IBM ILOG Views

Foundation V5.3

チュートリアル

2009 年 **6** 月

著作権の告知

商標

IBM、IBM ロゴ、ibm.com、Websphere、ILOG、ILOG のデザイン、および CPLEX は、世界中の多くの国の管轄権で登録されている International Business Machines Corp. の商標または登録商標です。その他の製品およびサービス名は、IBM またはその他の企業の商標です。IBM 社の現在の商標一覧は、http://www.ibm.com/legal/copytrade.shtml にある Copyright and trademark information (著作権と商標についての情報) にあります。

Adobe、Adobe のロゴ、PostScript、および PostScript のロゴは、米国およびその他の国における Adobe Systems Incorporated の商標または登録商標です。

Linux は、米国およびその他の国における Linus Torvalds の登録商標です。

Microsoft、Windows、Windows NT、および Windows のロゴは、米国およびその他の国における Microsoft Corporation の商標です。

Java およびすべての Java に基づいた商標とロゴは、米国およびその他の国の Sun Microsystems, Inc. の商標です。

その他の企業、製品およびサービス名は、その他の企業の商標またはサービス商標です。

告知

詳細は、インストールした製品の <*installdir*>/license/notices.txt を参照してください。

目 次

このチュートリアルについて

本チュートリアルでは、既存の IBM® ILOG® Views コードから ActiveX コントロールを作成する方法と、グラフィック・オブジェクトとして ActiveX コントロールを IBM ILOG Views アプリケーションに追加する方法について説明します。

前提事項

本書では、特定のウィンドウシステムを含め、ユーザが IBM® ILOG® Views を使用する PC や UNIX® 環境について精通していることが前提となっています。IBM ILOG Views は C++ 開発者用に作成されているため、このマニュアルでは、ユーザが C++ のコードを作成できること、および C++ の開発環境について精通しており、ファイルやディレクトリの操作、テキスト・エディタの使用、C++ プログラムのコンパイルおよび実行ができることも前提となっています。

表記法

書体の規則

以下の書体に関する規則は、このマニュアル全体に適用されます。

- コードの引用およびファイル名は、`"code"` 書体で記述されます。

- ダイアログ・ボックスなどのように、ユーザが行う入力は "code" 書体で記述されます。
- ユーザが指定するコマンド変数は*斜体*で記載されます。
- 初出の*斜体*の用語には、用語集で説明されているものがあります。

命名規則

以下の命名規則は、マニュアル全体を通して API に適用されます。

- IBM ILOG Views ライブラリで定義されている型、クラス、関数、マクロの名前は `Ilv` で始まります。たとえば、`IlvGraphic` のようになります。
- IBM ILOG Views 専用でない型、マクロの名前は `Il` で始まります。たとえば、`IlBoolean` のようになります。
- クラス名、およびグローバル関数は、最初の文字が大文字で表された連結語として記載されます。

  class `IlvDrawingView`;

- 仮想および通常メソッドの名前は小文字で始まります。スタティック・メソッドの名前は大文字で始まります。例 :

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

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

章

1

既存の IBM ILOG Views コードから ActiveX コントロールを作成する

IBM® ILOG® Views を Microsoft Foundation Classes (MFC) ライブラリまたは、Active Template Library (ATL) と併用し、ActiveX コントロールを作成することができます。

次に示す 2 つのチュートリアルでは、既存の IBM ILOG Views プログラムから ActiveX コントロールを作成する方法を示します。

- ATL ライブラリで ActiveX コントロールを作成する
- MFC ライブラリで ActiveX コントロールを作成する

次のファイルで提供されるサンプル IBM ILOG Views プログラムから、選択するライブラリに従って ATL COM AppWizard または MFC ActiveX ControlWizard を使って、ActiveX コントロールを構築します。`manager.cpp`

このプログラムでは、地図の拡大・縮小およびスクロールができる地図をマネージャ内に表示します。

また、ActiveX をゼロから作成することもできます。その場合の手順は、このチュートリアルに示すものと同じです。

これらのチュートリアルで学習する前提条件として、IBM ILOG Views、C++、Visual C++、COM、ATL または MFC ライブラリの知識が必要です。

ATL ライブラリで ActiveX コントロールを作成する

このチュートリアルには、4 つのステップがあります。

- ステップ 1: プログラムの準備
- ステップ 2: ATL で ActiveX コントロールを作成する
- ステップ 3: ActiveX コントロールへカスタム・プロパティを追加する
- ステップ 4: ActiveX コントロールへカスタム・イベントを追加する

第 2 ステップで作成する基本 ActiveX は、このチュートリアルがさらに進行するに従って、より多くの機能を統合するために強化されます。

> ***メモ:*** *ウィザードによって生成されたファイルに含まれる次のコメントは、* IBM ILOG Views *用に変更または追加されたコード行を示します。*
>
> ```
> // Added for IBM ILOG Views
> // End Added for IBM ILOG Views
> ```

ステップ 1: プログラムの準備

このチュートリアルの出発点として使用するサンプル・アプリケーションを準備するステップです。サンプルは最後に ActiveX コントロールに変換して、他のアプリケーション内で使用できます。このステップで必要な作業は基本的に、データをコードから分離して、トップ・ビュー (元のプログラムの場合) または親ウィンドウの一部となる子ビューとして、メインビューを作成できるようにすることです。

また、これらの変更によって、新しい機能を ActiveX コントロールに追加するのが簡単になります。変更されたファイルが、`step1` ディレクトリにあります。

次の処理を行います。

- コードからデータを分離して、.ilv ファイルに配置する
- Ctrl クラスの初期化コードをカプセル化する
- リソース・ファイル (.rc) を ilv2data で作成する
- manager.cpp ファイルの変更

コードからデータを分離して、.ilv ファイルに配置する

`.ilv` ファイルに格納されたデータは、`.cpp` ファイルに格納された文字列としてのデータよりも変更しやすくなります。また、あらゆる依存性の問題が、IBM ILOG Views データ・ファイル・パッケージ化ツールの ilv2data によって解決できます。

結果は、`step1\data\ctrl.ilv` ファイルにあります。

アタッチメントを追加したため、メイン・ビューを一貫した方法でリサイズできます。また、オブジェクトに名前を付けたので、プログラムからより容易にアクセスできます。これらの機能強化はすべて、IBM ILOG Views Studio で行われました。

> *メモ: コードからのデータ分離は、このチュートリアルで不可欠というわけではありませんが、変更を容易にするためにお勧めします。*

Ctrl クラスの初期化コードをカプセル化する

初期化コードは、ActiveX コントロールの作成と対話に使用する `Ctrl` クラスにカプセル化されています。

このクラスは、`step1\ctrl.h` ファイルで宣言し、`step1\ctrl.cpp` ファイルで定義します。この `cpp` ファイルには、コールバック関数も格納します。

`Ctrl` が基本クラスです。`IlvGadgetContainer` から派生します。3 つのコンストラクタと 4 つのメンバ関数だけで構成され、そのうち 1 つはプライベートです。

- 最初のコンストラクタは、`Ctrl` オブジェクトをファイルの元のバージョンと同様にトップ・ビューとして作成します。`step1\manager.cpp` の main 関数で使用され、次の署名があります。

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

- 2 番目のコンストラクタは、`Ctrl` オブジェクトを子ビューとして作成します。次の署名があります。

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

- 3 番目のコンストラクタは、`Ctrl` オブジェクトを既存のシステム・ビューで作成します。これは、ActiveX コントロールの作成に使用されます。次の署名があります。

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

- パブリック・メンバ関数 `getManagerRectangle`、`getManagerView`、`getManager` は、コントロールにロードされる各オブジェクトへのアクセスを提供します。. プライベート・メンバ関数 `init` はコントロールを初期化するために呼び出されます。つまり、データ・ファイルを読み込んだり、コールバックをオブジェクトに関連付けたりします。メンバ関数 `PopupManager` 内にあったコードのほとんどが、このクラスに移動しています。

