IBM ILOG Views

Foundation 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

*このユーザ・マニュアル*では、*IBM ILOG Views Foundation リファレンス・マニュアル*に詳しい説明がある C++ API の使用方法について紹介します。

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。

マニュアル構成

このマニュアルには、IBM® ILOG® Views Foundation を組み込むアプリケーションの開発に関する、概念的で実践的な情報が掲載されています。IBM® ILOG® Views グラフィック・オブジェクトの根底となる基礎概念を説明するとともに、グラフィック・オブジェクトの作成方法と使用方法について説明します。

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 Foundation の概要*では、IBM ILOG Views Foundation の概要を説明します。
- 2 章 *グラフィック・オブジェクト*では、グラフィック・オブジェクトの概念と `IlvGraphic` クラスから派生する多数のクラスの使用について説明します。
- 3 章 *グラフィック・リソース*では、グラフィック・オブジェクトおよびテキスト表示を定義するリソースおよびパレット・クラスについて説明します。
- 4 章 *グラフィック形式*では、IBM ILOG Views で使用できるベクトルおよびビットマップ形式について説明します。
- 5 章 *イメージ処理フィルタ*では、2 つのイメージをフィルタを選択して組み合わせるといったように、ビットマップ・イメージをさまざまな方法で処理できる `IlvBitmapFilter` のサブクラスについて説明します。
- 6 章 *ディスプレイ・システム*では、`IlvDisplay` について説明します。これはディスプレイ・システム接続に関する IBM ILOG Views の基本クラスです。
- 8 章 *描画ポート*では `IlvPort` ベース・クラスについて説明します。
- 7 章 *ビュー*では、IBM ILOG Views で使用されているビュー、すなわち視覚表示領域の概念について説明します。
- 9 章 *コンテナ*では、アプリケーションでの効率的な表示とグラフィック・オブジェクトの振る舞いを提供するコンテナの使用法について説明します。
- 10 章 *動的モジュール*では、ダイナミック・ライブラリ、すなわち DLL の作成および読み込みについて説明します。
- 11 章 *イベント*では、イベント・ループを実装するクラスについて説明します。
- 12 章 *IlvNamedProperty: 永続性プロパティ・クラス*では、アプリケーションに依存するデータを IBM ILOG Views のオブジェクトに関連付ける方法について説明します。
- 13 章 *IBM ILOG Views における印刷*では、IBM ILOG Views Printing Framework を使用してプリンタ、ドキュメント、用紙書式、その他の印刷制御を定義する方法について説明します。
- 14 章 *IBM ILOG Script プログラミング*では、IBM ILOG Views の高度なスクリプト言語である IBM ILOG Script の使用法について説明します。
- 15 章 *国際化*では、IBM ILOG Views アプリケーションのローカライズ言語バージョンの作成について説明します。

付録では、次のような補足情報や参考情報を提供します。

- 付録 A、*IBM ILOG Views アプリケーションのパッケージ化*では、アプリケーションを IBM ILOG Views でパッケージ化する `ilv2data` ツールについて説明します。

- 付録 B、*IBM ILOG Views を Microsoft Windows で使用する* では、IBM ILOG Views を Microsoft Windows とインターフェース接続する際の条件およびヒントについて説明します。
- 付録 C、*IBM ILOG Views を X Window システムで使用する* では、IBM ILOG Views を X Window システムとインターフェース接続する際の条件およびヒントについて説明します。
- 付録 D、*移植性の制約* では、複数のプラットフォームに渡る移植性を制限する可能性のある IBM ILOG Views Foundation のシステムに依存する側面について説明します。
- 付録 E、*エラー・メッセージ*では、ILOG Views Foundation で生成されるエラー・メッセージを示し、その原因と回避方法について説明します。
- 付録 F、*IBM ILOG Script 2.0 言語リファレンス* は、IBM ILOG Script 構文の参考文献です。

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。

- 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。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;
```

例に関する注意

このマニュアルには、IBM ILOG Views を効果的に使用するための例と説明が記載されています。さらに、例の中には、IBM ILOG Views をインストールしたディレクトリのすぐ下の、samples ディレクトリにある IBM ILOG Views、と一緒に配布されたソース・コードから抽出されているものもあります。

参考文献

下記の文献には、C++ プログラミング言語に関する情報が記載されています。

- Lippman, Stanley B. *C++ Primer*, 3rd ed. Reading, MA: Addison-Wesley, 1998.
- Stroustrup, Bjarne. *The C++ Programming Language*, 3rd ed. Reading, MA: Addison-Wesley, 1997.
- Stroustrup, Bjarne. *The Design and Evolution of C++*. Reading, MA: Addison-Wesley, 1994.
- ISO/IEC 14882:1998 *Programming Languages - C++* and
  ISO/IEC 14882-1998 *Information Technology - Programming Languages - C++*

  The ISO/ANSI C++ Standard. American National Standards Institute (http://www.ansi.org) からオンラインで印刷版が入手できます。

以下の文献には、グラフィック関連の問題についてのアドバイスが記載されています。

- Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed. Reading, MA: Addison-Wesley, 1996.
- *Graphics Gems.*
  Vol. I: Glassner, Andrew S. (ed.), 1990. Reissue 1993.
  Vol. II: Arvo, James (ed.), 1991. Reissue 1994.
  Vol. III: Kirk, David (ed.), 1992, 1994.
  Vol. IV: Heckbert, Paul S. (ed.), 1994
  Vol. V: Paeth, Alan W. (ed.), 1995.
  Boston: Academic Press.
- Murray, James D. and William van Ryper. *Encyclopedia of Graphics File Formats*, 2nd ed. Sebastopol, CA: O'Reilly and Associates, 1996.

- Nye, Adrian.
  Vol. 1 *Xlib Programming Manual*, 3rd ed., 1992.
  Vol. 2 *Xlib Reference Manual*, 3rd ed., 1992
  O'Reilly & Associates.
- O'Rourke, Joseph. *Computational Geometry in C*, 2nd ed. Cambridge University Press, 1998.
- Rogers, David F. and J. Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill Publishing Co., 1990.
- Young, Douglas A. *The X Window System: Programming and Applications with Xt, OSF/Motif*, 2nd ed. Prentice Hall, 1994.

章

1

IBM ILOG Views Foundation の概要

IBM® ILOG® Views Foundation は、基本 IBM ILOG Views パッケージであり、UNIX および PC 環境で実行するアプリケーションのグラフィカル・ユーザ・インターフェース (GUI) および対話的な 2 次元グラフィックを作成する開発者向けの中核機能を提供します。

ここでは以下を説明します。

- *アプリケーション・プログラミング・インターフェース (API)* では、グラフィック・インターフェース設計のための C++ ライブラリ一式について説明します。
- *IBM ILOG Views の使用*では、ビューおよびグラフィック・オブジェクトの基本概念について説明します。

アプリケーション・プログラミング・インターフェース (API)

IBM® ILOG® Views は、インターフェースの設計に役立てる C++ クラス・ライブラリ一式と編集用補足ツールで構成されています。

ライブラリ

IBM ILOG Views ライブラリは、アプリケーションのプログラム可能な部分の実装に必要な API を提供しています。真にオブジェクト指向の C++ ライブラリとして、IBM ILOG Views は継承によるコードの再利用を重視しています。各派生クラスはそのベース・クラスを専門化し、継承された構造および振る舞いを追加するかまたはそれらを変更します。これはつまり、特定のクラスに求める機能がない場合、その機能が継承されているかどうかそのベース・クラスも確認する必要があるということです。独自のクラスを派生する際に、既存のクラス機能を使用して新しく必要なコードのみを作成することができます。これにより開発および保守費用を削減できます。

IBM ILOG Views API は C++ で作成されています。これは C のスーパーセットであり、必要な場合は C ルーチンを呼び出すことができます。C++ は柔軟性とリソース効率に優れるため、もっとも広く使用されているオブジェクト指向言語です。

C++ により提供されるオブジェクト指向機能により、コードの再利用が可能になるため、コーディング時間が節約できます。クラス階層により、C++ クラスのライブラリはプロシージャ指向のライブラリに比べ、柔軟性、拡張性、信頼性に優れています。

オブジェクト指向のプログラミングは、特にグラフィック指向のアプリケーションに適しています。これは、グラフィック・オブジェクトは類似した演算を実行することが多いためです。たとえば、ボタンは矩形が特化したものであるため、再コーディングすることなく矩形の特性すべてを継承できます。この階層本質により、プロシージャの開発や保守がより簡単になり、時間の節約ができます。

オブジェクト指向コードでは、IBM ILOG Views 実装の知識がなくても、独自のアプリケーション (またはライブラリ) 用の IBM ILOG Views オブジェクトを拡張または専門化できます。同様に、実装の詳細を知らない顧客が、オブジェクトを専門化することもできます。さらに、サブクラスを作成することにより、IBM ILOG Views ライブラリに加えてアプリケーション用に独自のライブラリを作成できます。

クラス階層

IBM ILOG Views クラス階層の構成では、必要なものが簡単に見つかります。たとえば、下記のダイアグラムに示されたクラスを使用し、最低限のコーディングで高度なインターフェースを簡単に作成できます。

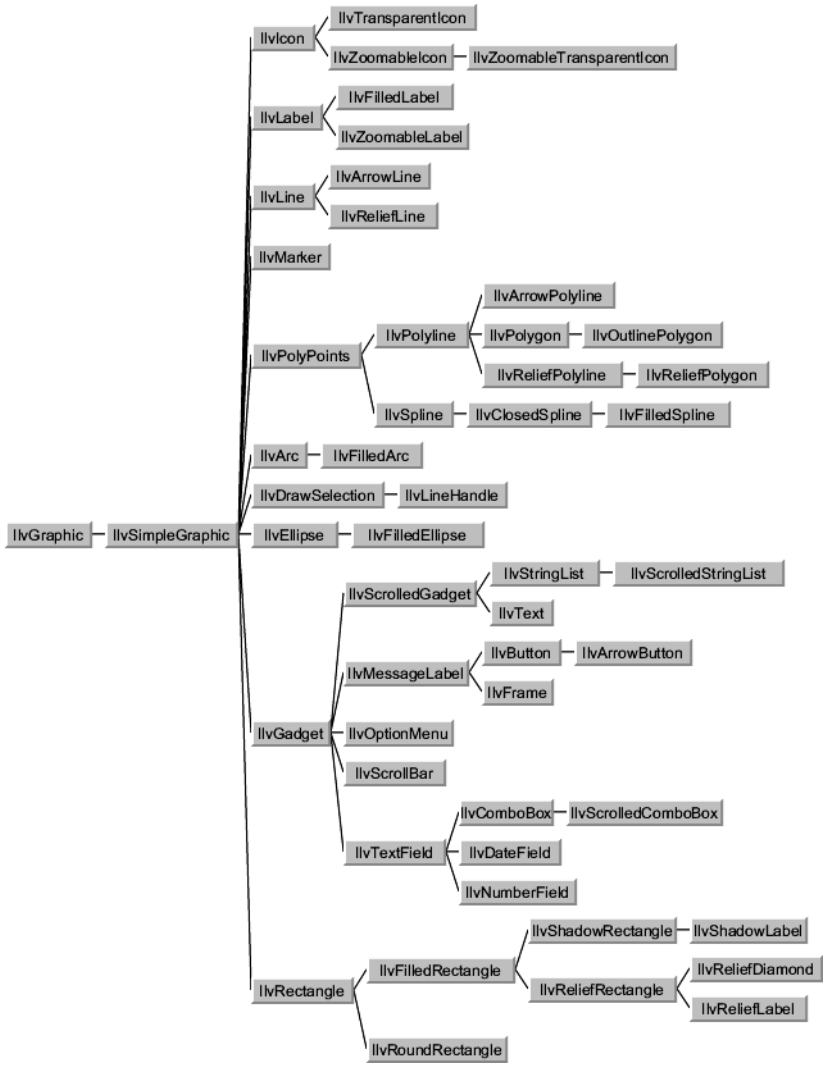


図 1.1 IBM ILOG Views *グラフィック・オブジェクトのクラス階層の一部*

IBM ILOG Views の使用

IBM® ILOG® Views Foundation は、UNIX および PC 環境で動作するアプリケーションのグラフィカル・ユーザ・インターフェース (GUI) および対話的な 2 次元グラフィックの作成に使用します。

ここでは、IBM ILOG Views に関する基本用語および概念について説明します。最後の章では、API でのクラス実装について説明します。

- *ウィンドウおよびビュー*では、ビュー、ウィンドウ、および関連用語を定義します。
- *コンテナ：ビューの制御*では、IBM ILOG Views 使用におけるコンテナの役割について説明します。また、コンテナとマネージャの違いについても扱います。
- *グラフィック・オブジェクトの概要*では、IBM ILOG Views でのグラフィック・オブジェクトの表示、変換について説明します。
- *アトリビュートおよびパレットの描画*では、グラフィック・オブジェクトの外観を変更するための豊富な色、フォント、その他の IBM ILOG Views リソースについて説明します。
- *基本描画タイプ*では、描画アトリビュートを線、領域、文字列と関連付けます。これらは、IBM ILOG Views における描画の基本タイプです。

ウィンドウおよびビュー

IBM® ILOG® Views では、ビューは基本サービスを追加できるオブジェクトです。オブジェクトは、UNIX での X Window™ などの基礎となるディスプレイ・システムのウィンドウと関連付けられます。描画は、オブジェクトまたはオブジェクトのサブセットのイメージを表すビュー内で頻繁に行われます。このイメージは、オブジェクト自体に影響を与えることなく、移動、拡大、または回転によって幾何学的に変換できます。

ビューの定義

IBM® ILOG® Views プログラムを作成する際にはまず、プログラムの表示と対話を行う *ビュー*を作成し、組み合わせます。

ビューは、視覚的なプレース・ホルダ、つまり画面上の矩形領域であり、IBM ILOG Views アプリケーション要素を表示します。

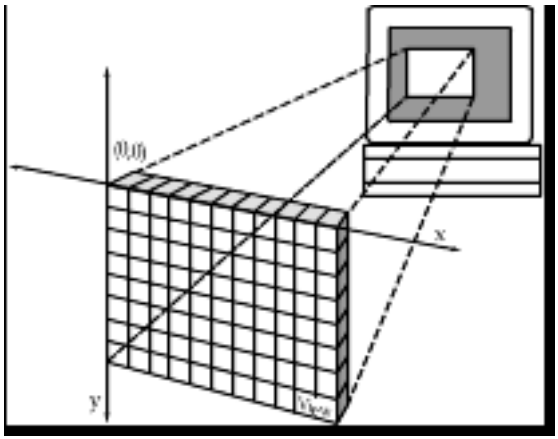


図1.2 *ビュー*

各ビューは、下記によって識別されます。

- **位置** (定義した x - y 座標)、
- **サイズ** (定義した高さと幅)、
- **可視性** (ビューは存在しても表示されない場合があります)。

ビューとその内容を組み合わせることにより、IBM ILOG Views アプリケーションの表示要素を作成します。

ビュー・ウィンドウの概要

次は、IBM® ILOG® Views ウィンドウの簡単な図です。

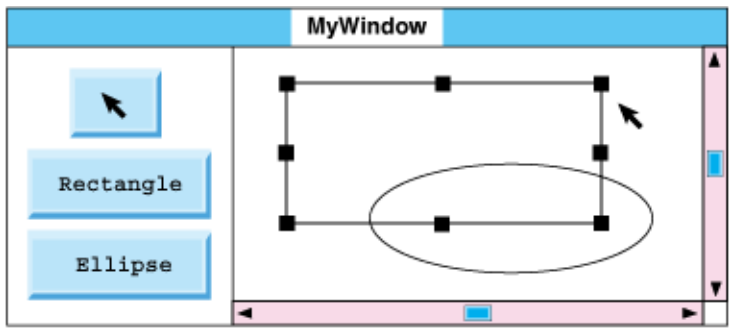


図1.3 IBM ILOG Views *ウィンドウ*

このウィンドウにあるボタンで、矩形や楕円を描くことができ、矢印ボタンで既存のオブジェクトを選択して移動やリサイズができます。スクロール・バーは、広い作業ビューの一部を表示領域に移動します。

このウィンドウは、以下の 4 つの異なった IBM ILOG Views ビューで構成されています。

- *上位レベル・ビュー：トップ・ウィンドウ*
- *スクロール・ビュー*
- *ツール・ビュー*
- *作業ビュー*

上位レベル・ビュー：トップ・ウィンドウ

トップ・ウィンドウの目的は、一般的にさまざまな種類の下位レベルのビューを表示することです。トップ・ウィンドウに直接描画することはほとんどなく、多くの場合はそこに表示される下位レベルのビューに描くことになります。

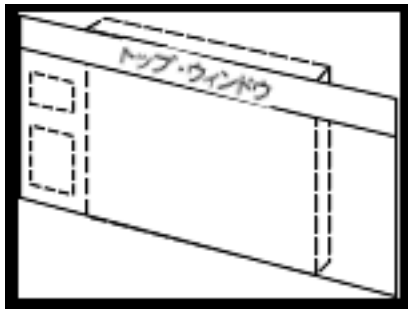


図 1.4 *トップ・ウィンドウ*

この種類のビューは、下記の項目に当てはまる唯一のビューです。

- タイトル・バーを含む。
- システム・メニューと関連付けることができ、プログラムのユーザがウィンドウのリサイズやアイコン化などに介入できる。

トップ・ウィンドウには、下位レベルのビューを必要な数だけ関連付けることができます。トップ・ウィンドウにはウィンドウのタイトルだけではなく、現在のサイズも表示されます。トップ・ウィンドウ内に表示されるビューはどれも、トップ・ウィンドウの矩形内でしか表示できません。すなわち、トップ・ウィンドウは下位レベルのビューでは表示できません。

スクロール・ビュー

スクロール・ビューは下位レベルのビューです。スクロール・ビューの唯一の目的は、スクロール・バーのペアを表示することであり、これによりスクロール・ビュー内に表示されている下位レベルの描画ビューをスクロールできます。

図1.5 *スクロール・ビュー*

> ***メモ:*** *このタイプのウィンドウは、*IBM ILOG Views *の Gadgets パッケージで提供されています。Foundation パッケージでは、Microsoft Windows および Motif ポートのみを対象に、ネイティブ実装されています。*

ツール・ビュー

ツール・ビューは、下位レベルのビューで、描画および選択コマンド・ボタンがあります。この種類のビューは、グラフィック・オブジェクトを格納および表示できるだけでなく、ユーザがオブジェクト上で実行するアクションを調整できます。

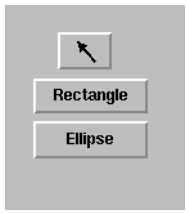


図1.5 *ツール・ビュー*

作業ビュー

作業ビューは最下位のビューです。作業ビューに表示されるのは、全体の一部です。次の図では、大きな灰色の矩形中の白色の矩形が表示されている部分です。たとえば、ビューの右上に楕円がありますが、この図では表示されていません。これを表示するには、ユーザは上位レベルのスクロール・ビューにあるスクロール・バーを使用します。

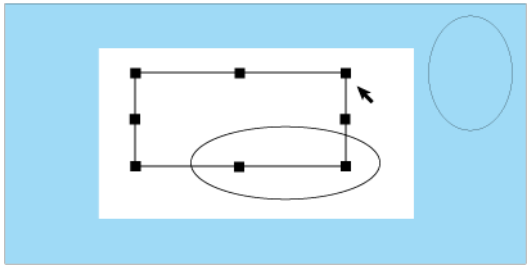


図1.6　作業ビュー

作業領域の変更は、*クリッピング*で制御できます。クリッピング領域がアクティブになっている間は、その領域の変更のみが表示されます。

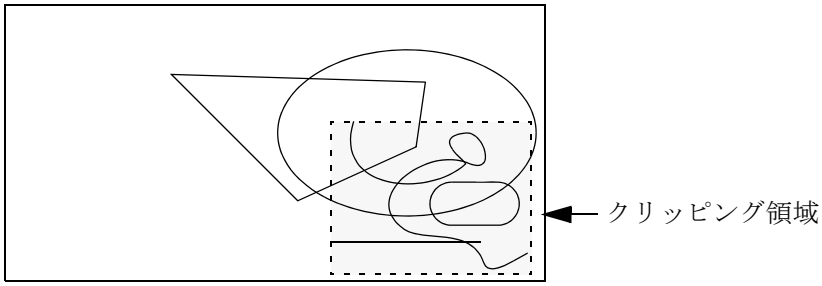


図1.7　クリッピング領域

コンテナ：ビューの制御

コンテナは、グラフィック・オブジェクトの格納と表示を調整します。

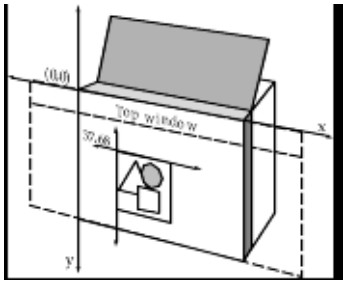


図1.8　コンテナ

基本的に、コンテナはビューであり、格納するグラフィック・オブジェクトの自動的更新とそのビューで発生するシステム・イベントおよびユーザ・イベントの処理を行う定義済みコールバックを備えています。

コンテナに格納されている各グラフィック・オブジェクトは固有のものであり、そのコンテナによってのみ表示されます。要約すると、コンテナは次の特徴を備えています。

- 基本的にビューの一種であり、どんな数のグラフィック・オブジェクトでも収集できます。
- ビュー内のすべての描画操作を自動的に管理します。
- インタラクタをそのオブジェクトへ関連付けて、特定の振る舞いを持たせます。
- オブジェクトへのアクセスをオブジェクト名で行います。
- オブジェクトの描画時に、トランスフォーマを使用して移動、ズーム、回転できます。
- 単一のアクションをビューが受け取ったイベントに関連付けます。

コンテナとマネージャ

IBM ILOG Views は、次の 2 つの基本タイプの記憶域データ構造のいずれかにオブジェクトをグループ化します。

- コンテナ
- マネージャ

ビューは、コンテナまたはマネージャに格納されているグラフィック・オブジェクト一式に関連付けられています。

コンテナは特定数のグラフィック・オブジェクトを格納しており、コンテナ内に格納されているオブジェクトを表示するビューと関連付けられています。各オブジェクトは特定の振る舞いに関連付けることができます。また、定義済み関数をすぐ呼び出すキーボード・イベントであるアクセラレータをコンテナ自体に付加できます。コンテナは Foundation パッケージの機能の一部です。

マネージャは別タイプのデータ構造であり、レイヤ、マルチビュー、高速再描画、永続性、および編集の機能が備わっています。マネージャの詳細については、Manager マニュアルを参照してください。

> ***メモ：*** *多数のオブジェクト、マルチビュー、レイヤを効率的に描画する場合は、コンテナではなくマネージャを使用してください。*

グラフィック・オブジェクトの概要

IBM® ILOG® Views は、2 次元ベクトル・グラフィック・エンジンを使用して、描画ポート (メモリ、画面、ダンプ・ファイル) および基本的な幾何学形状を作成する描画プリミティブの大規模セットを提供します。弧、曲線、矩形、ラベルなどの基本的な幾何学形状を描画できます。メモリ内の画面上での描画、PostScript などのダンプ・ファイルの生成が可能です。白黒およびカラーのイメージを作成できます。グラフィック・エンジンは、これらのプリミティブ上に構築され、グラフィック・オブジェクトを定義します。

グラフィック・オブジェクトの表示

グラフィック・オブジェクトとは、ユーザが画面上で見ることができるイメージです。グラフィック・オブジェクトを表示する際に、その座標を特定のコンテナの座標系と関連付けます。

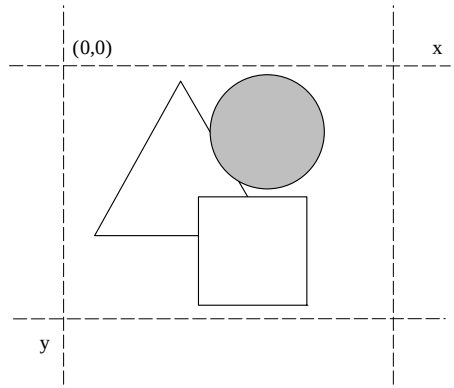


図 1.9 *グラフィック・オブジェクトの幾何学プロパティ*

幾何学プロパティは、グラフィック・オブジェクトの形状と位置を定義します。すべてのグラフィック・オブジェクトには x 軸値、y 軸値、および寸法 (幅と高さ) があります。x 軸値と y 軸値は、グラフィック・オブジェクトのバウンディング・ボックスの左上隅を示します。これは、オブジェクトがカバーする領域全体を含む最小の矩形です。

IBM ILOG Views ベースのプログラム内のグラフィック・オブジェクトの正確な形状を定義し、さまざまな描画メンバ関数で具体化します。他のメンバ関数がグラフィック・オブジェクトに関する情報を提供し、使用中の形状の幾何学テストを実行できます。たとえば、任意の座標にある点が特定の形状内にあるかどうかをチェックできます。

インタラクタ

IBM ILOG Views ではグラフィック・オブジェクトと振る舞いが明確に区別されているため、特定の振る舞いをオブジェクトに適用することができます。

IBM ILOG Views では、定義済みの振る舞いを「インタラクタ」と呼びます。インタラクタは、すべてのグラフィック・オブジェクトに適用して特定の振る舞いを持たせ、グラフィック・オブジェクトの機能を定義することができます。

たとえば、オブジェクトに「ボタン」インタラクタを適用することにより、視覚的な側面しかないオブジェクトをクリックすると直ちにボタンの振る舞いを持たせることができます。

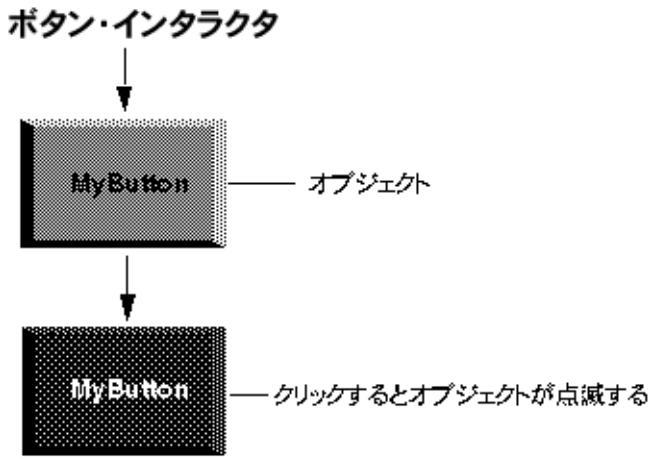


図 1.10 *オブジェクト / インタラクタの概念*

オブジェクトと振る舞いを分離するメリットは、特定のタイプの振る舞いをどんなタイプのグラフィック・オブジェクトにも適用できるという点です。たとえば、わずか 1 行のコードでボタンの振る舞いをビットマップ (アイコンなど) に適用できます。

さらに、IBM ILOG Views で提供されているインタラクタ・クラスをサブクラス化することにより、簡単に振る舞いを拡張できます。

アトリビュートおよびパレットの描画

IBM® ILOG® Views では、線のパターン、色、フォントのアトリビュートを自由に選択でき、グラフィック・オブジェクトやテキストに適用できます。

これらのリソースはパレットにグループ化されており、複数のオブジェクト間で共有されるため、最低限のメモリで全体を簡単に変更できます。

色

矩形などの簡単な描画では、実際の描画は*前景*色と呼ばれるもので、描画の「後ろ」は*背景*色と呼ばれるもので行われます。

- **前景色**　前景色は、点、円弧、線、ポリラインなどの描画に使用されます。また文字列の表示や、多角形、円弧などの領域を塗りつぶすのにも使われます。
- **背景色**　背景色は、パターンで塗りつぶすときや文字列の描画に、第 2 の色として使用されます。

線の種類と太さ

「実線」と呼ばれる通常の線のほかに、点線や破線で直線や曲線を描くことができます。これを線の種類といいます。線の太さは描画での線の幅を指します。

線の種類と幅は、すべての線描画およびポリラインやスプラインを含む線型描画の正確な視覚的側面を定義します。

パターン

パターンとは、表面を塗りつぶすのに使用するデザインです。IBM ILOG Views では、2 タイプのパターンがあり、適用できる色数で区別されます。

モノクロ・パターン

「パターン」という言葉は、モノクロ (または 2 色) デザインを示します。IBM ILOG Views ではすぐに使えるパターンが 16 種類あります。

以下はパターンの例です。

図 1.11　*パターン*

この特定のパターンは、図のように 16x16 ビット列で構成される*マスク*を使用して作成できます。

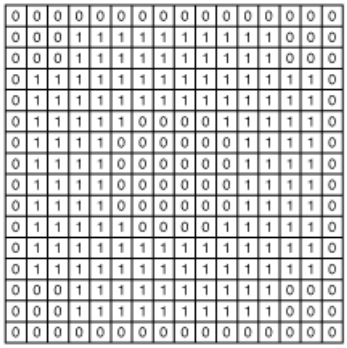


図 1.12　パターン・マスク

パターン・マスクの中では、1 ビットを*前景*とし、0 ビットを*後景*としています。言い換えれば、パターン・リソースが色リソースを呼び出すということです。

色のパターン

通常のパターンが 2 次元であるのに対し、色のパターンには 3 つ目の次元である*深さ*が組み入れられており、色の設定を行います。

配列の各位置に `1` や `0` を配置するだけでなく、色のパターンに使用する色を示す数を挿入します。色のパターンのデフォルト値は、パターンが何も使用されないことを示す `0` です。

フォント

フォントは文字列で、つまりテキストを描く際に使用します。

基本描画タイプ

基本的に、IBM® ILOG® Views には 3 種類の描画があります。これは線、領域、文字列であり、アトリビュートは各描画の要件と機能により適用されます。

線

このカテゴリには直線、曲線、直線または曲線を連結した延長可能な線一式が含まれます。アトリビュートは、次のように線に適用されます。

- **色**　直線または曲線は現在の前景色で描かれます。
- **線の種類**　現在の線の種類 (実線、点線、破線など) は、直線または曲線の描画方法を決定します。
- **線の太さ**　符号のない整数は現在の線の太さを示します。

- **パターン**　線はモノクロ (2 色) を使用して現在のパターンで描かれますが、これは太線以外では目立ちません。
- **色のパターン**　線は色を使用して現在の色のパターンで描かれます。ここでもまた、この効果は太線以外では目立ちません。

領域

この用語は、直線や曲線を連結して閉じたものを示します。アトリビュートは、次のように領域に適用されます。

- **色**　領域の周囲の閉じられた曲線は元の色、つまり現在の前景色を保持します。
- **パターン**　領域は現在の塗りつぶりパターンまたは塗りつぶしマスク・パターンで塗りつぶされます。
- **色のパターン**　領域は現在の色のパターンで塗りつぶされます。
- **塗りつぶしスタイル**　この値は、パターンがマスク、モノクロ・パターン、または色のパターンとして処理されるどうかを決定します。
- **塗りつぶしルール**　自己交差している多角形の塗りつぶし方法を決定します。詳細については、*85 ページの塗りつぶしルール* を参照してください。
- **円弧モード**　円弧を閉じて塗りつぶす方法を決定します。詳細については、*85 ページの円弧モード* を参照してください。

文字列

アトリビュートは、次のように文字列に適用されます。

- **色**　実際の文字は前景色で表示されます。
- **フォント**　文字列を表示するフォントです。

章

2

グラフィック・オブジェクト

IBM® ILOG® Views は、上位レベルのさまざまなグラフィック・オブジェクトを作成するクラス階層を提供します。これらのオブジェクトはクラス `IlvGraphic` や `IlvSimpleGraphic` が起点となります。

- *IlvGraphic: グラフィック・オブジェクト・クラス* は IBM ILOG Views グラフィック・オブジェクトの基本クラスです。
- *IlvSimpleGraphic クラス* は `IlvGraphic` から継承する基本クラスです。グラフィック・リソースを割り当て、グラフィック・オブジェクトに変換を適用できます。
- *定義済みグラフィック・オブジェクト* では、円弧や四角形などの標準幾何学形状を作成するために、IBM ILOG Views で提供される多数のグラフィック・オブジェクトを説明します。
- *複合グラフィック・オブジェクト* を使うと、グループ化によりさまざまな目的でオブジェクトを使用できるように最適化できます。
- *その他のベース・クラス* は、主に他の IBM ILOG Views パッケージで使用されている追加グラフィック・クラスについて説明します。
- *新規グラフィック・オブジェクト・クラスの作成* では、IBM ILOG Views で新しいカスタマイズ・グラフィック・オブジェクトを作成する方法について説明します。

IlvGraphic: グラフィック・オブジェクト・クラス

IBM® ILOG® Views グラフィック・オブジェクトは、抽象ベース・クラス IlvGraphic からアトリビュートを継承しています。このクラスを使うと、IBM ILOG Views グラフィック・オブジェクトが、特定の宛先ポートでそれ自体を描画でき、また必要に応じて IlvTransformer クラスに関連付けられたオブジェクトに従って座標を変換します。

IlvGraphic は幾何公差を設定、変更できるメンバ関数を備えています。アプリケーション固有の目的のため、オブジェクトと関連付けできるユーザ・プロパティを設定・取得するために多数のメンバ関数が用意されています。IlvGraphic クラスは、実際にはこれらのメンバ関数を実装していません。これらは、仮想メンバ関数として宣言し、IlvGraphic アトリビュートを継承するクラスでさまざまな動作を行うように定義します。幾何学形状やグラフィック・アトリビュートを操作するメンバ関数もありますが、そのような関数は何も行いません。

メンバ関数

IlvGraphic メンバ関数は、いくつかのグループで表されます。

- **幾何学プロパティ**　これらのメンバ関数は位置、サイズ、グラフィック・オブジェクトを描く IlvGraphic::draw メソッドを含む描画プロパティを扱います。仮想 IlvGraphic::draw メソッドは、メソッド IlvGraphic::boundingBox と定義する必要があります。これは、グラフィック・オブジェクトが覆う領域全体を含む最小矩形を定義します。
- **グラフィック・プロパティ**　これらのメンバ関数を使用して、オブジェクトの視覚的側面、つまり色やパターンを変更します。これにはグラフィック・オブジェクトのグラフィック・プロパティを示し、パレットを変更してグラフィック・オブジェクトに結び付けるメンバ関数を使用します。以下の例は、グラフィック・オブジェクトの背景の設定方法を示しています。

```
IlvButton* mybutton = new IlvButton(display,
                                    IlvPoint(20,20),
                                    "Quit");
IlvColor* color = display->getColor("gold");
if (color) mybutton->setBackground(color);
```

- **名前付きのプロパティ**　名前付きのプロパティは、グラフィック・オブジェクトに関連付けられたプロパティの永続性を扱います (12 章を参照)。
- **ユーザ・プロパティ**　IlvGraphic オブジェクトは、ソース・コードのユーザ・プロパティ一式と関連付けることができます。ユーザ・プロパティとは *キー値* ペアのセットで、キーが refcppfoundation:IlSymbol オブジェクトでその値は任意の種類の情報値です。ユーザ・プロパティに永続性はありません。

これらのメンバ関数で、グラフィック・オブジェクトをアプリケーションから取得した情報と簡単に関連付けられます。作成するオブジェクトへのポインタを格納することによってアプリケーションのグラフィック部分を追跡し、ユーザ・プロパティによりアプリケーションのグラフィックな側面に接続できます。次にその例を示します。

```
IlInt index = 10;
IlSymbol* key = IlGetSymbol("objectIndex");
mybutton->addProperty(key, (IlAny)index);
```

メンバ関数の中にはタグ管理ができるものがあります。タグはグラフィック・オブジェクトに適用してそれらを識別するマーカーです。さまざまなIBM ILOG Views 機能を使用して、タグ付きのオブジェクトだけを操作できます。

- **ガジェット・プロパティ**　ガジェット・プロパティはイベントに対するオブジェクトのセンシティビティ、オブジェクトがアクティブになった際のコールバック呼び出し、オブジェクトに格納されているクライアントのデータ、オブジェクト・クラスに関連付けられたオブジェクト・インタラクタを扱います。コールバックの使用方法については、*コールバック*を参照してください。
- **フォーカス・チェーン・プロパティ**　*フォーカス*は、キーボード・イベントを受け取る画面上のオブジェクトです。*フォーカス・チェーン*とは、フォーカスを受け取る画面上のオブジェクトの順番です。フォーカス・チェーンでは、一般的に フォーカスは Tab キーを押すと次のオブジェクトに、Shift と Tab キーを押すと前のオブジェクトに移動します。
- **クラス情報**　クラス `IlvGraphic` のサブタイプは、クラス・レベルの情報を扱います。これはつまり、特定のクラスのインスタンスはすべて同じ情報を共有できるということです。たとえば、ターゲットの `IlvGraphic` オブジェクトが

指定されたクラス引数のサブクラスである場合、IlvGraphic::className を使うとクラス名を取得でき、IlvGraphic::isSubtypeOf は IlTrue を返します。この例では、クラス名情報メンバ関数の使用方法を示しています。

```
IlvButton* button = new IlvButton(display,
                                  IlvPoint(10,10),
                                  "sample");
// Get the IlvClassInfo object associated with the button class.
IlvClassInfo* classInfo = button->getClassInfo();

// Get the name of the IlvGraphic class and print it: "IlvButton"
const char* name = classInfo->getClassName();
IlvPrint(name);

// Get the name of the super class and print it: "IlvMessageLabel"
name = classInfo->getSuperClass()->getClassName();
IlvPrint(name);

IlBoolean isSubtype =
   classInfo->isSubtypeOf(IlvSimpleGraphic::ClassInfo());
name = isSubtype ? "It's a subtype" : "error";
IlvPrint(name);
```

- **クラス・プロパティ**　スタティック・メンバ関数および相当する非スタティック関数を使うと、クラス・レベルのプロパティを扱うことができます。つまり、これらのプロパティは、クラスのインスタンスごとに定義されます。いくつかのメソッドでは、IlBoolean パラメータを使うと、一致するまでオブジェクトの各スーパークラスで反復動作できます。以下に示すのは、クラス・プロパティを扱うメンバ関数です。

```
void AddProperty(const IlSymbol* key, IlAny value);
void RemoveProperty(const IlSymbol* key);
void ReplaceProperty(const IlSymbol* key, IlAny value);
void GetProperty(const IlSymbol* key,
                 IlBoolean checkSuperClass = IlFalse);
const IlvClassInfo* HasProperty(const IlSymbol* key,
                                IlBoolean checkSuperClass = IlFalse);
void addClassProperty(const IlSymbol* key, IlAny value);
IlBoolean removeClassProperty(const IlSymbol* key);
IlBoolean replaceClassProperty(const IlSymbol* key,
                               IlAny value);
IlAny getClassProperty(const IlSymbol* key,
                       IlBoolean checkSupCl = IlFalse) const;
const IlvClassInfo* hasClassProperty(const IlSymbol* key,
                                     IlBoolean checkSupCl = IlFalse) const;
```

次に、クラス・プロパティの使用方法の例を示します。

グラフィック・インスタンスがトグルに類似したセンシティブな振る舞い (IBM ILOG Views は *インタラクタ*と呼ばれる特定のオブジェクトを提供し、これにより振る舞いをグラフィック・オブジェクトに関連付けできます) と表示されるマップを想定します。これらの要素を非センシティブにする場合があります。オブジェクトのリストをスキャンしてセンシティビティを IlFalse に設定する代わりに、クラス・レベルのプロパティを以下のように使用します。

myClass を IlvGraphic のサブクラスとし、IlvToggleInteractor のサブクラスである myInteractor を myClass に付加するインタラクタとします。

```
// Add the class-level property
myClass* obj = new myClass(display);
obj->addClassProperty(IlGetSymbol("sensitive"),
                      (IlAny)IlTrue);
```

適用されたコードでは、センシティビティが反転したかどうかをアプリケーションがテストします。ステートメントはどちらも同等なため、スタティック・メンバ関数を使用してプロパティをクラス・オブジェクトに追加する方法もあります。

```
if (anyValue == IlTrue)
{
   myClass::AddProperty(IlGetSymbol("sensitive"),
                        (IlAny)IlFalse);
}
```

myInteractor クラスの実装ファイルで、親クラスのメソッド IlvInteractor::handleEvent を特定の振る舞いに追加する、つまりある条件下においてセンシティビティをフリーズするように再定義します。

```
IlBoolean
myInteractor::handleEvent(IlvGraphic* object,
                          IlvEvent& event,
                          IlvContainer* cont,
                          IlvTransformer* transf)
{
   // gets the sensitivity state
   IlSymbol* symbol = IlGetSymbol(?sensitive?);
   if (object->hasClassProperty(symbol))
   {
      if (!object->getClassProperty(symbol))
         return IlFalse;
   }
   return IlvViewToggleInteractor::handleEvent(object,
                                               event,
                                               cont,
                                               transf);
}
```

- **入出力プロパティ**　これらのメンバ関数を使うと、IlvInputFile や IlvOutputFile として知られている特殊な種類のストリームでオブジェクト詳細の読み書きができます。これらのストリームは C++ ストリームでオブジェクトの読み書きを扱います。

  IBM ILOG Views は、これらのクラスを基本実装しています。これらは特定の情報を簡単に追加できるように設計されています。したがって、アプリケーションに依存するデータを保存したり読み込む必要がある場合は、これら 2 つのクラスのサブタイプを独自に作成します。

- **グラフィック・オブジェクトの書き込み**

  IlvOutputFile クラスは、オブジェクト一式の完全な詳細を出力ストリームに書き込みます。このクラスは、次のように使用できます。

```
// Open a file output stream
fstream outstream(?image.ilv?, ios::out | ios::trunc);

// Initialize the number of objects and their array of pointers
const IlvUInt n = 10;
IlvGraphic* outObjects[n];
for (IlvUInt i=0; i<n; i++)
   outObjects[i] = new IlvRectangle(display,
                                    IlvRect(0, 0, 200, 100));

// Create the IlvOutputFile
IlvOutputFile outfile(outstream);

// Write the objects and get in outTotalCount the number
// of objects actually stored
IlvUInt outTotalCount = 0;
outfile.saveObjects(n, outObjects, outTotalCount);
```

- **グラフィック・オブジェクトの読み込み**

  IlvInputFile クラスは、ストリームからオブジェクトを読み込むためのメイン・クラスです。次のコードは、入力ストリームから IlvGraphic オブジェクトを読み込む方法を示しています。

```
// Open a file input stream
fstream instream("image.ilv", ios::in);

// Create the IlvInputFile
IlvInputFile infile(instream);

// Get the number of created objects and their array of pointers
IlUInt InTotalCount = 0;
IlvGraphic* const* inObjects = infile.readObjects(display,
                                                  InTotalCount);
```

コールバック

オブジェクトに動作を実行するように指定した場合、自分で定義した特定の関数である*コールバック*を呼び出す必要が起こる場合もあります。これらの関数は、通常オブジェクトの handleEvent メソッドにより呼び出されます。

アクションがトリガされたときに特定のコールバックを呼び出すように設定するには、次の 2 つの方法があります。

◆ ユーザ定義の関数へのポインタとしてコールバックを登録します。

この関数は、IlvGraphicCallback タイプである必要があります。

IlvGraphicCallback タイプは、<ilviews/graphic.h > ファイルで定義されます。

```
#include <ilviews/graphic.h>
typedef void (* IlvGraphicCallback)(IlvGraphic* obj, IlAny arg);
```

最初の引数 (obj) はコールバックを呼び出したグラフィック・オブジェクトであり、2 つ目の引数 (arg) はユーザ・データです。ユーザ・データは、コールバックを特定のガジェットに設定する際に定義できます。データが何も定義されないと、パラメータは IlvGraphic::setClientData で設定できるグラフィック・オブジェクトのクライアント・データになります。

- グラフィック・オブジェクト・コンテナにより呼び出される関数に関連付けられているコールバック名を登録します。コールバック関数とその名前の関連付けは、特定のコンテナに固有でなければなりません。

コールバックの登録

コンテナでコールバックを登録するのに使用するメソッドは、次の通りです。

```
#include <ilviews/contain.h>

void   registerCallback(const char* callbackName,
                        IlvGraphicCallback callback);
void   unregisterCallback(const char* callbackName);
IlvGraphicCallback getCallback(const IlSymbol* callbackName) const;
```

コールバック・タイプ

オブジェクトは複数の *コールバック・タイプ* を定義できます。各コールバック・タイプは特定のアクションに対応します。たとえば、イベント・ガジェットには、ガジェットがキーボード・フォーカスを受け取ったときに呼び出されるコールバックである「Focus In (フォーカス・イン) 」というコールバック・タイプがあります。

各コールバック・タイプは、関連するイベントが発生したときに呼び出されるコールバックのリストを格納しています。IlvGraphic クラスには、特定のコールバック・タイプでコールバックを追加または削除できる汎用メソッドがあります。

```
#include <ilviews/graphic.h>

void addCallback(const IlSymbol*   callbackType,
                 IlvGraphicCallback callback);
void addCallback(const IlSymbol* callbackType,
                 const IlSymbol* callbackName);
void addCallback(const IlSymbol* callbackType,
                 IlvGraphicCallback callback,
                 IlAny data);
void addCallback(const IlSymbol* callbackType,
                 const IlSymbol* callbackName,
                 IlAny data);
void removeCallback(const IlSymbol*   callbackType,
                    IlvGraphicCallback callback);
void removeCallback(const IlSymbol* callbackType,
                    const IlSymbol* callbackName);
```

コールバックを追加するときに渡すことができる引数 data は、コールバックに送られます。これは、IlvGraphicCallback 定義の arg という引数に対応しています。

メイン・コールバック

メイン・コールバック・タイプは、オブジェクトのメイン・アクションを実行するのに使用できます。メイン・アクションは、複数のアクションを実行するようになっています。たとえば、ボタン・オブジェクトをアクティブにするときや文字列リストのアイテムをダブルクリックするときに使用できます。

次のメソッドでは、オブジェクトのメイン・コールバックを容易に設定できます。

```
#include <ilviews/graphic.h>

IlvGraphicCallback getCallback() const;
IlSymbol* getCallbackName() const;
void   setCallback(IlvGraphicCallback callback);
void   addCallback(IlvGraphicCallback callback);
void   setCallback(IlvGraphicCallback callback, IlAny data);
void   addCallback(IlvGraphicCallback callback, IlAny data);
void   setCallback(const IlSymbol* callbackName);
void   setCallback(const IlSymbol* callbackName, IlAny data);
void   setCallbackName(const IlSymbol* callbackName);
```

IlvSimpleGraphic クラス

IlvSimpleGraphic は IlvGraphic から継承された基本クラスです。IlvSimpleGraphic は IlvGraphic クラスのすべての機能を実装し、そのオブジェクトを描画するために使用される IlvPalette リソースを各インスタンスに追加します。このクラスは、グラフィック・オブジェクトに関連付けられた IlvPalette のインスタンスに集められた色、フォント、パターンなどのグラフィック・プロパティにアクセスし、変更する操作を実行します。オブジェクトのジオメトリ変換も適用できます。

IlvSimpleGraphic オブジェクトには独自の IlvPalette オブジェクトが含まれます。これはつまり、グラフィック・オブジェクトは同時に幾何学形状でもあり、この形状を表示するアトリビュート一式でもあるということです。したがって、このクラスからアプリケーションに必要な新しいオブジェクトを作成できます。必要なメンバ関数もあり、不必要なものもあります。IBM® ILOG® Views オブジェクト・ライブラリにはそのような多数のオブジェクトや、ほとんどすべての問題に対する幅広いソリューションが含まれています。

メンバ関数

IlvSimpleGraphic クラスには、パレットのアトリビュートにアクセスできるメンバ関数が含まれています。すべての IlvSimpleGraphic オブジェクトには

IlvPalette オブジェクトがあり、これはオブジェクトの間で共有できます。したがって、IlvSimpleGraphic オブジェクトに前景などのグラフィック・プロパティの変更を指示すると、次の操作が実行されます。

1. IlvDisplay::getPalette 関数を使用して新しい前景の新しい IlvPalette を検索します。
2. メンバ関数 IlvResource::lock が新しいパレットで呼び出されて参照カウントを増やします。
3. グラフィック・オブジェクトの旧パレットが呼び出されます。
4. メンバ関数 IlvResource::unlock が、旧パレットで呼び出されます。
5. 新しいパレットが、オブジェクトの現在のパレットとして登録されます。

これらの操作は、IlvPalette の共有を保証します。IlvPalette オブジェクトについては、この機構を使用することを推奨します。これは、IlvGraphic::setForeground などのグラフィック・アトリビュートを変更してリソースを操作できるメンバ関数が、仮想 関数として定義されているためです。

グラフィック・アトリビュート

IlvSimpleGraphic コンストラクタには、そこからリソースを取得する IlvPalette オブジェクトが必要です。palette パラメータは、特定の値に設定するか、または指定しないでおきます。この場合、値は、0 になります。パレットが 0 に設定されると、表示オブジェクトのデフォルトのパレットが使用されます。このパレットは、メンバ関数 IlvDisplay::defaultPalette によって返されるパレットです。palette パラメータは、オブジェクトの作成時にロックされ、削除時にロック解除されます。

定義済みグラフィック・オブジェクト

このセクションでは、定義済みグラフィック・オブジェクトを提供する基本クラス、IlvSimpleGraphic のサブクラスすべてについて説明します。

IlvArc

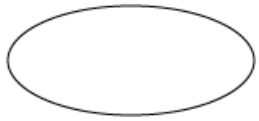
IlvArc オブジェクトは、楕円の輪郭付きの円弧として表示されます。

IlvFilledArc

IlvFilledArc オブジェクトは塗りつぶし円弧として表示されます。

IlvEllipse

IlvEllipse オブジェクトは、輪郭のある楕円として表示されます。



IlvFilledEllipse

IlvFilledEllipse オブジェクトは塗りつぶし楕円として表示されます。

IlvIcon

IlvIcon オブジェクトは、イメージとして表示されます。

IlvZoomableIcon

IlvZoomableIcon オブジェクトは、IlvIcon オブジェクトの一種で、ズームまたは形状変更ができます。

IlvTransparentIcon

IlvTransparentIcon オブジェクトは、透明な領域を持つイメージとして表示されます。

IlvZoomableTransparentIcon

IlvZoomableTransparentIcon オブジェクトは、IlvZoomableIcon オブジェクトの一種で、イメージの背景 (0 ビット) をそのままにします。

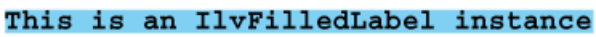
IlvLabel

IlvLabel オブジェクトは 1 行のテキストとして表示されます。ズームや形状変更はできません。

```
This object is an IlvLabel instance
```

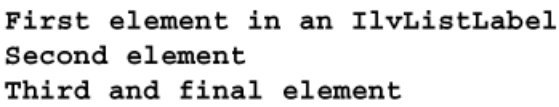
IlvFilledLabel

IlvFilledLabel オブジェクトは、塗りつぶし四角形の上に描画された 1 行のテキストとして表示されます。矩形の大きさはテキストのバウンディング・ボックスに正確に一致します。



IlvListLabel

IlvListLabel オブジェクトは文字列の縦リストとして表示され、IlvLabels が並んだように見えます。



IlvZoomableLabel

IlvZoomableLabel オブジェクトは、標準の IlvLabel オブジェクトのように動作しますが、ズームを含むすべての変換が適用できます。

IlvLine

IlvLine オブジェクトは、任意の 2 点を結ぶ直線として表示されます。

IlvArrowLine

IlvArrowLine オブジェクトは、線の軌道上に描かれた小さい矢先付きで、任意の 2 点を結ぶ直線として表示されます。

IlvReliefLine

IlvReliefLine オブジェクトは、立体的な線として表示されます。IlvReliefLine の外観は線幅により異なります。

IlvMarker

IlvMarker オブジェクトは、指定の位置に特定のグラフィック・シンボルとして描画されます。

IlvZoomableMarker

IlvZoomableMarker オブジェクトは、次のようにズームできる IlvMarker オブジェクトのバージョンです。

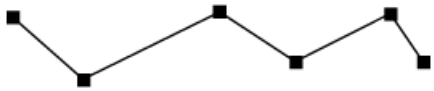
- 縮小では、現在のサイズが変換されたバウンディング・ボックスに一致するようになります。
- 拡大では、現在のサイズが IlvMarker::setSize メソッドで指定されたサイズに固定されます。

IlvPolyPoints

IlvPolyPoints は、複数の点座標からなる形を持つすべてのクラスが導出される抽象クラスです。

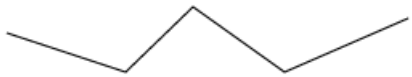
IlvPolySelection

IlvPolySelection クラスは、IlvPolyPoints タイプのオブジェクトのすべての点で四角を塗りつぶすために使用されます。



IlvPolyline

IlvPolyline オブジェクトはつながった線分として表示されます。



IlvArrowPolyline

IlvArrowPolyline オブジェクトは、ポリラインとして表示され、1 または複数の矢印をさまざまな線に追加します。

IlvPolygon

IlvPolygon オブジェクトは塗りつぶし多角形として表示されます。

IlvOutlinePolygon

IlvOutlinePolygon オブジェクトは、輪郭のある塗りつぶし多角形として表示されます。

IlvRectangle

IlvRectangle オブジェクトは、輪郭のある四角形として表示されます。

メモ: 四角形は、90 度、180 度、270 度、360 度に回転できます。他の角度に回転する必要がある場合は、多角形を使用します。

IlvFilledRectangle

IlvFilledRectangle オブジェクトは塗りつぶし四角形として表示されます。

IlvRoundRectangle

IlvRoundRectangle オブジェクトは、輪郭のある角丸四角形として表示されます。

IlvFilledRoundRectangle

IlvFilledRoundRectangle オブジェクトは、塗りつぶしの角丸四角形として表示されます。

IlvShadowRectangle

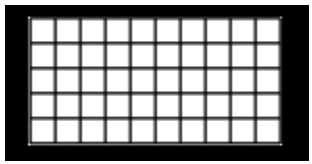
IlvShadowRectangle オブジェクトは、影付きの IlvFilledRectangle オブジェクトとして表示されます。

IlvShadowLabel

IlvShadowLabel オブジェクトは、それを含む矩形でクリッピングされた文字列を含む IlvShadowRectangle として表示されます。

IlvGridRectangle

IlvGridRectangle オブジェクトは、矩形グリッドとして表示されます。



IlvReliefRectangle

IlvReliefRectangle オブジェクトは、塗りつぶし四角形として立体的に表示されます。

IlvReliefLabel

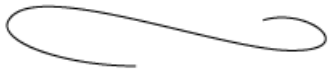
IlvReliefLabel オブジェクトは、テキストを 1 行含む立体四角形として表示されます。

IlvReliefDiamond

IlvReliefDiamond オブジェクトは、塗りつぶしひし形として立体的に表示されます。

IlvSpline

IlvSpline オブジェクトは、外郭ベジェ・スプラインとして表示されます。



IlvClosedSpline

`IlvClosedSpline` オブジェクトが、閉じたベジェ・スプラインとして表示されます。

IlvFilledSpline

`IlvFilledSpline` オブジェクトは、塗りつぶされたベジェ・スプラインとして表示されます。

複合グラフィック・オブジェクト

複合グラフィック・オブジェクト・クラスおよびサブクラスは、`IlvGraphic` のサブタイプ化されたオブジェクトのインスタンスを参照するメンバ関数を提供します。これらの参照は、次の場合に使用できます。

- 多角形を塗りつぶす。58 ページの *多角形の塗りつぶし : IlvGraphicPath* を参照してください。
- オブジェクトをグループ化する。59 ページの *オブジェクトのグループ化 : IlvGraphicSet* を参照してください。
- オブジェクト自体を複製または変更することなく、1 つのオブジェクトのプロパティを変更する。59 ページの *オブジェクトの参照 IlvGraphicHandle* を参照してください。

たとえば、千のトランジスタを表示するコンピュータのスキーマでは、千のイメージを個々に作成するよりも、トランジスタのイメージを 1 つとそれを参照する千のハンドル・オブジェクトを作成するほうがメモリの使用を大幅に抑えられます。

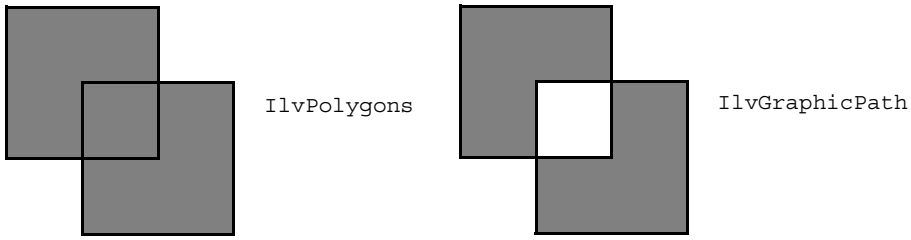
多角形の塗りつぶし：IlvGraphicPath

IlvGraphicPath オブジェクトは、*ポリポイント*・オブジェクトの集まりです。つまり各オブジェクトは一連の点で構成されています。ポリポイント・オブジェクトは、オブジェクトの描画ルールのアトリビュート値により異なって描かれます。

- IlvStrokeOnly: ポリライン
- IlvFillOnly: 塗りつぶし多角形。
- IlvStrokeAndFill: 上記の両方。つまり輪郭のある塗りつぶし多角形。

IlvSimpleGraphic サブクラスによって定義されたパレットは、多角形の輪郭を描画するのに使われます。IlvGraphicPath は、2 つめのパレット (backgroundPalette) が多角形を塗りつぶすように定義します。

IlvGraphicPath と IlvPolygon 関数の両方では、ポリポイントの描画に使用するリソース (色などのグラフィック・アトリビュート) が同じであるのに対し、形状が互いに影響を与える方法は異なります。各ポリポイントは他のポリポイントのレンダリングに影響します (IlvStrokeOnly モードには該当しません)。たとえば、ポリポイントはその点の位置によって、通常の多角形または別の多角形内の空白として表示されます。



IlvGraphicPath ではまた、ポリポイントを描画するときに、ユーザ固有のアクションを使用できます。これは、データ構造体を IlvGraphicPath に付加して行います。

IlvGraphicPath のバウンディング・ボックスは、そのパスで表示されているグラフィック・オブジェクトのバウンディング・ボックスを考慮していないことにご注意ください。IlvGraphic はステップ・データ構造でのみ認識されます。ただし、IlvGraphicPath は、グラフィック・パスのバウンディング・ボックスを指定した値だけ拡大できるメンバ関数を提供しています。

