebuild your project.

Implement a connection point interface.

1. In the ClassView tab of the Project Workspace window, click the class `CDemoATLCtrl` with the right mouse button.
2. Choose Implement Connection Point from the menu that appears.
3. In the Implement Connection Point interface, select the `_IDemoATLCtrlEvents` interface and click OK.

Check the parameter of the CONNECTION_POINT_ENTRY macro.

In the file `DemoATLCtrl.h`, check that the parameter of the macro `CONNECTION_POINT_ENTRY` is `DIID__IDemoATLCtrlEvents` and not `IID__IDemoATLCtrlEvents` in the `CONNECTION_POINT_MAP` macro.

Modify the constructor of the class DemoCtrl.

Modify this constructor so that it accepts a reference to the class `CDemoATLCtrl` as its second parameter. Add a forward declaration for the class `CDemoATLCtrl` to the file `DemoCtrl.h` and include the files `stdafx.h`, `DemoATL.h`, and `DemoATLCtrl.h` into the file `DemoCtrl.cpp`.

> ***Note:*** *You must include these files before the IBM ILOG Views header files because, although both IBM ILOG Views and the ATL force the definition of the* `STRICT` *macro, ATL does not check whether the macro is already defined.*

Modify the member function CDemoATLCtrl::onCreate.

In the member function `CDemoATLCtrl::OnCreate` (file `DemoATLCtrl.h`), add `*this` as the second parameter to the call to the `DemoCtrl` constructor.

Define the class TrackInteractor.

In the file `DemoCtrl.cpp`, define a new class `TrackInteractor` deriving from `IlvViewManagerInteractor`:

```
class TrackInteractor : public IlvManagerViewInteractor
    {
    public:
        TrackInteractor(IlvManager* m, IlvView* v, CDemoATLCtrl& ATLCtrl)
        : IlvManagerViewInteractor(m, v), m_ATLCtrl(ATLCtrl)
        {
        }
        ~TrackInteractor() {}
        void handleEvent(IlvEvent& event)
         {
             if (event.getType() == IlvPointerMoved) {
                 IlvGraphic* obj =
                     getManager()->lastContains(IlvPoint(event.x(), event.y()),
                                                getView());
                 CComBSTR bstrName("");
                 if (obj) {
                     const char* name = getManager()->getObjectName(obj);
                     bstrName = name;
                 }
                 m_ATLCtrl.Fire_PointerPosition(bstrName.Detach(),
                                                OLE_XPOS_PIXELS(event.x()),
                                                OLE_XPOS_PIXELS(event.y()));
            }
            if (!getManager()->dispatchToObjects(event, getView()))
                getManager()->shortCut(event, getView());
          }
private:
    CDemoATLCtrl& m_ATLCtrl;
};
```

Add the private m_Inter variable to the DemoCtrl class.

The `m_Inter` variable must be of the type `TrackInteractor*`.

To add this variable you can use the wizard as explained in *Step 3: Adding Custom Properties to the ActiveX Control*. Do not forget to include a forward declaration for the `TrackInteractor` class in the `DemoCtrl.h` file.

Initialize the m_Inter to the DemoCtrl constructor and attach it to the view through the manager.

```
DemoCtrl::DemoCtrl(HWND hWnd, CDemoATLCtrl& ATLCtrl)
: Ctrl(_Display, reinterpret_cast<IlvSystemView>(hWnd)), m_Inter(0)
{
    m_Inter = new TrackInteractor(getManager(), getManagerView(), ATLCtrl);
    getManager()->setInteractor(m_Inter, getManagerView());
}
```

Modify the destructor to delete the interactor stored in m_Inter.

```
DemoCtrl::~DemoCtrl()
{
    if (m_Inter) {
        getManager()->setInteractor(0, getManagerView());
        delete m_Inter;
         m_Inter = 0;
    }

}
```

Test the properties.

To test the properties, you can use either Internet Explorer or Visual Basic.

With Internet Explorer, you can modify the file `DemoATLCtrl.htm` to use the VBScript language.

The function `DemoATLCtrl_PointerPosition` is called each time a `PointerPosition` event is fired. It displays the coordinates of the object which the mouse is over, if any, and its name, if defined:

```
<HTML>
<HEAD>
<TITLE>ATL 3.0 test page for object DemoATLCtrl</TITLE>
</HEAD>

<SCRIPT LANGUAGE = "javascript">

<!--
function load(value)
{
    DemoATLCtrl.FileName = value;
}

-->
</SCRIPT>

<SCRIPT LANGUAGE = "VBScript">
```

```
<!--
Sub DemoATLCtrl_PointerPosition(name, x, y)
    text1.innerText = name & " (" & x & ", " & y & ")"
End Sub

-->
</SCRIPT>

<BODY>

<SELECT NAME=Files onchange = "load(value)" >
<OPTION Value="bunny.ilv">bunny
<OPTION Selected Value="data/browse.ilv">browser
<OPTION Value="buttrfly.ilv">butterfly
</SELECT>

<P>
<OBJECT WIDTH=400 HEIGHT=400 ID="DemoATLCtrl" CLASSID="CLSID:3B10417D-3E6C-
11D3-B74F-00C04F68A89D"></OBJECT>
<P>

<INPUT id=text1 name=text1>
</P>

</BODY>

</HTML>
```

***Note:** Since the class id is generated, the class id in your file may be different from the one in this example.*

See the project `VB/testatl.vbp` for an example of how to run a test via Visual Basic. The function `DemoATLCtrl1_PointerPosition` is called each time a `PointerPosition` event is fired.

Creating an ActiveX Control with the MFC Library

This tutorial has four steps:

- Step 1: Preparing the Program
- Step 2: Creating an ActiveX Control with MFC
- Step 3: Adding Custom Properties to the ActiveX Control
- Step 4: Adding a Custom Event to the ActiveX Control

The basic ActiveX control you will have created by the end of the second step will be enriched to integrate more functionality as you progress in this tutorial.