リソース・ファイル (.rc) を ilv2data で作成する

`ilv2data` を使って、ファイル `step1\data\ctrl.ilv` および `step1\data\browse.ilv` が含まれる `.rc` ファイルを作成します。このリソース・ファイルをサンプル・プログラムで使うこともできますが、このチュートリアル

では後で ActiveX コントロールを構築するために使用します。

ilv2data の詳細は、*IBM ILOG Views アプリケーションのパッケージ化*を参照してください。

manager.cpp ファイルの変更

ファイル step1\manager.cpp には、IlvDisplay のインスタンスと、新しい Ctrl クラスのインスタンスを作成する main 関数だけが含まれます。

```
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;
}
```

ステップ 2: ATL で ActiveX コントロールを作成する

このステップでは、*ステップ1: プログラムの準備* で定義した Ctrl クラスによって ActiveX コントロールを作成する方法について説明します。このステップの終了時に得られる ActiveX コントロールには、最小限度のインターフェースしかありません。

次の処理を行います。

- 新規プロジェクトの作成
- 新規 ATL オブジェクトの追加
- プロジェクトにファイルを追加する
- ウィザードで生成されたファイルを変更する
- CDemoATLCtrl クラスに新しい Windows メッセージ・ハンドラを追加する
- CDemoATLCtrl にプライベート・メンバ変数を追加する
- ウィザードで生成された DemoATLCtrl.h ファイルを変更する
- DemoATL.rc ファイルの変更
- プロジェクト設定 (DemoATL.dsp) の変更
- ActiveX コントロールの構築

新規プロジェクトの作成

1. Visual C++ で、[ファイル] メニューから [新規作成] を選択し、[プロジェクト] ページで ATL COM AppWizard を選んで、新規プロジェクトの作成を開始します。
2. [プロジェクト名] テキスト・フィールドに `DemoATL` と入力し、[OK] をクリックします。

   ATL COM AppWizard が表示されます。
3. [ダイナミック リンク ライブラリ (DLL)] オプションを選択し、[プロキシ / スタブ コードのマージを許可]、[MFC のサポート]、[MTS のサポート] はチェックなしのままにします。

> ***メモ:*** *これらのオプションは、このチュートリアルでは不要です。ただし、最後に取得するものによっては、後で選択が必要になる場合もあります。*

4. [終了] と [OK] をクリックします。

新規 ATL オブジェクトの追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブを、必要ならばクリックして有効にし、DemoATL クラスのラベルをマウスの右ボタンでクリックします。
2. 表示されたメニューから ［ATL オブジェクトの新規作成］を選択し、ATL オブジェクトを作成します。

   ATL オブジェクト・ウィザードが表示されます。
3. カテゴリで [コントロール] を選択して、オブジェクトで [フル コントロール] をダブルクリックします。
4. [ショートネーム] テキスト・フィールドに `DemoATLCtrl` と入力します。

   他のフィールドには自動的にラベルが入るので、そのままにしておきます。
5. [アトリビュート] タブをクリックして、[アパートメント スレッド モデル]、[デュアル インターフェイス]、最後に [アグリゲーション] で [はい] を選択します。[接続ポイントのサポート] オプションもチェックします。
6. [その他] タブをクリックして、[不透明]、[ソリッドな背景]、[正規化 DC]、[挿入可能]、[ウィンドウのみ] と選択します。

   他のオプションはチェックなしのままにします。ストック・プロパティの宣言は行いません。

> ***メモ:*** *[アトリビュート] タブと、[その他] タブのオプション選択は、最終的に取得したいものによって異なる場合があります。*

プロジェクトにファイルを追加する

1. ファイル ctrl.cpp および ctrl.h を、プロジェクトに追加します。
2. DemoCtrl.h ファイルを作成して、プロジェクトに追加します。

   このファイルには、それに続く DemoCtrl クラスの宣言を含む必要があります。このクラスがコントロールを初期化します。また、ウィザードによって生成されたコードと、IBM ILOG Views コードおよび / または Ctrl クラス間のインターフェースを提供します。

```
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. DemoCtrl.cpp ファイルを作成して、プロジェクトに追加します。

   このファイルはインラインで宣言されていない DemoCtrl クラスのメンバ関数と、スタティック・クラス変数 _Display を定義します。

◆ コンストラクタ DemoCtrl(HWND parent) は、対応する Ctrl クラスのコンストラクタを呼び出します。

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

◆ スタティック・メンバ関数 InitDisplay は、IlvDisplay のインスタンスを作成して、_Display に格納します。

