IBM ILOG Views

Application Framework 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

このチュートリアルでは、IBM® ILOG® Views の Application Framework パッケージに基づいてビットマップ・エディタを作成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。

- ユーザが行うべきコード抽出、ファイル名およびエントリは `courier` 書体で記載されます。

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。
- 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。

```
virtual IlvClassInfo* getClassInfo() const;

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

章

1

ビットマップ・エディタ・アプリケーション

このチュートリアルでは、IBM® ILOG® Views の Application Framework パッケージに基づいてビットマップ・エディタを作成する方法について説明します。ここでは、よく知られたビットマップ・エディタの基本的な機能を再作成します。ビットマップ・エディタのユーザ・インターフェースを設定するための Application Framework Wizard の使用方法およびドキュメント / ビュー・アーキテクチャの利点を活かすための API の使用方法を学びます。

このチュートリアル用のコードは、`ILVHOME/doc/tutorials/appframe/bitmaped` にあります。`ILVHOME` は、IBM ILOG Views がインストールされたルート・ディレクトリです。

ビットマップ・エディタの主要機能は、以下のとおりです。

- ビットマップ (PNG、BMP、JPEG ファイル) の読み込み
- 新規ビットマップの作成
- ビットマップ (PNG、BMP、JPEG ファイル) の書き込み
- 描画コマンドを用いてのビットマップの修正
- 描画コマンドの [元に戻す] および [やり直す]
- ビットマップの操作 - 拡大および縮小機能

- ドキュメント / ビュー・アーキテクチャ - この機能により、同一文書の異なる複数ビューを得ることができます。ビューで行われた修正は文書によって処理され、最後に加えられた修正の文書を示すすべてのビューに通知されます。

完成したアプリケーションは次のようになります。

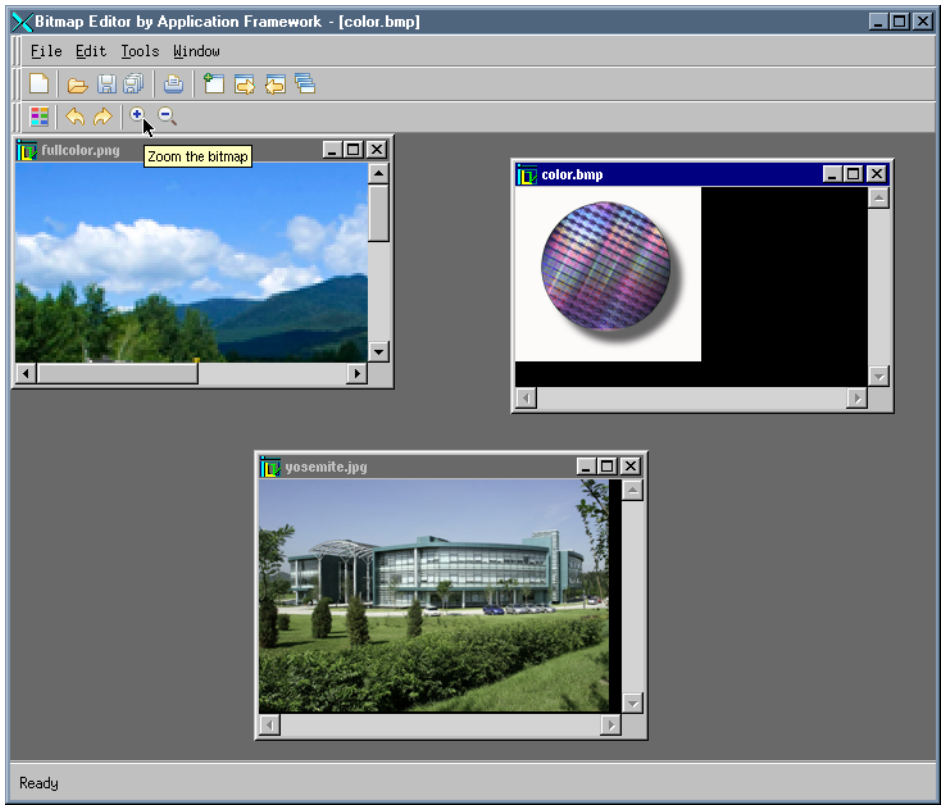


図 1.1 *完成したビットマップ・エディタ・アプリケーション*

このチュートリアルは、次の 5 つの手順から構成されています。

- ステップ 1:Wizard を使ったベース・ファイル生成
- ステップ 2: 文書とビューの実装
- ステップ 3: ビットマップの変更および保存
- ステップ 4:IBM ILOG Views Studio で行われたダイアログの挿入
- ステップ 5: ズーム・コマンドの追加

ステップ 1:Wizard を使ったベース・ファイル生成

この最初の手順では、Wizard (Application Framework Editor) を使用して、最終アプリケーションのベース、または基盤を作成する方法について説明します。

次のタスクについて説明します。

- オプション・ファイル (.odv ファイル) の作成と設定
- C++ コードおよびリソース・ファイルの生成

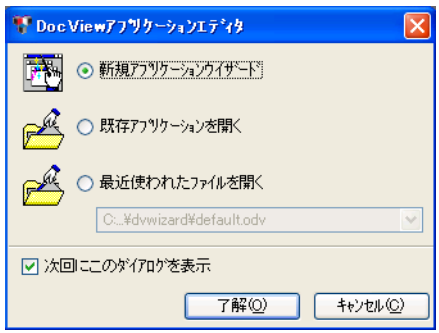
オプション・ファイル (.odv ファイル) の作成と設定

.odv ファイルを作成する方法

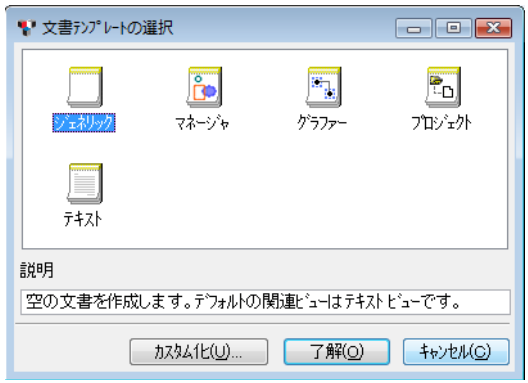
1. Application Framework Editor を起動します。

   Application Framework Editor を起動するには、<ILVHOME>/bin/<SYSTEM>/dvwizard を実行します。ここで、<ILVHOME> は IBM® ILOG® Views のインストール・ディレクトリ、<SYSTEM> はプラットフォーム (たとえば x86_.net2008_9.0 や ultrasparc32_10_11 など) を指します。パスに IBM ILOG Views ライブラリが含まれているかどうかを確認します。

2. 新規アプリケーション・ウィザードを選択し、[了解] をクリックして新規アプリケーションを作成します。



3. [文書テンプレートの選択] パネルから、[ジェネリック] を選択します、

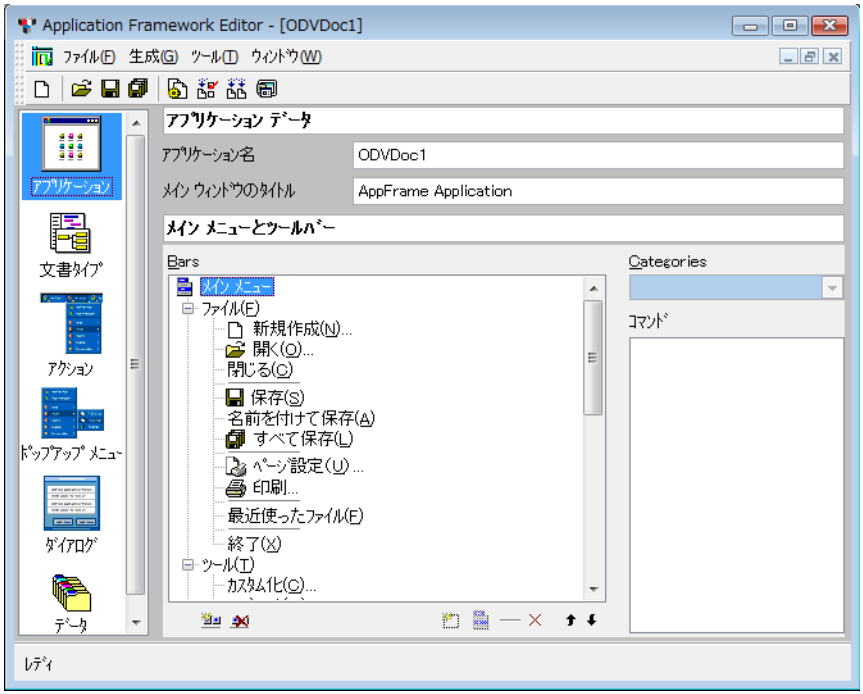


いくつかの定義済み文書タイプが使用できます。文書の各タイプは、そのデータ操作用の便利なメソッドを定義し、特定ビューに関連付けられます。

たとえば、マネージャの文書タイプは、`IlvManager` オブジェクトに挿入されている `IlvGraphic` オブジェクトを扱います。これは、グラフィック・エディタ開発に特に適しています。

一般文書タイプでは、文書タイプを何も想定しません。ほとんどのアプリケーションには、このオプションを選択します。

Application Framework Editor は、新規 `.odv` ファイルを開き、複数のウインドウを表示します。左の領域で、編集するアプリケーション・エンティティ (アプリケーション、文書タイプ、アクション、ポップアップ・メニュー、およびデータ) を選択します。右の領域では、選択したエンティティのパラメータの設定を行います。

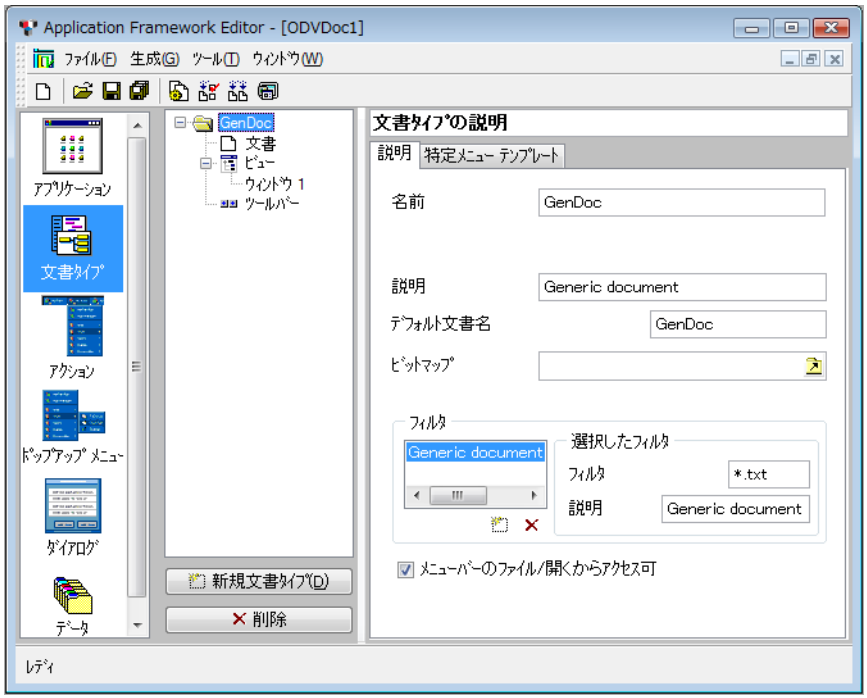


アプリケーション・パラメータの入力

1. アプリケーション名を `ODVDoc1` から `BitmapEditor` に変更します。デフォルトでは、この名前をディレクトリとプロジェクト名の作成に使用します。
2. メイン・ウィンドウのタイトルを `Bitmap Editor by Application Framework` に変更します。

コード生成に使用する文書パラメータの指定

1. ウインドウの左の領域から文書タイプをクリックします。



中央の領域は、ツリー内で、アプリケーション文書、それらのビューおよびツールバーを表します。右の領域は、ツリーで選択したオブジェクトの詳細設定です。

2. 中央の領域のツリーから、GenDoc を選択します。

   右の領域は、文書タイプの説明ページを表示します。

3. この説明から、名前フィールドを `BitmapEditor` に変更します。

   この文書名は、アプリケーションで使用します。

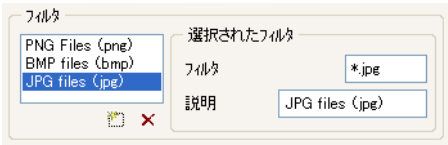
4. 説明フィールドを `Bitmap Editor Document` に変更します。

   このプロパティは、新規文書ダイアログ・ボックスで文書タイプの説明を表示するのに使用されます。

5. デフォルト文書名フィールドを `Bitmap` に設定します。

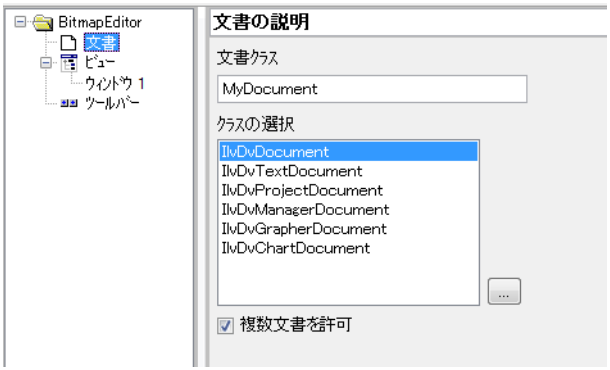
   このプロパティは、ユーザが保存する前に、新規文書にデフォルト名を割り当てるために使用されます。

6. フィルタを変更します。選択したフィルタは、*.png に変更されます。このプロパティは、新規文書をファイル・セレクタで開いたときに使用されます。*.png がデフォルト拡張子となります。ビットマップ・エディタで、PNG、BMP、JPEG ファイルを開くことができます。
7. 説明を PNG Files (.png) に設定します。
8. [フィルタの追加] ボタンをクリックして、BMP フィルタを追加します。
9. 選択したフィルタがフィルタ・フィールドで *.bmp に変更されます。
10. 説明を BMP Files (.bmp) に設定します。
11. [フィルタの追加] ボタンをクリックして、JPG フィルタを追加します。
12. 選択したフィルタがフィルタ・フィールドで *.jpg に変更されます。
13. 説明を JPG Files (.jpg) に設定します。