```
void setBBoxExtent(IlUInt extent);
```

通常は、次のようになります。

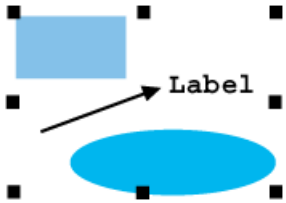
```
grpath->setPathDrawingData(new IlvPathDrawingData(step, obj));
grpath->setBBoxExtent(bboxExtension);
```

ここで、bboxExtension は obj のジオメトリおよびその表示方法 (回転が含まれるか否か) により計算されます。

オブジェクトのバウンディング・ボックスの対角線は、bboxExtension に適切な値です。

オブジェクトのグループ化 : IlvGraphicSet

IlvGraphicSet オブジェクトは、IlvGraphic オブジェクトのセットを編成します。



これは、オブジェクトが含むメンバ関数を呼び出すことにより (たとえば、IlvGraphicSet の draw メソッドは、グラフィック・セットに含まれるオブジェクトの draw メソッドを呼び出します)、すべてのジオメトリ・メンバ関数およびグラフィック・メンバ関数を実装します。

オブジェクトの参照 IlvGraphicHandle

IlvGraphicHandle オブジェクトは、IlvGraphic オブジェクトの参照に使用します。IlvGraphicHandle オブジェクトは*ハンドル*・オブジェクト (または単に*ハンドル*) と呼ばれ、IlvGraphic オブジェクトは*被参照*オブジェクトと呼ばれます。

オブジェクトの参照

この関係により、ハンドル・オブジェクトを使って被参照オブジェクトに間接的にアクセスできます。また、同じ被参照オブジェクトは複数のハンドルの間で共有できます。したがって、ハンドル・オブジェクトを通して新しいハンドルを作成することにより、複雑なグラフィック・オブジェクトを何度も複製できます。また、新しいハンドルはすべて同じ元のオブジェクトを参照します。ハンドルは新しいイメージを作成するのに比べメモリ使用が大幅に少ないため、ハンドル・オブジェクトの使用は非常に経済的です。

オブジェクトの所有

ハンドル・オブジェクトを関連付けられている固有の被参照オブジェクトの*所有者*にすることができます。この場合、被参照オブジェクトに直接アクセスせずにそのハンドルを通じてのみアクセスします。

ハンドルがその被参照オブジェクトを所有している場合、ハンドルを削除するとハンドルと被参照オブジェクトの両方が削除されます。一方、ハンドルとその被参照オブジェクトに所有関係がない場合は削除によってハンドルだけが削除され、もう一方のグラフィック・オブジェクトはそのまま残されます。

IlvTransformedGraphic

専用のグラフィック・ハンドル・サブクラス `IlvTransformedGraphic` を使用して、同じオブジェクトを複数回、異なったジオメトリ変換を適用して表示できます。

`IlvTransformedGraphic` クラスは、*handle* 基本クラスの `IlvGraphicHandle` から派生しています。`IlvTransformedGraphic` タイプのオブジェクトは、`IlvGraphic` クラスの特定の被参照オブジェクトに関連付けられたハンドルの 1 種です。`IlvTransformedGraphic` インスタンスは、グラフィック変換をその被参照オブジェクトに適用して導出されます。

オブジェクト・ジオメトリは、丸め誤差により、さまざまな変換で問題が起こります。これを避けるために、`IlvTransformedGraphic` をオブジェクトに関連付けることができます。

IlvFixedSizeGraphic

専用のグラフィック・ハンドル・サブクラスである `IlvFixedSizeGraphic` を使用して、オブジェクトを常に同じサイズで表示することができます。たとえば、マップ・ビューアを終了させるために使用する `IlvButton` オブジェクトで表示されているマップがあるとします。マップがズーム、アンズームされても、ボタンは同じサイズのままです。これを行うには、専用の `IlvGraphicHandle` オブジェクト `IlvFixedSizeGraphic` を持つボタンを参照します。

`IlvFixedSizeGraphic` クラスは `IlvTransformedGraphic` と同様に、`IlvGraphicHandle` クラスから派生しています。`IlvFixedSizeGraphic` オブジェクトは一種のハンドルであり、これは `IlvGraphic` クラスの特定の被参照オブジェクトに関連付けらていれます。`IlvFixedSizeGraphic` のインスタンスは、その被参照オブジェクトにグラフィック変換を適用することで導出されます。これで被参照オブジェクトの表示サイズは変更できません。

どのような変換が適用されても、オブジェクトは参照ポイントに比例して同じ寸法、定数オフセットを保持します。これらの値は内部的に IBM ILOG Views により計算されるか、またはユーザにより指定されます。

IlvGraphicInstance

専用のグラフィック・ハンドル・サブクラスである IlvGraphicInstance を使用して、オブジェクトをグラフィック・リソース変更と共にカプセル化できます。

IlvGraphicInstance は、他のパレットのアトリビュートで描画できるように、他のグラフィック・オブジェクトを参照します。オプションで、IlvTransformer を使用して、ジオメトリ変換をこのオブジェクトに適用できます。

その他のベース・クラス

IlvSimpleGraphic のサブクラスには、より複雑なグラフィック・オブジェクトのベース・クラスを形成するものがあります。

IlvGauge

ゲージは、最小値と最大値の間に含まれる特定の値を表すグラフィック・オブジェクトです。IlvGauge は、すべてのゲージ・オブジェクトが派生するメインの抽象クラスです。

IlvScale

IlvScale は、スケール・オブジェクト・クラスのインスタンスをすべて取得する抽象クラスです。これは、スケールに関する基本的な必要情報を管理します。

IlvGadget

IlvGadget クラスは、すべての IBM ILOG Views Gadgets パッケージ・クラス用のベース・クラスです。これは、グラフィック・オブジェクトを作成するのに必要なパラメータに影管理ができるパレットを提供することにより、ゲージのすべての基本機能を実装します。

ゲージの詳細については、『Gadgets』マニュアルを参照してください。

IlvGroupGraphic

IlvGroupGraphic は、グラフィック・オブジェクト一式をグループとして表示、操作するのに使用するグラフィック・オブジェクトです。このクラスは、IBM ILOG Views Prototypes パッケージで使われます。

詳細については、『Prototypes』マニュアルを参照してください。

IlvMapxx

IlvMapxx クラスのいくつかは IlvSimpleGraphic のサブクラスであり、スケールのような IBM ILOG Views Maps パッケージ用にさまざまなグラフィック・サービスを提供しています (IlvMapScale、IlvMapDefaultScaleBar、IlvMapDefaultNeedle など)。

すべてのマッピング・クラスの詳細については、『Maps』マニュアルを参照してください。

新規グラフィック・オブジェクト・クラスの作成

以下は、グラフィック・オブジェクトをサブタイプ化し、新規グラフィック・オブジェクト・クラスを作成する方法の例です。

例 : ShadowEllipse

この例では、新規グラフィック・オブジェクト ShadowEllipse を作成します。これは IlvSimpleGraphic から継承するものです。

ShadowEllipse オブジェクトは、通常の IlvEllipse オブジェクトで、下に影が付いています。

この例では、このようなオブジェクトを、IlvSimpleGraphic クラスのサブタイプを実装してゼロから設計する方法を説明します。これはもっともよく用いられる手順です。ここでは、幾何学プロパティと描画を処理するメンバ関数の実装方法、このオブジェクトのパレットの操作方法、さらにオブジェクトを永続的にする方法を説明します。

例の作成手順

例は、次の手順で作成します。

- *グラフィック・オブジェクトをサブタイプ化する基本手順*
- *IlvGraphic メンバ関数の再定義*
- *ヘッダー・ファイルの作成*
- *オブジェクト関数の実装*

- *パレットの更新*
- *オブジェクト記述の保存と読み込み*

グラフィック・オブジェクトをサブタイプ化する基本手順

IlvGraphic クラスの派生クラスを作成するには、以下の手順に従います。

1. 新規クラスと必要なオーバーロード・メンバ関数を宣言するヘッダー・ファイルを作成します。すべてのメンバ関数をオーバーロードする必要はありません。
2. DeclareTypeInfo(); ステートメントをクラス定義に追加します。

   これは、入出力操作およびクラス階層情報に必要なフィールドとメンバ関数宣言を作成します。
3. DeclareIOConstructors(ShadowEllipse); ステートメントをクラス宣言に追加します。これは 2 つの追加コンストラクタを宣言します。

- 次のコンストラクタは、新しい ShadowEllipse グラフィック・オブジェクトを初期化します。これは source の複製です。

```
ShadowEllipse(const ShadowEllipse& source);
```

- 次のコンストラクタは、新しい ShadowEllipse グラフィック・オブジェクトを、inputfile で読み込まれたパラメータから初期化します。

```
ShadowEllipse(IlvInputFile& inputfile,
              IlvPalette*        palette   = 0);
```

4. 実装ファイル (通常は class.cpp) を作成し、必要なメンバ関数を実装します。関数の本文外に、次の 2 つのマクロの呼び出しを追加します。

- IlvRegisterClass。クラス階層情報を更新します。
- IlvPredefinedIOMembers。メンバ関数 copy および read の定義に使用します。

IlvGraphic メンバ関数の再定義

次の IlvGraphic のメンバ関数は、常に再定義する必要があります (これらは IlvGraphic を抽象クラスにするメンバ関数です)。

```
virtual void draw(IlvPort* dst,
                  const IlvTransformer* t = 0,
                  const IlvRegion* clip = 0) const;
virtual void boundingBox(IlvRect& bbox,
                         const IlvTransformer* t = 0) const;
virtual void applyTransform(const IlvTransformer* t);
virtual void write(IlvOutputFile&) const;
```

IlvGraphic::move、IlvGraphic::resize、IlvGraphic::rotate、IlvGraphic::contains などのメンバ関数は、IlvGraphic クラスからデフォルト実装されています。つまり、IlvGraphic::resize は、applyTransform 関数などの呼び出しにより実装されます。

新規クラスにこれらのメンバ関数かを定義する親がある場合は、この親クラスから簡単に関数を継承できます。

ヘッダー・ファイルの作成

この例では、新規クラスと必要なオーバーロード・メンバ関数を宣言するヘッダー・ファイルを作成します。

ヘッダー・ファイル shadellp.h には次の行が含まれます。

```
#define DefaultShadowThickness 4

class  ShadowEllipse
: public IlvSimpleGraphic {
public:
    ShadowEllipse(IlvDisplay*    display,
                  const IlvRect&     rect,
                  IlUShort      thickness = DefaultShadowThickness,
                  IlvPalette*    palette   = 0)
    : IlvSimpleGraphic(display, palette),
      _rect(rect), _thickness(thickness)
    {
        _invertedPalette = 0;
        computeInvertedPalette();
    }
    ~ShadowEllipse();

    virtual void draw(IlvPort*, const IlvTransformer* t = 0,
                      const IlvRegion* clip = 0) const;
    virtual IlBoolean contains(const IlvPoint& p,
                               const IlvPoint& tp,
                               const IlvTransformer* t) const;
    virtual void boundingBox(IlvRect& rect,
                             const IlvTransformer* t = 0) const;
    virtual void applyTransform(const IlvTransformer* t);
    IlUShort getThickness() const
        { return  _thickness; }
    void      setThickness(IlUShort thickness)
        { _thickness = thickness; }

    virtual void setBackground(IlvColor* c);
    virtual void setForeground(IlvColor* c);
    virtual void setMode(IlvDrawMode m);
    virtual void setPalette(IlvPalette* p);

    DeclareTypeInfo();
    DeclareIOConstructors(ShadowEllipse);
protected:
    IlvRect     _rect;
    IlUShort   _thickness;
```

```
    IlvPalette* _invertedPalette;
    void computeInvertedPalette();
};
```

このオブジェクトは、標準 IBM ILOG Views ライブラリにある他の少数のオブジェクトと同様、2 つの異なった IlvPalette オブジェクトを利用します。これは、楕円とその影を描画する際に、ダミーのパレット・オブジェクトを作成する必要がないため、描画時間という点でオブジェクトの効率を高めたい場合に用いられる一般的な方法です。

ShadowEllipse クラスは、メンバ関数 draw、contains、boundingBox を定義します。また必要なパレット管理関連のメンバ関数も定義し、標準パレット・オブジェクト (IlvSimpleGraphic に格納されているもの) と新しいパレット・オブジェクトである _invertedPalette の両方を更新します。

この例では、入出力関数は宣言されていません。実際は、入出力関数はそれらを外部のものとして宣言する DeclareTypeInfo マクロにより宣言されます。これらのメンバ関数は、read、write、および copy です。これらにはデフォルトで実装されていないため、IlvGraphic クラスの各サブクラスにこれらのバージョンを提供する必要があります。DeclareTypeInfoRO と呼ばれるこの 2 番目のバージョンがあり、これはこのオブジェクトのタイプが保存されないことがわかっていれば、メンバ関数 write を宣言しません。

オブジェクト関数の実装

この例では、新規クラスと必要なオーバーロード・メンバ関数を宣言するヘッダー・ファイルを作成します。

このセクションでは、shadellp.cpp ファイルに実装されている関数のコードについて説明します。

computeInvertedPalette メンバ関数

```
void
ShadowEllipse::computeInvertedPalette()
{
    IlvPalette* newPalette = getDisplay()->getInvertedPalette(getPalette());
    newPalette->lock();
    if (_invertedPalette)
        _invertedPalette->unLock();
    _invertedPalette = newPalette;
}
```

メンバ関数 computeInvertedPalette は、メンバ関数 getPalette の呼び出しにより取得したパレットから反転パレットを計算します。この反転パレットを作成し、前のパレットがある場合はロック解除し、新しいパレットをロックします。

この関数は、(適切なメンバ関数をオーバーロードすることにより) 元のパレットが変更されるたびに、およびオブジェクトが最初に作成されるときに呼び出されます。

この 2 番目のパレットの作成は、多少奇妙に思われるかもしれません。メンバ関数 draw で、2 番目のパレットは、2 つの IlvDisplay 描画メンバ関数を呼び出すときのみに使われます。別のメソッドは、これらのメンバ関数を呼び出す前にメンバ関数 IlvPalette::invert を呼び出し、別の IlvPalette::invert を呼び出すことによりパレットを元の状態に戻してしまう可能性があります。IBM ILOG Views を使用した作成は、これがオブジェクトを操作する効率的な方法ではないことを示しています。パレット管理は、IBM ILOG Views によって実行される非常に効率的なタスクの 1 つです。必要なときはパレット管理を使用してください。

デストラクタ

```
ShadowEllipse::~ShadowEllipse()
{
    _invertedPalette->unLock();
}
```

デストラクタでは、反転したパレットをディスプレイに解放し、他のオブジェクトによって使用されない場合は削除できるようにする必要があります。

draw メンバ関数

メンバ関数 draw は、2 つの楕円を塗りつぶし、一番上の楕円ボーダーを描画します。グローバル・バウンディング矩形 (_rect) は、実際に両方の楕円を覆います。

メンバ関数 draw を下記に示します。これは、オブジェクトの描画は、IlvDisplay クラスのプリミティブ・メンバ関数にのいくつかを呼び出すだけであることを示しています。

```
void
ShadowEllipse::draw(IlvPort* dst, const IlvTransformer* t,
                        const IlvRegion* clip) const
{
    // Transform the bounding rectangle ____________________________
    IlvRect rect = _rect;
    if (t)
        t->apply(rect);

    // Store both the display and palette __________________________
    IlvPalette* palette = getPalette();

    // Find a correct value for thickness __________________________
    IlUShort thickness = _thickness;
    if ((rect.w() <= thickness) || (rect.h() <= thickness))
        thickness = IlMin(rect.w(), rect.h());

    // Compute actual shadow rectangle _____________________________
    rect.grow(-thickness, -thickness);
    IlvRect shadowRect = rect;
    shadowRect.translate(thickness, thickness);

#if defined(USE_2_PALETTES)
    // Set the clipping region for both palettes ___________________
    if (clip) {
        palette->setClip(clip);
        _invertedPalette->setClip(clip);
```

```
    }
   // Fill shadow Ellipse ____________________________________________
    dst->fillArc(palette, shadowRect, 0., 360.);

    // Fill inverted Ellipse ____________________________________________
    dst->fillArc(_invertedPalette, rect, 0., 360.);

    // Draw ellipse ____________________________________________________
    dst->drawArc(palette, rect, 0., 360.);
    if (clip) {
        palette->setClip();
        _invertedPalette->setClip();
    }
#else  /* !USE_2_PALETTES */
    // Set the clipping region for both palettes ____________________
    if (clip)
        palette->setClip(clip);

    // Fill shadow ellipse ____________________________________________
    dst->fillArc(palette, shadowRect, 0., 360.);

    // Compute inverted palette and fill inverted ellipse _____________
    palette->invert();
    dst->fillArc(palette, rect, 0., 360.);
    palette->invert();
 // Draw elliptic border ___________________________________________
    dst->drawArc(palette, rect, 0., 360.);

    // Set the clipping region for both palettes ____________________
    if (clip)
        palette->setClip();
#endif /* !USE_2_PALETTES */
}
```

ここではトランスフォーマ `t` を使用して描画を実行する必要はありません。これはどんな変換でも幅を同じにするためです。

複雑な描画を行う場合は、`clip` パラメータを使用できます (この場合は必要ありません)。この関数から返す前に、影響を受けたパレットすべてのクリッピング領域を空の領域にリセットする必要があります。

boundingBox メンバ関数

メンバ関数 `boundingBox` は、グローバル・バウンディング矩形を変換します。

```
void
ShadowEllipse::boundingBox(IlvRect& rect,
                           const IlvTransformer* t) const
{
    rect = _rect;
    if (t)
        t->apply(rect);
}
```

> ***メモ:*** *消去エラーを避けるために、バウンディング・ボックスには完全な描画が含まれていなければなりません。*

contains メンバ関数

メンバ関数 contains は、2 つの楕円のうち 1 つの中に点がある場合は IlvTrue を返します。すべての座標はビューの座標系を基準に指定されます。

```
static IlBoolean
IsPointInEllipse(const IlvPoint& p, const IlvRect& bbox)
{
    if (!bbox.contains(p))
        return IlFalse;
    IlUInt rx = bbox.w() / 2,
           ry = bbox.h() / 2;
    IlUInt dx = (p.x() - bbox.centerx()) * (p.x() - bbox.centerx()),
           dy = (p.y() - bbox.centery()) * (p.y() - bbox.centery()),
    rrx = rx*rx,
    rry = ry*ry;
    return (rrx * dy + rry * dx <= rrx * rry) ? IlTrue : IlFalse;
}
IlBoolean
ShadowEllipse::contains(const IlvPoint&, const IlvPoint& tp,
                        const IlvTransformer* t) const
{
    IlvRect rect = _rect;
    if (t)
        t->apply(rect);
    if ((rect.w() <= _thickness) || (rect.h() <= _thickness))
        return IsPointInEllipse(tp, rect);
    else {
        rect.grow(-_thickness, -_thickness);
        IlvRect shadowRect = rect;
        shadowRect.translate(_thickness, _thickness);
        return (IlBoolean)(IsPointInEllipse(tp, rect) ||
                           IsPointInEllipse(tp, shadowRect));
    }
}
```

contains がスタティック関数 IsPointInEllipse を呼び出しています。これは点のパラメータが矩形パラメータに定義された楕円の中にあるかどうかをテストします。

applyTransform メンバ関数

applyTransform メンバ関数は、トランスフォーマをグラフィックの矩形に適用します。

```
void ShadowEllipse::applyTransform(const IlvTransformer* t)
{
 if (t)
   t->apply(_rect);
}
```

パレットの更新

変更が元のパレットに適用された際にパレットが両方とも更新されていることを確認するには、次のメンバ関数をオーバーロードする必要があります。

```
void
ShadowEllipse::setBackground(IlvColor* color)
{
    IlvSimpleGraphic::setBackground(color);
    computeInvertedPalette();
}

// --------------------------------------------------------------------------
void
ShadowEllipse::setForeground(IlvColor* color)
{
    IlvSimpleGraphic::setForeground(color);
    computeInvertedPalette();
}

// --------------------------------------------------------------------------
void
ShadowEllipse::setMode(IlvDrawMode mode)
{
    getPalette()->setMode(mode);
    _invertedPalette->setMode(mode);
}

// --------------------------------------------------------------------------
void
ShadowEllipse::setPalette(IlvPalette* palette)
{
    IlvSimpleGraphic::setPalette(palette);
    computeInvertedPalette();
}
```

オブジェクト記述の保存と読み込み

ここで、クラス概要で `DeclareTypeInfo` マクロを使用して宣言した入出力関数を使用します。

copy メンバ関数および read メンバ関数

メンバ関数 `copy` および `read` を定義するために、次のマクロも使用できます。

```
IlvPredefinedIOMembers(IlvShadowEllipse);
```

このマクロは、`IlvRegisterClass` のように、関数定義ブロックの外部、実装ファイル内で使用しなければなりません。

これは、以下と同等です。

```
IlvGraphic*
ShadowEllipse::read(IlvInputFile& input, IlvPalette* palette)
{
    return new ShadowEllipse(input, palette);
}
IlvGraphic*
ShadowEllipse::copy() const
{
    return new ShadowEllipse(*this);
}
```

スタティック・メンバ関数 read は、クラス読み込みコンストラクタを呼び出し、新しいインスタンスを返します。マクロ DeclareIOConstructors は、ヘッダー・ファイルで読み込みコンストラクタおよびコピー・コンストラクタを宣言します。これらのコンストラクタの定義は、実装ファイルで次のように書く必要があります。

```
ShadowEllipse::ShadowEllipse(IlvInputFile& f,
                             IlvPalette* pal)
: IlvSimpleGraphic(f, pal),
  _rect(),
  _thickness(0)
{
    int thickness;
    f.getStream() >> _rect >> thickness;
    _thickness = (IlvDim)thickness;
    _invertedPalette = 0;
    computeInvertedPalette();
}
```

上記のコンストラクタは、スーパークラスの読み込みコンストラクタを呼び出し、そのコンストラクタは
ストリーム・オブジェクトからスーパークラス固有の情報を読み込みます。この後、サブクラスは自分の情報を読み込めます。

メンバ関数 copy は、IlvShadowEllipse のコピーを作成し、クラス・コピー・コンストラクタを呼び出します。

```
ShadowEllipse::ShadowEllipse(const ShadowEllipse& source)
: IlvSimpleGraphic(source),
  _rect(source._rect),
  _thickness(source._thickness)
{
    _invertedPalette = source._invertedPalette;
    _invertedPalette->lock();
}
```

write メンバ関数

メンバ関数 write は、矩形寸法および影の幅を指定した ostream 出力ストリームに書き込みます。

```
void
ShadowEllipse::write(IlvOutputFile& f) const
{
    f.getStream() << _rect << IlvSpc() << (int)_thickness;
}
```

この write メソッドは特別です。これは IlvSimpleGraphic スーパークラスに書き込む情報がないためです。スーパークラスに書き込む情報がない場合、スーパークラス write メソッドを呼び出して read メソッドとの一貫性を保ってください。以下は、通常の write メソッドの例です。

```
void
IlvRoundRectangle::write(IlvOutputFile& os) const
{
    IlvRectangle::write(os);
    os.getStream() << IlvSpc() << _radius;
}
```

IlvRegisterClass マクロ

```
IlvRegisterClass(ShadowEllipse, IlvSimpleGraphic);
```

関数の本文外で、IlvShadowEllipse クラスを IlvSimpleGraphic クラスのサブクラスとして登録する必要があります。

章

3

グラフィック・リソース

グラフィック・リソースを実装するクラスは、IlvResource とそのサブクラスです。IBM® ILOG® Views には、5 つの基本グラフィック描画リソース (IlvResource のサブクラス) があります。これらは、*色*、*線の種類*、*パターン*、*色のパターン*、*フォント*で、それぞれ対応するクラスにサポートされています。IlvResource の別のサブクラスである IlvPalette クラスはリソースのグループを管理します。また、IlvQuantizer は、すべての色変換クラスの抽象ベース・クラスです。

- *IlvResource: リソース・オブジェクトのベース・クラス*
- *IlvColor: 色クラス*
- *IlvLineStyle: 線の種類のクラス*
- *IlvPattern および IlvColorPattern: パターン・クラス*
- *IlvFont: フォント・クラス*
- *IlvCursor: カーソル・クラス*
- *その他の描画パラメータ*では、パレットを使用して制御する追加設定について説明します。
- *IlvPalette: リソースのグループを使用した描画*
- *IlvQuantizer: イメージ色量子化クラス*

IlvResource: リソース・オブジェクトのベース・クラス

IlvPort クラスの関数のすべての描画メンバ関数は、IlvPalette タイプのパラメータを取ります。これは、IlvResource のサブクラスです。

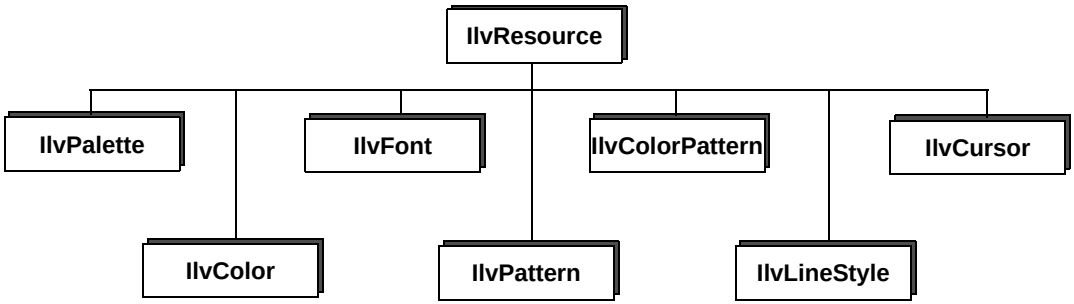


***図 3.0** IlvResource 階層*

リソースについての詳細を、以下のトピックに分けて説明します。

- *定義済みグラフィック・リソース*
- *名前付きのリソース*
- *リソースの作成と定義：ロックとロック解除*

定義済みグラフィック・リソース

以下は、定義済みグラフィック・リソースを生成する IlvDisplay メンバ関数を要約したリストです。

```
IlvColor*     defaultBackground() const;
IlvColor*     defaultForeground() const;
IlvFont*      defaultFont() const;
IlvLineStyle* defaultLineStyle() const;
IlvPattern*   defaultPattern() const;
IlvCursor*    defaultCursor() const;
```

- パレットの前景色のデフォルト値は、IlvDisplay::defaultForeground によって返される色です。
- パレットの背景色のデフォルト値は、IlvDisplay::defaultBackground によって返される色です。
- パレットのフォントのデフォルト値は、IlvDisplay::defaultFont によって返されるフォントです。
- パレットの線の種類のデフォルト値は、IlvDisplay::defaultLineStyle によって返される線の種類です。

- パレットの塗りつぶしパターンのデフォルト値は、IlvDisplay::defaultPattern によって返されるパターンです。
- パレットのカーソルのデフォルト値は、IlvDisplay::defaultCursor によって返されるパターンです。

グラフィック・リソース・クラスは IlvResource のサブクラスです。これらのサブクラスの詳細については、以下を参照してください。

- *IlvColor: 色クラス*
- *IlvLineStyle: 線の種類のクラス*
- *IlvPattern および IlvColorPattern: パターン・クラス*
- *IlvFont: フォント・クラス*
- *IlvCursor: カーソル・クラス*

IBM ILOG Views では、描画リソースは IlvPalette クラスのオブジェクトに組み込まれています。このクラスは IlvResource のサブクラスでもあります。パレットの詳細については、以下を参照してください。

- *IlvPalette: リソースのグループを使用した描画*

名前付きのリソース

次のように、IlvResource メンバ関数を使用してリソースに特定の名前を割り当てることができます。

```
void setName(const char* name);
const char* getName() const;
```

> ***メモ:*** *IlvFont および IlvColor は名前フィールドをプライベートに使用して、IlvResource::setName の使用を制限しています。IlvFont は IlvResource::setName の使用を無効にし、IlvColor は可変色の名前の変更のみができます。非可変色は定義済みの名前か、RGB 値に基づいたデフォルト名のいずれかです。*

リソースの作成と定義：ロックとロック解除

グラフィック・リソースの作成は、一般的にほとんどのグラフィック・システムでメモリを多く使用するため、IBM® ILOG® Views ではキャッシング・メカニズムを実装してグラフィック・リソースの割り当てを最小限に抑えています。

リソース・オブジェクトは、ユーザ・アプリケーションの IlvDisplay インスタンスが保持します。これらは通常、演算子 new や delete を使用した作成や破壊は行いません。代わりに、IBM ILOG Views が、以下のメンバ関数を提供します。

- IlvDisplay メソッド getXXX。ここで XXX は Ilv プレフィックスの付かないリソース・クラス名 (たとえば、IlvDisplay::getColor、IlvDisplay::getFont など) を表します。
- メソッド IlvResource::lock および IlvResource::unLock はそれぞれ、リソースの内部参照カウントを増加、減少します。このカウントがゼロに達すると、リソースは削除されます。

ロックとロック解除の手順

グラフィック・リソースは、以下の方法で使用してください。

1. IlvDisplay インスタンスにリソースの割り当てを要求します。システムにこのリソースがすでに存在する (たとえば、問い合わせをした色がパレットですでに使用されている) 場合、それ以上の割り当ては行われず、既存のリソースが返されます。
2. IlvResource::lock を呼び出してこのリソースを保護する旨を IBM ILOG Views に伝えてから、そのリソースを使用します。
3. IlvResource::unLock を使用して、IBM ILOG Views にリソースの使用を終了した旨を伝えます。

リソース管理は、リソースのロックおよびロック解除方法と緊密に関係しています。永続オブジェクトの 1 つに特定のリソースが必要な場合は、必ずこのメカニズムを使用して、IlvDisplay インスタンス内でそのリソースを確実に保護してください。アプリケーションで IlvDisplay, のインスタンスを 2 つ以上必要とする場合、リソースは異なる IlvDisplay コンテキスト間で共有できないため、環境ごとにリソースを作成する必要があります。

リソースの実行中は、IlvResource::lock の呼び出し数は IlvResource::unLock の呼び出し数と完全に一致しなければなりません。lock の呼び出し数の方が多い場合は、リソースが不必要になった場合、つまりアプリケーションの要求が制限されてもリソースは割り当てられたままになります。unlock の呼び出し数の方が多い場合は、メモリ・エラーによりアプリケーションが破壊される可能性があります。

ロックとロック解除のルール

グラフィック・リソースのロックおよびロック解除については、次のルールに従ってください。

- リソースを取得したらロックして使用し、使用が終了したらロック解除します。

- 確信が持てない限り、自分がロックしなかったリソースのロックを解除しないでください。
- ロックを解除した後にリソースを使用しないでください。これは、ポインタを解放した後に使用しないのと同様です。`IlvResource::unLock` は `delete` の可能性を意味します。
- リソースのロックやロック解除が必要でない場合があります。たとえば、オブジェクトの前景色を取得し、それをロックする別のオブジェクトに渡す場合です。この場合は、リソースのロックやロック解除は必要ありません。ロックやロック解除しても破損することはありません。

IlvColor: 色クラス

色についての詳細を、以下のトピックに分けて説明します。

- *色モデル*
- *IlvColor クラスの使用*
- *色モデルの変換*
- *影色の計算*

色モデル

IBM® ILOG® Views における色の記述は、`IlvColor` のインスタンスに格納されています。

RGB

IBM ILOG Views では､色はよく知られた RGB (赤/緑/青) システムを使用して処理されます。このシステムでは、色はその 3 つコンポーネント値によって完全に定義されます。`red`、`green`、および `blue` です。これらの値は、符号の付いていない 16 ビットの数字として格納されます。たとえば、*黒*はその 3 つのコンポーネントを 0 に設定し、*白*は、3 つのコンポーネントが 65535 に設定されています。

HSV

別の方法として、以下に示すように HSV (色相 / 彩度 / 輝度値) モデルを使用することもできます。

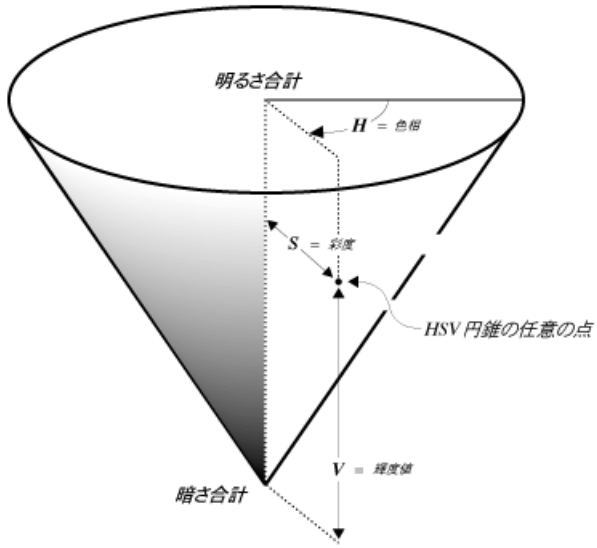


図 3.1 *色相、彩度、輝度値 (HSV) モデル*

前記の図は、3 つのパラメータに基づく数学モデルを示しています。これらは、H (色相)、S (彩度) および V (輝度値) です。次に示すのは、これらの可能な値です。

- **H** 色相パラメータである H は、0 ～ 360° の角度です。S および V パラメータの固定値 (それぞれの上限値までの適切な値) について、角度 H を 1 周させることにより色の全スペクトルを表示できます。
- **S** 任意の H パラメータについて、S 値を変えることで色の彩度が変化します。円錐の縦軸で、S がゼロになったところでは、彩度はありません。これは*灰色*の影があることを意味します。円錐の外面で、S の値が 1.0 になっているところでは、彩度が最高、つまり色がもっとも鮮やかになっています。
- **V** 3 つ目のパラメータ V は、角度 H を 1 周させることにより、色に当たる光の量、つまりスペクトルの*明るさ*を決定します。円錐の底で、V がゼロになっているところは、*黒*色を表します。上に移動するに従って、色相の円のスペクトルはどんどん明るくなります。縦軸の一番上で、V の値が 1.0 になっているところは白色を表します。

IlvColor クラスの使用

IBM ILOG Views では、`IlvColor` クラスを使うと RGA と HSV の両方の色モデルを操作できます。`IlvDisplay` オブジェクトに、色の取得を要求して色を取得します。

色は、一般的に、IlvDisplay オブジェクトによって内部で維持されている色のテーブルに格納されます。これはルックアップ・テーブルと呼ばれることがあります。メンバ関数 IlvColor::getIndex によって、色オブジェクトのインデックス (符号の付いていない長整数型) を取得できます。ピクセル値を色オブジェクトにマッピングする場合は、この数を色ビットマップに関する処理のリマップに使用できます。

色の名前

色には常に名前があります。IBM ILOG Views には、定義済みの色の名前一式があります。色の名前は、X Window の色命名方法に従っています。このセットの名前はそれぞれ、特定の RGB トリプレットに関連付けられています。色が定義済みの名前ではなく RGB 値により特定される場合は、この色のデフォルト名は次のように付けられます。"#RRRRGGGGBBBB"。ここで赤、緑、青の値はそれぞれ 16 進法の 4 桁で表されます。

ただし、名前を変えることができるのは可変色のみです。固定色の名前は変更できません。これは定義済みの色の名前か、色を定義する RGB 値から計算されるかのいずれかです。

新しい色

IlvColor にはパブリック・コンストラクタがないので、ディスプレイから色を取得する必要があります。いくつかの IlvDisplay::getColor メンバ関数を使うと新しい色を取得し、RGB 値、HSV 値、または名前を指定できます。また色が可変色か否かを示すこともできます。問題が発生して希望する色を作成できない場合、メンバ関数は 0 を返します。

IlvDisplay クラスは、未指定の色のコールバック値として使用されることが多い内部リソースを返す 2 つのメンバ関数を提供します。これらの関数は、以下の通りです。

```
IlvColor* defaultForeground();
IlvColor* defaultBackground();
```

通常、これらの前景色および背景色はそれぞれ黒、灰色です。これは、ディスプレイ・システムのリソース・メカニズムによって、どのような色でも簡単に設定することができます。

可変色

IBM ILOG Views の色は、*固定* (作成後に変更できない) または *可変* のいずれかです。後者の場合、画面にこの色で描画されている場合でも、変更メンバ関数を使用して色を動的に変更できます。

可変色は、IlvResource::setName メソッドを使用して名前を変更できます。固定色とは違い、可変色は透過的に共有されません。IlvDisplay::getColor メソッドは、パラメータ mutable が IlTrue の場合、常に新しいオブジェクトを作成します。内部リソース管理の点では、可変色は負担が大きくなります。

色モデルの変換

RGB システムから HSV に、またはその逆に色の値を変換するには、次の 2 つのグローバル関数が使用できます。

- IlvRGBtoHSV
- IlvHSVtoRGB

影色の計算

IlvComputeReliefColors グローバル関数を使用して、影付き効果を作る色を計算します。

IlvLineStyle: 線の種類のクラス

ユーザの IlvDisplay オブジェクトから IlvLineStyle リソース・オブジェクトを要求し、点線を描画する方法を指定することで、独自の線の種類を作成できます。

線の種類はメンバ関数 IlvLineStyle::getDashes が返す符号なし文字の配列です。配列の長さは、IlvLineStyle::getCount メンバ関数が返します。この配列は、削除したり変更したりしないでください。

IBM® ILOG® Views では、「pen down (ペン・ダウン) 」から始めて IlvLineStyle::getDashes 配列の最初の要素で示される前景色のピクセル数を描画します。次に 2 つ目の要素が、描画が再び開始されてから配列が完全に読み込まれるまでスキップされるピクセル数を示します。その後、ループが再開します。IlvLineStyle::getOffset メンバ関数は、ループが再開する前にスキップするピクセルの数を返します。

新しい線の種類

ユーザの IlvDisplay オブジェクトから IlvLineStyle リソース・オブジェクトを要求し、点線を描画する方法を指定することで、独自の線の種類を作成できます。

線の種類には名前を付けることができます。新しい線の種類を取得するには、IlvDisplay クラスの以下のメンバ関数を使用します。

```
IlvLineStyle* getLineStyle(IlUShort            count,
                           const unsigned char* dashes,
                           IlUShort            offset = 0);
IlvLineStyle* getLineStyle(const char* name)    const;
```

IlvDisplay クラスは、名前を使用して取得できる定義済みの線の種類一式を提供します。

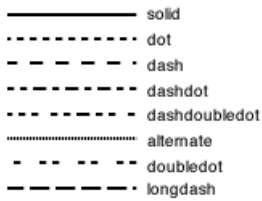


図3.2　線の種類

IlvPattern および IlvColorPattern: パターン・クラス

パターンは、ピクセルあたり 1 ビットのみで定義され、モノクロにすることもカラーにすることもできます。詳細は、以下を参照してください。

- *モノクロ・パターン*
- *色のパターン*

モノクロ・パターン

モノクロ・パターンは `IlvPattern` クラスによって処理されます。利用可能なデータにより使い分ける、以下の 2 つのコンストラクタが提供されています。

```
IlvPattern(IlvDisplay*    display,
           IlvDim         w,
           IlvDim         h,
           unsigned char* data);
IlvPattern(IlvBitmap*     bitmap);
```

最初のコンストラクタは、ピクセル幅が `w` でピクセル高が `h` のパターンを持つ新しい `IlvPattern` オブジェクトを初期化します。このオブジェクトは、ビット値の `data` 配列に格納されているデータを使用して塗りつぶされています。ピクセル値は左から右に、最上位ビットを最初に 16 ビット・ワードにパックします。そして各スキャン・ラインは上から下に格納し、補填して 16 ビットに合わせる必要があります。

2 つ目のコンストラクタは、新しい `IlvPattern` オブジェクトを、指定した `bitmap` モノクロ・イメージから初期化します。

以前定義したパターンを取得するには、メンバ関数 `IlvDisplay::getPattern` を使用します。

他のパターンは、IBM® ILOG® Views 内であらかじめ定義されており、名前別にアクセスできます。

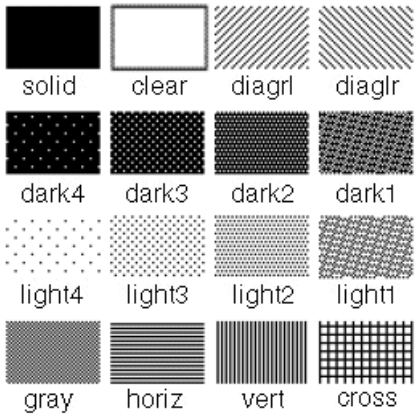


図3.3　定義済みパターン・リソース

色のパターン

パターンには色を付けることもでき、`IlvColorPattern` クラスのインスタンスによって表されます。

IlvFont: フォント・クラス

テキスト文字列は、次に示されるような特定のスペース値で描かれます。

図3.4　スペース値

`IlvFont` オブジェクトのパラメータは、メンバ関数 `IlvFont::getFamily`、`IlvFont::getSize`、`IlvFont::getStyle`、および `IlvFont::getFoundry`. で取得できます。

メンバ関数 `IlvFont::ascent`、`IlvFont::descent`、および `IlvFont::height` はフォント・メトリックを返します。

また、特定の文字列のメトリックを、メンバ関数 IlvFont::stringWidth、IlvFont::stringHeight、および IlvFont::sizes を呼び出して取得できます。

メンバ関数 IlvFont::isFixed は、フォント・オブジェクトがすべての文字で固定幅の場合は IlTrue を返します。

このフォントの最小文字幅および最大文字幅は、2 つのメンバ関数 IlvFont::maxWidth および IlvFont::minWidth で取得できます。返された値が同じである場合、IlvFont::isFixed は IlTrue を返します。

フォント・クラスの詳細については、以下を参照してください。

- *新しいフォント*
- *フォント名*

新しいフォント

IlvFont にはパブリック・コンストラクタがありません。新しいフォントは、2 つのメンバ関数 IlvDisplay::getFont のうち 1 つを使用して、ディスプレイから取得する必要があります。フォント名または次のようなフォントの特性一式を指定できます。

- ファミリ
- サイズ
- スタイル
- フォントのタイプ

フォント名

すべてのフォントには名前があります。フォントが有効なフォント名ではなくファミリ、サイズ、スタイル、フォントのタイプなどの値一式で作成される場合、IBM ILOG Views はこれらの値から名前を計算します。

```
"%family-size-style-foundry"
```

ここで:

- family は、パラメータ・ファミリとして指定される文字列です。
- size は、パラメータ・サイズの ASCII 表現です。
- style は、文字 B、I、U の組み合わせで、それぞれ太字、斜体、下線を表します (大文字小文字は関係ありません)。このフィールドは空欄にできます。この場合は通常のスタイルが想定されます。

- foundry は、文字列オプションです。これは多くの場合フォントをデザインした企業を識別します。このフィールドを指定することはほとんどありません。指定しない場合、終わりに付ける「-」も省略できます。

フォントの名前を変更することはできません。

次に示すのは、構文的に適切に作成された IBM ILOG Views フォント名の例です (これらはすべてのプラットフォームに存在する訳ではなく、有効なフォント名とは限りません)。

- "%helvetica-12-"
- "%time-12-BU"
- "%courier-14-i-adobe"
- "%terminal-11--bitstream"

IlvCursor: カーソル・クラス

IBM® ILOG® Views カーソルは画面に表示され、すべてのマウス動作に従うアイコンです。IBM ILOG Views では、カーソルは IlvCursor クラスで管理されています。他のカーソルはあらかじめ定義されており、名前別にアクセスできます。

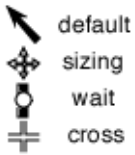


図 3.5 *定義済みカーソルとその名前*

その他の描画パラメータ

次のアトリビュートは描画操作に影響し、IlvPalette クラスで使われます。これらは、*線の太さ*、*塗りつぶしスタイル*、*塗りつぶしルール*、*円弧モード*、*描画モード*、*アルファ値*、および *アンチエイリアシング・モード*です。

これらのアトリビュートは、C++ タイプの定義で表されます。これらは IlvResource のサブクラスで表されないため、「リソース」とは呼ぶことができませんが、グラフィック・リソースと一緒に動作して IBM ILOG Views 描画の描画アトリビュートを定義します。

線の太さ

線の太さは符号なしの短整数型です。ゼロ (0) は有効な値で、最大限細く、早く描画できる幅線の線を作成します。

塗りつぶしスタイル

もっとも複雑なグラフィック・リソースはパターンのもので、モノクローム (2 色) ドメインの単純なマスキング・パターンと色の多彩なピクセル・パターンがあります。これを*塗りつぶしスタイル*といいます。

塗りつぶしの種類は、パターンが形を塗りつぶすときに使用する方法を示します。3 つの種類があり、`IlvFillStyle` 列挙型で表されます。モノクローム・パターンは、塗りつぶしモードが `IlvFillPattern` または `IlvFillMaskPattern` の場合に使用します。この値は、ユーザが作成したか、または `IlvDisplay` クラスの特定のメンバ関数が返す `IlvPattern` のインスタンスです。色のパターンとは、塗りつぶしモードが `IlvFillColorPattern` のときに形状を塗りつぶすために使用するものです。

IlvFillPattern

`IlvFillPattern` では、形状は選択したパターンでコピーされて塗りつぶされます。IBM ILOG Views のオブジェクトには、`IlvPattern` クラスのオブジェクトを参照するパターン・プロパティがあります。特定のパターンで形状を塗りつぶすには、次のようにします。

- 関連する `IlvPattern` オブジェクトの各「`0`」ピクセルは、現在の背景色でカラー・ピクセルを生成します。
- 関連する `IlvPattern` オブジェクトの各「`1`」ピクセルは、現在の前景色でカラー・ピクセルを生成します。

これは `IlvPalette` オブジェクトの塗りつぶし種類プロパティのデフォルト値です。

IlvFillMaskPattern

`IlvFillMaskPattern` は `IlvFillPattern` スタイルに似ていますが、関連する `IlvPattern` オブジェクトの「`0`」ピクセルは宛先ポートの対応するピクセルに影響しません。つまり、描画がその宛先をマスクします。

IlvFillColorPattern

`IlvFillColorPattern` の場合、形状を塗りつぶすために使用するパターンは、`IlvPalette` オブジェクトのパターン・プロパティではなく、その色のパターン・プロパティで示されます。これはフルカラー・パターンで領域を塗りつぶすとき、すなわち `IlvColorPattern` クラスの実際のオブジェクトで使用します。パターン・プロパティはこの塗りつぶしモードの場合には、何の役割も果たしません。

塗りつぶしルール

このアトリビュートは、自己交差している多角形の塗りつぶし方法を示します。これは多角形の表面の場合、「塗りつぶし」の意味が曖昧であるためです。

塗りつぶしルールは、塗りつぶす領域の形を定義する交差する線分の数に依存して、どのポイントが塗りつぶす多角形の内側にあると見なされるかを示します。IlvFillRule では次の 2 つのルールを提供しています。

- IlvEvenOddRule　このルールに基づくと、以下に示されるような複雑な多角形の場合、星の中央領域は多角形の内側にあるとはみなされないため、塗りつぶされません。これがデフォルト値となっています。
- IlvWindingRule　このルールに基づくと、星の中央領域は多角形の中にあると見なされるため塗りつぶされます。

図 3.6　IlvFillRule

円弧モード

円弧モードは、円弧を塗りつぶすためにそれらを閉じる方法を示します。これはつまり、どのように塗りつぶし円弧が描画されるかということです。これは V 字型の「パイ」を作る半径か、単純な「弦」線分のどちらかを使用します。IlvArIlvArcModecMode 列挙型が処理する 2 つの場合が考えられます。

- IlvArcPie　真円の中心から円弧の開始点と終了点に 2 つの線を追加することで円弧を閉じます。これはデフォルトのモードとなっています。
- IlvArcChord　開始点から終了点に線を追加することで円弧を閉じます。

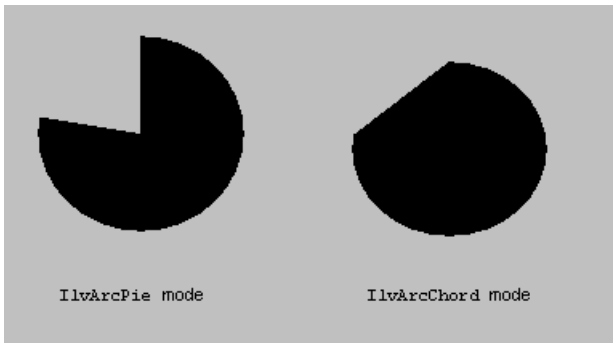


図3.7 *円弧モード*

描画モード

描画モードは、ピクセルが宛先ポートに送出されたときに実行される操作を指定します。この操作は、ソースのピクセル値がその場所に描画されようとしているときに、ターゲットのピクセルの値に影響するものです。描画モードには使用できる値がいくつかあり、これらは `IlvDrawMode` 列挙型で処理されます。一時的な描画に使用される `IlvModeXor` の値を除いて、これらのタイプは色で描画する場合に顕著な描画上の効果はありません。

- `IlvModeSet` できあがるピクセルはソース・ピクセルのコピーです。
- `IlvModeOr` できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの OR 演算の結果になります。
- `IlvModeAnd` できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの AND 演算の結果になります。
- `IlvModeXor` できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの XOR (exclusive or) 演算の結果になります。このモードは 2 度目には図形の削除に使用できます。
- `IlvModeNot` できあがるピクセルはターゲット・ピクセルへの NOT 演算の結果になります。ソース・ピクセルの値は使用しません。
- `IlvModeInvert` できあがるピクセルはソース・ピクセルへの NOT 演算の結果になります。
- `IlvModeNotOr` できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの NOT OR 演算の結果になります。
- `IlvModeNotAnd` できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの NOT AND 演算の結果になります。

- IlvModeNotXor　できあがるピクセルはソース・ピクセルとターゲット・ピクセルへの NOT XOR 演算の結果になります。IlvModeNotXor のセットで同じオブジェクトを 2 度描画すると、図形は非表示になります。

アルファ値

アルファ値は、描画の透明度を示します。0 値は描画結果が完全に透明になること、つまり何も描かれないことを意味します。IlvFullIntensity 値は、描画結果が不透明になることを意味します。

描画には、次の 2 つのオブジェクトを使用します。

- IlvPort オブジェクト。描画を実行するポート。詳細については、8 章 *描画ポート* を参照してください。
- IlvPalette オブジェクト。描画に使用するグラフィック・リソース一式。詳細については、IlvPalete を参照してください。

両方のレベルで透明度を制御することができます。たとえば、(IlvPort::setAlpha を使用して) ポートおよび描画に使用するパレットに (IlvPalette::setAlpha を使用して) アルファ値を設定します。この場合、結果として描かれる描画は 2 つのアルファ値の合成を使用します。

> ***メモ:*** *このアトリビュートは現在、GDI+ を使用した Windows プラットフォームでのみサポートされています。詳細については、281 ページの IBM ILOG Views で GDI+ 機能を使用する　を参照してください。*

アンチエイリアシング・モード

アンチエイリアシング・モードは、アンチエイリアシングを使って円滑な線の描画を行うかどうかを示します。このモードには以下の値を使用できます。

- IlvDefaultAntialiasingMode　アンチエイリアシング・モードは明示的に指定しません。これはデフォルト値を使用して継承します。
- IlvNoAntialiasingMode　アンチエイリアシングを描画に使用しません。
- IlvUseAntialiasingMode　アンチエイリアシングを使用して描画します。

アンチエイリアシング・モードはさまざまなレベルで指定できます。

- IlvDisplay　ディスプレイのデフォルト・アンチエイリアシング・モードを設定します (IlvDisplay::setAntialiasingMode)。
- IlvPort　ポート全体のアンチエイリアシング・モードを設定します (IlvPort::setAntialiasingMode)。

- IlvPalette ポート全体のアンチエイリアシング・モードを設定します (IlvPalette::setAntialiasingMode)。

以下のルールは、最終的な描画にアンチエイリアシングを使用するかどうかを決定する際に適用されます。

- パレット：
  - (メンバ関数 IlvPalette::setAntialiasingMode を使用して) パレットのアンチエイリアシング・モードが設定されている場合、このモードを使用します。
  - その他の場合、パレットには IlvDefaultAntialiasingMode が適用され、ポートのモードは描画に使うものを使用します。
- ポート：
  - (メンバ関数 IlvPort::setAntialiasingMode を使用して) ポートのアンチエイリアシング・モードが設定されている場合、このモードを使用します。
  - その他の場合、ポートには IlvDefaultAntialiasingMode が適用され、ディスプレイのモードを使用します。
- ディスプレイに関しては、デフォルトのアンチエイリアシング・モードは IlvNoAntialiasingMode です。この設定は次のいずれかの方法で変更できます。
  - メンバ関数 IlvDisplay::setAntialiasingMode を使用する。
  - Antialiasing リソースを true に設定する。
  - 環境変数 IlvAntialiasing を true に設定する。

> ***メモ:*** *このアトリビュートは現在、GDI+ を使用した Windows プラットフォームでのみサポートされています。詳細については、281 ページの IBM ILOG Views で GDI+ 機能を使用する を参照してください。*

IlvPalette: リソースのグループを使用した描画

IBM® ILOG® Views では、描画リソースは IlvPalette クラスのオブジェクトに組み込まれています。描画の唯一の方法は、IlvPalette オブジェクトを使用することです。定義済みグラフィック・オブジェクトのほとんどは、IlvPalette を 1 つ (場合によっては複数) 使用して、それ自体を描画します。

他の IlvResource サブクラスとは異なり、IlvPalette は、パブリック・コンストラクタを持っています。ただし、標準的には、IlvDisplay::getPalette メ

ソッドから取得してパレットを作成します。パブリック・コンストラクタは、共有できないパレットを作成する場合にのみ使用します。

IlvPalette を共有することも、しないこともできます。名前付きのパレットは共有パレットのサブセットです。

詳細は、以下のセクションを参照してください。

- *リソースのロックとロック解除*
- *クリッピング領域*
- *非共有パレットの作成*
- *共有パレットの作成*
- *パレットに名前を付ける*

リソースのロックとロック解除

IlvPalette は、含まれるリソースすべてをロックし、不要になったときにロック解除します (パレットの破壊またはリソースの置換)。

クリッピング領域

以下のメソッドを使用して、パレットの描画に使用するクリッピング領域を変更します。

```
void setClip(const IlvRect* = 0) const;
void setClip(const IlvRegion*) const;
```

描画メソッドを使用する場合、描画はクリッピング領域にのみ表示され、他の領域は変更されません。

このため、IlvGraphic サブクラスの draw メソッドを作成する際に、描画をクリップする必要があります。draw メソッドでは、クリッピング領域がパラメータとなります。描画に使用するパレットすべてにこのクリップを設定する必要があります。描画が完成したら、各パレットのクリップを前のクリップにリセットしてください。これはパレットが共有されているためです。これには IlvPushClip クラスを使用します。

以下に例を示します。

