IBM ILOG Views

Application Framework V5.3

ユーザ・マニュアル

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

本書について

本*ユーザ・マニュアル*では、IBM® ILOG® Views の Application Framework パッケージについて説明します。Application Framework は、グラフィック作成と制御を行う IBM ILOG Views Component Suite C++ ライブラリに基づいて、アプリケーション用グラフィカル・ユーザ・インターフェース (GUI) 開発タスクを単純化するために設計されたライブラリです。

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。

マニュアル構成

このマニュアルは、以下の章で構成されています。

- 1 章 *IBM ILOG Views Application Framework について*では、IBM® ILOG® Views Application Framework パッケージのドキュメント / ビュー・アーキテクチャやその他の機能の概要を説明します。
- 2 章 *Application Framework Editor の使用*では、使いやすい GUI である Application Framework Editor について説明します。
- 3 章 *アプリケーションの実装* では、Application Framework のドキュメント / ビュー・アーキテクチャを組み込むために必要な手順とクラスについて説明します。
- 4 章 *Application Framework インターフェース* では、Application Frameowrk インターフェース・メカニズムの組み込み方について説明します。
- 5 章 *アクション* では、Application Framework でアクションを有効にし、処理する方法について説明します。

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。

- コードの引用やファイル名は `courier` 書体で記載されます。
- ユーザが入力する項目は、*`courier`* 書体で記載されます。
- 初出の*斜体*の用語には、ユーザ・マニュアルの用語集で解説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。