Note: *The following comments in the files generated by the wizard indicate lines of code that have been modified or added specifically for IBM ILOG Views:*
`// Added for` *IBM ILOG Views*
`// End Added for` *IBM ILOG Views*

Warning:

Before starting with this tutorial, you should be aware that you might encounter problems when building the ActiveX control. For some reason, symbols such as `_DllMain`, `new`, and `delete` are defined several times at link time. None of these symbols is defined in the code provided in this tutorial nor in the IBM ILOG Views libraries. It seems that the problem arises from a conflict between the following files: `nafxcw.lib` (used by the MFC), `libcmt.lib` (used by the C runtime library), and `libcpmt.lib` (used by the C++ runtime library).

However there is a workaround to that problem:

1. Place comment characters before all the lines referring to the `DemoCtrl` class in the files `DemoMFC.cpp` and `DemoMFCCtrl.cpp`.
2. Set the files `ctrl.cpp` and `DemoCtrl.cpp` in the project as inactive.
3. Build the ActiveX control.
4. Remove comment characters entered at step 1.
5. Reset the files `ctrl.cpp` and `DemoCtrl.cpp` as active.
6. Rebuild the ActiveX control.

Following this procedure eliminates the problem. To make things easier for you, the lines to be commented are enclosed in the following `if/endif` macro in the source files:

```
#if define(USE_VIEWS)
```

```
#endif
```

This macro is defined in the file `ctrl.h`. Simply place comment characters before the lines of code that this macro defines and remove them as needed.

We will report this problem to Microsoft as soon as we have identified precisely where it originates from.

Step 1: Preparing the Program

This step prepares the sample application used as a starting point for this tutorial so that it can be transformed to an ActiveX control and used inside another application. Basically what you have to do is separate the data from the code and make it possible to create the main view either as a top view (which is the case in the original program) or as a child view which can become part of a parent window.

Also these modifications will make it easier for you to add new functionality to the ActiveX control. The modified files are in the `step1` directory.

You are going to:

- Separate data from the code and put it into an .ilv file.
- Encapsulate the initialization code in a Ctrl class.
- Create a resource file (.rc) with ilv2data.
- Modify the manager.cpp file.

Separate data from the code and put it into an .ilv file.

Data stored in `.ilv` files is easier to modify than data stored as a string in a `.cpp` file. Also, any dependency problems can be solved with the IBM ILOG Views data files packaging tool, ilv2data.

The result is in the file `step1\data\ctrl.ilv`.

Notice that we have added attachments so that the main view can be resized in a consistent way. Also, we have named objects so that they can be accessed more easily from the program. All of these enhancements were made with IBM ILOG Views Studio.

> ***Note:*** *The separation of data from the code is not really necessary in this tutorial. However it is recommended to facilitate modifications.*

Encapsulate the initialization code in a Ctrl class.

The initialization code is encapsulated in the `Ctrl` class that will be used to create the ActiveX control and interact with it.

This class is declared in the file `step1\ctrl.h` and is defined in the file `step1\ctrl.cpp`. This `cpp` file also stores the callback functions.

The `Ctrl` class is basic. It derives from `IlvGadgetContainer`. It is composed of only three constructors and four member functions, of which one is private:

- The first constructor creates the `Ctrl` object as a top view, as in the original version of the file. It is used in the main function in `step1\manager.cpp` and has the following signature:

  ```
  Ctrl(IlvDisplay* display, const IlvRect& size, const char* filename = 0)
  ```

- The second constructor creates the `Ctrl` object as a child view. It has the following signature:

```
Ctrl(IlvDisplay* display,
     IlvSystemView parent,
     const IlvRect& size,
     const char* filename = 0)
```

- The third constructor creates the `Ctrl` object using the existing system view. It will be used to create the ActiveX control. It has the following signature:

```
Ctrl(IlvDisplay* display, IlvSystemView parent, const char* filename = 0)
```

- The public member functions `getManagerRectangle`, `getManagerView`, and `getManager` provide access to the objects loaded in the control. The private member function `init` is called to initialize the control, that is, read the data file, associate the callbacks with the objects, and so on. Most of the code that was originally in the member function `PopupManager` has been transferred to this class.

Create a resource file (.rc) with ilv2data.

Using `ilv2data`, we create an `.rc` file that contains the files `step1\data\ctrl.ilv` and `step1\data\browse.ilv`. This resource file could be used in this program, but we will use it later to build the ActiveX control.

For more information on ilv2data, see Packaging IBM ILOG Views Applications.

Modify the manager.cpp file.

The file `step1\manager.cpp` contains only the main function, which creates an instance of `IlvDisplay` and an instance of the new `Ctrl` class.

```
int
main(int argc, char* argv[])
{
    IlvDisplay* display =  new IlvDisplay("IlvDemo", "", argc, argv);
    if (!display || display->isBad()) {
        if (display)
            delete display;
        IlvFatalError("Couldn’t create display");
        return 1;
    }
    Ctrl* ctrl = new Ctrl(display, IlvRect(0, 0, 400, 400));
    IlvMainLoop();
    return 0;
}
```

Step 2: Creating an ActiveX Control with MFC

This step shows how to create an ActiveX control with the `Ctrl` class defined in *Step 1: Preparing the Program*. The ActiveX control you will obtain once this step is completed has a minimal interface.

You are going to:

- Create a new project.
- Add files to the project.
- Modify the file generated by the wizard.
- Add a new windows message handler to the CDemoMFCCtrl class.
- Add a private member variable for CDemoMFCCtrl.
- Modify the DemoMFCCtl.h file generated by the wizard.
- Modify the DemoMFC.rc file.
- Modify the project settings (DemoMFC.dsp)
- Now you can build the ActiveX control and test it.

Create a new project.

1. In Visual C++, choose New from the File menu and select MFC ActiveX ControlWizard in the Project page to begin creating a new project.
2. Type `DemoMFC` in the Project name text field and leave the proposed options as they are.
3. Click OK.