```
void MyGraphic::draw(IlvPort* dst,
                     const IlvTransformer* t,
                     const IlvRegion* clip) const
{
     IlvPalette* myPalette = getPalette();
     IlvPushClip (*myPalette, clip);
     IlvPoint p1(10, 10), p2(50, 50);
     // Do my drawings
     dst->drawLine(myPalette, p1, p2);
```

}

非共有パレットの作成

IlvPalette のパブリック・コンストラクタを使用して、共有できないパレットを作成できます。

```
IlvPalette(IlvDisplay* display);
IlvPalette(IlvPalette* palette);
IlvPalette(IlvDisplay* display,
           IlvColor*   background,
           IlvColor*   foreground,
           IlvFont*    font,
           IlvPattern* pattern);
```

最初のコンストラクタはデフォルトのパレットを作成し、2 つ目はその引数として与えられたパレットのコピーを作成し、3 つ目は引数として渡される特性を持つパレットを作成します。新しいパレットを作成したら、そのメンバ関数を使用して内部リソースを作成します (共有パレットでは作成しないでください)。

この技術は、パレットを共有したくない場合やパレット共有を完全に制御したい場合などの稀な場合に使用できます。パレットが不要になった場合は、削除する必要があります。この方法で作成したパレットは、共有リソース (色、フォントなど) を使用していることにご注意ください。

共有パレットの作成

各 IlvDisplay インスタンスは、共有パレットのリストを管理しています。新しいパレットが必要な場合、ディスプレイにそのサポートを要求する必要があります。このクラスは、IlvDisplay::getPalette メソッドを提供し、これを使用してパレットの内部リソースを指定できます。別のメンバ関数 IlvDisplay::getPalette(const char* name) については、名前付きのパレットに関する次のセクションで説明します。

要求に合ったパレットが既にリストにある場合は、そのパレットが返されます。共有パレットのリストに要求に合ったパレットがない場合は、新しいパレットが作成され、リストに追加され、返されます。メンバ関数 IlvDisplay::getPalette は、返されたパレットをロックしません。この関数のリソース・パラメータのいくつかは NULL に設定できます。ディスプレイは対応するデフォルトのリソースを使用します。

共有パレットの使用は、非常に一般的であり、ほとんどのアプリケーションには十分です。ただし、これらのパレットが実際に共有されており、その 1 つを変更すると好ましくない二次作用が起こる可能性が高いことに注意してください。ほとんどの場合、パレットはグラフィック・オブジェクト (つまり、IlvGraphic のサブクラス) の描画方法を制御するものです。

パレット自体は変更せず、グラフィック・オブジェクトのメンバ関数を使用してグラフィック・プロパティを変更してください。グラフィック・オブジェクトはディスプレイ用に別のパレットを取得し、アプリケーションの別の部分に使用する場合に備えて元のパレットを保持します。

以下のコードは、パレート使用法の正誤を示したものです。

```
// To set the foreground color of IlvGraphic* graphic
IlvColor* color = graphic->getDisplay()->getColor("blue");

// The following line will affect all objects sharing the palette
graphic->getPalette()->setForeground(color);  // Wrong way

// The following line will give another palette to the graphic object
// and will not affect objects pointing to the previous palette
graphic->setForeground(color);      // Right way
```

パレットに名前を付ける

他のほとんどのリソースと同様、パレットはその `IlvResource::setName` メンバ関数を使用して名前を付けることができます。この関数は既存の名前を上書きするため、パレットに名前を付ける前にパレットに既に名前が付いているかどうかを確認してください。

メンバ関数 `IlvDisplay::getPalette(const char* name)` を使用して、共有パレットを名前で検索できます。

パレットの名前は、グラフィック・オブジェクトが出力ストリームに書き込まれる際に保存されます。このデータが入力ストリームとして読み込まれる際に、ディスプレイは最初に同じ名前を持つ既存のパレットを検索します。何も見つからなければ、ディスプレイはパレットを通常の方法で読み込みます (つまり、詳細に合致する既存パレットを検索し、見つからなければ新しいパレットを作成して名前を付けます)。

IlvQuantizer: イメージ色量子化クラス

`IlvQuantizer` は、すべての色変換クラスの抽象ベース・クラスです。このクラスは、トゥルー・カラー・イメージを任意の色数のインデックス・イメージに変換します。またディザリングなどすべての IBM® ILOG® Views 量子化クラスに共通する基本的な関数を定義します。

サブクラスは、`IlvQuantizer::computeColorMap` メソッドを再定義して、適切な `IlvColorMap` を返さねばなりません。

これには以下の 2 つの主要なサブクラスのカテゴリがあります。

◆ 1 つ目のカテゴリは、固定カラーマップを使用します。

◆ 2 つ目のカテゴリは入力イメージからカラーマップを計算します。

IBM ILOG Views には現在、次の 4 つの定義済み量子化クラスがあります。

◆ The `IlvFixedQuantizer` は、トゥルー・カラー・イメージをユーザ定義のカラーマップに従ったインデックス付きカラーマップにリマップします。

◆ `IlvQuickQuantizer` は、`IlvFixedQuantizer` を、カラー・キューブに分布する定義済みカラーマップで特殊化します。赤色成分に 3 ビット、緑色成分に 3 ビット、青色成分に 2 ビットが割り当てられたマップから、256 色のマップがキューブに作成されます。

◆ `IlvNetscapeQuantizer` は、Netscape カラーマップとして知られている定義済みのカラーマップを持つ `IlvFixedQuantizer` を特殊化します。このカラーマップには 216 項目が含まれます。このカラーマップで生成されたイメージは、Netscape ウェブ・ブラウザではディザリングされないことが保証されています。

◆ `IlvWUQuantizer` は、Wu アルゴリズムを使用して入力イメージからカラーマップを計算します。このアルゴリズムは、極めて少ない色数でも非常に正確なカラーマップを生成します (量子化サンプルを参照してください)。ただし、他に比べて時間がかかります。

IBM ILOG Views には含まれていないその他の量子化方法として、Neural Nets や Octrees があります。

コード・サンプル :

```
IlvWUQuantizer quantizer;
// bdata is an instance of an IlvRGBBitmapData
IlvIndexedBitmapData* idata = quantizer.quantizer(bdata, 64);
```

章

4

グラフィック形式

IBM® ILOG® Views は、主にベクトル・エンティティを操作するツールです。ベクトル・エンティティとは、幾何学特性に基づいて視覚的側面を簡単に変更できる線や曲線で構成された形です。しかし、IBM ILOG Views では、ラスター・イメージ、すなわちビットマップ・イメージも操作できます。

- *サポートされているグラフィック形式 IBM*
- *ビットマップ*では、ビットマップ・イメージの特性を説明します。
- *IlvBitmap: ビットマップ・イメージ・クラス*
- *IlvBitmapData: ポータブル・ビットマップ・データの管理クラス*

サポートされているグラフィック形式 IBM

IBM® ILOG® Views では、以下のベクトルおよびビットマップ形式が使用できます。

- ベクトル
  - **DXF** (入出力)
  - **DCW** (入力)
  - **WMF** (Microsoft Windows のみ、出力)

- **PostScript** (出力)

◆ ビットマップ (入出力)

- **BMP** - 標準 Microsoft Windows ビットマップ形式
- **JPG** または **JPEG** - 特に写真などに多用されている形式の 1 つ
- **PNG**
- **SGI RGB** - 主に SGI プラットフォームで使用
- **TIFF** - タグの付いたイメージ形式ファイル
- **PPM** - 主に UNIX プラットフォームで使用
- **WBMP** - WAP デバイスで使用

IBM ILOG Views でのビットマップ使用についての詳細は、95 ページの *ビットマップ* および `IlvBitmap` クラスを参照してください。

ビットマップ

IBM® ILOG® Views は、ビットマップ (またはラスター) ・イメージをサポートしています。ビットマップには以下の特性があります。

カラー・ビットマップ

ディスプレイ・システムにトゥルー・カラー機能がない場合、各ピクセル値は色インデックスを表します。このピクセルに表示する正確な色を見つけるには、システムのルックアップ・テーブルを参考にします。ディスプレイ・システムにトゥルー・カラー機能がある場合、ビットマップの各ピクセルは完全な色情報を格納しています。

モノクロ・ビットマップ

イメージはモノクロにもできます。この場合、ピクセルあたり 1 ビットのみを使用します。これらの 1 ビット深度のビットマップは、「`1`」ピクセルを `IlvDisplay` インスタンスに与えられたパレットの前景色に、「`0`」ピクセルをパレットの背景色に設定して描画します。透明ビットマップとして表示される場合、「`0`」ピクセルの宛先ポートは変更されません。

透明ビットマップとマスク

カラー・ビットマップは、マスクに関連付けることができます。マスクは、実際のソース・イメージのどのピクセルが表示されるかを示すモノクロ・ビットマップです。マスクの「`0`」ビットに対応するビットマップのピクセルは表示されず、透明ビットマップとなります。透明ビットマップは、透明な部分のあるカラー・ビットマップです。

IlvBitmap: ビットマップ・イメージ・クラス

ラスター、またはビットマップ・イメージは、IlvBitmap クラスのインスタンスで表されます。IlvBitmap 使用の詳細については、以下を参照してください。

- *ビットマップに関連するメンバ関数*
- *ビットマップ形式*
- *ビットマップの読み込み：ストリーマ*
- *透明ビットマップの読み込み*

ビットマップに関連するメンバ関数

ビットマップを処理する特殊なメンバ関数は、IlvDisplay にあります。

ビットマップは多くの場合、異なったオブジェクト間で共有されます。たとえば、同じビットマップを塗りつぶしパターンとしてもそれ自体のイメージとしても使用できます。したがって、ビットマップ・リソースの管理が必要です。これはロック / ロック解除ポリシーで行います。

ビットマップ管理は、ビットマップのロックまたはロック解除方法と緊密に関係しています。永続性オブジェクトの 1 つに特定のビットマップが必要な場合は、必ず以下のメカニズムを使用して、IlvDisplay インスタンス内で確実に保護する必要があります。

```
void lock();
```

IlvBitmap クラスのこのメンバ関数は、オブジェクトが IBM ILOG Views にビットマップの変更、破壊を指示する前に、ビットマップが変更または破壊されないようにします。基本的に、この関数によって、0 に設定されている初期の参照カウントが増加します。

```
void unLock();
```

この関数によって、ビットマップのロックが解除されます。つまり、ビットマップの参照カウントが減少し、カウントが 0 に達するとそのビットマップを削除します。new および delete C++ 演算子による IlvBitmap オブジェクトの作成 / 削除メカニズムは、共有されていないときに一時的に使用されるビットマップ・オブジェクト用に保持しておきます。

ビットマップ形式

IBM ILOG Views では、さまざまな形式のイメージを含むファイルまたはストリームから IlvBitmap オブジェクトを作成できます。これらの形式は、以下のとおりです。

- **BMP (** すべてのサブタイプ、RLE 、RGB、インデックス付、トゥルー・カラー **)**。この形式は、Microsoft Windows プラットフォームでよく使用されます。未圧縮。
- ポータブル・ネットワーク・グラフィックス **(PNG)**。最近よく使用されている形式です。透明領域または色にインデックスを付け、高解像度のトゥルー・カラー・サブタイプを持たせます。

> ***メモ:*** *これは GIF に変わる形式で、特許を必要としません。*
> `http://www.libpng.org/pub/png/` *を参照してください。*

- ジョイント・フォトグラフィック・エキスパート・グループ **(JPEG)** 写真イメージ用に多く使用されている形式です。この形式は「損失が多い」、つまり元の情報が JPEC イメージで失われてしまいます。この形式は、重要な圧縮ファクタをインポートできます。
- ポータブル・ピックスマップ **(PPM**、**PGM**、**PBM)** UNIX プラットフォームでよく使用される形式です。未圧縮で、サイズの大きなファイルを生成します。
- WAP ビットマップ **(WBMP)** この形式は、携帯電話などの WAP デバイスで使用されます。これはモノクロ形式です。

ビットマップの読み込み：ストリーマ

各ビットマップ形式は、ストリーマ・オブジェクト **(** クラス `IlvBitmapStreamer`**)** と関連付けられています。

ストリーマは、コンパイル時または実行時に登録できます。ストリーマをコンパイル時に登録する場合、この形式用のヘッダー・ファイルが含まれます。

表 4.1 *ビットマップ形式のヘッダー・ファイル*

ビットマップ形式	ヘッダー・ファイル
JPEG	`ilviews/bitmaps/jpg.h`
PNG	`ilviews/bitmaps/png.h`
BMP	`ilviews/bitmaps/bmp.h`
PPM	`ilviews/bitmaps/ppm.h`
SGI RDB	`ilviews/bitmaps/rgb.h`
TIFF	`ilviews/bitmaps/tiff.h`
WBMP	`ilviews/bitmaps/wbmp.h`

次に、以下の呼び出しを使用してイメージをビットマップに読み込みます。

```
IlvBitmap* IlvDisplay::readBitmap(const char* filename);
```

イメージ・タイプはファイルの署名で認識され、IlvDisplay::readBitmap によって正しいストリーマが自動的に呼び出されます。

すべてのビットマップ・ストリーマは動的モジュールです。これは、リーダまたはライタが必要に応じて動的に読み込まれるということです。したがって、IBM ILOG Views にはイメージの読み込み / 書き込みの要求をするだけでよく、これはよく使われるすべての形式で同様です。

ストリーマはモジュールであり、不明な (または登録されていない) ファイル形式を読み込む場合は、実行時にロードできます。対応するモジュールがある場合は読み込まれ、ストリーマが登録されます。これはモジュールをサポートするプラットフォームでのみ有効です。

次の形式は常に登録されており、モジュールを必要としません。

- XPM
- XBM

透明ビットマップの読み込み

IlvTransparentIcon オブジェクトはビットマップとして表示されます。0 値のソース・ビットマップのピクセルは、描画時に宛先ポートに影響しません。通常、ビットマップ・アイコンの透明な領域によって、背景パターンが透けて表示されます。このプロセスは、透明マスクまたは透明色インデックスのあるモノクロ・ビットマップまたはカラー・ビットマップのみで有効です。

IBM ILOG Views では、次のファイル形式から透明ビットマップを読み込めます。

- XPM

  透明領域はビットマップ記述ファイルで「none」と定義されている領域に一致します。この情報が省略されると、ビットマップは透明アイコンとしては読み込まれません。

- PNG

  ILOG Views は、PNG ストリームの透明情報を使用して、ビットマップに透明領域を作成します。

IlvBitmapData: ポータブル・ビットマップ・データの管理クラス

IlvBitmapData および関連付けられたクラスは、ポータブル・ビットマップのデータ管理を提供します。詳細は、以下を参照してください。

- *IlvBitmapData クラス*

- *IlvIndexedBitmapData クラス*
- *IlvRGBBitmapData クラス*
- *IlvBWBitmapData クラス*

IlvBitmapData クラス

X11 や Windows などのディスプレイ・システムのラスター・イメージは、通常は極めてシステムへの依存度が高い形で表示されます。これらの表示は、ディスプレイ・システム設定により大きく異なるため、ディスプレイの奥行きに依存するコードを作成する必要があります。`IlvBitmapData` クラスは、一般的なポータブル API を使用してラスター・イメージを記述できます。`IlvBitmapData` は、3 つのサブクラスのベース・クラスです。これらのサブクラスはインデックス付きのイメージ、アルファ・チャネル付きのトゥルー・カラー・イメージ、マスキングおよびクリッピングによく使用されるモノクロ・イメージを管理します。ビットマップ・データはリソースと同様に管理され、ロックおよびロック解除が可能です。このクラスは、通常、直接使用することはありません。`IlvBitmapData` クラスはメモリ、参照カウント、イメージのピクセルへの一般的なアクセスを管理します。また、ストレッチなどの基本的なイメージ処理メソッドへのアクセスも提供します。IBM® ILOG® Views ではさらに、トゥルー・イメージをインデックス付きイメージに変換する機能も備えています。これは量子化と呼ばれるプロセスです (91 ページの *IlvQuantizer: イメージ色量子化クラス* を参照してください)。また、完全な SVG 仕様フィルタへもアクセスでき、非常に高度なイメージ処理機能が利用できます (5 章を参照してください)。

IlvIndexedBitmapData クラス

`IlvIndexedBitmapData` クラスは、インデックス付のカラー・イメージ専用で、ラスター・データはカラー・マップのインデックスとして記述されます (8 ビット値、つまりインデックス付きのビットマップ・データでは 256 色しか使用できません)。

256 x 256 ピクセルのインデックス付きビットマップ・データの作成

最初に、カラーマップを作成します。ここでは 256 項目のカラーマップを作成します。次に、このカラーマップにグレイスケール値を追加します。各コンポーネントは、8 ビットで記述します。

```
IlvColorMap* cmap = new IlvColorMap(256);
for (IlUInt idx = 0; idx < 256; ++idx) {
   // sets the red, green and blue components for a entry
   cmap->setEntry(idx, idx, idx, idx);
}
```

次に、目的のサイズのインデックス付きビットマップ・データを作成します。

```
IlvIndexedBitmapData* idata = new IlvIndexedBitmapData(256, 256, cmap);
```

ビットマップ・データにインデックスの階調を追加します。

```
for (IlUInt h = 0; h < 256; ++h)
   for (IlUInt w = 0; w < 256; ++w)
      idata->setPixel(w, h, h);
```

このビットマップ・データを画面で表示するには、このデータから IlvBitmap を作成する必要があります。

```
IlvBitmap* bitmap = new IlvBitmap(display, idata);
```

これで IlvIcon クラスを使用して、グラフィック・オブジェクトを作成できます。

IlvRGBBitmapData クラス

IlvRGBBitmapData は、トゥルー・カラー・イメージ専用です。このイメージでは、ラスター・データはピクセルの色でそのまま表示されます。

256 x 256 ピクセルのトゥルー・カラー・ビットマップ・データの作成と階調の追加

```
IlvRGBBitmapData* bdata = new IlvRGBBitmapData(256, 256);
for (IlUInt h = 0; h < 256; ++h)
  for (IlUInt w = 0; w < 256; ++w)
    bdata->fastSetRGBPixel(w, h, w, h, w);
```

IlvIndexedBitmapData と同様に、IlvBitmap を作成し、IBM ILOG Views 標準メソッドを使用して表示できます。

8 ビットのカラー・ディスプレイの場合、IBM® ILOG® Views ライブラリは、極めて高品質のイメージを生成するアルゴリズムを使用して、自動的にこのトゥルー・カラー・イメージをインデックス付きのイメージに変換します。

トゥルー・カラー・ビットマップ・データの内部表現は、幅 x 高さのエントリ配列です。各エントリは、次のようにピクセルを記述する 4 バイトで構成されます。

- 最初のバイトは、アルファ・コンポーネントです。
- 次のバイトは、赤コンポーネントです。
- 3 つめのバイトは、緑コンポーネントです。
- 4 つめのバイトは、青コンポーネントです。

配列は上から下または下から上に記述できるため、IlvBitmapData::getRowStartData を使用したライン・アクセス・メソッドを使用できます。

ラスター・データへのアクセスには、以下のようにいくつかのメソッドを利用できます。

- IlvBitmapData::getData は、生ラスター・データへポインタを返します。

- IlvRGBBitmapData::getRGBPixel は、特定のピクセル値を取得できます。
- IlvRGBBitmapData::getRGBPixels は、特定の矩形を表現する RGB を取得します。
- IlvRGBBitmapData::fill は、特定の色で矩形を塗りつぶします。
- IlvRGBBitmapData::copy は、ビットマップ・データの矩形を他のビットマップ・データの特定の位置にコピーします。
- IlvRGBBitmapData::blend は、ブレンド係数を使用して、ビットマップ・データを別のビットマップ・データにスムーズにブレンドします。
- IlvRGBBitmapData::alphaCompose はアルファ・チャネルを使用して、2 つのビットマップ・データを構成します。
- IlvRGBBitmapData::tile は、ビットマップ・データを別のビットマップ・データと並べて表示します。
- IlvRGBBitmapData::stretch は、ビットマップ・データの一部を別のビットマップ・データに伸張します。
- IlvRGBBitmapData::stretchSmooth は、極めて高度なリサンプリング・メソッドを使用して、ビットマップ・データの一部を別のビットマップ・データに伸張します。

特定のピクセルの色およびアルファ値に個々にアクセスすることもできます。

IlvBWBitmapData クラス

このクラスは、モノクロ・イメージ専用で、特定のピクセルに対してオンとオフの 2 つの値のみ利用可能です。

章

5

イメージ処理フィルタ

この章では、IBM® ILOG® Views が提供するさまざまなイメージ処理クラスについて説明します。

これらのクラスはすべて、SVG フィルタと関連しています (この機能についての詳細は、`http://www.w3.org/TR/2000/CR-SVG-20001102/filters.html` を参照してください)。

IlvBitmapFilter: イメージ処理クラス

`IlvBitmapFilter` は、IBM® ILOG® Views のイメージ処理クラスすべてのベース・クラスです。これは、次の単一メソッドを使用してイメージ処理クラスのインターフェースを定義します。

```
IlvBitmapFilter::apply
```

このメソッドは、`IlvBitmapData` の配列を受け取り、別の `IlvBitmapData` を返します。

IBM ILOG Views Foundation パッケージの `ilvbmpflt` ライブラリは、`IlvBitmapFilter` のサブクラスの多くを定義します。このほとんどは W3C の SVG フィルタ仕様の実装です。次のセクションでは、さまざまなイメージ処理クラスとその機能の説明をします。

- *IlvBlendFilter クラス*
- *IlvColorMatrixFilter クラス*
- *IlvComponentTransferFilter クラス*
- *IlvComposeFilter クラス*
- *IlvConvolutionFilter クラス*
- *IlvDisplaceFilter クラス*
- *IlvFloodFilter クラス*
- *IlvGaussianBlurFilter クラス*
- *IlvImageFilter クラス*
- *IlvLightingFilter クラス*
- *IlvLightSource クラス*
- *IlvMergeFilter クラス*
- *IlvMorphologyFilter クラス*
- *IlvOffsetFilter クラス*
- *IlvTileFilter クラス*
- *IlvTurbulenceFilter クラス*
- *IlvFilterFlow クラス*
- *IlvFilteredGraphic を使用してフィルタ・フローをグラフィック・オブジェクトに適用する*

IlvBlendFilter クラス

`IlvBlendFilter` クラスを使うと、さまざまなモードを使用して、2 つのイメージ A と B を 1 つに融合できます。

ブレンド・モードは以下の数式を定義します。

- 標準ブレンド・モード： `cr = (1 - qa) * cb + ca`
- 乗算ブレンド・モード： `cr = (1 - qa) * cb + (1 - qb) * ca + ca * cb`
- 画面ブレンド・モード： `cr = cb + ca - ca * cb`
- ダーク・ブレンド・モード: `cr = Min((1 - qa) * cb + ca, (1 - qb) * ca + cb)`
- ライト・ブレンド・モード: `cr = Max((1 - qa) * cb + ca, (1 - qb) * ca + cb)`

ここで:

cr	結果としてできる色 (RGB) ? 乗算済み
qa	イメージ A の任意のピクセルの不透明値
qb	イメージ B の任意のピクセルの不透明値
ca	イメージ A の任意のピクセルの色 (RGB) (乗算済み)
cb	イメージ B の任意のピクセルの色 (RGB) (乗算済み)

すべてのブレンド・モードでは、結果として生じる不透明度 qr は次のように計算されます。

```
qr = 1 - (1 - qa) * (1 - qb)
```

IlvColorMatrixFilter クラス

IlvColorMatrixFilter クラスを使うと、入力イメージの RGBA コンポーネントにマトリックス変換を適用できます。

マトリックスは、5 x 4 行の主な順位係数になっています。

```
| R' |     | a00 a01 a02 a03 a04 |   | R |
| G' |     | a10 a11 a12 a13 a14 |   | G |
| B' |  =  | a20 a21 a22 a23 a24 | * | B |
| A' |     | a30 a31 a32 a33 a34 |   | A |
| 1  |     |  0   0   0   0   1  |   | 1 |
```

このクラスには、特定の係数を持つ 3 つのサブクラスがあります。

IlvSaturationFilter クラス

IlvSaturationFilter は、次の数式から変換マトリックスを計算します。

```
| R' |     |0.213+0.787s  0.715-0.715s  0.072-0.072s 0  0 |   | R |
| G' |     |0.213-0.213s  0.715+0.285s  0.072-0.072s 0  0 |   | G |
| B' |  =  |0.213-0.213s  0.715-0.715s  0.072+0.928s 0  0 | * | B |
| A' |     |          0           0            0  1  0 |   | A |
| 1  |     |          0           0            0  0  1 |   | 1 |
```

ここで、s は彩度です。

IlvHueRotateFilter クラス

IlvHueRotateFilter は、次の数式から変換マトリックスを計算します。

```
| R' |     | a00  a01  a02  0  0 |   | R |
| G' |     | a10  a11  a12  0  0 |   | G |
| B' |  =  | a20  a21  a22  0  0 | * | B |
| A' |     | 0    0    0    1  0 |   | A |
| 1  |     | 0    0    0    0  1 |   | 1 |
```

ここで、a00、a01 などは、次のように計算されます。

```
     | a01 a01 a02 |    [+0.213 +0.715 +0.072]
     | a10 a11 a12 | =  [+0.213 +0.715 +0.072] +
     | a20 a21 a22 |    [+0.213 +0.715 +0.072]

                           [+0.787 -0.715 -0.072]
    cos(hueRotate value) *  [-0.212 +0.285 -0.072] +
                           [-0.213 -0.715 +0.928]

                           [-0.213 -0.715 +0.928]
    sin(hueRotate value) *  [+0.143 +0.140 -0.283]
                           [-0.787 +0.715 +0.072]
```

ここで、value は色調の回転角です。

したがって、色調マトリックスの左上の項は、次のようになります。

```
.213 + cos(hueRotate value)*.787 - sin(hueRotate value)*.213
```

IlvLuminanceToAlphaFilter クラス

IlvLuminanceToAlphaFilter は、次の数式から変換マトリックスを計算します。

```
| R' |     |      0        0        0  0  0 |   | R |
| G' |     |      0        0        0  0  0 |   | G |
| B' |  =  |      0        0        0  0  0 | * | B |
| A' |     | 0.2125   0.7154   0.0721  0  0 |   | A |
| 1  |     |      0        0        0  0  1 |   | 1 |
```

このフィルタは、色のイメージをグレイスケール・イメージに変換します。

IlvComponentTransferFilter クラス

IlvComponentTransferFilter クラスを使うと、次のようにイメージでコンポーネントのリマップができます。

```
R' = feFuncR( R )
G' = feFuncG( G )
B' = feFuncB( B )
A' = feFuncA( A )
```

feFuncR、feFuncG、feFuncB、および feFuncA は、各コンポーネントの変換関数を定義します。

これにより、明るさ調整、コントラスト調整、色バランス、閾値などの処理が利用できます。

次の 5 つの変換関数が定義されています。

- 同一：`C' = C`
- テーブル：関数はアトリビュート値によるルックアップ・テーブルへのリニア補間で定義されます。これは n 補間の値を求めるために、n+1 値 (すなわち v0 から vn) のリストを提供します。補間には、次の式を使います。

  ```
  k/N <= C < (k+1)/N => C' = vk + (C - k/N)*N * (vk+1 - vk)
  ```

- 離散：この関数は、アトリビュート値により定義されるステップ関数で定義され、これは n ステップから成るステップ関数を識別するために、n 値 (すなわち、v0 から vn-1) のリストを提供します。このステップ関数は、以下の式によって定義されます。

  ```
  k/N <= C < (k+1)/N => C' = vk
  ```

- 線形：この関数は、以下の一次方程式によって定義されます。

  ```
  C' = slope * C + intercept
  ```

  ここで、slope および intercept はユーザ定義です。

- ガンマ：この関数は、以下の指数関数によって定義されます。

  ```
  C' = amplitude * pow(C, exponent) + offset
  ```

  ここで、amplitude、exponent、offset はユーザ定義です。

変換関数はそれぞれのクラスを持ち、再定義が可能です (IlvTransferFunction、IlvIdentityTransfer、IlvLinearTransfer、IlvTableTransfer、IlvDiscreteTransfer、IlvGammaTransfer を参照してください)。

IlvComposeFilter クラス

IlvComposeFilter クラスを使うと、イメージ・スペースにおいて、次の Porter-Duff 合成の 1 つを使用して、ピクセル単位で 2 つの入力イメージを組み合わせることができます。これらの Porter-Duff 合成は、over、in、atop、out、xor です。また、コンポーネント単位の計算ができます (結果は [0..1] の間に正規化されます)。ページ 107 の ï 5.1 に示されるように、これら 6 つの演算子を使用して合成することができます。

結果としてできる色は、次の式から得られます。

```
Cresult = Fa * Ca + Fb * Cb
```

ここで :

- Fa および Fb は、ページ 107 の ï 5.1 に示されるように、演算子により異なります。
- Ca は最初のイメージの色で、Cb は 2 つ目のイメージの色です。
- 表中で、Aa は最初のイメージのアルファ値で、Ab は 2 つ目のイメージのアルファ値です。

表 5.1 *合成演算子*

演算子	演算
over	Fa = 1, Fb = 1 - Aa
in	Fa = Ab, Fb = 0
out	Fa = 1 - Ab, Fb = 0
atop	Fa = Ab, Fb = 1 - Aa
xor	1 - Ab, Fb = a - Aa
計算	Cresult = k1 * Ca * Cb + k2 * Ca + k3 * Ca * Cb + k4

IlvConvolutionFilter クラス

IlvConvolutionFilter クラスを使用すると、マトリックス畳み込みフィルタ効果を適用できます。畳み込みは、入力イメージのピクセルを隣接するピクセルと結合して、新しいイメージを作ります。畳み込みを通じて、ぼかし、エッジ検出、シャープ化、エンボス効果、面取りなど、さまざまな処理をイメージに施すことができます。

畳み込み行列は、n x m の行列 (畳み込みカーネル) に基づき、入力イメージの任意のピクセル値が、隣接するピクセル値とどのように結合して新しいピクセル値を生成しているのかを示します。結果としてできるピクセルは、カーネル・マトリックスを対応するソース・ピクセルおよび隣接するピクセルに適用して決定されます。

これを説明するために、次のような色値を持つ 5 x 5 ピクセルの入力イメージがあると想定します。

```
  0  20  40 235 235

100 120 140 235 235

200 220 240 235 235

225 225 255 255 255

225 225 255 255 255
```

また、3 x 3 畳み込みカーネルを次のように定義するとします。

```
 1 2 3

4 5 6

7 8 9
```

イメージの 2 行目および 2 列目のピクセルに注目してみましょう (ソース・ピクセル値は 120)。新しいピクセル値は、次のようになります。

```
(1*  0 + 2* 20 + 3* 40 +

4*100 + 5*120 + 6*140 +

7*200 + 8*220 + 9*240) / (1+2+3+4+5+6+7+8+9)
```

除数 (マトリックス畳み込みを除算した結果) およびバイアス (マトリックス畳み込みを加算した結果) を指定することができます。

カーネルは、`IlvBitmapDataKernel` クラスで定義されています。

IlvDisplaceFilter クラス

`IlvDisplaceFilter` クラスは、別のイメージのピクセル値を使用してあるイメージのピクセルを変位させます。

次のような変換が実行されます。

```
P'(x,y) <- P( x + scale * ((XC(x,y) - .5), y + scale * (YC(x,y) -
.5))
```

ここで:

- `P(x,y)` は入力イメージです。

- P'(x,y) はターゲットです。
- XC(x,y) および YC(x,y) は、変位マップのコンポーネント値です。これらはイメージ・マップの色コンポーネントから自由に選択できます (たとえば、赤コンポーネントを X に、アルファ・コンポーネントを Y に変位できます)。
- scale は、ユーザ定義のスケール値です。

IlvFloodFilter クラス

IlvFloodFilter クラスは、イメージを任意の色で塗りつぶします。

IlvGaussianBlurFilter クラス

IlvGaussianBlurFilter クラスは、イメージにガウスぼかし効果を適用します。ガウスぼかしカーネルは、標準の畳み込みに似ています。

```
H(x) = exp(-x^2/ (2*s^2)) / sqrt(2* pi*s^2)
```

ここで、s は、ユーザ定義の偏差です。

このフィルタは、IlvConvolutionFilter を内部的に使用します。

IlvImageFilter クラス

IlvImageFilter クラスは、イメージ名を記述する文字列からイメージを読み込みます。

IlvLightingFilter クラス

IlvLightingFilter クラスは、アルファ・チャネルをバンプ・マップとして使用してイメージを明るくします。明るさのタイプはいくつか指定できます (後の説明を参照)。

このクラスは抽象クラスで、使用可能なサブクラスが 2 つあります。

IlvDiffuseLightingFilter クラス

IlvDiffuseLightingFilter クラスでは、新しいイメージは光源色に基づいた不透明な RGBA イメージで、アルファ値はどこでも 1.0 になります。明るさの計算は、Phong ライト・モデルの標準散光コンポーネントに従っています。新しいイメージは、光源色、光源位置、入力バンプ・マップの平面ジオメトリに基づきます。

このフィルタ・プリミティブで作成されたライト・マップは、IlvComposeFilter 合成計算メソッドの乗算項を使用してテキスチャ・イメージと組み合わせることができます。これをテキスチャ・イメージに適用する前にこれらのライト・マップのいくつかをまとめて追加することにより、複数の光源をシミュレートできます。

結果としてできる RGBA イメージは、次のように計算されます。

```
Dr = kd * N.L * Lr
Dg = kd * N.L * Lg
Db = kd * N.L * Lb
Da = 1.0
```

ここで:

kd	散光定数
N	平面の通常の単位ベクトル、x と y の関数 (下記を参照)
L	平面から光源までを示す単位ベクトル、点とスポット・ライトの場合は x と y の関数
Lr, Lg, Lb	光源の RGB コンポーネント、スポット・ライトの場合は x と y の関数

N は x と y の関数で、次のように平面の階調によって異なります。

入力アルファ・イメージ Ain (x,y) で表される平面は、次のとおりです。

```
Z (x,y) = surfaceScale * Ain (x,y)
```

通常の平面は、Sobel 階調 3x3 フィルタを使って次のように計算されます。

```
Nx (x,y)= - surfaceScale * 1/4*(( I(x+1,y-1) + 2*I(x+1,y) + I(x+1,y+1))
                                - (I(x-1,y-1) + 2*I(x-1,y) + I(x-1,y+1)))

Ny (x,y)= - surfaceScale * 1/4*(( I(x-1,y+1) + 2*I(x,y+1) + I(x+1,y+1))
                                - (I(x-1,y-1) + 2*I(x,y-1) + I(x+1,y-1)))

Nz (x,y) = 1.0

N = (Nx, Ny, Nz) / Norm((Nx,Ny,Nz))
```

イメージ・サンプルから光源までの単位ベクトルである L の詳細については、IlvLightSource を参照してください。

IlvSpecularLightingFilter クラス

IlvSpecularLightingFilter では、新しいイメージは光源色に基づく RGBA イメージになります。明るさの計算は、Phong ライト・モデルの標準スペキュラ・コンポーネントに従っています。新しいイメージは、光源色、光源位置、入力バンプ・マップの平面ジオメトリに基づきます。明るさの計算の結果が追加されます。フィルタのプリミティブは、視点が限りなく z 方向であると仮定します (すなわち、視点方向の単位ベクトルはどこでも (0,0,1) となっています)。

このフィルタ・プリミティブは、ライティング計算のスペキュラ・リフレクション部分を含むイメージを生成します。このようなマップは、IlvComposeFilter 計算メソッドの加算項を使用したテキスチャと組み合わされます。これをテキス

チャ・イメージに適用する前にこれらのライト・マップのいくつかを追加することにより、複数の光源をシミュレートできます。

結果としてできる RGBA イメージは、次のように計算されます。

```
Sr = ks * pow(N.H, specularExponent) * Lr
Sg = ks * pow(N.H, specularExponent) * Lg
Sb = ks * pow(N.H, specularExponent) * Lb
Sa = max(Sr, Sg, Sb)
```

ここで:

ks	スペキュラ・ライティング定数
N	平面の通常の単位ベクトル、x と y の関数 (下記を参照)
H	視点の単位ベクトルと光源の単位ベクトルの間の「中間」単位ベクトル
Lr, Lg, Lb	光源の RGB コンポーネント

N および (Lr, Lg, Lb) の定義については、IlvDiffuseLightingFilter を参照してください。

H の定義は、一定の視点ベクトルのここでの仮定を次のように反映しています。E = (0,0,1):

```
H = (L + E) / Norm(L+E)
```

ここで、L は光源の単位ベクトルです。

IlvLightSource クラス

IlvLightSource クラスは、光源をモデル化します。これには使用可能なサブクラスが 3 つあります。

IlvDistantLight クラス

IlvDistantLight は、方位と標高を使って無限光源をモデル化します。

```
Lx = cos(azimuth)*cos(elevation)
Ly = sin(azimuth)*cos(elevation)
Lz = sin(elevation)
```

IlvPointLight クラス

IlvPointLight は、Lightx、Lighty、Lightz の 3 つの座標を使用して位置を設定できる光源をモデル化します。

IlvSpotLight クラス

IlvSpotLight は、Lightx、Lighty、Lightz の 3 つの座標を使用して位置を設定できるスポット・ライト光源をモデル化します。

```
Lx = Lightx - x
Ly = Lighty - y
Lz = Lightz - Z(x,y)
L = (Lx, Ly, Lz) / Norm(Lx, Ly, Lz)
```

ここで:

Lightx, Lighty, および Lightz	入射光源の位置
Lr, Lg, Lb	光源色のベクトルは、スポット・ライトの場合のみの位置関数です。 Lr = Lightr*pow((-L.S),specularExponent) Lg = Lightg*pow((-L.S),specularExponent) Lb = Lightb*pow((-L.S),specularExponent)

S を、x-y 平面の光源から点 (pointsAtX、pointsAtY、pointsAtZ) を指す単位ベクトルとすると、以下のようになります。

```
Sx = pointsAtX - Lightx
Sy = pointsAtY - Lighty
Sz = pointsAtZ - Lightz
S = (Sx, Sy, Sz) / Norm(Sx, Sy, Sz)
```

IlvMergeFilter クラス

IlvMergeFilter クラスは、over 演算子を使用して入力イメージ・レイヤを 1 つずつ重ねて作成します。

多くの効果は、最終的な出力イメージを作成するために、いくつかの中間レイヤを生成します。このフィルタは、これらを 1 つのイメージに折りたたみます。これは n-1 IlvComposeFilter フィルタを使用することも可能ですが、この一般的な処理をこの形で使えるようにしておいた方が便利で、実装もより柔軟になります。

IlvMorphologyFilter クラス

IlvMorphologyFilter は、イメージを「膨張」または「収縮」させることができます。これは特に、アルファ・チャネルの膨張または収縮に便利です。

拡張 (または縮小) カーネルは、幅 2*x-radius+1、高さ 2*y-radius+1 の矩形で、radius はユーザ定義の値です。拡張では、出力ピクセルは入力イメージのカーネ

ル矩形で、R、G、B、A 値に対応するそれぞれのコンポーネントの最大値です。縮小では、出力ピクセルは入力イメージのカーネル矩形で、R、G、B、A 値に対応するそれぞれのコンポーネントの最小値です。

IlvOffsetFilter クラス

`IlvOffsetFilter` クラスは、指定した x-y 値にイメージをオフセットします。これは、下に影をつけるなどの効果で重要です。

IlvTileFilter クラス

`IlvTileFilter` クラスは、タイル・パターンを繰り返すターゲット・イメージを作ります。

IlvTurbulenceFilter クラス

`IlvTurbulenceFilter` は、Perlin タービュランス関数を使用してイメージを作成します。雲や大理石などの人工のテキスチャを合成できます。

> ***メモ:*** *Perlin タービュランス関数の詳細については、Ebert 他著『Texturing and Modeling』(AP Professional、1994 年) を参照してください。*

新しいイメージは、イメージ・スペース最大のサイズになります。

1 オクターブだけを合成して、帯域の限られたノイズを作成できます。

フラクタル・ノイズまたはタービュランスを作成するかどうかを、および使用するノイズ生成関数の反復 (オクターブ) 数を選択できます。

IlvFilterFlow クラス

`IlvFilterFlow` クラスは、名前を入出力として使用して、`IlvBitmapFilter` インスタンスを連結します。このフローの例として、下に影を付ける効果の作成があります。イメージのアルファ・チャネルをガウスぼかし用の入力として使用し、ぼかしたイメージをオフセットして元のイメージとマージします。

IlvFilterFlow クラスはプログラムで作成できますが、フィルタ・フローの XML 表現を使用したほうが便利です。これは SVG でのフィルタ定義方法と似ています。

以下の XML ファイル は、このフローの例です ($ILVHOME/data/filters には、多数の定義済み XML フィルタ・フローがあります)。

```
<?xml version="1.0"?>
<ÉtÉBÉãÉ^>
  <filter id="DropShadow2" x="-10" y="-10" width="125" height="125">
    <desc>Applies a drop shadow effect</desc>
    <feGaussianBlur in="SourceAlpha" stdDeviation="3"/>
    <feOffset dx="2" dy="2" result="offsetBlur"/>
    <feComposite in="SourceGraphic" in2="offsetBlur" operator="over"/>
  </filter>
</filters>
```

以下は各行の説明です。

```
<?xml version="1.0"?>
```

IBM ILOG Views フィルタの定義は、XML 規則に従っています。

```
<ÉtÉBÉãÉ^>
```

フィルタの要素を開きます (フィルタはいくつでも含めることができます)。

```
<filter id="DropShadow2" x="-10" y="-10" width="125" height="125">
```

フィルタ名 DropShadow2 の要素を開きます。

いくつかのフィルタは、ソース・イメージをすべての方向に何ピクセルか拡張しています。したがって、フィルタがどれだけ拡張するかを指定する必要があります。

このフィルタは、ソース・イメージを左と上に 10 ピクセル、幅と高さを 25 ピクセル拡張します。

```
<desc>Applies a drop shadow effect</desc>
```

このタグはフィルタの説明です。

```
<feGaussianBlur in="SourceAlpha" stdDeviation="3"/>
```

このフローで使用する最初の構成要素フィルタは、IlvGaussianBlurFilter で、両方向の偏差は 3 です。

フィルタ・フローによって次の 2 つの名前が定義されます。

- SourceAlpha: 入力イメージのアルファ値のみを含みます。
- SourceGraphic: 入力イメージを含みます。

ここで必要なのはイメージのアルファ・コンポーネントにぼかしを適用することだけです。

```
<feOffset dx="2" dy="2" result="offsetBlur"/>
```

2 つ目の構成要素フィルタは IlvOffsetFilter で、変位は 2 です。

何も指定されていない場合、フィルタからの入力は前のフィルタの出力となるため指定する必要はありません。結果として、offsetBlur という名前で格納されるイメージができます。

```
<feComposite in="SourceGraphic" in2="offsetBlur" operator="over"/>
```

3 つ目、そして最後の構成要素フィルタは、IlvComposeFilter で、over 演算子を使用して入力イメージでぼかしたイメージのオフセットを作成します。

```
</filter>
```

フィルタ・フロー記述を終了します。

```
</filters>
```

フィルタ列挙を終了します。

このような IlvFilterFlow は、次の行で作成できます。ここではフィルタ・フローを含むファイルが、standard.xml という名前でディスクに保存されていると想定します。

```
IlIUrlStream input("standard.xml");
IlvFilterFlow* flow = new IlvFilterFlow(input, "DropShadow2");
```

IlvFilteredGraphic を使用してフィルタ・フローをグラフィック・オブジェクトに適用する

IBM® ILOG® Views Foundation パッケージでは、フィルタ・フローをグラフィック・オブジェクトに適用する簡単な方法を用意しています。IlvFilteredGraphic クラス

このクラスはグラフィック・オブジェクトをカプセル化し、オブジェクトの draw メソッドから IlvBitmapData を内部的に計算します。次に任意のフィルタ・フローをこの IlvBitmapData に適用して、画面にその結果を描画します。こうするとベクトル・オブジェクトにイメージ処理効果を簡単に加えられます。

コード・サンプル：

```
IlvZoomableLabel* embosssource = new IlvZoomableLabel(display,
                                                      IlvPoint(100, 100),
                                                      "Views");

    embosssource->setForeground(display->getColor((IlvIntensity)(5 * 255),
                                                  (IlvIntensity)(5 * 255),
                                                  (IlvIntensity)(56 * 255)));
    embosssource->setFont(display->getFont("%Helvetica-75-"));
    IlvFilteredGraphic* emboss = new IlvFilteredGraphic(display,
                                                        embosssource,
                                                        "standard.xml#DropShadow",
```

```
                                                    IlTrue);
```

IBM ILOG Views 配布の `$ILVHOME/data/filters` ディレクトリには、多くの定義済みフィルタ・フローが用意されています。これらを使用して IBM ILOG Views Studio アプリケーションと対話ができます。

章

6

ディスプレイ・システム

`IlvDisplay` クラスは、IBM® ILOG® Views ライブラリの基本クラスです。このクラスは、ディスプレイ・システムに関するすべての側面を処理します。本セクションは、次のような構成になっています。

- *IlvDisplay: ディスプレイ・システム・クラス*
- *ディスプレイ・サーバとの接続*
- *ディスプレイ・システム・リソース*
- *Home*
- *ディスプレイ・パス*

IlvDisplay: ディスプレイ・システム・クラス

IBM® ILOG® Views を使用してグラフィック・アプリケーションを開発するには、`IlvDisplay` メンバ関数一式、すなわち IBM ILOG Views *プリミティブ*を使用します。

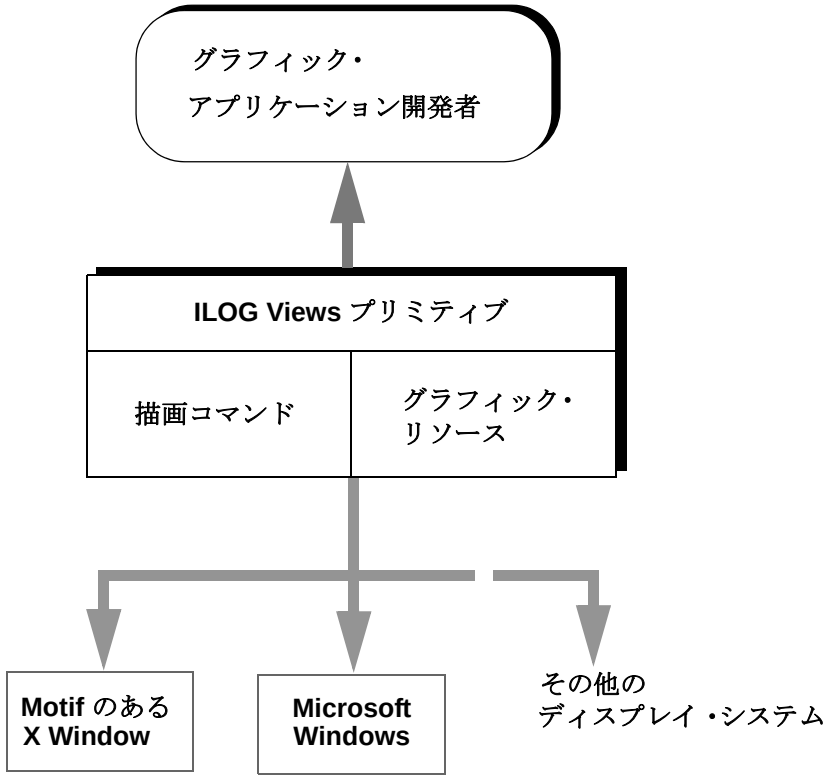


図 6.0 *IlvDisplay 描画メンバ関数 プリミティブ*

`IlvDisplay` クラスを使うと、X Window や Microsoft Windows などのディスプレイ・システムと透過的に通信できます。

2 つの基本タスクは、描画コマンドとグラフィック・リソース処理です。

- **描画コマンド**　描画コマンドは、点、矩形、領域 (矩形の羅列)、曲線、文字列などエンティティの基本的な幾何学クラスを処理します。

  この種類の描画メンバ関数は、20 以上あります (詳細については、`IlvPort` クラスを参照してください)。描画処理は、`IlvPort` クラスのインスタンスとして定義される領域内にその結果を生成します。これはメモリ内または画面上のいずれかです。

- **グラフィック・リソース**　いくつかの IlvDisplay 描画メンバ関数は、色、線の種類、パターン、フォントなどのグラフィック・リソースを処理します。これらのリソースは、IlvResource と呼ばれるクラスの機能を継承します。これらはさまざまな IlvDisplay メンバ関数を使って作成されます。特定のリソースは、IlvPalette クラスのオブジェクトにグループ化され、描画に使われます。

ディスプレイ・サーバとの接続

IBM® ILOG® Views セッションを初期化するには、IlvDisplay クラスのインスタンスを作成する必要があります。この手順は以下のとおりです。

- *接続を開いてディスプレイを確認する*
- *接続を閉じてセッションを終了する*

接続を開いてディスプレイを確認する

メンバ関数 IlvDisplay::isBad はブール型値を返します。これは以下のコードに示されるように、IlvDisplay オブジェクトが問題なく作成されたかどうかを知らせます。ディスプレイ・エラーの原因は、ディスプレイ・システムにより異なります。

```
IlvDisplay* display = new IlvDisplay("AppName",
                                     "DisplayName",
                                     argc,
                                     argv);
if (display->isBad()) {
   delete display;
   IlvFatalError("Could not create display");
   IlvExit(-1);
}

const char* dirName  = "./localDirectory/subDirectory";
const char* fileName = "foo.txt";
display->prependToPath(dirName);
// Now, if a file such as
//    "./localDirectory/subDirectory/foo.txt"
// or
//    ".\localDirectory\subDirectory\foo.txt"
// exists, we should be able to find it.
const char* filePath = display->findInPath(fileName);
if (filePath)
    IlvPrint("File %s found at %s", fileName, filePath);
else
    IlvWarning("File %s not found", fileName);
```

IBM ILOG Views エラー・メッセージの詳細については、付録 E、*エラー・メッセージ*を参照してください。

接続を閉じてセッションを終了する

ディスプレイ・サーバへの接続を閉じるには、IlvDisplay オブジェクトを破壊します。デストラクタである IlvDisplay::~IlvDisplay は、このディスプレイが使用しているすべてのグラフィック・リソースを直ちに解放します。

ディスプレイ・システムへの既存のリンクに基づいて IlvDisplay が作成された場合、delete はこのリンクを破壊しません。マルチディスプレイ・アプリケーションの稀な場合を除いて、IlvDisplay の破壊とはセッションの終了を意味します。これはディスプレイなしではほとんど何もできないためです。

セッションを正しく終了するには、IlvExit を呼び出す必要があります。これにより IBM ILOG Views が割り当てたメモリが解放されます。これは Microsoft Windows を使用している場合に特に重要です。このメモリはシステムによって自動的に解放されないためです。

```
delete display;
IlvExit(0);
```

ディスプレイ・システム・リソース

ディスプレイ・システム・リソースは、*2* つの文字列と関連付けられています。つまり、name と value です。ディスプレイ・システム・リソースは、カスタマイズ可能なアプリケーションの構築に非常に便利です。

> ***メモ:*** *「ディスプレイ・システム」リソースは「グラフィック」リソースとは異なるものです。グラフィック・リソースについては、3 章 グラフィック・リソースを参照してください。*

リソース名とリソース値の対は、以下のようにリソース・ファイルの特定のセクションで指定できます (たとえば、UNIX では .Xdefaults ファイル、PC では .INI ファイル)。

```
[MyApplication]
view.background=green
label.txt=This is my contents
```

リソース名に関連付けられている値は、エンドユーザがランタイムに変更できます。

> ***メモ:*** *リソース・ファイルには、[IlogViews] セクションがあり、これはすべての IBM ILOG Views アプリケーションに共通しています。*

ディスプレイ・システム・リソースについての詳細は、以下のトピックを参照してください。

- *getResource メソッド*
- *ディスプレイ・システム・リソースの格納方法*
- *デフォルトのディスプレイ・システム・リソース*
- *環境変数およびリソース名*
- *Windows のディスプレイ・システム・リソース*

getResource メソッド

アプリケーションは、IlvDisplay::getResource メソッドを使用してディスプレイ・システムからリソース値を取得します。

```
const char* res = display->getResource("resourceName", default);
```

IlvDisplay::getResource メソッドは、現在の IBM ILOG Views セッションのアプリケーション名に関連付けられている値文字列を返します。これは IlvDisplay コンストラクタおよびリソース名を表す文字列で指定されています。指定した文字列に一致するリソースがない場合、このメンバ関数は、オプションの default 文字列パラメータに使用されているデフォルト値を戻します。
IlvDisplay::getResource が返す唯一のタイプは、const char* です。文字列を別のデータ・タイプに変換するかどうかは、アプリケーションのプログラマ次第ということになります。IlvDisplay::getResource 呼び出しの結果が格納されるメモリ中の場所は、関数を呼び出すたびに使用されます。したがって、前回の結果は上書きされます。結果を保存する場合は、すぐにコピーする必要があります。

ディスプレイ・システム・リソースの格納方法

リソースがディスプレイ・システム設定ファイル内に格納される方法は、システムに依存しています。

- Microsoft Windows では、以下の行を VIEWS.INI ファイルまたはアプリケーションに依存している .INI ファイルに追加します。(VIEWS.INI ファイルは、Windows ディレクトリにあります。)

  ```
  [AppName]
  myDialogTitle=Load file
  ```

- X Window システムでは、以下の行をリソース・マネージャに渡します。

  ```
  AppName*myDialogTitle: Load file
  ```

  xrdb プログラムを使用するか、またはそれを X クライアントが読み込むファイル (.Xdefaults ファイルまたは XENVIRONMENT 変数が指定したファイル) に含めることができます。

デフォルトのディスプレイ・システム・リソース

IlvDisplay クラスのインスタンスが作成されると、デフォルトのディスプレイ・システム・リソースは、システム・リソース・メカニズムを使用して初期化されます。

***表 6.1** IlvDisplay デフォルト・リソース*

IlvDisplay メソッド	システム・リソース名	デフォルト値
IlvDisplay::defaultForeground	foreground	black
IlvDisplay::defaultBackground	background	gray
IlvDisplay::defaultFont	font	システム依存
IlvDisplay::defaultNormalFont	normalfont	システム依存
IlvDisplay::defaultBoldFont	boldfont	システム依存
IlvDisplay::defaultItalicFont	italicfont	システム依存
IlvDisplay::defaultLargeFont	largefont	システム依存

環境変数およびリソース名

UNIX および PC のデフォルト環境変数は、リソース・ファイル (すなわち、.Xdefault および .INI) で指定されたリソース名に優先します。以下の表は、環境変数とそれに関連付けられているリソース名の一覧です。

***表 6.2** 環境変数およびリソース名*

環境変数名	リソース名
ILVHOME	home 詳細は、*Home* を参照してください。
ILVLANG	lang 詳細は、*IlvMessageDatabase クラス*を参照してください。

表 6.2 環境変数およびリソース名

環境変数名	リソース名
ILVDB	messageDB 詳細は、*IlvMessageDatabase クラス*を参照してください。
ILVLOOK	look look リソースは、次のいずれかの値、motif、windows、または win95 を使用します。詳細については、Gadgets マニュアルを参照してください。

Windows のディスプレイ・システム・リソース

Microsoft Windows 環境では、標準リソースに加えて Windows 固有のリソースを定義できます。これらのリソースは、次のとおりです。

- [TTY] このリソースが TRUE に設定されている場合、すべての IBM ILOG のメッセージを送るメッセージ・ウィンドウが作成されます。デフォルトの値は FALSE です。
- [TTYw], [TTYh], [TTYx], [TTYy] これらのリソースは、TTY が TRUE に設定されている場合、メッセージ・ウィンドウのサイズと位置を指定します。デフォルト値は、TTYw=200、TTYh=100、TTYx=screen_width-TTYw、TTYy=screen_height-TTYh です。
- [UseRightButton] このリソースが TRUE に設定されており、マウスにボタンが 2 つある場合、IBM ILOG Views ライブラリによって生成された IlvEvent は IlvRightButton 値を持つことになります。その他の場合は、IlvMiddleButton 値が含まれます。デフォルトの値は FALSE です。

  詳細については、メンバ関数 IlvDisplay::isRightButtonValueUsed および IlvDisplay::useRightButtonValue を参照してください。
- [SolidColors] このリソースが TRUE に設定されている場合、VGA システムのパレットを使用します。システムで利用可能な色数が 16 色以上の場合、デフォルト値は FALSE です。それ以外の場合は、TRUE です。これは、色数の少ないグラフィック・システム上のディザ・イメージを防ぎます。
- [Warnings] このリソースが TRUE に設定されていると、警告メッセージが表示されます。デフォルトの値は FALSE です。

メンバ関数 IlvDisplay::getResource は、いくつかのファイルでリソース定義を検索します。下記はこれらのファイルを優先度の高い順に示したものです。

1. EXECDIR\APP.INI

2. EXECDIR\PROG.INI
3. EXECDIR\VIEWS.INI
4. WINDIR\APP.INI
5. WINDIR\PROG.INI
6. WINDIR\VIEWS.INI

EXECDIR は、実行可能なプログラムを含むディレクトリで、WINDIR は Microsoft Windows がインストールされているディレクトリです。APP は、アプリケーション名を表します。この文字列は IlvDisplay コンストラクタに提供したものです。PROG は実行可能ファイルのベース名で、完全な名前は PROG.EXE です。

Home

ほとんどの IBM® ILOG® Views アプリケーションでは、定義済みデータ・ファイルを読み込む必要があります。データ・ファイルを透過的に読み込むには、ライブラリがこれらのデータ・ファイルのディスク上の位置を特定する方法が必要です。これを行うには、ILVHOME 環境変数の値を取得します。この変数が定義されていない場合、IBM ILOG Views はディスプレイ・システム・リソース home の値を取得しようとします。一般的に、この値は ILOG Views がインストールされたディレクトリ (サブディレクトリ include、lib、data などを含むディレクトリ) に設定されています。

> ***メモ:*** *旧 IlvHome ディスプレイ・システム・リソースは、下位互換性を保つために保持されていますが、現在は使用されていません。*

home の設定を強制するグローバル関数には次の 2 つがあります。

- IlvGetDefaultHome
- IlvSetDefaultHome

環境変数 ILVHOME またはリソース home の設定を要求せずに妥当なデフォルト値を特定のアプリケーションに提供する場合は、これらの関数を使用します。

home の値は、次のセクションで説明されているディスプレイ・パスのデフォルト値の計算に使用します。

ディスプレイ・パス

*ディスプレイ・パス・メカニズム*によって、ファイルへのアクセスが非常に簡単になります。ファイルを開き、読み込む関数の呼び出しで使用されるパス名が相対

的な場合、関数はディスプレイ・パスで指定されたディレクトリでファイル名を検索します。

IlvDisplay のメンバ関数は、ディスプレイ・パスをチェックしたり操作したり、ファイル名がディスプレイ・パスで指定されたディレクトリに存在するかどうかをチェックするために使用できます。

ディスプレイ・パス・メカニズムの詳細については、以下を参照してください。

- *ディスプレイ・パスの設定*
- *パス・リソース*
- *ILVPATH 環境変数*
- *ディスプレイ・パスの問い合わせまたは変更*
- *例 ディレクトリをディスプレイ・パスに追加する*

ディスプレイ・パスの設定

ディスプレイ・パスは複数のディレクトリ・パス名を含む文字列で、通常のシステム・パス区切り文字 (*UNIX* では「:」、*DOS* では「;」) で区切られています。

最初 (IlvDisplay インスタンスの作成時) は、ディスプレイ・パスは次のように 3 つの異なる要素の連結に設定されています (UNIX 表記をパスに使用)。