- ILOG Views Foundation 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。例 :

```
virtual IlvClassInfo* getClassInfo() const;

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

章

1

IBM ILOG Views Application Framework について

IBM® ILOG® Views Application Framework によって、アプリケーションのユーザ・インターフェースを定義する使いやすいグラフィカル・ユーザ・インターフェース (GUI) が提供されます。この章では、IBM ILOG Views Application Framework パッケージの概要を説明します。本章は、次のような構成になっています。

- *Application Framework とは*
- *ドキュメント / ビュー・アーキテクチャ*

Application Framework とは

Application Framework は、図 1.1 に示すような完全なアプリケーションが開発できるライブラリです。

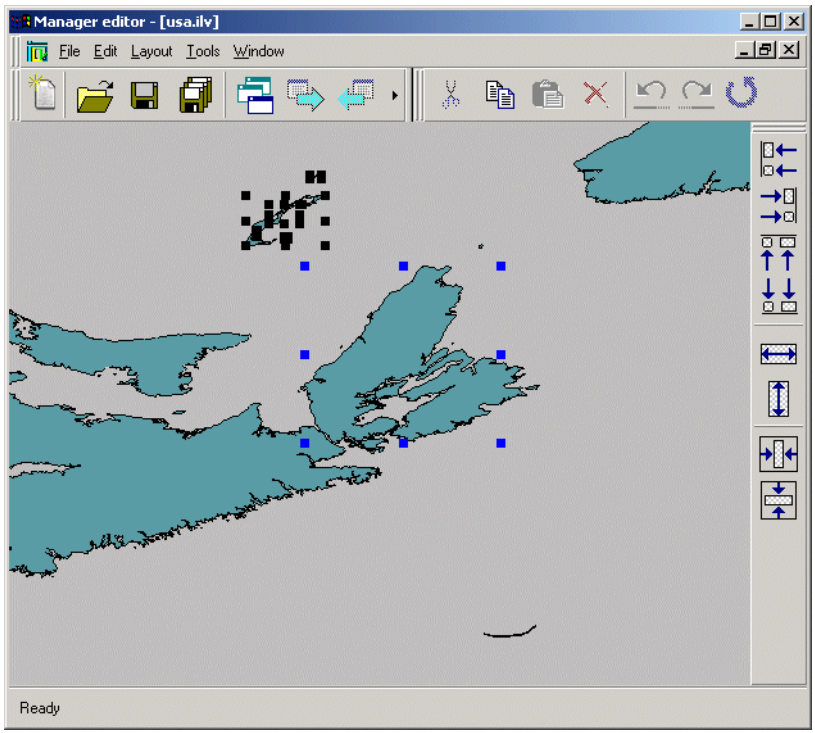


図 1.1 *Application Framework によって開発されたアプリケーション*

アプリケーションがグラフィカルに編集できる Application Framework Editor と呼ばれるツールを備わっています。メニュー、バー、アクション、動的メニューなどはすべて、このツールを使って指定します。第 2 章では、Application Framework Editor の起動方法と使用方法を詳しく説明します。

Application Framework にはまた、オブジェクトを追跡し、GUI イベントを処理できるメカニズムがあります。このメカニズムについては、4 章 *Application Framework インターフェース* で説明します。

ドキュメント / ビュー・アーキテクチャ

Application Framework は、ほとんどの Windows アプリケーションと同様、ドキュメント / ビュー・アーキテクチャに基づいて構築されています。このタイプのアーキテクチャでは、アプリケーションは複数のツールバーとメニューを保持する、1 つの *フレーム・ウィンドウ* で、それらのツールバーやメニューによって同時に複数文書が編集できます。このフレーム・ウィンドウが操作する *文書* ([ファイル]> [開く]、[ファイル]>[新規作成] などのメニューによって開かれるデータ) を、

ユーザは *ビュー*の内部で編集できます。ビューは通常、フレーム・ウィンドウ内に作成されます。

たとえば、Microsoft Excel の文書は、`.xls` ファイルからメモリに読み込まれた 1 つの表でこの文書を表示し、変更できるビューがシートまたはチャートになります。

> ***警告:*** *Microsoft アプリケーションの場合、「ドキュメント / 文書」という用語は、メモリ内のデータという意味にも、ユーザがそれらのデータを編集できるようにするビューの意味にも使われます。*

ドキュメント / ビュー・アーキテクチャについては、3 章 *アプリケーションの実装* で詳しく説明します。

章

2

Application Framework Editor の使用

IBM® ILOG® Views Application Framework によって、アプリケーションのユーザ・インターフェースを定義する使いやすいグラフィカル・ユーザ・インターフェース (GUI) が提供されます。

Application Framework Editor の起動

最初に Application Framework Editor を起動すると、次のダイアログ・ボックスが表示されます。

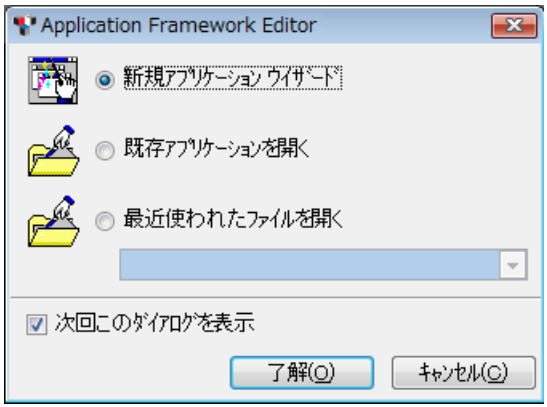


図 2.1 *Application Framework Editor ウィザード*

このダイアログ・ボックスには、次のオプションがあります。

- ［新規アプリケーション・ウィザード］ー新規アプリケーションの作成を開始します。*新規アプリケーションの作成*を参照してください。
- [既存アプリケーションを開く] ーこのブラウズ・ダイアログ・ボックスでは既存アプリケーションを選択して開きます。
  *オプション・ファイル (.odv ファイル) の作成と設定*を参照してください。
- [最近使われたファイルを開く] - ドロップダウン・リストからアプリケーションをすぐに選択できます。*Application Framework Editor のメイン・ウィンドウ*を参照してください。

[次回このダイアログを表示] オプションの選択解除で、この画面をバイパスできます。その場合は、アプリケーション・エディタのメイン・ウィンドウへ直接移動します。

Application Framework Editor のメイン・ウィンドウ

Application Framework Editor のメイン・ウィンドウには、メニュー・バー、アクション・ツールバー、ステータス・バー、アプリケーション・コンポーネント・パレット、複数文書の作業領域が表示されます。起動時のウィンドウを 図 2.2 に示します。

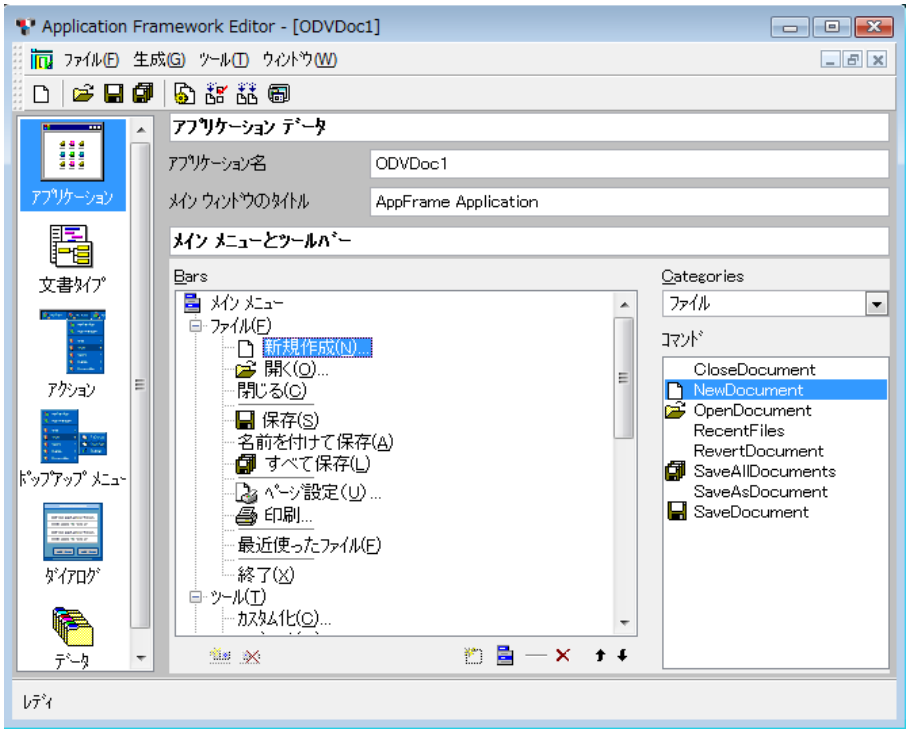


図2.2 アプリケーション・エディタのメイン・ウィンドウ

コンポーネント・パレット

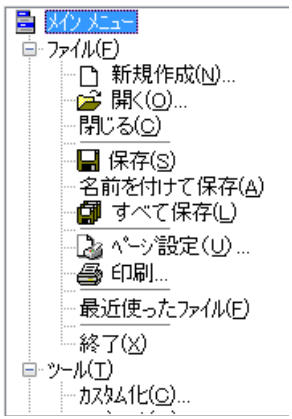
左側のアプリケーション・コンポーネント・パレットを使うと、編集するアプリケーション・エンティティが選択できます。

- 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
- ダイアログ - ダイアログ・ボックスとウィンドウ用
- データ

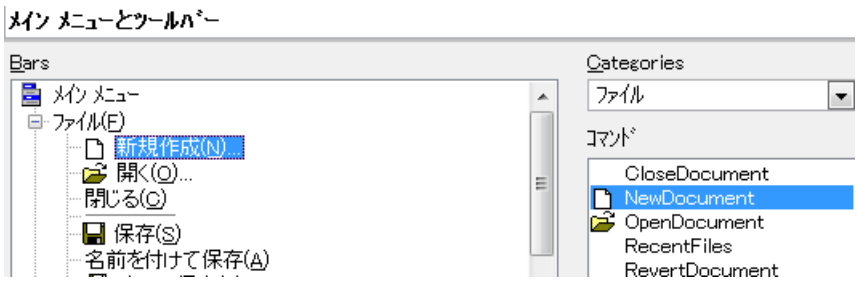
作業領域

右側に表示される作業領域で、各アプリケーション・エンティティ用に選択した項目のパラメータを設定できます。

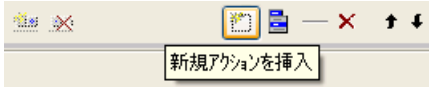
- 作業領域の中央部分にある階層ツリーでは、編集または追加する項目の場所が選択できます。例：



- ツリーの中の項目を選択すると、作業領域にある他の使用可能なパラメータのエントリ・フィールドがアクティブになります。フィールドは、特定の操作用にカスタマイズできます。例：



- 作業領域下部の作業領域ツールバーでは、現在の作業領域にカスタマイズされた操作を簡単に選択できます。例：



アプリケーション・エンティティのタイプによって、作業領域ツールバーは変化し、項目の灰色表示 / アクティブが切り替わります。可能なツールバー・アイコンを ï 2.1 に示します。

表 2.1 *作業領域ツールバー*

ツールバー・アイコン	説明
	新規アクション、ポップアップ、またはダイアログ項目を現在選択しているツリーまたは項目の下に追加するか、新規カテゴリ、アクセラレータ、その他の項目をリストに挿入します。
	新規ポップアップ・メニューを挿入します。
	新規区切り文字を挿入します。
	現在選択している項目を削除します。
	選択した項目をツリーの上位へ移動します。
	選択した項目をツリーの下位へ移動します。
	新規ツールバーを挿入します。
	選択したツールバーを削除します。

新規アプリケーションの作成

Application Framework でアプリケーションを開発するには、次の基本ステップに従います。

1. Application Framework Editor を起動して、アプリケーションのオプションすべてを編集します。アプリケーション内で表示される異なるメニューとツールバーの編集、アプリケーションが開くことのできるすべての文書タイプの記述などを行います。これらの項目はオプション・ファイルに保存され、生成したアプリケーションを初期化するときに読み込みます。

2. アプリケーション・コードの生成には、Application Framework Editor を使用します。
3. 生成したコードを完成させます。
   - データ (文書) を管理するには
     - データをシリアル化し、
     - アクセサを追加します。
   - データ (ビュー) を表示し、編集するには
     ーデータに応じてビューを初期化します (ツリー・ガジェットの構築はその一例です)。
     ービューのユーザ・イベントによって生成されたコマンドを管理します。

> ***メモ:*** *ステップ1 で説明したアプリケーション・オプションは、アプリケーション開発中にいつでも変更できます。これらのオプションの変更に当たって、コードの再生成は必要ありません。*

本章の残りの部分では、Application Framework Editor の一般的なナビゲーション機能と操作機能について説明します。ステップ毎の説明については、Application Framework のチュートリアル・サンプルを参照してください。

文書タイプの選択

新規アプリケーション作成の第 1 ステップは、文書タイプの選択です。[新規アプリケーション] を開始すると、[文書テンプレートの選択] ダイアログ・ボックスが表示されます。

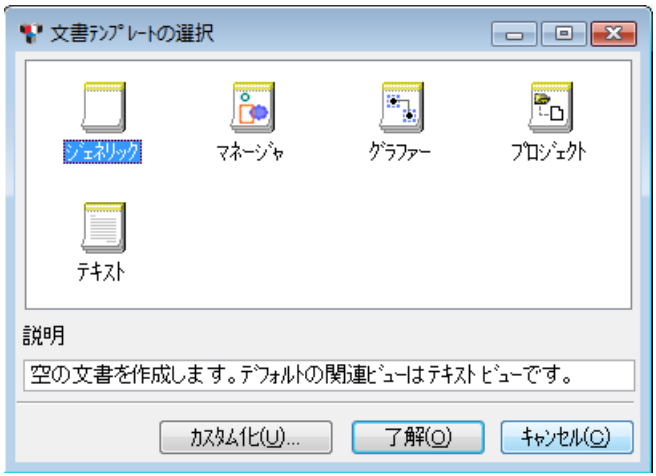


図2.3 文書タイプの選択

いくつかの定義済み文書タイプが使用できます。各文書タイプにはそれぞれ、そのデータを操作する便利な方法が定義されている上、あらかじめ特定のビューが関連付けられています。

文書タイプの詳細を、ï 2.2 に示します。

表2.2 文書タイプの選択

文書タイプ	用途	説明
	一般アプリケーションの作成	ジェネリック文書タイプでは、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。
	マネージャ・アプリケーションの作成	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。
	グラファー・アプリケーションの作成	グラファーの文書タイプでは、ノードまたはリンクとして `IlvGrapher` に挿入されている `IlvGraphic` オブジェクトを扱います。

表 2.2 文書タイプの選択 （続き）

文書タイプ	用途	説明
	プロジェクト・アプリケーションの作成	プロジェクトの文書タイプは、フォルダとサブフォルダ内のファイル編成です。
	テキスト・アプリケーションの作成	テキストの文書タイプは、任意のテキスト文書です。

文書タイプの選択が終わると、Application Framework Editor のメイン・ウィンドウが表示されます。*Application Framework Editor のメイン・ウィンドウ*を参照してください。

オプション・ファイル (.odv ファイル) の作成と設定

Application Framework では、アプリケーションを記述するパラメータすべてを、1 つのオプション・ファイル (`.odv` 拡張子) に格納します。

新規アプリケーションを作成 ([ファイル] メニュー、ツールバー、または初期ウィザードから [新規作成] を選択) し、文書タイプを選択するたびに、Application Framework Editor によって新しい `.odv` ファイルが開かれます。

アプリケーション・パラメータの設定

Application Framework Editor のパレットから [*アプリケーション*] を選択し、Application Framework Editor でアプリケーション・パラメータを設定します。

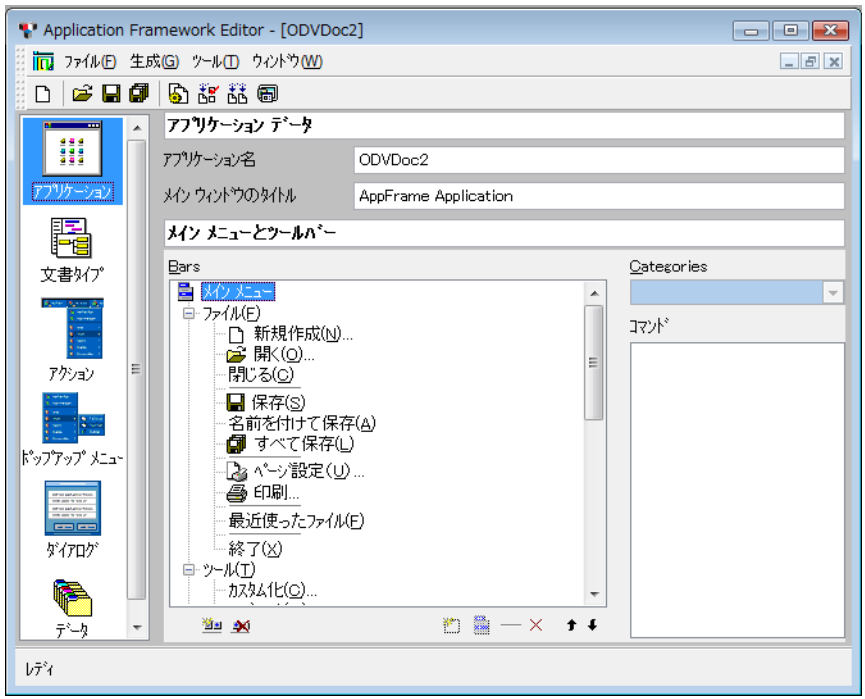


図 2.4 *パレットから選択したアプリケーション*

Application Framework Editor のアプリケーション・データ作業領域では、次を設定します。

- [*アプリケーション名*] フィールドのアプリケーション名。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。図 2.2 に示されているデフォルト名は、`ODVDoc1` です。
- [*メイン・ウィンドウのタイトル*] フィールドのメイン・ウィンドウ・タイトルです。これは、作成するアプリケーションのタイトルとして表示される名前です。

メニュー項目の追加

Application Framework Editor のパレットから [*アプリケーション*] を選択したら、作業領域の [メイン・メニューとツールバー] セクションにメニュー項目を追加します。

1. 新規項目を挿入する [メイン・メニュー] ツリーにある項目を選択します。選択した項目の後に、新規項目が挿入されます。
2. [新規アクションの挿入] ボタン  をクリックします。

3. [カテゴリ] コンボ・ボックスと [コマンド] リストの関連するアクションを選択すると、挿入した項目を変更できます。

メニュー項目を変更する場合は、目的の項目を選択してから [コマンド] リストの新規アクション選択によって変更します。

新規コマンドを挿入する場合は、*アクションの作成*を参照してください。

メイン・メニュー・バーから項目を削除する場合は、削除する項目を選択してから [削除] ボタン  をクリックします。

ツールバー項目の追加

Application Framework Editor のパレットから [*アプリケーション*] を選択したら、作業領域の [メイン・メニューとツールバー] セクションにツールバー項目を追加します。

1. ツールバーの新規ボタンを挿入する [標準] ツリーの項目を選択します。選択した項目の後に、新規項目が挿入されます。