文書 C++ クラスの指定

1. 中央ウインドウのツリーから文書を選択します。



右の領域で、クラスに名前を付け、ビットマップ文書が継承することになる文書クラスを決定します。

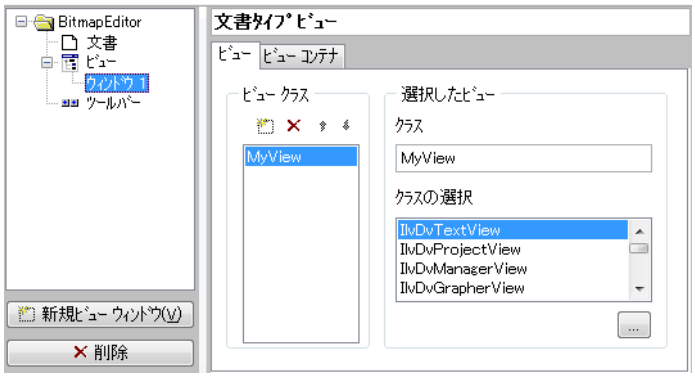
2. 文書クラス・フィールドを BitmapDocument に変更します。

文書クラスは、IlvDvDocument (すべての文書用のベース・クラス) から継承されます。BitmapDocument C++ クラスが生成されます。生成されたコードは後で完成します。

以上で、ビットマップ文書の特性がいくつか指定されました。文書、ツールチップおよび情報領域に表示される説明、ファイル・セレクタでビットマップを開くために使用されるファイル拡張子などに名前を付けました。最後に、生成される C++ クラスを選択しました。

ビューおよびユーザ・インターフェースの指定

1. 中央の領域のツリーからウィンドウ 1 を選択します。



文書タイプ View ページで、文書に付加できるビューのタイプを決定し、生成する C++ クラスを指定します。Application Framework は、文書と使用するさまざまなビューを提供しています。各ビューは、対応する IBM ILOG Views オブジェクトを実装し、文書とビューの間でのインタラクションを簡素化する特定メソッドを提供します。

ビットマップ・エディタの文書にビットマップがあるため、文書ビューに表示する必要があります。Application Framework で提示されている定義済み文書ビューからは、ビットマップを表示させることはできません。そのため、IlvDvFormView を選択します。これは IBM ILOG Views ファイル (.ilv) の読み込みを可能にする一般クラスです。

> ***メモ:*** *手順 2 では、ビットマップの表示方法を説明します。*

2. View ノートブック・ページでクラス・フィールドが、引き続き IlvDvFormView から派生する BitmapView によって置き換わります。
3. ビュー・コンテナ・ノートブック・ページで、タイプを [**MDI 子フレームで表示**] (デフォルト選択) に設定します。

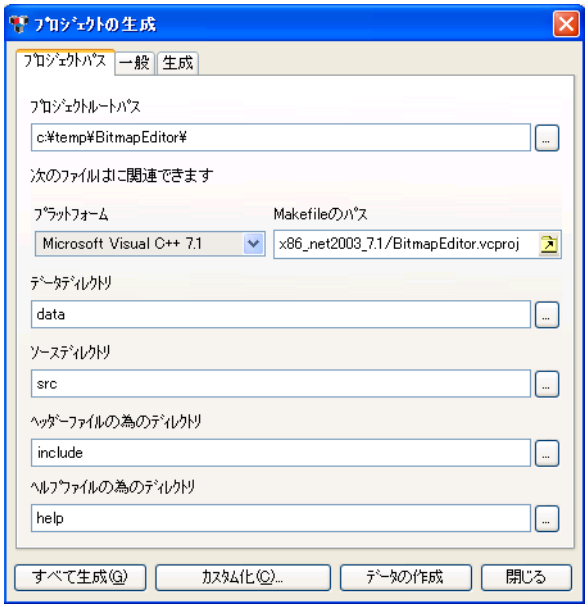
このパラメータにより、次のような複数のビューの表示方法の中から選択できるようになります。MDI 子フレーム、MDI 最大子フレーム、左、右、上、下のいずれかにドッキング、またはウィンドウをどこにもドッキングさせない。

文書およびビュー両方のパラメータが設定されたので、アプリケーションを構築し実行させるのに使用される C++ コードおよび設定ファイル (.odv) を生成できます。

C++ コードおよびリソース・ファイルの生成

アプリケーションのコードを今宣言したオプションで生成するには、次の手順に従います。

1. 標準ツールバーの [生成] メニューを開きます。
2. パラメータを選択します。次のウインドウが表示されます。



> ***メモ:*** *<WORKDIR> ディレクトリで作業していると想定します。*

- プロジェクト・ルート・パスを <WORKDIR> に設定します。
- プラットフォームを選択します。
- [すべて生成] ボタンをクリックします。

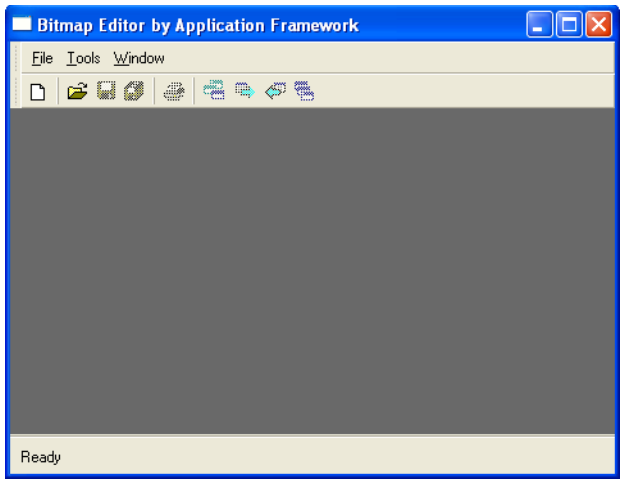
> ***メモ:*** *生成されたファイルは、作業しているプラットフォームに応じて、* `.cpp`、`.h`、`.odv`、`.dbm`、*Makefile あるいは* `.dsp` *ファイルになります。*

まとめ

これで、手順 1 が終了しました。Application Framework Editor のみを使用して、次のような機能があるビットマップ・エディタ・アプリケーションが作成されました。

- 最近使用されたファイル・リストの管理
- MDI フレームの処理

コンパイルして、生成したコードを実行すると、次のようなベーシック・アプリケーションが得られます。



> ***メモ:*** *アプリケーションを実行させるプラットフォームに応じて、アプリケーションがそのデータにアクセスできるようにするため、環境変数* `ILVPATH` *を* `../data` *に設定する必要がある場合があります。*

次のファイルが生成されました。

- `main.cpp` - このコードは、設定ファイル (`.odv`) を読み込み、アプリケーションを起動させます。

- `BitmapDocument.cpp` および `BitmapDocument.h` - 文書クラスを実装します。次のメソッドを生成し、完成させる必要があります (手順 2 を参照してください)。
  - コンストラクタおよびデストラクタ
  - `initializeDocument` - [新規] コマンドを呼び出すときに実行されます。
  - `clean` - 文書を破壊するときに実行されます (通常は、文書で開かれている最後のビューを閉じる時)。
  - `serialize` - ファイルから文書を読み込む、または保存するときに実行されます (通常は、[開く] および [閉じる] コマンド)。
- `BitmapView.cpp` および `BitmapView.h` - ビュー・クラスを実装します。次のメソッドを生成し、完成させる必要があります (手順 2 を参照してください)。
  - コンストラクタおよびデストラクタ
  - `initializeView` - 文書作成時あるいは新規ビューが文書に対して作成された時に実行されます。
  - `getBitmapDocument` - ビューに関連する `BitmapDocument` を戻します。
- `BitmapEditor.odv` - アプリケーション設定、コード生成、そしてコンパイルに関する永続的な情報を含んでいます。このファイルは、(たとえば) 新規コマンドを追加するために Application Framework Editor に再ロードされることがあります。

手順 2 は、文書およびビットマップ・エディタ・アプリケーションのビューに実装するために開発される C++ コードに焦点を当てます。

ステップ 2: 文書とビューの実装

ビットマップ・エディタ・アプリケーションのスケルトンが構築されたので (ステップ 1:Wizard を使ったベース・ファイル生成 を参照してください)、手順 2 では文書の実装方法を示します。文書の読み込み方法、新規文書の作成方法、そしてビットマップの表示方法を扱います。生成、および使用されたクラスは、`IlvDvDocument` から派生する `BitmapDocument`、および `IlvDvFormView` から派生する `BitmapView` です。どちらのクラスも手順 1 で Application Framework Editor によって生成されています。

BitmapDocument クラスの実装

これは、アプリケーション・データ (ビットマップ) を操作するため、アプリケーションのもっとも重要なクラスです。データで実行されるすべてのアクションは

このクラスによって行われます。まずビットマップ・ファイルの読み込み方法、そして新規ビットマップの作成方法を学びます。

BitmapDocument クラスは、IlvBitmap オブジェクトに対するポインタをメンバとして所有します。IlvBitmap オブジェクトは内部データです。このクラスは、BitmapDocument.h で宣言されており、BitmapDocument.cpp で定義されています。

ビットマップの読み込み

文書は次の場合に読み込まれます。

- [ファイル > 開く] をアプリケーション・メニュー・バーから選択したとき。
- あるいは、ツールバーから [開く] ボタンを選択したとき。
- あるいは、CTRL-O アクセラレータで選択したとき。

ユーザ・アクションと文書アクションの間の関連付けは、自動的に Application Framework によって実行されます。

[ファイル] > [開く] を選択すると、Application Framework は、BitmapDocument::readDocument メソッドを呼び出します。このメソッドは、ビットマップ・ファイルを読み込むときに実行される実際の作業を実装するためにオーバーライドされます。