```
<.:user path:system path:>
```

- 最初のセクションは、現在のディレクトリのみを含みます (ë.í と記述)。
- 次のセクションである user path は、ディスプレイ・リソース path の内容で構成されており、環境変数 ILVPATH の内容が続きます。
- 3 つ目のセクションである system path には、IlvHome のサブディレクトリが含まれます。

すなわち、IlvPath の初期値は次のとおりです (ILVHOME が定義済みの場合)。

```
.:<path resource>:$ILVPATH:$ILVHOME/data:$ILVHOME/data/icon:$ILVHOME
data/images
```

パス・リソース

- X Window では、path リソースは次の例のようになります。

  ```
  AppName*path: /usr/local/views/ilv
  ```

- Microsoft Windows では、path リソースは VIEWS.INI またはアプリケーションに依存するファイル .INI にあります。

  ```
  [AppName]
  path=C:\USER\DATA\ILV
  ```

ILVPATH 環境変数

ILVPATH 環境変数は、アプリケーションの起動前に設定できます。

- UNIX では、この設定は次の行で定義できます。

```
$ ILVPATH=/usr/home/user/ilvimages:/usr/home/user/ilvpanels
$ export ILVPATH
```

- Microsoft Windows コマンドのプロンプト・ウィンドウでは、設定は次のようになります。

```
C:\> SET ILVPATH=C:\USER\DATA\ILV;C:\USER\DATA\IMAGES
```

ディスプレイ・パスの問い合わせまたは変更

IlvDisplay クラスは、ディスプレイ・パスを操作するメンバ関数を備えています。これらは、IlvDisplay::getPath、IlvDisplay::setPath、IlvDisplay::appendToPath、および IlvDisplay::prependToPath です。

これらのメソッドにより、*user path* (ディスプレイ・パスの 2 つ目のセクション) を取得、設定、変更できます。ディスプレイ・パスの構造は変更されず、以下のとおりになります。

```
<.:user path:system path.>
```

IlvDisplay::findInPath メソッドは以下に使用します。

- ファイルがディスプレイ・パスにあるかどうかをチェックします。
- 絶対パス名を取得します。

例 ディレクトリをディスプレイ・パスに追加する

次の例は、ディレクトリをディスプレイ・パスに追加し、ファイルがディスプレイ・パスにあるかどうかをチェックする方法を示しています。

```
IlvDisplay* display = new IlvDisplay("AppName",
                                     "DisplayName",
                                     argc,
                                     argv);
if (display->isBad()) {
   delete display;
   IlvFatalError("Could not create display");
   IlvExit(-1);
}

const char* dirName  = "./localDirectory/subDirectory";
const char* fileName = "foo.txt";
display->prependToPath(dirName);
// Now, if a file such as
//    "./localDirectory/subDirectory/foo.txt"
// or
//    ".\localDirectory\subDirectory\foo.txt"
// exists, we should be able to find it.
const char* filePath = display->findInPath(fileName);
if (filePath)
    IlvPrint("File %s found at %s", fileName, filePath);
else
    IlvWarning("File %s not found", fileName);
```

章

7

ビュー

8 章 *描画ポート*では、IlvPort クラスが管理する描画ポートの概念について説明しました。ビュー階層は別の IlvPort サブクラスである IlvAbstractView およびその派生サブクラスを処理します。IlvView は主要サブクラスで、画面上で実際に描画を行う場所を表します。

- *ビュー階層 : 2 つの観点*では、ビューを構成するコンポーネントの概念を説明します。次のようなアプローチがあります。
  - *ウィンドウ指向のビュー階層*
  - *クラス指向のビュー階層*
- *IlvAbstractView: ベース・クラス*
- *IlvView: 描画クラス*
- *IlvView サブクラス*
- *IlvScrollView クラス*

ビュー階層 : 2 つの観点

ビューの構成方法には、次の 2 つの観点があります。

- *ウィンドウ指向*。画面の前に座って IBM ILOG Views を使用するという観点で、色々な種類のビューがさまざまな**ウィンドウ**に組み込まれ、異なった形で表示されるという見方です。このようなビューは、次々に作成されます。
- *クラス指向*。C++ の観点から、`IlvView` および関連する **クラス**を使用してさまざまな種類のビューを作成し、それらをまとめてウィンドウを構成するという見方です。

ウィンドウ指向のビュー階層

次は、ウィンドウ指向のビュー階層の概略と対応するクラスを表したものです。

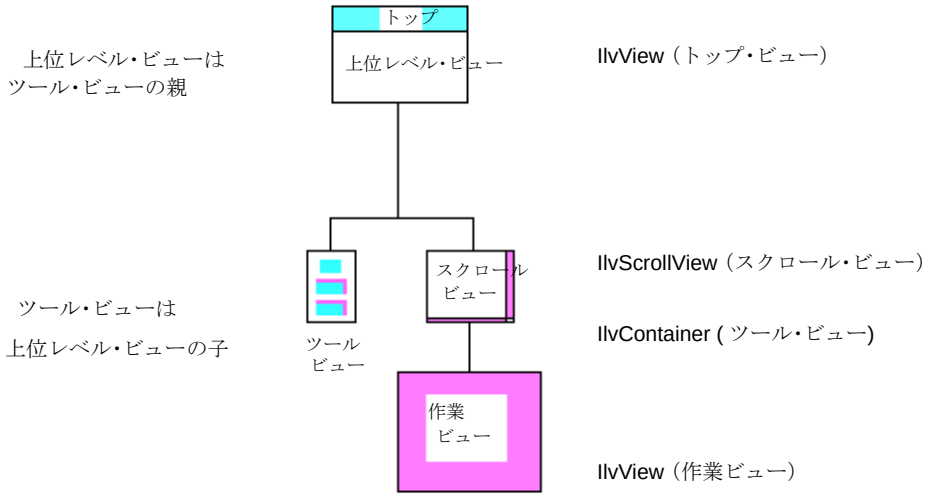


図 7.1 *ビュー階層*

親子関係

「親」および「子」という用語は、ビューの対の関係を指し、どちらかが他方を含むという考え方に基づいています。図では、上位レベル・ビューがツール・ビューおよびスクロール・ビューの親であり、後者は作業ビューの親です。逆に、スクロール・ビューは上位レベル・ビューの子です。

C++ クラスおよびサブクラスもまた、親子関係を表しています。ウィンドウ指向ビュー階層とクラス指向ビュー階層の間には、1 対 1 の対応関係がありません。C++ クラスの階層は、ビューの構成を違った観点からとらえています。クラス指向のダイアグラムについては、*クラス指向のビュー階層* on page 132 を参照してください。

クラス指向のビュー階層

ウィンドウ指向のビュー階層 on page 131 では、4 つのビューについて簡単に説明しました。これらは次のクラス (またはサブクラス) のインスタンスです。

- 上位レベル・ビュー (トップ・ウィンドウ) および作業ビュー：`IlvView`
- ツール・ビュー：`IlvContainer`
- スクロール・ビュー：`IlvScrollView`

下記のダイアグラムは、これらのクラスと `IlvAbstractView` から派生したその他のクラスです。

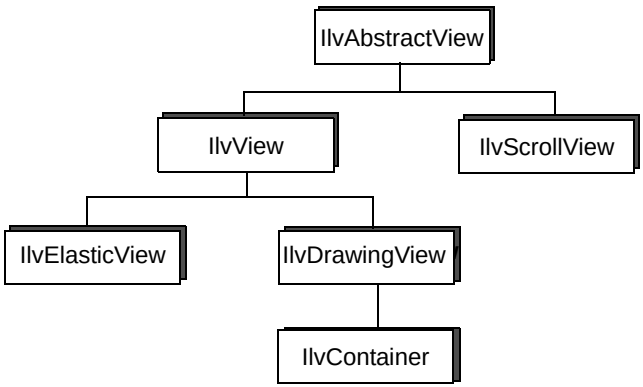


図 7.2 *IlvAbstractView ベース・クラス*

これらのビュー・クラスによって、画面に表示される実際のウィンドウとビューが作り出されます。`IlvAbstractView` の派生クラスの 1 つをインスタンス生成するときに、取得するオブジェクトを *ビュー* といいます。画面のウィンドウは、実際は関連付けられた 1 つ以上のビューの組です。

詳細は、以下を参照してください。

- *IlvAbstractView: ベース・クラス*
- *IlvView: 描画クラス*
- *IlvScrollView クラス*
- `IlvElasticView`、`IlvDrawingView`、および `IlvContainer` についての詳細は、*IlvView サブクラス* を参照してください。

IlvAbstractView: ベース・クラス

IlvAbstractView は (インスタンスがサブタイプ・クラスからのみ作成できる) 抽象クラスです。このクラスにはサイズ、可視性、色などの基本プロパティを処理する関数が備わっています。IlvAbstractView オブジェクトが、ディスプレイ・システムの実際のインターフェース・オブジェクト (すなわちシステム・ビュー。ウィジェットと呼ぶ場合もあります) をカプセル化します。このインターフェース・オブジェクトはプラットフォーム依存型であり、次の関数でアクセスできます。

```
IlvSystemView getSystemView() const;
```

ここで、IlvSystemView はディスプレイ・システム・ウィジェットの基本タイプです。

IlvView: 描画クラス

IlvView クラスは、*クラス指向のビュー階層* on page 132 に示される IlvAbstractView (つまり IlvPort) の子孫の 1 つです。

IlvView サブクラスは主要クラスです。これは画面上で実際に描画を行う場所を表すためです。IlvView のインスタンスもまた、マウスのクリックに反応するゾーンを表します。

IlvView クラスおよびそのサブクラスは、画面での描画に使用するオブジェクトを備えています。これは上位レベル・ウィンドウの場合もあれば、以前作成した親ビューの子の場合もあります。

特に 2 つのコンストラクタを使用して、新しいトップ・ウィンドウをディスプレイ・インスタンスに作成します。

```
IlvView(IlvDisplay*    display,
                const char* name,
        const char* title,
        const IlvRect& size,
        IlBoolean     visible = IlTrue);

IlvView(IlvDisplay*    display,
                const char* name,
        const char* title,
        const IlvRect& size,
        IlUInt        properties,
        IlBoolean     visible       = IlTrue,
        IlvSystemView  transientFor  = 0);
```

2 つ目のコンストラクタを使用すると、ボーダー、バナー、ハンドルを処理するトップ・ウィンドウのアスペクトを指定できます。有効なシステム・ビュー値を

transientFor パラメータに提供できます。こうすると、新しい IlvView オブジェクトがそのシステム・ビューに対してトランジェントになります。これにはシステム・ビューがアイコン化されるとビューも暗示的にアイコン化されるという効果があります。

IlvView クラスのメンバ関数には、ビューがトップ・ウィンドウの場合のみ特定の意味を持つものがあります。

その他のコンストラクタは、以下のとおりです。

```
IlvView(IlvAbstractView* parent,
        const IlvRect& size,
        IlBoolean     visible = IlTrue);

IlvView(IlvDisplay*    display,
        IlvSystemView parent,
        const IlvRect& size,
        IlBoolean     visible = IlTrue);
```

parent パラメータは、IlvAbstractView または既存の IlvSystemView のいずれかです。

IlvView の最後のコンストラクタを使用すると、IBM® ILOG® Views は、以下のような別のアプリケーションで作成された既存の IlvSystemView を管理できます。

```
IlvView(IlvDisplay*    display,
        IlvSystemView existingWindow);
```

ネイティブ・アプリケーション (Microsoft Windows SDK または MFC、UNIX 上、X Window または Motif コードで作成されたもの) を IBM ILOG Views のグラフィック機能で拡張したい場合は、このコンストラクタを使用します。

既存の接続から IlvDisplay を作成することになります (*ディスプレイ・サーバとの接続*を参照)。これはウィンドウ階層が既に設定されていることが多く、IlvView オブジェクトが既存のウィンドウを管理する必要があるためです。

IlvView サブクラス

*クラス指向のビュー階層*の一部である IlvView のサブクラスは、以下のとおりです。

IlvElasticView クラス

IlvElasticView クラスは、このクラスのインスタンスがリサイズされるとき、伸縮自在にその子をリサイズすることを除けば、IlvView と同じ機能を備えています。これは、自動的にリサイズする他のビューを含むビューで使用できる特殊な IlvView クラスです。

IlvDrawingView クラス

IlvDrawingView もまた IlvView のサブクラスです。IlvDrawingView は、イベントのエクスポーズやリサイズなどの到着イベントを処理する定義済みメンバ関数を備えています。

IlvContainer クラス

IlvContainer クラスは*クラス指向のビュー階層*の最初のクラスで、グラフィック・オブジェクトの格納および表示を調整します。*コンテナ*には、多数の専門化されたサブクラスに関する説明があります。

IlvScrollView クラス

*クラス指向のビュー階層*の特殊なタイプのビューは、IlvScrollView クラスのインスタンスが管理しています。このクラスを正常に動作させるためには、アプリケーションに関連したウィジェット・ツールキットが必要です。

> ***メモ:*** *IlvScrollView は Microsoft Windows および Motif の上 (ネイティブ制御のあるポート) のみで実装されています。*IBM ILOG Views *Gadgets パッケージをご使用の場合は、IlvScrolledView クラスに類似サービスが用意されています。*

IlvScrollView には、親ビューよりも通常大きい子ビューが 1 つ必要です。IlvScrollView は、すべての自動スクロール動作を処理します。これには、スクロール・ビューの右下にあるスクロール・バー使用します。スクロール・バーを操作して、サブビューの新しい領域を表示することができます。

IlvScrollView オブジェクト内の非 IBM® ILOG® Views ウィンドウ・オブジェクトを処理できます。これは、他のアプリケーションが作成したものなど、どのようなシステム・ウィンドウでもかまいません。

章

8

描画ポート

IlvPort クラスで定義される描画ポートは、ユーザが描画を行う領域です。これは画面やプリンタなど、あらゆる出力デバイスになります。次のトピックに分けて、説明します。

- *IlvPort: 描画ポート・クラス*
- *IlvPort の派生クラス*
- *IlvSystemPort クラス*
- *IlvPSDevice クラス*

IlvPort: 描画ポート・クラス

IlvPort クラスは描画ポートを定義します。IlvPort クラスは、プリンタなどの具体的なダンプ・デバイスへ形状を描くのに必要なメンバ関数を備えています。これらのメンバ関数は、以下のとおりです。

- 仮想メンバ関数 IlvPort::initDevice は、init 関数によって呼び出され、ダンプ・デバイスを初期化してその結果を filename ファイルに書き込みます。
- 仮想メンバ関数 IlvPort::isBad は、ダンプ・デバイスが無効である場合、IlTrue を返します。この戻り値は、初期化エラーを意味します。

- 仮想メンバ関数 `IlvPort::end` は、ダンプ・デバイスを終了し、必要なクリーンアップをすべて実行します。
- 仮想メンバ関数 `IlvPort::send` を使うと、あらゆる文字列すなわち情報を出力デバイスに送ることができます。
- 仮想メンバ関数 `IlvPort::newPage` は、出力ページを作成し、新しいページ用にダンプ・デバイスを準備します。エラーが発生した場合は `IlFalse` を返します。この場合は、出力データの作成を中止してください。
- 仮想メンバ関数 `IlvPort::setTransformer` は、追加トランスフォーマ、すなわちジオメトリ変換を描画関数を「送る」座標に適用します。

IlvPort の派生クラス

下記の図は `IlvPort` から派生した定義済みクラスのいくつかの例を示したものです。

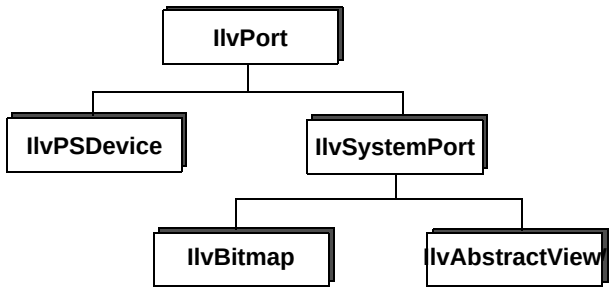


***図 8.1** IlvPort Base クラス*

`IlvPort` クラスは、描画ポートを以下のいずれかの方法で定義します。

- *IlvSystemPort クラス* を使用して、物理的に画面またはビットマップとして定義します。
- フィルタまたはプリンタとして専用ゾーンで定義します。

その他に以下の 2 つのサブクラスがあります。

- `IlvBitmap`。*ビットマップ*で説明されているように、ビットマップをサポートします。
- ビューのベース・クラスである `IlvAbstractView` サブクラス。詳細については、7 章 *ビュー*をご覧ください。特に、*IlvAbstractView: ベース・クラス*を参照してください。

IlvSystemPort クラス

IlvSystemPort クラスは、ユーザが描画可能な矩形領域を定義します。これは、実際の場所、または仮想の場所のいずれかです。実際の場所の場合、ワークステーションの画面領域に直接描画します。仮想の場所の場合、メモリのビットマップに描画します。クラス IlvBitmap および IlvAbstractView は、これら 2 つを実行するために IlvSystemPort から派生したものです。IlvBitmap クラスについては、本章の後半で説明します。IlvAbstractView クラスおよびそのサブクラスについては、次の章で説明します。

IlvPSDevice クラス

描画処理をプリンタやプロッタなどのダンプ・デバイスに転送するために、IBM? ILOG? Views はすべての描画処理を実装するサブクラスのメンバ関数を呼び出します。これらのメンバ関数は、ダンプ・デバイスのさまざまな実装に必要な描画処理を定義するためにオーバーロードされます。

IBM ILOG Views には、すぐに「表示されているものを印刷」できるように、定義済みのクラス IlvPSDevice が用意されています。

IlvPSDevice クラスを使うと、ビューのあらゆる領域をテキスト・ファイルに印刷できます。これは *PostScript* プリンタですぐに印刷可能です。さらに、すべての描画メンバ関数は、予想される結果に対応する *PostScript* コードを作成するために実装されます。

章

9

コンテナ

コンテナは IlvContainer クラスのインスタンスで、グラフィック・オブジェクトを格納、表示する特別なタイプのビュー です。コンテナについて、次のセクションに分けて説明します。

- *IlvContainer: グラフィック・プレースホルダ・クラス*
- *コンテナの表示*
- *イベントの管理：アクセラレータ*
- *イベントの管理：オブジェクト・インタラクタ*
- *複雑な振る舞いを持つオブジェクトの作成*

IlvContainer: グラフィック・プレースホルダ・クラス

IlvContainer は、グラフィック・プレースホルダ・クラスです。コンテナでオブジェクトを処理するメンバ関数については、以下で説明します。

- *汎用メンバ関数*
- *関数をオブジェクトに適用する*
- *タグ付きオブジェクト*

- *オブジェクト・プロパティ*

汎用メンバ関数

`IlvContainer::addObject` や `IlvContainer::removeObject` などの `IlvContainer` クラスのいくつかのメンバ関数は、コンテナ内のオブジェクトを格納、削除します。(詳細については、*IBM ILOG Views リファレンス・マニュアル*を参照してください)。`IlvContainer` オブジェクトはグラフィック・オブジェクトをリストに格納します。

関数をオブジェクトに適用する

ユーザ定義の関数をコンテナ・オブジェクトに適用するメンバ関数もあります。詳細は、以下のとおりです。

- `IlvContainer::applyToObjects`　ユーザ定義の関数を `IlvContainer` の各オブジェクトに適用します。
- `IlvContainer::applyToObject`　ユーザ定義の関数を指定されたグラフィック・オブジェクトのみに適用します。

タグ付きオブジェクト

`IlvContainer` クラスは、タグ付きのグラフィック・オブジェクトを管理するメンバ関数を備えています。タグとは `IlSymbol` クラスのオブジェクトで表されるマーカーの一種で、複数のグラフィック・オブジェクトを関連付けることができます。

- `IlvContainer::applyToTaggedObjects` は、タグ付きオブジェクトに使用することを除いては、`IlvContainer::applyToObjects` メソッドに類似しています。
- `IlvContainer::getTaggedObjects` は、指定されたタグの付いたコンテナに格納されたオブジェクトへのポインタ配列を返します。
- `IlvContainer::removeTaggedObjects` は、指定されたタグの付いたすべてのオブジェクトをコンテナから削除します。

IlSymbol クラス

IBM ILOG Views では、タグのような特定のエンティティを操作する文字列定数を必要とする場合があります。このために、任意のアプリケーションで固有の文字列を処理する一般的な方法があります。この文字列は *シンボル* と呼ばれ、`IlSymbol` クラスによって管理されます。

`IlGetSymbol` グローバル関数は、新しいシンボルを作成したり、作成済みのシンボルにアクセスできます。

オブジェクト・プロパティ

IlvContainer クラスのメンバ関数には、コンテナ・オブジェクト・プロパティを管理するものがあります。これは IlvContainer::getObject、IlvContainer::setObjectName、IlvContainer::setVisible などの関数です。たとえば、以下の目的に使用されます。

- さまざまな基準に従ってグラフィック・オブジェクトにアクセスします。これらの基準とは、名前またはそのインデックス識別子です。これはコンテナがリストにオブジェクトを格納するためです。
- 2 つのオブジェクトのコンテナのリスト順を入れ替えます (IlvContainer::swap メソッド)。
- オブジェクトの可視性、またはステータスの変更を要求します。(可視性とは画面上でオブジェクトが表示されるかどうかを意味します)。
- スタティック・メソッド IlvContainer::GetContainer により、グラフィック・オブジェクトの格納場所、すなわち格納コンテナを要求します。オブジェクトを複数のコンテナに格納することはできません。

> ***メモ:*** *メンバ関数* IlvContainer::GetContainer *はスタティックであるため、* IlvContainer *の既存インスタンスに適用する必要はありません。以下のメソッドは、任意の場所で、次のように記述して使用できます。*
> ```
> IlvContainer::GetContainer(myobject);
> ```

コンテナの表示

コンテナを表示するメンバ関数については、以下で説明します。

- *描画メンバ関数*
- *ジオメトリ変換*
- *ダブル・バッファリングの管理*
- *ディスクからオブジェクトを読み込む*

描画メンバ関数

メンバ関数 IlvContainer::draw および IlvContainer::reDraw は、親クラスの IlvDrawingView から継承されたものですが、コンテナとは特定の方法で対話します。下記は、これらの特定のメンバ関数です。

- IlvContainer::draw　は、IlvContainer オブジェクトに格納されているすべての IlvGraphic オブジェクトを描画します。オブジェクトをコンテナに追加するには、このメソッドを呼び出すだけで十分です。2 つの仮想メンバ関数を使うと、宛先ポートやトランスフォーマに関係なく、指定されたクリッピング領域に描画できます。
- IlvContainer::reDraw　作業領域を更新する場合 (オブジェクトを移動する場合など) はこのメソッドを使用します。これは IlvContainer::draw メソッドを呼び出す前に、指定されたクリッピング領域を消去します。
- IlvContainer::reDrawObj　グラフィック・オブジェクトに適用すると、オブジェクトのバウンディング・ボックスを再描画します。
- IlvContainer::bufferedDraw　隠しピクセル・マップで仮描画を実行して、画面上に速やかにピクセル・マップを表示します。ダブル・バッファリングとの違いは、この処理が、矩形、領域、またはオブジェクトにローカライズされ、描画処理中に限りこれが続くことです。

  1 つの関数が指定された IlvRect オブジェクトに含まれる領域を描画し、別の関数が指定された IlvRegion オブジェクトに含まれる領域を描画します。これは両方ともコンテナの座標系に描画します。3 つ目の関数が、指定された IlvGraphic オブジェクトをオブジェクトの座標系に描画します。

ジオメトリ変換

メンバ関数の中にはコンテナ・ビューに適用するジオメトリ変換を処理するものがあります。これはビューに関連付けられた IlvTransformer オブジェクトを処理します。

- IlvContainer::getTransformer　コンテナ・ビューに関連付けられたトランスフォーマを返します。0 が返される場合、このコンテナにはトランスフォーマがありません。つまり、オブジェクトとそのディスプレイが同一だということです。
- IlvContainer::setTransformer　指定されたトランスフォーマのパラメータを設定します。
- IlvContainer::addTransformer　パラメータとして与えられたトランスフォーマで、現在のトランスフォーマを設定して、結果的に得られたトランスフォーマを、新たに現在のトランスフォーマとして設定します。
- IlvContainer::translateView および IlvContainer::zoomView　現在のトランスフォーマを移動トランスフォーマおよびズーム・トランスフォーマでそれぞれ設定します。
- IlvContainer::fitToContents　コンテナ・ビューをリサイズして、可視アトリビュートが IlTrue にセットされているすべてのオブジェクトが、バウンディング・ボックスにぴったり収まるようにします。ビューの左上の座標は、

同じ位置に留まります。このメソッドは一般的に、コンテナがファイルから IlvGraphic オブジェクト一式を読み込む場合に使用されます。オブジェクトの位置は事前に認識されません。

```
IlvRect size(0, 0, 300, 300);
IlvContainer* cont = new IlvContainer(display, "Cont", "My Window",
                                      size, IlTrue, IlFalse);
cont->readFile("myfile.ilv");
cont->fitToContents();
```

- IlvContainer::fitTransformerToContents　可視アトリビュートが IlTrue に設定されたすべてのオブジェクトがビューで表示できるように、新しいトランスフォーマを計算します。コンテナのビュー・サイズは変更されません。IlBoolean 引数が IlTrue に設定されると、IlvDrawingView::reDraw が呼び出されます。このメソッドは一般的に、マップを何度かズームした後に全体を見る場合に使用します。

```
static void
ShowAllMap(IlvContainer* container)
{
    container->fitTransformerToContents(IlTrue);
}
```

ダブル・バッファリングの管理

ダブル・バッファリング・モードではアニメーション表示や多数のオブジェクトをちらつきなく表示することが可能になります。このモードは、次のメンバ関数によって処理されます。

- IlvContainer::setDoubleBuffering　コンテナがダブル・バッファリングを使用するかどうかを指定します。
- IlvContainer::isDoubleBuffering　ダブル・バッファリングを使用中かどうかを通知します。

ディスクからオブジェクトを読み込む

オブジェクトをディスクから読み込むには、次の 2 つのメンバ関数を利用します。

- IlvContainer::readFile　名前がパラメータとして指定されているファイルを読み込みます。
- IlvContainer::read　パラメータとして指定された入力ストリームから読み込みます。

これらのメンバは両方とも、読み込みの結果を返します。問題なく読み込めた場合は IlTrue、エラーが発生した場合は IlFalse となります。

> ***メモ:*** *コンテナはマネージャと異なり、コンテンツを保存する* write *メンバ関数を備えていません。マネージャの詳細については、『Manager』マニュアルを参照してください。*

イベントの管理：アクセラレータ

アクセラレータは、それが付加されたコンテナで発生する単一のユーザ・イベントを管理します。アクセラレータは、このシングル・ユーザ・イベントと関数の呼び出しを直接結び付けます。適切なイベントが発生した場合に呼び出される任意の関数を宣言できます。適切なイベントが発生した場合、アクセラレータはそれが付加されたコンテナから目に見える応答をトリガします。

また、アクセラレータのリストの中から IlvContainerAccelerator クラスのインスタンスをインストールすることもできます。このインスタンスには監視するイベントが記述されており、このイベントが発生するとこのクラスのメンバ関数が呼び出されます。

詳細は、以下を参照してください。

- *メンバ関数*
- *アクセラレータの実装：IlvContainerAccelerator*
- *定義済みのコンテナ・アクセラレータ*

メンバ関数

メンバ関数の中にはアクセラレータを処理するものがあります。

- IlvContainer::addAccelerator　コンテナに新しいアクセラレータをインストールします。下記の例では、あるイベントがキーボード・イベントである「Q キーの解除」と一致すると、Quit 関数がトリガされます。

```
static void
Quit(IlvContainer* cont, IlvEvent&, IlAny)
{
   IlvDisplay* d = cont->getDisplay();
   delete d;
   IlvExit(0);
}IlvRect size(0, 0, 300, 300);
IlvContainer* cont = new IlvContainer(display, "Cont", "My Window",
                                      size, IlTrue, IlFalse);
cont->addAccelerator(Quit, IlvKeyUp, 'Q', 0);
```

- IlvContainer::removeAccelerator　引数で与えられたイベント記述と IlvContainer::addAccelerator により以前設定されたアクションの関連付けを解除します。
- IlvContainer::getAccelerator　特定のアクセラレータ・アクションとユーザ引数に関して、コンテナに問い合わせます。

アクセラレータの実装：IlvContainerAccelerator

パラメータをコールバック関数に追加する必要がある場合、IlvContainerAccelerator クラスを使用してクラスをサブタイプ化し、アクセラレータを実装できます。

このクラスを処理するメンバ関数は次のとおりです。

- IlvContainer::addAccelerator　コンテナに IlvContainerAccelerator オブジェクトをインストールします。前述の例は、以下のように記述することができます。

```
IlvContainerAccelerator* acc =
   new IlvContainerAccelerator(Quit, IlvKeyUp, 'Q', 0);
cont->addAccelerator(acc);
```

- IlvContainer::removeAccelerator　特定のアクセラレータ引数をコンテナのリストから削除します。アクセラレータは削除されません。
- IlvContainer::getAccelerator　特定のイベント引数に一致する IlvContainerAccelerator インスタンスへのポインタを返します。一致するアクセラレータがない場合は、0 を返します。

定義済みのコンテナ・アクセラレータ

IBM ILOG Views には、定義済みのアクセラレータが多数用意されており、コンテナに格納されているオブジェクトの表示的側面を簡単に操作するプログラムが作成できます。

表 9.1 *定義済みのコンテナ・アクセラレータ*

イベント・タイプ	キーまたはボタン	アクション
IlvKeyDown	i	トランスフォーマを同一に設定します。
IlvKeyDown	< 右 >	ビューを左に移動します。
IlvKeyDown	< 左 >	ビューを右に移動します。
IlvKeyDown	< 下へ >	ビューを上に移動します (x を減らす)。

表 9.1 定義済みのコンテナ・アクセラレータ

イベント・タイプ	キーまたはボタン	アクション
IlvKeyDown	< 上へ >	ビューを下に移動します。(x を増やす)
IlvKeyDown	Z	ビューを拡大する。
IlvKeyDown	U	ビューを縮小します。
IlvKeyDown	R	ビューを、反時計回りに 90 度回転させます。
IlvKeyDown	f	すべてのオブジェクトを表示できるように、新しいトランスフォーマを計算します。

イベントの管理：オブジェクト・インタラクタ

オブジェクト・インタラクタは、それが付加されているグラフィック・オブジェクトについて、ユーザ・イベントのフィルタリングを行います。適切なイベントが続いて発生すると、オブジェクト・インタラクタはグラフィック・オブジェクトから目に見える応答をトリガします。この応答をオブジェクトの振る舞いと呼びます。

IBM® ILOG® Views は、包括的な定義済みオブジェクト・インタラクタを備えています。IBM ILOG Views で定義されていない特殊な機能が必要な場合は、インタラクタ・クラスの 1 つをサブタイプ化し、そのメンバ関数 `handleEvent` を必要な機能と置き換えます。

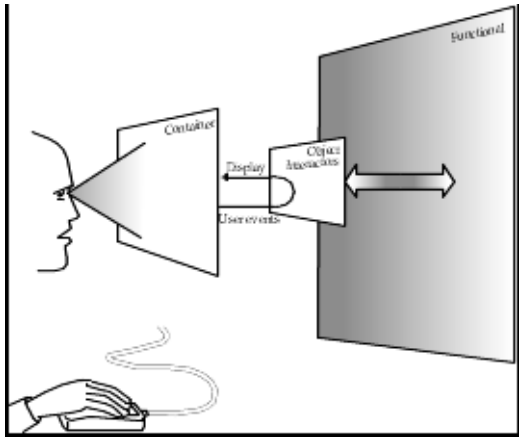


図9.1 *インタラクタをオブジェクトに付加する*

`IlvInteractor` クラスは、振る舞いをオブジェクトに関連付けます。

> ***メモ:*** *これらのオブジェクト・インタラクタは、エディタ作成用ではありません。対話的エディタの作成には、個々のオブジェクトではなくビュー全体に関連付けられている* IBM ILOG Views*Manager を使用します。*

オブジェクト・インタラクタの詳細については、以下を参照してください。

- *オブジェクト・インタラクタの使用*
- *定義済みのオブジェクト・インタラクタ*
- *例：インタラクタとアクセラレータのリンク*

オブジェクト・インタラクタの使用

オブジェクト・インタラクタを処理するメンバ関数は次のとおりです。

- `IlvGraphic::getInteractor`　引数として与えられた `IlvGraphic` オブジェクトに関連付けられた `IlvInteractor` インスタンスを返します。
- `IlvGraphic::setInteractor`　引数として与えられた `IlvInteractor` オブジェクトと `IlvGraphic` オブジェクトを関連付けます。

下記の例では、定義済みの IBM ILOG Views インタラクタを IlvLabel グラフィックと関連付けています。このオブジェクト上でマウスの左ボタンをクリックしてオブジェクトを移動させるのは、IlvMoveInteractor クラスのインスタンスです。

```
IlvRect size(0, 0, 300, 300);
IlvContainer* cont = new IlvContainer(display, "Cont", "My Window",
                                      size, IlTrue, IlFalse);
IlvLabel* label = new IlvLabel(display, IlvPoint(100,100),
                               "Hello world!");
cont->addObject(label);
label->setInteractor(IlvInteractor::Get("Move"));
```

スタティック・メンバ関数の IlvInteractor::Get は、「Move」というオブジェクト・インタラクタの固有インスタンスを返します。通常はインタラクタのコントラクタを直接呼び出してインタラクタを作成するのではなく、このスタティック・メンバ関数を使用します。これはほとんどのオブジェクト・インタラクタが多数のグラフィック・オブジェクトで同時に共有できるためです。

新しい IlvInteractor サブクラスの登録

IlvInteractor クラスをサブタイプ化する場合、スタティック・メンバ関数 IlvInteractor::Get を使用するためにサブクラスを登録する必要があります。下記は、MyInteractor クラスの IlvInteractor クラスをサブタイプ化するヘッダー・ファイルのサブクラス登録部分です。

```
class MyInteractor
: public IlvInteractor {
public:
    IlBoolean handleEvent(IlvGraphic*     obj,
                          IlvEvent& event,
                          IlvTransformer* t);
    ...
     DeclareInteractorTypeInfo(MyInteractor);
};
```

ここで クラスの永続化と登録を行う DeclareInteractorTypeInfo マクロを呼び出す行を追加しています。このマクロにより IlvInputFile への参照を必要とするコンストラクタ、MyInteractor への参照を必要とするコピー・インストラクタ、インタラクタ・インスタンスの保存に必要な write メンバ関数を定義しなくてはならなくなります。当然、コンストラクタと write 関数は一致する必要があります。これは IlvGraphic クラスの場合と同様です。

インタラクタに保存する追加情報がない場合は、DeclareInteractorTypeInfoRO マクロを使用します。これは write メンバ関数を定義する必要がありません。

この例では保存する追加情報はありませんが、完全な例を提供するという観点から、保存、読み込みを行うダミーの整数値を使用しています。

```
MyInteractor::MyInteractor(IlvInputFile& file)
: IlvInteractor(file)
{
    IlInt i;
    file.getStream() >> i; // Read a (dummy) integer value
}

IlvInteractor*
MyInteractor::write(IlvOutputFile& file)
{
    file.getStream() << (IlInt)0;
}
```

DeclareInteractorTypeInfoRO を使用した場合、コンストラクタは空になり、write 関数は定義されません。

ソース・ファイルにあるその他の関数の本文外で、次の 2 つのインストラクションを書く必要があります。

- `IlvPredefinedInteractorIOMembers(MyInteractor)`

  IlvPredefinedInteractorIOMembers は入力ファイルからコンストラクタを呼び出して、copy メンバ関数を定義するプロキシ関数を生成するマクロです。

- `IlvRegisterInteractorClass(MyInteractor, IlvInteractor);`

  IlvRegisterInteractorClass は MyInteractor クラスを新しく利用可能なインタラクタ・クラスとして登録するマクロです。2 つ目のパラメータは親クラスの名前でなければなりません。

定義済みのオブジェクト・インタラクタ

定義済みのオブジェクト・インタラクタをプログラミングする場合に役に立つクラスがいくつかあります。

- `IlvButtonInteractor`　このクラスをあらゆるグラフィック・オブジェクトに付加して、標準のインターフェース・ボタンとして振る舞うようにできます。
- `IlvRepeatButtonInteractor`　このクラスは、IlvButtonInteractor クラスのサブタイプです。これは、ユーザが任意の速度でマウス・ボタンを押したり放したりしているかのように、ボタンのアクションを自動的に繰り返します。
- `IlvToggleInteractor`　このクラスは、IlvButtonInteractor クラスのサブタイプです。このサブクラスは、ユーザがそのオブジェクトでマウス・ボタンをクリックすると、(invert メンバ関数を呼び出す) オブジェクトをこのインタラクタが関連付けられているオブジェクトに反転させます。
- `IlvMoveInteractor`　オブジェクトをクリックし、ポインティング・デバイスを別の場所にドラッグしてオブジェクトを移動します。

- `IlvReshapeInteractor` はマウスの右ボタンで矩形を作成し、オブジェクトの形状を変更します。矩形はオブジェクトの新しいバウンディング・ボックスになります。
- `IlvMoveReshapeInteractor` 2 つのインタラクタ `IlvMoveInteractor` と `IlvReshapeInteractor` を組み合わせます。
- `IlvDragDropInteractor` オブジェクトをコンテナから別のビューにドラッグ・アンド・ドロップします。オブジェクトをクリックし、オブジェクトのコピーをコンテナの外部にまでも移動できます。

例：インタラクタとアクセラレータのリンク

以下の例では、下記のウィンドウとその中にある 2 つの描画を作成します。

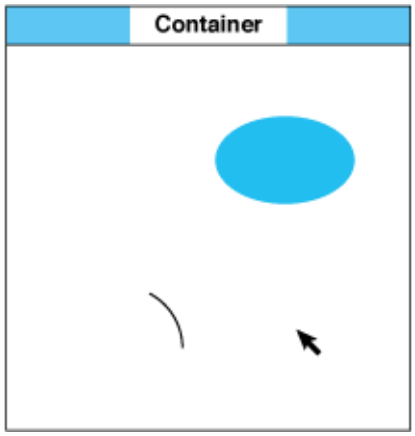


図 9.2 *2 つのオブジェクトを含むコンテナ*

すなわち、トップ・ウィンドウのコンテナを作成し、オブジェクトを 2 つ追加します。これらは灰色の楕円と円弧です。ただし、これらのオブジェクトを実際にコンテナに配置する前に、形状変更インタラクタを作成し、マウスを使用してグラフィック・オブジェクトの形状を変更します。このインタラクタ 1 つを 2 ヶ所で使用します。

- 楕円をコンテナに追加した直後に、形状変更インタラクタをこのオブジェクトに関連付けます。
- 円弧をコンテナに追加した直後に、同じ形状変更インタラクタをこのオブジェクトに関連付けます。

これで 2 つのグラフィック・オブジェクトのいずれかをクリックすると、形状変更インタラクタにより、マウスをドラッグして選択したオブジェクトの形状を変更できます。

さらに、2 つのオブジェクトと固有のインタラクタを作成した後、コンテナにアクセラレータを作成し、オブジェクトでマウスの左ボタンをダブルクリックしてその情報を表示させます。

- 楕円をダブルクリックすると、次のメッセージが表示されます。
  `Object is an IlvFilledEllipse`
- 円弧をダブルクリックすると、次のメッセージが表示されます。
  `Object is an IlvArc`

ここでは `PrintType` という 3 つの引数を持つ関数を作成し、これらのメッセージを生成します。この関数には定義済みの `IlvContainerAction` タイプが備わっています。この関数の名前は、`IlvContainer::addAccelerator` メンバ関数を呼び出す最初の引数としてコンテナに与えられます。

次に、デモ用プログラム全体を示します。例を短くするために、プログラムの終了部分は記述していません。

```
#include <ilviews/contain/contain.h>
#include <ilviews/graphics/ellipse.h>
#include <ilviews/graphics/arc.h>
#include <ilviews/graphics/inter.h>

static void PrintType(IlvContainer*, IlvEvent&, IlAny);

int
main(int argc, char* argv[])
{
    IlvDisplay* display = new IlvDisplay("Demo", "", argc, argv);
    if (!display || display->isBad()) {
        IlvFatalError("Couldn't open display");
        delete display;
        IlvExit(-1);
    }
    IlvContainer* container =
        new IlvContainer(display, "Demo", "Demo",
                         IlvRect(0, 0, 200, 200),
                         IlTrue, IlFalse);
    IlvInteractor* reshape = IlvInteractor::Get("Reshape");
    IlvGraphic* object =
        new IlvFilledEllipse(display, IlvRect(150, 50, 40, 20));

    container->addObject(object);
    object->setInteractor(reshape);
    container->addObject(object =
        new IlvArc(display, IlvRect(10, 150, 40, 40), 0., 60.));
    object->setInteractor(reshape);
    container->addAccelerator(PrintType,
                              IlvDoubleClick,
                              IlvLeftButton);
    container->show();
    IlvMainLoop();
    return 0;
}

static void
PrintType(IlvContainer* view, IlvEvent& event, IlAny)
{
    IlvGraphic* object =
        view->contains(IlvPoint(event.x(), event.y()));
    if (object)
        IlvPrint("Object is an `%s'\n" , object->className());
}
```

例の分析

このセクションでは、上記の例で使用したコードについて説明します。

```
static void PrintType(IlvContainer*, IlvEvent&, IlAny);
```

ユーザ定義の `PrintType` 関数がアクセラレータにより呼び出されます。アクセラレータはユーザがグラフィック・オブジェクト上でマウスの左ボタンをクリック

すると起動します。PrintType 関数の署名は、IlvContainerAction と呼ばれるタイプに対応しています。

```
IlvContainer* container = new IlvContainer(display, "Demo", "Demo",
                                    IlvRect(0, 0, 200, 20), IlTrue, IlFalse);
```

コンテナが上位レベル・ビューとして作成されます。

```
IlvInteractor* reshape = IlvInteractor::Get("Reshape");
```

形状変更インタラクタが特定されます。これは <ilviews/graphics/inter.h > を追加した際に自動的に登録されています。

```
IlvGraphic* object = new IlvFilledEllipse(display, IlvRect(150,50, 40,20));
container->addObject(object);
```

塗りつぶし楕円が作成され、コンテナに追加されます。

```
object->setInteractor(reshape);
```

形状変更インタラクタを塗りつぶし楕円に関連付けます。

```
container->addObject(object =
                     new IlvArc(display, IlvRect(10, 150, 40, 40), 0., 60.));
```

円弧をコンテナに追加します。

```
object->setInteractor(reshape);
```

形状変更インタラクタを円弧に関連付けます。

```
container->addAccelerator(PrintType, IlvDoubleClick, IlvLeftButton);
```

アクセラレータをコンテナに追加します。このアクセラレータは、ユーザがオブジェクト上でマウスの左ボタンをクリックするたびに、PrintType というユーザ定義の関数に適用されます。

```
static void
PrintType(IlvContainer* view, IlvEvent& ev, IlAny)
{
    IlvGraphic* object = view->contains(IlvPoint(ev.x(), ev.y()));
    if (object)
        IlvPrint("Object is a '%s'\n", object->className());
}
```

これはユーザ定義の PrintType 関数を実際に実装するものです。ここでは IlvContainer::contains メンバ関数を呼び出して、マウス・ポインタの下にあるオブジェクトにアクセスします。ここでは IlvPrint 関数を使用して、この例の移植性を確実にしています。

複雑な振る舞いを持つオブジェクトの作成

コンテナとそのインタラクタが処理する振る舞いは通常、単なる一連のアクションよりもずっと高度なものです。

このようなオブジェクトの作成を、以下のトピックに分けて説明します。

- *例：スライダの作成*
- *振る舞いとデバイスの関連付け*
- *デバイスの構築と拡張*

> ***メモ：****また* IBM ILOG Views *の* Prototypes *パッケージでも独自の専用オブジェクトが作成できます。*

例：スライダの作成

マルチメディア・システムを操作する C++ プログラムを作成したと想定し、IBM ILOG Views を使用してソフトウェアのグラフィカル・ユーザ・インターフェースを作成してみましょう。次の種類のデバイスを GUI に組み込むことにします。

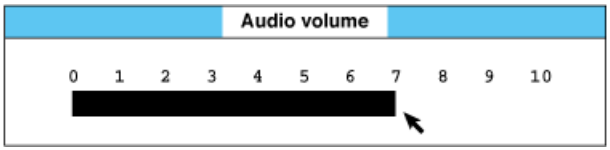


図 9.3 *ビジュアル・デバイスの作成：スライダ*

ユーザが黒色のバー上にカーソルを合わせ、マウスを左右にドラッグすることでバーの幅を変更できるようにします。また、バーの幅を変更した直後に、それに合わせて C++ ソフトウェアの関数が音声システムの音量を変更するようにします。スライダの振る舞いには 2 つの異なる側面があります。

- **表示要素**　視覚的な観点では、デバイスの外観はユーザがマウスを操作すると変更されます。カーソルをドラッグする方向により、黒色の矩形が縮小または拡張されます。

  したがって、これは幾何学形状の変更です。この例では、変更されるのは黒色矩形の幅ですが、他にも計器、ゲージ、ダイアルなど同じ基本的振る舞いを持つものが多数あります。たとえば、カーソルをドラッグする方向により、楕円形のオブジェクトを縮小したり拡大したりできます。

- **機能的要素**　スライダの真の目的は当然、画面上のグラフィック環境とはまったく異なるドメイン、つまり音声ドメインを変更することです。これはスライダの *操作的な振る舞い*です。

  スライダの操作的な振る舞いを処理するには、スライダの現在の値を返す `SliderValue` 関数や、スライダの設定変更により正または負の値を返す `SliderChange` 関数を使用します。これら 2 つの値を使用して、ソフトウェアの音声セクションへのリンクを確立し、音声が連動して変更されるようにします。

振る舞いとデバイスの関連付け

IBM ILOG Views では、振る舞いオブジェクト、すなわち *インタラクタ*という特殊なオブジェクトを作成することで、*例：スライダの作成*で説明されている特定の振る舞いをカプセル化します。すなわち、これは基本クラス `IlvInteractor` のインスタンスです。振る舞いオブジェクトを作成すると、それをウィンドウ、ビュー、黒色の矩形などさまざまな特定要素に関連付け、画面上に実際のスライダを作成できます。

スライダの場合では、まず `IlvInteractor` を使用して、上記で説明した操作的な振る舞いに使用する `SliderValue` や `SliderChange` のようなメンバ関数を持つ `IlvGaugeInteractor` というクラスを派生させます。

この `IlvGaugeInteractor` クラスの `handleEvent` メンバ関数は、スライダの現在の値を示す矩形 (または他の形) のサイズを変更するように作成されています。

スライダを使用するアプリケーション・ドメインはやや特殊なので、`AudioSlider` と呼ばれる `IlvGaugeInteractor` のサブクラスを派生します。このサブクラスには、マルチメディア・システムの音声パラメータ値を設定する特殊なメンバ関数が備わっています。したがって、クラス階層は次のようになります。

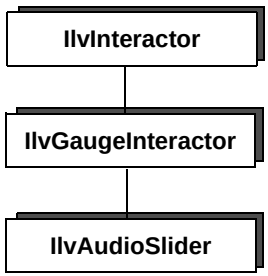
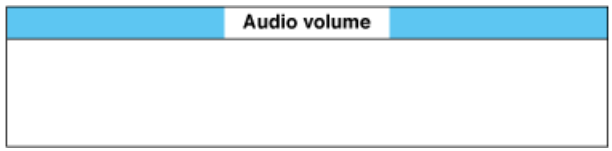


図 9.4　*IlvAudioSlider 階層*

デバイスの構築と拡張

次の例では、IBM ILOG Views などの拡張可能な製品の一般的な拡張方法を説明します。

1. アプリケーションをディスプレイ・サーバに接続します。
2. アプリケーションの要素を構成、表示する空のトップ・ウィンドウを作成します。



次のコードでコンテナを作成します。これは (0, 0) に位置するトップ・ウィンドウで、サイズは幅 100 ユニット、高さ 35 ユニットとなっています。

```
IlvRect viewsize(0, 0, 100, 35);
IlvContainer* topview = new IlvContainer(display, "Audio volume",
                                         "Audio volume", viewsize);
```

コンテナは、グラフィック・オブジェクトの格納と表示を調整します。

このウィンドウ内に特定のオブジェクトを描画し、これらのオブジェクトの 1 つにインタラクタを関連付けて特別な振る舞いを持たせるために、コンテナを利用する必要があります。

3. グラフィック・オブジェクトをコンテナに配置します。

次のコードで 2 つの別々のオブジェクトをコンテナ内に配置します。

```
container->addObject(scale);
container->addObject(bar);
```

4. 機能的な振る舞いを作成します。

この最終ステップでは、通常は音声スライダと関連付けられる音量スライダ的な振る舞いをバーに持たせます。この振る舞いは、AudioSlider クラスにカプセル化されています。したがって、次の行で音量スライダの振る舞いを持たせることができます。

```
IlvInteractor* inter = new AudioSlider();
bar->setInteractor(inter);
```

ここで、AudioSlider クラスは次のように定義されます。

```
class Audioslider
: IlvGaugeInteractor {
public:
    .../...
    virtual void doIt(IlvGauge* gauge)
    {
       setVolume(gauge->getValue());
    }
};
```

章

10

動的モジュール

動的モジュールは、共有ライブラリ (*ダイナミック・リンク・ライブラリ* または DLL とも呼ばれる) に含まれているオブジェクト・ファイル一式で構成されています。IBM® ILOG® Views はオン・ザ・フライでも実行時でも、動的モジュールの動的な読み込みが可能で、動的モジュールによって新しいクラスを定義できるので、実行中のプログラムがより機能的になります。

一般的には、動的モジュールはファイルが読み込まれるときに使われます。たとえば、データ・ファイルが `IlvGraphic` サブクラスへの参照を含んでおり、そのファイルを読み込むアプリケーションがそのサブクラスの存在を予期していない場合、IBM ILOG Views はエラー・メッセージを生成し、ファイルの読み込みを直ちに中止します。IBM ILOG Views は動的モジュールを使用して、このクラスを定義するコードを読み込むことができ、動的に利用できるようにします。

動的モジュール使用の詳細については、以下を参照してください。

- *IlvModule: 動的モジュール・クラス*
- *動的モジュールの作成*
- *動的モジュールの読み込み*
- *例 : 動的アクセス*

IlvModule: 動的モジュール・クラス

すべての動的モジュールは IlvModule ベース・クラスのサブクラスのインスタンスで、<ilviews/base/modules.h> ヘッダー・ファイルで定義されています。

すべての動的モジュールは、サブクラスを 1 つだけ定義します。このクラスのコンストラクタは、IBM® ILOG® Views がモジュールを読み込ときに呼び出され、モジュールが要求するすべての静的な初期化 (グラフィック・クラスの登録など) を実行できるようにします。

モジュール・クラスを (パブリック・ヘッダー・ファイルで) 宣言し、定義したら、IBM ILOG Views がそれを読み込めるようにする必要があります。これを行うには、いずれかの関数の本文外部で、ILVINITIALIZEMODULE マクロの呼び出しを追加し、唯一のパラメータとして定義するモジュールのクラス名を提供します。

動的モジュール・コードのスケルトン

IBM ILOG Views 動的モジュールのスケルトンは次のとおりです。

```
#include <ilviews/base/modules.h>

// Pre-initialization code goes here
// This is, typically, the declaration of global variables or
// static data members.

class MyModule
: public IlvModule {
public:
    MyModule(void*)
    {
        // Initialization code goes here
    }
};

ILVINITIALIZEMODULE(MyModule);
```

モジュール・クラスのコンストラクタに備わっているパラメータにより、アプリケーションに依存するデータをモジュールの初期化に送ることができます。これは、外部データを使用してモジュールを正しく初期化する際に必要です。これについては、この章の後半で詳しく説明します (*明示的モード*を参照してください)。

> ***メモ:*** *ほとんどの場合、モジュールの完全な初期化は 2 つの部分に分けられます。つまり、関数の外部コードでの変数の宣言とその初期化です。これには関数の呼び出しが必要な場合もあり、モジュール・コンストラクタに表示されなければなりません。*

動的モジュールの作成

動的モジュールは、システムに依存しています。このセクションでは、動的モジュールを正しくコンパイルしてインストールする方法、すなわち正しく読み込む共有ライブラリの作成方法について、システム別に説明します。

UNIX システム

UNIX を使用している場合、以下の一般構文を使用します。