2. [新規アクションの挿入] ボタン  をクリックします。
3. [カテゴリ] コンボ・ボックスと [コマンド] リストの関連するアクションを選択すると、挿入した項目を変更できます。

メニュー項目を変更する場合は、目的の項目を選択してから [コマンド] リストの新規アクション選択によって変更します。

新規コマンドを挿入する場合は、*アクションの作成*を参照してください。

ツールバーから項目を削除する場合は、削除する項目を選択してから [削除] ボタン  をクリックします。

文書パラメータの設定

Application Framework Editor のパレットから [*文書タイプ*] を選択し、Application Framework Editor で文書パラメータを設定します。

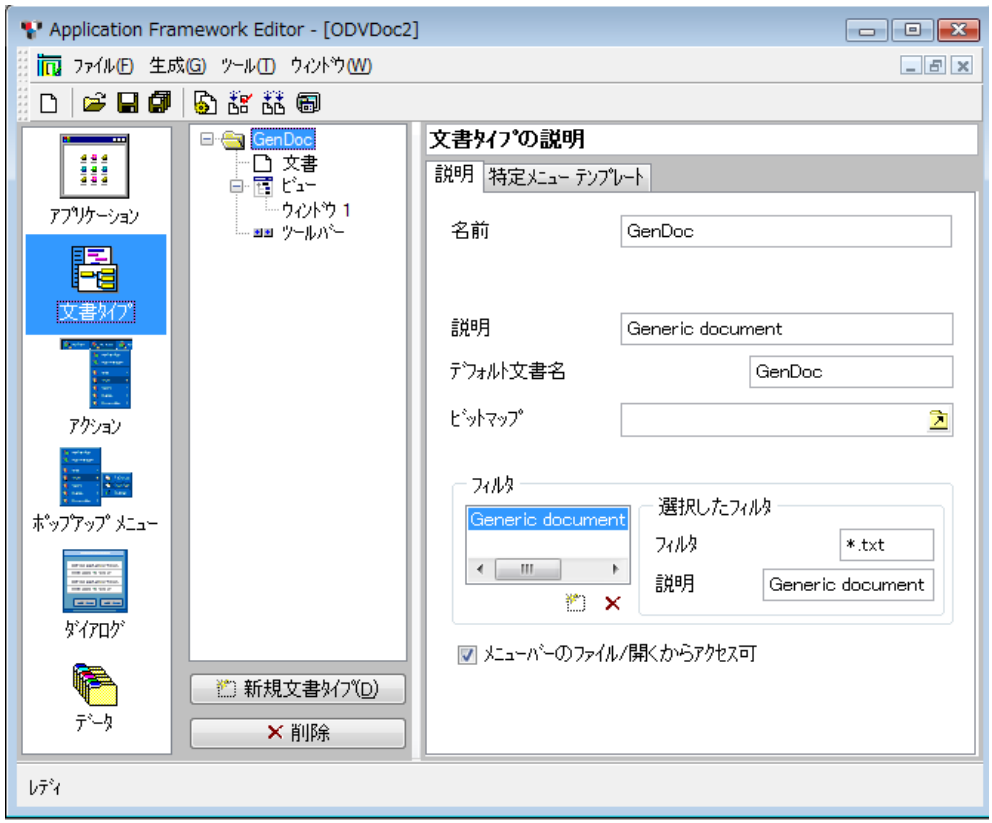


図 2.5 パレットから選択した文書タイプ

中央の列に、選択した文書タイプ (図 2.5 の GenDoc、もしくは Grapher、Project、Text、または Manager) を親ディレクトリとする文書ツリーが表示されます。この列には、作成するアプリケーションで処理できる文書タイプが表示されます。[新規文書タイプ] ボタン によって、多くの文書タイプを追加できます。

右の列では、中央の列で選択した項目のパラメータを、次に説明するように変更できます。

一般文書パラメータの設定

ツリーで GenDoc を選択すると、作業領域に次のタブが表示されます。

◆ [*説明*] タブには、文書タイプに関する基本情報が含まれます。

***図 2.6** [説明] タブ (文書タイプの説明)*

- [名前] : 選択した文書タイプの作成に使用できる文書テンプレートを検索する場合に、この名前を使用します。
- [説明] : 文書タイプの説明。
- [デフォルト文書名] : 選択したタイプの文書を作成する場合、この名前が使用されます。作成された文書がこの名前になります。
- [ビットマップ] : 選択した文書タイプに関連付けられたビットマップです。
- [フィルタ] セクション : このセクションでは、作成するアプリケーションの [文書を開く] ダイアログ・ボックスに表示される要素を定義できます。これらの要素は、[文書を開く] ダイアログ・ボックスの [ファイル・タイプ] セクションで使用されます。
- [フィルタ] : 選択した文書ファイルの拡張子です。このフィールドのデータ形式は、`*.xxx` である必要があります。`xxx` が拡張子です。
- [説明] : フィルタの説明です。
- [メニューバーのファイル / 開くからアクセス可] : このチェックボックスのチェックをオンにすると、作成した最終アプリケーションの [開く] ダイアログ・ボックスにこれらのフィルタが表示されます。

◆ [*特定メニュー・テンプレート*] タブでは、文書の可視機能を設定します。

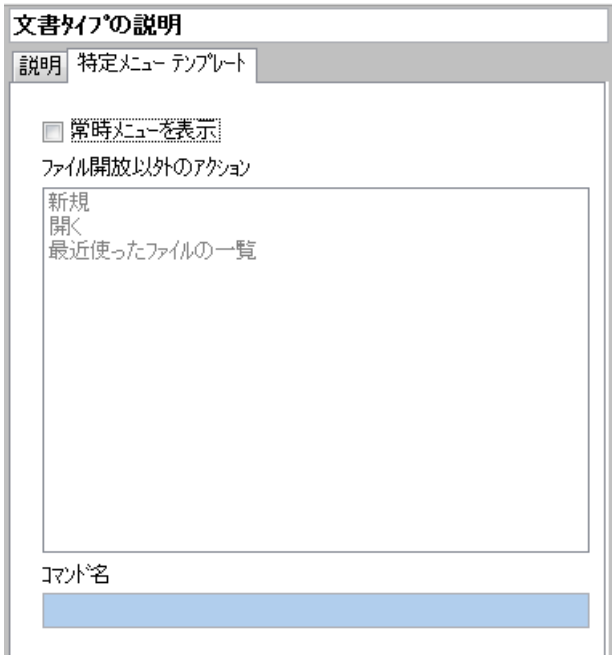


図 2.7 [特定メニュー・テンプレート] タブ

◆ [常時メニューを表示] : このトグルのチェックをオンにすると、選択したタイプの文書がたとえアクティブでなくても、その特定メニューとツールバーが表示に加わります。

選択した文書のパラメータ設定

文書タイプの「文書」項目をツリーで選択すると、作業領域に次のフィールドが表示されます。

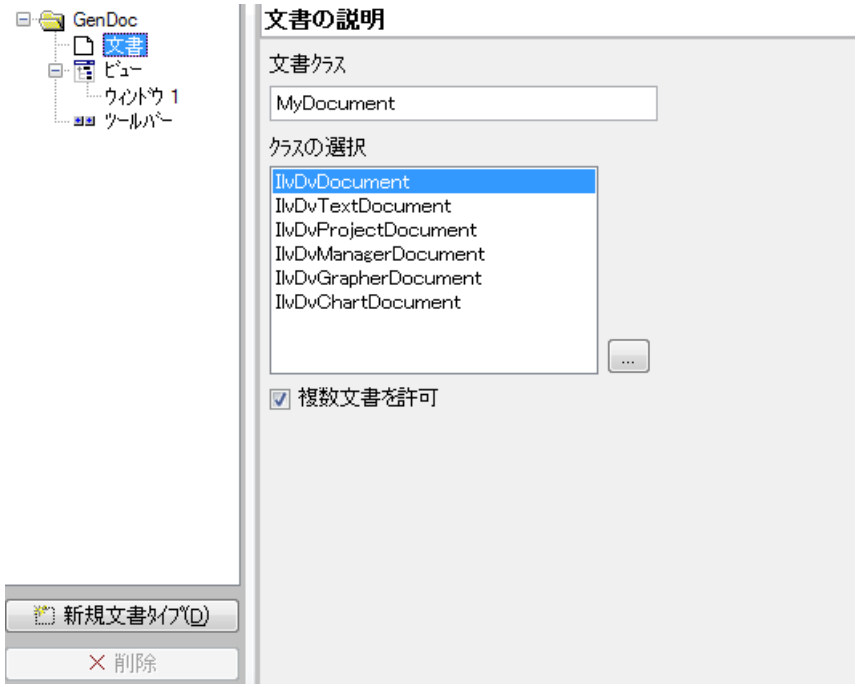


図2.8 文書の説明

- [文書クラス] には、コード生成で使用するクラス名を指定します。
- 定義済みクラスが一覧されている [クラスの選択] 文字列リストから 1 つの項目を選択することで、親クラスが指定できます。
- [複数文書を許可] チェック・ボックスのオン / オフで、選択したタイプの文書を処理できるのは 1 つだけか複数かを指定できます。

ウィンドウ・パラメータの設定

文書タイプのウィンドウ項目をツリーで選択すると、作業領域に次のタブが表示されます。

◆ [*ビュー*] タブでは、文書のビューのクラスが定義できます。

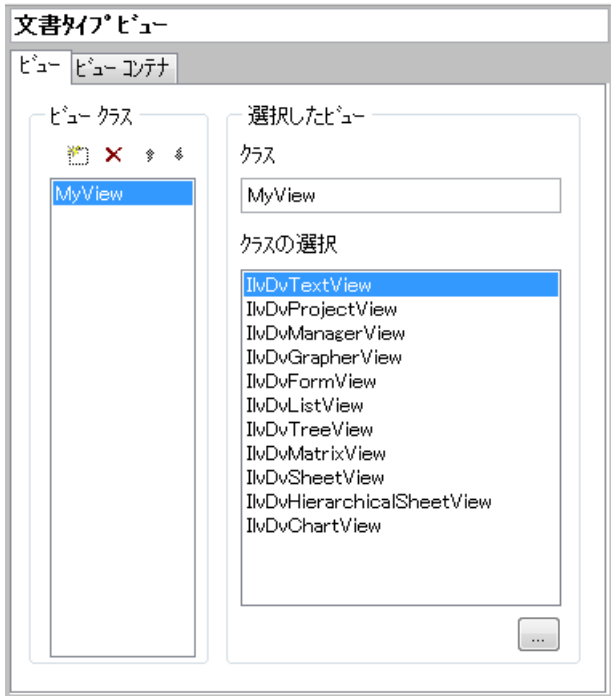


***図 2.9** [ビュー] タブ*

- [クラス] テキスト・フィールドには、コード生成で使用するクラス名を指定します。
- [クラスの選択] リストで、選択したビュー・クラスの派生元クラスを選びます。

◆ [*ビュー・コンテナ*] タブでは、ビューを含むコンテナのパラメータが追加できます。

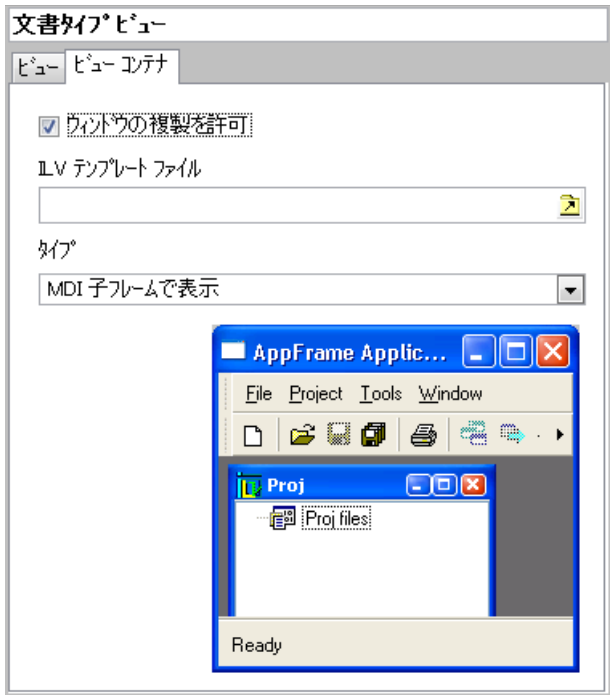


図 2.10 *[ビュー・コンテナ] タブ*

- [ウィンドウの複製を許可] チェック・ボックスのオン / オフで、同じ文書のビューを処理できるのは 1 つだけか複数かを指定できます。
- [タイプ] コンボ・ボックスでは、ウィンドウの初期設定を選択できます。以下の選択肢があります。

  — MDI 子フレームで表示

  — MDI 最大子フレームで表示

  —ウィンドウを左にドッキングする

  —ウィンドウを右にドッキングする

  —ウィンドウを上にドッキングする

  —ウィンドウを下にドッキングする

  —ウィンドウをどこにもドッキングさせない

- すべてのドッキング設定で、このウィンドウの表示または非表示時に呼び出すメソッドを [アクションを表示 / 非表示] テキスト・フィールドに指定で

きます。このテキスト・フィールドは、ドッキング設定を選択したときのみ表示されます。

文書タイプのツールバー・パラメータ設定

ツリーの「ツールバー」項目を選択すると、作業領域でこのタイプの文書がアクティブな時だけ表示されるように、特定ツールバーの定義または変更ができます。このツールバーの編集については、*ツールバー項目の追加*を参照してください。

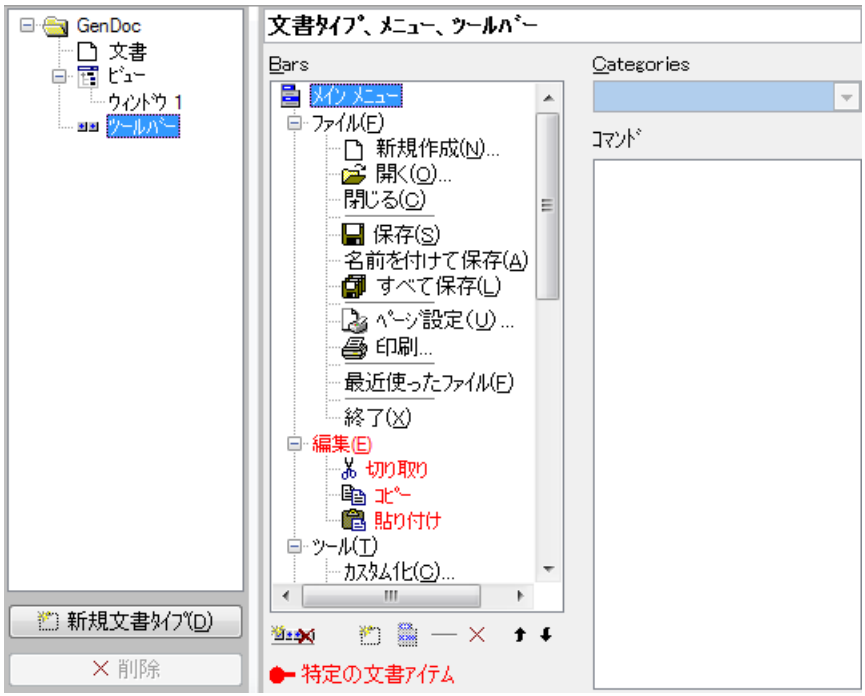


***図2.11** [文書タイプ・メニューとツールバー]*

アクション・パラメータの設定

Application Framework Editor のパレットから [*アクション*] を選択し、Application Framework Editor でアクション・パラメータを設定します。

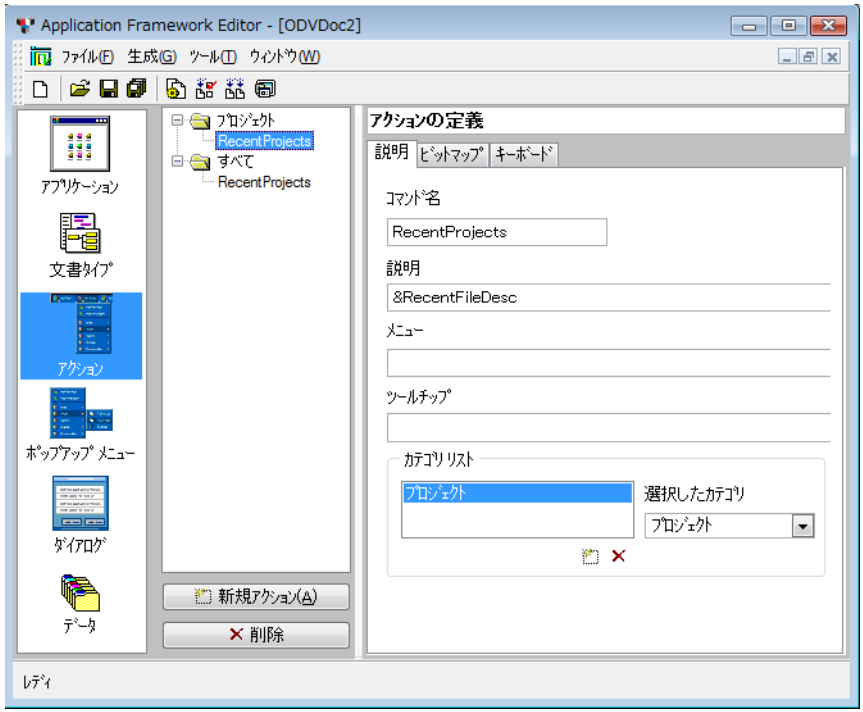


***図2.12** パレットから選択した [アクション]*

中央の列にアクション・ツリーが表示されます。ツリーにある `RecentProjects` のどちらかを選択すると、[アクションの定義] が表示されます。

アクションの定義

[アクションの定義] 作業領域には、次に示すタブがあります。

◆ [*説明*] タブには、アクションに関する基本情報が含まれます。

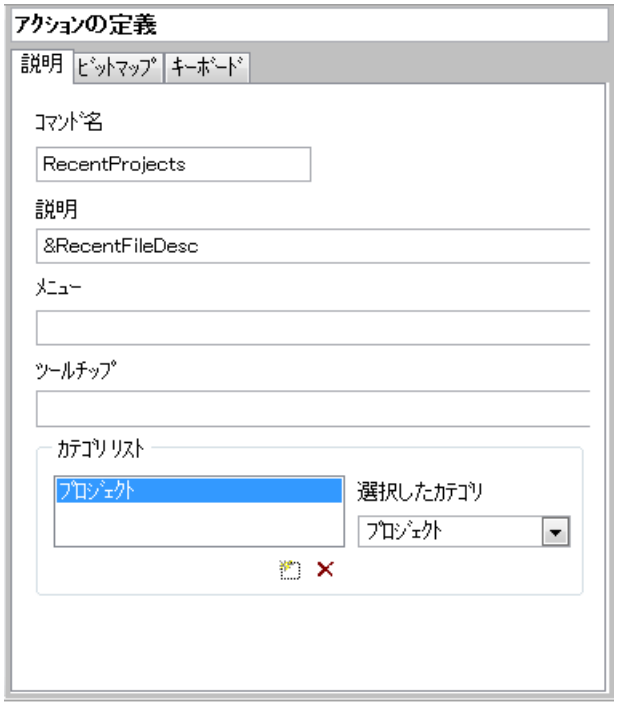


図 2.13 *アクション定義の [説明] タブ*

- [コマンド名] : アクションを識別するために、ユーザが付ける名前です。
- [説明] : このフィールドにユーザは、アクションの説明を入力できます。
- [メニュー] : メニューに表示される項目名 (たとえば [開く]、[新規作成]、[保存] など) です。
- [ツールチップ] : ツールチップ・テキストに表示される項目名 (たとえば 、[新規作成 (Ctrl+N)]) です。
- [カテゴリ・リスト] : アクションのカテゴリが表示されます。
- [選択したカテゴリ]: 選択できる現在のカテゴリ ([アプリケーション],[ファイル]、[プロジェクト] など) が一覧表示されます。

[説明]、[メニュー]、[ツールチップ] などの項目はテキストでも、`.dbm` ファイルに含まれるテキストの `&identifier` 形式のメッセージ識別子でも構いません。(`.dbm` ファイル・タイプについては、*IBM ILOG Views Foundation ユーザ・マニュアル*を参照してください。)

◆ [*ビットマップ*] タブには、このアクションに関連付けられたビットマップ・アイコンの特性が表示されます。このアイコンは、メニュー、ツールバー、ツリー・リストにアクション名と一緒に表示されます。

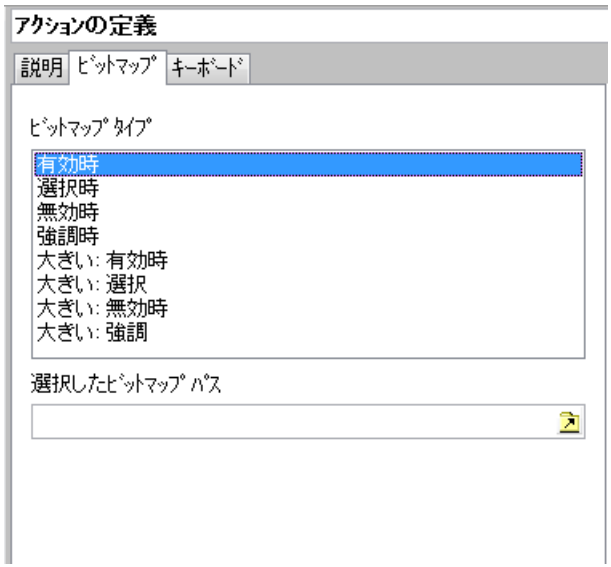


図 2.14 *アクション定義の [ビットマップ] タブ*

- [ビットマップ・タイプ] : 文字列リストにあるそれぞれ違うタイプに対して、ビットマップのセットを定義できます。これらのビットマップは、アクションの状態に応じて使用されます。
- [選択したビットマップ・パス] : 選択したビットマップ・タイプのアイコンが存在するパスです。

◆ [キーボード] タブには、アクションのキーボード・ショートカットが表示されます。

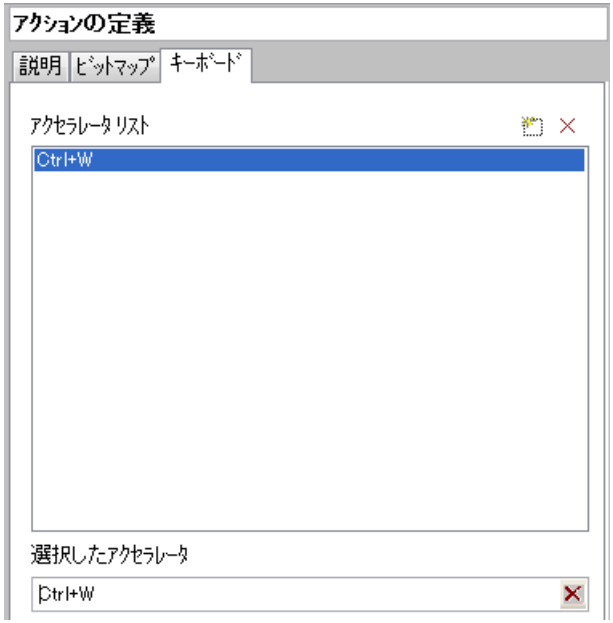


図 2.15　アクション定義の [キーボード] タブ

- [アクセラレータ・リスト] : アクセラレータを追加する場合は、[追加] ボタン をクリックして、[選択したアクセラレータ] フィールドを使用します。たとえば、[Ctrl+W] がデフォルトで最初のアクセラレータとして追加されます。
- [選択したアクセラレータ] : 現在のアクセラレータ、またはキーボード・ショートカットです。ショートカットを変更する場合は、フィールドの中をクリックしてからキーボードからショートカットのシーケンスを入力します。

アクションの作成

アクション実装の詳細については、5 章 *アクション*を参照してください。

ポップアップ・メニューのパラメータ設定

Application Framework Editor のパレットから [*ポップアップ・メニュー*] を選択し、Application Framework Editor でポップアップ・メニューのパラメータを設定します。

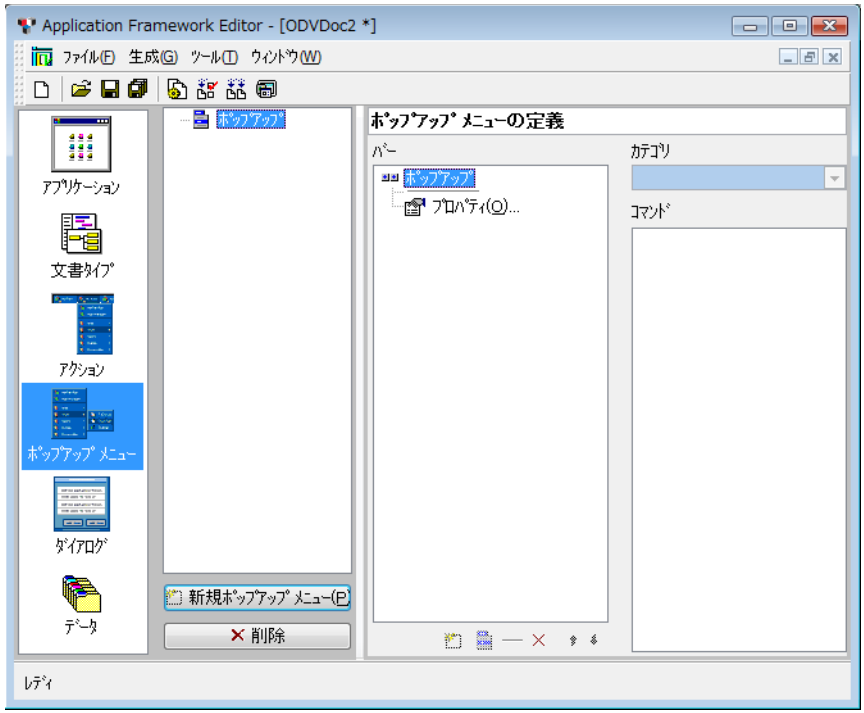


図2.16 *パレットから選択した [ポップアップ・メニュー]*

ポップアップメニューの追加を開始すると、[ポップアップ・メニューの定義] 作業領域がアクティブになります。

[ポップアップ・メニューの定義]

[ポップアップ・メニューの定義] 作業領域では、作成するアプリケーションからアクセスできるポップアップを定義できます。

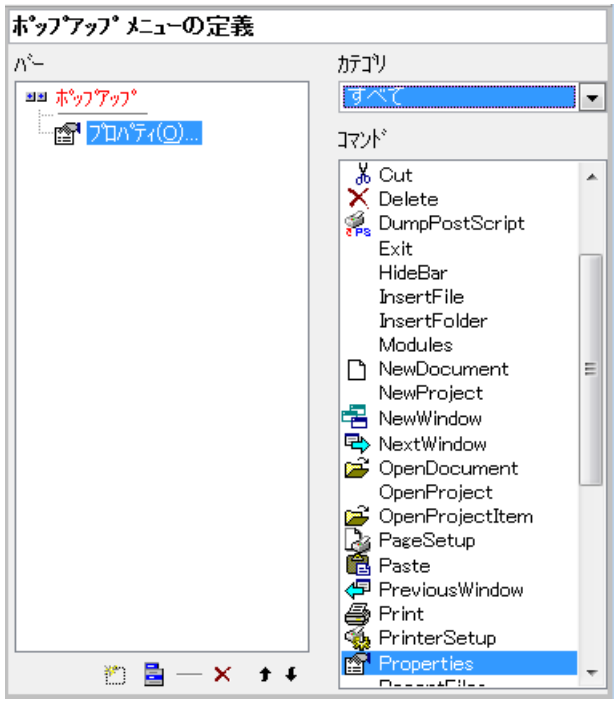


図2.17 [ポップアップ・メニューの定義]

- このツリーには、ポップアップ・メニューのレイアウトが示されます。ポップアップ項目を選択すると、残りのフィールドがアクティブになります。
- [カテゴリ]: 可能なカテゴリを示すリストです。アクションをより容易に検索するための、アクション・カテゴリが選択できます。カテゴリをすべて選択すると （図を参照）、全コマンドがアルファベット順に [コマンド] リストに表示されます。
- [コマンド] : 選択したカテゴリで使用できるコマンドです。

ポップアップ・メニューの作成

新規ポップアップ・メニューの定義を開始するには、メイン・ツリーの下にある [新規ポップアップ] ボタン をクリックします。この列に、作成したポップアップ・メニューであるツリーの新項目 (図 2.16 を参照) が表示されます。作成したポップアップにはデフォルトで、2 つの項目 (セパレータ項目とプロパティ項目) を持つレイアウトが用意されますが、このレイアウトは変更できます。

ポップアップ・メニューのデフォルト名は、Popup*xx* で、*xx* は、複数のポップアップ・メニューを挿入すると自動的に増分されるインクリメンタル番号です。

[ポップアップ・メニューの定義] ウィンドウでルート項目を選択してから、F2 キーを押して新しい名前を入力すれば、このポップアップ名が変更できます。

`IlvDvApplication::readPopup(const IlvSymbol*)` 関数を使うと、コード内でこのポップアップを取得できます。

[ポップアップ・メニューの定義] ウィンドウでは、項目の追加または削除によってポップアップのレイアウトを定義できます。ポップアップ・メニューへの項目追加については、*ポップアップ項目の追加*を参照してください。

ポップアップ項目の追加

1. 新規項目を挿入するポップアップ・レイアウトの項目を選択します。選択した項目の後に、新規項目が挿入されます。
2. [新規アクションの挿入] ボタン  をクリックします。
3. [コマンド] リストの関連するアクションを選択すると、挿入した項目を変更できます。

ポップアップ項目を変更する場合は、目的の項目を選択してから [コマンド] リストの新規アクション選択によって変更します。

新規コマンドを挿入する場合は、*アクションの作成*を参照してください。

ポップアップから項目を削除する場合は、削除する項目を選択してから [削除] ボタン  をクリックします。

新規ポップアップ・サブメニューの追加

[ポップアップ・メニューの定義] 作業領域では、ポップアップにサブメニューを追加できます。

- [ポップアップ・メニューの定義] ツリーで、サブメニューを挿入する項目を選択します。選択した項目の後に、サブメニューが挿入されます。
- [ポップアップ・メニューの定義] ツールバーにある、[新規作成] ポップアップ・メニュー・ボタン  をクリックします。
- ポップアップ項目のラベルを変更してから (F2 アクセラレータ使用)、新しいラベルを入力します。
- サブメニューの項目を追加または編集することでサブメニューを変更します (*ポップアップ項目の追加*を参照してください)。

ダイアログのパラメータ設定

Application Framework Editor のパレットから [*ダイアログ*] を選択し、Application Framework Editor でダイアログ・ボックスのパラメータを設定します。

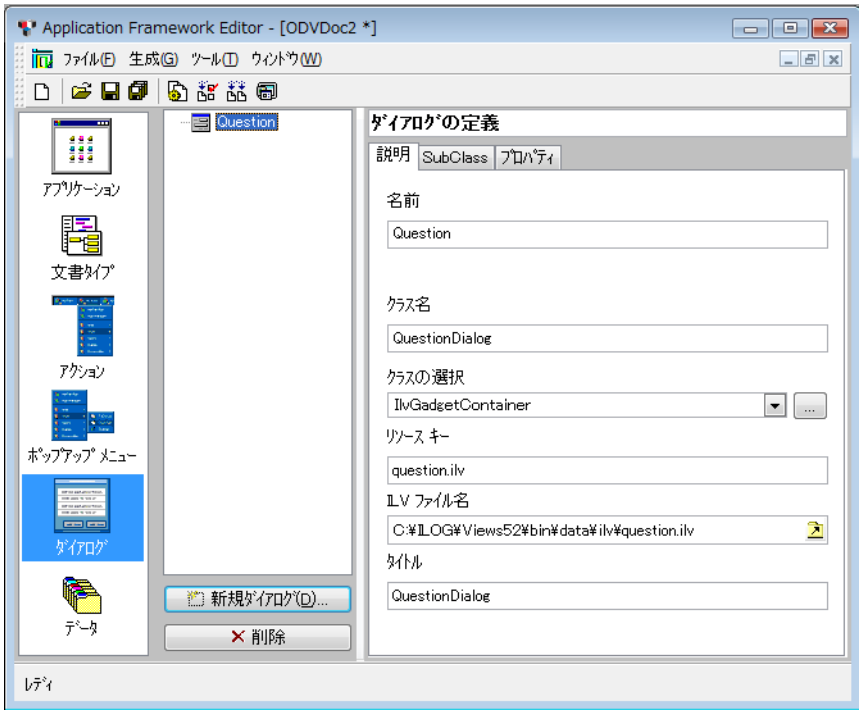