```
IlvBoolean
BitmapDocument::readDocument(const IlvPathName& pathname)
{
   IlvDisplay*         display = getDisplay();
   // Read the bitmap
   IlvBitmap* bitmap = display->readBitmap(pathname);
   // If the bitmap has not been read correctly
   if (!bitmap || bitmap->isBad()) {
      delete bitmap;
      IlvFatalError("Cannot load %s bitmap !", getPathName());
      return IlvFalse;
   } else {
      // The bitmap read by readBitmap is a shared bitmap.
      // Here you do not want to use a shared bitmap because the editor may
      // modify it.So, the name set by the call to IlvDisplay::readBitmap
      // is removed.
      bitmap->setName(0);
      setBitmap(bitmap);
      setPalette(display->defaultPalette());
      return IlvTrue;
   }
}
```

> ***メモ:*** `IlvDvDocument::readDocument` *メソッドのデフォルト実装は、*`serialize` *仮想メソッドを呼び出します。文書内部の I/O を扱う別の方法は、*`serialize` *メソッドのみをオーバーライドすることです。しかし、*`serialize` *メソッドはここでは使用できません。ビットマップはフル・パスを使用して読み込む必要があるからです。*`serialize` *メソッドはストリームをパラメータとして受け取りますが、ビットマップを読み込むのに使用されるメソッドは、文字列をパラメータ (*`IlvDisplay::readBitmap(const char*)`*) として受け取ります。*`IlvDvDocument::readDocument` *および*`IlvDvDocument::serialize` *メソッドに関する詳細は、リファレンス・マニュアルを参照してください。*

以上で、ビットマップ・エディタ・アプリケーションは、ビットマップを読み込み、ビットマップ・データを `BitmapDocument` クラスに格納することができるようになりました。

新規ビットマップの作成

ビットマップ・エディタでは、新規文書、つまり、新規ビットマップを作成することもできます。チュートリアルを簡素化するために、この手順では、新しく作成されたビットマップは 128x128 ピクセルのサイズを持つと想定します。手順 4 では、ユーザがビットマップのサイズを設定できるダイアログ・ボックスの導入方法について説明します。

ユーザが次のアクションのいずれかを行うと、新しい文書が作成されます。

- メニュー・バーから [ファイル] > [新規] を選択
- メイン・ツールバーから [新規] ボタンを選択
- CTRL+N アクセラレータ

ユーザ・アクションおよび文書作成の間の関連付けは、`BitmapDocument::initializeDocument` メソッドを呼び出す Application Framework によって透過的に扱われます。このメソッドは、新規 `IlvBitmap` オブジェクトを作成し、これをビットマップ・エディタのビットマップ・メンバ・フィールドに `BitmapDocument::setBitmap` を使用して保存します。