   Step 1 of the MFC ActiveX ControlWizard is displayed.
4. Keep the default values and click Next.

   Step 2 of the MFC ActiveX ControlWizard is displayed.
5. Check the following options: Activates when visible and Available in “Insert Object” dialog.

   You might also check Has an “About” box, but this is optional.

   Leave the other properties unchecked. The window must not be a subclass of any control. Also be sure that Windowless activation is unchecked in the Advanced ActiveX Features dialog box.
6. Click Finish and OK.

Add files to the project.

1. Add the file `ctrl.cpp` to the project.
2. Create the `DemoCtrl.h` file and add it to the project.

   This file should contain the declaration of the `DemoCtrl` class that follows. This class initializes the control. It also provides an interface between the code generated by the wizard and the IBM ILOG Views code and/or the `Ctrl` class.

   ```
   class DemoCtrl : public Ctrl
   {
   ```

```
public:
    DemoCtrl(HWND parent);
    ~DemoCtrl();
    static IlvDisplay* getDisplay()throw() { return _Display; }
    static bool InitDisplay(HINSTANCE hInstance) throw();
    static void CleanDisplay();

private:
    DemoCtrl(const DemoCtrl&); // Not defined to avoid
                               // default copy constructor.
    static IlvDisplay* _display;
};
```

3. Create the `DemoCtrl.cpp` file and add it to the project.

This file defines the member functions of the `DemoCtrl` class that are not declared inline and the static class variable `_Display` .

◆ The constructor `DemoCtrl(HWND parent)` calls the corresponding constructor of the `Ctrl` class:

```
DemoCtrl::DemoCtrl(HWND hWnd)
: Ctrl(_Display, reinterpret_cast<IlvSystemView>(hWnd))
{
}
```

◆ The static member function `InitDisplay` creates an instance of `IlvDisplay` and stores it into `_Display`.

```
bool
DemoCtrl::InitDisplay(HINSTANCE hInstance) throw ()
{
    try {
        _Display = new IlvDisplay(hInstance, "MFC Views Sample");
        if (!_Display || _Display->isBad()) {
            IlvFatalError("Can't initialize IlvDisplay.");
            if (_Display) {
                delete _Display;
                _Display = 0;
            }
        }
    }
    catch(...) {
        CleanDisplay();
    }
     return _Display ? true : false;
}
```

◆ The static function `CleanDisplay` releases the instance of `IlvDisplay` stored in `_Display`.

```
void
DemoCtrl::CleanDisplay()
{
    if (_Display) {
        delete _Display;
        _Display = 0;
        }
    }
```

Modify the file generated by the wizard.

Modify the file `DemoMFC.cpp` generated by the wizard to create and then delete an instance of the `IlvDisplay` class. Use the static member functions of the `DemoCtrl` class as follows:

1. Include the header file `DemoCtrl.h`.
2. In the `CDemoMFCApp::InitInstance` member function, add the following line to the `bInit` test.

   ```
   bInit = DemoCtrl::InitDisplay(AfxGetInstanceHandle());
   ```

3. Add the following line at the beginning of the `CDemoMFCApp::ExitInstance` member function:

   ```
   DemoCtrl::CleanDisplay();
   ```

Add a new windows message handler to the CDemoMFCCtrl class.

1. In the ClassView tab of the Project Workspace window, expand the project folder DemoMFC classes.
2. Click `CDemoMFCCtrl` with the right mouse button.
3. Choose Add Windows Message Handler from the menu that appears and double-click `WM_CREATE`.

Add a private member variable for CDemoMFCCtrl.

1. Expand DemoMFC classes and select `CDemoMFCCtrl`.
2. Press the right mouse button and choose Add Member Variable.
3. Enter `DemoCtrl*` in the Variable Type field.
4. Enter `m_Ctrl` in the Variable Name field and click OK.

Modify the DemoMFCCtl.h file generated by the wizard.

Modify the files `DemoMFCCtl.h` and `DemoMFCCtl.cpp` generated by the wizard to create the instance of the class `DemoCtrl` used by the application as follows:

1. Add a forward declaration for the class `DemoCtrl` to the file `DemoMFCCtl.h`.
2. Include the header file `DemoCtrl.h` in the file `DemoMFCCtl.cpp`.
3. Initialize the variable `m_Ctrl` to `0` in the constructor (file `DemoMFCCtl.cpp`):

   ```
   CDemoMFCCtrl()
   // Added for IBM ILOG Views.
   : m_Ctrl(0)
   // End Added for IBM ILOG Views.
   ```

```
{
       InitializeIIDs(&IID_DDemoMFC, &IID_DDemoMFCEvents);
       // TODO: Initialize your control's instance data here.
}
```

4. Add the following lines to the member function `CDemoMFCCtrl::OnCreate` (file `DemoMFCCtl.cpp`):

```
try {
    m_Ctrl = new DemoCtrl(m_hWnd);
} catch (...) {
    if (m_Ctrl) {
        delete m_Ctrl;
        m_Ctrl = 0;
     }
}
```

Modify the DemoMFC.rc file.

Add the following line at the end of the file `DemoMFC.rc`.

```
#include "viewsdata.rc"
```

Modify the project settings (DemoMFC.dsp)

1. Choose Settings from the Project menu.
2. In the Project Settings dialog box, select Win32 Debug and click the C/C++ tab.
3. Select the General or Preprocessor option in the Category drop-down list and replace the preprocessor definition `_DEBUG` by `NDEBUG`.
4. Select All Configurations and click the General tab.
5. Choose “Use MFC in a Static Library” from the Microsoft Foundation Classes drop-down list.
6. Activate the C++ tab, select the General or Preprocessor option in the Category drop-down list, and add `ILVSTD` to the “Preprocessor definitions” text field.
7. Select the C++ language in the Category drop-down list and the option “Enable Exception handling and Enable Run-Time Type Information (RTTI).”
8. Select Code Generation in the Category drop-down list and “Multithreaded in Use run-time library.”
9. Select Preprocessor in the Category drop-down list and type the following line in the “Additional include directories” text field:

   ```
   "., c:\ilog\views\include"
   ```

   Here we assume that IBM ILOG Views is installed in `C:\ilog\views\`.
10. Select “Precompiled headers” in the Category drop-down list and select the option “Not using precompiled headers” for the files `ctrl.cpp` and `DemoCtrl.cpp`.