```
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;
}
```

- スタティック関数 CleanDisplay は、_Display に格納された IlvDisplay のインスタンスを解放します。

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

ウィザードで生成されたファイルを変更する

ウィザードで生成された DemoATL.cpp ファイルを変更して、IlvDisplay クラスのインスタンスを作成してから、削除します。DemoCtrl クラスのスタティック・メンバ関数を、次の手順で使用します。

1. ヘッダー・ファイル DemoCtrl.h をインクルードします。
2. DllMain 関数で、DLL_PROCESS_ATTACH ケースの最後に次の行を追加します。

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

3. DllMain 関数で、DLL_PROCESS_DETACH ケースの最初に次の行を追加します。

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

CDemoATLCtrl クラスに新しい Windows メッセージ・ハンドラを追加する

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、プロジェクト・フォルダ DemoATL クラスを展開します。
2. CDemoATLCtrl をマウスの右ボタンでクリックします。
3. 表示されるメニューから [Windows メッセージ・ハンドラの追加] を選択して、WM_CREATE をダブルクリックします。

CDemoATLCtrl にプライベート・メンバ変数を追加する

1. DemoATL クラスを展開して、CDemoATLCtrl を選択します。
2. マウスの右ボタンを押して、[メンバ変数の追加] を選択します。
3. [変数] 入力フィールドに、DemoCtrl* とタイプします。
4. [変数名] フィールドに m_Ctrl と入力してから、[OK] をクリックします。

ウィザードで生成された DemoATLCtrl.h ファイルを変更する

ウィザードで生成されたファイル DemoATLCtrl.h を変更して、アプリケーションで使用する DemoCtrl クラスのインスタンスを、次の手順で作成します。

1. DemoCtrl クラスのフォワード宣言を、ファイル DemoATLCtrl.h に追加します。
2. ファイル DemoATLCtrl.h に、ヘッダー・ファイル DemoCtrl.h をインクルードします。
3. コンストラクタ (ファイル DemoATLCtrl.h) 内の変数 m_Ctrl を 0 に初期化します。

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

4. 次に示すコード行を、メンバ関数 CDemoATLCtrl::OnCreate (ファイル DemoATLCtrl.h) に追加します。

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

DemoATL.rc ファイルの変更

次の行を、ファイル DemoATL.rc の最後に追加します。

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

プロジェクト設定 (DemoATL.dsp) の変更

1. [プロジェクト] メニューで [設定] を選択します。
2. [プロジェクトの設定] ダイアログ・ボックスで、[Win32Debug] を選択して、[C/C++] タブをクリックします。
3. [カテゴリ] ドロップダウン・リストで [一般]、または [プリプロセッサ] オプションを選択し、プリプロセッサの定義 _DEBUG を NDEBUG に変更します。
4. 必要ならば、[すべての構成] を選択して [C/C++] タブをクリックします。
5. [カテゴリ] ドロップダウン・リストで [一般]、または [プリプロセッサ] オプションを選択し、[プリプロセッサ定義] テキスト・フィールドに ILVSTD を追加します。

6. [カテゴリ] ドロップダウン・リストで C++ 言語を選んでから、[例外処理を有効にする]、[ランタイム タイプ情報 (RTTI) を有効にする] オプションを選択します。

7. [カテゴリ] ドロップダウン・リストで [コード生成] を選択してから、[使用するランタイム ライブラリ] の [マルチスレッド] を選択します。

8. [カテゴリ] ドロップダウン・リストで [プリプロセッサ] を選択し、[インクルード ファイルのパス] テキスト・フィールドに次の行を入力します。

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

   ここでは、IBM ILOG Views が `C:\ilog\views\` にインストールされていることを前提にしています。

9. [カテゴリ] ドロップダウン・リストの [プリコンパイル済みヘッダー] を選んでから、ファイル `ctrl.cpp` および `DemoCtrl.cpp` について [プリコンパイル済みヘッダーを使用しない] オプションを選択します。

   IBM ILOG Views の初期化メカニズムのため、ほとんどの IBM ILOG Views ヘッダーはプリコンパイルできません。

10. [リンク] ページの [カテゴリ] ドロップダウン・リストで [入力] を選んでから、[追加ライブラリ パス] テキスト・フィールドに、IBM ILOG Views ライブラリへのパスを追加します。

    デフォルトのパスは `c:\ilog\lib\msvc6_mta` です。

11. [カテゴリ] ドロップダウン・リストで [一般] または [入力] を選択してから、IBM ILOG Views ライブラリ (すなわち `winviews.lib`、`views.lib`、`ilvgadgt.lib`) および、`wsock32.lib`、`imm32.lib` ライブラリを追加します。

12. すべてのリリース設定を選択してから、[プリプロセッサの定義] テキスト・フィールドから、`_ATL_MIN_CRT` を削除します。

    標準 C++ ランタイム・ライブラリを使用するため、このオプションは削除する必要があります。

ActiveX コントロールの構築

これで ActiveX コントロールを構築してテストできます。ウィザードは、html ファイル `DemoATLCtrl.htm` を生成します。これを使って ActiveX コントロールが、Internet Explorer にロードできます。

この ActiveX コントロールはデフォルト・サイズが小さいため、その幅と高さを次のように変更できます。

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

メモ: クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。

ステップ 3: ActiveX コントロールへカスタム・プロパティを追加する

このステップでは、ActiveX コントロールにファイル名と背景色を追加する方法について説明します。ファイル名は、マネージャによって読み込まれるファイルの名前です。デフォルトでこのファイルは `data/browse.ilv` で、これはステップ 2 において `.rc` ファイルにインクルードしました。また背景色は、マネージャ・ビューの背景色になります。

次の処理を行います。

- ファイル名プロパティの追加
- 背景色プロパティの追加
- 新しいプライベート・メンバ変数の追加
- CDemoAtlCtrl コンストラクタの変更
- setFileName メンバ関数の追加
- setBackground メンバ関数の追加
- ウィザードで生成されたメンバ関数の変更
- プロパティ・マップ・マクロにコード行を追加する
- CDemoATLCtrl::OnCreate メンバ関数の変更
- プロパティのテスト

ファイル名プロパティの追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、`IDemoAtlCtrl` をマウスの右ボタンでクリックして、[プロパティの追加] を選択します。
2. [プロパティ名] フィールドに、`FileName` と入力します。
3. [プロパティのタイプ] ドロップダウン・リストから、`BSTR` を選択します。
4. [パラメータ] フィールドが空欄であることを確認します。
5. 必要ならば、[取得関数]、[設定]、[PropPut] を選択します。

   [PropPutRef] はチェックなしのままにします。アトリビュートは `id` および `helpstring` です。

6. [OK] をクリックします。

背景色プロパティの追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、IDemoAtlCtrl をマウスの右ボタンでクリックして、[プロパティの追加] を選択します。
2. [プロパティ名] フィールドに、Background と入力します。
3. [プロパティ タイプ] ドロップダウン・リストから、OLE_COLOR を選択します。
4. [パラメータ] フィールドは空欄のままにします。
5. 必要ならば、[取得関数]、[設定]、[PropPut] を選択します。

   [PropPutRef] はチェックなしのままにします。アトリビュートは id (値 2) および helpstring です。
6. [OK] をクリックします。

新しいプライベート・メンバ変数の追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、CDemoATLCtrl をマウスの右ボタンでクリックします。
2. 表示されるメニューから [メンバ変数の追加] を選択します。
3. [変数タイプ] フィールドに CComBSTR、[変数名] フィールドに m_FileName と入力し、[OK] をクリックします。
4. [メンバ変数の追加] を再び選択します。
5. [変数タイプ] フィールドに OLE_COLOR、[変数名] フィールドに m_Background と入力し、[OK] をクリックします。

CDemoAtlCtrl コンストラクタの変更

ファイル DemoATLCtrl.h で CDemoATLCtrl コンストラクタを次のように変更して、メンバ変数 m_FileName および m_Background を初期化します。

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

setFileName メンバ関数の追加

このメンバ関数を DemoCtrl クラスに追加する必要があります。この関数には、新しいファイル名を示すタイプ const char* のパラメータが 1 つあります。その目的は、対応ファイルのロードです。

1. DemoCtrl::setFileName の宣言を、ファイル DemoCtrl.h に追加します。

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

2. DemoCtrl::setFileName の宣言を、ファイル DemoCtrl.cpp に追加します。

```
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();
}
```

setBackground メンバ関数の追加

このメンバ関数を DemoCtrl クラスに追加する必要があります。この関数には、新しい色を示すタイプ OLE_COLOR のパラメータが 1 つあります。その目的は、OLE_COLOR を IBM ILOG Views が解釈し、ビューを再描画できる IlvColor* に変更することです。

1. DemoCtrl::setBackground の宣言を、ファイル DemoCtrl.h に追加します。

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

2. DemoCtrl::setBackground の定義を、ファイル DemoCtrl.cpp に追加します。

```
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();
}
```

ウィザードで生成されたメンバ関数の変更

1. ウィザードによって生成された DemoATLCtrl.cpp ファイルにあるメンバ関数 CDemoATLCtrl::put_FileName および CDemoATLCtrl::get_FileName を、次のように変更します。

```
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. ウィザードによって生成された DemoATLCtrl.cpp ファイルにあるメンバ関数 CDemoATLCtrl::put_Background および CDemoATLCtrl::get_Background を、次のように変更します。

```
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;
    }
```

プロパティ・マップ・マクロにコード行を追加する

次の行を DemoATLCtrl.h ファイルの BEGIN_PROP_MAP/END_PROP_MAP マクロ宣言ブロックに追加します。

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

CDemoATLCtrl::OnCreate メンバ関数の変更

ファイル DemoATLCtrl.h のメンバ関数 CDemoATLCtrl::OnCreate を変更し、対応プロパティが定義される時点でファイル名および / または背景色を設定します。

```
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;
}
```

プロパティのテスト

Internet Explorer または Visual Basic のいずれかを使用して、プロパティをテストできます。

Internet Explorer では、ファイル DemoATLCtrl.html を変更して、JavaScript 言語が使用できます。たとえば VBScript による例の場合は、*ステップ 4: ActiveX コントロールへカスタム・イベントを追加する* のプロパティのテスト を参照してください。

ここでは FileName プロパティをテストするため、3 つの定義済み選択肢を持つコ

ンボ・ボックスを使用します。

> ***メモ:*** *この例では、*IBM ILOG Views *データ・ファイルが* ILVHOME *または* ILVPATH *環境変数パスにインストールされていることを前提にしています。*

```
<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>
```

> ***メモ:*** *クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。*

Visual Basic でテストを実行する方法の例については、プロジェクト VB/testatl.vbp を参照してください。Visual Basic では、[プロパティ] タブで編集するか、または小さなプログラムの実行によって、FileName および Background プロパティを動的に変更できます。