図 2.18 *パレットから選択した [ダイアログ]*

ダイアログの追加を開始すると、[ダイアログの定義] 作業領域がアクティブになります。

ダイアログの定義

[ダイアログの定義] 作業領域で、ダイアログ・ボックスのプロパティを定義できます。ダイアログ・ボックスは最初に、IBM ILOG Views Studio 内で定義する必要があります。

[ダイアログの定義] 作業領域には、次に示すタブがあります。

◆ [*説明*] タブには、ダイアログ・ボックスに関する基本情報が含まれます。

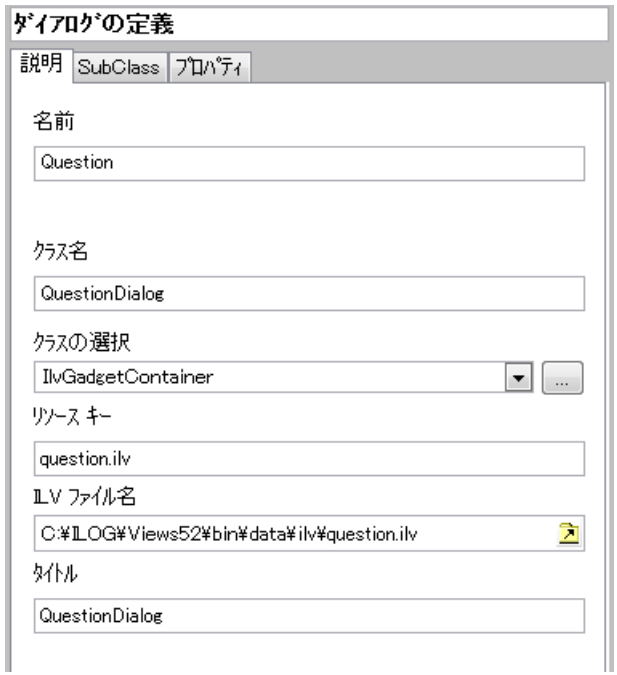


図 2.19 *ダイアログ定義の [説明] タブ*

- [名前] : この名前のデフォルトは .ilv ファイルの名前で、たとえば Question ならば、読み込まれるファイルは question.ilv です。
- ［クラス名］: この名前のデフォルトは [名前]+Dialog で、たとえば図の例では、QuestionDialog となっています。この名前はコード生成で使用されます。
- ［クラスの選択］: ダイアログ・ボックスの派生元となる IBM ILOG Views クラスです。リストの中から選択します。
- ［リソース・キー］: ファイルの再読み取りに使用されるリソース名で、デフォルトは .ilv ファイル名です。
- [ILV ファイル名] : 読み込まれた .ilv ファイルの完全パス名です。
- [タイトル]: このタイトルは、Windows のタイトル・バーに表示されます。デフォルトは [クラス名] です。

- *[サブクラス]* タブには、ダイアログ・ボックスのサブクラス名が含まれます。この項目はオプションです。

図 2.20 *ダイアログ定義の [サブクラス] タブ*

- [*プロパティ*] タブでは、ダイアログ・ボックスの標準プロパティを指定できます。任意のプロパティ、またはこれらプロパティのすべてを選択します。

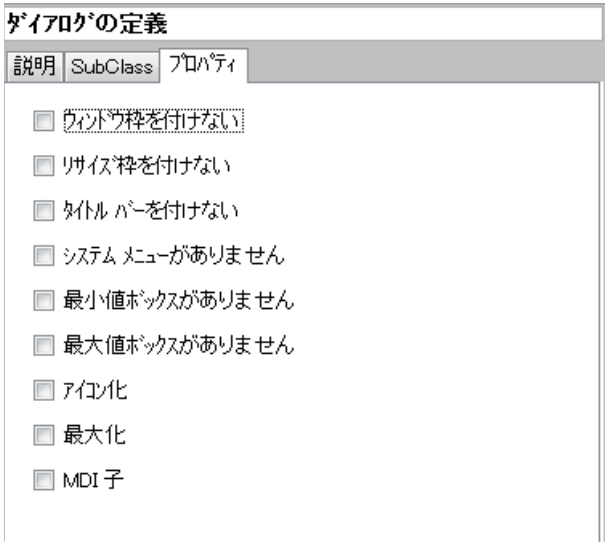


図 2.21 *ダイアログ定義の [プロパティ] タブ*

ダイアログ・ボックスの作成

定義を開始する場合は、[新規ダイアログ] ボタン  をクリックします。定義済みの .ilv ファイルを開くように要求されます。

> ***メモ:*** *Application Framework Editor でダイアログ・ボックスを作成する場合は、まずIBM ILOG Views Studio でダイアログ・ボックスを定義し、それを .ilv ファイルに保存する必要があります。Application Framework Editor でダイアログ・ボックスを作成するとき、このファイル名が必要です。*

中央の列に [ダイアログ] ツリーが、右には [ダイアログの定義] 作業領域が表示されます。[ダイアログの定義] タブで必要なデータを完成させます (詳細については *ダイアログの定義*を参照してください)。

データ・パラメータの設定

Application Framework Editor のパレットから [データ] を選択し、Application Framework Editor でデータ・パラメータを設定します。

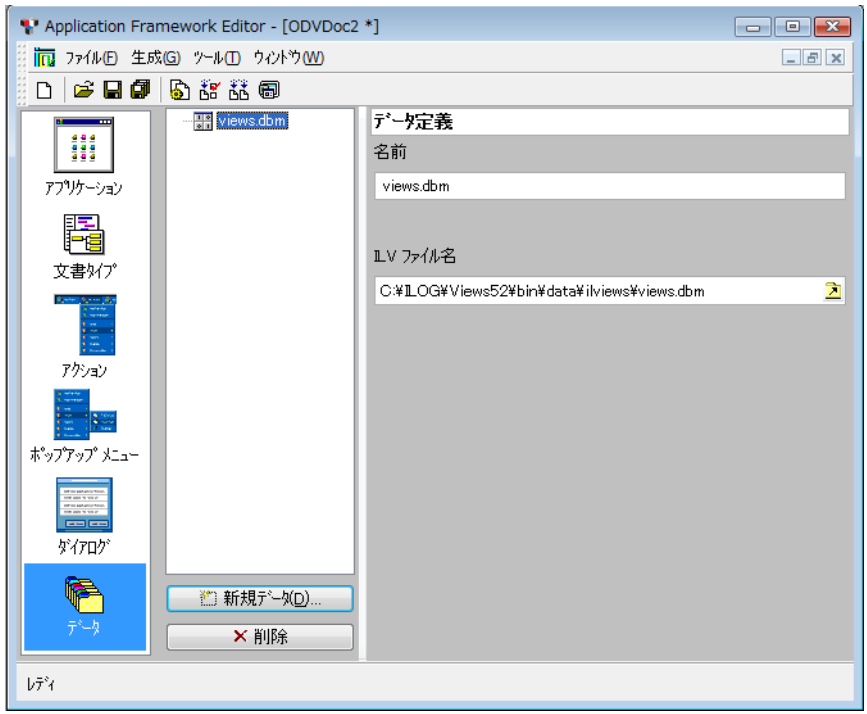


***図 2.22** パレットから選択した [データ]*

この機能を使って、実行可能なアプリケーションにデータ・ファイルが追加できます。追加できるのは任意のデータ・ファイルで、.dbm、ビットマップ、.ilv、またはその他のユーザ・データ・ファイルです。

[新規データ] ボタン をクリックして、データ・ファイルの追加を開始すると、[データ定義] 作業領域がアクティブになります。

データの定義

[データ定義] 作業領域 (図 2.22 を参照) では、ファイルを含めるために必要なデータ・プロパティが定義できます。

- [名前]: データの取得に必要なファイルの再読み込みで使用される名前です。デフォルトは、読み込まれたファイルの名前です。テキスト・フィールドの編集で、この名前を変更できます。
- [ILV ファイル名] : ILV ファイルの完全パス名です。 のクリックと新しいパスの選択で、このパス名を変更できます。

パラメータの生成

Application Framework Editor によってアプリケーション・パラメータを定義したら、それらを生成する必要があります。

[生成] メニューでは、アプリケーションを生成するためのコマンドが提供されます。

パラメータ・コマンド

最初の生成では、[生成] > [パラメータ] を選択すると、[プロジェクトの生成] ウィンドウが表示されます。

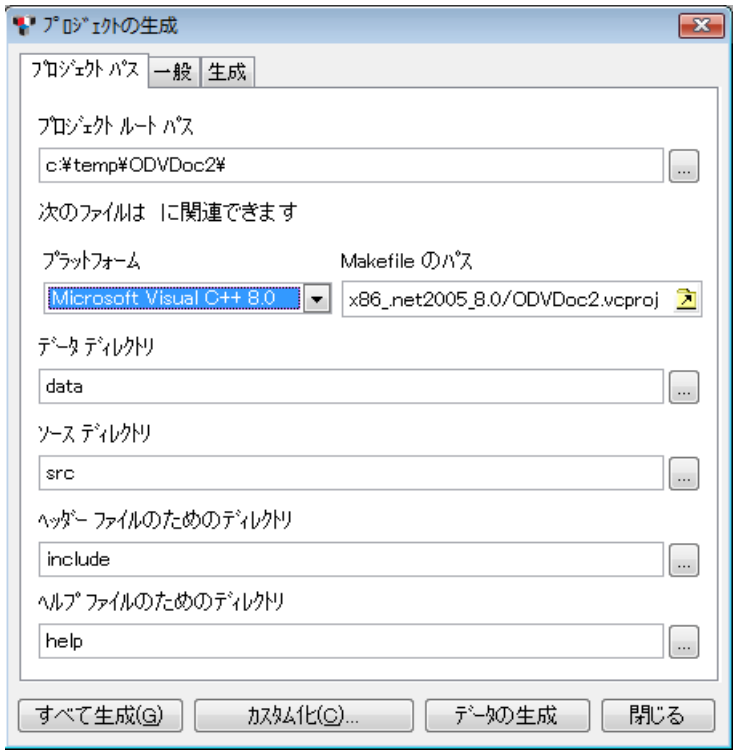


図 2.23 *[プロジェクトの生成] ウィンドウ、[プロジェクト・パス] タブ*

このウィンドウには 3 つのタブがあります。

- [*プロジェクト・パス*] タブ (図 2.23 を参照) には次のフィールドが含まれます。
  - [プロジェクト・ルート・パス] : プロジェクトを保存するルート・パスです。以下の各パスは、このルート・パスに対する相対パスで指定できます。

- [プラットフォーム] : Makefile のプラットフォームです。
- [Makefile のパス]: [プラットフォーム] 選択に基づく、Makefile のパスです。
- [ディレクトリ]: [データ] ファイル、[ソース] ファイル、[ヘッダー] ファイル、[ヘルプ] ファイルの各ディレクトリです。図のように、それぞれのフィールドで指定します。すべてに所定のデフォルトがありますが、変更可能です。

◆ [*一般*] タブでは、一般情報フィールドを設定できます。

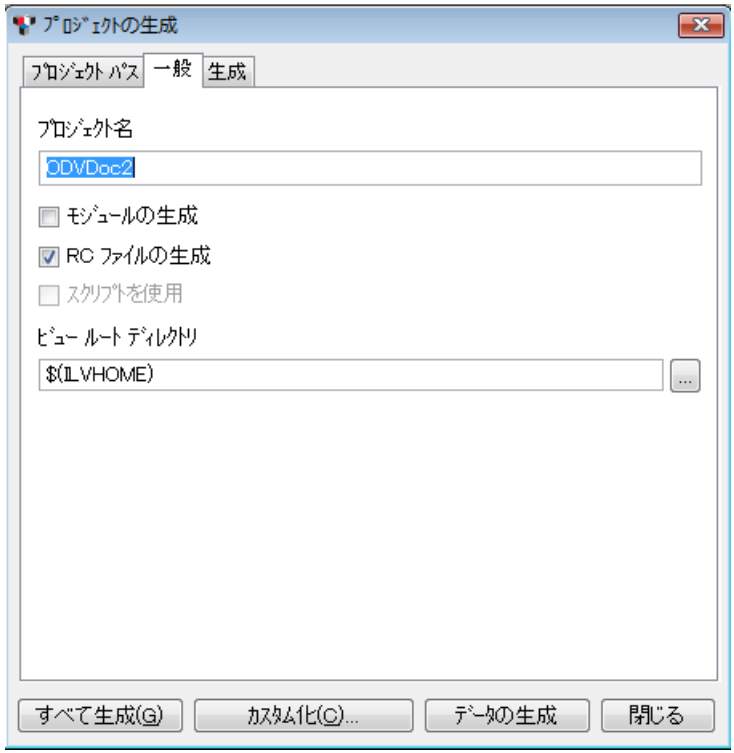


図 2.24 *[一般] タブ (プロジェクト生成)*

◆ [*生成*] タブでは、プロジェクト生成に関する情報が提供されます。

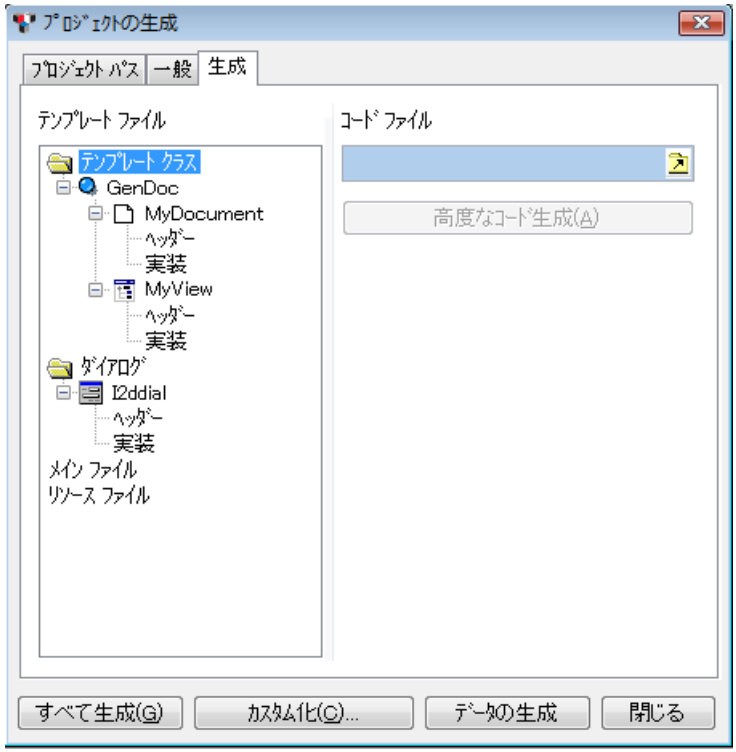


図 2.25 *[生成] タブ (プロジェクト生成)*

[すべて生成]

[すべて生成] ボタンで、アプリケーションの全ファイルを生成できます。

> ***重要:*** *この操作によって、アプリケーションの既存生成ファイルはすべて置換されます。置換処理を進める前に、確認を求めるダイアログ・ボックスが表示されます。*

[カスタム化]

[カスタム化] ボタンを使用すると、アプリケーションの一部だけ、または選択した部分だけ生成されます。たとえば新しいダイアログ・ボックスを追加する場合、この方法が使えます。

[データの生成]

[データの生成] ボタンは、ソース・コードの不要な更新やアプリケーションを最初に生成した後の Makefile 実行に使用します。たとえば、アクション、文書タイプ、ポップアップ、新しいデータ・ファイルの追加などに使用できます。

GUI アクション・サマリー

表 2.3 *メニュー操作とツールバー操作*

アクション	ツールバー・アイコン	メニュー操作	コメント
ファイル操作			
新規プロジェクトの作成		[ファイル] > [新規作成]	
.odv ファイルの読み込み		[ファイル] > [開く]	
現在の .odv ファイルを閉じる		[ファイル] > [閉じる]	
.odv ファイルの保存		[ファイル] > [保存] [ファイル] > [名前を付けて保存]	[名前を付けて保存] する場合は、新規ファイル名と拡張子を入力します。
開いているすべての .odv ファイルを保存		[ファイル] > [すべて保存]	
生成操作			
プロジェクトの生成パラメータ設定		[生成] > [パラメータ]	
アプリケーションすべてを生成		[生成] > [すべてのアプリケーションを生成する]	
現在のアプリケーションの特定ファイルを生成		[生成] > [特定ファイルの生成]	[カスタム化] ダイアログ・ボックスを開く
データの生成		[生成] > [データの生成]	生成レポート・ログを表示

表 2.3 メニュー操作とツールバー操作 （続き）

アクション	ツールバー・アイコン	メニュー操作	コメント
ツール			
アプリケーションのカスタマイズ		[ツール] > [カスタム化]	[カスタム化] ウィンドウを開く
モジュールの挿入 / 削除		[ツール] > [モジュール]	[モジュールの挿入 / 削除] ダイアログ・ボックスを開く
プロジェクトのスクリプト作成		[ツール] > [プロジェクトのスクリプト作成]	スクリプト・プロジェクト・ファイル (.spj) を作成または開く
ウィンドウ操作			
新規ウィンドウの起動		[ウィンドウ] > [新規ウィンドウ]	
次のウィンドウを表示		[ウィンドウ] > [次のウィンドウ]	
前のウィンドウを表示		[ウィンドウ] > [前のウィンドウ]	
すべての文書ビューを重ねて表示		[ウィンドウ] > [ウィンドウを重ねて表示]	
すべての文書ビューを水平に並べて表示		[ウィンドウ] > [ウィンドウを水平に並べて表示]	
すべての文書ビューを垂直に並べて表示		[ウィンドウ] > [ウィンドウを垂直に並べて表示]	
Application Framework Editor の終了		[ファイル] > [終了]	終了する前に未保存ファイルについて確認

章

3

アプリケーションの実装

本章では、Application Framework によるアプリケーションの実装方法に加え、必要なクラスやファイルの記述について説明します。このセクションでは、次のトピックを扱います。

- *Application Framework の機能*
- *オプション・ファイル*
- *メイン・ファイル*
- *文書クラスの実装*
- *コマンド*
- *文書ビュー・クラスの実装*

Application Framework の機能

Application Framework は、ドキュメント / ビュー・アーキテクチャ (*ドキュメント / ビュー・アーキテクチャ*を参照) 上に構築されています。図 3.1 に、ドキュメント / ビューのメカニズムが依存するさまざまなクラスを示します。

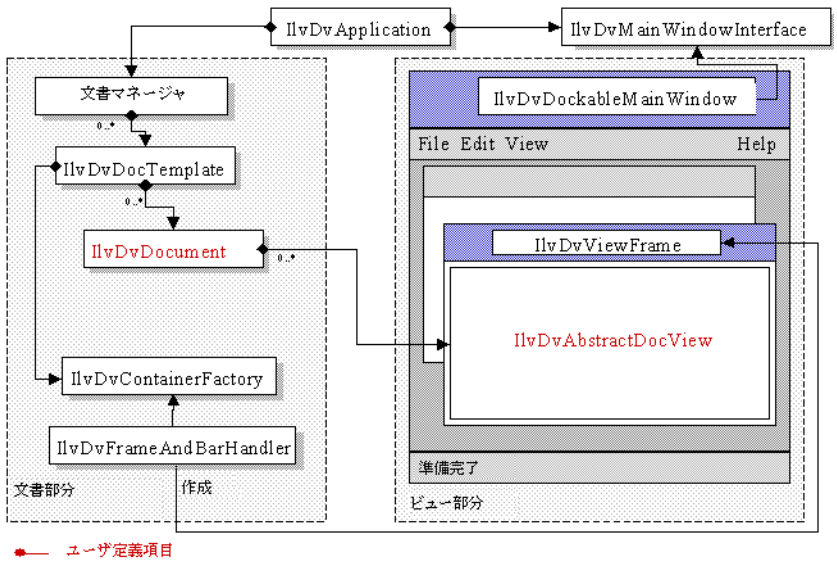


図3.1 文書/ビュー・クラス

`IlvDvApplication` クラス、`IlvDvDocument` 階層、`IlvDvDocViewInterface` 階層を除いて、図示のクラスはすべて非表示で、開発者からは見えません。これらクラスのインスタンスは、アプリケーションの初期化中に、読み込まれるアプリケーション・オプションに従って、自動的に生成されます。

Application Framework アプリケーションのコードは、次から構成されます。

- ***オプション・ファイル***: Application Framework Editor によって編集される、1 つ以上のオプション・ファイルです。
- ***メイン・ファイル***: プログラムのメイン・エントリ・ポイントを含むメイン・ファイルで、1 つの `IlvDvApplication` (または派生クラス) オブジェクトをインスタンス生成する必要があります。このファイルは Application Framework Editor によって生成されますが、`Text` サンプルで示すように、非常に特別な場合のみそれを完成させる必要があります。
- ***文書クラスの実装***: 文書クラスを実装するファイルで、`IlvDvDocument` のサブクラスです。
- ***文書ビュー・クラスの実装***: 文書ビュー・クラスを実装するファイルで、`IlvDvDocViewInterface` のサブクラスです。

オプション・ファイル

初期化中に Application Framework アプリケーションは、3 つのオプション・ファイルからデータを読み込みます。このデータは、メニューやツールバーの内容、最近使用したファイルのリスト、文書テンプレートなどです。

オプション・ファイルは次のとおりです。

- Application Framework オプション・ファイル。パスは `<ILVHOME>/data/ilviews/appframe/docview.odv` で、`<ILVHOME>/data/res/appframe.rc` ファイルで含めることができます。

  このファイルには、デフォルト・アクション (OpenDocument、SaveDocument、Cut、Copy など)、デフォルト・メニュー、デフォルト・ツールバーの記述が含まれます。

- アプリケーション・オプション・ファイル。ファイル・パスは、`IlvDvApplication` オブジェクトに、`IlvDvApplication::setAppOptionsFilename` メソッドによって与えられます。

  このファイルは Application Framework Editor によって編集され、次の情報を含みます。

  - メイン・ウィンドウのアプリケーション名とタイトル
  - 異なる文書テンプレートの記述
  - Application Framework オプション・ファイルに格納されたデフォルトと違う場合の、メイン・メニューとツールバー
  - アクションの説明
  - ユーザ・アプリケーション・データ

- ユーザ・プロファイル・オプション・ファイル。デフォルト・パスは次のように与えられます。

  - Windows の場合。

    ```
    <Windows directory>/Profiles/<Username>/Application Data/
    <Application name>.odv
    ```

  - UNIX の場合 :

    ```
    $(HOME)/<Application name>.odv
    ```

  違うパスを指定する場合は、次のメソッドを使用します。