Because of the initialization mechanism of IBM ILOG Views, most of the IBM ILOG Views headers cannot be precompiled.

11. In the Link page, select Input in the Category drop-down list and add the path to the IBM ILOG Views libraries to the “Additional library path” text field.

By default, the path is `c:\ilog\lib\msvc6_mta`.

12. Select General or Input in the Category drop-down list and add the IBM ILOG Views libraries (that is, `winviews.lib`, `views.lib`, and `ilvgadgt.lib`) and also the `wsock32.lib` and `imm32.lib` libraries.

Now you can build the ActiveX control and test it.

Create an html test file, like this:

```
<HTML>
<HEAD>
<TITLE>Test page for object DemoATLCtrl</TITLE>
</HEAD>

<BODY>

<OBJECT WIDTH=400 HEIGHT=400 ID="DemoATLCtrl" CLASSID="CLSID:3D31E7B8-400B-
11D3-B74F-00C04F68A89D"></OBJECT>

</BODY>

</HTML>
```

***Note:** Since the class id is generated, the class id in your file may be different from the one in this example.*

Step 3: Adding Custom Properties to the ActiveX Control

This step shows how to add a file name and a background color to the ActiveX control. The file name is that of the file read by the manager. By default, this file is `data/browse.ilv`, which we included in the `.rc` file in Step 2. The background color is that of the manager view.

- Add the FileName property.
- Add the Background property.
- Add new private member variables.
- Modify the CDemoMFCCtrl constructor.
- Add the setFileName member function.
- Add the setBackground member function.

- Modify the member functions generated by the wizard.
- Modify the CDemoMFCCtrl::OnCreate member function.
- Test the properties.

Add the FileName property.

1. In the ClassView tab of the Project Workspace window, click `_DDemoMFC` with the right mouse button and select Add Property.
2. Enter `FileName` in the External Name field.
3. Select the "Get/Set methods" option.
4. Choose `BSTR` from the Property Type drop-down list.
5. Be sure the Parameters list is empty.

Add the Background property.

1. In the ClassView tab of the Project Workspace window, click `_DDemoMFC` with the right mouse button and select Add Property.
2. Enter `Background` in the External Name field.
3. Select the option "Get/Set methods".
4. Choose `OLE_COLOR` from the Property Type drop-down list.
5. Leave the Parameters field empty.

Add new private member variables.

1. In the ClassView tab of the Project Workspace window, click `CDemoMFCCtrl` with the right mouse button.
2. Choose Add Member Variable from the menu that appears.
3. Enter `CString` in Variable Type and `m_FileName` in Variable Name and click OK.
4. Choose Add Member Variable again.
5. Enter `OLE_COLOR` in Variable Type and name `m_Background` in Variable Name and click OK.

Modify the CDemoMFCCtrl constructor.

In the file `DemoMFCCtl.cpp`, modify the `CDemoMFCCtrl` constructor as follows to initialize the member variables `m_FileName` and `m_Background`:

```
CDemoMFCCtrl()
// Added for IBM ILOG Views.
: m_Ctrl(0), m_FileName(), m_Background(-1)
// End Added for IBM ILOG Views.
{
        InitializeIIDs(&IID_DDemoMFC, &IID_DDemoMFCEvents);

        // TODO: Initialize your control's instance data here.
}
```

Add the setFileName member function.

This member function must be added to the `DemoCtrl` class. It has one parameter of type `const char*` that represents the new file name. Its purpose is to load the corresponding file.

1. Add the declaration of `DemoCtrl::setFileName` to the file `DemoCtrl.h`:

    ```
    void setFileName(const char*);
    ```

2. Add the definition `DemoCtrl::setFileName` to the file `DemoCtrl.cpp`:

    ```
    void
    DemoCtrl::setFileName(const char* filename)
    {
        if (!filename || !*filename)
            return;
        IlvManager* manager = getManager();
        if (!manager)
            return;
        manager->initReDraws();
        manager->deleteAll(IlvTrue, IlvFalse);
        manager->read(filename);
        manager->fitTransformerToContents(getManagerView());
        manager->reDrawViews();
    }
    ```

Add the setBackground member function.

This member function must be added to the `DemoCtrl` class. It has one parameter of type `OLE_COLOR` that represents the new color. Its purpose is to convert the `OLE_COLOR` into an `IlvColor*` that IBM ILOG Views can interpret and to redraw the view.

1. Add the declaration of `DemoCtrl::setBackground` to the file `DemoCtrl.h`:

    ```
    void setBackground(OLE_COLOR);
    ```

2. Add the definition of `DemoCtrl::setBackground` to the file `DemoCtrl.cpp`:

    ```
    void
    DemoCtrl::setBackground(OLE_COLOR oleColor)
    {
         if (oleColor == -1)
             return;
         IlvManager* manager = getManager();
         if (!manager)
             return;
         COLORREF colRef;
         HPALETTE hPal = NULL;
         if (getDisplay()->screenDepth() <= 8)
              hPal = HPALETTE(getDisplay()->getPaletteHandle());
         if (S_OK != OleTranslateColor(oleColor, hPal, &colRef))
             return;
    ```

```
         IlvIntensity red, blue, green;
         getDisplay()->pixelToRGB((unsigned long)(colRef), red, blue, green);
         IlvColor* color = getDisplay()->getColor(red, blue, green);
         manager->initReDraws();
         manager->setBackground(getManagerView(), color);
         manager->reDraw();
         manager->reDrawViews();
}
```

Modify the member functions generated by the wizard.