ステップ 4: ActiveX コントロールへカスタム・イベントを追加する

このステップでは、カスタム・イベント (PointerPosition) を ActiveX コントロールに追加する方法を示します。ユーザがマウス・ポインタを移動するたびに、ビュー・インタラクタのメンバ関数 handleEvent がこのイベントを実行して、コンテナへ送ります。このイベントには 3 つのパラメータがあります。

- BSTR パラメータは、マウス・ポインタの下にオブジェクトがあれば、その名前を示します。
- OLE_XPOS_PIXELS および OLE_YPOS_PIXELS パラメータは、IlvManagerView に対するマウスの相対的位置を示します。

次の処理を行います。

- 新しいカスタム・イベントの定義
- ファイル DemoATL.idl をコンパイルするか、プロジェクトを再構築する
- 接続ポイント・インターフェースの実装
- CONNECTION_POINT_ENTRY マクロのパラメータをチェックする
- DemoCtrl クラスのコンストラクタを変更する
- メンバ関数 CDemoATLCtrl::onCreate を変更する
- TrackInteractor クラスの定義
- m_Inter プライベート変数を DemoCtrl クラスに追加する
- DemoCtrl コンストラクタの m_Inter を初期化して、マネージャ経由でビューに付加する
- デストラクタを変更して、m_Inter に格納されたインタラクタを削除する
- プロパティのテスト

新しいカスタム・イベントの定義

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、_IDemoATLCtrlEvents インターフェースをマウスの右ボタンでクリックします。
2. 表示されるメニューから [メソッドの追加] を選択します。
3. ダイアログ・ボックスに次に示す値を入力します。[戻り型] には HRESULT、[メソッド名] には PointerPosition、パラメータには BSTR name、OLE_XPOS_PIXELS x、OLE_YPOS_PIXELS y。

   ID およびヘルプ文字列以外のアトリビュートは定義しません。

ファイル DemoATL.idl をコンパイルするか、プロジェクトを再構築する

接続ポイント・インターフェースの実装

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、CDemoATLCtrl をマウスの右ボタンでクリックします。
2. 表示されるメニューから [コネクション ポイントのインプリメント] を選択します

3. [コネクション ポイントのインプリメント] インターフェースで、_IDemoATLCtrlEvents インターフェースを選択して [OK] をクリックします。

CONNECTION_POINT_ENTRY マクロのパラメータをチェックする

ファイル DemoATLCtrl.h で、マクロ CONNECTION_POINT_ENTRY のパラメータが DIID__IDemoATLCtrlEvents であること、CONNECTION_POINT_MAP マクロのパラメータが IID__IDemoATLCtrlEvents でないことを確認します。

DemoCtrl クラスのコンストラクタを変更する

このコンストラクタを、CDemoATLCtrl クラスの参照が 2 番目のパラメータとして受け入れられるように変更します。CDemoATLCtrl クラスのフォワード宣言をファイル DemoCtrl.h に追加して、ファイル stdafx.h、DemoATL.h、DemoATLCtrl.h をファイル DemoCtrl.cpp にインクルードします。

> ***メモ:*** IBM ILOG Views *および ATL はいずれも* STRICT *マクロの定義を強制しますが、これらのファイルは、*IBM ILOG Views *ヘッダー・ファイルの前にインクルードする必要があります。なお ATL は、このマクロがすでに定義されているかどうかをチェックしません。*

メンバ関数 CDemoATLCtrl::onCreate を変更する

メンバ関数 CDemoATLCtrl::OnCreate (ファイル DemoATLCtrl.h) で、DemoCtrl コンストラクタ呼び出しの 2 番目のパラメータとして、*this を追加します。

TrackInteractor クラスの定義

ファイル DemoCtrl.cpp 内で、IlvViewManagerInteractor から派生する新しいクラス TrackInteractor を定義します。

```
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;
};
```

m_Inter プライベート変数を DemoCtrl クラスに追加する

この m_Inter 変数は、タイプ TrackInteractor* である必要があります。

この変数を追加するためには、*ステップ3: ActiveX コントロールへカスタム・プロパティを追加する*で説明したように、ウィザードが使用できます。TrackInteractor クラスのフォワード宣言を、DemoCtrl.h ファイルにインクルードすることを忘れないでください。

DemoCtrl コンストラクタの m_Inter を初期化して、マネージャ経由でビューに付加する

```
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());
}
```

デストラクタを変更して、m_Inter に格納されたインタラクタを削除する

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

}
```

プロパティのテスト

Internet Explorer または Visual Basic のいずれかを使用して、プロパティをテストできます。

Internet Explorer では、ファイル DemoATLCtrl.htm を変更して、VBScript 言語が使用できます。

関数 DemoATLCtrl_PointerPosition が、PointerPosition イベント実行のたびに呼び出されます。この関数は、マウス・オーバーしているオブジェクトがあればその座標と、定義されていればその名前を表示します。

```
<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>
```

メモ: *クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。*

Visual Basic でテストを実行する方法の例については、プロジェクト `VB/testatl.vbp` を参照してください。関数 `DemoATLCtrl1_PointerPosition` が、`PointerPosition` イベント実行のたびに呼び出されます。

MFC ライブラリで ActiveX コントロールを作成する

このチュートリアルには、4 つのステップがあります。

- ステップ 1: プログラムの準備
- ステップ 2: MFC で ActiveX コントロールを作成する
- ステップ 3: ActiveX コントロールへカスタム・プロパティを追加する
- ステップ 4: ActiveX コントロールへカスタム・イベントを追加する

第 2 ステップで作成する基本 ActiveX コントロールは、このチュートリアルがさらに進行するに従って、より多くの機能を統合するために強化されます。

> ***メモ:*** *ウィザードによって生成されたファイルに含まれる次のコメントは、*IBM ILOG Views *用に変更または追加されたコード行を示します。*
> ```
> // Added for IBM ILOG Views
> // End Added for IBM ILOG Views
> ```

警告:

このチュートリアルを開始する前に、ActiveX コントロールの構築に当たっては、いくつかの問題が起きる可能性のあることをあらかじめご承知ください。たとえばある理由で、`_DllMain`、`new`、`delete` のような記号は、しばしばリンク時に定義されることがあります。このチュートリアルで提供されるコードにも、IBM ILOG Views ライブラリにもこれらの記号に関する定義は一切含まれません。この問題は、次のファイル間の競合が原因となっているものと考えられます。すなわち、`nafxcw.lib` (MFC によって使用される)、`libcmt.lib` (C ランタイム・ライブラリによって使用される)、`libcpmt.lib` (C++ ランタイム・ライブラリによって使用される) 間の競合です。

ただし、次に示すように、この問題を回避する方法があります。

1. ファイル `DemoMFC.cpp` および `DemoMFCCtrl.cpp` の `DemoCtrl` クラスを参照する行すべての前に、コメント文字を挿入します。
2. プロジェクト内のファイル `ctrl.cpp` および `DemoCtrl.cpp` を、非アクティブに設定します。
3. ActiveX コントロールを構築します。
4. ステップ 1 で挿入したコメント文字を削除します。
5. ファイル `ctrl.cpp` および `DemoCtrl.cpp` をアクティブに再設定します。
6. ActiveX コントロールを再構築します。

以上の手順に従えば、この問題は回避できます。手順をより簡単にする場合は、コメントする行をソース・ファイル内で、次の `if/endif` マクロで囲みます。

```
#if define(USE_VIEWS)