  ```
  IlvDvApplication::setUserOptionsFilename
  ```

このファイルには、アプリケーションの最後の実行時に、ユーザによって変更されたアプリケーション・データが含まれます。主として次のようなデータです。

- 最近使用したファイル・リスト
- アプリケーション・メイン・ウィンドウの位置とサイズ
- ドッキング可能なツールバーすべての位置と、状態 (表示または非表示)
- ツールバーの内容のカスタマイズ
- アクションのカスタマイズ

メイン・ファイル

Application Framework アプリケーションを作成する場合、主要手順の `IlvDvApplication` オブジェクトを、単純な IBM® ILOG® Views アプリケーションで `IlvApplication` オブジェクトを作成するのと同じ方法で作成します。

> ***メモ:*** *メイン・ファイルは、Application Framework Editor によって自動的に作成されます。*

```
int
main(int argc, char* argv[])
{
   IlvDvApplication* app = new IlvDvApplication("", 0, argc, argv);
   IlvDisplay* display = app->getDisplay();
   if (!display || display->isBad()) {
      IlvFatalError("Couldn't create display");
      delete display;
      return -1;
   }
   // Adding the options file
   app ->setAppOptionsFilename((const char*)"myapp.odv");

   // Adding the data base file
   display->getDatabase()->read((const char*)"myapp.dbm", display);