```
IlvBoolean
BitmapDocument::initializeDocument(IlvAny data)
{
   if (!IlvDvDocument::initializeDocument(data))
      return IlvFalse;

   IlvDisplay*         display = getDisplay();
   // Creates the bitmap with a default size of 128x128
   IlvDim width = 128;
   IlvDim height = 128;
   IlvBitmap* bitmap =
```

```
            new IlvBitmap(display, width, height, display->screenDepth());
   setBitmap(bitmap);

   // Initialize it with the display palette
   setPalette(display->defaultPalette());
   getPalette()->invert();
   bitmap->fillRectangle(getPalette(), IlvRect(0, 0, width, height));
   getPalette()->invert();

   return IlvTrue;
}
```

ビットマップ・ビューの実装

手順 1 で、IlvDvFormView から継承される BitmapView と呼ばれるビューの C++ コードが生成されました。ここでは、このビューのコードを書き込みます。BitmapView クラスが BitmapDocument オブジェクトでビューを実装します。その目的は、ビットマップを表示し、ピクセルの色を変更するなどの編集を可能にすることです。

実装されるビューの要件は、次のようになります。

- IlvBitmap オブジェクトを表示する
- スクロールを許可する

IBM ILOG Views で IlvBitmap オブジェクトを表示するにはいくつかの方法があります。このチュートリアルでは、IlvZoomableIcon クラスがこの目的に使用されます。

すべてのグラフィック・オブジェクトを表示するには、それをコンテナに配置する必要があります。スクロール機能も必要なため、IlvSCGadgetContainerRectangle が使用されます (このクラスは、コンテナを所有し、スクロールを可能にします)。

これでビットマップの表示方法が決まったため、オブジェクトを BitmapView クラスに挿入する必要があります。IlvDvFormView のサブクラスとして、BitmapView は、IlvDvFormView::setFileName メソッドを使用して、IBM ILOG Views (.ilv) ファイルを読み込むことができます。このファイルは、文書の要素を表示するグラフィック・オブジェクトを記述します。

これで °¢.ilv ファイルが、IBM ILOG Views Studio を使って作成されます。この .ilv ファイルは、ビットマップを扱う IlvZoomableIcon が置かれる IlvSCGadgetContainerRectangle (スクロールするビュー) を含みます。

1. ivfstudio. を起動します。
2. [ファイル] > [新規] > [ガジェット] を選択します。
3. ガジェット・パレットで [矩形ビュー] を選択します。

4. IlvSCGadgetContainerRectangle をドラッグして、ガジェット・バッファでドロップします。
5. アタッチメント・モード・アイコンをクリックします。
6. IlvSCGadgetContainerRectangle オブジェクトを選択します。
7. 垂直および水平ガイドを設定します。
8. ガイドをダブルクリックします。
9. すべての距離にゼロを設定し、[適用] をクリックします。
10. [ファイル] > [名前を付けて保存] を選択し、<WORKDIR>/data/bitmapview.ilv と指定します。

これで、bitmapview.ilv ファイルは BitmapView クラスに統合されます。ビューの初期化は、BitmapDocument オブジェクトのインスタンス化が行われるときに、BitmapView::initializeView メソッドを介して実行されます。このメソッドのコードは、.ilv ファイルを読み込み、グラフィック・オブジェクトを文書に接続します。BitmapView クラスのコードは BitmapView.cpp ファイルで定義されています。