#endif
```

このマクロを、`ctrl.h` ファイルで定義します。このマクロが定義するコード行の前にコメント文字を挿入し、必要に応じて削除します。

問題の原因が正確に特定でき次第、Microsoft に報告する予定です。

ステップ 1: プログラムの準備

このチュートリアルの出発点として使用する、サンプル・アプリケーションを準備するステップです。サンプルは最後に ActiveX コントロールに変換して、他のアプリケーション内で使用できます。このステップで必要な作業は基本的に、データをコードから分離して、トップ・ビュー (元のプログラムの場合) または親ウィンドウの一部となる子ビューとして、メインビューを作成できるようにすることです。

また、これらの変更によって、新しい機能を ActiveX コントロールに追加するのが簡単になります。変更されたファイルが、step1 ディレクトリにあります。

次の処理を行います。

- コードからデータを分離して、.ilv ファイルに配置する
- Ctrl クラスの初期化コードをカプセル化する
- リソース・ファイル (.rc) を ilv2data で作成する
- manager.cpp ファイルの変更

コードからデータを分離して、.ilv ファイルに配置する

.ilv ファイルに格納されたデータは、.cpp ファイルに格納された文字列としてのデータよりも変更しやすくなります。また、あらゆる依存性の問題が、IBM ILOG Views データ・ファイル・パッケージ化ツールの ilv2data によって解決できます。

結果は、step1\data\ctrl.ilv ファイルにあります。

アタッチメントを追加したため、メイン・ビューを一貫した方法でリサイズできます。また、オブジェクトに名前を付けたので、プログラムからより容易にアクセスできます。これらの機能強化はすべて、IBM ILOG Views Studio で行われました。

> ***メモ:*** *コードからのデータ分離は、このチュートリアルで不可欠というわけではありませんが、変更を容易にするためにお勧めします。*

Ctrl クラスの初期化コードをカプセル化する

初期化コードは、ActiveX コントロールの作成と対話に使用する Ctrl クラスにカプセル化されています。

このクラスは、step1\ctrl.h ファイルで宣言し、step1\ctrl.cpp ファイルで定義します。この cpp ファイルには、コールバック関数も格納します。

Ctrl が基本クラスです。IlvGadgetContainer から派生します。3 つのコンストラクタと 4 つのメンバ関数だけで構成され、そのうち 1 つはプライベートです。

- 最初のコンストラクタは、Ctrl オブジェクトをファイルの元のバージョンと同様にトップ・ビューとして作成します。step1\manager.cpp の main 関数で使用され、次の署名があります。

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

- 2 番目のコンストラクタは、Ctrl オブジェクトを子ビューとして作成します。次の署名があります。

```
Ctrl(IlvDisplay* display,
     IlvSystemView parent,
     const IlvRect& size,
```

```
        const char* filename = 0)
```

- 3 番目のコンストラクタは、Ctrl オブジェクトを既存のシステム・ビューで作成します。これは、ActiveX コントロールの作成に使用されます。次の署名があります。

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

- パブリック・メンバ関数 getManagerRectangle、getManagerView、getManager は、コントロールにロードされる各オブジェクトへのアクセスを提供します。. プライベート・メンバ関数 init はコントロールを初期化するために呼び出されます。つまり、データ・ファイルを読み込んだり、コールバックをオブジェクトに関連付けたりします。メンバ関数 PopupManager 内にあったコードのほとんどが、このクラスに移動しています。

リソース・ファイル (.rc) を ilv2data で作成する

ilv2data を使って、ファイル step1\data\ctrl.ilv および step1\data\browse.ilv が含まれる .rc ファイルを作成します。このリソース・ファイルをサンプル・プログラムで使うこともできますが、このチュートリアルでは後で ActiveX コントロールを構築するために使用します。

ilv2data の詳細は、IBM ILOG Views アプリケーションのパッケージ化 を参照してください。

manager.cpp ファイルの変更

ファイル step1\manager.cpp には、IlvDisplay のインスタンスと、新しい Ctrl クラスのインスタンスを作成する main 関数だけが含まれます。

```
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;
}
```

ステップ 2: MFC で ActiveX コントロールを作成する

このステップでは、*ステップ1: プログラムの準備* で定義した Ctrl クラスによって ActiveX コントロールを作成する方法について説明します。このステップの終了時に得られる ActiveX コントロールには、最小限度のインターフェースしかありません。

次の処理を行います。

- 新規プロジェクトの作成
- プロジェクトにファイルを追加する
- ウィザードで生成されたファイルを変更する
- CDemoMFCCtrl クラスに新しい Windows メッセージ・ハンドラを追加する
- CDemoMFCCtrl にプライベート・メンバ変数を追加する
- ウィザードで生成された DemoMFCCtl.h ファイルを変更する
- DemoMFC.rc ファイルの変更
- プロジェクト設定 (DemoMFC.dsp) の変更
- これで ActiveX コントロールを構築してテストできます。

新規プロジェクトの作成

1. Visual C++ で、[ファイル] メニューから [新規作成] を選択し、[プロジェクト] ページで MFC ActiveX ControlWizard を選んで、新規プロジェクトの作成を開始します。
2. [プロジェクト名] テキスト・フィールドに `DemoMFC` と入力し、他のオプションはそのままにします。
3. [OK] をクリックします。

   MFC ActiveX ControlWizard のステップ 1 が表示されます。
4. デフォルト値をそのままにして [次へ] をクリックします。

   MFC ActiveX ControlWizard のステップ 2 が表示されます。
5. 次のオプションをチェックします。[オブジェクトの挿入] ダイアログ・ボックスの [可視状態の時にアクティブにする] および [使用可能] をオンにします。

   [バージョン情報] チェック・ボックスをオンにすることもできますが、これはオプションです。

   他のプロパティはチェックなしのままにします。ウィンドウが、コントロールのサブクラスになっていないことを確認します。また、[ActiveX 機能の詳細] ダイアログ・ボックスで、[ウィンドウなし] の有効がオフになっていることも確認します。
6. [終了] と [OK] をクリックします。

プロジェクトにファイルを追加する

1. ファイル `ctrl.cpp` をプロジェクトに追加します。
2. `DemoCtrl.h` ファイルを作成して、プロジェクトに追加します。

このファイルには、それに続く DemoCtrl クラスの宣言を含む必要があります。このクラスがコントロールを初期化します。また、ウィザードによって生成されたコードと、IBM ILOG Views コードおよび / または Ctrl クラス間のインターフェースを提供します。

```
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. DemoCtrl.cpp ファイルを作成して、プロジェクトに追加します。

このファイルはインラインで宣言されていない DemoCtrl クラスのメンバ関数と、スタティック・クラス変数 _Display を定義します。

◆ コンストラクタ DemoCtrl(HWND parent) は、対応する Ctrl クラスのコンストラクタを呼び出します。

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

◆ スタティック・メンバ関数 InitDisplay は、IlvDisplay のインスタンスを作成して、_Display に格納します。

```
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;
}
```

◆ スタティック関数 CleanDisplay は、_Display に格納された IlvDisplay のインスタンスを解放します。

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

ウィザードで生成されたファイルを変更する

ウィザードで生成された DemoMFC.cpp ファイルを変更して、IlvDisplay クラスのインスタンスを作成してから、削除します。DemoCtrl クラスのスタティック・メンバ関数を、次の手順で使用します。

1. ヘッダー・ファイル DemoCtrl.h をインクルードします。
2. CDemoMFCApp::InitInstance メンバ関数内で、次のコード行を bInit テストに追加します。

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

3. CDemoMFCApp::ExitInstance メンバ関数の最初に、次のコード行を追加します。

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

CDemoMFCCtrl クラスに新しい Windows メッセージ・ハンドラを追加する

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、プロジェクト・フォルダ DemoMFC クラスを展開します。
2. CDemoMFCCtrl をマウスの右ボタンでクリックします。
3. 表示されるメニューから [Windows メッセージ・ハンドラの追加] を選択して、WM_CREATE をダブルクリックします。