   // Continue...
   application->run();
   return 0;
}
```

`IlvDvApplication` は `IlvApplication` のサブクラスで、オプション・データの管理機能および、メニュー項目とツールバー項目の処理機能に加え、アクションとその状態の処理機能があります。

この IlvDvApplication オブジェクトは、ドキュメント / ビュー・メカニズムに関連するオブジェクトのすべてを認識します (図 3.1 を参照)。同様にまた、これらオブジェクトのすべては、このアプリケーション・オブジェクトを認識します。たとえば、アクションの状態を文書または、文書ビューから変更する場合に、このアプリケーション・オブジェクトが役立ちます。

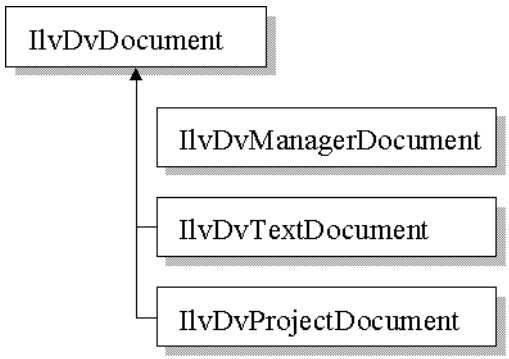
文書クラスの実装

文書クラスは、IlvDvDocument クラスから派生します。

文書は *ユーザ・データ* です。文書クラスは、データを読み込み、保存し、さらにデータを変更するための文書ビューで使用されるアクセサも提供します。

> ***メモ:*** *このユーザ・データは、Model View Controller (MVC) アプローチに似ています。*

Application Framework では、次に示す継承ツリーに見られるような、IBM ILOG Views マネージャ、テキストバッファ、またはプロジェクトなど、特定のデータを管理する文書クラスが提供されます。



新規文書

派生した文書クラスは、IlvDvDocument::initializeDocument メソッドをオーバーライドする必要があります。

文書作成のため、[ファイル]> [新規コマンド] を実行すると、このメソッドが呼び出されます。

メソッドはまず、IlvDvDocument::initializeDocument. を呼び出す必要があります。次に、特定データの初期化が必要です。

シリアル化

IlvDvDocument::serialize メソッドは次のとおりです。