```
<CCC> -c -O -I$ILVHOME/include moduleSrc.cpp
<MAKESHLIB> -o module.<SHEXT> moduleSrc.o [other object files...]
```

下記の表は、IBM® ILOG® Views のそれぞれのポートで利用できるさまざまなオプションです。モジュール・ファイル名の拡張子は必ず指定してください。

***表 10.1** UNIX システムの動的モジュールのコンパイル・オプション*

ポート名	CCC	MAKESHLIB	SHEXT
alpha_5.1_6.5	cxx -x cxx	/usr/lib/cmplrs/cc/ld -shared	so
hp32_11_3.73	aCC +DAportable -mt - AA -z +Z	aCC -b -n -mt -AA -Wl,+s	sl
hp64_11_3.73	aCC +DAportable -mt - AA -z +DA2.0W	aCC -b -n -mt -AA +DA2.0W	sl
x86_sles10.0_4.1 x86_RHEL4.0_3.4 x86-64_RHEL4.0_3.4	g++ -fPIC	g++ -shared	so
rs6000_5.1_6.0 power32_aix5.2_7.0	xlC -qrtti=all	xlC -qmkshrobj=1024	so
ultrasparc32_8_6.2 ultrasparc32_10_11	CC -KPIC	CC -G -h	so
ultrasparc64_8_6.2 ultrasparc64_10_11	CC -KPIC -mt -xtarget=ultra -xarch=v9	CC -xtarget=ultra -xarch=v9 -G -h	so

使用システムにより、以下のようになります。

- **Linux または Solaris ユーザ** `LD_LIBRARY_PATH` 変数に読み込むモジュールへのパスが含まれていることを確認してください。Linux の場合、実行可能ファイルは `libdl.so` ライブラリに必ずリンクさせてください。

Windows システム

Microsoft Windows の場合、実際の動的モジュールは DLL です。

この機能は、dll_mda 形式でのみ利用可能です。これは IBM ILOG Views が登録されたクラスのみをグローバル変数に格納するためで、「クライアント」(すなわちモジュールを読み込むアプリケーション) へ静的にリンクすると、各モジュールはクラスを再びローカルに作成します。

オブジェクト・ファイルのコンパイル時にモジュールを作成するには、/DILVDLL フラグを追加するだけです。コードを最適化するための、モジュール・ファイルのコンパイラ・フラグ一式は以下のようになります。

***表 10.2** Windows システムの動的モジュールのコンパイル・オプション*

ポート名	コンパイラ・フラッグ
x86_.net2003_7.1	CL /Gs /Ot /Ox /O2 /c /DWIN32 /MD /W3 /G3 /DILVDLL /I<ILVHOME>/include moduleSrc.cpp
x86_.net2005_8.0 and x86_.net2008_9.0	CL /Gs /O1 /c /DWIN32 /MD /W3 /DILVDLL /I<ILVHOME>/ include moduleSrc.cpp
x64_.net2008_9.0	CL /Gs /O1 /c /DWIN64 /MD /W3 /DILVDLL /I<ILVHOME>/ include moduleSrc.cpp

次のリンクを使用してモジュールにリンクします。

```
LINK /SUBSYSTEM:WINDOWS /DLL <ILVHOME>/lib/[PLATFORM]/dll_md/<lib>.lib\
<systemLibs> -OUT:<moduleName>.dll moduleSrc.obj [objectfiles...]
```

ここで、[PLATFORM] は、x86_.net2003_7.1x86_.net2005_8.0、x86_.net2008_9.0、x64_.net2008_9.0 のいずれかの値をとります。

x64_.net2008_9.0 については、次のリンカー・オプションも追加する必要があります。

```
/MACHINE:X64
```

上記の説明にあるように、モジュールを有効にするアプリケーションは IBM ILOG Views の DLL にも必ずリンクさせてください。DLL の使用時は必ず、システム・パスがモジュールを格納するディレクトリのパスへのアクセスを含むことを確認してください。

バージョン化に関する注意

IBM ILOG Views の動的モジュールは、バージョン化のメカニズムを備えていません。インストールされている動的モジュールがそれを読み込むアプリケーションとバイナリ互換であることを必ず確認してください。これは、IBM ILOG Views バージョン X.Y で作成したモジュールは、すべての X.Y.Z. バージョンで動作しますが、IBM ILOG Views X.W で開発したアプリケーションにより読み込まれた場合は再度コンパイルする必要があるということです。

動的モジュールの読み込み

基本的に、動的モジュールは暗示的、明示的のいずれかの方法で読み込むことができます。

- *暗示的モード*: 暗示的モードは、エンドユーザに対して透過的です。つまり、新しいクラスが名前で参照されたときに、アプリケーションがそのクラスを読み込みます。
- *明示的モード*: 明示的モードでは、読み込むモジュールとその場所を細かく指定します。

暗示的モード

暗示的モードでは、IBM® ILOG® Views は新しいクラスがデータ・ファイルで見つかり、クラス名が読み込むアプリケーションに登録されていない場合に、そのクラスを読み込みます。

このモードを正しく動作させるには、*モジュール定義ファイル*を作成する必要があります。定義されているクラスや使用している動的モジュールを IBM ILOG Views に自動的に認識させるには、*モジュール定義ファイル* (`.imd` 拡張子付き) を作成し、共有ライブラリと同じディレクトリに配置する必要があります。このファイルの名前はモジュールと同じでなければなりませんが、拡張子は `so`、`sl`、`dll` ではなく、必ず `imd` を使用します。このファイルを作成することにより、IBM ILOG Views は定義されているクラスを、検索したすべての動的モジュールを開いて読み出すことなく定義できます。

`myModule.dll` モジュールを Microsoft Windows で作成したと想定します。`myModule.dll` と同じディレクトリに `myModule.imd` というファイルを作成し、IBM ILOG Views がこのファイルに定義されたクラスを自動的に呼び出せるようにする必要があります。

このファイルの内容は ASCII テキストで、これを IBM ILOG Views の起動時に読み込んで定義されているクラスと配置されているモジュールを認識します。ファイルは、必ず次の行から始まります。

```
<?xml version="1.0"?>
<module>
```

次の終了タグで終了します。

```
</module>
```

次の形式で、`<module>` ブロック内に複数のグループを記述できます。

```
<class name = "NewClass" rootClass = "RootClass"/>
```

ここで、RootClass はクラス階層の最上位のクラス名 (IlvGraphic など) で、NewClass は作成したクラス名です。モジュールに複数のクラスを定義した場合、別の <class> タグを追加します。

モジュールで定義したクラスが複数のルート・ベース・クラスから派生する場合は、ルート・ベース・クラスと同じ数のブロックを新しく追加できます。

IBM ILOG Views アプリケーションを起動すると、モジュール・パスが読み込まれて利用可能なモジュール (正しい拡張子を持つファイル) を検索します。モジュールと関連付けられた imd ファイルが見つかった場合、記述ファイルが読み込まれ、それに含まれる情報は暗示的なクラス読み込み用に格納されます。

明示的モード

アプリケーションが読み込むモジュールとその場所を認識している場合は、明示的モジュール読み込みモードを使用できます。

myModule.dll モジュールを Microsoft Windows の C:\ilog\Views\Modules で作成したと想定します。このディレクトリが PATH 変数にある場合、スタティック・メンバ関数 Load を呼び出してモジュールを直接読み込む (適切な初期化すべてを行う) ことができます。

```
IlvModule* myModule = IlvModule::Load("myModule", myParameter);
```

IBM® ILOG® Views はこのモジュールを開き、パラメータ myParameter をモジュール・コンストラクタに送ろうとします。これを使用して詳しい情報を特定のモジュールに送ることができます。明示的モードでモジュールを読み込む場合、パラメータは常に 0 に設定されています。

最初のパラメータでは、モジュールの名前を指定します。IlvModule へのポインタではなく、モジュール名を操作する方が簡単な場合があります。

読み込みが失敗した場合、Load は 0 を返します。

返されたモジュール・インスタンスが破壊されると、動的ライブラリはアンロードされます。

例：動的アクセス

この例では、グラフィック・クラスを動作中のアプリケーショから動的にアクセスできるようにします。

IlvFilledRectangle から派生した CrossedRectangle クラスがあり、中に十字が描かれている輪郭付き矩形として表示されていると想定します。例は、次の手順で作成します。

- *サンプル・モジュール定義ファイルを書く*

- *新しいクラスの実装*
- *例の読み込みと登録*
- *マクロの登録*
- *サンプル・クラスを動的モジュールに追加する*

サンプル・モジュール定義ファイルを書く

まず、データ・ファイルがクラスの名前を参照したときに、IBM ILOG Views がこのクラスのコードを正しく読み込んでおり、対応するモジュールが明示的に読み込まれることを確認する必要があります。

これを行うには、モジュール定義ファイルを作成します。この例では 1 クラスしか使用しないため、非常に簡単になっています。

以下は、このモジュールの適切なモジュール定義ファイルの内容です。ここではクラスが 1 つしかないため、定義ファイルは次のようになります。クラスのルート・ベース・クラスは `IlvGraphic` です。

```
<?xml version = "1.0"?>
<module name="correct" version="1.0">
<class name = "CrossedRectangle" rootClass = "IlvGraphic"/>
</module>
```

このモジュール定義ファイルには、他の多数のクラスを追加することもできます。`IlvGraphic` から継承していないクラスでも使用できます。同じモジュールにクラスをインクリメンタルに追加できます。

新しいクラスの実装

モジュール定義ファイルを作成したら、この新しいクラスを実装するための作業を行います。この手順は非常に簡単です。

- 中に十字が描かれた塗りつぶし矩形を作成します。
- この新しいクラスが永続的であることを確認します。
- このクラスを動的モジュールに追加します。

最初の 2 ステップは既に馴染み深いものかもしれませんが、下記に対応するコードを示します。最後の部分はもっとも難しい箇所です。クラスを正しく登録するためにどのような操作を行うか確認しなければなりません。残念ながら、`IlvRegisterClass` マクロは、少なくとも移植可能な方法では、動的読み込みに使用できません。IBM ILOG Views では、この問題を解決するために、既に使い慣れているマクロに類似したマクロを数種類用意しています。これについては、`CrossedRectangle` クラスのコードを示した後に説明します。コードは次のとおりです。

```
#include <ilviews/graphics/rectangl.h>

class CrossedRectangle
: public IlvFilledRectangle {
public:
    MyRectangle(IlvDisplay* display,
                const IlvRect& size, IlvPalette* pal=0)
    : IlvFilledRectangle(display, drawrect, palette)
    {}
    virtual void draw(IlvPort* dst, const IlvTransformer* t = 0,
                      const IlvRegion* clip = 0) const;
    DeclareTypeInfoRO();
    DeclareIOConstructors(MyRectangle);
};

// Copy constructor
CrossedRectangle::CrossedRectangle(const CrossedRectangle& source)
: IlvFilledRectangle(source)
{}

// Read constructor
CrossedRectangle::CrossedRectangle(IlvInputFile& is,
                                   IlvPalette* pal)
: IlvFilledRectangle(is, pal)
{}

void
CrossedRectangle::draw(IlvPort* dst, const IlvTransformer* t,
                       const IlvRegion* clip) const
{
    if (clip)
        _palette->setClip(clip);
    IlvRect r = _drawrect;
    if (t)
        t->apply(r);
    dst->drawRectangle(_palette, r);
    dst->drawLine(_palette, r.upperLeft(), r.lowerRight()));
    dst->drawLine(_palette, r.upperRight(), r.lowerLeft());
    if (clip)
        _palette->setClip();
}

IlvPredefinedIOMembers(CrossedRectangle)
```

draw は、わかりやすいメソッドです。

このクラスには、必要に応じてコピー・コンストラクタおよび永続性に関連するメソッドが備わっていることが分かります。(ここでは保存する情報がないため、write メソッドが備わっていません。このため、クラス宣言に DeclareTypeInfoRO マクロを使用しています)。

新しいクラスを実装する 3 項目のうち 2 つについて既に説明しました。

このクラスを IBM ILOG Views の永続性メカニズムで登録するには、通常は一般的なステートメントを使用します。

```
IlvRegisterClass(CrossedRectangle, IlvFilledRectangle);
```

これは、関数の本文外に表示されます。

アプリケーションがそのコードにリンクしている場合、CrossedRectangle クラスのインスタンスの操作、保存、読み込みができます。ただし、CrossedRectangle クラスを必ず既存アプリケーションに認識させてください。既存アプリケーションは開発時にこのクラスを認識していないため、このクラスが定義されるアプリケーションが生成するデータ・ファイルを読み込むようになっています。これを行うには、このクラスを動的モジュールに接続する必要があります。その結果、以前のマクロをこの目的に使用することができなくなります。

例の読み込みと登録

ここでは、モジュールが読み込まれるとどうなるか、登録とは実際に何を行うのかを理解します。

- モジュールの読み込み時に、そのコンストラクタが呼び出されます。
- 登録とは、クラスレベルのアトリビュートを格納するクラス・レベルの変数宣言とこれらの変数を実際に更新する関数の呼び出しです。

IBM ILOG Views は動的モジュール機能により、IlvRegisterXXXClass マクロの代替を用意しています。これは多くの場所で使用されます (IlvGraphic および IlvNamedProperty サブクラスなど)。このマクロ一式は、登録の宣言と定義を切り離します。

登録の**宣言**の部分に使用されるマクロ名は、IlvRegister が IlvPreRegister となっていることを除き、IlvRegisterXXXClass に類似しています。IlvRegisterXXXClass の 2 つ目のパラメータが省略されています。このマクロ呼び出しは、関数の本文外に記述されなければなりません (これはクラス・レベルの変数のみを宣言します)。

登録の**定義**の部分に使用されるマクロ名は、IlvRegister が IlvPostRegister となっていることを除き、IlvRegisterXXXClass に類似しています。IlvRegisterXXXClass の 2 つ目のパラメータはそのままです。したがって、マクロ呼び出しは、呼び出されて実際に登録を正しく行う関数の本文内に記述されなければなりません (新しいクラスを登録するコードは呼び出しません)。

この例では、グラフィック・クラスを登録するのに、IlvPreRegisterClass マクロ (関数の本文外) および IlvPostRegisterClass マクロ (関数の本文内) の両方を使用する必要があります。

マクロの登録

以下は、永続的な IBM ILOG Views クラスのほとんどの登録に使用する必要があるマクロのリストです (c は登録されているクラス名を、s は親クラス名を示します)。

表 10.3 *動的モジュールでのマクロ登録*

クラス名	静的な登録マクロ	動的モジュールでのマクロ登録
IlvGraphic	IlvRegisterClass(c, s);	IlvPreRegisterClass(c); IlvPostRegisterClass(c, s);
IlvNamedProperty	IlvRegisterPropertyClass(c, s);	IlvPreRegisterPropertyClass(c); IlvPostRegisterPropertyClass(c, s);
IlvView	IlvRegisterViewClass(c, s);	IlvPreRegisterViewClass(c); IlvPostRegisterViewClass(c, s);
IlvGadgetItem	IlvRegisterGadgetItemClass(c, s);	IlvPreRegisterGadgetItemClass(c); IlvPostRegisterGadgetItemClass(c, s); ガジェットの詳細については、Gadgets マニュアルを参照してください。
IlvNotebookPage	IlvRegisterNotebookPageClass(c, s);	IlvPreRegisterNotebookPageClass(c); IlvPostRegisterNotebookPageClass(c, s);
IlvSmartSet	IlvRegisterSmartSetClass(c, s);	IlvPreRegisterSmartSetClass(c); IlvPostRegisterSmartSetClass(c, s);
IlvGroup	IlvRegisterGroupClass(c, s);	IlvPreRegisterGroupClass(c); IlvPostRegisterGroupClass(c, s);
IlvGroupNode	IlvRegisterGroupNodeClass(c, s);	IlvPreRegisterGroupNodeClass(c); IlvPostRegisterGroupNodeClass(c, s);
IlvUserAccessor	IlvRegisterUserAccessorClass(c, s);	IlvPreRegisterUserAccessorClass(c); IlvPostRegisterUserAccessorClass(c, s);

ここでは IlvGraphic のサブクラスを使用しているため、モジュールのソース・コード作成に必要なのは IlvPreRegisterClass マクロおよび IlvPostRegisterClass マクロのみです。

サンプル・クラスを動的モジュールに追加する

以下は、最終ソース・コードを IBM ILOG Views 動的モジュールにコンパイルできるように、CrossedRectangle クラスの定義に追加するコードです。

```
#include <ilviews/modules.h>

IlvPreRegisterClass(CrossedRectangle);

class MyModule
: public IlvModule
{
public:
    MyModule(void*)
    {
        IlvPostRegisterClass(CrossedRectangle, IlvFilledRectangle);
    }
};

ILVINITIALIZEMODULE(MyModule);
```

このコードをプレコンパイラ・ブロックである #if defined()/#else/#endif に追加することができます。この場合、通常の

```
IlvRegisterClass(CrossedRectangle, IlvFilledRectangle);
```

を if の #else に埋め込みます。これにより、コードを通常の静的オブジェクト・ファイルとして、または動的モジュールとしてコンパイルできます。

```
#if defined(MAKE_A_MODULE)
#include <ilviews/modules.h>

IlvPreRegisterClass(CrossedRectangle);

class MyModule
: public IlvModule
{
public:
    MyModule(void*)
    {
        IlvPostRegisterClass(CrossedRectangle, IlvFilledRectangle);
    }
};

ILVINITIALIZEMODULE(MyModule);
#else  /* DONT_MAKE_A_MODULE */
IlvRegisterClass(CrossedRectangle, IlvFilledRectangle);
#endif /* DONT_MAKE_A_MODULE */
```

章

11

イベント

この章では、イベントおよびイベント・ループについて説明します。タイマの操作、外部データ・ソースの追加、イベント・ループのカスタマイズ方法を扱います。下記のセクションを参照してください。

- *IlvEvent: イベント・ハンドラ・クラス*
- *IlvTimer クラス*
- *外部入力ソース (UNIX のみ)*
- *アイドル・プロシージャ*
- *下位レベルのイベント処理*

IlvEvent: イベント・ハンドラ・クラス

マウス・イベントおよびキーボード・イベントは、`IlvEvent` クラスによって処理されます。

イベント・シーケンスの記録と再生 IlvEventPlayer

IBM® ILOG® Views では、`IlvEventPlayer` を使用して制御するビューで発生するイベント・シーケンスを記録できます。これらのイベント・シーケンスは、デー

タ・ファイルに保存したり、そこから読み込んだり、あらゆる速度で再生できます。

イベント記録を処理する機能

イベントの記録を開始し、イベント・シーケンスを再生できるグローバル関数一式は、以下のとおりです。

- `IlvCurrentEventPlayer`
- `IlvRecordingEvents`

IlvTimer クラス

IBM® ILOG® Views は、*タイマ*を実装する内部メカニズムを備えています。内部メカニズムは、隠れており、システムに依存しています。このメカニズムは、`IlvTimer` クラスをベースとしています。

タイマは、指定した周期ごとに 1 度、関数を繰り返し呼び出すためのものです。このような形で関数を呼び出したい場合は、`IlvTimer` インスタンスを作成し、そのメンバ関数である `IlvTimer::run` を呼び出します。タイマ・オブジェクトは、そのメンバ関数 `IlvTimer::doIt` を周期が切れるたびに呼び出します。タイマは、ディスプレイ・システムのタイムアウト・メカニズムに基づいています。

タイマは、1 度のみ実行するように設定されている場合を除き、周期が終わるたびに `IlvTimer::doIt` を自動的に繰り返し呼び出します。`IlvTimer::doIt` を呼び出す前に、イベント・ループがタイマを無効にします。`IlvTimer::doIt` から返された後は、タイマが 1 度だけ実行されるよう設定されておらず、さらに無効になっていると (タイマのコールバック内から `IlvTimer::run` を呼び出して有効にできます)、再び有効になります。このメカニズムにより、コールバックにローカル・イベント・ループが含まれている場合でも、タイマはそのコールバック中はアクティブになりません。これはタイマがイベント・ループによりトリガされる場合のみで、アプリケーションが明示的に `IlvTimer::doIt` メソッドを呼び出す場合は該当しません。アプリケーションは、それが作成したタイマを削除する責任があります。

> ***メモ:*** *タイマに呼び出された関数の実行が周期と比べて大幅に時間がかかる場合、その周期が適切でない場合があります。*

`IlvTimer` クラスは、次の 2 つの場合に使用できます。

- 最初のケースは、`IlvTimerProc` タイプに一致する必要のあるユーザ定義関数がある場合です。

```
typedef void (* IlvTimerProc)(IlvTimer* timer,IlAny    userarg);
```

この場合は、IlvTimer オブジェクトをインスタンス生成し、この関数と使用する引数を指定するだけです。

- 2 つ目のケースでは、メンバ関数 IlvTimer::doIt をオーバーロードして、IlvTimer の派生サブクラスを使用します。

外部入力ソース (UNIX のみ)

UNIX プラットフォームの場合、IBM® ILOG® Views ではアプリケーションがファイル記述子を使用して、新しい入力ソースを追加できます。これらの代替入力ソースは、IlvEventLoop メソッド IlvEventLoop::addInput、IlvEventLoop::addOutput、IlvEventLoop::removeInput、IlvEventLoop::removeOutput で登録、登録解除ができます。下位互換性のため、旧関数である IlvRegisterInput、IlvRegisterOutput、IlvUnRegisterInput、IlvUnRegisterOutput がサポートされています。これらは以下と同等です。

```
IlvEventLoop::getEventLoop()->[add|remove][Input|Output]()
```

IBM ILOG Views は、これらの入力ソースからデータを読み込みませんが、これらを監視してファイル記述子が入力を受け取ったとき、または書き込み準備ができたときに通知します。このとき、IBM ILOG Views が特定の入力ソースと関連付けられたアプリケーション・コールバック・ルーチンを呼び出します。このコールバック・ルーチンは、ファイル記述子からの読み込み、またはファイル記述子への書き込みを処理します。アプリケーションはまた、ファイル記述子が新しい入力ソースとして IBM ILOG Views に追加される前にファイル記述子を開き、削除後に閉じる役割を担います。

以下は、標準入力から読み込み、標準出力へ 1 行に 1 語ずつコピーする簡単な IBM ILOG Views プログラムの例です。

```
#include <strstream.h>
#include <string.h>
#include <ilviews/view.h>

static void MyInputCallback(int, IlAny) {
   char buffer[1048];
   cin >> buffer;
   cout << buffer << endl;
   if (!strcasecmp(buffer, "quit"))
       exit(0);
}

int main(int, char*[]) {
   IlvEventLoop::getEventLoop()->addInput(0 /*stdin*/,
                                    MyInputCallback, 0, 0);
   IlvMainLoop();
}
```

アイドル・プロシージャ

アイドル・プロシージャとは、アプリケーションが提供するもので、アプリケーションがアイドル状態でイベント待ちの際にイベント・ループが呼び出す関数です。アイドル・プロシージャは簡単な計算を行う必要があります。アイドル・プロシージャが長すぎる場合、アプリケーションの対話応答に影響する場合があります。

アイドル・プロシージャは、他のタスクを続行する前に実行する必要がないタスクの実行に有用です。アイドル・プロシージャはすぐに完了しなくても、アプリケーションにとっては問題ありません。たとえば、アイドル・プロシージャは、隠れたダイアログ・ボックスをユーザ・アクションが要求する前に作成するのに使用します。

アイドル・プロシージャが `IlTrue` を返すと、それを自動的に削除して再び呼び出さないようにします。`IlFalse` を返した場合、アプリケーションがアイドルになるたびにそれが呼び出されます。これはアプリケーションが `IlTrue` を返すまで、またはそれがアプリケーションにより明示的に削除されるまで続きます。

アイドル・プロシージャを登録、登録解除するために、アプリケーションは `IlvEventLoop` メソッド `IlvEventLoop::addIdleProc` および `IlvEventLoop::removeIdleProc` を使用します。`IlvEventLoop::addIdleProc` の戻り値は、`IlvEventLoop::removeIdleProc` を呼び出して明示的にアイドル・プロシージャを削除するのに使用する ID です。一般的に、アイドル・プロシージャは `IlTrue` を返すため、削除する必要はありません。

下位レベルのイベント処理

アプリケーションがイベントを処理する最も一般的な方法は、アプリケーションを初期化した後に `IlvMainLoop` を呼び出すことです。`IlvMainLoop` は、次の到着イベントを取得し、適切なコンポーネントにディスパッチする無限ループです。ただし、アプリケーションには独自のイベント・ループを定義する必要のないものもあります。このため IBM® ILOG® Views では、次の関数またはメソッドを用意しています。

- イベント・ループを定義する `IlvDisplay` メソッドは次のとおりです。
  - `IlvDisplay::hasEvents`
  - `IlvDisplay::readAndDispatchEvents`
  - `IlvDisplay::waitAndDispatchEvents`
- イベント・ループを定義する `IlvEventLoop` メソッドは次のとおりです。
  - `IlvEventLoop::pendingInput`

- IlvEventLoop::processInput
- IlvEventLoop::nextEvent
- IlvEventLoop::dispatchEvent

メイン・ループの定義 例

以下は、IlvMainLoop と同等なコンストラクタの一覧です。

```
while (1)
    display->waitAndDispatchEvents();

while (1)
   IlvEventLoop::getEventLoop()->processInput(IlvInputAll);
```

Windows プラットフォームのみ

```
MSG msg;
while (IlvEventLoop::getEventLoop()->nextEvent(&msg))
    IlvEventLoop::getEventLoop()->dispatchEvent(&msg);

MSG msg; // obsolete version
while (IlvNextEvent(&msg))
    IlvDispatchEvent(&msg);
```

UNIX プラットフォームのみ

```
XEvent xev;
while (1) {
    IlvEventLoop::getEventLoop()->nextEvent(&xev);
    IlvEventLoop::getEventLoop()->dispatchEvent(&xev);
}

XEvent xev; // obsolete version
while (1) {
    IlvNextEvent(&xev);
    IlvDispatchEvent(&xev);
}
```

(libxviews とは対照的に **)** libmviews のみを使用した **UNIX** プラットフォーム

```
XtAppMainLoop(IlvApplicationContext());

XEvent xev;
while (1) {
    XtAppNextEvent(IlvApplicationContext(), &xev);
    XtDispatchEvent(&xev);
}
```

章

12

IlvNamedProperty: 永続性プロパティ・クラス

クラス IlvNamedProperty は、アプリケーションに依存する情報を IBM® ILOG® Views オブジェクトに関連付けます。この情報は名前付きプロパティと呼ばれ、定義した IlvNamedProperty サブクラスに格納されます。ユーザ・プロパティと異なり、名前付きプロパティはオブジェクトにコピーされ、永続的になり、データ・ファイルを保存、読み込む場合に内容が保持されます。

名前付きプロパティの使用と拡張について、以下のセクションに分けて説明します。

- *名前付きプロパティをオブジェクトと関連付ける*
- *名前付きプロパティの拡張*

名前付きプロパティをオブジェクトと関連付ける

ユーザ・プロパティと同様に、IlSymbol を使用して名前付きのプロパティをグラフィック・オブジェクトと関連付けます。IlSymbol を IlvNamedProperty のサブクラスの 1 つだけに付加して、確実に正しいタイプのプロパティを取得します。

名前付きプロパティを処理するには、次の IlvGraphic クラスの 3 つのメンバ関数を使います。

```
IlvNamedProperty* getNamedProperty(const IlSymbol*) const;
IlvNamedProperty* setNamedProperty(IlvNamedProperty*);
```

```
IlvNamedProperty* removeNamedProperty(IlSymbol*);
```

処理している名前付きプロパティを示すのに必要なのは、IlSymbol のみであることが分かります。

定義済み名前付きプロパティ：ツールチップ

定義済み名前付きプロパティの例として、*ツールチップ*があります。これは、ポインティング・デバイスがパネル制御要素 (ガジェットなど) に入り、しばらく留まったときにポップ・アップする小さなテキスト・ウィンドウです。IlvToolTip クラスは <ilviews/graphics/tooltip.h> ヘッダー・ファイルで定義されます。

グラフィック・オブジェクトにツールチップを設定、取得したり、そこから削除するには、次を記述します。

```
obj->setNamedProperty(new IlvToolTip("Text"));
...
IlvToolTip* toolTip = IlvToolTip::GetToolTip(obj);
...
delete obj->removeNamedProperty(toolTip->getSymbol());
```

名前付きプロパティを、別のオブジェクトへ移動させる場合があります。名前付きプロパティのリストから削除せずに名前付きプロパティを移動するには、IlvGraphic::removeNamedProperty メンバ関数を使用します。名前付きプロパティを削除し、使用していたメモリをクリアにするには、明示的に delete を呼び出します。

IlvNamedProperty のサブクラスとそれを参照するシンボル (すなわち IlSymbol へのポインタ) は緊密に結合しているため、ほとんどの場合、このシンボルはプロパティ・クラスのスタティック・データ・メンバになります。ただし、これは必須ではありません。プロパティ・クラスのスタティック・データ・メンバを使用すると、オブジェクトから名前付きプロパティを取得することができます。これに使用するシンボルはユーザに認識されない場合がありますが、クラスからは直接アクセスできます。

上記の例で、ツールチップ・プロパティが使用するシンボルは表示されないことがわかります。名前付きプロパティを取得するには、IlvNamedProperty::getSymbol メンバ関数を使用してプロパティのシンボルを取得するだけです。

名前付きプロパティはオブジェクトにコピーされて保存され、オブジェクトを削除すると削除されます。名前付きプロパティは既存クラスの追加データ・メンバのように振る舞い、パワフルな API を定義して名前付きプロパティ・クラスのデータにアクセスでききます。

名前付きプロパティの拡張

独自の名前付きプロパティ・クラスを作成するのは簡単で、次の 3 つの手順に従います。

1. `IlvNamedProperty` のサブクラスを作成します。
2. このプロパティにアクセスするシンボルを作成します。

   IBM ILOG Views では、「`_ilv`」で名前が始まるシンボルはすべて内部用に予約されています。
3. このクラスの永続性を定義し、IBM ILOG Views へ登録します。

次のセクションでは、2 つの値を格納する非常に簡単な例を使用して、グラフィック・オブジェクトと関連付けられる名前付きプロパティの作成方法を説明します。ただし、メンバ関数を使用して名前付きプロパティを作成してより複雑なデータ・メンバを処理したり、既存クラスへのポインタを格納する名前付きプロパティを作ったりすることもできます。名前付きプロパティを使用すると、必要最小限のコーディングで、アプリケーション・クラスの API を変更することなく、複雑なアプリケーション・データを軽量なグラフィック・オブジェクトにリンクできます。

例 : 名前付きプロパティの作成

ここでは整数と文字列の両方を含む名前付きプロパティを作成し、それを簡単にアクセス可能にし、永続性を持たせます。

一般的な操作は次のとおりです。

- *名前付きプロパティの宣言 : ヘッダー・ファイル*
- *名前付きプロパティにアクセスするシンボルの定義*
- *名前付きプロパティのコンストラクタを定義する*
- *setString メンバ関数の定義*
- *永続性およびコピー・コンストラクタの定義*
- *write メンバ関数の定義*
- *読み込み / コピー・コンストラクタへのエントリ・ポイントを提供する*
- *クラスの登録*
- *新規名前付きプロパティの使用*

名前付きプロパティの宣言：ヘッダー・ファイル

ここで作成する名前付きプロパティは、IlvNamedProperty のサブクラスである必要があります。名前付きプロパティは整数と文字列を格納します。完全なヘッダー・ファイルは以下のとおりです。

```
#include <ilviews/base/graphic.h>

class MyProperty
: public IlvNamedProperty
{
public:
    MyProperty(int   integer,
               char* string);
    virtual ~MyProperty();

    int         getInteger() const      { return _integer; }
    void        setInteger(int integer) { _integer = integer; }

    const char* getString()  const { return _string; }
    void        setString(const char* string);

    static IlSymbol* GetSymbol();

    DeclarePropertyInfo();
    DeclarePropertyIOConstructors(MyProperty);

private:
    int   _integer;
    char* _string;
    static IlSymbol* _Symbol;
};
```

2 つのデータ・メンバ _integer と _string (およびそのアクセサ) の他に、_Symbol スタティック・データ・メンバおよびクラスの宣言部分に表示される 2 つのマクロ DeclarePropertyInfo と DeclarePropertyIOConstructors に焦点を当てます。

クラスのデストラクタは、IlvNamedProperty ベース・クラスの 1 つと同様、仮想のものです。

名前付きプロパティにアクセスするシンボルの定義

まず、このクラスへのアクセスに使用するシンボルを定義します。このシンボルを簡単に定義するには、シンボルをプロパティ・クラス _Symbol のスタティック・データ・メンバにし、パブリック・アクセサ GetSymbol を提供します。これにより、インスタンスとオブジェクトの関連付けに使用するシンボルを認識する必要なく、どのアプリケーションでも MyProperty のインスタンスでも取得できるようになります。

パブリックおよびスタティック・アクセサの GetSymbol は、適切な IlSymbol を返し、必要に応じてそれを作成するように定義付けられます。

下記のコードは実装ファイルからの抽出であり、プロパティ・シンボルのアクセサと 0 に初期化するスタティック・データ・メンバの両方を定義します。シンボルは MyProperty::GetSymbol で最初に問い合わせを行うときに作成されます。

```
IlSymbol*
MyProperty::GetSymbol()
{
    if (!_Symbol)
       _Symbol = IlGetSymbol("MyPropertySymbol");
    return _Symbol;
}

IlSymbol* MyProperty::_Symbol = 0;
```

名前付きプロパティのコンストラクタを定義する

ここでは、コンストラクタとデストラクタを扱います。ここで必要なのは、親クラスである IlvNamedProperty のコンストラクタを呼び出し、データ・メンバを初期化することです。

```
MyProperty::MyProperty(int   integer,
                       char* string)
: IlvNamedProperty(GetSymbol()),
  _integer(integer),
  _string(0)
{
    setString(string);
}

MyProperty::~MyProperty()
{
    if (_string)
        delete [] _string;
}
```

最初に MyProperty タイプのプロパティを作成する際に、スタティック・メンバ関数である GetSymbol が呼び出され、スタティック・データ・メンバ _Symbol を有効値に設定します。

setString メンバ関数によって string パラメータがコピーされ、_string データ・メンバが有効であるかどうかチェックされます。このために、このデータ・メンバをコンストラクタのイニシャライザで 0 に初期化することが必要です。このパラメータは、有効である場合、デストラクタによって破壊されます。

setString メンバ関数の定義

下記は文字列をコピー、格納する setString メンバ関数の定義です。

```
void
MyProperty::setString(const char* string)
{
    if (_string)
        delete [] _string;
    _string = string
                ? strcpy(new char [strlen(string)+1], string)
```

```
                : 0;
}
```

このコードは非常に簡単です。有効な文字列、つまり非ヌル文字列が格納された場合、文字列は破壊されます。パラメータが有効、つまり非ヌルの場合、この文字列がコピーされ、格納されます。パラメータが有効でない場合、データ・メンバは単に 0 にリセットされます。

この段階で、この例のクラスは整数と文字列値の両方を格納、取得できます。

永続性およびコピー・コンストラクタの定義

この例で名前付きプロパティを完成させるには、クラス・レベルの情報と永続性に関連するメンバ関数を追加する必要があります。これをもっとも簡単に行うには、次の 2 つのマクロをクラス宣言の本文に使用します。

- `DeclarePropertyInfo` は、`MyProperty` クラスのクラス情報データ・メンバを宣言します。これらのメンバは、クラス名やその階層などの情報を取得するために使用します。これはまた、このクラスの永続性実装に必要なメンバ関数も宣言します。
- `DeclarePropertyIOConstructors` は、永続性およびコピーに必要なコンストラクタを宣言します。

これらのマクロにより、クラスにコピー機能および永続性機能を非常に簡単に追加できます。

メンバ関数を宣言したら、コピーおよび永続性機能をクラスに追加するために、コピー・コンストラクタとパラメータとして `IlvInputFile` を参照するコンストラクタを定義します。

```
MyProperty::MyProperty(const MyProperty& source)
: IlvNamedProperty(GetSymbol()),
  _integer(source._integer),
  _string(0)
{
    setString(source._string);
}

MyProperty::MyProperty(IlvInputFile& i, IlSymbol* s)
: IlvNamedProperty(GetSymbol()),
  _integer(0),
  _string(0)
{
    // 's' should be equal to GetSymbol()
    i.getStream() >> _integer >> IlvQuotedString();
    setString(IlvQuotedString().Buffer);
}
```

最初のコンストラクタはそのソース・パラメータのコピーで `MyProperty` の新しいインスタンスを初期化します。

2 つ目のコンストラクタは、提供された入力ストリームを読み込んで、そのインスタンスを読み込まれたもので初期化します。

write メンバ関数の定義

これでクラスの新しいインスタンスを読み込み、保存できます。このためには、write メンバ関数を定義します。これは暗示的に DeclarePropertyInfo マクロで宣言されます。

```
void
MyProperty::write(IlvOutputFile& o) const
{
    o.getStream() << _integer << IlvSpc() << IlvQuotedString(_string);
}
```

保存順は読み込み順と同じになります。

保存する追加情報がない名前付きプロパティを定義する場合もあります。この場合は、クラス宣言で DeclarePropertyInfo ではなく DeclarePropertyInfoRO マクロを使用して、無効になる write メンバ関数を削除します。

読み込み / コピー・コンストラクタへのエントリ・ポイントを提供する

IBM ILOG Views に読み込みおよびコピー・コンストラクタへのエントリ・ポイントを提供するには、別のマクロを実装ファイルに追加する必要があります。これは次のように、関数の本文外で行います。

```
IlvPredefinedPropertyIOMembers(MyProperty)
```

このマクロを呼び出すと実際に、読み込みコンストラクタを呼び出す read スタティック・メンバ関数が作成されます。これはまた、コピー・コンストラクタを呼び出す copy メンバ関数の定義も行います。

クラスの登録

名前付きプロパティをアプリケーションで実行する最終ステップとして、以下のように MyProperty クラスを IBM ILOG Views に登録します。

```
IlvRegisterPropertyClass(MyProperty, IlvNamedProperty);
```

IlvRegisterPropertyClass マクロの呼び出しにより、MyProperty クラスを IBM ILOG Views の永続性メカニズムに登録します。

新規名前付きプロパティの使用

これでこの新しい名前付きプロパティを、関連付けられるあらゆるグラフィック・オブジェクトの拡張として使用できます。

```
IlvGraphic* myObject = ...;
myObject->setNamedProperty(new MyProperty(12, "Some text"));
...
MyProperty* property =
          (MyProperty*)(myObject->getNamedProperty(MyProperty::GetSymbol());
if (property && (property->getInteger() == someValue))
```

```
    doSomething();
```

グラフィック・オブジェクトの API が、ベース・クラスをサブクラス化をすることなく永続的に拡張されました。

章

13

IBM ILOG Views における印刷

IBM® ILOG® Views は、印刷フレームワークを提供します。このフレームワークは、以下のクラスで構成されています。

- *IlvPrintableDocument クラス*は、文書、すなわちページ・レイアウトに関連付けられている印刷可能オブジェクトのリストを扱います。
- *IlvPrintable クラス*は、プリント可能オブジェクト、印刷コンテナ、マネージャ・ビュー、テキストなどを処理するさまざまなサブクラスを扱います。
- *IlvPrintableLayout クラス*は、背景、前景、ヘッダー、フッターなど定義済み領域を使用した文書のページ・レイアウトを扱います。定義済みレイアウトを使うと、1 ページまたは複数ページでの印刷や、同一レイアウトが可能になります。
- *IlvPrinter クラス*は、プリンタおよび用紙書式、マージン、色、グレイスケール機能、用紙の向きなどの物理的な特性を扱います。
- *IlvPrintUnit クラス*は、印刷単位を扱い、パイカ、センチメートル、インチ、ポイントなど各種単位の変換が可能になります。
- *IlvPaperFormat クラス*は、A4、レターなどの物理的な用紙書式を扱います。
- *ダイアログ*では、プリンタおよびプリンタの特性を選択するために、IBM ILOG Views が提供するユーザ・インターフェース・ダイアログについて説明します。Gadgets パッケージでは、印刷プレビュー・ダイアログも使用できます。

IlvPrintableDocument クラス

IlvPrintableDocument クラスは、印刷可能オブジェクトのリストを管理します。これはイテレータを使用して、プリント可能オブジェクトの順序を決定します。デフォルトのレイアウトも用意されていますが、印刷可能オブジェクトはそれぞれ独自のレイアウトを指定できます。

文書を複数印刷するには、次の 2 つのモードが使用できます。

- 文書全体を n 回印刷します。
- 1 ページを n 回印刷した後、次のページを印刷します。

イテレータ

イテレータは、IlvPrintableDocument::Iterator 内部クラスのインスタンスです。多くの場合は次の IlvPrintableDocument メソッドによって返されます。

- IlvPrintableDocument::begin() const;
- IlvPrintableDocument::end() const; イテレータは変数と同様に使用されます。

例

```
IlvPrintableDocument document;
// add some printables to the document
document.append(new IlvPrintableContainer(container);
.....
// the iterate through the printables
IlvPrintableDocument::Iterator begin = document.begin();
IlvPrintableDocument::Iterator end = document.end();

for (IlvPrintableDocument::Iterator iter = document.begin();
     iter != end;
     ++iter) {
   // do something with the printable.
   IlvPrintable* printable = iter.getPrintable();

}
```

IlvPrintable クラス

IlvPrintable は、印刷可能なオブジェクトを記述するベースを提供する抽象クラスです。これは任意のジョブの印刷パラメータを含む印刷可能ジョブに関連付けられています。

印刷可能オブジェクトは、次のメソッドをサブクラス化することで記述できます。

- `public virtual IlvRect getBBox(IlvPrintableJob const& job) const = 0;`
- `protected virtual IlBoolean internalPrint(IlvPrintableJob const& job) const = 0;`

`IlvPrintable` のサブクラスのいくつかを利用できます。

- `IlvPrintableContainer` は、`IlvContainer` のインスタンスをカプセル化します。

```
// declares a printable using the given region of a container.
// if the rectangle is null then the whole container is printed.
IlvPrintableContainer* printcont = new IlvPrintableContainer(container,
                                                             &rect);
```

- `IlvPrintableText` は、テキストを印刷します。また整列パラメータを指定できます。

```
// declares a printable using a simple text.
IlvPrintableText* printtext = new IlvPrintableText
                                        (display->defaultPalette(),
                                         "This is a text",
                                          IlvCenter);
```

- `IlvPrintableFormattedText` は、さまざまな定義済みアトリビュートを持つテキストを印刷します。各変換指定は、`%` という文字で始まります。下記のアトリビュートが定義されます。

`%p`	ページのインデックスを印刷します。
`%P`	総ページ数を印刷します。
`%N`	文書名を印刷します。
`%y`	年を印刷します。
`%M`	月 (数字) を印刷します。
`%d`	日付を印刷します。
`%h`	時間 (0 ～ 24) を印刷します。
`%H`	時間を印刷します。
`%m`	分を印刷します。
`%s`	秒を印刷します。
`%AM`	大文字で AM/PM インジケータを印刷します。
`%am`	小文字で AM/PM インジケータを印刷します。

%p コードを %\p に置換して印刷できます。

```
// declares a printable formated text
IlvPrintableFormattedText* printftext = new
   IlvPrintableFormattedText(display->defaultPalette(),
                             "%N : (Page %p/%P - %d/%M/%y - %h:%m:s)");
```

- IlvPrintableGraphic は、IlvGraphic のインスタンスをカプセル化します。あらゆる IlvGraphic オブジェクトが印刷できます。

```
// declares a printable graphic
IlvGraphic* ellipse = new IlvFilledEllipse(display,
                                           IlvRect(0, 0, 100, 50));
IlvPrintableGraphic* printgraphic = new IlvPrintableGraphic(ellipse);
```

- IlvPrintableFrame は、簡単な矩形をカプセル化します。

```
// declares a printable frame
IlvPrintableFrame* printframe = new IlvPrintableFrame
                                          (display->defaultPalette());
```

- IlvPrintableManager、IlvPrintableMgrView、および IlvPrintableManagerLayer (マネージャ・パッケージでのみ使用可能) は、マネージャ全体、マネージャ・ビュー、マネージャ・レイヤをそれぞれ印刷します。
- IlvPrintableComposite を使うと、複数の印刷可能オブジェクトを構成して定義できます。

IlvPrintableLayout クラス

IlvPrintableLayout は、ページ・レイアウトを記述するベース・クラスの抽象クラスです。これは左、右、上、下、溝マージンを指定して印刷可能領域を定義します。

また印刷可能領域内に 5 つのサブ領域を定義して、印刷可能オブジェクトに関連付けます。

- メイン印刷可能オブジェクトの印刷に使用するメイン領域。

  印刷可能オブジェクトを印刷可能領域に伸張したり、印刷可能オブジェクトの縦横比を維持する選択ができます。
- ヘッダー印刷可能オブジェクトの印刷に使用するヘッダー領域。
- フッター印刷可能オブジェクトの印刷に使用するフッター領域。
- メイン領域の背景の印刷に使用する背景領域。
- メイン領域の前景の印刷に使用する前景領域。

ヘッダー領域およびフッター領域のサイズは特定できます。

定義済みレイアウトは次のとおりです。

- IlvPrintableLayoutOnePage は、印刷可能オブジェクトを 1 ページにレイアウトします。このレイアウトでは、印刷可能オブジェクトを単一ページでレンダリングします。
- IlvPrintableLayoutMultiplePages は、印刷可能オブジェクトを数ページにレイアウトします。ページ・マトリックスのサイズはユーザが定義します。

  このレイアウトは、複数ページにわたる仮想ページを定義します。ヘッダー領域は仮想ページの上部を定義し、フッター領域は下部を定義します。
- IlvPrintableLayoutIdentity を使用すると、印刷可能オブジェクトと同じサイズで印刷文書を定義します。

  このレイアウトは IlvPrintableLayoutMultiplePages からの継承で、ページ数を必要なだけ使用します。
- IlvPrintableLayoutFixedSize を使うと、印刷文書のサイズを選択できます。

  このレイアウトは IlvPrintableLayoutMultiplePages からの継承で、ページ数を必要なだけ使用します。

IlvPrinter クラス

IlvPrinter クラスは、用紙サイズ、用紙の向き、物理的マージンなどプリンタの物理的特性を記述します。これは、IlvPort のインスタンスをカプセル化します。

このクラスは抽象クラスで、2 つの定義済みサブクラスがあります。

- IlvPSPrinter を使うと、PostScript ファイルを印刷できます。

```
// creating a PostScript printer
IlvPSPrinter* psprinter = new IlvPSPrinter(display);
psprinter->setPaperFormat(*IlvPaperFormat::Get("A3"));
psprinter->setOrientation(IlvPrinter::Landscape);
psprinter->setDocumentName("viewsprint.ps");
```

- IlvWindowsPrinter は、Windows コンピュータに接続されたプリンタでの印刷を可能にします (このクラスは Windows でのみ利用可能)。

  特性の中には用紙サイズやマージンのようにプリンタに依存するものがあり、これらは設定できません。

```
// creating a Windows printer
IlvWindowsPrinter* wprinter = new IlvWindowsPrinter(display);
```

IlvPrintUnit クラス

IlvPrintUnit クラスを使うと、サイズの単位を記述できます。さまざまなタイプの単位を変換できます。

もっとも一般的に使用される 4 つの単位が定義されています。

- IlvPrintPointUnit は、単位をセンチメートルで表します。これは参照単位です。
- IlvPrintCMUnit は、単位をセンチメートルで表します。
- IlvPrintInchUnit は、単位をインチで表します。
- IlvPrintPicaUnit は、単位をセンチメートルで表します。

このクラスは、主に IlvPSPrinter を使用する場合に役立ちます。

変換単位：

```
IlvPrintCMUnit oneMeter(100.0);
IlvPrintInchUnit oneMeterInInches(oneMeter);
IlvDim result = oneMeterInInches.getUnits();
```

IlvPaperFormat クラス

IlvPaperFormat クラスは用紙書式を記述します。用紙書式は登録でき、名前で問い合わせ可能です。

一般的に使用される用紙書式がいくつか定義されています。サイズは、必ず PostScript ポイントで指定します。

> ***メモ:*** *Windows プラットフォーム上で、*IlvWindowsPrinter *を使用する場合は、プリンタ・ドライバで用紙サイズが決まるため、このクラスは* IlvPSPrinter *でのみ使用します。*

用紙書式を取得します。

```
IlvPaperFormat* letterformat = IlvPaperFormat::Get("Letter");
```

新しい用紙書式を作成します。

```
IlvPrintCMUnit width(100.0);
IlvPrintCMUnit height(100.0);
IlvPaperFormat::Register("MyFormat", width.getPoints(), height.getPoints());
```

定義済みの用紙書式を、ï 13.1 に示します。

表 13.1 *定義済み用紙書式*

名前	幅 (ポイント)	高さ (ポイント)
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B4	709	1003
B5	516	729
C5	459	649
Quarto	610	780
Folio	612	936
Statement	396	612
Monarch	279	540
Executive	540	720
Ledger	1224	792
Tabloid	792	1224
Legal	612	1008
Letter	612	792

ダイアログ

Gadgets パッケージには、定義済みダイアログが用意されており、印刷ジョブのプレビューや PostScript 印刷機能が選択できます。

`IlvPostScriptPrinterDialog` クラスを使用すると、以下のようなさまざまな PostScript 印刷機能を選択できます。

- 出力ファイル名
- 向き
- 色モード
- 用紙書式
- 部単位印刷モード
- 印刷部数
- マージン

使用例 (図 13.1 を参照してください)。

```
IlvPostScriptPrinterDialog psdialog(display);
   psdialog.get();
   IlvPrinter::Orientation orientation = psdialog.getOrientation();
   IlBoolean collate = psdialog.isCollateOn();
```

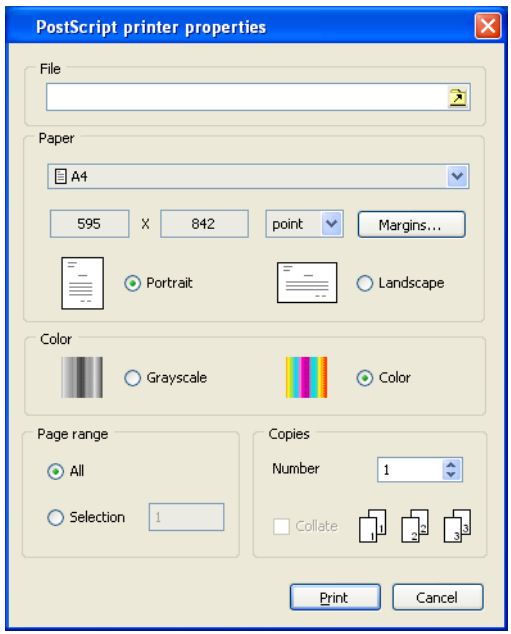


図 13.1 *PS プリンタの例*

IlvPrinterPreviewDialog クラスを使うと、印刷ジョブのプレビューができます (図 13.2 を参照してください)。これは以下のような各種モードをサポートしています。

- 1 ページ・プレビュー
- 2 ページ・プレビュー

◆ タイル・プレビュー

ズーム係数を指定することも可能です。

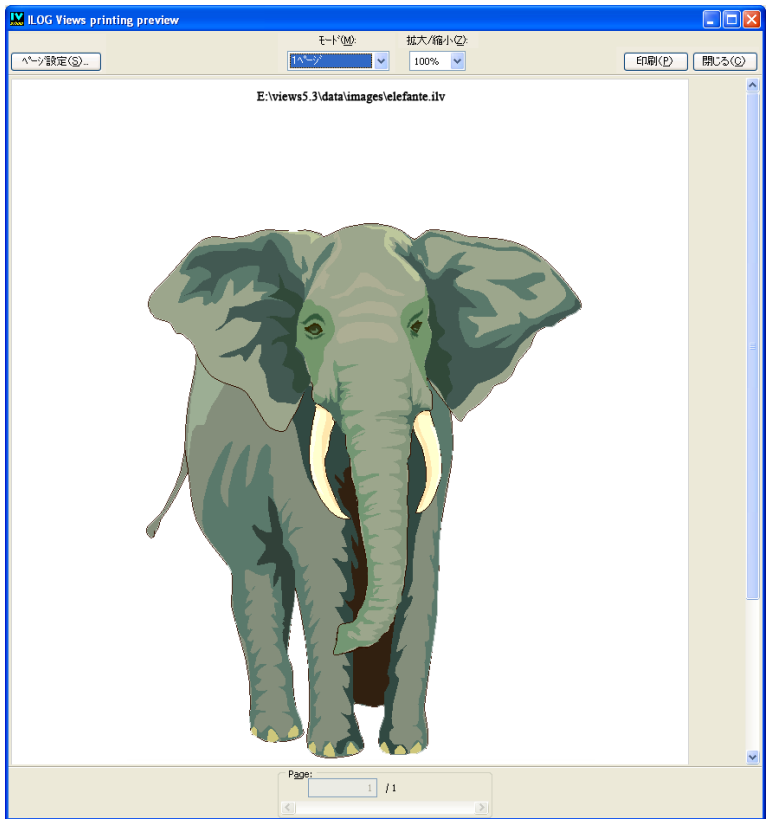


図 13.2 *印刷プレビュー例*

章

14

IBM ILOG Script プログラミング

この章は、IBM ILOG Script for IBM® ILOG® Views のプログラミング・ガイドとなっています。以下のトピックから構成されています。

- *IBM ILOG Script for IBM ILOG Views* の概要
- *IBM ILOG Views アプリケーションをスクリプト可能にする*
- *IBM ILOG Views オブジェクトの結合*
- *IBM ILOG Script モジュールの読み込み*
- *IBM ILOG Script コールバックの使用*
- *パネル・イベントの処理*
- *ランタイムに IBM ILOG Views オブジェクトを作成する*
- *IBM ILOG Views オブジェクトの共通プロパティ*
- *IBM ILOG Script for IBM ILOG Views でリソースを使用する*
- トピックの最後には、*スクリプト可能アプリケーション作成のガイドライン*およびスクリプトに使用できる *リソース名* の参照テーブルが添付されています。

IBM ILOG Script の構文についての詳細は、付録 F、*IBM ILOG Script 2.0 言語リファレンス*を参照してください。

IBM ILOG Script for IBM ILOG Views

IBM ILOG Script for IBM® ILOG® Views は、高機能グラフィック・アプリケーション開発用のオブジェクト指向スクリプト言語です。

IBM ILOG Script for IBM ILOG Views は、IBM ILOG Views の高機能グラフィック・オブジェクトのほとんどにアクセスできる JavaScript™ スクリプト言語を IBM ILOG が実装した IBM ILOG Script の改良版です。

IBM ILOG スクリプト機能の詳細については、以下を参照してください。

- この章では、IBM ILOG Script を使用した IBM ILOG Views グラフィック・オブジェクトのプログラム方法について説明します。

  > ***メモ:*** *ここではオプションのパネルやガジェットのプログラムに関する資料を扱います。これらのオプションの詳細は、該当する『IBM ILOG Views』マニュアルのパッケージを参照してください。*

- IBM ILOG Script *マニュアル*では、`ivfstudio` から IBM ILOG Script for IBM ILOG Views を使用する方法について説明しています。IBM ILOG Script で IBM ILOG Views アプリケーションを作成できるようにする IBM ILOG Views Studio の拡張についても扱います。
- *IBM ILOG Views Foundation リファレンス・マニュアル*には、IBM ILOG Script for IBM ILOG Views がサポートしている IBM ILOG Views オブジェクトに関するすべての情報が記載されています。

IBM ILOG Views アプリケーションをスクリプト可能にする

IBM ILOG Views アプリケーションで IBM ILOG Script を使用するには、このアプリケーションを*スクリプト可能*にしなければなりません。IBM ILOG Script for IBM ILOG Views のインタプリタは、C++ ライブラリとして実装されています。したがって、IBM ILOG Script for IBM ILOG Views を IBM ILOG Views アプリケーションで使用する場合は、以下の操作を行います。

- *ヘッダー・ファイルの追加*に説明されているように、適切なヘッダー・ファイルをアプリケーションのソース・ファイルに追加します。

- *IBM ILOG Script for IBM ILOG Views ライブラリへのリンク*に説明されているように、提供されている IBM ILOG Script for IBM ILOG Views ライブラリを使用してアプリケーションをリンクします。

> ***メモ:*** *IBM ILOG Script で IBM ILOG Views アプリケーションを作成できる IBM ILOG Views Studio の拡張から IBM ILOG Views のスクリプト可能なアプリケーションを生成することもできます。詳細については、『IBM ILOG Views Studio ユーザ・マニュアル』を参照してください。*

ヘッダー・ファイルの追加

次のヘッダー・ファイルを、アプリケーションのメイン・ソース・ファイルに追加します。

```
#include <ilviews/jvscript/script.h>
```

このファイルの追加は 1 度だけ行います。アプリケーションの各ソース・ファイルに追加する必要はありません。

IBM ILOG Script for IBM ILOG Views ライブラリへのリンク

IBM ILOG Views ライブラリの他に、アプリケーションを次の IBM ILOG Script for IBM ILOG Views ライブラリとリンクさせる必要があります。

Microsoft Windows の場合

- `ilvjs.lib`
- `iljs.lib`
- `iljsgide.lib`

UNIX の場合

- `libilvjs`
- `libiljs`
- `libiljsgide`

IBM ILOG Views オブジェクトの結合

ガジェットやパネルなどの IBM® ILOG® Views オブジェクトを IBM ILOG Script で使用するには、結合の手順に従ってこれらのオブジェクトをアクセス可能にします。オブジェクトを結合するには以下の操作を行います。

- まず、IBM ILOG Script コンテキストを取得します。この詳細については、*グローバル IBM ILOG Script コンテキストの取得*を参照してください。
- *IBM ILOG Views オブジェクトの結合*で説明されている `bind` メソッドを呼び出します。

結合オブジェクトが IBM ILOG Script からアクセス可能になります。

グローバル IBM ILOG Script コンテキストの取得

IBM ILOG Script コンテキストは IBM® ILOG® Views とスクリプト言語の間のゲートウェイで、IBM ILOG Views オブジェクトを結合する前に作成する必要があります。前セクションで説明したとおり、アプリケーションのヘッダー・ファイルに `script.h` を追加してそのアプリケーションを適切なライブラリにリンクした場合、グローバル・コンテキストは自動的に作成されます。

このコンテキストを有効にするには、以下の関数を呼び出します。