CDemoMFCCtrl にプライベート・メンバ変数を追加する

1. DemoMFC クラスを展開して、CDemoMFCCtrl を選択します。
2. マウスの右ボタンを押して、[メンバ変数の追加] を選択します。
3. [変数] 入力フィールドに、DemoCtrl* とタイプします。
4. [変数名] フィールドに m_Ctrl と入力し、[OK] をクリックします。

ウィザードで生成された DemoMFCCtl.h ファイルを変更する

ウィザードで生成されたファイル DemoMFCCtl.h と DemoMFCCtl.cpp を変更して、アプリケーションで使用する DemoCtrl クラスのインスタンスを、次の手順で作成します。

1. DemoCtrl クラスのフォワード宣言を、ファイル DemoMFCCtl.h に追加します。
2. ファイル DemoMFCCtl.cpp に、ヘッダー・ファイル DemoCtrl.h をインクルードします。
3. コンストラクタ (ファイル DemoMFCCtl.cpp) 内の変数 m_Ctrl を 0 に初期化します。

```
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. 次に示すコード行を、メンバ関数 CDemoMFCCtrl::OnCreate (ファイル DemoMFCCtl.cpp) に追加します。

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

DemoMFC.rc ファイルの変更

次の行を、ファイル DemoMFC.rc の最後に追加します。

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

プロジェクト設定 (DemoMFC.dsp) の変更

1. [プロジェクト] メニューで [設定] を選択します。
2. [プロジェクトの設定] ダイアログ・ボックスで、[Win32Debug] を選択して、[C/C++] タブをクリックします。
3. [カテゴリ] ドロップダウン・リストで [一般]、または [プリプロセッサ] オプションを選択し、プリプロセッサの定義 _DEBUG を NDEBUG に変更します。
4. [すべての構成] を選択して、[一般] タブをクリックします。
5. [Microsoft Foundation Classes] ドロップダウン・リストから [MFC のスタティック ライブラリを使用] を選択します。
6. [C++] タブを有効にして、[カテゴリ] ドロップダウン・リストで [一般]、または [プリプロセッサ] オプションを選択し、[プリプロセッサ定義] テキスト・フィールドに ILVSTD を追加します。

7. [カテゴリ] ドロップダウン・リストで C++ 言語を選んでから、[例外処理を有効にする]、[ランタイム タイプ情報 (RTTI) を有効にする] オプションを選択します。

8. [カテゴリ] ドロップダウン・リストで [コード生成] を選択してから、[使用するランタイム ライブラリ] の [マルチスレッド] を選択します。

9. [カテゴリ] ドロップダウン・リストで [プリプロセッサ] を選択し、[インクルード ファイルのパス] テキスト・フィールドに次の行を入力します。

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

   ここでは、IBM ILOG Views が `C:\ilog\views\` にインストールされていることを前提にしています。

10. [カテゴリ] ドロップダウン・リストの [プリコンパイル済みヘッダー] を選んでから、ファイル `ctrl.cpp` および `DemoCtrl.cpp` について [プリコンパイル済みヘッダーを使用しない] オプションを選択します。

    IBM ILOG Views の初期化メカニズムのため、ほとんどの IBM ILOG Views ヘッダーはプリコンパイルできません。

11. [リンク] ページの [カテゴリ] ドロップダウン・リストで [入力] を選んでから、[追加ライブラリ パス] テキスト・フィールドに、IBM ILOG Views ライブラリへのパスを追加します。

    デフォルトのパスは `c:\ilog\lib\msvc6_mta` です。

12. [カテゴリ] ドロップダウン・リストで [一般] または [入力] を選択してから、IBM ILOG Views ライブラリ (すなわち `winviews.lib`、`views.lib`、`ilvgadgt.lib`) および、`wsock32.lib`、`imm32.lib` ライブラリを追加します。

これでActiveX コントロールを構築してテストできます。

html テスト・ファイルを、下記のように作成します。

```
<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>
```

メモ: クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。

ステップ 3: ActiveX コントロールへカスタム・プロパティを追加する

このステップでは、ActiveX コントロールにファイル名と背景色を追加する方法について説明します。ファイル名は、マネージャによって読み込まれるファイルの名前です。デフォルトでこのファイルは `data/browse.ilv` で、これはステップ 2 において `.rc` ファイルにインクルードしました。また背景色は、マネージャ・ビューの背景色になります。

- ファイル名プロパティの追加
- 背景色プロパティの追加
- 新しいプライベート・メンバ変数の追加
- CDemoMFCCtrl コンストラクタの変更
- setFileName メンバ関数の追加
- setBackground メンバ関数の追加
- ウィザードで生成されたメンバ関数の変更
- CDemoMFCCtrl::OnCreate メンバ関数の変更
- プロパティのテスト

ファイル名プロパティの追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、`_DDemoMFC` をマウスの右ボタンでクリックして、[プロパティの追加] を選択します。
2. [外部名] フィールドに、`FileName` と入力します。
3. [Get/Set メソッド] オプションを選択します。
4. [プロパティのタイプ] ドロップダウン・リストから、`BSTR` を選択します。
5. [パラメータ] リストが空欄であることを確認します。

背景色プロパティの追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、`_DDemoMFC` をマウスの右ボタンでクリックして、[プロパティの追加] を選択します。
2. [外部名] フィールドに、`Background` と入力します。
3. オプション [Get/Set メソッド] を選択します。

4. [プロパティのタイプ] ドロップダウン・リストから、OLE_COLOR を選択します。
5. [パラメータ] フィールドは空欄のままにします。

新しいプライベート・メンバ変数の追加

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、CDemoMFCCtrl をマウスの右ボタンでクリックします。
2. 表示されるメニューから [メンバ変数の追加] を選択します。
3. [変数タイプ] フィールドに CString、[変数名] フィールドに m_FileName と入力し、[OK] をクリックします。
4. [メンバ変数の追加] を再び選択します。
5. [変数タイプ] フィールドに OLE_COLOR、[変数名] フィールドに m_Background と入力し、[OK] をクリックします。

CDemoMFCCtrl コンストラクタの変更

ファイル DemoMFCCtl.cpp で CDemoMFCCtrl コンストラクタを次のように変更して、メンバ変数 m_FileName および m_Background を初期化します。

```
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.
}
```

setFileName メンバ関数の追加

このメンバ関数を DemoCtrl クラスに追加する必要があります。この関数には、新しいファイル名を示すタイプ const char* のパラメータが 1 つあります。その目的は、対応ファイルのロードです。

1. DemoCtrl::setFileName の宣言を、ファイル DemoCtrl.h に追加します。

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

2. DemoCtrl::setFileName の宣言を、ファイル DemoCtrl.cpp に追加します。

   ```
   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();
}
```

setBackground メンバ関数の追加

このメンバ関数を DemoCtrl クラスに追加する必要があります。この関数には、新しい色を示すタイプ OLE_COLOR のパラメータが 1 つあります。その目的は、OLE_COLOR を IBM ILOG Views が解釈し、ビューを再描画できる IlvColor* に変更することです。

1. DemoCtrl::setBackground の宣言を、ファイル DemoCtrl.h に追加します。

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

2. DemoCtrl::setBackground の定義を、ファイル DemoCtrl.cpp に追加します。

   ```
   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();
   }
   ```

ウィザードで生成されたメンバ関数の変更

1. ウィザードによって生成された DemoMFCCtl.cpp ファイルにあるメンバ関数 CDemoMFCCtrl::SetFileName および CDemoMFCCtrl::GetFileName を、次のように変更します。

   ```
   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. ウィザードによって生成された DemoMFCCtl.cpp ファイルにあるメンバ関数 CDemoMFCCtrl::SetBackground および CDemoMFCCtrl::GetBackground を、次のように変更します。

```
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;
}
```