1. Modify the member functions `CDemoMFCCtrl::SetFileName` and `CDemoMFCCtrl::GetFileName` generated by the wizard in the file `DemoMFCCtl.cpp` as follows:

```
void CDemoMFCCtrl::SetFilename(LPCTSTR lpszNewValue)
{
    // TODO: Add your property handler here
    // Added for IBM ILOG Views.
    if (!lpszNewValue || !*lpszNewValue)
        return;
    if (m_FileName != lpszNewValue) {
        m_FileName = lpszNewValue;
        if (m_Ctrl && m_FileName.GetLength())
            m_Ctrl->setFileName(lpszNewValue);
    }
    // End Added for IBM ILOG Views.
    SetModifiedFlag();
}



BSTR CDemoMFCCtrl::GetFilename()
{
    CString strResult;
    // TODO: Add your property handler here
    strResult = m_FileName;
    return strResult.AllocSysString();
}
```

2. Modify the member functions `CDemoMFCCtrl::SetBackground` and `CDemoMFCCtrl::GetBackground` generated by the wizard in the file `DemoMFCCtl.cpp`:

```
void CDemoMFCCtrl::SetBackground(OLE_COLOR nNewValue)
{
    // TODO: Add your property handler here
    if (m_Background != nNewValue) {
        m_Background = nNewValue;
        if (m_Ctrl && (m_Background != -1))
            m_Ctrl->setBackground(m_Background);
    }
    SetModifiedFlag();
}
```

```
   OLE_COLOR CDemoMFCCtrl::GetBackground()
   {
       // TODO: Add your property handler here
       return m_Background;
   }
```

Modify the CDemoMFCCtrl::OnCreate member function.

Modify the member function `CDemoMFC::OnCreate` in the file `DemoMFCCtl.h` to set the file name and/or the background color when the corresponding properties are defined:

```
LRESULT OnCreate(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled)
{
    // TODO : Add Code for message handler. Call DefWindowProc if necessary.
    // Added for IBM ILOG Views.
    try {
        m_Ctrl = new DemoCtrl(m_hWnd);
        // Added for the FileName property.
        m_Ctrl->setFileName(m_FileName);
        // Added for the Background property.
        m_Ctrl->setBackground(m_Background);
     } catch (...) {
         if (m_Ctrl) {
             delete m_Ctrl;
             m_Ctrl = 0;
          }
      }
      // End Added for IBM ILOG Views.
      return 0;
}
```

Test the properties.

To test the properties, you can use either Internet Explorer or Visual Basic.

With Internet Explorer, you can modify the file `test.htm` and use the JavaScript language. For an example with VBScript, see the section Test the properties. in Step 4: Adding a Custom Event to the ActiveX Control.

To test the `FileName` property, we use a combo box that has three predefined choices.

Note: *This example assumes that the IBM ILOG Views data files are installed the* `ILVHOME` *or* `ILVPATH` *environment variable path.*

```
<HTML>
<HEAD>
<TITLE>Test page for object DemoMFCCtl</TITLE>
</HEAD>

<SCRIPT LANGUAGE = JavaScript>

function load(value)
{
     DemoMFCCtrl.FileName = value;
}
```

```
</SCRIPT>
<BODY>

<SELECT NAME=Files onchange = "load(value)" >
<OPTION Value="bunny.ilv">bunny
<OPTION Selected Value="data/browse.ilv">browser
<OPTION Value="buttrfly.ilv">butterfly
</SELECT>
<P>

<OBJECT WIDTH=400 HEIGHT=400 ID="DemoMFCCtrl" CLASSID="CLSID:3D31E7B8-400B-
11D3-B74F-00C04F68A89D"></OBJECT>
</BODY>
</HTML>
```

Note: *Since the class id is generated, the class id in your file may be different from the one in this example.*

See the project `VB/testMFC.vbp` for an example of how to run a test via Visual Basic. Visual Basic lets you modify the `FileName` and `Background` properties dynamically, either by editing them in the Properties tab or by running a small program.

Step 4: Adding a Custom Event to the ActiveX Control

This step demonstrates how to add a custom event (`PointerPosition`) to the ActiveX control. Each time the user moves the mouse pointer, the member function `handleEvent` of the view interactor will fire the event and send it to the container. This event has three parameters:

◆ A `BSTR` parameter, which represents the name of the object underneath the mouse, if any.

◆ `OLE_XPOS_PIXELS` and `OLE_YPOS_PIXELS` parameters, which represent the position of the mouse relative to the `IlvManagerView`.

You are going to:

◆ Define the new custom event.

◆ Compile the file DemoMFC.odl or rebuild your project.

◆ Modify the member function CDemoMFCCtrl::onCreate.

◆ Define the class TrackInteractor.

◆ Add the private m_Inter variable to the DemoCtrl class.

◆ Declare the class TrackInteractor as a friend of the class CDemoMFCCtrl (in the file DemoMFCCtl.h).

- Initialize the m_Inter variable in the DemoCtrl constructor and attach it to the view through the manager.
- Modify the destructor to delete the interactor stored in m_Inter.
- Test the properties.

Define the new custom event.

1. In the ClassView tab of the Project Workspace window, click the `_DDemoMFCEvents` interface with the right mouse button.
2. Choose Add Event from the menu that appears.
3. In the dialog box, enter the following values: `PointerPosition` in External Name, and `BSTR name`, `OLE_XPOS_PIXELS x`, `OLE_YPOS_PIXELS y` as the parameters.

Compile the file DemoMFC.odl or rebuild your project.

Modify the constructor of the class DemoCtrl.

Modify this constructor so that it accepts a reference to the class `CDemoMFCCtrl` as its second parameter. Add a forward declaration for the class `CDemoMFCCtrl` to the file `DemoCtrl.h` and include the files `stdafx.h`, `DemoMFC.h`, and `DemoMFCCtl.h` into the file `DemoCtrl.cpp`.

Note: *You must include these files before the IBM ILOG Views header files.*

Modify the member function CDemoMFCCtrl::onCreate.