```
IlvScriptLanguage* jvscript = IlvScriptLanguage::Get("JvScript");
IlvScriptContext* theContext = jvscript->getGlobalContext();
```

この関数は、グローバル IBM ILOG Script コンテキストへのポインタを返します。

IBM ILOG Views オブジェクトの結合

IBM® ILOG® Views オブジェクトを結合するには、以下の関数を呼び出します。

```
IlvScriptContext::bind(IlvValueInterface* object,
                       const char* name);
```

この関数は結合させるオブジェクトへのポインタを最初のパラメータとし、オブジェクトを結合させる文字列を 2 つ目のパラメータとします。IBM ILOG Script のプログラマはこの名前を使用して、関連付けられたオブジェクトにアクセスできます。ポインタのタイプは `IlvValueInterface` で、これはほとんどの IBM ILOG Views クラスのスーパークラスです

したがって、`IlvApplication` オブジェクトを次のコードで結合できます。

```
IlvScriptLanguage* jvscript = IlvScriptLanguage::Get("JvScript);
IlvScriptContext* theContext = jvscript->getGlobalContext();
theContext->bind(theApp, "Application");
// theApp is the pointer to an IlvApplication
```

上記のコードは `Application` シンボルを使用して、`IlvApplication` オブジェクトを IBM ILOG Script に結合します。この結果、`Application` に付加されたプロパティに IBM ILOG Script からアクセスできるようになります。

```
var name = Application.name;
```

IBM ILOG Views Script の IBM ILOG Views オブジェクトへアクセスする

アプリケーションにあるすべての IBM ILOG Views オブジェクトを結合する場合もあります。これを行う最適の方法は、ルート・オブジェクトだけを結合することです。これにより直接、間接を問わず、そのオブジェクトを起点とするその他の IBM ILOG Views オブジェクトのほとんどにアクセスできます。

たとえば、ivfstudio から生成した IBM ILOG Views アプリケーションで、IlvApplication::getPanel メソッドを呼び出し、アプリケーションへのポインタを使用してパネルへのポインタにアクセスできます。同様に、IlvContainer::getObject メソッドを呼び出してパネルのガジェットにアクセスできます。このため、このようなアプリケーションで結合する唯一のオブジェクトは IlvApplication オブジェクトとなります。

アプリケーション・オブジェクト

ivfstudio で生成したアプリケーションでは、IlvApplication オブジェクトは Application シンボルに結合します。Application オブジェクトを起点とするその他の IBM ILOG Views オブジェクトすべてにアクセスできます。

アプリケーションに myPanel というパネルが 1 つ含まれていると想定します。以下のようにして、IBM ILOG Script でパネルにアクセスできます。

```
var panel = Application.getPanel("myPanel");
```

このタイトルを変更するには、以下のように入力します。

```
panel.title = "New title";
```

アプリケーションのパネルに myButton というボタンが含まれている場合、次のコードを使用してアクセスします。

```
var button = panel.getObject("myButton");
```

ボタンのラベルを変更するには、以下のように入力します。

```
button.label = "A new label";
```

パネルおよびガジェットへのアクセス

以下はアプリケーションのパネルとガジェットへの簡単なアクセス方法です。

IBM ILOG Script の myPanel というパネルにアクセスするには、次のように入力します。

```
var panel = Application.myPanel;
```

このタイトルを変更するには、以下のように入力します。

```
Application.myPanel.title = "A new title";
```

パネルのガジェットにアクセスするには、以下のように入力します。

```
var button = panel.myButton;
```

この方法でアクセスできるのは、通常の名前が付いたパネルおよびガジェットのみです。パネル名またはガジェット名に、&、+、-、=、空白などの特殊文字が使用されている場合、上記の方法ではアクセスできません。これらの文字をパネル名やガジェット名に使用しないように注意してください。

IBM ILOG Script モジュールの読み込み

スクリプト可能な IBM ILOG Views アプリケーションに読み込めるスクリプトは、以下の 3 種類です。

- *インライン・スクリプト*
- *IBM ILOG Script のデフォルト・ファイル*
- *IBM ILOG Script の独立ファイル*

*IBM ILOG Script のスタティック関数*で説明されているように、これらのスクリプトで定義されるスタティック関数の機能は限られています。

インライン・スクリプト

グラフィック・パネルの作成中に `ivfstudio` から作成するスクリプトは、インライン・スクリプトと呼ばれます。これらのスクリプトは、`ivfstudio` によって `.ilv` ファイルに保存されています。インライン・スクリプトは、スクリプトが格納される `.ilv` ファイルと同時に、ファイルが IBM ILOG Views のスクリプト可能アプリケーションに読み込まれるときにロードされます。参照：*IBM ILOG Views アプリケーションをスクリプト可能にする*

IBM ILOG Script のデフォルト・ファイル

インライン・スクリプトを含まない `panel.ilv` のような `.ilv` ファイルを IBM ILOG Views のスクリプト可能なアプリケーションに読み込む際に、IBM ILOG Views はそのファイルがあるディレクトリを検索し、同じ名前で `.js` 拡張子を持つ IBM ILOG Script ファイルがあるかどうかを確認し、自動的にロードします。`panel.js` ファイルは `panel.ilv` の IBM ILOG Script のデフォルト・ファイルと呼ばれます。

IBM ILOG Script の独立ファイル

`IlvScriptContext::loadScript` メソッドを使用して、IBM ILOG Views のスクリプト可能なアプリケーションに IBM ILOG Script の独立モジュールを読み込めます。

```
IlvScriptContext::loadScript("c:\\myscripts\\myscript.js");
```

このメソッドを使用して、いくつかのアプリケーションで共有できる IBM ILOG Script ファイルを読み込めます。

IBM ILOG Script のスタティック関数

インライン・スクリプトおよびデフォルト・スクリプトは、.ilv ファイルに関連付けられています。これらのスクリプトで定義されたスタティック関数の名前は、範囲が限られています。

- IBM ILOG Script のスタティック関数の名前の範囲は、それが定義されるモジュールに限られます。これは他の IBM ILOG Script モジュールからは見えなくなっています。
- これは関連付けられた .ilv ファイルにあるガジェットおよび関連する .ilv ファイルで定義されたガジェット・コンテナのサブパネルにあるガジェットがあれば、そのコールバックとして使用されるだけです。

以下はスタティック関数の例です。

```
static function OnClick(graphic)
{
    graphic.foreground = "red";
}
```

IBM ILOG Script コールバックの使用

IBM ILOG Views ガジェットは、特定のマウス・イベントまたはキーボード・イベントを認識できます。これらのイベントが適用されると、関連付けられた定義済み IBM ILOG Script コールバック関数を呼び出します。

コールバックを使用してガジェット・イベントを処理するには、以下の操作を行います。

- コールバック関数を作成します。*コールバックの作成*を参照してください。
- コールバックを設定します。*IBM ILOG Script コールバックの設定*を参照してください。

コールバックの作成

IBM ILOG Script for IBM ILOG Views では、コールバックは次の署名を備えています。

```
function CallBack(gadget, value)
```

以下は、コールバックの例です。

```
function OnButtonClick(button, value)
{
    button.foreground = "red";
    writeln("The additional value is:" + value);
}
```

コールバックに渡される 2 つ目の引数は、コールバックを ivfstudio で設定する際に指定できるオプションの値です。したがって、コールバック関数の内容は次のようになります。

```
function OnGadgetClick(gadget)
{
    gadget. foreground = "red";
}
```

IBM ILOG Script コールバックの設定

コールバックをガジェットに設定するには、次の 2 つの方法があります。

- パネルを設計する際に IBM ILOG Views Studio でコールバックを設定します。これはもっとも簡単な方法です。
- `IlvGraphic::setCallback` メソッドでコールバックを設定します。このメソッドでは、コールバックをガジェットに設定するか、またはランタイムにコールバックを変更します。以下に例を示します。

```
myGadget.setCallback("Generic",  "myCallback", "JvScript");
```

  最初の引数は、処理するイベントを識別するコールバック・タイプです。2 つ目の引数は定義したコールバック関数です。3 つ目の引数は常に `JvScript` です。

パネル・イベントの処理

IBM ILOG Script 関数を使用して、アプリケーションのパネルが作成、表示、非表示、削除されたかどうかを通知することができます。これらのイベントを処理するには、以下を使用します。

- *OnLoad 関数*
- *onShow プロパティ*
- *onHide プロパティ*
- *onClose プロパティ*

OnLoad 関数

IlvContainer オブジェクトを作成したら、IBM ILOG Script 関数である OnLoad を検索して呼び出し、コンテナがその引数として渡されます。OnLoad 関数がいくつかある場合は、コンテナは以下の順でモジュール内を探し、最初に見つけた OnLoad 関数を呼び出します。

1. インライン・スクリプト・モジュール
2. IBM ILOG Script のデフォルト・モジュール
3. その他の IBM ILOG Script モジュール

IBM ILOG Script 関数の OnLoad は、次の署名を備えています。

```
function OnLoad(theContainer)
{
     // Initialization code
}
```

OnLoad 関数は通常、アプリケーションにパネルが作成された後、初期化の実行に使用されます。

onShow プロパティ

IlvContainer はIBM ILOG Script 関数に渡すことができる onShow プロパティを備えています。画面でコンテナが表示されると、指定された関数が呼び出されます。

例 :

```
function OnShow(theContainer)
{
    writeln("Hi, " + theContainer.name + " is displayed.");
}

function OnLoad(theContainer)
{
   theContainer.onShow = OnShow;
}
```

この例の場合、OnShow 関数は theContainer に適用される onShow イベントを処理するのに使用されます。

onHide プロパティ

onHide プロパティは、コンテナが非表示になったときに指定した関数が呼び出されることを除いては、onShow プロパティと類似しています。

例 :

```
function WhenPanelHides(theContainer)
{
    writeln("Hi, I am " + theContainer.name + ", see you later.");
}

function OnLoad(theContainer)
{
   theContainer.onHide = WhenPanelHides;
}
```

onClose プロパティ

onClose プロパティは、コンテナが閉じられているときに指定した関数が呼び出されることを除いては、onShow プロパティと類似しています。

例：

```
function OnClose(theContainer)
{
    writeln("Hi, " + theContainer.name + " has terrible news ...");
}

function OnLoad(theContainer)
{
   theContainer.onClose = OnClose;
}
```

ランタイムに IBM ILOG Views オブジェクトを作成する

IBM ILOG Script for IBM® ILOG® Views では、new オペレータを使用して、文字列、数字など他の IBM ILOG Script ネイティブ・オブジェクトを作成するように、ランタイムに IBM ILOG Views オブジェクトを作成できます。

以下のタイプのオブジェクトをランタイムに作成することができます。

- IlvPoint
- IlvRect
- IlvGadgetContainer

IlvButton などのガジェットを作成するには、IBM ILOG Views Studio の使用をお勧めします。

IlvPoint と IlvRect

一部の IBM ILOG Views オブジェクト・メソッドは、引数として IlvPoint または IlvRect を使用します。これらは、ランタイムに下記のように作成できます。

```
var myPoint = new IlvPoint(20, 20);
myPanel.move(myPoint);
```

IlvGadgetContainer

ランタイムに新しいパネルを作成できます。以下に例を示します。

```
var size = new IlvRect(20, 20, 400, 300);
var myNewPanel = new IlvGadgetContainer("Panel", "My panel", size);
myNewPanel.readFile("panel.ilv");
myNewPanel.readDraw();
Application.addPanel(myNewPanel);
```

新しいパネルを作成した後、Application に追加することをお勧めします。

詳細は、*Gudgets アクセサ・リファレンス・マニュアル*の「IlvGadgetContainer クラスのアクセサ」を参照してください。

IBM ILOG Views オブジェクトの共通プロパティ

スクリプトに有用な以下のプロパティは、すべての IBM® ILOG® Views オブジェクトに共通します。

- *className*
- *name*
- *help*

className

className は、オブジェクトのタイプを示す読み取り専用の文字列です。オブジェクトのタイプについては、*IBM ILOG Views Foundation リファレンス・マニュアル*を参照してください。

name

name はオブジェクトを識別する文字列です。アプリケーション内のパネルには固有の名前が必要です。コンテナ内のガジェットには固有の名前が必要です。

help

help は、オブジェクトを説明する読み取り専用の文字列です。このプロパティは、スクリプト可能な IBM ILOG Views アプリケーションをデバッグする際に非常に有効です。

たとえば、IBM ILOG Views Studio スクリプト・デバッガで、Application.help と入力してサポートされているプロパティとメソッドのリストを以下のように取得します。

```
> Application.help
= ViewsObject :
Method getPanel;
Method addPanel;
Method removePanel;
Method setState;
Method quit;
Object rootState;
String name;
String className;
```

IlvApplication::getPanel メソッドの詳細を取得するには、Application.getPanel.help と入力します。

```
> Application.getPanel.help
= Object getPanel(String name)
```

IlvApplication::getPanel メソッドの詳細は、Application.getPanel と入力しても取得できます。

```
> Application.getPanel
= [Views method:Object getPanel(String name)]
```

IBM ILOG Script for IBM ILOG Views でリソースを使用する

色、ビットマップ、フォントなどの IBM ILOG Script for IBM® ILOG® Views のリソースは、名前または文字列で識別されます。以下のセクションでは、これらの使い方について説明します。

- *IBM ILOG Script for IBM ILOG Views でリソース名を使用する*
- *IBM ILOG Script for IBM ILOG Views でビットマップを使用する*
- *IBM ILOG Script for IBM ILOG Views でフォントを使用する*

IBM ILOG Script for IBM ILOG Views でリソース名を使用する

任意の IBM ILOG Views オブジェクト (色、パターン、線画、塗りつぶし、円弧モードなど) と関連付けられているリソースを変更するには、その名前を使用します。次にいくつかの例を示します。

```
myButton.foreground = "red";
myButton.pattern = "solid";
myLabel.alignment = "right";
```

リソース名については、*リソース名*を参照してください。

IBM ILOG Script for IBM ILOG Views でビットマップを使用する

ビットマップは、名前で識別されます。IBM® ILOG® Views ビットマップを変更するには、以下の例に示されるようにその名前を使用します。

```
myButton.bitmap = "ilog.ic";
myPanel.backgroundBitmap = "subdir/mybmp.gif";
```

指定されたビットマップは、ILVPATH で定義されたディレクトリに格納されます。ビットマップが別のディレクトリにある場合は、完全なアクセス・パスを示します。

```
myButton.bitmap = "/mybmps/myicon.gif";
```

IBM ILOG Script for IBM ILOG Views でフォントを使用する

IBM ILOG Script for IBM® ILOG® Views では、フォントは通常、次の形式で文字列によって識別されます。

```
%fontName-fontSize-fontFlags
```

fontName は、Courier、Helvetica、Times などのフォント・ファミリの名前です。fontSize はフォント・サイズを表す整数です。fontFlags はフォント・スタイルを表す文字一式です。B は太字、I は斜体、U は下線を表します。フォントを標準表示にする場合は、このフィールドを空欄にしておいてください。

たとえば、IlvLabel のフォントを変更する場合は、次のように入力します。

```
myLabel.font = "%times-16-I";
```

スクリプト可能アプリケーション作成のガイドライン

スクリプト可能 IBM® ILOG® Views アプリケーションを新しく作成する、または既存のアプリケーションをスクリプト可能にするには、以下のガイドラインに従います。

1. IlvApplication クラスまたは派生クラスのオブジェクトをアプリケーションのルート・オブジェクトとして使用します。IlvApplication オブジェクトを作成したら、Application という名前を使用して結合します。
2. すべてのパネルを IlvApplication オブジェクトに追加して、これらを IBM ILOG Script for IBM ILOG Views の Application オブジェクトからアクセス可能にします。
3. IlvDisplay を作成したら、次のコードを使用して IBM ILOG Script for IBM ILOG Views 補助ライブラリを初期化します。

```
IlvJvScriptLanguage::InitAuxiliaryLib(appli->getDisplay());
```

IBM ILOG Script for IBM ILOG Views で `IlvCommonDialog` オブジェクトを使用する場合、または `IlvPoint`、`IlvRect`、または `IlvGadgetContainer` オブジェクトを作成する場合、必ず初期化を行ってください。

リソース名

このセクションには、IBM ILOG Script for IBM® ILOG® Views のリソース名のリストが記載されています。

表 14.1 *色名*

Color Name	RGB Definition
aliceblue	240, 248, 255
antiquewhite	250, 235, 215
aquamarine	127, 255, 212
azure	240, 255, 255
beige	245, 245, 220
bisque	255, 228, 196
black	0, 0, 0
blanchedalmond	255, 235, 205
blue	0, 0, 255
blueviolet	138, 43, 226
brown1	65, 42, 42
burlywood	222, 184, 135
cadetblue	95, 158, 160
chartreuse	127, 255, 0
chocolate	210, 105, 30
coral	255, 127, 80
cornflowerblue	100, 149, 237
cornsilk	255, 248, 220
cyan	0, 255, 255
darkgoldenrod	184, 134, 11
darkgreen	0, 100, 0
darkkhaki	189, 183, 107
darkolivegreen	85, 107, 47
darkorange	255, 140, 0
darkorchid	153, 50, 204
darksalmon	233, 150, 122
darkseagreen	143, 188, 143
darkslateblue	72, 61, 139
darkslategray	47, 79, 79
darkslategrey	47, 79, 79
darkturquoise	0, 206, 209
darkviolet	148, 0, 211
deeppink	255, 20, 147
deepskyblue	0, 191, 255
dimgray	105, 105, 105
dimgrey	105, 105, 105
dodgerblue	30, 144, 255
firebrick	178, 34, 34
floralwhite	255, 250, 240
forestgreen	34, 139, 34
gainsboro	220, 220, 220
ghostwhite	248, 248, 255
gold	255, 215, 0
goldenrod	218, 165, 32
gray	192, 192, 192
green	0, 255, 0
greenyellow	173, 255, 47
grey	192, 192, 192
honeydew	240, 255, 240
hotpink	255, 105, 180
indianred	205, 92, 92
ivory	255, 255, 240
khaki	240, 230, 140
lavender	230, 230, 250
lavenderblush	255, 240, 245

```
lawngreen          124, 252, 0
lemonchiffon       255, 250, 205
lightblue          173, 216, 230
lightcoral         240, 128, 128
lightcyan          224, 255, 255
lightgoldenrod     238, 221, 130
lightgoldenrod     250, 250, 210
lightgray          211, 211, 211
lightgrey          211, 211, 211
lightpink          255, 182, 193
lightsalmon        255, 160, 122
lightseagreen      32, 178, 170
lightskyblue       135, 206, 250
lightslateblue     132, 112, 255
lightslategray     119, 136, 153
lightslategrey     119, 136, 153
lightsteelblue     176, 196, 222
lightyellow        255, 255, 224
limegreen          50, 205,  50
linen              250, 240, 230
magenta            255,  0, 255
maroon             176,  48,  96
mediumaquamarine   102, 205, 170
mediumblue         0, 0, 205
mediumorchid       186, 85, 211
mediumpurple       147, 112, 219
mediumseagreen     60, 179, 113
mediumslateblue    123, 104, 238
mediumspringgreen  0, 250, 154
mediumturquoise    72, 209, 204
mediumvioletred    199,  21, 133
midnightblue       25, 25, 112
mintcream          245, 255, 250
mistyrose          255, 228, 225
moccasin           255, 228, 181
navajowhite        255, 222, 173
navy               0, 0, 128
navyblue           0, 0, 128
oldlace            253, 245, 230
olivedrab          107, 142,  35
orange             255, 165, 0
orangered          255,  69, 0
orchid             218, 112, 214
palegoldenrod      238, 232, 170
palegreen          152, 251, 152
paleturquoise      175, 238, 238
paleviolet         219, 112, 147
papayawhip         255, 239, 213
peachpuff          255, 218, 185
peru               205, 133,  63
pink               255, 192, 203
plum               221, 160, 221
powderblue         176, 224, 230
purple             160,  32, 240
red                255, 0, 0
rosybrown          188, 143, 143
royalblue          65, 105, 225
saddlebrown        139,  69,  19
```

salmon	250, 128, 114
sandybrown	244, 164, 96
seagreen	46, 139, 87
seashell	255, 245, 238
sienna	160, 82, 45
skyblue	135, 206, 235
slateblue	106, 90, 205
slategray	112, 128, 144
slategrey	112, 128, 144
snow	255, 250, 250
springgreen	0, 255, 127
steelblue	70, 130, 180
tan	210, 180, 140
thistle	216, 191, 216
tomato	255, 99, 71
turquoise	64, 224, 208
violet	238, 130, 238
violetred	208, 32, 144
wheat	245, 222, 179
white	255, 255, 255
whitesmoke	245, 245, 245
yellow	255, 255, 0
yellowgreen	154, 205, 50

表 14.2 *方向*

left
right
top
bottom
topLeft
bottomLeft
topRight
bottomRight
center
horizontal
vertical

表 14.3 *円弧モード*

ArcPie
ArcChord

表 14.4 *塗りつぶしルール*

EvenOddRule
WindingRule

表 14.5 *塗りつぶしスタイル*

FillPattern
FillColorPattern
FillMaskPattern

表 14.6 *パターン*

solid
clear
diaglr
diagrl
dark1
dark2
dark3
dark4
light1
light2
light3
light4
gray
horiz
vert
cross

表 14.7 *線の種類*

solid
dot
dash
dashdot
dashdoubledot
alternate
doubledot
longdash

章

15

国際化

IBM® ILOG® Views では、ソフトウェアの国際化バージョンを作成できます。この章は、以下のトピックに分かれています。

- *i18n とは？*では国際化の概要を説明します。
- *ローカライズされた環境のチェックリスト*では、プログラムの作成と実行の要件について説明します。ロケール、フォント、ローカライズされたメッセージ・データベース・ファイルについての詳細も扱います。作成中にローカライズされたメッセージがシステムに表示されない場合は、*トラブルシューティング*のチェックリストが問題解決の役に立ちます。
- *極東アジア言語で IBM ILOG Views を使用する*では、マルチバイト文字言語に関する事項について説明します。
- *データ入力要件*
- *国際化機能の制限*
- *リファレンス：エンコーディング・リスト*は IBM ILOG Views にサポートされているエンコーディングです。また、*リファレンス：各プラットフォームでサポートされているロケール*には、多数の表が掲載されています。

> ***メモ：** この章では、IBM ILOG Views の国際化機能の使い方を説明します。国際化ソフトウェアの作成方法の詳細は、このテーマに関する概説書を参照してください。*

i18n とは？

国際化 (または一般的に「i18n」と略される) は、ユーザがそのネイティブ言語を使用してソフトウェアと対話できるソフトウェア設計方法です。国際化ソフトウェアはデータを処理して、ユーザの言語ルールが優先されるようにします。ユーザがソフトウェアに求める要件は、以下の通りです。

- 使用言語文字の入力、処理、表示ができる。
- 使用言語でシステムと対話できる。プロンプトおよびエラー・メッセージが使用言語で表示される。
- データのフォーマットや処理は、ユーザのローカル・ルールおよび環境に基づく。

ロケール

IBM ILOG Views は、POSIX ロケール・モードに基づいて i18n をサポートしています。*ロケール*とは、データおよび / またはメソッドの集合で、これにより、国際化された C ライブラリ機能およびシステムに依存したライブラリ機能をユーザの言語、ローカルの慣習、およびデータのエンコーディングに適合させることができます。ロケールによって言語の表示に使用する文字やフォントが決まります。また、日付、時刻、通貨および数字をプログラムがどのように表示またはソートするかも決まります。

ローカライズされた環境のチェックリスト

プログラムをローカル言語で使用する前に、それが適切な言語で動作するかどうか下記の項目を確認してください。

- プログラムはローカライズされた環境で動作するように作成すること。*ローカライズされた環境で実行するプログラムの作成*を参照してください。
- システムがロケール (使用言語) をサポートしていること。*ロケール要件*を参照してください。
- IBM® ILOG® Views が使用言語をサポートしていること。*IBM ILOG Views ロケールのサポート*を参照してください。
- 言語の表示に必要なフォントが、システムにインストールされていること。*必要なフォント*を参照してください。
- メッセージおよびその他のシステム・テキスト (`.dbm` ファイル) を含むファイルがローカル言語に翻訳され、正しいサブディレクトリで利用可能であること。*IBM ILOG Views のローカライズ・メッセージ・データベース*を参照してください。

これらの要件が満たされると、ローカライズ・ソフトウェアが実行可能になります。

ローカライズされた環境で実行するプログラムの作成

国際化環境で使用するプログラムを作成する場合、その他のプログラムで通常行うのと同様にコーディングします。ただし、プログラムの最初に必ず `IlvSetLocale` グローバル関数を呼び出します。この呼び出しは `IlvDisplay` のインスタンスを作成する前に行ってください。`IlvSetLocale` 呼び出しは、IBM® ILOG® Views がデフォルトのロケール環境で正しく実行するための基盤情報を設定するために必要です。

> ***メモ:*** *プログラムに* `IlvSetLocale` *の呼び出しがない場合、ローカライズされたメッセージは画面に表示されず、マルチバイト・サポートは無効になります。プログラムは C ロケールで実行するように振る舞うため、英語のメッセージのみが表示されます。*

次の例は、国際化環境ですぐに使用できる、さまざまな言語で実行可能な簡単なプログラムです。`IlvSetLocale()` がプログラムの最初に呼び出されていることに注目してください。

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/setLocale.cpp
// ------------------------------------------------------------------------
//  Copyright (C) 1990-2000 by ILOG.
//  All Rights Reserved.
// ------------------------------------------------------------------------

#include <ilviews/gadgets/gadcont.h>
#include <ilviews/gadgets/textfd.h>
#include <stdio.h>

static void
Quit(IlvView*, IlAny)
{
    IlvExit(0);
}

int main(int argc, char* argv[])
{
    if (!IlvSetLocale()) {
printf(OFalling back to the C locale.\nO);
    }

    IlvDisplay* display = new IlvDisplay(OTestO, 0, argc, argv);
    IlvRect rect(20,20,250,80);
    IlvGadgetContainer* cont = new IlvGadgetContainer(display, OContainerO,
OContainerO, rect);
    cont->setDestroyCallback(Quit, 0);
    IlvRect rect1(10,10,220,50);

    IlvTextField* tf = new IlvTextField(display, OThis is a text field.O,
rect1);
    cont->addObject(tf);
    IlvMainLoop();

    return 0;
}
```

ロケール要件

ロケールはシステムにサポートされている必要があります。ロケール要件についての詳細を、以下のトピックに分けて説明します。

- *システムのロケール要件を確認する*
- *ロケール名形式*
- *現在のデフォルト・ロケール*
- *現在のデフォルト・ロケールを変更する*

システムのロケール要件を確認する

システムがロケール要件を満たしているかどうかは、以下のように確認します。

- オペレーティング・システムが必要なロケールをサポートしているかどうかをシステム管理者に問い合わせます。オペレーティング・システムがロケールをサポートしていない場合、ローカライズ・プログラムは実行できません。
- システムがロケールをサポートしているかどうかは、使用システムにより次のように確認できます。
  - *UNIX システムの場合*
  - *X ライブラリのサポート (UNIX のみ)*
  - *Microsoft Windows システムの場合*

> ***メモ:*** *ロケール名はシステムに依存しています。システムに依存する名前の各例について、ここでは HP-UX (10.x または 11)、Solaris (2.6 または 2.7)、Windows のフランス語および日本語設定についてのみ扱います。*

UNIX システムの場合

次のユーティリティ・プログラムを実行して、システムがサポートするロケールのリストを取得します。

```
$ locale -a
```

(フランスで話されている) フランス語および日本語のみがサポートされている場合、HP-UX システムでは以下が取得できます。

```
fr_FR.iso88591
fr_FR.iso885915@euro
fr_FR.roman8
fr_FR.utf8
ja_JP.SJIS
ja_JP.eucJP
ja_JP.kana8
ja_JP.utf8
```

フランス語および日本語のみがサポートされている場合、Solaris システムでは以下が取得できます。

```
fr
fr.ISO8859-15
fr.UTF-8
fr_FR
fr_FR.ISO8859-1
fr_FR.ISO8859-15
```

```
fr.ISO8859-15@euro
fr_FR.UTF-8
fr_FR.UTF-8@euro
ja
ja_JP.eucJP
ja_JP.PCK
ja_JP.UTF-8
japanese
```

Microsoft Windows システムの場合

コントロール・パネルの「地域と言語のオプション」を確認します。

1. Windows デスクトップで、[スタート] > [設定] > [コントロール パネル] をクリックします。
2. 「地域と言語のオプション」アイコンをダブルクリックして、「地域と言語のオプションのプロパティ」ダイアログ・ボックスにアクセスします。
3. 「地域オプション」のノートブック・ページで、サポートされているロケールのリストが表示されます。

ロケール名形式

ここに示されるように、ロケール名はシステムに依存しています。ただし、ほとんどのシステムでは、XPG (X/Open Portability Guide) の命名規則に従っており、ここではローカル名は次の形式になっています。

`language_territory.encoding`
`language` は言語名です。`territory` は地域名です (1 つの言語が異なる地域や国で話されている場合があります。たとえば、フランス語はフランス、カナダ、ベルギー、スイスなどで話されています)。`encoding` は、コード・セット、すなわち文字をコード化するエンコード方式です。

UNIX システムの場合

以下の例は、さまざまな UNIX システムに表示されるロケール名の形式です。このロケールはフランスで話され、Latin1 のエンコーディングで表記されているフランス語を示します。

Solaris 8	`fr` または `fr_FR.iso8859-1`
HP-UX 11	`fr_FR.iso88591`
Red Hat Entreprise Linux 4.0	`fr_FR.iso88591`
Suze 10.0	`fr_FR`
AIX 5.1	`fr_FR` または `fr_FR.ISO8859-1`

Microsoft Windows システムの場合

以下は、Windows システムのロケール形式の例です。このロケールはフランスで使われ、Windows コード・ページ 1252 で表記されているフランス語を示します。

Windows XP	`French_France.1252`

現在のデフォルト・ロケール

システムにはデフォルト・ロケールが設定されます。通常、デフォルト・ロケールは使用する言語に設定します。システムの現在のデフォルト・ロケールを確認するには、次のプログラムのいずれかを実行します。

UNIX システムの場合

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/checkUnixLocale.cpp
// ------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// ------------------------------------------------------------------------

#include <locale.h>
#include <stdio.h>
#include <langinfo.h>
#if defined(linux) && !defined(CODESET)
#define CODESET _NL_CTYPE_CODESET_NAME
#endif /* linux */

int main()
{
    char* loc = setlocale(LC_ALL, "");
    if (loc) {
        printf("default locale: %s\n",loc);
        printf("encoding %s\n", nl_langinfo(CODESET));
    } else
        printf("System does not support this locale\n");
    return 0;
}
```

システムがフランス語に設定されている場合、HP-UX では以下が取得できます。

```
default locale: fr_FR.iso88591 fr_FR.iso88591 fr_FR.iso88591
fr_FR.iso88591 fr_FR.iso88591 fr_FR.iso88591
```

Solaris では、以下が取得できます。

```
default locale: fr
```

Microsoft Windows システムの場合

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                IlogViews userman source file
// File: doc/foundation/userman/src/internationalization/checkWindowsLocale.cpp
// ------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// ------------------------------------------------------------------------

#include <locale.h>
#include <stdio.h>
#include <windows.h>

int main(int argc, char* argv[])
{
    printf("default locale: %s\n", setlocale(LC_ALL, ""));
    printf("encoding %d\n", GetACP());
    return 0;
}
```

Windows XP で、地域設定がフランス語 (標準) になっている場合、以下が取得できます。

```
default locale: French_France.1252
encoding 1252
```

現在のデフォルト・ロケールを変更する

ローカライズされたメッセージを画面に表示するために、現在のデフォルト・ロケールを変更しなければならない場合があります。

UNIX システムの場合

次の環境変数のどちらかを使用できます。LANG または LC_ALL。適切なロケール名については、システムのマニュアルを参照してください。

たとえば、EUC エンコーディングで日本語を使用したい場合は次のようにします。

HP-UX の場合、以下を入力します。

```
$ LANG=ja_JP.eucJP
```

Solaris の場合、次を入力します。

```
$ LANG=ja or LANG=japanese
```

Microsoft Windows システムの場合

コントロール・パネルの「地域と言語のオプション」で言語を変更します。

X ライブラリのサポート (UNIX のみ)

X Window システムは適切な言語をサポートしている必要があります。適切な X ライブラリがシステムで利用可能かどうかを確認するには、次のプログラムを実行します。

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/checkXLocale.cpp
// ------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// ------------------------------------------------------------------------

#include <X11/Xlib.h>
#include <X11/Xlocale.h>
#include <stdlib.h>
#include <stdio.h>

int
main(int argc, char* argv[])
{
    char* loc = setlocale(LC_CTYPE, "");
    if (loc == NULL) {
        fprintf(stderr, "System does not support this locale.\n");
        exit(1);
    }
    if (!XSupportsLocale()) {
        fprintf(stderr, "X does not support locale %s.\n", loc);
        exit(1);
    }
    if (XSetLocaleModifiers("") == NULL) {
        fprintf(stderr, "Warning: cannot set locale modifiers for %s.\n", loc);
    } else
        fprintf(stderr, "Locale %s is supported by Xlib.\n", loc);
    exit(0);
}
```

たとえば、アラビア語がサポートされていない HP システムでは、LANG を ar_DZ.arabic8 に設定すると以下のように表示されます。

```
X does not support locale ar_DZ.arabic8.
```

IBM ILOG Views ロケールのサポート

ロケール名はシステムに依存していますが、各システムにはロケール情報を識別する独自の方法があります。IBM® ILOG® Views は、システムに依存しないスキームをローカライゼーション向けにサポートしています。

IBM ILOG Views ロケール名

IBM ILOG Views がロケールに依存した情報をシステムに依存しない方法で使用するために、IBM ILOG Views は、システムに依存しない名前を持つ *IBM ILOG Views ロケール*の概念を定義します。このロケールは以下の形式になっています。

```
ll_TT.encoding
```

ここで:

`ll` は、言語名 の小文字 2 文字の略語です。

`TT` は、地域名 の大文字 2 文字の略語です。

`encoding` は、使用されている コード・セットまたはエンコード・メャ b ドを識別する文字列です。

たとえば、IBM ILOG Views のロケール名が `fr_FR.ISO-8859-1` の場合、`fr` は言語名つまりフランス語を、`FR` は地域名つまりフランスを、`ISO-8859-1` は言語のエンコード方式つまり ISO 8859-1 を表します。

以下は、UNIX プラットフォーム上での IBM ILOG Views ロケール名の数例です。

- `fr_FR.ISO-8859-1`
- `de_DE.ISO-8859-1`
- `ja_JP.EUC-JP`
- `ja_JP.Shift_JIS`

以下は、Windows プラットフォーム上での IBM ILOG Views ロケール名の数例です。

- `fr_FR.windows-1252`
- `de_DE.windows-1252`
- `ja_JP.Shift_JIS`

言語名の仕様

IBM ILOG Views ロケールでは、ISO 639 言語名の表記規格コードで言語名が指定されています。以下に、例をいくつか示します。

- `en` (英語)
- `fr` (フランス語)
- `de` (ドイツ語)
- `ja` (日本語)

ISO 639 標準は、次の Web サイトで確認できます。

http://www.loc.gov/standards/iso639-2/ascii_8bits.html

または ftp://std.dkuug.dk/i18n/iso-639-2.txt

一般的には、ISO 639 標準は、次の Web サイトで確認できます。

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_639.html

地域名の仕様

IBM ILOG Views ロケールでは、ISO 3166 国名の表記規格コードで地域名が指定されています。

ISO 3166 標準は、次の Web サイトで確認できます。

http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm

以下に、例をいくつか示します。

- US (米国)
- NL (オランダ)
- FR (フランス)
- DE (ドイツ)
- JP (日本)

エンコーディングの仕様

IBM ILOG Views では、エンコーディングがコード・セット、すなわち言語に使用されているエンコード方式を識別します。以下はエンコード方式の例です。

- ISO-8859-1 (ISO 8859/1)
- Shift_JIS (Shift Japanese Industrial Standard)

IANA により登録されている文字エンコーディングが使用可能です。現在は、*リファレンス：エンコーディング・リスト*に記載されている文字セットのみが IBM ILOG Views でサポートされています。これらの文字セットは、MIME での表記を専ら使用しています。

詳細については、以下の Web サイトを確認してください。

http://www.iana.org/assignments/character-sets

ロケールの IBM ILOG Views サポートを判断する

IBM ILOG Views が適切なロケールをサポートしているかどうかを判断するには、次のプログラムを実行します。

> ***メモ:*** *この例では、実際のアプリケーションでは使用すべきではないプライベート・コードを使用しています。*

```
// --------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/checkViewsLocale.cpp
// --------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// --------------------------------------------------------------------------

#include <ilviews/ilv.h>
#include <ilviews/base/locale.h>

int main(int argc, char* argv[])
{
    if (!IlvSetLocale()) {
exit(1);
    }

    char* stdLocale = IlLocale::GetStdLocaleName(setlocale(LC_CTYPE, NULL));
    if (stdLocale)
IlvPrint("Standard Views locale name: %s\n", stdLocale);
    else
IlvPrint("Views does not support this locale.\n");

    return 0;
}
```

たとえば、HP-UX システムで LANG を fr_FR.iso88591 に設定している場合、あるいは Solaris システムで LANG を fr に設定している場合、次のようになります。

```
Standard Views locale name: fr_FR.ISO-8859-1
```

Windows システムで 日本語に設定している場合は、次のようになります。

```
Standard Views locale name: ja_JP.Shift_JIS
```

必要なフォント

システムがロケールに必要なフォントをサポートしている必要があります。

UNIX システムの場合

X リソースが IBM ILOG Views アプリケーションに使用されているフォントに設定されていることを確認してください。これを行うには、home ディレクトリにある .Xdefaults ファイルを編集します。このファイルが存在しない場合は作成します。以下のステートメントをリソース・ファイルに追加してください。

```
IlogViews*font: a-valid-font-set-name-for-your-locale
IlogViews*normalfont: a-valid-font-set-name-for-your-locale
IlogViews*italicfont: a-valid-font-set-name-for-your-locale
IlogViews*boldfont: a-valid-font-set-name-for-your-locale
IlogViews*largefont: a-valid-font-set-name-for-your-locale
IlogViews*monospacefont: a-valid-font-set-name-for-your-locale
IlogViews*ButtonFont: a-valid-font-set-name-for-your-locale
IlogViews*MenuFont: a-valid-font-set-name-for-your-locale
```

a-valid-font-set-name-for-your-locale の値は、使用する言語と環境によって異なります。フォントはすべてのシステムで同じとは限りません。

CDE 環境で実行しており、使用言語でデスクトップをスタートさせた場合は、以下の例に示されるように「-dt」エイリアスをフォントに使用できます。

```
IlogViews.font: -dt-interface user-medium-r-normal-m*-*-*-*-*-*-*-*-*
IlogViews.normalfont: -dt-interface user-medium-r-normal-s*-*-*-*-*-*-*-*-*
IlogViews.boldfont: -dt-interface user-bold-r-normal-m*-*-*-*-*-*-*-*-*
IlogViews.italicfont: -dt-interface user-medium-i-normal-m*-*-*-*-*-*-*-*-*
IlogViews.largefont: -dt-interface user-medium-r-normal-xl*-*-*-*-*-*-*-*-*
IlogViews.monospacefont: -dt-interface user-medium-r-normal-m*-*-*-*-*-*-*-*-*
IlogViews.MenuFont: -dt-interface user-bold-r-normal-m*-*-*-*-*-*-*-*-*
IlogViews.ButtonFont: -dt-interface user-bold-r-normal-m*-*-*-*-*-*-*-*-*
```

フォントに「-dt-」エイリアスを使用しない場合、.Xdefaults file に独自のフォント・ステートメントを追加する必要があります。

以下は、HP-UX システム (日本語) で使用されているフォント・ステートメントの例です。

```
IlogViews.ButtonFont: -hp-gothic-bold-r-normal--14-101-100-100-c-*-*-*,
                        -misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
IlogViews.MenuFont: -hp-gothic-bold-r-normal--14-101-100-100-c-*-*-*,
                        -misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
IlogViews.boldfont: -hp-gothic-bold-r-normal--14-101-100-100-c-*-*-*,
                        -misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
IlogViews.font: -misc-fixed-medium-r-normal--14-*-75-75-c-*-*-*,
                      -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
IlogViews.italicfont: -misc-fixed-medium-r-normal--14-*-75-75-c-*-*-*,
                    -adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859-1
IlogViews.largefont: -hp-fixed-medium-r-normal--24-230-75-75-c-*-*-*,
                    -sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1
IlogViews.monospacefont: -misc-fixed-medium-r-normal--14-*-75-75-c-*-*-*,
                       -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
IlogViews.normalfont: -misc-fixed-medium-r-normal--14-*-75-75-c-*-*-*,
                       -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
```

以下は、日本語対応 Solaris システムで使用されているフォント・ステートメントの例です。

```
IlogViews.ButtonFont: -sun-gothic-bold-r-normal--14-120-75-75-c-*-*-*,
                              -*-helvetica-bold-r-normal--14-*-*-*-*-*-iso8859-1
IlogViews.MenuFont: -sun-gothic-bold-r-normal--14-120-75-75-c-*-*-*,
                              -*-helvetica-bold-r-normal--14-*-*-*-*-*-iso8859-1
IlogViews.boldfont: -sun-gothic-bold-r-normal--14-120-75-75-c-*-*-*,
                              -*-helvetica-bold-r-normal--14-*-*-*-*-*-iso8859-1
IlogViews.font: -sun-gothic-medium-r-normal--14-*-75-75-c-*-*-*,
                             -*-helvetica-medium-r-normal--14-*-*-*-*-*-iso8859-1
IlogViews.italicfont: -sun-gothic-medium-r-normal--14-*-75-75-c-*-*-*,
                             -*-helvetica-medium-o-normal--14-*-*-*-*-*-iso8859-1
IlogViews.largefont: -sun-gothic-medium-r-normal--22-200-75-75-c-*-*-*,
                             -*-helvetica-medium-r-normal--24-*-*-*-*-*-iso8859-1
IlogViews.monospacefont: -sun-gothic-medium-r-normal--14-*-75-75-c-*-*-*,
                             -*-helvetica-medium-r-normal--14-*-*-*-*-*-iso8859-1
IlogViews.normalfont: -sun-gothic-medium-r-normal--14-*-75-75-c-*-*-*,
                             -*-helvetica-medium-r-normal--14-*-*-*-*-*-iso8859-1
```

Microsoft Windows システムの場合

多くの場合は、フォントのデフォルト設定で十分です。アプリケーションで使用するフォントを変更する必要がある場合は、`views.ini` ファイルを (すべてのアプリケーション用に) 編集して次のステートメントの一部または全体を含めます。

```
[IlogViews]
font=a-valid-font-for-your-language
normalfont=a-valid-font-for-your-language
italicfont=a-valid-font-for-your-language
boldfont=a-valid-font-for-your-language
largefont=a-valid-font-for-your-language
monospacefont: a-valid-font-for-your-locale
buttonFont=a-valid-font-name-for-your-locale
menuFont=a-valid-font-name-for-your-locale
```

値 `a-valid-font-for-your-language` は、使用する言語と環境によって異なります。Microsoft Word またはテキスト・エディタを使用して、使用する言語でテキストを表示するのに適切なフォント名を確認してください。`.ini` ファイルのエントリは、次の形式になります。

```
%<font name>-<font size>-<style>
```

たとえば、Helvetica、太字、サイズ 12 の場合は `%helvetica-12-B` になります。フォントを標準表示する場合は、`style` パラメータを空白にし、`%helvetica-12-` と入力すると Helvetica のサイズ 12 で表示されます。

以下は、Windows システムの日本語対応フォントの例です。

```
font=%ＭＳ明朝-12-
normalfont=%ＭＳ明朝-12-
buttonfont=%ＭＳＰゴシック-12-
boldfont=%ＭＳゴシック-12-B
italicfont=%ＭＳゴシック-12-I
largefont=%ＭＳ明朝-16-
monospacefont=%ＭＳ明朝-12-
menuFont=%ＭＳＰゴシック-12-
toolBarFont=%ＭＳ明朝-12-
```

IBM ILOG Views のローカライズ・メッセージ・データベース

IBM® ILOG® Views では、メッセージ・テキスト、メニュー・アイテム・テキスト、その他ユーザ・インターフェースに表示されるテキストにメッセージ・データベース・ファイル (.dbm ファイル) を使用しています。この 3 つのファイルに関しては、次のトピックで詳しく説明します。

- *IlvMessageDatabase クラス*では、ローカライゼーションのクラス・メカニズムについて説明します。
- *メッセージ・データベース・ファイルの言語*の説明にあるように、ローカライゼーションではメッセージ・データベース・ファイルがローカル言語に翻訳されている必要があります。
- また、メッセージ・データベースを読み込む際に IBM ILOG Views がファイルを見つけられるように、ファイルが正しいディレクトリにある必要があります。詳細については、*メッセージ・データベース・ファイルの場所*を参照してください。
- *メッセージ・データベース・ファイルのパラメータを決定する* は、ロケール、言語、メッセージ・データベース・ファイルを検索する簡単なプログラムを備えています。
- *メッセージ・データベースの読み込み*では、デフォルト言語の自動読み込みを説明し、デフォルト言語に上書きして別の言語を読み込むさまざまな方法を紹介します。
- *.dbm ファイル形式*では、メッセージ・データベース・ファイルの形式と旧 (バージョン 3.0 以前の) フォーマット済みファイルの処理について説明します。
- *表示言語の動的な変更方法*の説明に従い、`setCurrentLanguage` を使用してオン・ザ・フライで言語を変更できます。

IlvMessageDatabase クラス

IBM ILOG Views には、多言語対応アプリケーションを操作する上で役立つ簡単なメカニズムが用意されています。このメカニズムはメッセージ・メカニズムと呼ばれ、IlvMessageDatabase クラスに基づいています。

これは、1 つのメッセージに対するさまざまな翻訳文を格納するデータベースを使用します。現在の言語に応じて、適切なメッセージにアクセスします。IlvDisplay クラスの各インスタンスは、独自のメッセージ・データベースを作成します。このデータベースでは、環境変数 ILVDB に与えたファイル名からデータベースの詳細を読み込みます。またこの変数が設定されていない場合は、views .dbm から読み込みます。このファイルはディスプレイ・パスで検索されます。このデータベースは、IlvDisplay::getDatabase メンバ関数を呼び出すことでアクセスできます。

それぞれの言語で表示させたい各文字列は、それぞれの翻訳文でデータベースに格納できます。すなわち、1 つの*メッセージ識別子*が、使用する言語に従いそれぞれの*メッセージ文字列*に関連付けられます。言語はシンボル・オブジェクト (IlSymbol クラス) で指定します。以下は、サンプル・コードです。

```
IlvMessageDatabase database;
IlSymbol* en_US = IlGetSymbol("en_US");
IlSymbol* fr_FR = IlGetSymbol("fr_FR");
database.putMessage("&cancel", en_US, "Cancel");
database.putMessage("&cancel", fr_FR, "Annuler");
```

IBM ILOG Views 環境変数 ILVLANG を使用すると、現在の言語 (英語、フランス語、日本語など) を上書きできます。詳細については、*IBM ILOG Views のローカライズ・メッセージ・データベース*を参照してください。

> ***メモ:*** IBM ILOG Views *は多言語対応アプリケーションをサポートしていませんが、互換性のあるエンコーディングを使用する場合は同じアプリケーションで多言語を使用できます。*

メッセージ・データベース・ファイルの言語

IBM® ILOG® Views は、メッセージ・データベース・ファイルを英語とフランス語でリリースしています。サポートされている言語それぞれのファイルは別々のディレクトリにあり、それぞれのエンコード方式を使用して多言語を使用します。英語、フランス語以外の言語が必要な場合は、.dbm ファイルを適切な言語に翻訳する必要があります。ファイルが正しい .dbm 形式であることを確認してください (詳細については、*.dbm ファイル形式*を参照してください)。

メッセージ・データベース・ファイルの場所

ローカライズ・メッセージ・データベースは、locale ディレクトリの下のサブディレクトリにあります。このサブディレクトリは、対応する言語および使用され

たエンコード方式により名前が付けられます。サブティレクトリ名は以下の形式になっています。

```
<ll_TT.encoding>
```

たとえば、UNIX システムでは、フランス語のメッセージ・データベース・ファイルは、`locale` ディレクトリの下の `fr_Fr.ISO-8859-1` サブディレクトリにあります。サブディレクトリ名の基になる IBM ILOG Views ロケール名の命名規則については、*IBM ILOG Views ロケール名* を参照してください。

UNIX システムの場合

フランス語のメッセージ・データベース・ファイルは、以下のディレクトリにあります。

- `<$ILVHOME>/bin/data/locale/fr_FR.ISO-8859-1/editpnl.dbm`
- `<$ILVHOME>/bin/data/locale/fr_FR.ISO-8859-1/ilv2data.dbm`
- `<$ILVHOME>/bin/data/locale/fr_FR.ISO-8859-1/ilvedit.dbm`
- `<$ILVHOME>/data/ivprotos/locale/fr_FR.ISO-8859-1/protos.dbm`
- `<$ILVHOME>/data/iljscript/locale/fr_FR.ISO-8859-1/gide.dbm`
- `<$ILVHOME>/data/iljscript/locale/fr_FR.ISO-8859-1/messages.js`
- `<$ILVHOME>/data/ilviews/locale/fr_FR.ISO-8859-1/views.dbm`
- `<$ILVHOME>/studio/data/ivprotos/locale/fr_FR.ISO-8859-1 / prstudio.dbm`
- `<$ILVHOME>/studio/data/ivstudio/locale/fr_FR.ISO-8859-1/ jsstudio.dbm`
- `<$ILVHOME>/studio/data/ivstudio/locale/fr_FR.ISO-8859-1/ studio.dbm`
- `<$ILVHOME>/studio/data/ivstudio/locale/fr_FR.ISO-8859-1/ vrstudio.dbm`

Microsoft Windows システムの場合

フランス語のメッセージ・データベース・ファイルは、以下のディレクトリにあります。

- `<$ILVHOME>/bin/data/locale/fr_FR.windows-1252/editpnl.dbm`
- `<$ILVHOME>/bin/data/locale/fr_FR.windows-1252/ilv2data.dbm`
- `<$ILVHOME>/bin/data/locale/fr_FR.windows-1252/ilvedit.dbm`
- `<$ILVHOME>/data/ivprotos/locale/fr_FR.windows-1252/protos.dbm`
- `<$ILVHOME>/data/iljscript/locale/fr_FR.windows-1252/gide.dbm`
- `<$ILVHOME>/data/iljscript/locale/fr_FR.windows-1252/messages.js`
- `<$ILVHOME>/data/ilviews/locale/fr_FR.windows-1252/views.dbm`

- <$ILVHOME>/studio/data/ivprotos/locale/fr_FR.windows-1252 / prstudio.dbm
- <$ILVHOME>/studio/data/ivstudio/locale/fr_FR.windows-1252/ jsstudio.dbm
- <$ILVHOME>/studio/data/ivstudio/locale/fr_FR.windows-1252/ studio.dbm
- <$ILVHOME>/studio/data/ivstudio/locale/fr_FR.windows-1252/ vrstudio.dbm

以下のプログラムを実行して、メッセージ・データベース・ファイルの場所を検索できます。

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/checkLocalizedPath.cpp
// ------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// ------------------------------------------------------------------------


#include <ilviews/ilv.h>
#include <ilog/pathname.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    if (!IlvSetLocale()) {
exit(1);
    }

    IlPathName pname("");
    pname.localize();
    IlvPrint("\nLooking under directories: .../%s\n\n",
     pname.getString().getValue());

    return 0;
}
```

*メモ: C ロケール (*IBM ILOG Views *ロケール* en_US*) は標準とされており、上記のルールの例外です。*IBM ILOG Views .dbm *ファイルは、ファイルを使用するライブラリのディレクトリにあります。たとえば、*views.dbm *は* <$ILVHOME>/data/ilviews/views.dbm*. にあります。ディレクトリ* en_US.US-ASCII *を作成する必要はありません。*IBM ILOG Views *は自動的に通常のデータ・ディレクトリに戻ります。*

たとえば、HP-UX システムで LANG を fr_FR.iso88591 に設定している場合、あるいは Solaris システムで LANG を fr に設定している場合、次のような結果になります。

```
Looking under directories: .../locale/fr_FR.ISO-8859-1/
```

Windows システムで 日本語に設定している場合は、次のようになります。

```
Looking under directories: .../locale\ja_JP.Shift_JIS\
```

メッセージ・データベース・ファイルのパラメータを決定する

以下のプログラムを実行して、現在の IBM® ILOG® Views 表示言語と IBM ILOG Views がメッセージ・データベース・ファイルを検索するパス名を決定できます。

パス名を表示するためには、verboseFindInPath 環境変数を true に設定します。ILVPATH 環境変数を変更して、その結果表示されるパス名を確認します。

```
// ------------------------------------------------------------- -*- C++ -*-
//                                                 IlogViews userman source file
// File: doc/fondation/userman/src/internationalization/checkLocalizedDbm.cpp
// --------------------------------------------------------------------------
//  Copyright (C) 1990-2008 by ILOG.
//  All Rights Reserved.
// --------------------------------------------------------------------------

#include <ilviews/ilv.h>
#include <ilviews/base/message.h>
#include <ilog/pathlist.h>

int main(int argc, char* argv[])
{
    if (!IlvSetLocale()) {
        exit(1);
    }

    IlvDisplay* display = new IlvDisplay("CheckLocalizedDbm", 0, argc, argv);

    IlvPrint("Current Views display language: %s\n",
     display->getCurrentLanguage()->name());

    const char* path = display->getPath();

    IlPathList plist(path? path : "./");
    IlvPrint("Current path: %s\n", plist.getString().getValue());

    display->getDatabase()->read("my-file.dbm", display);

    return 0;
}
```

たとえば、LANG=fr_FR.iso88591 に設定されている HP-UX システムまたは LANG=fr に設定されている Solaris システムでこのプログラムを実行すると、次のような結果になります。

```
IlvPathList::findInPath file ilviews/locale/fr_FR.ISO-8859-1/views.dbm not in ./.
IlvPathList::findInPath found: <$ILVHOME>/data/ilviews/locale/fr_FR.ISO-8859-1/views.dbm.

Current Views display language: fr_FR

 Current path: ./

IlvPathList::findInPath file locale/fr_FR.ISO-8859-1/my-file.dbm not in ./.
IlvPathList::findInPath file locale/fr_FR.ISO-8859-1/my-file.dbm not in <$ILVHOME>/data/.
IlvPathList::findInPath file locale/fr_FR.ISO-8859-1/my-file.dbm not in <$ILVHOME>/data/icon/.
IlvPathList::findInPath file locale/fr_FR.ISO-8859-1/my-file.dbm not in <$ILVHOME>/data/images/
.
 IlvDisplay::findInPath Couldn't find 'locale/fr_FR.ISO-8859-1/my-file.dbm'
IlvPathList::findInPath file my-file.dbm not in ./.
IlvPathList::findInPath file my-file.dbm not in <$ILVHOME>/data/.
IlvPathList::findInPath file my-file.dbm not in <$ILVHOME>/data/icon/.
IlvPathList::findInPath file my-file.dbm not in <$ILVHOME>/data/images/.
 IlvDisplay::findInPath Couldn't find 'my-file.dbm'
```

メッセージ・データベースの読み込み

IBM® ILOG® Views は、自動的に locale/<ll_TT.encoding> ディレクトリにある正しいメッセージ・データベースを読み込みます。

たとえば、ISO 8859-1 フランス語環境で作業する場合、次が呼び出されます。

```
display->getDatabase()->read("/my-directory-path/my-file.dbm");
```

これは、UNIX システムの次のディレクトリで自動的にファイルを検索します。

```
/my-directory-path/locale/fr_FR.ISO-8859-1/my-file.dbm
```

Microsoft Windows システムの場合は、次のディレクトリを検索します。

```
/my-directory-path/locale/fr_FR.windows-1252/my-file.dbm
```