CDemoMFCCtrl::OnCreate メンバ関数の変更

ファイル DemoMFCCtl.h のメンバ関数 CDemoMFC::OnCreate を変更し、対応プロパティが定義される時点でファイル名および / または背景色を設定します。

```
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;
}
```

プロパティのテスト

Internet Explorer または Visual Basic のいずれかを使用して、プロパティをテストできます。

Internet Explorer では、ファイル `test.htm` を変更して、JavaScript 言語が使用できます。たとえば VBScript による例の場合は、ステップ 4: ActiveX コントロールへカスタム・イベントを追加する のプロパティのテスト を参照してください。

ここでは `FileName` プロパティをテストするため、3 つの定義済み選択肢を持つコンボ・ボックスを使用します。

> ***メモ:*** *この例では、*IBM ILOG Views *データ・ファイルが* `ILVHOME` *または* `ILVPATH` *環境変数パスにインストールされていることを前提にしています。*

```
<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>
```

> ***メモ:*** *クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。*

Visual Basic でテストを実行する方法の例については、プロジェクト VB/testMFC.vbp を参照してください。Visual Basic では、[プロパティ] タブで編集するか、または小さなプログラムの実行によって、FileName および Background プロパティを動的に変更できます。

ステップ 4: ActiveX コントロールへカスタム・イベントを追加する

このステップでは、カスタム・イベント (PointerPosition) を ActiveX コントロールに追加する方法を示します。ユーザがマウス・ポインタを移動するたびに、ビュー・インタラクタのメンバ関数 handleEvent がこのイベントを実行して、コンテナへ送ります。このイベントには 3 つのパラメータがあります。

- BSTR パラメータは、マウス・ポインタの下にオブジェクトがあれば、その名前を示します。
- OLE_XPOS_PIXELS および OLE_YPOS_PIXELS パラメータは、IlvManagerView に対するマウスの相対的位置を示します。

次の処理を行います。

- 新しいカスタム・イベントの定義
- ファイル DemoMFC.odl をコンパイルするか、プロジェクトを再構築する
- メンバ関数 CDemoMFCCtrl::onCreate を変更する
- TrackInteractor クラスの定義
- m_Inter プライベート変数を DemoCtrl クラスに追加する
- クラス TrackInteractor を、クラス CDemoMFCCtrl のフレンドとして宣言する (ファイル DemoMFCCtl.h 内)。
- DemoCtrl コンストラクタの m_Inter 変数を初期化して、マネージャ経由でビューに付加する
- デストラクタを変更して、m_Inter に格納されたインタラクタを削除する
- プロパティのテスト

新しいカスタム・イベントの定義

1. [プロジェクト ワークスペース] ウィンドウの [ClassView] タブで、_DDemoMFCEvents インターフェースをマウスの右ボタンでクリックします。
2. 表示されるメニューから [イベントの追加] を選択します。
3. ダイアログ・ボックスに次に示す値を入力します。PointerPosition を [外部名] に、BSTR name、OLE_XPOS_PIXELS x、OLE_YPOS_PIXELS y をパラメータにします。

ファイル DemoMFC.odl をコンパイルするか、プロジェクトを再構築する

DemoCtrl クラスのコンストラクタを変更する

このコンストラクタを、CDemoMFCCtrl クラスの参照が 2 番目のパラメータとして受け入れられるように変更します。CDemoMFCCtrl クラスのフォワード宣言をファイル DemoCtrl.h に追加して、ファイル stdafx.h、DemoMFC.h、DemoMFCCtl.h をファイル DemoCtrl.cpp にインクルードします。

> ***メモ:*** *これらのファイルは* IBM ILOG Views *ヘッダー・ファイルの前に含める必要があります。*

メンバ関数 CDemoMFCCtrl::onCreate を変更する

メンバ関数 CDemoMFCCtrl::OnCreate (ファイル DemoMFCCtl.cpp) で、DemoCtrl コンストラクタ呼び出しの 2 番目のパラメータとして、*this を追加します。

TrackInteractor クラスの定義

ファイル DemoCtrl.cpp 内で、IlvViewManagerInteractor から派生する新しいクラス TrackInteractor を定義します。

```
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;
};
```

m_Inter プライベート変数を DemoCtrl クラスに追加する

この m_inter 変数は、タイプ TrackInteractor* である必要があります。

この変数を追加するためには、*ステップ3: ActiveX コントロールへカスタム・プロパティを追加する*で説明したように、ウィザードが使用できます。
TrackInteractor クラスのフォワード宣言を、DemoCtrl.h ファイルにインクルードすることを忘れないでください。

クラス TrackInteractor を、クラス CDemoMFCCtrl のフレンドとして宣言する (ファイル DemoMFCCtl.h 内)。

DemoCtrl コンストラクタの m_Inter 変数を初期化して、マネージャ経由でビューに付加する

```
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());
}
```

デストラクタを変更して、m_Inter に格納されたインタラクタを削除する

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

}
```

プロパティのテスト

Internet Explorer または Visual Basic のいずれかを使用して、プロパティをテストできます。

Internet Explorer では、ファイル test.htm を変更して、VBScript 言語が使用できます。

関数 DemoMFCCtrl_PointerPosition が、PointerPosition イベント実行のたびに呼び出されます。この関数は、マウス・オーバーしているオブジェクトがあればその座標と、定義されていればその名前を表示します。