In the member function `CDemoMFCCtrl::OnCreate` (file `DemoMFCCtl.cpp`), add `*this` as the second parameter to the call to the `DemoCtrl` constructor.

Define the class TrackInteractor.

In the file `DemoCtrl.cpp`, define a new class `TrackInteractor` deriving from `IlvViewManagerInteractor`:

```
class TrackInteractor : public IlvManagerViewInteractor
{
public:
    TrackInteractor(IlvManager* m, IlvView* v, CDemoMFCCtrl& MFCCtrl)
    : IlvManagerViewInteractor(m, v), m_MFCCtrl(MFCCtrl)
    {
    }
    ~TrackInteractor() {}
    void handleEvent(IlvEvent& event)
    {
         if (event.getType() == IlvPointerMoved) {
              IlvGraphic* obj =
              getManager()->lastContains(IlvPoint(event.x(), event.y()),
                                         getView());
              CString mfcName;
              if (obj) {
                  const char* name = getManager()->getObjectName(obj);
                  mfcName = name;
```

```
                }
                BSTR bstrName = mfcName.AllocSysString();
                m_MFCCtrl.FirePointerPosition(&bstrName,
                                              OLE_XPOS_PIXELS(event.x()),
                                              OLE_XPOS_PIXELS(event.y()));
                ::SysFreeString(bstrName);
            }
            if (!getManager()->dispatchToObjects(event, getView()))
                 getManager()->shortCut(event, getView());
        }
private:
    CDemoMFCCtrl& m_MFCCtrl;
};
```

Add the private m_Inter variable to the DemoCtrl class.

The `m_inter` variable must be of the type `TrackInteractor*`.

To add this variable you can use the wizard as explained in *Step 3: Adding Custom Properties to the ActiveX Control*. Do not forget to include a forward declaration for the `TrackInteractor` class in the `DemoCtrl.h` file.

Declare the class TrackInteractor as a friend of the class CDemoMFCCtrl (in the file DemoMFCCtl.h).

Initialize the m_Inter variable in the DemoCtrl constructor and attach it to the view through the manager.

```
DemoCtrl::DemoCtrl(HWND hWnd, CDemoMFCCtrl& MFCCtrl)
: Ctrl(_Display, reinterpret_cast<IlvSystemView>(hWnd)), m_Inter(0)
{
    m_Inter = new TrackInteractor(getManager(), getManagerView(), MFCCtrl);
    getManager()->setInteractor(m_Inter, getManagerView());
}
```

Modify the destructor to delete the interactor stored in m_Inter.

```
DemoCtrl::~DemoCtrl()
{
    if (m_Inter) {
        getManager()->setInteractor(0, getManagerView());
        delete m_Inter;
         m_Inter = 0;
    }

}
```

Test the properties.

To test the properties, you can use either Internet Explorer or Visual Basic.

With Internet Explorer, you can modify the file `test.htm` to use the VBScript language.

The function `DemoMFCCtrl_PointerPosition` is called each time a `PointerPosition` event is fired. It displays the coordinates of the object which the mouse is over, if any, and its name, if defined:

```
<HTML>
<HEAD>
```

```
<TITLE>Test page for object DemoMFCCtl</TITLE>
</HEAD>

<SCRIPT LANGUAGE = "javascript">

<!--
function load(value)
{
    DemoMFCCtrl.FileName = value;
}

-->
</SCRIPT>

<SCRIPT LANGUAGE = "VBScript">

<!--
Sub DemoMFCCtrl_PointerPosition(name, x, y)
    text1.innerText = name & " (" & x & ", " & y & ")"
End Sub

-->
</SCRIPT>

<BODY>

<SELECT NAME=Files onchange = "load(value)" >
<OPTION Value="bunny.ilv">bunny
<OPTION Selected Value="data/browse.ilv">browser
<OPTION Value="buttrfly.ilv">butterfly
</SELECT>

<P>
<OBJECT WIDTH=400 HEIGHT=400 ID="DemoMFCCtrl" CLASSID="CLSID:3D31E7B8-400B-
11D3-B74F-00C04F68A89D"></OBJECT>
<P>

<INPUT id=text1 name=text1>
</P>

</BODY>

</HTML>
```

***Note:** Since the class id is generated, the class id in your file may be different from the one in this example.*

See the project `VB/testMFC.vbp` for an example of how to run a test via Visual Basic. The function `DemoMFCCtrl1_PointerPosition` is called each time a `PointerPosition` event is fired.

C H A P T E R

2

Adding ActiveX Controls as Graphic Objects into an IBM ILOG Views Application

The purpose of this tutorial is to show how to add ActiveX™ controls as graphic objects into an IBM® ILOG® Views application. The example used in this tutorial is actually a very simple application built around an `IlvGrapher`.

This graphic object is built above the ATL library, and more specifically above the ATL Control Hosting API, which has been available since Version 4.0 of the ATL library. For more information, see the Microsoft documentation.

The current version has only been tested with Visual C++ 6.0 and Visual C++7.0.

A knowledge of IBM ILOG Views and C++ is a prerequisite to work through this tutorial. While a knowledge of COM and the ATL library is a not a strict requirement, it will allow you to better understand some technical parts of this tutorial.

Presentation of the IBM ILOG Views Application of this Example

This tutorial has the following steps:

◆ *Step 1: Creating an IlvGraphicCOMAdapter Object*

- *Step 2: Getting an Object Interface*
- *Step 3: Adding an Object Interactor*
- *Step 4: Adding a Browser*
- *Step 5: Getting the Value of the Property of a Control*

The example is just built above an `IlvGrapher`.

You can find the definition of the classes in the files include/ActiveXGraphicAdapterApp.h, include/ActiveXGraphicAdapter.h, and include/makenode.h. The definition of the member functions are in src/ActiveXGraphicAdapterApp.cpp, src/ActiveXGraphicAdapter.cpp, and src/specific.cpp. The data are in the file data/ActiveXGraphicAdapter.ilv. An rc file, src/ ActiveXApp.rc, was generated with `ilv2data`. We'll use it so that we don't worry about the path for the data.