> ***メモ:*** IBM ILOG Views *3.0 以前では、*ILVLANG *環境変数を使用する言語に設定する必要がありました。*

デフォルトの振る舞いを上書きする

デフォルトの振る舞いを上書きして IBM ILOG Views で別の表示言語を使用する場合、ILVLANG 環境変数を使用します。XPG4 互換の UNIX システムの場合、LC_MESSAGES 環境変数も使用できます。IBM ILOG Views は、以下の順序でメッセージ・データベース・ファイルを検索します。

UNIX システムの場合

1. ILVLANG
2. LC_MESSAGES
3. 実行中ロケールの LC_CTYPE カテゴリ

Windows システムの場合

1. ILVLANG
2. 実行中ロケールの LC_CTYPE カテゴリ

> ***メモ:*** IBM ILOG Views *表示言語を* LC_MESSAGES *または* ILVLANG *環境変数で上書きして変更したい場合は、メッセージに使用するエンコーディングと同等、または強力な (すなわちエンコーディングのスーパーセット) エンコーディングでプログラムが実行することを確認してください。これは、*.dbm *ファイルが、プログラムが実行される* IBM ILOG Views *ロケール・エンコーディングに基づいて読み込まれるためです。たとえば、プログラムを日本語またはフランス語で実行する場合、英語を読むことはできてもその逆はできません。*

LANG リソースを使用してデフォルトの振る舞いを上書きする

UNIX システムの場合

ILVLANG 環境変数を設定して、デフォルト以外の言語を使用できます。ILVLANG は IBM ILOG Views アプリケーションのみに適用され、システムに依存しません。

たとえば、IBM ILOG Views の現在のロケールがフランス語で、スペイン語のメッセージを表示したい場合、UNIX システムでは ILVLANG=es_ES を使用できます。設定は次のようになります。

```
Current Views locale running: fr_FR.ISO-8859-1
Current Views display language: es_ES
Looking under directories: .../locale/es_ES.ISO-8859-1/
```

この場合には、IBM ILOG Views アプリケーションのメッセージのみがスペイン語で表示されます。システム・メッセージは変更されません。

Microsoft Windows システムの場合

views.ini ファイルの lang 変数を設定して、デフォルト以外の言語を使用できます。たとえば、IBM ILOG Views の現在のロケールがフランス語で、スペイン語のメッセージを表示したい場合、lang=es_ES を使用できます。設定は次のようになります。

```
Current Views locale running: fr_FR.windows-1252
Current Views display language: es_ES
Looking under directories: .../locale/es_ES.windows-1252/
```

LC_MESSAGES (Unix のみ) を使用してデフォルトの振る舞いを上書きする

LC_MESSAGES 環境変数を設定して、デフォルト以外の言語を使用できます。LC_MESSAGES 環境変数を使用すると、すべてのシステム・メッセージが上書きされることにご注意ください。

たとえば、IBM ILOG Views の現在のロケールがフランス語で、イタリア語のメッセージを表示したい場合、HP-UX システムでは `LC_MESSAGES=it_IT.iso88591`、Solaris システムでは `LC_MESSAGES=it` を使用できます。設定は次のようになります。

```
Current Views locale running: fr_FR.ISO-8859-1
Current Views display language: it_IT
Looking under directories: .../locale/it_IT.ISO-8859-1/
```

この場合、IBM ILOG Views がイタリア語で表示されるだけではなく、すべてのシステム・メッセージもイタリア語になります。

.dbm ファイル形式

`.dbm` 形式では、サポートされている言語それぞれのファイルは別々のディレクトリにあり、それぞれのエンコード方式を使用して多言語を処理します。

`.dbm` ファイルは以下の形式になっています。

```
// IlvMessageDatabase ...
// Language: <ll_TT>
// Encoding: <encoding>
"&message" "message translation..."
```

最初の行は IBM® ILOG® Views のバージョン、作成日、および `IlvMessageDatabase` に関する情報です。

言語は IBM ILOG Views の `ll_TT` 命名規則を使用して表示されています。ここで、`ll` は言語名の 2 文字略語で、`TT` は地域名の 2 文字略語です。

エンコード方式は、IBM ILOG Views でサポートされているものでなければなりません。サポートされているエンコード方式のリストについては、*リファレンス：エンコーディング・リスト* を参照してください。

以下の例は、フランス語のメッセージ・データベース・ファイルの一部です。

```
// IlvMessageDatabase
// Language: fr_FR
// Encoding: ISO-8859-1
"&AlignmentLabelPicture" "Alignement texte / image"
"&Appearance" "Apparence"
"&April" "avril"
```

.dbm ファイルをローカル言語に翻訳する場合、ファイルが .dbm 形式であることを確認してください。

> ***メモ:*** *アメリカ英語のローカライズ・データベース・メッセージ・ファイルに関しては、以下に従うようにしてください。ファイルの* locale *ディレクトリに* en_US.US-ASCII *サブディレクトリを* ***作成しない*** *でください。ファイルをデータ・ディレクトリに直接配置してください。たとえば、*views.dbm *は* <$ILVHOME>/data/ilviews/ *に置きます。アメリカ英語用に使用するエンコーディングが US-ASCII ではない場合でも、以下の例のようにファイルの内容を設定してください。これは US-ASCII がもっとも弱いエンコーディングであり、*IBM ILOG Views *がサポートするその他のエンコーディングによって読む込めるためです。*
>
> ```
> // IlvMessageDatabase
> // Language: en_US
> // Encoding: US-ASCII
> "&AlignmentLabelPicture" "Alignment text / picture"
> "&Appearance" "Appearance"
> "&April" "April"
> ```

バージョン 3.0 以前の .dbm ファイル形式

IBM ILOG Views バージョン 3.0 は、.dbm ファイル形式に互換性のないさまざまなエンコード方式を使用するさまざまな言語をサポートするように改良されました。バージョン 3.0 以前の .dbm ファイル形式では、メッセージ・データベースにサポートされている各言語に対応するメッセージの翻訳が含まれていました。すなわち、サポートされているすべての言語の翻訳が同じデータベースにありました。旧 .dbm 形式のファイルは、IBM ILOG Views 3.0 以降を使用して読み込み可能ですが、新しいファイルは新しい形式で生成されます。

データベース・ファイルが旧 .dbm 形式である場合、サポートされている言語ごとに、データベースをいくつかのファイルに分けることをお勧めします。これで管理が簡単になるだけでなく、エンコーディングの非互換性を避けることができます。

旧形式のデータベース・ファイルを分割するには、次のプログラムを使用します。

$ILVHOME/bin/src/splitdbm.cpp

このプログラムは .dbm ファイルでさまざまな言語を検索するだけでなく、各言語の新しい言語名 (IBM ILOG Views 命名規則 ll_TT の使用を推奨)、エンコード方

式 (選択されたエンコード方式は現在のものと互換性がなければなりません)、新しい形式のファイル名を提案します。

> ***メモ:*** *splitdbm プログラムはもっとも強力なエンコーディングを使用して実行してください。もっとも強力なエンコーディングとは、他を含むものです。たとえば、英語 (US-ASCII) および日本語 (Shift_JIS) を含むファイルを分割したい場合、日本語ロケールを使用して* splitdbm *を実行してください。Shift_JIS は US-ASCII を含みますが、逆は成立しません。したがって、Shift_JIS がもっとも強力なエンコーディングとなり、プログラムの実行に使用することになります。*

> ***メモ:*** *アメリカ英語のメッセージを含むファイルを分割する場合、以下に従うようにしてください。* splitdbm *が情報の入力を促したら、US-ASCII エンコーディングを選択し、ローカライズされたファイルを* locale *サブディレクトリではなくデータ・ディレクトリに格納してください。*

例

この例は、データベース・メッセージ・ファイルの分割方法を示しています。

次の your_data_dir/testall.dbm ファイルには、アメリカ英語、フランス語、イタリア語の 3 言語のメッセージ・テキストが含まれています。

```
// IlvMessageDatabase 3 Web Jun 3 11:50:35 1998
"&Hello" 3
"en_US" "Hello"
"fr_FR" "Bonjour"
"it_IT" "Buongi?rno"
"&Goodbye" 3
"en_US" "Goodbye"
"fr_FR" "Au revoir"
"it_IT" "Ciao"
```

このファイルの各言語ごとに 3 つのファイルに分割するには、フランス語またはイタリア語ロケールで splitdbm プログラムを実行する必要があります。プログラムの実行時に、プログラムの終了に必要な情報の入力を促されます。実行が終了すると、言語ごとのメッセージを含む 3 つのファイルが作成されます。UNIX システムの場合、結果としてできるファイルは次のようになります。

```
your_data_dir/test.dbm
locale/fr_FR.ISO-8859-1/test.dbm
locale/it_IT.ISO-8859-1/test.dbm
```

各ファイルの内容は、次のようになります。

```
// IlvMessageDatabase
// Language: en_US
// Encoding: US-ASCII
"&Goodbye" "Goodbye"
"&Hello" "Hello"
```

```
// IlvMessageDatabase
// Language: fr_FR
// Encoding: ISO-8859-1
"&Goodbye" "Au revoir"
"&Hello" "Bonjour"

// IlvMessageDatabase
// Language: it_IT
// Encoding: ISO-8859-1
"&Goodbye" "Ciao"
"&Hello" "BuongiÞrno"
```

.dbm ファイルのエンコーディングの互換性

IBM ILOG Views アプリケーションは、システム環境と互換性のあるエンコーディングを使用して作成された .dbm ファイルのみを読み込みます。同じ文字セットを共有する場合、エンコーディングには互換性があります。エンコーディング情報を含まない旧形式の .dbm を読み込む場合、これらのファイルは現在のロケールのエンコーディングとなっています。そうでない場合、情報は不適切に読み込まれるか、または全体が読み込まれない場合があります。

表示言語の動的な変更方法

表示言語を動的に変更することができます。これにはディスプレイで IlvDisplay::setCurrentLanguage を呼び出すだけです。IBM ILOG Views は自動的に現在読み込んだすべてのデータ・ファイルを再度読み込み、新しい言語を表示します。これを行うには、システムで利用可能なこれらのファイルのローカライズ・バージョンを提供する必要があります。

> ***メモ:*** *言語を切り替える場合、アプリケーションをもっとも強力なエンコーディングでスタートする必要があります。たとえば、フランス語と英語を切り替える場合、アプリケーションをフランス語でスタートします。ただし、アプリケーションのスタート時に英語のメッセージを表示したい場合、*ILVLANG *環境変数または Windows の* lang *リソースを使用して、スタート時に表示される言語を上書きします。*

すべてのメッセージが my_messages.dbm というファイルで定義されているアプリケーションを作成したとします。

display がディスプレイである場合、アプリケーションの開始時に次を呼び出してこのファイルを読み込みます。

```
display->getDatabase()->read("my_messages.dbm", display);
```

フランス語ロケールでプログラムをスタートした場合、IBM ILOG Views は locale/fr_FR.ISO-8859-1/my_messages.dbm にあるファイルを読み込みます。

表示言語を変更するには、使用する新しい言語の IlvDisplay::setCurrentLanguage をここで呼び出します。たとえば、イタリア語をディスプレイに表示させたい場合、次を呼び出します。

```
display->setCurrentLanguage(IlGetSymbol("it_IT"));
```

IBM ILOG Views は、以下にあるファイルを自動的に読み込みます。

```
locale/it_IT.ISO-8859-1/my_messages.dbm
```

これはまた、既に開いている他のすべてのデータ・ファイルも読み込みます。

フランス語に戻すには、以下のように再び IlvDisplay::setCurrentLanguage を呼び出します。

```
display->setCurrentLanguage(IlGetSymbol("fr_FR"));
```

メモ: *これは、エンコーディングに互換性がある場合のみ有効です。*

samples/foundation/i18n/changelang にあるサンプルは、この特性を示したものです。

極東アジア言語で IBM ILOG Views を使用する

システムで日本語、韓国語、中国語などの極東アジア言語をサポートする場合は、このセクションをお読みください。極東アジア言語はマルチバイト文字言語であり、IBM® ILOG® Views を使用する場合にその特性を考慮する必要があります。

API を変更しなくても、char* 値は極東アジア言語ロケールでマルチバイト文字を使用できます。

たとえば、次のマルチバイト文字列を渡すことができます。

```
void IlvListLabel::setText(const char* text);
```

戻り値として、次のマルチバイト文字列が返されます。

```
const char* IlvListLabel::getText() const;
```

これはすべてのガジェット・クラス (すなわち IlvText、IlvTextField およびそのサブクラス IlvMessageLabel、IlvStringList など) および IlvManagerMakeStringInteractor または IlvManagerMakeTextInteractor のようなマネージャ・ビューのインタラクタにあてはまります。

プログラマがテキスト領域への入力を制御できるように、mbCheck メソッドが IlvTextField および IlvText ガジェットに追加されています。API は次のように定義されます。

IlvText の場合

```
virtual IlBoolean  mbCheck(const char* text);
```

IlvTextField の場合

```
virtual const char* mbCheck(const char* text);
```

> ***メモ:*** *mbCheck メソッドは、モノバイト・ロケールで実行している場合、check メソッドを呼び出します。*

IlvPasswordTextField はマルチバイト文字列をサポートし、マスクは描画可能な文字に適用されます。これはマルチバイト文字列を使用して、IlvPasswordTextField で setLabel を作成または実行できるということです。

内部的には、IBM ILOG Views は wide-char* 値で処理を行っていますが、外部 API にはそれが文書化されたものはありません。wide-char* 値を使用したい場合、外部 API を呼び出す前に char* 値を切り替える必要があります。macros.h ファイルで wchar_t と定義されている IlvWChar タイプを、独自の国際化 API に使用することができます。

データ入力要件

- IBM® ILOG® Views は、240 ページの *Input Method (IM)* で説明されているように、Input Method をサポートしています。
- 入力コンテキストの制御については、241 ページの *IBM ILOG Views でテスト済みの極東アジア言語 Input Method サーバ*を参照してください。
- 241 ページの *データ入力に使用する言語の制御方法* の例に示されているように、ローカライズした入力を禁止できます。

Input Method (IM)

極東アジア言語などの言語は、多くの文字を使用します。Input Method (IM) という概念は、これらの文字をキーボードで入力できるように作成されました。Input Method は、プロシージャ、マクロ、さらにキーの入力を現在のロケールのコード・セットでエンコーディングされる文字に変換する個別のプロセスである場合もあります。

UNIX システムの場合、ヨーロッパ言語の Input Method は X ライブラリで直接サポートされています。しかし、極東アジア言語を実行するには個別のプロセスが必要です。

これらの言語の場合、Input Method (フロントエンド・プロセッサともいう) をシステムで実行しなければならず、それに従って環境を設定する必要があります。UNIX システムの場合は、XMODIFIERS 環境変数などを設定します。必要な操作については、ローカル・システムのマニュアルを確認してください。

Input Method サーバを使用した入力は、以下のクラスでサポートされています。

- IlvText、IlvTextField およびそのサブクラス (IlvDateField、IlvNumberField、IlvPasswordTextField を除く)。
- テキストの入力に IlvTextField を使用するクラスもまた、Input Method サーバが使用できます。これは、たとえば、IlvMatrix または IlvManagerMakeStringInteractor に該当します。

IBM ILOG Views でテスト済みの極東アジア言語 Input Method サーバ

UNIX システムでは、入力コンテキストの使用方法を制御できます。デフォルトでは、上位レベル・ウィンドウの 1 つの入力コンテキストを共有します。これは、この上位レベル・ウィンドウの入力テキスト領域はすべて入力コンテキストを共有するということです。

入力テキスト領域のそれぞれに異なった入力コンテキストを使用したい場合は、ILVICSHARED 環境変数を「no」に設定します。

HP-UX の場合、次の IM サーバが IBM ILOG Views でテスト済みです。

- 日本語：xjim、atok8
- 中国語：xtim、xsim
- 韓国語： xkim

Solaris の場合、次の IM サーバが IBM ILOG Views でテスト済みです。

- 日本語： htt
- 中国語：htt
- 韓国語：htt

Windows の場合、IBM ILOG Views はデフォルトの IME サーバに直接接続します。

データ入力に使用する言語の制御方法

IlvText または IlvTextField のサブクラスである入力フィールド・オブジェクトは、(SetLocale が呼び出された場合に限り) 自動的に Input Method に接続するので、現在のロケールへの入力が可能になります。

アプリケーションにこの振る舞いを禁止させる (すなわち、ローカライズされた入力でなく ASCII 入力だけを使用する) 場合は、パラメータ値を IlFalse とする

SimpleGraphic オブジェクトの setNeedsInputContext メソッドを呼び出す必要があります。

```
virtual void setNeedsInputContext(IlBoolean val)
```

例

samples/foundation/i18n/controlinput のコード・サンプルは、テキスト・フィールドが 2 つあるガジェットを作成します。最初のテキスト・フィールドは Input Method に接続し、現在のロケールでの入力を可能にします。2 つ目のテキスト・フィールドは Input Method に接続しません。これは英語のみをフィールドに入力できることを意味します。

国際化機能の制限

現在の国際化サポート機能には、次の制限があります。

- 文字列の .ilv ファイルでの保存または読み込みは、現在のロケールのエンコーディングで行われます。現在のロケールでエンコーディングされていないファイルは読む込めません。
- Input Method サーバを使用した入力は、パスワード・テキスト・フィールドでサポートされていません。これはユーザが自分で入力しているものを見る必要がないためです。
- Input Method サーバを使用した入力は、データ・フィールドおよび数字フィールドで現在サポートされていません。Windows プラットフォームの場合、ユーザは FEP から切断されます。これは、これらのガジェットへのテキスト入力を可能にするためです。
- IlvAnnoText は、国際化をサポートしていません。
- 使用するフォントあるいは言語により、IBM ILOG Views Studio のメイン・ウィンドウは極めて小さくなります。ただし、どの IBM ILOG Views Studio パネルでも、メイン・ウィンドウのサイズは studio.pnl ファイルで幅を設定してカスタマイズできます。

  ```
  panel "MainPanel" {
   // ....
  width 900;
  }
  ```

- このバージョンでは、マルチバイト変数はサポートされていません。IBM ILOG Views の変数モジュールについての詳細は、『Manager』マニュアルを参照してください。

- 単一バイト文字をニーモニック向けに IBM ILOG Views で定義できます。ただし、キーボードに対応した単一バイト文字に、ニーモニックをインストールすることをお勧めします。たとえば、ヨーロッパ言語や日本語の半角文字でアクセント記号を使用することはお勧めしません。
- UNIX システムの場合、バックエンド・アーキテクチャを使用して実装した Input Method のみがサポートされています。たとえば、Solaris で `htt` を使用している場合、アプリケーションを実行する前に `XIMP_TYPE` 環境変数を `XIMP_SYNC_BE_TYPE2` に設定してください。

トラブルシューティング

ローカライズされたメッセージが画面に表示されない場合は、次の手順に従ってください。

1. プログラムの最初に `IlvSetLocale` を呼び出したことを確認してください。
2. 表示するためのロケールおよびフォントがシステムにサポートされていることを確認してください。ほとんどの UNIX システムの場合、`locale ?a` コマンドを実行できます。*ロケール要件* を参照してください。
3. `ILVLANG` 環境変数を設定しないでください。
4. UNIX プラットフォームの場合、`LANG` をシステムまたは X Window システムにサポートされているロケールに設定してください。たとえば、LANG 変数をフランス語に設定するには、以下のように入力します。

   Solaris の場合 `LANG=fr`

   HP_UX の場合 `LANG=fr_FR.iso88591`
5. ローカライズされた .dbm ファイルが `.../locale/<ll_TT.encoding>/your_file.dbm` というサブディレクトリにあることを確認します。*メッセージ・データベース・ファイルの場所* を参照してください。
6. `.dbm` ファイルの内容が、以下のような新しい形式になっていることを確認します。*.dbm ファイル形式* を参照してください。

   ```
   // IlvMessageDatabase ...
   // Language: <ll_TT>
   // Encoding: <encoding>
   "&message" "message translation..."
   ```
7. IBM ILOG Views Studio エディタで旧形式の `.dbm` ファイルを読み込んだ (旧形式の `.dbm` ファイルは IBM ILOG Views バージョン 3.0 以前で作成されています) 時に、ファイルの表示が切り捨てられている場合、エンコーディングに互換性がないことを意味します。この場合は、`.dbm` ファイルを分割します。*バージョン 3.0 以前の .dbm ファイル形式* を参照してください。

リファレンス：エンコーディング・リスト

IBM ILOG Views では、以下のエンコーティングをサポートしています。

- US-ASCII
- ISO-8859-1 (Latin1)
- ISO-8859-2 (Latin2)
- ISO-8859-3 (Latin3)
- ISO-8859-4 (Latin4)
- ISO-8859-5 (LatinCyrillic)
- ISO-8859-6 (LatinArabic)
- ISO-8859-7 (LatinGreek)
- ISO-8859-8 (LatinHebrew)
- ISO-8859-9 (Latin5)
- ISO-8859-10 (Latin6)
- ISO-8859-11 (LatinThai)
- ISO-8859-13 (Latin7)
- ISO-8859-14 (Latin8)
- SO-8859-15 (Latin9)
- EUC-JP
- Shift_JIS
- EUC-KR
- GB2312
- Big5
- Big5-HKSCS
- EUC-TW
- hp-roman8
- IBM850
- windows-1250
- windows-1251

- windows-1252
- windows-1253
- windows-1254
- windows-1255
- windows-1256
- windows-1257
- windows-1258
- windows-874
- windows-949
- UTF-8

ISO-8859-1

Latin1 には、以下のようなほとんどの西ヨーロッパ言語が含まれます。

- アフリカーンス語 (af)
- アルバニア語 (sq)
- バスク語 (eu)
- カタロニア語 (ca)
- デンマーク語 (da)
- オランダ語 (nl)
- 英語 (en)
- フェロー語 (fo)
- フィンランド語 (fi)
- フランス語 (fr)
- ガリシア語 (gl)
- ドイツ語 (de)
- アイスランド語 (is)
- アイルランド語 (ga)
- イタリア語 (it)
- ノルウェー語 (no)
- ポルトガル語 (pt)

- スコットランド語 (gd)
- スペイン語 (es)
- スウェーデン語 (sv)

ISO-8859-2

Latin2 には中央ヨーロッパおよび東ヨーロッパの言語が含まれます。

- クロアチア語 (hr)
- チェコ語 (cs)
- ハンガリー語 (hu)
- ポーランド語 (pl)
- ルーマニア語 (ro)
- スロバキア語 (sk)
- スロベニア語 (sl)

ISO-8859-3

Latin3 は、エスペラント語 (eo)、マルタ語でよく使われ、Latin5 が作成される前はトルコ語が含まれていました。

ISO-8859-4

Latin4 でエストニア語、バルト語、ラトビア語、リトアニア語、グリーンランド語、ラップ語が使用できるようになりました。これは Latin6 の基となったものです。

ISO-8859-5

これらのキリル文字を使用して、ブルガリア語 (bg)、ベラルーシ語 (be)、マケドニア語 (mk)、ロシア語 (ru)、セルビア語 (sr)、ウクライナ語 (uk) を入力できます。

ISO-8859-6

これはアラビア語 (ar) のアルファベットです。

> ***メモ:*** IBM ILOG Views *の本バージョンは双方向テキストをサポートしていません。*

ISO-8859-7

これは現代ギリシャ語 (el) です。

ISO-8859-8

これはヘブライ語 (iw) です。

> ***メモ:*** *IBM ILOG Views の本バージョンは双方向テキストをサポートしていません。*

ISO-8859-9

Latin5 では Latin1 でほとんど使用されないアイスランド語の文字がトルコ語 (tr) の文字と置き換えられています。

ISO-8859-10

Latin6 は Latin4 を再構成したもので、これまでは含まれていなかったイヌイット語 (グリーンランドのエスキモー語) および非スコルト・サーメ語 (ラップ語) の文字が追加され、稀に使用されるアイスランド語の文字が再追加されて北欧全体がカバーされています。

- エストニア語 (et)
- ラップ語
- ラトビア語 (lv)
- リトアニア語 (lt)

スコルト・サーメ語には、数個のアクセントを追加する必要があります。

ISO-8859-11

タイ語をカバーしています。UNIX システムの場合、これは tis620 エンコーディングに類似しています。

ISO-8859-13

バルト海沿岸地域をカバーしています。Latin7 ではバルト海沿岸地方をカバーし、Latino6 でサポートされなくなったラトビア語 (lv) を再追加し、地域で使用される疑問符が含まれる予定です。これは WinBaltic、すなわち windows-1257 と類似しています。

ISO-8859-14

ケルト語をカバーしています。Latin8 には Latin1 に含まれていたゲール語およびウェールズ語 (cy) が追加され、すべてのケルト語がカバーされるようになりました。

ISO-8859-15

Euro および oe 合字を含む Latin1 に類似しています。Latin0 と呼ばれる新しい Latin9 は Latin1 を更新したもので、稀にしか使用されない記号 ??? を、サポートさ

れていなかったフランス語およびフィンランド語の文字と置き換え、U+20AC Euro 記号を以前の国際通貨記号 ? のセル =A4 に置いています。

EUC-JP

日本語対応拡張 UNIX コード。

OSF、UNIX International、UNIX Systems Laboratories Pacific により標準化されました。以下の選択には ISO 2022 を使用しています。

- コード・セット 0：JIS ローマ字 (シングル 7 ビット・バイト・セット)
- コード・セット 1：上位下位両バイトが A0-FF に限られた JIS X0208-1990 (ダブル 8 ビット・バイト・セット)
- コード・セット 2：SS2 を文字の接頭辞に必要とする半角カタカナ (シングル 7 ビット・バイト・セット)
- コード・セット 3：SS3 を文字のプレフィックスに必要とし、上位下位両バイトが A0-FF に限られた JIS X0212-1990 (ダブル 7 ビット・バイト・セット)

Shift_JIS

最初のバイトの値が 81-9F または E0-EF の範囲にあるときに 2 つ目のバイトを追加することにより、csHalfWidthKatakana を拡張して漢字を含めた Microsoft のコード。

EUC-KR (KS C 5861-1992)

韓国語対応拡張 UNIX コード。

GB2312

中国により標準化されたマルチバイト・エンコーディング。

Big5

台湾により標準化されたマルチバイト・エンコーディング

Big5-HKSCS

香港用の補助文字セット。

EUC-TW (cns11643)

繁体中国語対応拡張 UNIX コード。

hp-roman8

HP 固有

IBM850

IBM 固有

windows-1250

Windows 3.1 東ヨーロッパ言語

windows-1251

Windows 3.1 キリル語

windows-1252

Windows 3.1 英語 (米国、ANSI)

windows-1253

Windows 3.1 ギリシャ語

windows-1254

Windows 3.1 トルコ語

windows-1255

ヘブライ語

> ***メモ:*** IBM ILOG Views *の本バージョンは双方向テキストをサポートしていません。*

windows-1256

アラビア語

> ***メモ:*** IBM ILOG Views *の本バージョンは双方向テキストをサポートしていません。*

windows-1257

バルト語

windows-1258

ベトナム語

windows-874

タイ語

windows-949

韓国語 (Wansung)

UTF-8

Unicode UTF-8

リファレンス：各プラットフォームでサポートされているロケール

以下は、Views ロケールとしてテスト済みのプラットフォームのリストです。

最初の表は、Microsoft Windows プラットフォームで現在サポートされているロケールを表示しています。特定のプラットフォームでサポートされているロケールに制限がある場合、最後の列に記載されています。この列が空白の場合、すべてのプラットフォーム (2000 から Vista まで) でサポートされています。

.

表 15.1 *Microsoft Windows ロケール・サポート*

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
Afrikaans_South Africa	1252	af_ZA.windows-1252	
Albanian_Albania	1250	sq_AL.windows-1250	
Arabic_Algeria	1256	ar_DZ.windows-1256	
Arabic_Bahrain	1256	ar_BH.windows-1256	
Arabic_Egypt	1256	ar_EG.windows-1256	
Arabic_Iraq	1256	ar_IQ.windows-1256	
Arabic_Jordan	1256	ar_JO.windows-1256	
Arabic_Kuwait	1256	ar_KW.windows-1256	
Arabic_Lebanon	1256	ar_LB.windows-1256	
Arabic_Libya	1256	ar_LY.windows-1256	
Arabic_Morocco	1256	ar_MA.windows-1256	
Arabic_Oman	1256	ar_OM.windows-1256	
Arabic_Qatar	1256	ar_QA.windows-1256	
Arabic_Saudi Arabia	1256	ar_SA.windows-1256	
Arabic_Syria	1256	ar_SY.windows-1256	
Arabic_Tunisia	1256	ar_TN.windows-1256	
Arabic_U.A.E.	1256	ar_AE.windows-1256	
Arabic_Yemen	1256	ar_YE.windows-1256	

表 15.1 Microsoft Windows ロケール・サポート （続き）

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
Azeri (Cyrillic)_Azerbaijan	1251	az_AZ.windows-1251	
Azeri (Latin)_Azerbaijan	1254	az_AZ.windows-1254	
Basque_Spain	1252	eu_ES.windows-1252	
Belarusian_Belarus	1251	be_BY.windows-1251	
Bulgarian_Bulgaria	1251	bg_BG.windows-1251	
Catalan_Spain	1252	ca_ES.windows-1252	
Chinese_Hong Kong	950	zh_HK.Big5	2000
Chinese_Hong Kong S.A.R	950	zh_HK.Big5-HKSCS	XP (下記の *HKSCS のサポート* を参照)
Chinese_Macau	950	zh_MO.Big5	2000
Chinese_People?s Republic of China	936	zh_CN.GB2312	
Chinese_Singapore	936	zh_SG.GB2312	
Chinese_Taiwan	950	zh_TW.Big5	
Croatian_Croatia	1250	hr_HR.windows-1250	
Czech_Czech Republic	1250	cs_CZ.windows-1250	
Danish_Denmark	1252	da_DK.windows-1252	
Dutch_Belgium	1252	nl_BE.windows-1252	
Dutch_Netherlands	1252	nl_NL.windows-1252	
English_Australia	1252	en_AU.windows-1252	
English_Belize	1252	en_BZ.windows-1252	
English_Ireland	1252	en_IE.windows-1252	
English_Jamaica	1252	en_JM.windows-1252	
English_New Zealand	1252	en_NZ.windows-1252	
English_Republic of the Philippines	1252	en_PH.windows-1252	

表 15.1 *Microsoft Windows ロケール・サポート （続き）*

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
English_South Africa	1252	en_ZA.windows-1252	
English_Trinidad y Tobago	1252	en_TT.windows-1252	2000
English_Zimbabwe	1252	en_ZW.windows-1252	
English_United States	1252	en_US.windows-1252	
English_United Kingdom	1252	en_GB.windows-1252	
Estonian_Estonia	1257	et_EE.windows-1257	
Faeroese_Faeroe Islands	1252	fo_FO.windows-1252	2000
Farsi_Iran	1256	fa_IR.windows-1256	
Finnish_Finland	1252	fi_FI.windows-1252	
French_Belgium	1252	fr_BE.windows-1252	
French_Canada	1252	fr_CA.windows-1252	
French_France	1252	fr_FR.windows-1252	
French_Luxembourg	1252	fr_LU.windows-1252	
French_Principality of Monaco	1252	fr_MC.windows-1252	
French_Switzerland	1252	fr_CH.windows-1252	
German_Austria	1252	de_AT.windows-1252	
German_Germany	1252	de_DE.windows-1252	
German_Liechtenstein	1252	de_LI.windows-1252	
German_Luxembourg	1252	de_LU.windows-1252	
German_Switzerland	1252	de_CH.windows-1252	
Greek_Greece	1253	el_GR.windows-1253	
Hebrew_Israel	1255	iw_IL.windows-1255	
Hungarian_Hungary	1250	hu_HU.windows-1250	
Icelandic_Iceland	1252	is_IS.windows-1252	

表 15.1 Microsoft Windows ロケール・サポート （続き）

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
Indonesian_Indonesia	1252	in_ID.windows-1252	
Italian_Italy	1252	it_IT.windows-1252	
Italian_Switzerland	1252	it_CH.windows-1252	
Kazakh_Kazakstan	1251	kk_KZ.windows-1251	
Japanese_Japan	932	ja_JP.Shift_JIS	
Korean_Korea	949	ko_KR.windows-949	
Latvian_Latvia	1257	lv_LV.windows-1257	
Lithuanian_Lithuania	1257	bo_LT.windows-1257	
Macedonian_Former Yugoslav Republic of Macedonia	1251	mk_MK.windows-1251	
Malay_Brunei Darussalam	1252	ms_BN.windows-1252	
Malay_Malaysia	1252	ms_MY.windows-1252	
Norwegian (BokmÃl)_Norway	1252	no_NO.windows-1252	
Norwegian (Nynorsk)_Norway	1252	no_NO.windows-1252	
Norwegian_Norway	1252	no_NO.windows-1252	2000
Polish_Poland	1250	pl_PL.windows-1250	
Portuguese_Brazil	1252	pt_BR.windows-1252	
Portuguese_Portugal	1252	pt_PT.windows-1252	
Romanian_Romania	1250	ro_RO.windows-1250	
Russian_Russia	1251	ru_RU.windows-1251	
Serbian (Latin)_Serbia	1250	sh_YU.windows-1250	2000
Serbian (Cyrillic)_Serbia	1251	sr_YU.windows-1251	2000
Slovak_Slovakia	1250	sk_SK.windows-1250	
Slovenian_Slovenia	1250	sl_SI.windows-1250	
Spanish_Argentina	1252	es_AR.windows-1252	

表 15.1 *Microsoft Windows ロケール・サポート （続き）*

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
Spanish_Bolivia	1252	es_BO.windows-1252	
Spanish_Chile	1252	es_CL.windows-1252	
Spanish_Colombia	1252	es_CO.windows-1252	
Spanish_Costa Rica	1252	es_CR.windows-1252	
Spanish_Dominican Republic	1252	es_DO.windows-1252	
Spanish_Ecuador	1252	es_EC.windows-1252	
Spanish_El Salvador	1252	es_SV.windows-1252	
Spanish_Guatemala	1252	es_GT.windows-1252	
Spanish_Mexico	1252	es_MX.windows-1252	
Spanish_Honduras	1252	es_HN.windows-1252	
Spanish_Nicaragua	1252	es_NI.windows-1252	
Spanish_Panama	1252	es_PA.windows-1252	
Spanish_Paraguay	1252	es_PY.windows-1252	
Spanish_Peru	1252	es_PE.windows-1252	
Spanish - Modern Sort_Spain	1252	es_ES.windows-1252	2000
Spanish_Puerto Rico	1252	es_PR.windows-1252	
Spanish - Traditional Sort_Spain	1252	es_ES.windows-1252	2000
Spanish_Spain	1252	es_ES.windows-1252	
Spanish_Uruguay	1252	es_UY.windows-1252	
Spanish_Venezuela	1252	es_VE.windows-1252	
Swahili_Kenya	1252	sw_KE.windows-1252	
Swedish_Finland	1252	sv_FI.windows-1252	
Swedish_Sweden	1252	sv_SE.windows-1252	

表 15.1 Microsoft Windows ロケール・サポート （続き）

Windows ロケール名	コード・ページ	Views ロケール名	Windows での制限
Tatar_Tatarstan	1251	tt_TS.windows-1251	2000
Thai_Thailand	874	th_TH.windows-874	
Turkish_Turkey	1254	tr_TR.windows-1254	
Ukrainian_Ukraine	1251	uk_UA.windows-1251	
Urdu_Islamic Republic of Pakistan	1256	ur_PK.windows-1256	
Uzbek_Republic of Uzbekistan	1251	uz_UZ.windows-1251	2000

HKSCS のサポート

香港用の補助文字セットをサポートするためには、専用のパッケージを Windows 2000 および Windows XP にインストールする必要があります (`http://www.microsoft.com/hk/hkscs/` を参照)。

表 15.2 HP-UX 11 ロケール・サポート

HP-UX ロケール名	エンコーディング	Views ロケール名
C	roman8	en_US.US-ASCII
POSIX	roman8	en_US.hp-roman8
C.iso88591	iso88591	en_US.ISO-8859-1
C.utf8	utf8	en_US.UTF-8
univ.utf8	utf8	en_US.UTF-8
ar_SA.iso88596	iso88596	ar_SA.ISO-8859-6
bg_BG.iso88595	iso88595	bg_BG.ISO-8859-5
cs_CZ.iso88592	iso88592	cs_CZ.ISO-8859-2
da_DK.iso88591	iso88591	da_DK.ISO-8859-1
da_DK.roman8	roman8	da_DK.hp-roman8
de_DE.iso88591	iso88591	de_DE.ISO-8859-1

表 15.2 HP-UX 11 ロケール・サポート （続き）

HP-UX ロケール名	エンコーディング	Views ロケール名
de_DE.roman8	roman8	de_DE.hp-roman8
el_GR.iso88597	iso88597	el_GR.ISO-8859-7
en_GB.iso88591	iso88591	en_GB.ISO-8859-1
en_GB.roman8	roman8	en_GB.hp-roman8
en_US.iso88591	iso88591	en_US.ISO-8859-1
en_US.roman8	roman8	en_US.hp-roman8
es_ES.iso88591	iso88591	es_ES.ISO-8859-1
es_ES.roman8	roman8	es_ES.hp-roman8
fi_FI.iso88591	iso88591	fi_FI.ISO-8859-1
fi_FI.roman8	roman8	fi_FI.hp-roman8
fr_CA.iso88591	iso88591	fr_CA.ISO-8859-1
fr_CA.roman8	roman8	fr_CA.hp-roman8
fr_FR.iso88591	iso88591	fr_FR.ISO-8859-1
fr_FR.roman8	roman8	fr_FR.hp-roman8
hr_HR.iso88592	iso88592	hr_HR.ISO-8859-2
hu_HU.iso88592	iso88592	hu_HU.ISO-8859-2
is_IS.iso88591	iso88591	is_IS.ISO-8859-1
is_IS.roman8	roman8	is_IS.hp-roman8
it_IT.iso88591	iso88591	it_IT.ISO-8859-1
it_IT.roman8	roman8	it_IT.hp-roman8
iw_IL.iso88598	iso88598	iw_IL.ISO-8859-8
ja_JP.SJIS	SJIS	ja_JP.Shift_JIS
ja_JP.eucJP	eucJP	ja_JP.EUC-JP
ko_KR.eucKR	eucKR	ko_KR.EUC-KR
nl_NL.iso88591	iso88591	nl_NL.ISO-8859-1

表 15.2 HP-UX 11 ロケール・サポート （続き）

HP-UX ロケール名	エンコーディング	Views ロケール名
nl_NL.roman8	roman8	nl_NL.hp-roman8
no_NO.iso88591	iso88591	no_NO.ISO-8859-1
no_NO.roman8	roman8	no_NO.hp-roman8
pl_PL.iso88592	iso88592	pl_PL.ISO-8859-2
pt_PT.iso88591	iso88591	pt_PT.ISO-8859-1
pt_PT.roman8	roman8	pt_PT.hp-roman8
ro_RO.iso88592	iso88592	ro_RO.ISO-8859-2
ru_RU.iso88595	iso88595	ru_RU.ISO-8859-5
sk_SK.iso88592	iso88592	sk_SK.ISO-8859-2
sl_SI.iso88592	iso88592	sl_SI.ISO-8859-2
sv_SE.iso88591	iso88591	sv_SE.ISO-8859-1
sv_SE.roman8	roman8	sv_SE.hp-roman8
tr_TR.iso88599	iso88599	tr_TR.ISO-8859-9
zh_CN.hp15CN	hp15CN	zh_CN.GB2312
zh_TW.big5	big5	zh_TW.Big5
zh_TW.eucTW	eucTW	zh_TW.EUC-TW

表 15.3 Solaris ロケール・サポート

Solaris ロケール名	エンコーディング	Views ロケール名
POSIX	646	en_US.US-ASCII
C	646	en_US.US-ASCII
iso_8859_1	ISO8859	en_US.US-ASCII
ar	ISO8859-6	ar_AA.ISO-8859-6
bg_BG	ISO8859-5	bg_BG.ISO-8859-5
cz	ISO8859-2	cs_CZ.ISO-8859-2

表 15.3 *Solaris ロケール・サポート（続き）*

Solaris ロケール名	エンコーディング	Views ロケール名
da	ISO8859-1	da_DK.ISO-8859-1
da.ISO8859-15	ISO8859-15	da_DK.ISO-8859-15
da.ISO8859-15@euro	ISO8859-15	da_DK.ISO-8859-15
de	ISO8859-1	de_DE.ISO-8859-1
de.ISO8859-15	ISO8859-15	de_DE.ISO-8859-15
de.ISO8859-15@euro	ISO8859-15	de_DE.ISO-8859-15
de.UTF-8	UTF-8	de_DE.UTF-8
de.UTF-8@euro	UTF-8	de_DE.UTF-8
de_AT	ISO8859-1	de_AT.ISO-8859-1
de_AT.ISO8859-15	ISO8859-15	de_AT.ISO-8859-15
de_AT.ISO8859-15@euro	ISO8859-15	de_AT.ISO-8859-15
de_CH	ISO8859-1	de_CH.ISO-8859-1
el	ISO8859-7	el_GR.ISO-8859-7
el.sun_eu_greek	sun_eu_greek	
en_AU	ISO-8859-1	en_AU.ISO-8859-1
en_CA	ISO8859-1	en_CA.ISO-8859-1
en_GB	ISO8859-1	en_GB.ISO-8859-1
en_GB.ISO8859-15	ISO8859-15	en_GB.ISO-8859-15
en_GB.ISO8859-15@euro	ISO8859-15	en_GB.ISO-8859-15
en_IE	ISO8859-1	en_IE.ISO-8859-1
en_IE.ISO8859-15	ISO8859-15	en_IE.ISO-8859-15
en_IE.ISO8859-15@euro	ISO8859-15	en_IE.ISO-8859-15
en_NZ	ISO8859-1	en_NZ.ISO-8859-1
en_US	ISO-8859-1	en_US.ISO-8859-1
en_US.UTF-8	UTF-8	en_US.UTF-8

表 15.3 Solaris ロケール・サポート（続き）

Solaris ロケール名	エンコーディング	Views ロケール名
es	ISO-8859-1	es_ES.ISO-8859-1
es.ISO8859-15	ISO8859-15	es_ES.ISO-8859-15
es.ISO8859-15@euro	ISO8859-15	es_ES.ISO-8859-15
es.UTF-8	UTF-8	es_ES.UTF-8
es.UTF-8@euro	UTF-8	es_ES.UTF-8
es_AR	ISO8859-1	es_AR.ISO-8859-1
es_BO	ISO8859-1	es_BO.ISO-8859-1
es_CL	ISO8859-1	es_CL.ISO-8859-1
es_CO	ISO8859-1	es_CO.ISO-8859-1
es_CR	ISO8859-1	es_CR.ISO-8859-1
es_EC	ISO8859-1	es_EC.ISO-8859-1
es_GT	ISO8859-1	es_GT.ISO-8859-1
es_MX	ISO8859-1	es_MX.ISO-8859-1
es_NI	ISO8859-1	es_NI.ISO-8859-1
es_PA	ISO8859-1	es_PA.ISO-8859-1
es_PE	ISO8859-1	es_PE.ISO-8859-1
es_PY	ISO8859-1	es_PY.ISO-8859-1
es_SV	ISO8859-1	es_SV.ISO-8859-1
es_UY	ISO8859-1	es_UY.ISO-8859-1
es_VE	ISO8859-1	es_VE.ISO-8859-1
et	ISO8859-1	et_EE.ISO-8859-1
fi	ISO8859-1	fi_FI.ISO-8859-1
fi.ISO8859-15	ISO8859-15	fi_FI.ISO-8859-15
fi.ISO8859-15@euro	ISO8859-15	fi_FI.ISO-8859-15
fr	ISO8859-1	fr_FR.ISO-8859-1

表 15.3 *Solaris ロケール・サポート（続き）*

Solaris ロケール名	エンコーディング	Views ロケール名
fr.ISO8859-15	ISO8859-15	fr_FR.ISO-8859-15
de.ISO8859-15@euro	ISO8859-15	fr_FR.ISO-8859-15
fr.UTF-8	UTF-8	fr_FR.UTF-8
fr.UTF-8@euro	UTF-8	fr_FR.UTF-8
fr_BE	ISO8859-1	fr_BE.ISO-8859-1
fr_BE.ISO8859-15	ISO8859-15	fr_BE.ISO-8859-15
fr_BE.ISO8859-15@euro	ISO8859-15	fr_BE.ISO-8859-15
fr_CA	ISO8859-1	fr_CA.ISO-8859-1
fr_CH	ISO8859-1	fr_CH.ISO-8859-1
hr_HR	ISO8859-2	hr_HR.ISO-8859-2
he	ISO8859-8	iw_IL.ISO-8859-8
hu	ISO8859-2	hu_HU.ISO-8859-2
it	ISO8859-1	it_IT.ISO-8859-1
it.ISO8859-15	ISO8859-15	it_IT.ISO-8859-15
it.ISO8859-15@euro	ISO8859-15	it_IT.ISO-8859-15
it.UTF-8	UTF-8	it_IT.UTF-8
it.UTF-8@euro	UTF-8	it_IT.UTF-8
lv	ISO8859-13	lv_LV.ISO-8859-13
lt	ISO8859-13	lt_LT.ISO-8859-13
mk_MK	ISO8859-5	mk_MK.ISO-8859-5
nl	ISO8859-1	nl_NL.ISO-8859-1
nl.ISO8859-15	ISO8859-15	nl_NL.ISO-8859-15
nl.ISO8859-15@euro	ISO8859-15	nl_NL.ISO-8859-15
nl_BE	ISO8859-1	nl_BE.ISO-8859-1
nl_BE.ISO8859-15	ISO8859-15	nl_BE.ISO-8859-15

表 15.3 Solaris ロケール・サポート（続き）

Solaris ロケール名	エンコーディング	Views ロケール名
nl_BE.ISO8859-15@euro	ISO8859-15	nl_BE.ISO-8859-15
no	ISO8859-1	no_NO.ISO-8859-1
no_NY	ISO8859-1	no_NY.ISO-8859-1
nr	ISO8859-2	nr_NA.ISO-8859-2
pl	ISO8859-2	pl_PL.ISO-8859-2
pt	ISO8859-1	pt_PT.ISO-8859-1
it.ISO8859-15	ISO8859-15	pt_PT.ISO-8859-15
pt.ISO8859-15@euro	ISO8859-15	pt_PT.ISO-8859-15
pt_BR	ISO8859-1	pt_BR.ISO-8859-1
ro_RO	ISO8859-2	ro_RO.ISO-8859-2
ru	ISO8859-5	ru_RU.ISO-8859-5
sk_SK	ISO8859-2	sk_SK.ISO-8859-2
sl_SI	ISO8859-2	sl_SI.ISO-8859-2
sq_AL	ISO8859-2	sq_AL.ISO-8859-2
sr_SP	ISO8859-5	sr_SP.ISO-8859-5
sv	ISO-8859-1	sv_SE.ISO-8859-1
sv.ISO8859-15	ISO8859-15	sv_SE.ISO-8859-15
sv.ISO8859-15@euro	ISO8859-15	sv_SE.ISO-8859-15
sv.UTF-8	UTF-8	sv_SE.UTF-8
sv.UTF-8@euro	UTF-8	sv_SE.UTF-8
th_TH	TIS620.2533	th_TH.ISO-8859-11
th	TIS620.2533	th_TH.ISO-8859-11
tr	ISO8859-9	tr_TR.ISO-8859-9

表 15.4 AIX ロケール・サポート

Aix ロケール名	エンコーディング	Views ロケール名
C	ISO8859-1	en_US.US-ASCII
POSIX	ISO8859-1	en_US.ISO-8859-1
ar_AA	ISO8859-6	ar_AA.ISO-8859-6
ar_AA.ISO8859-6	ISO8859-6	ar_AA.ISO-8859-6
Ar_AA		
Ar_AA.IBM-1046		
bg_BG	ISO8859-5	bg_BG.ISO-8859-5
bg_BG.ISO8859-5	ISO8859-5	bg_BG.ISO-8859-5
ca_ES	ISO8859-1	ca_ES.ISO-8859-1
ca_ES.ISO8859-1	ISO8859-1	ca_ES.ISO-8859-1
Ca_ES	IBM-850	ca_ES.IBM850
Ca_ES.IBM-850	IBM-850	ca_ES.IBM850
cs_CZ	ISO8859-2	cs_CZ.ISO-8859-2
cs_CZ.ISO8859-2	ISO8859-2	cs_CZ.ISO-8859-2
da_DK	ISO8859-1	da_DK.ISO-8859-1
da_DK.ISO8859-1	ISO8859-1	da_DK.ISO-8859-1
Da_DK	IBM-850	da_DK.IBM850
Da_DK.IBM-850	IBM-850	da_DK.IBM850
de_CH	ISO8859-1	de_CH.ISO-8859-1
de_CH.ISO8859-1	ISO8859-1	de_CH.ISO-8859-1
De_CH	IBM-850	de_CH.IBM850
De_CH.IBM-850	IBM-850	de_CH.IBM850
de_DE	ISO8859-1	de_DE.ISO-8859-1
de_DE.ISO8859-1	ISO8859-1	de_DE.ISO-8859-1
De_DE	IBM-850	de_DE.IBM850

表 15.4 AIX ロケール・サポート （続き）

Aix ロケール名	エンコーディング	Views ロケール名
De_DE.IBM-850	IBM-850	de_DE.IBM850
el_GR	ISO8859-7	el_GR.ISO-8859-7
el_GR.ISO8859-7	ISO8859-7	el_GR.ISO-8859-7
en_GB	ISO8859-1	en_GB.ISO-8859-1
en_GB.ISO8859-1	ISO8859-1	en_GB.ISO-8859-1
En_GB	IBM-850	en_GB.IBM850
En_GB.IBM-850	IBM-850	en_GB.IBM850
en_US	ISO8859-1	en_US.ISO-8859-1
en_US.ISO8859-1	ISO8859-1	en_US.ISO-8859-1
En_US	IBM-850	en_US.IBM850
En_US.IBM-850	IBM-850	en_US.IBM850
es_ES	ISO8859-1	es_ES.ISO-8859-1
es_ES.ISO8859-1	ISO8859-1	es_ES.ISO-8859-1
Es_ES	IBM-850	es_ES.IBM850
Es_ES.IBM-850	IBM-850	es_ES.IBM850
Et_EE		
Et_EE.IBM-922		
ET_EE	UTF-8	et_EE.UTF-8
ET_EE.UTF-8	UTF-8	et_EE.UTF-8
fi_FI	ISO8859-1	fi_FI.ISO-8859-1
fi_FI.ISO8859-1	ISO8859-1	fi_FI.ISO-8859-1
Fi_FI	IBM-850	fi_FI.IBM850
Fi_FI.IBM-850	IBM-850	fi_FI.IBM850
fr_BE	ISO8859-1	fr_BE.ISO-8859-1
fr_BE.ISO8859-1	ISO8859-1	fr_BE.ISO-8859-1

表 15.4 *AIX ロケール・サポート （続き）*

Aix ロケール名	エンコーディング	Views ロケール名
Fr_BE	IBM-850	fr_BE.IBM850
Fr_BE.IBM-850	IBM-850	fr_BE.IBM850
fr_CA	ISO8859-1	fr_CA.ISO-8859-1
fr_CA.ISO8859-1	ISO8859-1	fr_CA.ISO-8859-1
Fr_CA	IBM-850	fr_CA.IBM850
Fr_CA.IBM-850	IBM-850	fr_CA.IBM850
fr_CH	SO8859-1	fr_CH.ISO-8859-1
fr_CH.ISO8859-1	ISO8859-1	fr_CH.ISO-8859-1
Fr_CH	IBM-850	fr_CH.IBM850
Fr_CH.IBM-850	IBM-850	fr_CH.IBM850
fr_FR	ISO8859-1	fr_FR.ISO-8859-1
fr_FR.ISO8859-1	ISO8859-1	fr_FR.ISO-8859-1
Fr_FR	IBM-850	fr_FR.IBM850
Fr_FR.IBM-850	IBM-850	fr_FR.IBM850
hr_HR	ISO8859-2	hr_HR.ISO-8859-2
hr_HR.ISO8859-2	ISO8859-2	hr_HR.ISO-8859-2
hu_HU	ISO8859-2	hu_HU.ISO-8859-2
hu_HU.ISO8859-2	ISO8859-2	hu_HU.ISO-8859-2
is_IS	ISO8859-1	is_IS.ISO-8859-1
is_IS.ISO8859-1	ISO8859-1	is_IS.ISO-8859-1
Is_IS	IBM-850	is_IS.IBM850
Is_IS.IBM-850	IBM-850	is_IS.IBM850
it_IT	ISO8859-1	it_IT.ISO-8859-1
it_IT.ISO8859-1	ISO8859-1	it_IT.ISO-8859-1
It_IT	IBM-850	it_IT.IBM850

表 15.4 AIX ロケール・サポート （続き）

Aix ロケール名	エンコーディング	Views ロケール名
It_IT.IBM-850	IBM-850	it_IT.IBM850
iw_IL	ISO8859-8	iw_IL.ISO-8859-8
iw_IL.ISO8859-8	ISO8859-8	iw_IL.ISO-8859-8
Iw_IL		
Iw_IL.IBM-856		
ja_JP	IBM-eucJP	ja_JP.EUC-JP
ja_JP.IBM-eucJP	IBM-eucJP	ja_JP.EUC-JP
Ja_JP	IBM-932	ja_JP.Shift_JIS
Ja_JP.IBM-932	IBM-932	ja_JP.Shift_JIS
Jp_JP.pc932		
Jp_JP		
ko_KR	IBM-eucKR	ko_KR.EUC-KR
ko_KR.IBM-eucKR	IBM-eucKR	ko_KR.EUC-KR
Lt_LT		
Lt_LT.IBM-921		
LT_LT	UTF-8	lt_LT.UTF-8
LT_LT.UTF-8	UTF-8	lt_LT.UTF-8
Lv_LV		
Lv_LV.IBM-921		
LV_LV	UTF-8	lv_LV.UTF-8
LV_LV.UTF-8	UTF-8	lv_LV.UTF-8
mk_MK	ISO8859-5	mk_MK.ISO-8859-5
mk_MK.ISO8859-5	ISO8859-5	mk_MK.ISO-8859-5
nl_BE	ISO8859-1	nl_BE.ISO-8859-1
nl_BE.ISO8859-1	ISO8859-1	nl_BE.ISO-8859-1

表 15.4 AIX ロケール・サポート （続き）

Aix ロケール名	エンコーディング	Views ロケール名
Nl_BE	IBM-850	nl_BE.IBM850
Nl_BE.IBM-850	IBM-850	nl_BE.IBM850
nl_NL	ISO8859-1	nl_NL.ISO-8859-1
nl_NL.ISO8859-1	ISO8859-1	nl_NL.ISO-8859-1
Nl_NL	IBM-850	nl_NL.IBM850
Nl_NL.IBM-850	IBM-850	nl_NL.IBM850
no_NO	ISO8859-1	no_NO.ISO-8859-1
no_NO.ISO8859-1	ISO8859-1	no_NO.ISO-8859-1
No_NO	IBM-850	no_NO.IBM850
No_NO.IBM-850	IBM-850	no_NO.IBM850
pl_PL	ISO8859-2	pl_PL.ISO-8859-2
pl_PL.ISO8859-2	ISO8859-2	pl_PL.ISO-8859-2
pt_BR	ISO8859-1	pt_BR.ISO-8859-1
pt_BR.ISO8859-1	ISO8859-1	pt_BR.ISO-8859-1
pt_PT	ISO8859-1	pt_PT.ISO-8859-1
pt_PT.ISO8859-1	ISO8859-1	pt_PT.ISO-8859-1
Pt_PT	IBM-850	pt_PT.IBM850
Pt_PT.IBM-850	IBM-850	pt_PT.IBM850
ro_RO	ISO8859-2	ro_RO.ISO-8859-2
ro_RO.ISO8859-2	ISO8859-2	ro_RO.ISO-8859-2
ru_RU	ISO8859-5	ru_RU.ISO-8859-5
ru_RU.ISO8859-5	ISO8859-5	ru_RU.ISO-8859-5
sh_SP	ISO8859-2	sh_SP.ISO-8859-2
sh_SP.ISO8859-2	ISO8859-2	sh_SP.ISO-8859-2
sk_SK	ISO8859-2	sk_SK.ISO-8859-2