```
void
BitmapView::initializeView()
{
   IlvDvFormView::initializeView();
   BitmapDocument* document = getBitmapDocument();
   if (document->getBitmap()) {
      // Load the file
      setFilename("bitmapview.ilv");
      // Retrieve the IlvSCGadgetContainerRectangle
      IlvSCGadgetContainerRectangle* rectangle =
                        (IlvSCGadgetContainerRectangle*)getContainer()->
                         getObject((IlvUInt)0);
      // Change the color of the clip view to 'black'
      IlvColor* color = getDisplay()->getColor("black");
      rectangle->getScrolledView()->getClipView()->setBackground(color);
      // Add the zoomable icon
      IlvContainer* container = rectangle->getContainer();
      _icon = new IlvZoomableIcon(getDisplay(),
                                  IlvPoint(0, 0),
                                  document->getBitmap(),
                                  document->getPalette());
      container->addObject("Icon", _icon);
      container->fitToContents();
   }
}
```

BitmapDocument および BitmapView が実装されたので、ビットマップ・エディタ・アプリケーションをコンパイルして起動させることができます。これで、次が可能になります。

- PNG、BMP、あるいは JPG ファイルを 1 つあるいは複数開く。

- ビットマップの同時ビューを持つ。
- 新規ビットマップを作成する (ビットマップは現在は空)。

手順 2 で完成したビットマップ・エディタ・アプリケーションを、図 1.2 に示します。

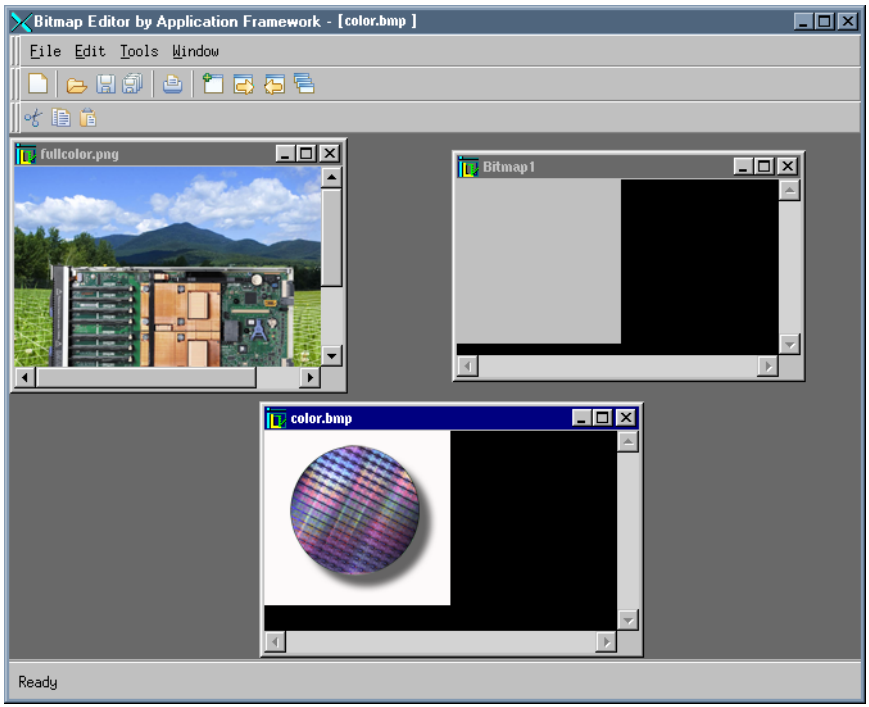


図 1.2　手順 2 以降のビットマップ・エディタ・アプリケーション

ステップ 3: ビットマップの変更および保存

ここでは、編集機能を、ビットマップ・エディタ・アプリケーションに追加します。必要な基本的編集機能によって、`BitmapDocument` のピクセルの色を変更できるようにします。

IBM® ILOG® Views では、編集はインタラクタ・オブジェクトを通じて行われます。インタラクタ・オブジェクトは、ビューあるいはグラフィック・オブジェクトに関連付けて、これらのオブジェクトにアクションを実行させるイベントを処理します。

ほとんどのドキュメント / ビュー・アプリケーションで、文書への変更はそのビューを通じて行われます。ユーザがビューでいくつかのアクションを行い、その結果コマンド呼び出しを通じて文書が変更されます。コマンドが一度実行されると、文書は、変更を反映させることができるようにビューに通知します。文書に実行されたコマンドは、元に戻す / やり直す操作を有効にするため、メモリに保存されます。

ピクセル編集を実装するには、次の手順に従います。

- BitmapDocument のピクセルの色を変更する IlvDvCommand のサブクラスの定義。特に、doIt および undo メソッドは実装される必要があります。
- IlvZoomableIcon オブジェクトと関連付けられている IlvInteractor のサブクラスの定義。このインタラクタは、BitmapView 上のユーザ・イベントを処理し、上記コマンドを呼び出します。
- 文書ビューに通知を追加します。
- Application Framework Editor で元に戻す / やり直すを有効化します。

BitmapView は、BitmapDocument を表示するために IlvZoomableIcon オブジェクトを使用します。文書のピクセルを編集するために、マウス・イベントを処理する IlvInteractor オブジェクトを関連付けて、適切なコマンドを呼び出さなくてはなりません。

DrawRectangleCommand の定義

DrawRectangleCommand クラスは、IlvDvCommand のサブクラスです。これは、drawcmd.h で宣言されており、drawcmd.cpp で定義されています。DrawRectangleCommand クラスの目的は、塗りつぶし四角形を文書ビットマップの特別な場所に、指定した色で描画して文書を変更することです。このため、このコマンドから、次にアクセスする必要があります。

- 文書
- 変更される四角形
- ビットマップに描画するために使用されたパレット

コンストラクタは、次のように表示されます。

```
DrawRectangleCommand::DrawRectangleCommand(BitmapDocument* document,
                                           const IlvPoint& point,
                                           IlvDim size,
                                           IlvPalette* palette,
                                           const char* name)
```

*メモ:*point *および*size *引数が、コマンドによって変更される四角形を計算するために使用されます。*

DrawRectangleCommand::doIt メソッドが、コマンドが実行されるときに呼び出されます (IlvDvDocument::doCommand メソッドを参照)。文書を変更する前に、コマンドを元に戻せるようにするために変更される四角形は保存してください。

```
void
DrawRectangleCommand::doIt()
{
   // Save the initial bitmap
   _bitmap->drawBitmap(_document->getPalette(),
                       _document->getBitmap(),
                       _rect,
                       IlvPoint(0,0));
   // Then draw in the document's bitmap
   _document->getBitmap()->fillRectangle(_palette, _rect);
   // Finally, refresh all the views connected to the document
   _document->refreshViews(_rect);
}
```

refreshViews メソッドは、この手順の後半で説明します。その目的は、変更されている文書に接続されているすべてのビューを更新することです。

undo メソッドは単に、doIt メソッドに保存されているビットマップを回復し、ビューを更新します。

```
void
DrawRectangleCommand::undo()
{
   // Restore the bitmap saved in _bitmap into the document's bitmap
   _document->getBitmap()->drawBitmap(_document->getPalette(),
                                      _bitmap,
                                      IlvRect(0,
                                              0,
                                              _bitmap->width(),
                                              _bitmap->height()),
                                      IlvPoint(_rect.x(), _rect.y()));
   // Then, refresh all the views connected to the document
   _document->refreshViews(_rect);
}
```

DrawBitmapInteractor の定義

DrawBitmapInteractor クラスは、IlvInteractor のサブクラスです。これは、drawinter.h で宣言されており、drawinter.cpp で定義されています。DrawBitmapInteractor クラスの目的は、BitmapView オブジェクトで生じるインタラクションを処理することです。マウスをクリックし、ドラッグすると、文書を変更する DrawRectangleCommands を生成します。以下に handleEvent メソッドのコードを示します。

```
IlvBoolean
DrawBitmapInteractor::handleEvent(IlvGraphic* g,
                                  IlvEvent& event,
                                  const IlvTransformer* t)
{
   switch (event.getType()) {
   case IlvButtonDown:
   case IlvButtonDragged: {
         IlvPoint point(event.x(),event.y());
         if (t)
            t->inverse(point);
         _document->drawRectangle(point, 2, _document->getPalette());
         return IlTrue;
   }
default:
      return IlFalse;
   }
}
```

インタラクタは、BitmapDocument::drawRectangle メソッドを呼び出して、文書ビットマップに描画します。このメソッドは単に、DrawRectangleCommand を作成して、これを実行します。