The files src/ActiveXGraphicAdapterApp.cpp and src/ActiveXGraphicAdapter.cpp were generated by `ivstudio` and modified for the purpose of the application. For more precisions on this part of this application, see the other manuals of IBM ILOG Views. The file src/specific.cpp contains the main part of the subject of this tutorial. This file contains, among other functions, `ActiveXGraphicAdapter::specificInitialization`. Most of the work of this tutorial will be done in this function.

The application contains a menu and a scrolled grapher rectangle. Through the menu, you can:

- Clean the grapher (File/New),
- Load a file into the grapher (File/Open),
- Store the content of the grapher into a file (File/Save or File/SaveAs),
- Quit the application (File/Quit),
- Attach an interactor to the grapher in order to build new nodes (Interactors/Node Creation interactor),
- Attach an interactor to the grapher in order to build new links (Interactors/Link Creation Interactor),
- Attach a selection interactor to the grapher (Interactors/Selection Interactor),
- Detach all view interactor in order to use the object interactors, if any(Interactors/Object Interactor),
- Print help.

***Note:** In the code subdirectory for this tutorial, please note that the samples for this overview are in “step1,” those for the first step are in “step2,” and so forth.*

Step 1: Creating an IlvGraphicCOMAdapter Object

This step creates the `IlvGraphicCOMAdapter` object. Because the graphic object uses the ATL Control Hosting API, you may specify the identifier of the object in one of three ways. You can specify creating from a:

- CLSID – the default
- ProgID – define as SECOND_VERSION
- URL – define as THIRD_VERSION

You can test any of these by simply recompiling the file specific.cpp, after defining as shown if not using the default.

The example uses the Microsoft Calendar Control. If it is not available on your computer, you may modifiy the values of `DefaultIdentifier`.

The creation of the object is very simple:

```
IlvGraphic* obj = new IlvGraphicCOMAdapter(size, identifier, getDisplay());
```

where:

- `size` is the size of the object.
- `identifier` is the identifier of the control as explained above. Here are examples of the values that you can attach to the identifier:
  - CLSID: "{8E27C92B-1264-101C-8A2F-040224009C02}"
  - ProgID: "MSCAL.Calendar"
  - URL: "http://www.ilog.com/"

You also have to include the following header file in order to get the definition of the class `IlvGraphicAdapter`, and to add either the library `Ilvcomstat.lib` or `ilvcomdyn.lib`:

```
#include <ilviews/windows/comgadap.h>
```

You must also include the `oledlg.h` file in order to be able to use the common ActiveX dialog.

The following code is called in the `doIt` member function of the `MakeNodeInteractor` class:

```
void
MakeNodeInteractor::doIt(IlvRect& size)
{
    // the manager should actually be a grapher.
    IlvGrapher* grapher = ILVDYNAMICCAST(IlvGrapher*, manager());
    if (!grapher)
        return;
    // Creates the new IlvGraphicCOMAdapter object.
    const char* identifier = DefaultIdentifier;
    IlvGraphic* obj = new IlvGraphicCOMAdapter(size, identifier, getDisplay());
    // Deselects the previous object(s) if any.
    grapher->deSelect();
    // Adds the new object as a node into the grapher.
    grapher->addNode(obj, IlvTrue, grapher->getInsertionLayer());
    // Selects it.
    grapher->makeSelected(obj);
    // Deals with the commands if needed.
    if (grapher->isUndoEnabled())
        grapher->addCommand(new IlvAddNodeCommand(grapher,
                                                  obj,
                                                  grapher
                                                    ->getInsertionLayer()));
}
```

In order to be able to use this class you have to include several header files:

- `ilviews/grapher/commands.h` that contains the definition of the class `IlvAddNodeCommand` used in the member function `doIt` of `MakeNodeInteractor`. For more details, see the Managers section.
- `makenode.h` that contains the definition of the class `MakeNodeInteractor`;

Here is the definition of the class `MakeNodeInteractor`:

```
class MakeNodeInteractor : public IlvMakeRectangleInteractor
{
public:
    MakeNodeInteractor(IlvManager* manager, IlvView* view)
    : IlvMakeRectangleInteractor(manager, view)
    {
    }
    MakeNodeInteractor() : IlvMakeRectangleInteractor() {}

    void doIt(IlvRect&);
};
```

Step 2: Getting an Object Interface

In this step we just get an `IOleObject` interface—it exists on any of the controls we can insert—and we print the user type name through an accelerator bound to Ctrl-i.

The `IlvGraphicCOMAdapter` class uses only two interfaces directly: `IOleObject` and `IViewObject` (the ATL API uses more). However, it is possible to ask for a new interface in order to use the functionalities offered by the control. You can do this with the `IlvGraphicCOMAdapter::queryInterface` member function. This function is simply an encapsulation for the COM `QueryInterface` function, with the same parameters and the same return value.

Here is the code:

```
// --------------------------------------------------------------------------
// This function, associated to an accelerator, gets the user type name of
// the control and prints it.
```

```
static void
GetTypeAccelerator(IlvManager* man, IlvView*, IlvEvent&amp; ev, IlvAny)
{
    // Gets the object above which is the mouse pointer.
    IlvGraphic* graphic = man->lastContains(IlvPoint(ev.x(), ev.y()));
    // Any object?

    if (graphic) {
        // Is it an IlvGraphicCOMAdapter?
        IlvGraphicCOMAdapter* adapt=
           ILVDYNAMICCAST(IlvGraphicCOMAdapter*, graphic);

        if (adapt) {
          // Queries for the IOleObject interface.
          IOleObject* pInterface;
          HRESULT hr =
               adapt->queryInterface(IID_IOleObject, (void**)&pInterface);
           // Smart pointer.
           IlvCOMInterface<IOleObject> oleObject(pInterface);

            // Succeeded?
           if (SUCCEEDED(hr)) {
               // Asks for the user type name.
               LPOLESTR userType;
               hr = oleObject->GetUserType(USERCLASSTYPE_FULL, &userType);

               // Succeeded?
               if (SUCCEEDED(hr)) {
                   // Prints the result.
                   IlvCOut << "The type is: " << userType << ".\n";
                   // Asks for the global allocator.
                   LPMALLOC pMalloc;
                   if (SUCCEEDED(CoGetMalloc(1, &pMalloc))) {
                       // Releases the memory allocated by GetUserType.
                       pMalloc->Free(userType);
                       // Releases the global allocator.
                       pMalloc->Release();
                   }
              }
           }
        }
    }
}
```