```
void IlvDvDocument::serialize(IlvDvStream& stream);
```

文書作成のためにファイルを開くと呼び出され、stream.isSaving() が呼び出されると false を返します。その他の場合は、文書を保存する必要があります。

通常、このメソッド本体は次の形式に従います。

```
IlvDvDocument::serialize(stream);
if (stream.isSaving()) {
   // Here, write your persistent data
}
else {
   // Here, read data from stream
}
```

ストリームと呼ばれるパラメータによって、データを読み込み、保存する 2 つの方法があります。

istream または ostream オブジェクトを使うのが最初の方法です。これらのオブジェクトは、istream* getInStream() const メソッド、および ostream* getOutStream() const メソッドを直接呼び出すことで与えられます 。

2 番目は、IlvDvStream クラスによって提供される特定のシリアル化メソッドを使う方法で、通常はこの方が簡単です。これらのメソッドを次に示します。

- 演算子

```
// Storing operators
IlvDvStream& operator<<(IlvInt i);
IlvDvStream& operator<<(IlvUShort w);
IlvDvStream& operator<<(IlvShort ch);
IlvDvStream& operator<<(IlvUInt u);
IlvDvStream& operator<<(IlvBoolean b);
IlvDvStream& operator<<(IlvFloat f);
IlvDvStream& operator<<(IlvDouble d);
IlvDvStream& operator<<(const IlvString& s); //?s? must not contain blanks

// Reading operators
IlvDvStream& operator>>(IlvInt& i);
IlvDvStream& operator>>(IlvUShort& w);
IlvDvStream& operator>>(IlvShort& ch);
IlvDvStream& operator>>(IlvUInt& u);
IlvDvStream& operator>>(IlvBoolean& b);
IlvDvStream& operator>>(IlvFloat& f);
IlvDvStream& operator>>(IlvDouble& d);
IlvDvStream& operator>>(IlvString& s);
```

- `void serialize(IlvString&, IlvBoolean betweenQuotes = IlvTrue);`

このメソッドは、文字列を読み込み、保存するのに安全な方法です。betweenQuotes パラメータが true に設定されている場合、文字列は引用符に囲まれて保存されます。この方法では、空白スペースを含むことができます。

- `void serializeBitmap(IlvBitmap*&, IlvBoolean lock = IlvTrue);`

  ビットマップ・パスをシリアル化します。

- オブジェクトのシリアル化

  ユーザ・クラスを実装する場合は、IlvDvSerializable クラスから派生させるようにしてください。このクラスは抽象インターフェースで、安全なダウンキャストのメカニズムと、シリアル化メソッドの両方を提供します。

  `virtual void serialize(IlvDvStream& stream);`

  - `void serializeObjects(IlvArray&);`

    IlvDvSerializable オブジェクトの配列を読み込み、保存します。

  - `void writeObject(const IlvDvSerializable*);`
  - `IlvDvSerializable* readObject();`

- `virtual void clean();`

  このメソッドは、文書データをクリーンアップするために呼び出されます。対応する文書タイプが、複数の文書を開けない文書 (SDI 文書タイプで、通常はプロジェクト文書) の場合のみ使用されます。同じタイプの文書がすでに 1 つ開いているときにユーザが新規文書を作成しようとしても、Application Framework は現在開いている文書を、もう 1 つの文書作成のために削除できません。代わりに、(このメソッドを呼び出すことで) 開いている文書をクリーンアップし、それを再初期化します。

コマンド

文書のデータを変更する場合は、次のような利点がある Application Framework コマンド・メカニズムの使用を推奨します。

- 元に戻す/やり直すメカニズム - [元に戻す]、[やり直す]、[繰り返す] アクションが自動的に、それらの状態とともに処理されます。
- 文書の変更状態は自動的に管理されます。未変更文書にコマンドを追加すると、その文書は変更済みとしてマークされます (文書に関連するビューを含む、フレームのタイトルに星が表示されます)。同様に、このコマンドを元に戻すと、文書の未変更状態が復元され (フレームのタイトルから星も削除され) ます。
- 文書に加えられたすべての変更のログを保持します。

たとえば、次に示す文書クラスを考えてみましょう。

```
class MyDocument
: public IlvDvDocument
{
...
   void setX(int x) { _x = x; }
   int getX() const { return _x; }
   protected:
   int _x;
};
```

文書ビューのイベント / アクションまたは、文書アクションを処理する一方で、文書の X プロパティを変更する場合、文書の setX メソッドを直接呼び出さないようにしてください。このプロパティを変更する (この例では ChangeXPropertyCommand と呼ばれる)、コマンド・クラスの実装がより適切です。

```
class ChangeXPropertyCommand
: public IlvDvCommand
{
   ChangeXPropertyCommand(MyDocument* document, int newX)
   : _document(document),
   _newX(newX)
   {
      _oldX = document->getX();
   }
   virtual void doIt() { setX(_newX); }
   virtual void undo() { setX(_oldX); }
   void setX(int x) { _document->setX(x); }
   protected:
   MyDocument* _document;
   int _newX;
   int _oldX;
};
```

したがって、ビューまたは文書それ自体の実装は、(文書の setX メソッドを直接呼び出す代わりに) X プロパティを変更する次のコードを呼び出す必要があります。

```
document->doCommand(new ChangeXPropertyCommand(document, newX));
```

このコードは、その doIt メソッドを呼び出すことで ChangeXPropertyCommand コマンドを実行し、文書によって内部管理されるコマンド履歴内にそれを格納します。

コマンドの管理には、次に示す IlvDvDocument クラスのメソッドを使用します。

```
void doCommand(IlvDvCommand* cmd,
               IlvBoolean updateUI = IlvTrue,
               IlvBoolean bSetModified = IlvTrue);
```

このメソッドは、コマンド・オブジェクト cmd を内部コマンド履歴に追加するため、呼び出されます。次に、その IlvDvCommand::doIt メソッド呼び出しにより、このコマンドが実行されます。updateUI パラメータは、[元に戻す]、[やり直

す]、[繰り返す] コマンドの UI の更新を指定します。bSetModified パラメータは、文書変更フラグの、true に設定する必要の有無を指定します。

[元に戻す]/[やり直す]/[繰り返す] アクション

[元に戻す]、[やり直す]、[繰り返す] アクションは、文書によって自動的に管理されます。これらのアクションを処理するため、文書は次のメソッド (特定用途のためオーバーライドできます) を呼び出します。

- `virtual IlvBoolean canUndo() const;`

  実行したばかりのコマンドを元に戻せる場合、このメソッドは true を返します。たとえば、文書が変更されなかったときのように、元に戻せるコマンドがない場合、このメソッドは false を返します。

- `virtual void undo(IlvBoolean bUpdateUI = IlvTrue);`

  このメソッドは、実行した最後のコマンドの undo メソッドを呼び出します。bUpdateUI パラメータは、[元に戻す]、[やり直す]、[繰り返す] コマンドの UI の更新を指定します。

- `virtual IlvBoolean canRedo() const;`

  元に戻したばかりのコマンドを、IlvDvCommand::doIt メソッドの呼び出しでやり直せる場合、このメソッドは true を返します。たとえば、文書が変更されなかったときのように、やり直せるコマンドがない場合、このメソッドは false を返します。

- `virtual void redo(IlvBoolean bUpdateUI = IlvTrue);`

  このメソッドは、元に戻した最後のコマンドの doIt メソッドを呼び出します。bUpdateUI パラメータは、[元に戻す]、[やり直す]、[繰り返す] コマンドの UI の更新を指定します。

- `virtual void repeat(IlvBoolean bUpdateUI = IlvTrue);`

  このメソッドは、最後に実行したコマンドを繰り返します。IlvDvCommand::copy メソッドの呼び出しによって、このコマンドをコピーします。コピーをコマンド履歴に追加してから実行します。bUpdateUI パラメータは、[元に戻す]、[やり直す]、[繰り返す] コマンドの UI の更新を指定します。

データに加えられた変更を関連ビューへ反映させる

ここまでコマンドによる文書データの変更方法を検討してきましたが、文書の関連ビューにそれらの変更が反映される方法を、さらに考える必要があります。

1 つの文書は、タイプの違う複数のビューを持つことができます。したがって、それらのビューに通知するため文書は、タイプに応じ各ビューによって解釈される

一般メッセージを送信します。一般メッセージをビューに送信するため、文書は次のメソッドを使用します。