```
void
BitmapDocument::drawRectangle(const IlvPoint& point,
                              IlvDim size,
                              IlvPalette*        palette )
{
doCommand(new DrawRectangleCommand(this,
                                   point,
                                   size,
                                   palette,
                                   "drawRectangle"));
}
```

インタラクタは、BitmapView オブジェクトで文書ビットマップを表示する IlvZoomableIcon オブジェクトに設定されます。これは、BitmapView::initializeView メソッドで行われます。このメソッドのコードは、手順 2 と同じです。インタラクタは、呼び出しによってメソッドの最後に設定されます。

```
_icon->setInteractor(new DrawBitmapInteractor(document));
```

文書ビューに通知を追加

文書の変更を通知するための BitmapDocument::refreshViews メソッドを呼び出すために使用された DrawRectangleCommand メソッド (この手順については既述)。ここでは、このメソッドを実装します。

```
void
BitmapDocument::refreshViews(const IlvRegion& region)
{
   notifyViews(IlvGetSymbol("BitmapHasChanged"), 0, &region);
}
```

このメソッドは、指定した四角形を引数として与え、BitmapHasChanged メッセージをブロードキャストします。メッセージをキャッチするために、BitmapView がそのインターフェースでメソッドを宣言します。

```
IlvDvBeginInterface(BitmapView)
   Method1(BitmapHasChanged, bitmapHasChanged, IlAny, region)
IlvDvEndInterface1(IlvDvFormView)
```

- Method1 マクロの最初の引数は、メッセージ名です。
- 2 番目の引数は、メッセージ BitmapHasChanged を受け取るときに呼び出される BitmapView のメソッドです。
- 3 番目の引数は、bitmapHasChanged メソッドを呼び出すときに渡される最初の引数のタイプです。
- 4 番目の引数は、bitmapHasChanged メソッドを呼び出すときに渡される最初の引数の名前です。

> ***メモ:*** *インターフェース宣言では、簡単なタイプのみがサポートされています。変更された四角形がどれかを知るために* IlvRegion *が必要になるため、* IlAny *(つまり、non-typed ポインタ) が使用されます。*

以下に bitmapHasChanged メソッドのコードを示します。

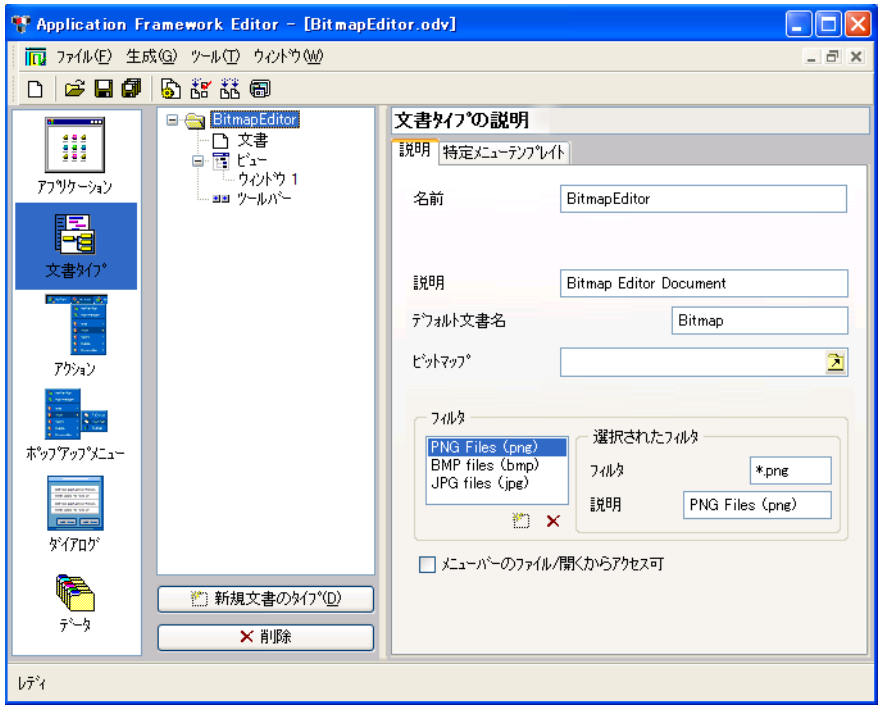
```
void
BitmapView::bitmapHasChanged(IlAny region)
{
   IlvRegion redrawRegion(*(IlvRegion*)region);
   IlvContainer* container = IlvContainer::GetContainer(_icon);
   // Deal with the container transformer
   if (container->getTransformer())
      redrawRegion.apply(container->getTransformer());
   // Optimization:Clip using the visible size of the container
   IlvRect  rect;
   container->sizeVisible(rect);
   redrawRegion.intersection(rect);
   // Then redraw the region
   container->bufferedDraw(redrawRegion);
}
```

Application Framework Editor で元に戻す / やり直すを有効化

まず、ユーザが元に戻す / やり直すコマンドを実行することができるように、GUI を変更しなくてはなりません。これは、Application Framework Editor によって行われます。元に戻す / やり直すは、メニュー・バーの [編集] メニュー、および文書特有のツールバーに追加されます。元に戻す / やり直すは定義済みのアクションなので、新規アクションを作成する必要はありません。

1. Application Framework Editor を起動します。

2. `BitmapEditor.odv` ファイルを開きます。
3. 左に位置する文書タイプ・アイコンをクリックします。

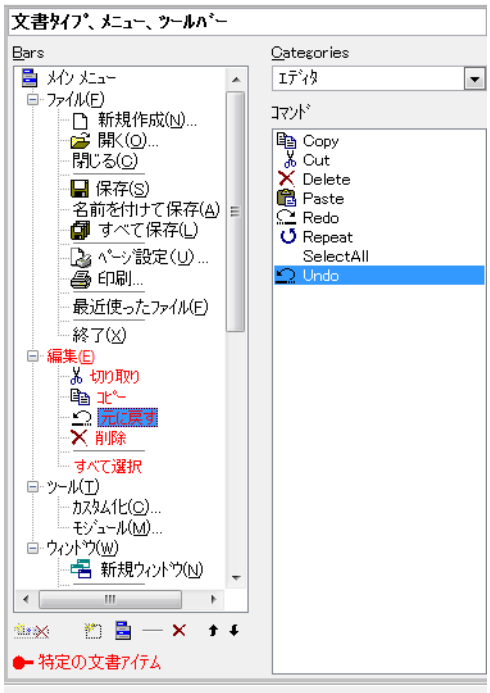


4. ツールバーをクリックします。

5. メニュー・バーおよびツールバーを表示しているツリーに位置する [編集] メニューの [貼り付け] を選択します。
6. 新規アクションを挿入します。

新しいアクションが追加されました。

7. コマンド・セクションで [元に戻す] を選択します。



8. 新規アクションを挿入します。
9. カテゴリを「エディタ」に変更します。
10. コマンド・セクションで [Undo] を選択します。
11. エディタに実装されていないデフォルト・エディタ・アクションを削除します。切り取り / コピー / 貼り付けを行います。

    [編集] メニューから、[切り取り] を選択し、コマンドをメニューから削除します。他の該当しないアクションにこれを繰り返します。
12. ツリーで文書特有のツールバーを探します (メニューの最後のツールバー)。
13. ツールバーの最後の項目を選択します。
14. 手順 6 から 11 を文書特有のツールバー用に繰り返し、手順 15 に進みます。
15. ファイルを保存します。

これでユーザは、元に戻す / やり直すアクションをトリガさせることができます。これらのアクションは自動的に、文書でキャッチされます。

アプリケーションをコンパイルして実行させることができます。

1. アプリケーションを起動します。

2. 新規ビットマップを作成します。
3. マウスをクリックして作成されたビューにドラッグします。変更が確認できるはずです。
4. メニュー・バーから [編集] メニューを開きます。元に戻すアクションが利用できるはずです。
5. 元に戻すアクションをクリックします。最後に描いたポイントが消えるはずです。

文書パレットの変更

この手順の前半で、文書ビットマップを変更するために使用されたパレットは、文書のパレットです (`DrawBitmapInteractor::handleEvent` メソッドを参照)。色の描画を変更する 1 つの方法は、文書パレットを変更することです。これを行うには、ColorChooser アクションという新規アクションをまずアプリケーションに追加する必要があります。

1. Application Framework Wizard を起動します。
2. `BitmapEditor.odv` ファイルを開きます。
3. アクションをクリックします。

4. [新規アクション] をクリックして新しいアクションを追加します。
5. 作成したアクションのコマンド名を `ColorChooser` に変更します。
6. アクションの説明を変更します。
7. ツールチップの説明を変更します。
8. [ビットマップ] タブで、アクションのビットマップを `icrespan.png` に変更します。
9. [文書タイプ] をクリックします。
10. ツリーのツールバーをクリックします。

11. 文書特有のツールバーで、コマンドの 1 つ (MyCommands) を選択します。

12. 新しいアクションをツールバーに追加します。

13. カテゴリを `Project` に変更します。

14. 作成したアクションを ColorChooser アクションに変更します。

15. 文書を保存します。

イベントは、文書レベルでキャッチされる必要があります。これを行うため、文書インターフェースを次のように変更します。