We also define the following operator in order to be able to print a string in UNICODE:

```
// -----------------------------------------------------------------------
// This operator prints wchar_t strings into a standard ostream.
ostream&
operator<<(ostream& o, const wchar_t* str)
{
    size_t len = wcslen(str);
    size_t size = wcstombs(0, str, len) + 1;
    char* buf = new char[size];
    wcstombs(buf, str, len+1);
    o << buf;
    delete[] buf;
    return o;
}
```

As you can see, most of the code is pure COM one. In this code, we use a smart pointer that prevents worrying about the release of the interface.

This code uses the class `IlvCOut`. So you have to include the header file `ilviews/base/error.h`.

Step 3: Adding an Object Interactor

In this step we show the usage of the object interactor `IlvGraphicComAdapterInteractor` (this class is defined in the header file `ilviews/windows/comgint.h`). It lets you interact with the control (although there are some limitations) even when you apply a transformation to the graphic object. In fact, you just have to attach an instance of the `IlvGraphicComAdapterInteractor` to the `IlvGraphicCOMAdapter` object, which is accomplished with the function `AttachInteractorAccelerator`:

```
void
AttachInteractorAccelerator(IlvManager* man,
                            IlvView*,
                            IlvEvent& ev,
                            IlvAny arg)
{
  // The arg parameter is actually an object interactor.
  IlvManagerObjectInteractor* inter =
    ILVREINTERPRETCAST(IlvObjectInteractor*, arg);
  // Gets the object above which is the mouse pointer.
  IlvGraphic* graphic = man->lastContains(IlvPoint(ev.x(), ev.y()));
  // Any object?

  if (graphic) {
    // Is it an IlvGraphicCOMAdapter?
    IlvGraphicCOMAdapter* adapt =
      ILVDYNAMICCAST(IlvGraphicCOMAdapter*, graphic);

  if (adapt) {
    // Attaches the interactor to the object.
    adapt->setInteractor(inter);
    }
  }
}
```

The `IlvGraphicComAdapterInteractor` object is given as a parameter of the function and was allocated in the function `ActiveXGraphicAdapter::specificInitialization`:

```
_objectInteractor = new IlvGraphicComAdapterInteractor;
    // Adds an accelerator in order to attach an object interactor to the
    // control under the mouse pointer.
    grapher->addAccelerator(AttachInteractorAccelerator,
                            IlvKeyUp,
                            IlvCtrlChar('b'),
                            0,
                                        _objectInteractor);
```

Step 4: Adding a Browser

This step shows how to add a browser. You may use a control chooser to get the identifier of the control. You just have to use the value returned by the `show` function as the identifier, in the same way you use the value returned by `IlvFileBrowser::show()` to get a filename.

Here is the new code for the member function `IlvMakeNodeInteractor::doIt`:

```
void
MakeNodeInteractor::doIt(IlvRect& size)
{
    // the manager should actually be a grapher.
    IlvGrapher* grapher = ILVDYNAMICCAST(IlvGrapher*, manager());
    if (!grapher)
        return;
    // Creates the new IlvGraphicCOMAdapter object.
    IlvControlBrowser cBrowser(getView());
    const char* identifier = cBrowser.show();
    if (!identifier)
        identifier = DefaultIdentifier;
    IlvGraphic* obj = new IlvGraphicCOMAdapter(size, identifier, getDisplay());
    // Deselects the previous object(s) if any.
    grapher->deSelect();
    // Adds the new object as a node into the grapher.
    grapher->addNode(obj, IlvTrue, grapher->getInsertionLayer());
    // Selects it.
    grapher->makeSelected(obj);
    // Deals with the commands if needed.
    if (grapher->isUndoEnabled())
        grapher->addCommand(new IlvAddNodeCommand(grapher, obj,
                                              grapher ->getInsertionLayer()));
}
```

The class `IlvControlBrowser` is defined in the header file `ilviews/windows/cbrowser.h`. You must add the library `oledlg.lib` in order to be able to use this class.

Step 5: Getting the Value of the Property of a Control

This step shows how to get the value of a property of the control. The code is based on the property `Day` of the `Calendar` control. You will have to modify the name of the property and its type if you choose another property and/or another control.

We will just modify slightly the code of step3 so that the `GetTypeAccelerator` also displays the value of the property `Day` when the control type is `Calendar`.

To do this, we replace the line:

```
IlvCOut << "The type is: " << userType << ".\n";
```

by the first line below:

```
IlvCOut << "The type is: " << userType;
```

```
// Code added for step 5.
if (!wcscmp(userType, L"Calendar")) {
  IlvValue value("Day");
  IlInt day = adapt->queryValue(value);
  IlvCOut << ", the day is: " << day;
}
IlvCOut << ".\n";
```

After having checked that the type of the control is effectively `Calendar`, an `IlvValue` object whose name is the name of the property (`Day` in this case) is instantiated. Then the value of this property is obtained by calling `queryValue` on the `IlvGraphicCOMAdapter` object (as for any value object in IBM ILOG Views). Then the value is displayed with `IlvCOut`.

You see that it is very easy to access a property and to get its value. To modify its value would also be simple. For example, to set the `Day` property to 5, you would write:

```
IlvValue value("Day", 5);
adapt->changeValue(value);
```

See the reference pages of `IlvValue` and `IlvGraphic` for more details.

Index