```
<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>
```

> ***メモ:*** *クラス ID が生成されるので、ご使用ファイルのクラス ID がこの例とは異なる場合があります。*

Visual Basic でテストを実行する方法の例については、プロジェクト `VB/testMFC.vbp` を参照してください。関数 `DemoMFCCtrl1_PointerPosition` が、`PointerPosition` イベント実行のたびに呼び出されます。

章

2

ActiveX コントロールをグラフィック・オブジェクトとしてIBM ILOG Views アプリケーションに追加する

このチュートリアルの目的は、ActiveX™ コントロールをグラフィック・オブジェクトとして、IBM® ILOG® Views アプリケーションに追加する方法を示すことです。ここで使用されるサンプルは、`IlvGrapher` を中心に作成された、非常に単純なアプリケーションです。

このグラフィック・オブジェクトは ATL ライブラリ上に構築されます。より具体的には、ATL ライブラリのバージョン 4.0 以降で使用できる、ATL コントロールのホスト API 上です。詳細については、Microsoft のマニュアルを参照してください。

現在のバージョンは、Visual C++ 6.0 および Visual C++7.0 でのみテスト済みです。

このチュートリアルで学習する前提条件として、IBM ILOG Views および C++ の知識が必要です。COM および ATL ライブラリの知識は必須ではありませんが、このチュートリアルのテクニカルな側面を理解する助けになります。

例を使った IBM ILOG Views アプリケーションの説明

このチュートリアルは、以下の手順で構成されています。

- *ステップ1: IlvGraphicCOMAdapter オブジェクトの作成*

- *ステップ2: オブジェクト・インターフェースの取得*
- *ステップ3: オブジェクト・インタラクタの追加*
- *ステップ4: ブラウザの追加*
- *ステップ5: コントロールのプロパティ値を取得する*

この例は、`IlvGrapher` 上に構築されます。

このクラスの定義は、ファイル include/ActiveXGraphicAdapterApp.h、include/ActiveXGraphicAdapter.h、include/makenode.h にあります。メンバ関数の定義は、src/ActiveXGraphicAdapterApp.cpp、src/ActiveXGraphicAdapter.cpp、および src/specific.cpp にあります。データは、ファイル data/ActiveXGraphicAdapter.ilv にあります。rc ファイルの src/ActiveXApp.rc は、`ilv2data` で生成されました。データのパスを心配することなく、このファイルを使用できます。

ファイル src/ActiveXGraphicAdapterApp.cpp および src/ActiveXGraphicAdapter.cpp は、`ivstudio` によって生成され、本アプリケーションの目的に合うように修正されています。本アプリケーションのこの部分についての詳細は、IBM ILOG Views の他のマニュアルを参照してください。ファイル src/specific.cpp には、このチュートリアルにおけるテーマの主要部分が含まれます。. このファイルには他の関数とともに、`ActiveXGraphicAdapter::specificInitialization` 関数が含まれます。本チュートリアルの作業のほとんどは、この関数内で実行します。

本アプリケーションには、1 つのメニューとスクロール・グラフィック矩形が含まれます。メニューでは、次の処理が行えます。

- グラファーを消去する ([ファイル]/[新規作成])
- ファイルをグラファーにロードする ([ファイル]/[開く])
- グラファーの内容をファイルに格納する ([ファイル]/[保存]、[ファイル]/[名前を付けて保存])
- アプリケーションを終了する ([ファイル]/[終了])
- 新規ノード作成のため、グラファーにインタラクタを付加する ([インタラクタ]/[ノード作成インタラクタ])
- 新規リンク作成のため、グラファーにインタラクタを付加する ([インタラクタ]/[リンク作成インタラクタ])
- 選択インタラクタをグラファーに付加する ([インタラクタ]/[選択インタラクタ])
- オブジェクト・インタラクタを使用するため、表示インタラクタがあればすべて削除する ([インタラクタ]/[オブジェクト インタラクタ])
- ヘルプを印刷する

メモ: 本チュートリアルの code サブディレクトリでは、この概要のサンプルが step 1 に、ステップ 1 のサンプルが step 2 に、ステップ 2 のサンプルが step 3 に、というように、1 つずつずれて入っています。

ステップ 1: IlvGraphicCOMAdapter オブジェクトの作成

このステップでは、`IlvGraphicCOMAdapter` オブジェクトを作成します。このグラフィック・オブジェクトは ATL コントロールのホスト API を使用するため、オブジェクトの識別子を次の 3 つの方法で指定できます。オブジェクトの作成を、次から指定できます。

- CLSID - デフォルト
- ProgID - SECOND_VERSION として定義
- URL - THIRD_VERSION として定義

デフォルトを使用しない場合は、上記の定義後にファイル specific.cpp を再コンパイルするだけで、テストできます。

この例では、Microsoft Calendar コントロールを使用します。コンピュータで Calendar コントロールが使用できない場合は、`DefaultIdentifier` の値を変更します。

オブジェクトの作成は非常に簡単です。

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

ここで:

- `size` は、オブジェクトのサイズです。
- `identifier` は、上で説明したコントロールの識別子です。識別子に付加できる値の例を次に示します。
  - CLSID: "{8E27C92B-1264-101C-8A2F-040224009C02}"
  - ProgID: "MSCAL.Calendar"
  - URL: "http://www.ilog.com/"

クラス `IlvGraphicAdapter` の定義を取得して、ライブラリ `Ilvcomstat.lib` または `ilvcomdyn.lib` に追加するため、次のヘッダー・ファイルをインクルードする必要もあります。

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

また、共通 ActiveX ダイアログの使用を可能にするため、`oledlg.h` ファイルをインクルードする必要もあります。

次のコードが、MakeNodeInteractor クラスの doIt メンバ関数で呼び出されます。

```
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()));
}
```

このクラスを使用可能にするには、次に示すヘッダー・ファイルをインクルードしなければなりません。

- MakeNodeInteractor のメンバ関数 doIt で使用されるクラス IlvAddNodeCommand の定義を含む ilviews/grapher/commands.h。詳細については、マネージャ のセクションを参照してください。
- クラス MakeNodeInteractor の定義を含む makenode.h。

MakeNodeInteractor クラスの定義は次のようになります。

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

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

ステップ 2: オブジェクト・インターフェースの取得

このステップでは、IOleObject インターフェース - 挿入できるすべてのコントロールに存在します - を取得して、Ctrl-i に連結されるアクセラレータを介して、ユーザのタイプ名を印刷します。

IlvGraphicCOMAdapter クラスでは、2 つのインターフェースだけを直接使用します。すなわち、IOleObject および IViewObject です (ATL API ではさらに多くのインターフェースを使用します)。ただし、このコントロールによって提供される機能を使用するため、新しいインターフェースを要求することは可能です。それを行うには、IlvGraphicCOMAdapter::queryInterface メンバ関数を使用します。この関数は単に、同じパラメータおよび同じ戻り値を持つ COM 関数 QueryInterface のカプセル化したものです。

以下にコードを示します。

```
// --------------------------------------------------------------------------
// 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;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();
                   }
              }
           }
        }
    }
```

```
}
```

UNICODE で文字列を印刷するため、次のオペレータも定義します。

```
// ------------------------------------------------------------------------
// 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;
}
```

ここからわかるように、このコードの大半は純粋な COM のコードです。このコードではスマート・ポインタを使用して、インターフェースのリリースに関する懸念を回避します。

ここでは、クラス IlvCOut を使用します。したがって、ヘッダー・ファイル ilviews/base/error.h のインクルードが必要です。

ステップ 3: オブジェクト・インタラクタの追加

このステップでは、オブジェクト・インタラクタ IlvGraphicComAdapterInteractor の使用法を示します (このクラスは、ヘッダー• ファイル ilviews/windows/comgint.h で定義されます)。たとえグラフィック・オブジェクトへの変換時でも、これによってコントロールと (いくらか制約があるにせよ) 対話できます。実際の作業は、IlvGraphicComAdapterInteractor のインスタンスを、関数 AttachInteractorAccelerator によって実行される IlvGraphicCOMAdapter オブジェクトに付加するだけです。

```
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);
    }
  }
}
```

IlvGraphicComAdapterInteractor オブジェクトは、関数のパラメータとして与えられ、関数 ActiveXGraphicAdapter::specificInitialization に割り当てられています。

```
_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);
```

ステップ 4: ブラウザの追加

このステップでは、ブラウザの追加方法を示します。コントロールの識別子を取得するため、コントロール • チューザが使用できます。ここでは、show 関数から識別子として返される値を使用するだけです。同様にファイル名を取得するためには、IlvFileBrowser::show() 関数の戻り値を使用します。

次に示すのは、メンバ関数 IlvMakeNodeInteractor::doIt の新コードです。

```
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()));
}
```

IlvControlBrowser クラスは ilviews/windows/cbrowser.h ヘッダー・ファイルで定義されます。このクラスを使用できるためには、ライブラリ oledlg.lib の追加が必要です。

ステップ 5: コントロールのプロパティ値を取得する

このステップでは、コントロールのプロパティ値を取得する方法について示します。このコードは、Calendar コントロールのプロパティ Day に基づいています。別のプロパティや別のコントロールを選ぶ場合は、プロパティの名前とタイプを変更する必要があります。

step3 のコードを少しだけ変更して、コントロールのタイプが Calendar である場合も、GetTypeAccelerator でプロパティ Day の値を表示できるようにします。

そのために、次の行：

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

を、下記の第 1 行に置き換えます。

```
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";
```

コントロールのタイプが Calendar であることを確認したら、その名前がプロパティ (この例では Day) の名前である IlvValue オブジェクトがインスタンス化されます。次に、このプロパティの値が、(IBM ILOG Views の任意の値オブジェクトの場合と同様に) IlvGraphicCOMAdapter オブジェクトの queryValue の呼び出しによって取得されます。さらに、その値が IlvCOut によって表示されます。

このように、プロパティにアクセスしてその値を取得するのはきわめて容易です。また、その値を変更するのも簡単です。たとえば、Day プロパティを 5 に設定するためには、次のように記述します。

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

詳細については、IlvValue および IlvGraphic の参照ページをご覧ください。

索引