```
IlvDvBeginInterface(BitmapDocument)
   Action(ColorChooser, colorChooser)
IlvDvEndInterface1(IlvDvDocument)
```

`BitmapDocument::colorChooser` メソッドは、ColorChooser アクションがトリガされる時に呼び出されるということです。以下に `BitmapDocument::colorChooser` メソッドのコードを示します。

```
void
BitmapDocument::colorChooser()
{
   IlvColorSelector dialog(getDisplay());
   IlvColor* color = dialog.get(IlTrue);
   if (color) {
      IlvPalette* palette = getPalette();
      setPalette(getDisplay()->getPalette(palette->getBackground(),
                                          color,
                                          palette->getPattern(),
                                          palette->getColorPattern(),
                                          palette->getFont(),
                                          palette->getLineStyle(),
                                          palette->getLineWidth(),
                                          palette->getFillStyle(),
                                          palette->getArcMode(),
                                          palette->getFillRule()));
   }
}
```

メソッドは、カラー・セレクタを表示し、文書パレットを、`BitmapDocument::setPalette` メソッドを呼び出して変更します。

手順 3 で完成したビットマップ・エディタ・アプリケーションを、図 1.3 に示します。

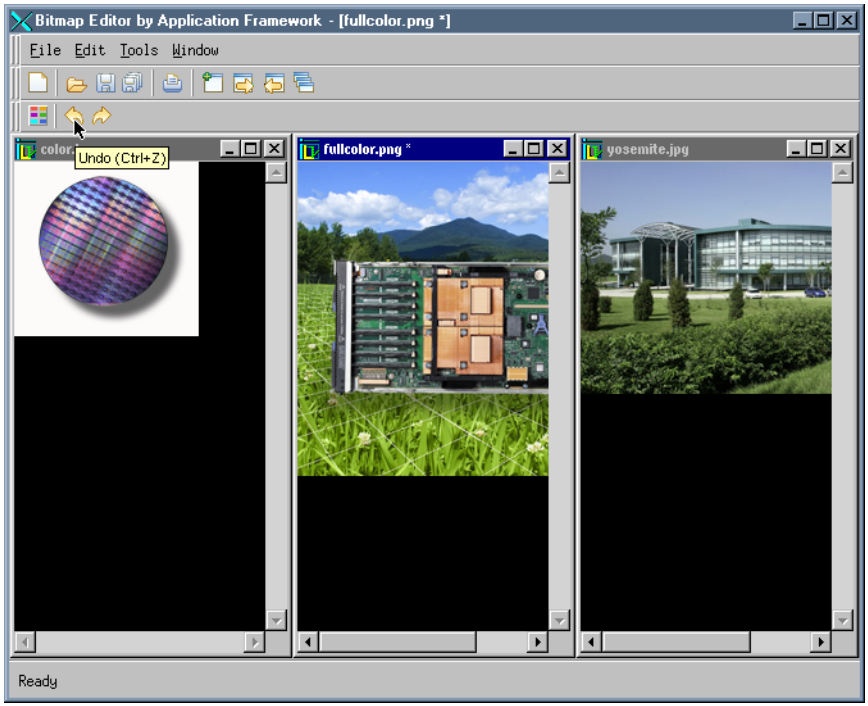


図 1.3 *手順 3 以降のビットマップ・エディタ・アプリケーション*

ステップ 4:IBM ILOG Views Studio で行われたダイアログの挿入

この手順では、IBM® ILOG® Views Studio で設計されたダイアログ・ボックスを Application Framework ベースのアプリケーション内部に統合します。この点を示すため、作成されるビットマップのサイズを選択できるようにします。次のダイアログ・ボックスが、ユーザが新規文書を要求するたびに表示されます。

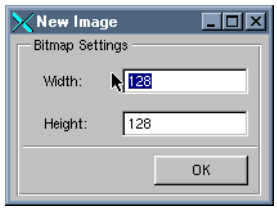


図 1.4 *サンプル・ダイアログ・ボックス*

この手順では、以下のトピックを扱います。

- IBM ILOG Views Studio を使用したダイアログ・ボックスの設計
- ビットマップ・エディタ・アプリケーションへのダイアログ・ボックスの統合

IBM ILOG Views Studio を使用したダイアログ・ボックスの設計

1. IBM ILOG Views Studio を起動します。IBM ILOG Views Studio についての詳細は、IBM ILOG Views Studio ユーザ・マニュアルを参照してください。
2. [ファイル] > [新規] > [ガジェット] を選択します。これで、空のガジェット・バッファが作成されます。
3. ガジェット項目を選択します。

4. `IlvFrame` をオブジェクト・パレットからドラッグして、ガジェット・バッファにドロップします。
5. `IlvNumberField` をドラッグして、その名前を `Width` に設定します。この詳細設定を行い、デフォルト値を `128` に変更します。
6. `IlvNumberField` をドラッグして、その名前を `Height` に設定します。この詳細設定を行い、デフォルト値を `128` に変更します。
7. 2 つの `IlvMessageLabel` をドラッグして、これらのラベルを変更します。1 つは `Width` に、そしてもう 1 つは `Height` にします。
8. `IlvButton` をドラッグして、そのラベルを `OK` に設定します。そのコールバックも `apply` に変更します。このコールバックは、`IlvDialog` オブジェクトのサブクラス用定義済みコールバックです。詳細は、`IlvDialog::apply` を参照してください。
9. `IlvReliefLine` をドラッグして、ボタンを数値フィールドから離します。
10. 図 1.4 に示すようにオブジェクトを配置します。
11. ファイルをビットマップ・エディタ・アプリケーションの `data` ディレクトリに `bmpsize.ilv` として保存します。
12. [ファイル] > [新規] > [デフォルト･アプリケーション作成] を選択します。これは、C++ コード生成のために必要です。

13. [コード] > [パネル・クラス・インスペクタ] を選択します。

14. クラス名を `BitmapSizeDialog` に変更します。

15. ベース・クラスを `IlvDialog` に変更します。

16. ディレクトリ・ヘッダーおよびソースを、ビットマップ・エディタ・アプリケーションのディレクトリに合うように変更します。

17. [適用] をクリックします。

18. [コード] > [パネルのクラスを生成] を選択して、コードを生成します。

2 つのファイル、ヘッダー・ファイル (`include/bmpsize.h`) とソース・ファイル (`src/bmpsize.cpp`) が生成されました。

19. IBM ILOG Views Studio を終了します。

ビットマップ・エディタ・アプリケーションへのダイアログ・ボックスの統合

ダイアログ・ボックス用コードが生成されました。これをアプリケーションに統合させなくてはなりません。

新規ビットマップを作成するときにがサイズを選択するためのダイアログ・ボックスが表示されます。手順 2 では、新規文書を作成するために、`BitmapDocument::initializeDocument` メソッドがどのように呼び出されるかを示しました。今度はこのメソッドを、ダイアログ・ボックスを表示させるように変更する必要があります。