```
void notifyViews(const char* messageName,
                 IlvDvDocViewInterface* exceptView, ...);
```

- メッセージの名前は `messageName` です。このメッセージ名には、(Copy、OpenDocument、など) アクションの名前は付けないでください。
- `exceptView` パラメータは、通知されないビューを指定します。このパラメータの値は常に 0 です。ただし、(文書に現在イベントを通知中のビューが返る) `getCurrentCallerView()` メソッドの呼び出しで返るビューにすることもできます。その場合、文書への通知前にこのビューは、内容をすでに変更している可能性があるため、通知*しない*ビューに指定することもできます。
- パラメータの変数リストは、メッセージ名に応じて変化します。たとえば、文書に従業員の情報が含まれ、コマンドがその従業員の名前を変更したばかりだとすると、文書はビューにこの変更を次のように通知します。

```
void EmployeeDocument::changeName(const char* name)
{
   _employeeName = name;
   notifyViews("NameChanged", 0 /* Notify all views */, name);
}
```

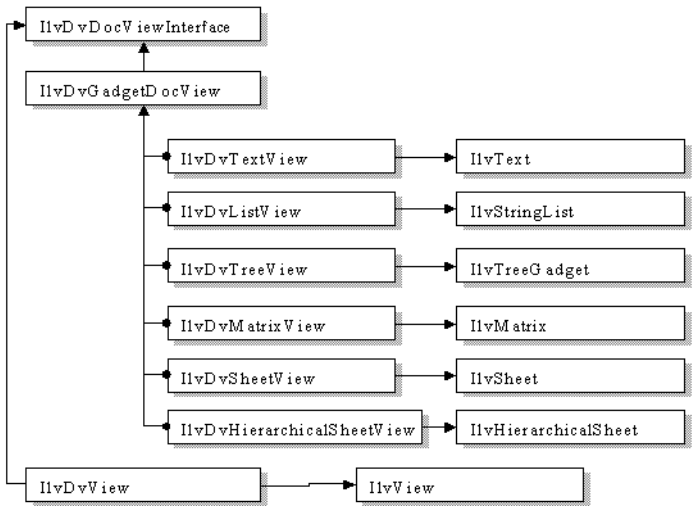
メッセージを受信するため、ビュー (または `IlvDvInterface` インターフェースを実装する他の任意のクラス) は、インターフェース宣言内のエントリを 1 つ指定する必要があります。このエントリはメッセージ名とパラメータを、クラスのメソッドに対応させます。詳細については、4 章 *Application Framework インターフェース* を参照してください。

このサンプルはこれで完成できるため、従業員文書のビューでメッセージ `NameChanged` を受信できます。

```
IlvDvBeginInterface(EmployeeView)
/* The message "NameChanged" with one parameter const char* name
   is processed by the method:
   EmployeeView::nameChanged with one parameter const char* name */

   Method1(NameChanged, nameChanged, const char*, name)
IlvDvEndInterface1(IlvDvFormView)

/* The nameChanged method is automatically called when an EmployeeDocument
   notifies its views giving the message name "NameChanged" */

void EmployeeView::nameChanged(const char* name)
{
   IlvTextField* nameField = getEmployeeNameField();
   nameField->setLabel(name, IlvTrue);
}
```

文書ビュー・クラスの実装

文書ビュー・クラスは、IlvDvDocViewInterface クラスから派生します。関連する文書の内容を示し、ユーザによる編集が可能です。次に、IlvDvDocViewInterface クラスの継承ツリーを示します。



このクラスの派生には、IlvDvDocViewInterface::initializeView メソッドのオーバーライドだけが必要です。

```
virtual void initializeView();
```

このメソッドは、文書データに従って文書ビュー・オブジェクトを初期化するために、呼び出されます。たとえば、リスト・ビューを文書に格納されたデータ・セットに従って、記入できます。メソッドの本体サンプルを次に示します。

```
void
ListView::initializeView()
{
   IlvDvListView::initializeView();
   ListDocument* document = getListDocument();
   IlvUInt count;
   Element* const* elements = document->getElements(count);
   for(IlvUInt iElement = 0; iElement < count; iElement++)
   addString(elements[iElement]->getName());
}
```

インタラクション

文書ビューが、どのようにして文書の内容を表示するかを見てきました。

しかし、さらにビューとのインタラクションによって、文書の内容を編集する必要があります。また、ビューとのインタラクションを、文書データでの変更に変換する場合もあります。

注記: *それには、Application Framework コマンドを使用することをお勧めします。*

たとえば、ビューに関連する文書内に格納された名前リストを、ユーザに編集させるリスト・ビューを考えてみます。ユーザが Del キーを押すことで、名前 (ビューで選択された名前) を削除できるようにしたいとします。その場合は、次の手順に従います。

```
// The list view tracks the event and makes the changes to the document
// through a command
void ListView::handleGadgetEvent(IlvEvent& event)
{
   if (event.type() == IlvKeyDown) {
      IlvUShort c = event.data();
      if (c == IlvDeleteKey) {
      getNamesDocument()->doCommand(new RemoveNameCommand(getNamesDocument(),
                                                          getSelectedName());
      return IlvTrue;
      }
   }
   return IlvStringList::handleGadgetEvent(event);
}

// Here is the implementation of the command class
class RemoveNameCommand
: public IlvDvCommand
{
public:
   RemoveNameCommand(NamesDocument* document, const char* name):
   _document(document), _name(name) {}
   virtual void doIt() { _document->removeName((const char*)_name); }
   virtual void undo() { _document->insertName((const char*)_name); }
protected:
   NamesDocument* _document;
   IlString _name;
};
```

これで、ビューで発生したイベントを、コマンドの使用によってビューに反映させる方法が分かりました。ただし、この変更が反映されるには、ビューをさらに更新する必要があります。

この例ではさらに、ユーザが Del キーを押したときに、選択した名前項目をリストから削除しなければなりません。そのためには、文書が関連するビューに名前リストからその名前を削除するタイミングを通知します。ビューに通知するため、文書は関連するビューにセクション コマンド で示すとおり、一般メッセージを送信します。

これで、サンプル例は完成します。

```
// The document notifies its views when it removes a name from its
```

```
// list of names
void NamesDocument::removeName(const char* name)
{
   _namesArray.removeName(name);
   notifyViews("RemoveName", 0, name);
}

// The list view updates its list when the document notifies it that it
// has removed a name from its list. First, the list view class associates
// the RemoveName message with its removeName method. Thus, this method will
// be called when the user notifies its views with the RemoveName message.

IlvDvBeginInterface(ListView)
Method1(RemoveName, removeName, const char*, name);
IlvDvEndInterface1(IlvDvListView)
void ListView::removeName(const char* name)
{
   IlShort index = getPosition(name);
   if (index != (IlShort)-1) {
      remove(index);
      reDraw();
   }
}
```

文書ビューにおけるイベント管理の詳細については、サンプル manager および synedit を参照してください。いずれも samples ディレクトリにあります。

章

4

Application Framework インターフェース

この章では、Application Framework インターフェースの使用法について説明します。このセクションでは、次のトピックを扱います。

- *インターフェース・メカニズム*
- *インターフェースの宣言*
- *マクロの命名規則*

インターフェース・メカニズム

Application Framework が提供するインターフェース・メカニズムでは、次の処理を行うことができます。

- GUI アクションの追跡と処理 (*アクション*の章を参照してください)。
- クラスに対するイントロスペクションの実行。
- クラスのスクリプト。

このインターフェース・メカニズムは、名前をクラスのメソッドまたはフィールドに関連付けます。このメソッドの名前またはフィールドは、インターフェース・メカニズムの使い方によって決まります。イントロスペクションおよびスクリプト作成では、名前がメソッドの識別を行う上で重要な役割を果たします。

インターフェースの宣言

次に示すのは、インターフェースを (A と呼ばれる) クラスに宣言する方法を示す小さなサンプルです。

```
// Declaration of class A
class A
: public IlvDvInterface
{
public:
   void setX(int x) { _x = x; } < /FONT >
   int getX() const { return _x; }

protected:
   int _x;
};

// Implementation file of A. Use the following macros to
// introspect methods setX, getX, and field _x:

IlvDvBeginInterface(A)
   Method1(SetX, setX)
   TypedMethod (GetX, getX)
   Field(X, _x)
IlvDvEndInterface()


....
// Using an instance of A as an interface, it is possible
// to invoke its methods and to modify its field
// without being aware of class A !!!
A* a = new A;
IlvDvInterface* interf = a;

// First we invoke its methods:
IlvDvValue returnedValue;
interf->callMethod(IlvGetSymbol("SetX"), &returnedValue, 100);
interf->callMethod(IlvGetSymbol("GetX"), &returnedValue);
assert((IlvInt) returnedValue == 100);

// Then, we modify its field directly:
interf->setFieldValue(IlvGetSymbol("X"), 200);
assert((IlvInt)interf->getFieldValue(IlvGetSymbol("X"),
                                     &returnedValue) == 200);
```

マクロの命名規則

このセクションでは、スクリプト作成とイントロスペクションに使用されるマクロの命名規則について説明します。

メソッドの場合。

- マクロ名のルートは必ず `Method` とします。

- 宣言したメソッドが値を返す場合、マクロ名は必ずプレフィックス `Typed` で開始します。
- 宣言したメソッドに引数が含まれる場合、マクロ名は必ずサフィックス `[Number of Parameters]` で終了します。

フィールドの場合。

- マクロ名は `Field` です。

ï 4.1: に例を示します。

表 4.1 *マクロの例*

エクスポートするメソッド	マクロ宣言
Violate getPosition() const;	TypedMethod (GetPosition, getPosition, IlvFloat)
const char* set(int);	TypedMethod1 (Set, set, int, ExportedFirstParameterName, const char*)
void setPosition(int x, int y);	Method2 (SetPosition, setPosition, int, X, int, Y)

スクリプト作成およびイントロスペクションの場合、最初のマクロ・パラメータは、2 番目のパラメータとして与えられるメソッドまたはフィールドの識別に使用されます。

イントロスペクションの詳細については、*samples* ディレクトリにある、イントロスペクションの処理サンプルを参照してください。

章

5

アクション

この章では、Application Framework で提供されるアクション・イベントの使用法について説明します。本章は、次のような構成になっています。

- *アクション・イベントの有効化*
- *アクション・イベントの処理*

アクション・イベントの有効化

複数のインターフェースによって、アクションの処理を容易にするメカニズムが、Application Framework には備わっています。このアプリケーションでは、アクション・イベントの生成によって、メニューやツールバーにおけるメニュー項目の起動を処理します。起動するメニュー項目に関連するアクションに応じて、アクション・イベントが生成されます。

アクション・イベントは、次のターゲットへ順番に送られます。

- アクティブなビュー・フレーム内のアクティブ・ビューである、アクティブな文書ビュー
- アクティブな文書ビューに関連付けられた文書
- アクティブなビュー・フレーム

- ドッキング可能なバーに挿入される、ビューとその関連文書
- メイン・ウィンドウ
- アプリケーションに宣言されたアクション・プロセッサ
- アプリケーション自体

アクション・イベントの処理

アクション・イベントを処理するには、クラスのインターフェース内に Action マクロが挿入されていなければなりません。Action マクロの最初のパラメータは、アクションの名前、2 番目のパラメータは、そのアクションを処理するメソッドの名前です。

たとえば、次の例は Cut アクション・イベントを管理するテキスト・ビューです。

```
IlvDvBeginInterface(MyTextView)
    Action(Cut, myCut)
IlvDvEndInterface1(IlvDvTextView)

void
MyTextView::myCut()
{
  ...
}
```

文書またはビューは、アクション・イベントを処理するのと同じ方法で、アクションの状態を管理できます。つまり、マクロ RefreshAction([ActionName], [MethodName]) を使用します。

たとえば、次の例は Cut アクションの状態を管理するテキスト・ビューです。

```
IlvDvBeginInterface(MyTextView)
     RefreshAction(Cut, refreshCut)
IlvDvEndInterface1(IlvDvTextView)

void
MyTextView::refreshCut(IlvDvActionDescriptor* desc)
{
   desc->setValid(isRelevantSelection());
}
```

refreshCut このメソッドは、文書 (とその関連する文書ビュー) がアクティブになるたびに、呼び出されます。これだけでは十分ではない場合があるため、アプリケーション・メソッド refreshAction([Action Name]) の呼び出しによってアクションの状態を強制的にチェックできます。前のサンプルでは、たとえばテキスト・ビューでの選択が変わるたびに、refreshAction(IlvGetSymbol("Cut")) を呼び出せます。

索引

も

よ