```
IlvBoolean
BitmapDocument::initializeDocument(IlvAny data)
{
   if (!IlvDvDocument::initializeDocument(data))
      return IlvFalse;

   IlvDisplay*         display = getDisplay();
   // Pops-up a dialog to let the user choose the initial size
   BitmapSizeDialog dialog(display, "New Image", "New Image");
   dialog.moveToMouse(IlvCenter);
   dialog.wait();
   IlvDim width = dialog.getWidth()->getIntValue();
   IlvDim height = dialog.getHeight()->getIntValue();
   IlvBitmap* bitmap =
       new IlvBitmap(display, width, height, display->screenDepth());
   setBitmap(bitmap);

   // Initialize it with the display palette
   setPalette(display->defaultPalette());
   getPalette()->invert();
   bitmap->fillRectangle(getPalette(), IlvRect(0, 0, width, height));
   getPalette()->invert();
   return IlvTrue;
}
```

bmpsize.cpp ファイルを、アプリケーションをリンクするために、Makefile あるいはプロジェクトに追加する必要があります。

手順 4 で完成したビットマップ・エディタ・アプリケーションを、図 1.5 に示します。

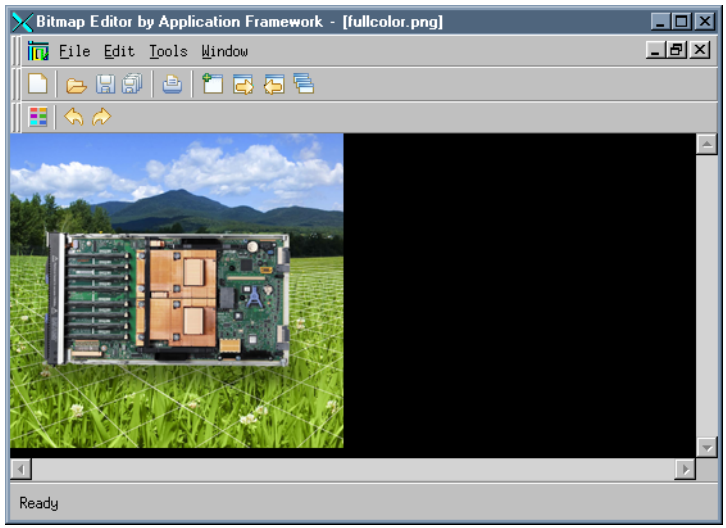


***図 1.5** 手順 4 以降のビットマップ・エディタ・アプリケーション*

ステップ 5: ズーム・コマンドの追加

この手順では、ビットマップ・エディタ・アプリケーションをズーム操作ができるように変更します。次のタスクを行います。

- Application Framework Editor を使用したズーム・アクションの追加
- ビットマップ・ビュークラスを変更して新規アクションをキャッチする
- ビットマップ・ビュー・クラスのズームの実装

Application Framework Editor を使用したズーム・アクションの追加

1. Application Framework Editor を起動します。
2. BitmapEditor.odv ファイルを開きます。
3. アクションをクリックします。

4. [新規アクション] をクリックして新しいアクションを追加します。
5. 作成したアクションのコマンド名を ZoomIn に変更します。
6. アクションの説明を変更します。
7. ツールチップの説明を変更します。
8. [ビットマップ] タブで、アクションのビットマップを iczoomm.png に変更します。
9. [文書タイプ] をクリックします。
10. ツリーのツールバーをクリックします。

11. 文書特有のツールバーで、コマンドの 1 つ (MyCommands) を選択します。
12. 新しいアクションをツールバーに追加します。
13. 作成したアクションを ZoomIn アクションに変更します。

ZoomOut アクションを追加するために、この手順を繰り返します (icuzoomm.png ビットマップを使用)。次に、文書を保存して、Application Framework Editor を終了します。

ビットマップ・ビュークラスを変更して新規アクションをキャッチする

まず、新しいアクションを宣言するためにビューのインターフェースを変更する必要があります。

```
IlvDvBeginInterface(BitmapView)
   Method1(BitmapHasChanged, bitmapHasChanged, IlAny, region)
   Action(ZoomIn, zoomIn)
   Action(ZoomOut, zoomOut)
IlvDvEndInterface1(IlvDvFormView)
```

次に、BitmapView::zoomIn および BitmapView::zoomOut メソッドを実装します。

ビットマップ・ビュー・クラスのズームの実装

BitmapView::zoomIn および BitmapView::zoomOut メソッドはインラインされ、BitmapView::zoom メソッドが呼び出されます。

```
void zoomIn() { zoom((IlFloat)2.); }
void zoomOut() { zoom((IlFloat).5); }
```

実装しなくてはならない唯一のメソッドは、BitmapView::zoom メソッドです。

```
void
BitmapView::zoom(IlFloat factor)
{
   IlvContainer* container = IlvContainer::GetContainer(_icon);
   container->zoomView(IlvPoint(0,0), factor, factor);
   // Resize the container to fit the bitmap
   IlvRect bbox;
   container->boundingBox(bbox);
   container->resize(bbox.w()*factor, bbox.h()*factor);
}
```

これは、文書ビットマップを表示する IlvZoomableIcon の描画に使用するトランスフォーマを変更するため IlvContainer::zoomView メソッドを使用します。

次に、スクロール・ビューのスクロール・バーが更新されるように、アイコンのコンテナをリサイズします。

最終的に完成したビットマップ・エディタ・アプリケーションを、図 1.6 に示します。

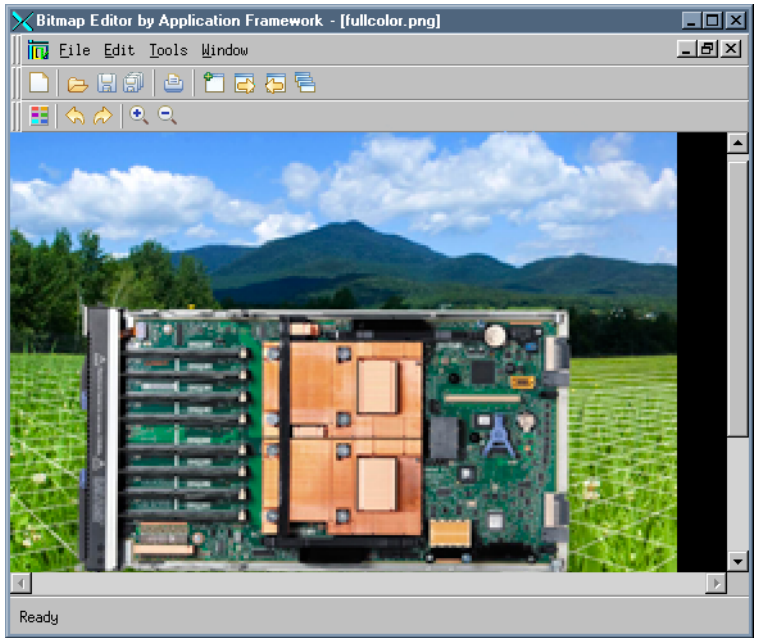


図 1.6 *完成したビットマップ・エディタ・アプリケーション*

索引