表 15.4 AIX ロケール・サポート （続き）

Aix ロケール名	エンコーディング	Views ロケール名
sk_SK.ISO8859-2	ISO8859-2	sk_SK.ISO-8859-2
sl_SI	ISO8859-2	sl_SI.ISO-8859-2
sl_SI.ISO8859-2	ISO8859-2	sl_SI.ISO-8859-2
sq_AL	ISO8859-1	sq_AL.ISO-8859-1
sq_AL.ISO8859-1	ISO8859-1	sq_AL.ISO-8859-1
sr_SP	ISO8859-5	sr_SP.ISO-8859-5
sr_SP.ISO8859-5	ISO8859-5	sr_SP.ISO-8859-5
sv_SE	ISO8859-1	sv_SE.ISO-8859-1
sv_SE.ISO8859-1	ISO8859-1	sv_SE.ISO-8859-1
Sv_SE	IBM-850	sv_SE.IBM850
Sv_SE.IBM-850	IBM-850	sv_SE.IBM850
tr_TR	ISO8859-9	tr_TR.ISO-8859-9
tr_TR.ISO8859-9	ISO8859-9	tr_TR.ISO-8859-9
zh_CN	IBM-eucCN	zh_CN.GB2312
zh_CN.IBM-eucCN	IBM-eucCN	zh_CN.GB2312
ZH_CN	UTF-8	zh_CN.UTF-8
ZH_CN.UTF-8	UTF-8	zh_CN.UTF-8
zh_TW	IBM-eucTW	zh_TW.EUC-TW
zh_TW.IBM-eucTW	IBM-eucTW	zh_TW.EUC-TW
Zh_TW	big5	zh_TW.Big5
Zh_TW.big5	big5	zh_TW.Big5

表 15.5 OSF ロケール・サポート

Osf ロケール名	エンコーディング	Views ロケール名
C	ISO8859-1	en_US.US-ASCII
POSIX	ISO8859-1	en_US.ISO-8859-1
da_DK.ISO8859-1	ISO8859-1	da_DK.ISO-8859-1
de_CH.ISO8859-1	ISO8859-1	de_CH.ISO-8859-1
de_DE.ISO8859-1	ISO8859-1	de_DE.ISO-8859-1
el_GR.ISO8859-7	ISO8859-7	el_GR.ISO-8859-7
en_GB.ISO8859-1	ISO8859-1	en_GB.ISO-8859-1
en_US.ISO8859-1	ISO8859-1	en_US.ISO-8859-1
en_US.cp850	cp850	en_US.IBM850
es_ES.ISO8859-1	ISO8859-1	es_ES.ISO-8859-1
fi_FI.ISO8859-1	ISO8859-1	fi_FI.ISO-8859-1
fr_BE.ISO8859-1	ISO8859-1	fr_BE.ISO-8859-1
fr_CA.ISO8859-1	ISO8859-1	fr_CA.ISO-8859-1
fr_CH.ISO8859-1	ISO8859-1	fr_CH.ISO-8859-1
fr_FR.ISO8859-1	ISO8859-1	fr_FR.ISO-8859-1
is_IS.ISO8859-1	ISO8859-1	is_IS.ISO-8859-1
it_IT.ISO8859-1	ISO8859-1	it_IT.ISO-8859-1
nl_BE.ISO8859-1	ISO8859-1	nl_BE.ISO-8859-1
nl_NL.ISO8859-1	ISO8859-1	nl_NL.ISO-8859-1
no_NO.ISO8859-1	ISO8859-1	no_NO.ISO-8859-1
pt_PT.ISO8859-1	ISO8859-1	pt_PT.ISO-8859-1
sv_SE.ISO8859-1	ISO8859-1	sv_SE.ISO-8859-1
tr_TR.ISO8859-9	ISO8859-9	tr_TR.ISO-8859-9

A

IBM ILOG Views アプリケーションのパッケージ化

このセクションでは、IBM® ILOG® Views の提供するツール `ilv2data` の使用方法を説明します。これはアプリケーション・データ・ファイルを IBM ILOG Views アプリケーションと同じ実行可能ファイルに安全にパッケージ化するためのものです。

ilv2data とは？

`ilv2data` 実行可能ファイルを使用すると、`.ilv` ファイルや `dbm` ファイルおよびビットマップ (`.gif`、`.bmp`、`.pbm` など) などのすべてのアプリケーション・リソースを `ilv2data` で生成される 1 つのファイルにまとめ、アプリケーション・プロジェクトに追加する (PC の場合) か、またはコンパイルして IBM ILOG Views アプリケーションにリンクする (UNIX の場合) ことができます。

このファイルは Microsoft Windows の*データ・リソース・ファイル* (`.rc`) であり、Microsoft Resource Compler (`RC.EXE`) でコンパイルできます。UNIX プラットフォームでは、このファイルは静的データの定義のみを含む通常の C++ ソース・ファイルです。このファイルは通常の C++ コンパイラでコンパイルできます。このセクションでは、これ以降このファイルを*リソース・ファイル*と呼びます。

リソース・ファイルは、リソース・ファイルの構築時に関連付けられた名前を使用して実行時に取得できる一連のデータ・ブロックを格納します。

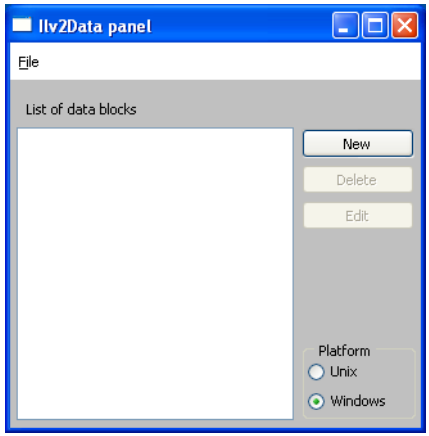
このセクションの内容

ilv2data の起動

`ilv2data` を起動する

1. ディレクトリ `<ILVHOME>/bin/<system>` に移動します。
2. まだ実行可能ファイルがない場合は、それをコンパイルします (`ilv2data` は Gadgets パッケージを使用することに注意してください)。
3. `ilv2data` と入力して実行可能ファイルを起動します。

   次のパネルが表示されます。



ilv2data パネル

`ilv2data` パネルは、以下の要素で構成されます。

- リソース・ファイルを処理するために使用される [ファイル] (File) メニュー。リソース・ファイルとは、アプリケーションにパッケージ化するすべてのリソースを加えるファイルです。完成して有効にすると、このファイルは選択したプ

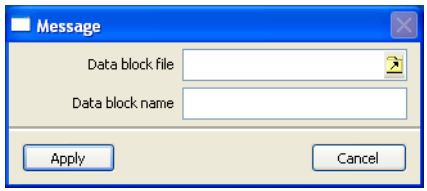
ラットフォームに応じて .rc ファイルまたは .cpp ファイルとして保存されます (Microsoft Windows または UNIX)。**ファイル**・メニューには以下のメニュー・アイテムがあります。

- 新規作成 (New) - 新規リソース・ファイルを作成します。
- 開く (Open) - リソース・ファイルを開きます。
- 保存 (Save) - リソース・ファイルにデータを保存し、.rc ファイルまたは .cpp ファイルを生成します (選択したプラットフォームに応じて決まります)。

◆ 3 つのボタンがあります。

- 新規 (New) - データ・ブロックをリストに追加します。
- 削除 (Delete) - データ・ブロックをリストから削除します。
- 編集 (Edit) - リスト内の選択アイテムに関連する値を編集します。

[新規] または [編集] をクリックすると、以下のダイアログ・ボックスが表示されます。



データ・ブロック・ファイル入力フィールドには、リストに追加したいリソース・ファイルの物理名を入力します。ファイルを探すためにファイル・ブラウザを使用したい場合、入力フィールドの右にあるアイコンをクリックしてファイル・チューザを表示します。

デフォルトでデータ・ブロック名フィールドには、プログラムがデータ・ブロックを読み込むために使用するプログラムの論理名が設定されます。最初は、この名前はデータ・ブロック・ファイル入力フィールドに入力したものと同じです。

[適用] (Apply) ボタンによってデータが検査され、[取り消し] (Cancel) ボタンでプロシージャが中止されます。

> ***メモ:*** *.dbm ファイルなどの Views データ・ファイル (または Data Access などの拡張 Views) を追加する場合は、*`$ILVHOME/data`* からファイルへのパスを忘れずに付けてください。たとえば、*`dataccess.dbm`* ファイルを追加する場合、データ・ブロック名は* `dataccess/dataccess.dbm` *とする必要があります。これは、*`dataccess.dbm`* のフル・ファイル名が* `$ILVHOME/data/dataccess/dataccess.dbm` *となっているためです。*

バッチ・コマンドで ilv2data を起動する

さまざまな基本操作を実行するために各種オプションを指定して、コマンド・ラインから `ilv2data` を起動できます。

利用可能なオプションとその説明は、下記のとおりです。

```
ilv2data [-a key[=val]] [-c] [-d key] [-h] [-i dir] [-l]
[-m key[=val]] [-u|w] [-v 0|1] file
```

- `-a key[=val]`: 追加オプション

  データ・ブロック名 `key` をリソースのリストに追加します。`val` は、挿入するファイルの名前を指定します。デフォルトの値は `key` です。

- `-c`: チェック・オプション

  `file` の一貫性をチェックします。

- `-d key`: 削除オプション

  データ・ブロック・キーを、リソースのリストから削除します。

- `-h`: ヘルプ・オプション

  コマンドの使用方法を表示します。

- `-i dir`: インクルード・オプション

  データ・ブロック・ファイルの検索先パスのリストに、ディレクトリ `dir` を追加します。

- `-l`: リスト・オプション

  `file` で利用可能なデータ・ブロックをすべて一覧化します。

- `-m key[=val]`: 変更オプション

  データ・ブロック `key` を、ファイル `val` で更新します。

- `-u|w`: 再生成オプション

ファイル file を UNIX モード (-u) または Windows モード (-w) で再生成します。表示パスでファイルを探すには、データ・ブロックの名前を使用します。このファイルがあれば使用します。なければ、file に含まれるデータ・ブロックの以前の定義を使用します。

- -v 0|1 : 冗長オプション

  オプションが 1 に設定されている場合は、実行時にコメントを印刷します。オプションが 0 に設定されている場合でも、エラーと警告は表示されます。

このコマンドは、実行が成功した場合は 0 を、失敗した場合は 1 を返します。

UNIX ライブラリにリソース・ファイルを追加する

UNIX ライブラリにリソース・ファイルを追加するには、最終アプリケーションから呼び出されることがわかっているライブラリ内のモジュールに、以下の 2 行を追加します。

```
extern IlUInt IL_MODINIT(<name>Resources)();
static IlUInt forceRes = IL_MODINIT(<name>Resources)();
```

ilv2data は、次の形式でファイル名を生成します。<name>.cpp.

Windows DLL にリソース・ファイルを追加する

すべての DLL モジュールに次の行を追加します。

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

extern "C" {
    void _declspec(dllimport) IlvAddHandleToResPath(long, int);
    void _declspec(dllimport) IlvRemoveHandleFromResPath(long);
}

BOOL WINAPI
DllEntryPoint(HINSTANCE instance, DWORD reason, LPVOID)
{
    switch (reason) {
    case DLL_PROCESS_ATTACH:
        IlvAddHandleToResPath((long)instance, -1);
        return 1;
    case DLL_PROCESS_DETACH:
        IlvRemoveHandleFromResPath((long)instance);
        return 0;
    }
    return 0;
}
BOOL WINAPI
```

```
DllMain(HINSTANCE hinstance, DWORD reason, LPVOID reserved)
{
    return DllEntryPoint(hinstance, reason, reserved);
}
```

B

IBM ILOG Views を Microsoft Windows で使用する

このセクションは、Microsoft Windows 上でアプリケーションを開発するプログラマや IBM® ILOG® Views と Windows のコードをマージするプログラマを対象に書かれたものです。以下のトピックから構成されています。

- *IBM ILOG Views アプリケーションを Microsoft Windows 上で新規作成する*
- *Windows コードを IBM ILOG Views アプリケーションに組み込む*
- *IBM ILOG Views コードを Windows アプリケーションに組み込む*
- *Microsoft Windows 上で実行するアプリケーションの終了*
- *Windows 特有のデバイス*
- *IBM ILOG Views で GDI+ 機能を使用する*
- *IBM ILOG Views で複数表示モニタを使用する*

IBM ILOG Views アプリケーションを Microsoft Windows 上で新規作成する

Windows コードが含まれない新規の IBM® ILOG® Views アプリケーションを作成する場合は、main 関数を作成してそのコンストラクタにアプリケーション名を提供することにより IlvDisplay クラスをインスタンス化します。

```
int
main(int argc, char* argv[])
{
    IlvDisplay* display = new IlvDisplay("IlogViews", "", argc, argv);
    ...
}
```

「main」は Microsoft Windows 上で稼働するアプリケーションの標準エントリ・ポイントではない点に注意してください (標準では「WinMain」です)。
IBM ILOG Views のソース・コードの移植性や、コマンドライン・パラメータの構文解析を容易にするため、標準の C++ 「main」エントリ・ポイントを使用します。この選択の効果は、このトピックで詳しく説明します。

リソース・スキャニングではアプリケーション名を使用しています (*ディスプレイ・システム・リソース*を参照)。Microsoft Windows の場合は 2 つ目の引数は使用しません。そのため、空の文字列に置き換えられます。(これは X Window で X display に対応する場合に使用されます。)Microsoft Windows では最後の 2 つのパラメータも使用しません。

次に、ビュー構造およびオブジェクトを構築し、グローバル関数 IlvMainLoop を呼び出すことができます。

```
int
main(int argc, char* argv[])
{
    IlvDisplay* display = new IlvDisplay("IlogViews", "", argc, argv);
    ...
    ...
    IlvMainLoop();
    return 0;
}
```

ここで、アプリケーションを開始するために Microsoft Windows が予期する WinMain エントリ・ポイントの代わりに main 関数が提供されるため、オブジェクト・ファイルを ILVMAIN.OBJ ファイルにリンクする必要があります。このファイルは IBM ILOG Views に含まれており、必要なすべての初期化操作を行うデフォルトの WinMain 関数を定義し、main 関数を呼び出します。

> ***メモ:*** *一部のコンパイラが提供する* main *関数のその他の定義との矛盾を回避するため、プリプロセッサ・マクロは* main *関数を* IlvMain *として再定義します。このマクロは、ヘッダー・ファイル* <ilviews/ilv.h> *で宣言されています。*

たとえば、BIN ディレクトリの make ファイルや project ファイルを参照してください。

Windows コードを IBM ILOG Views アプリケーションに組み込む

Microsoft Windows がサポートする数々のインターフェース・ジェネレータのいずれかで作成された Windows メニューやパネルを、独自の IBM® ILOG® Views アプリケーションに簡単に組み込むことができます。この例は、<ILVHOME>\samples の下にあるサブディレクトリ foundation\windows で参照できます。オンライン・マニュアルの『Views Foundation チュートリアル』も参照してください。

以下の例は、いずれかのインターフェース・ビルダで作成されたパネルをリソース・コンパイラでアプリケーションにリンクした例を示しています。

```
#define VIEW_ID 1010 // The ID of a sub-window in the panel
int PASCAL ILVEXPORTED
DialogProc(HWND dlg, UINT msg, WPARAM wParam, LPARAM lParam)
{
  switch (msg) {
  case WM_INITDIALOG:
     // Create some IlogViews object in the dialog.
     InitIlogViews((IlvDisplay*)lParam, GetDlgItem(dlg, VIEW_ID));
     return 1;
  case WM_COMMAND:
     if (wParam == QUIT_ID) {
         EndDialog(dlg, 1);    // Close the dialog
         ReleaseIlogViews();   // Delete IlogViews objects
         PostQuitMessage(0);   // Exit the event loop
         return 1;
     }
  }
  return 0;
}


int
main(int argc, char* argv[])
{
     // Connect to the windowing system.
     IlvDisplay* display = new IlvDisplay("IlogViews", "", argc, argv);
     if (display->isBad()) {
         IlvFatalError("Couldnít connect to display system");
         delete display;
         return 1;
     }
     // Create the dialog box.
     if (DialogBoxParam(display->getInstance(), "MY_PANEL", 0,
                        (FARPROC)DialogProc, (long)display) == -1)
        IlvFatalError("Couldnít create dialog");
     delete display;
     return 1;
}
```

```
void
InitIlogViews(IlvDisplay* display, HWND wnd)
{
    // For example:a container that uses the ìwndî window.
    container = new IlvContainer(display, wnd);
    ...
}void
ReleaseIlogViews()
{
    delete container;
}
```

InitIlogViews メンバ関数では、既存の Windows パネル wnd を保持する新規の IlvContainer オブジェクトが作成されます。ユーザ・インターフェース・ジェネレータでは、そのウィンドウで使用する WindowsClass が IlogViewsWndClass であることを指定する必要があります。

この例では、アプリケーションを開始するために Microsoft Windows が予期する WinMain エントリ・ポイントの代わりに main 関数が提供されるため、オブジェクト・ファイルを ILVMAIN.OBJ ファイルにリンクする必要があります。このファイルは IBM ILOG Views に含まれており、必要なすべての初期化操作を行うデフォルトの WinMain 関数を定義し、main 関数を呼び出します。

IBM ILOG Views コードを Windows アプリケーションに組み込む

IBM® ILOG® Views コードを Microsoft Windows 上で実行する既存のアプリケーションに組み込むには、IlvDisplay クラスの 2 番目のコンストラクタを使用するだけです。これは、そのアプリケーションのインスタンスを引数として取ります。

```
int PASCAL
WinMain(HANDLE appInstance, HANDLE, LPSTR, int)
{
    ...
    IlvDisplay* display = new IlvDisplay((IlAny)appInstance,
                                         "ApplicationName");
    ...
}
```

ただし、IlvDisplay オブジェクトを削除しても QUIT メッセージは表示されません。これはイベント・ループを終了しないための措置です。IBM ILOG Views セッションを閉じた後も、作業を続けたい場合があるからです。

ここで、使用アプリケーションに WinMain エントリ・ポイントを提供するため、ILVMAIN.OBJ ファイルを使用して実行可能ファイルをリンクする必要はありません。

Microsoft Windows 上で実行するアプリケーションの終了

すべてのオペレーティング・システムでは、アプリケーションを終了する前にメモリやシステム・リソースを解放することが推奨されます。Microsoft Windows の初期のバージョン (3.1、95) では、これは特に重要でした。システムの GDI リソース (色、フォントなど) に大きな制約があり、自動的に解放できなかったためです。後続バージョン (NT 4、2000、XP) では、この点は改善されています。しかし、終了する前にシステム・リソースとメモリを解放するという正当な方法でアプリケーションを終了することをお勧めします。アプリケーション・データを解放し、`IlvDisplay` と `call IlvExit(0)` を削除する関数を記述すると、これを楽に行うことができます。この関数は、アクセラレータ、ボタン・コールバック、トップ・ウィンドウ破棄コールバックなどとして使用できます。

> ***メモ:*** *すべてのマネージャおよび表示インスタンスは削除する必要があります。コンテナとマネージャを破棄すると、それらの格納されていたオブジェクトが削除されることに注意してください。マネージャの情報については、『IBM ILOG Views Managers』のマニュアルを参照してください。*

> ***メモ:*** *IBM ILOG Views は動的に割り当てられる内部メモリを使用します。このメモリはアプリケーションの終了時にのみ解放されます。*

Windows 特有のデバイス

Windows デバイスを管理するために、IBM® ILOG® Views は次の 2 つのクラスを提供しています。`IlvWindowsVirtualDevice` および `IlvWindowsDevice`。

印刷

IBM ILOG Views から Microsoft Windows が制御する任意のプリンタに印刷するには、`IlvWindowsDevice` ダンプ・デバイスを使用します。

プリンタの選択

プリンタを選択する場合は、以下のグローバル関数を呼び出します。

```
const char*
IlvGetWindowsPrinter (Ilboolean dialog = IlTrue);
```

この関数は、使用するプリンタを説明する文字列を返します。この文字列は、内的に管理されるため、変更または削除できません。

ダイアログ・パラメータに IlTrue 値を設定して呼び出すとダイアログ・ボックスが表示されるので、使用するプリンタおよび適用するサイズ・パラメータと向きパラメータを指定できます。この関数が IlFalse パラメータで呼び出されると、現在のデフォルト・プリンタを記述した文字列が返されます。エラーが発生するか、[取消し] ボタンをクリックすると、ヌルが返されます。

IBM ILOG Views で GDI+ 機能を使用する

GDI+ について

GDI+ は、Microsoft Windows プラットフォーム上で描画を行う方法です。透明度やアンチエイリアシングなどの興味深い機能を提供します。GDI+ の追加情報は、Microsoft のインターネット・サイトを参照してください。

ダイナミック・リンク・ライブラリ (DLL) の使用

ダイナミック IBM ILOG Views ライブラリ (dll_mda) を使用すると、GDI+ を簡単に使用できます。Microsoft は ILOG Views アプリケーションからアクセスできる DLL (gdiplus.dll) を提供しています。この DLL は、ダイナミック IBM ILOG Views ライブラリ (dll_mda) と同じディレクトリで提供されています。最新の再配布可能 gdiplus.dll は、http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm. でダウンロードできます。

スタティック・ライブラリの使用

スタティック IBM ILOG Views ライブラリ (stat_mda、stat_mta) を使用する場合は、Microsoft Platform SDK をインストールする必要があります。これは、アプリケーションを gdiplus.lib ライブラリとリンクするためです。この SDK は次のサイトで取得できます。http://www.microsoft.com/msdownload/platformsdk/sdkupdate

また、コンパイルする際に <ilviews/windows/ilvgdiplus.h> ファイルもインクルードし、アプリケーションを ilvgdiplus.lib ライブラリにリンクする必要があります。このライブラリは、ディレクトリ ILVHOME/lib/[platform]/[subplatform] にあります。ここで、ILVHOME は IBM ILOG Views がインストールされたルート・ディレクトリ、*subplatform* は stat_mda or stat_mta、platform (プラットフォーム) は次のいずれかになります。

- x86_.net2003_7.1
- x86_.net2005_8.0
- x86_.net2008_9.0

GDI+ および IBM ILOG Views

GDI+ がインストールされると、IBM ILOG Views はこれを活用するために `IlvPalette` クラスおよび `IlvPort` クラスに専用の API を提供します。透明度とアンチエイリアシングを処理するために、以下のメソッドが追加されました。

- `IlvPalette::setAlpha`
- `IlvPalette::getAlpha`
- `IlvPort::setAlpha`
- `IlvPort::getAlpha`
- `IlvPalette::setAntialiasingMode`
- `IlvPalette::getAntialiasingMode`
- `IlvPort::setAntialiasingMode`
- `IlvPort::getAntialiasingMode`

87 ページの *アルファ値* と 87 ページの *アンチエイリアシング・モード* を参照してください。

GDI+ 機能のランタイム制御

リソース全体に対して GDI+ を使用するかどうかを指定することができます。以下の表は、異なるリソースとその可能な値、およびそれぞれの値の効果の概要をまとめたものです。

***表 B.1** GDI+ リソース*

リソース (.ini ファイル)	環境変数	値
UseGdiPlus	ILVUSEGDIPLUS	**needed**:GDI+ は必要なときにのみ使用されます。たとえば、透明度やアンチエイリアシングが必要な場合です。これはデフォルトです。 **true**: 可能な場合に GDI+ を使用します。 **false**:GDI+ は使用できません。
Antialiasing	ILVANTIALIASING	**false**: ディスプレイのアンチエイリアシング・モードは IlvNoAntialiasingMode に設定されます。これはデフォルトです。 **true**: ディスプレイのアンチエイリアシング・モードは IlvUseAntialiasingMode に設定されます。

たとえば、以下の views.ini ファイルにより、アプリケーション全体でアンチエイリアシングが有効になります。UseGdiPlus リソースが指定されていないため、デフォルトが使用されます。つまり、GDI+ は必要な場合にのみ使用されます。

```
[IlogViews]

Antialiasing=true
```

制約

以下の表は、IBM ILOG Views で GDI+ を使用する場合の制約とサポートされない機能の概要です。

***表 B.2** GDI+ を IBM ILOG Views で使用する場合の制約*

フォント	GDI+ は True Type フォントをサポートします。True Type フォント以外のフォントを使用する場合、GDI+ は使用できません。さらに、トランスフォーマで文字列を描画する場合、文字列を垂直に描画したりすると、GDI で行ったレンダリングと結果が多少異なる場合があります。これは、GDI+ と GDI では使用するレンダリング・エンジンが完全に同一ではないためです。
ブラシ	Windows のハッチ・パターンは、GDI+ HatchStyle に完全にはマッピングできません。そのため、GDI+ に切り替えると、パターンの描画が少し変わる可能性があります。
印刷	IBM ILOG Views はプリンタへの描画時には GDI+ を使用しません。
円弧	GDI+ では平面の円弧を正しく描画できません。そのため、GDI+ は平面の円弧の描画に使われません。
描画モード	GDI+ は `IlvModeSet` 以外のモードはサポートしません。たとえば、`IlvModeXor` を使用する場合、GDI+ は使用されません。
Windows XP のルック・アンド・フィール	Windows XP で描画されたガジェットでは、透明度とアンチエイリアシングは利用できません。

IBM ILOG Views で複数表示モニタを使用する

複数表示モニタは、複数の表示デバイスをアプリケーションで同時に使用できるようにする機能セットを提供します。複数のモニタが 1 つの大きなモニタとして表示されるため、1 つの画面から他の画面へと移動できるようになります。

IBM® ILOG® Views はこの機能を考慮しており、モニタの座標を取得するために API (`IlvDisplay::screenBBox`) が追加されています。このメソッドにより、アプリケーションは特定の矩形が位置するモニタの座標を取得できます。たとえば、単一のモニタの中央に 1 つのウィンドウを配置するためにこれを使用します。詳細については、リファレンス・マニュアルを参照してください。

既存のアプリケーションに対するこの機能の影響は、トップ・ウィンドウの管理に限られます。トップ・ウィンドウを表示するたびに、その位置を正確に計算する必要があります。問題を回避するには、トップ・ウィンドウの位置は画面ではなく他のトップ・ウィンドウに対して相対的に設定します。たとえば、大部分のアプリケーションにはメイン・パネルと複数のダイアログがあり、すべてがトップ・ウィンドウです。ダイアログの位置は、画面全体に対して中央に設定する (`IlvView::ensureInScreen` メソッドを使用する) よりも、メイン・パネルの位置に対して指定する (`IlvView::moveToView` メソッドを使用する) 方が良いでしょう。

> ***メモ:*** `IlvView::ensureInScreen` *メソッドは、ビューをモニタ内に設定します。ビューが配置されるモニタは、作業モニタと見なされます。たとえば、ビューがモニタ 2 にある場合、ビューで* `IlvView::ensureInScreen` *を呼び出すと、ビューがモニタ 2 に残ります。*

C

IBM ILOG Views を X Window システムで使用する

この付録では、UNIX 環境の X Window システムで IBM® ILOG® Views を使用するための情報を提供します。

- *ライブラリ*では、Xlib または Motif をベースにした 2 つのバージョンの IBM ILOG Views を説明します。
- *新規入力ソースの追加*による機能
- *ONC-RPC 統合*の実行
- *libmviews を使用して IBM ILOG Views を Motif アプリケーションと統合する*
- *libxviews を使用して IBM ILOG Views を X アプリケーションと統合する*

ライブラリ

IBM® ILOG® Views ライブラリは 2 つバージョンで提供されています。

- Xlib をベースにした `libxviews`
- Motif をベースにした `libmviews`

IBM ILOG Views アプリケーションを開発するときには、純粋な Xlib アプリケーションまたは Motif と統合する方が簡単なアプリケーションのいずれかを、リンク時に作成できます。作成するアプリケーションの種類に応じて、ファイルを

libxviews (純粋な Xlib アプリケーションの場合) または libmviews (Motif ベースのアプリケーションの場合) のいずれかとリンクします。ソース・コードはリンク先として選択したライブラリからは独立しています。

これらのライブラリを使用するための詳細については、以下を参照してください。

- *Xlib バージョン、libxviews の使用*
- *Motif バージョン、libmviews の使用*

Xlib バージョン、libxviews の使用

IlvDisplay オブジェクトを作成すると、ディスプレイ・システムとの通常の接続が確立します。X Window 側から見ると、IlvDisplay に提供される IlvSystemView タイプは Window タイプと同等です。イベント・ループ管理は、ディスプレイ・システムへの接続に対応するファイル記述子での select 呼び出しに基づいています。libxviews ライブラリと libX11 ライブラリにリンクします。

制限

初期の IBM ILOG Views リリースでは、ガジェットがまだ提供されておらず、標準ダイアログを中心とした基本ポータブル GUI コンポーネントの数々は Motif (UNIX の場合) および Microsoft SDK (Windows の場合) を使用して実装されていました。IBM ILOG Views ではこれらの機能の代わりとなる新しい同等のコンポーネントを提供していますが、下位互換性を保つため、これらの機能もそのまま維持されています。これらは libmviews に実装されていますが、Motif ベースではない libxviews には実装されていません。詳細は、以下のとおりです。

- 標準システム・ダイアログ : IlvPromptDialog, IlvInformationDialog、IlvQuestionDialog、IlvFileSelector および IlvPromptStringsDialog。

  これらのクラスは、ヘッダー・ファイル ilviews/dialogs.h で宣言されています。ガジェット・ライブラリ libilvgadgt は、類似ダイアログのポータブル・バージョン (純粋な IBM ILOG Views コード) を提供します。ilviews/stdialog.h ヘッダー・ファイルを参照してください。

- Motif XmScrolledWindow をベースにした IlvScrollView

  クラス IlvScrolledView と IlvScrolledGadget は類似のサービスを提供します。

Motif バージョン、libmviews の使用

IlvDisplay オブジェクトを作成すると、Xt ライブラリが初期化され、トップ・シェル・ウィジェットが作成されます。メンバ関数 IlvAbstractView::getSystemView または IlvAbstractView::getShellSystemView は実際の Motif ウィジェットです。イベント・ループ管理は XtAppMainLoop の呼び出しとまったく同じです。プラット

フォームに Motif をインストールし、libmviews ライブラリを libXm ライブラリ、libXt ライブラリおよび libX11 ライブラリとリンクする必要があります。

これらの相違点はこの付録の残りの部分で詳しく説明します。また、どちらかのモードを使用する方法の例も提供されています。

重要な制限：

libmviews は、共有ライブラリ形式で使用されていません。バージョン 4.0 以降、IBM ILOG Views で提供されるすべての共有ライブラリは libxviews を使って構築されており、libmviews との互換性がありません。

libmviews は、他の IBM ILOG Views ライブラリのスタティック・バージョンとのみ併用することができます。

新規入力ソースの追加

IBM® ILOG® Views を使用すると、アプリケーションでファイル記述子を新しい入力ソースとして追加できます。詳細については、メンバ関数 IlvEventLoop::addInput と IlvEventLoop::addOutput を参照してください。

ONC-RPC 統合

XtAppAddInput 関数にアクセスすると、BSD ソケットまたは ONC-RPC を IBM® ILOG® Views と併用できるようになります。

ONC-RPC の追加情報については、*Sun Network* マニュアルまたはご使用のシステムのマニュアルを参照してください。

libmviews を使用して IBM ILOG Views を Motif アプリケーションと統合する

IBM® ILOG® Views は既存の Motif アプリケーションと簡単に統合できるように設計されています。ライブラリ libmviews は IlvView を既存の Motif ウィジェットと接続する方法と、ユーザのアクションに対応するために必要なメカニズムを提供します。

以下のセクションには、次のトピックに関する情報が記載されています。

- *アプリケーションの初期化*
- *接続情報の取得*
- *既存ウィジェットの使用*

- *メイン・ループの実行*
- *Motif および IBM ILOG Views を使用するサンプル・プログラム*

アプリケーションの初期化

IBM ILOG Views コードを Motif ベースのアプリケーションと統合するときは、IBM ILOG Views セッションを次の 2 つの方法で作成できます。標準 IBM ILOG Views 初期化プロシージャまたは Motif アプリケーションの初期化ブロックのいずれかを使用して、以下のように `IlvDisplay` クラスの 2 番目のコンストラクタを呼び出します。

標準 IBM ILOG Views 初期化プロシージャ

```
IlvDisplay* display = new IlvDisplay("Program", "", argc, argv);
```

ここでは、IBM ILOG Views はディスプレイ・システムとの接続を確立します。

Motif アプリケーション初期化プロシージャ

```
Widget top = XtInitialize("", "Program", NULL, NULL, (Cardinal*)&argc, argv);
if (!top) {
    IlvFatalError("Couldn't open display");
    exit(1);
}
IlvDisplay* display = new IlvDisplay(XtDisplay(top), "X");
```

ここで、標準 Xt 関数呼び出しにより接続が初期化されます。`IlvDisplay` のコンストラクタにアプリケーション名を指定し、この文字列からディスプレイ・リソースを検索できるようにする必要があります。

接続情報の取得

`IlvDisplay` クラスのメンバ関数 `topShell` を呼び出すことにより、IBM ILOG Views の作成した一番上のシェルにアクセスできます。戻り値は `Widget` に変換しなければなりません。

Xt アプリケーション・コンテキストは関数 `IlvApplicationContext` で返されます。戻り値は `XtAppContext` に変換しなければなりません。

X Window アプリケーションへの接続についての情報をすべて取得するには、以下の関数を使用する必要があります。

```
XtAppContext appContext = (XtAppContext)IlvApplicationContext();
Widget topLevel = (Widget)display->topShell();
```

`IlvApplicationContext` 関数を使用する前に、以下をアプリケーション・コードに追加します。

```
extern XtAppContext IlvApplicationContext();
```

XtAppContext オブジェクトの使用方法は、Xt のマニュアルを参照してください。

既存ウィジェットの使用

IlvView クラスから継承されるクラスの大部分は、使用対象として既存のウィジェットを指定するコンストラクタを定義します。新たに作成する必要はありません。ウィジェットから IlvView オブジェクトを作成する方法は以下のとおりです。

```
IlvDisplay* display = ... // display initialization
// Create a DrawingArea widget
Widget drawingArea =
    XtVaCreateManagedWidget("ilvview",
                            xmDrawingAreaWidgetClass, parent,
                            XmNwidth, 100,
                            XmNheight, 100,
                            0);
// Realize this widget
XtRealizeWidget(drawingArea);
// Create a IlvView object from this widget.
IlvView* aView = new IlvView(display, drawingArea);
```

唯一の制約事項としては、IlvView のコンストラクタを呼び出す前に使用するウィジェットを *realize* しなければなりません (Xt 用語です。つまり、ウィジェットにウィンドウを作成しておく必要があります)。

メイン・ループの実行

libmviews では、関数 IlvMainLoop は、XtAppMainLoop とまったく同じように機能します。好きな方を使用できますが、IlvMainLoop は異なるプラットフォームへのコードの移植性を実現するために提供されています。

アプリケーションを他のプラットフォームに移植する予定がある場合、IBM ILOG Views コードを Motif コードからはっきりと分離することをお勧めします。

Motif および IBM ILOG Views を使用するサンプル・プログラム

以下のサンプル・プログラムは IBM ILOG Views コードを Motif アプリケーションに統合する方法の完全なサンプルです (samples/foundation/xlib/src/ilvmotif.cpp)。

```
// ------------------------------------------------------------- -*- C++ -*-
// IBM ILOG Views samples source file
// File: samples/foundation/xlib/src/ilvmotif.cpp
// --------------------------------------------------------------------------
// Using the grapher in a Motif widget
// --------------------------------------------------------------------------
```

```
#include <ilviews/contain/contain.h>
#include <ilviews/graphics/all.h>
#include <stdio.h>
#include <stdlib.h>

// --------------------------------------------------------------------------
// Integration Part with Motif
// --------------------------------------------------------------------------
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xlib.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>

// --------------------------------------------------------------------------
static void
Quit(Widget, XtPointer display, XtPointer)
{
    delete (IlvDisplay*)display;
    exit(0);
}

#define INPUT_MASK (unsigned long)(ButtonPressMask | ButtonReleaseMask |\
   KeyPressMask | KeyReleaseMask |\
   ButtonMotionMask | EnterWindowMask |\
   LeaveWindowMask | PointerMotionMask | \
   ExposureMask | StructureNotifyMask)

extern "C" void IlvDispatchEvent(XEvent* xevent);

static void
ManageInput(Widget, XtPointer, XEvent* xevent, Boolean*)
{
    IlvDispatchEvent(xevent);
}

// --------------------------------------------------------------------------
IlvDisplay*
IlvGetDisplay(Display* xdisplay)
{
    static IlvDisplay* ilv_display = 0;
    if (!ilv_display)
       ilv_display = new IlvDisplay(xdisplay, "IlvMotif");
    return ilv_display;
}

// --------------------------------------------------------------------------
IlvContainer*
CreateContainer(Widget widget)
{
    IlvContainer* c = new IlvContainer(IlvGetDisplay(XtDisplay(widget)),
       (IlvSystemView)XtWindow(widget));
    XtAddEventHandler(widget, INPUT_MASK, False,
       ManageInput, (XtPointer)c);
    return c;
}
```

```
// --------------------------------------------------------------------------
int
main(int argc, char* argv[])
{
    Widget toplevel = XtInitialize("", "IlvMotif", NULL, 0,
                                   &argc, argv);
    if (!toplevel)
        exit(1);
    Widget drawArea = XtVaCreateManagedWidget("ilvview",
      xmDrawingAreaWidgetClass,
      (Widget)toplevel,
      XtNwidth,  400,
      XtNheight, 400,
      (IlAny) 0);
    Widget pushb = XtVaCreateManagedWidget("Quit",
                   xmPushButtonWidgetClass,
                   drawArea,
                   (IlAny) 0);
    XtRealizeWidget(toplevel);
    IlvContainer* container = CreateContainer(drawArea);
     XtAddCallback(pushb, XmNactivateCallback, Quit, container->getDisplay());
    container->readFile("demo2d.ilv");
    XtMainLoop();
    return 0;
}
```

libxviews を使用して IBM ILOG Views を X アプリケーションと統合する

Xlib バージョンでは、Xlib ベースのアプリケーションに `Display` オブジェクトへのポインタ、描画先の `Window`、およびそのアプリケーションからイベントを受信する方法が指定されると、すぐにこのアプリケーションを統合することができます。

以下のセクションには、次のトピックに関する情報が記載されています。

- *統合ステップ*
- *完全なテンプレート*
- *Motif による完全な例*

統合ステップ

IBM ILOG Views を Xlib ベースのツールキットを使用するには、次を行う必要があります。

1. 既存の X Display を使用して `IlvDisplay` インスタンスを作成する。

   `IlvDisplay` コンストラクタを使用する。

   ```
   IlvDisplay::IlvDisplay(IlAny exitingXDisplay, const char* name);
   ```

例：

```
Display* xdisplay;
// ... initialize this Display*: xdisplay = XOpenDisplay(...);
IlvDisplay* ilvdisplay = new IlvDisplay((IlAny)xdisplay, "Views");
```

2. 既存の X Window を使用していずれかの IlvView インスタンスまたは IlvContainer インスタンスを作成する。

IlvView コンストラクタを使用する。

```
IlvView::IlvView(IlvDisplay* display,
                 IlvSystemView existingXWindow)
```

例：

```
IlvDisplay* display;
// initialize this 'display'
Window xWindow;
// initialize this X window
IlvView* view = new IlvView(display, (IlvSystemView)xWindow);
```

または

```
IlvContainer* container = new IlvContainer(display,
                                           (IlvSystemView)xWindow);
```

3. これらの IlvView ビューでイベントを管理する。

X イベント受信後に、次を呼び出す必要があります。

```
IlvEventLoop::getEventLoop()->dispatchEvent(&xevent);
```

完全なテンプレート

メイン・プロシージャは、以下のようになっています。

```
main()
{
    // Initialize your toolkit
    Display* xdisplay;
    xdisplay = // XOpenDisplay...;
    // Initialize an IlvDisplay
    IlvDisplay* ilvdisplay = new IlvDisplay((IlAny)xdisplay, "Views");
    // Create an X window:
    Window xwindow;
    xwindow = // ...;
    // Create an IlvContainer
    IlvContainer* container = new IlvContainer(display,(IlvSystemView)xwindow);
    container->addObject(new IlvLabel(...));
    // Now call the toolkit main event loop
}
```

Motif による完全な例

Motif は X ベース・ツールキットの例としてのみ選択されています。
IBM ILOG Views と Motif を統合するより良い方法は、これらが既に統合されている標準 IBM ILOG Views ライブラリ `libmviews` を使用することです。以下の例は、`libmviews` が使用できない場合の処理を示すものです (`samples/xlib/ilvmotif.cc`):

```
// --------------------------------------------------------------
// Integration of IlogViews, pure XLib version into a Motif
// application
// --------------------------------------------------------------
#include <ilviews/contain.h>
#include <ilviews/label.h>

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <X11/StringDefs.h>

// Define the default input mask for the window
#define INPUT_MASK (unsigned long)(ButtonPressMask    | \
                                   ButtonReleaseMask  | \
                                   KeyPressMask       | \
                                   KeyReleaseMask     | \
                                   ButtonMotionMask   | \
                                   EnterWindowMask    | \
                                   LeaveWindowMask    | \
                                   PointerMotionMask  | \
                                   ExposureMask       | \
                                   StructureNotifyMask)

// --------------------------------------------------------------
// This will be called by Xt when events of any of the
// types specified in INPUT_MASK occur.
// To do this, we call upon the XtAddEventHandler function call
// (see main()).
// --------------------------------------------------------------

static void
ManageInput(Widget, XtPointer view, XEvent* xevent, Boolean*)
{
    IlvEventLoop::getEventLoop()->dispatchEvent(xevent);
}
// --------------------------------------------------------------
void
main(int argc,char** argv)
{
    // Initialize X Window:
    Widget toplevel = XtInitialize("", "IlvXlib", NULL, NULL,
    // XtInitialize has a new specific signature in X11r5
#if defined(XlibSpecificationRelease) && (XlibSpecificationRelease >= 5)
                                   &argc,
#else
                                   (Cardinal*)&argc,
#endif
```

```
                                    argv);
    // If the top shell couldn?t be created, exit
    if (!toplevel)
        exit(1);

    // Create a Motif widget to draw to
    Widget drawArea = XtVaCreateManagedWidget("ilvview",
                                              xmDrawingAreaWidgetClass,
                                              (Widget)toplevel,
                                              XtNwidth,  400,
                                              XtNheight, 400,
                                              0);
    XtRealizeWidget(toplevel);

    // Create an IlvDisplay instance from the existing Display
    IlvDisplay* display = new IlvDisplay(XtDisplay(drawArea), "Views");

    // Create a container associated with the drawing area:
    IlvContainer* container =
          new IlvContainer(display, (IlvSystemView)XtWindow(drawArea));

    // Create a graphic object in the container
    container->addObject(new IlvLabel(display,
                                      IlvPoint(30, 30),
                                      "an IlvLabel instance"));
    // Let IlogViews know about the events
    XtAddEventHandler(drawArea, INPUT_MASK, IlFalse, ManageInput, NULL);

    // Wait for events to occur
    XtMainLoop();
}
```

ディレクトリ `samples/xlib/` には、さまざまなツールキットに適用される例が他にも含まれています。`ilvmotif.cpp` は Motif との統合の別の例です。同様に、`ilvolit.cpp` は OpenWindow との統合、`ilvxview.cpp` は、Xview との統合を示しています。

D

移植性の制約

このセクションでは、IBM® ILOG® Views 機能のうち、システム依存性によって移植性に制約のある機能のリストを提供します。以下のセクションでは、これらの機能を次のようにグループ化します。

- *サポートされない機能または制約のある機能*: 特定のシステムでは部分的にしかサポートされないか、まったくサポートされない機能。
- *メイン・イベント・ループ*: 使用されるシステムに応じて結果が変わる機能。特に、メイン・イベント・ループ。

サポートされない機能または制約のある機能

以下の表は、特定のシステムでは部分的にしかサポートされないか、まったくサポートされない IBM® ILOG® Views 機能のリストです。

表 D.1 *サポートされない機能または制約のある機能*

BitPlanes	Microsoft Windows ではサポートされません。
モーダル・モード	Windows NT ではサポートされません。

表 D.1 *サポートされない機能または制約のある機能 （続き）*

パターン・サイズ	Microsoft Windows パターンのサイズは制限されます。より大きいパターンも作成できますが、左上隅のみが最終パターンを決定します。
透明なパターン	Microsoft Windows では、Microsoft Windows の定義済みハッチ・ブラシでしか透明パターンは利用できません。つまり、ユーザ定義パターンおよび一部の IBM ILOG Views 定義済みパターンは透明にはできません。定義済み Microsoft Windows ハッチ・パターン上に構築された IBM ILOG Views パターンのリストは、`dialoglr`、`dialogrl`、`horiz`、`vert`、`cross` です。この制約は GDI+ の使用時には該当しません。これは、GDI+ ではすべての透明ブラシがサポートされるためです。
線の種類	Microsoft Win9x 上で線を描画する場合には、次の種類の線は有効ではありません。`dashdot`、`doubledot`、および `longdash`。これらはすべて `dash` スタイルになります。線の太さを 1 より大きい値に設定すると、線の種類が失われます。
カーソル・サイズ	Microsoft Windows では、カーソルのサイズは固定され、ドライバに依存します。サイズが不適切なビットマップが `IlvCursor` コンストラクタに指定されると、エラー・メッセージが送信されます。IBM ILOG Views は、カーソルが正常に作成されたかどうかをテストするために、`IlvCursor::isBad` メソッドを提供しています。
マウス・ボタン	特定の種類のマウスには 2 つのボタンしかありません。この場合、右のボタンにリンクされたイベントは `IlvMiddleButton` として設定されます。当初のインタラクタでは `IlvMiddleButton` を使用し、`IlvRightButton` はほとんど使用しなかったことから、このようになっています。`UseRightButton` アプリケーション・リソースを使用して、この振る舞いを変更できます。
Windows アイコン	Windows 95 および Windows NT4 では、各ビューに関連付けられているアイコンはアプリケーションのすべてのビューで同じです。
透明アンチエイリアシンク	Windows 上で GDI+ を使用する場合にのみ利用可能です。付録 B を参照してください。
多角形	Windows 95 では、多角形の角の上限は 16381 です。 ただし、特定の場合には、これより多い角を持つ多角形を作成できます (たとえば、凸状多角形)。

表 D.1 *サポートされない機能または制約のある機能 （続き）*

可変色	可変色は*擬似色*モデルのみで使用できます。擬似色モデルとは、画面の深度に基づいてピクセルを色に任意にマッピングするモデルで、カラー・マップ (UNIX システムの場合) またはパレット (PC の場合) に格納されます。擬似色は*直接色*モデルや*トゥルー・カラー*・モデルでは機能しません。
ウィンドウの不透明度	UNIX プラットフォームではサポートされません。
ズーム可能ラベル	UNIX では、`IlvZoomableLabel` オブジェクトはズーム、回転などが可能なビットマップです。Microsoft Windows ではビットマップを回転できないため、ビットマップは使用できません。そのため、`IlvZoomableLabel` オブジェクトは True Type フォントを使用して実装します。 この制限は、True Type フォントは本当の意味でのベクトル・フォントではなく、段階的に処理するために発生します。さらに、Microsoft Windows システムはフォントを実際のサイズでは表現できません (Microsoft Win32 Programmerfs Reference 第 1 巻、688 ページに『Windows の場合、フォントのサイズは、不正確な値になります』と記載されています。) **メモ：** *同じ制約は* <ILVHOME>/tools/vectfont *ディレクトリで提供されているベクトル・フォントにも適用されます。UNIX プラットフォームではベクトル・フォントを Hershey フォントにより実装し、Microsoft Windows プラットフォームでは True Type フォントまたは Hershey フォントにより実装します。*
XOR モードでの文字列	これは X Window で機能します。Microsoft Windows では XOR モードで文字列を描画することはできないため、 IBM ILOG Views はテキストと同じサイズの XOR ドット矩形を描画します。実際の文字列を表示するには、メソッド `IlvPort::drawString` または `IlvPort::drawIString` を使用して XOR ラベルを表示します。

メイン・イベント・ループ

グローバル関数 `IlvMainLoop` で定義されるメイン IBM® ILOG® Views イベント・ループは、X Window システムと Microsoft Windows システムでは同じ動きをしません。X Window サーバは非同期モードで機能しますが、Microsoft Windows は同期モードで機能します。また、タイマ管理は使用システムによって異なります。

- **同期モード対非同期モード：** X Window では、サーバに送信された要求は、関数が値を返しても即時には処理されません。メイン・ループが値を返した後でのみ処理されます。たとえば、ビューを表示する要求は、X Window サーバがマップ通知イベントを送り返し、このイベントが IBM ILOG Views API で処理されるまでは実行されません。
- **タイマ管理：** Microsoft Windows では、タイマ通知はイベント・ループで処理可能な Windows イベントです。X Window では、タイマ通知はイベントではないため、メイン・ループはこれがアクティブであるかどうかを認識しません。

E

エラー・メッセージ

このセクションでは、*IlvError クラス*に基づく IBM® ILOG® Views エラー・メッセージ生成について説明します。IBM ILOG Views がアプリケーション実行時に生成する可能性のあるメッセージのリストが含まれます。ここで、メッセージ・テキスト、そのエラー・メッセージが生成される可能性のある理由についての説明、および解決策を提示します。

リストはアルファベット順となっており、次の 2 つに分かれています。

- *致命的エラー*
- *警告*

デフォルトのエラー・ハンドラをオーバーロードしていない場合、致命的なエラー・メッセージの前には # が 2 つ付き、警告には - が 2 つ付きます。

> ***メモ:*** *これらは統合リストであるため、インストールされていない IBM ILOG Views パッケージ用のエラー・メッセージが含まれている場合もあります。*

IlvError クラス

IBM® ILOG® Views は、IlvError クラスに基づくエラー・メッセージ・メカニズムを提供します。すべての IBM ILOG Views アプリケーションに自動的にインストールされるデフォルトの IlvError インスタンスがあります。

このクラスは、メッセージ・パラメータを出力するだけで警告と致命的なエラーを定義します。より複雑な動作を実行するためにこのクラスのサブタイプを作成し、IBM ILOG Views で使用することができます。

2 つのグローバル関数は、現在のエラー処理を取得および設定します。

```
extern "C" IlvError* IlvGetErrorHandler();
extern "C" void      IlvSetErrorHandler(IlvError* errorHandler);
```

IBM ILOG Views で実際にエラー・ハンドラを呼び出すには、それぞれのエラー・メッセージを次のいずれかのグローバル関数を通じて送信する必要があります。

```
extern "C" void IlvWarning(const char* format, ...);
extern "C" void IlvFatalError(const char* format, ...);
```

このパラメータ format は、通常の C 関数 printf と同じフォーマットです。上の 2 つのグローバル関数のパラメータは、printf と同様にします。

致命的エラー

xxx was called with no arguments（引数なしで呼び出されました）

演算式の実行では、指定された定義済み関数を少なくとも 1 つのパラメータを指定して呼び出す必要があります。

Bad image description header（無効なイメージ記述ヘッダー）

XPM ファイルに認識できないビットマップ・ヘッダーがあります。

Bad image colors description（無効な色記述ヘッダ）

XPM ファイルに認識できない色記述子があります。

Cannot open xxx for writing（書き込みモードで xxx を開けません）

指定されたファイル名を書き込みモードで開けませんでした。UNIX バージョンではより詳細な情報が提供されます。

couldn't open dump file（ダンプ・ファイルを開けませんでした）

書き込み用にダンプ・ファイルを開けませんでした

couldn't open xxx（xxx を開けませんでした）

これは、ファイル名を読み込みまたは書き込み用に開けないことを示しています。

File xxx has a bad format（ファイル xxx のフォーマットは無効です）

指定されたファイル名が、IBM ILOG Views データ・ファイルではありません。

File IlogViews versions do not match（ファイル IlogViews のバージョンが一致しません）

現在実行しているライブラリよりも新しいバージョンで作成した IBM ILOG Views データ・ファイルを読み込もうとしています。

Format not implemented（フォーマットは実装されていません）

この BMP フォーマットは実装されていません。

IlvBitmap::read: couldn't open file xxx (IlvBitmap::read: ファイル xxx を開けません)

IlvDisplay::readBitmap: couldn't open file xxx (IlvDisplay::readBitmap: ファイル xxx を開けません)

指定されたファイル名を読み込み用に開けません。

IlvBitmap::read: bad format xxx (IlvBitmap::read: 不正なフォーマット xxx)

定義済みのフォーマット (XBM、XPM あるいは PBM P0 または P4) としてファイルを読み込めませんでした。

IlvBitmap::read: unknown color index xxx (IlvBitmap::read: 不明なカラー・インデックス xxx)

これは、不適切な色割り当てを示しています。

IlvBitmap::save: couldn't open file xxx (IlvBitmap::save: ファイル xxx を開けません)

書き込み用にファイルを開けませんでした。

IlvBitmap::saveAscii: Too many colors for ascii format
(IlvBitmap::saveAscii: ascii フォーマットに色が多すぎます)

＝ c（続き）

すべての必要な色と共に、このビットマップの読み込みに割り当てられた色が多すぎます。IBM ILOG Views は可能な限り正確にイメージを表現するために、最も近い既存の色を探します。

IlvContainer::readFile: couldn't open file xxx (check ILVPATH)
(IlvContainer::readFile: ファイル xxx を開けません (ILVPATH を確認してください))

指定されたファイルをロードできませんでした。`ILVPATH` 環境変数を確認してください。

IlvContainer::read: wrong format (IlvContainer::read: 不正なフォーマット)

ファイル・コンテンツをロードできませんでした。

IlvDisplay::readAsciiBitmap: wrong type xxx
(IlvDisplay::readAsciiBitmap: 不正なタイプ xxx)

これは IlvDisplay::readAsciiBitmap では認識できないタイプです。

IlvDisplay::readBitmap: unknown format xxx (IlvDisplay::readBitmap: 不明なフォーマット xxx)

指定されたファイルに、既知のビットマップ・フォーマットが含まれていません。

IlvEventPlayer::load: couldn't open file xxx (IlvEventPlayer::load: xxx を開けません)

読み込み用にイベント・ファイルを開けませんでした。

IlvEventPlayer::save: couldn't open file xxx (IlvEventPlayer::save: xxx を開けません)

書き込み用にイベント・ファイルを開けませんでした。

IlvGetViewInteractor: xxx not registered (IlvGetViewInteractor: xxx は未登録です)

指定されたビュー・インタラクタ・クラス名は登録されていません。マクロ IlvLoadViewInteractor に呼び出しを追加する必要があるかもしれません。

IlvGifFile() - xxx

GIF エラー・メッセージ。自明なので説明を省略します。

IlvInputFile::readNext: unknown class: xxx (IlvInputFile::readNext: 不明クラス : xxx)

指定されたクラスは、ご使用のバイナリでは不明です。いずれかのモジュール・ソース・ファイルに、このクラスが使用されるヘッダー・ファイルを含めてください。

IlvInputFile::readObject: bad format for: xxx (IlvInputFile::readObject: 不正なフォーマット : xxx)

有効な IBM ILOG Views ヘッダーではありません。

IlvInputFile::readObjectBlock: no object (IlvInputFile::readObjectBlock: オブジェクトがありません)

オブジェクト・ブロックを正常に読み込めませんでした。

IlvManagerViewInteractor:no such view (IlvManagerViewInteractor: そのようなビューはありません)

マネージャに接続されていないビューにビュー・インタラクタを設定しようとしました。

IlvReadAttribute: unknown attribute class xxx (IlvReadAttribute: 不明のアトリビュート・クラス xxx)

指定されたアトリビュート・クラス名が既知のクラスと一致しません。このアトリビュート・クラスの登録が必要である可能性があります。

IlvReadPBMBitmap: bad format (IlvReadPBMBitmap: 不正なフォーマット)

PBM ファイルのヘッダーが不適切です。

IlvReadPBMBitmap:unknown bitmap format (IlvReadPBMBitmap: 不明なビットマップ・フォーマット)

PBM フォーマットが正しくありません。既知のフォーマットは P1 ～ P6 です。

IlvPSDevice::drawTransparentBitmap: cannot use image mask (IlvPSDevice::drawTransparentBitmap: イメージ・マスクを使用できません)

実際にはカラー・イメージである透明ビットマップをダンプしようとしています。

IlvPSDevice::setCurrentPalette: file not opened (IlvPSDevice::setCurrentPalette: ファイルを開いていません)

ダンプ・ファイルを開いていませんが、ダンプ・プロセスが開始されました。

IlvVariable::setFormula: error in xxx (IlvVariable::setFormula: xxx にエラー)

式を読み込もうとしたときにエラーが発生しました。

IlvVariableContainer::connect: unknown attribute class xxx (IlvVariableContainer::connect: 不明なアトリビュート・クラス xxx)

IlvVariableManager::connect: unknown attribute class xxx (IlvVariableManager::connect: 不明なアトリビュート・クラス xxx)

このアトリビュートのタイプを登録する必要があります。このエラー・メッセージは現在使用されていません。

ILOG Views のデータ ファイルではありません

指定されたファイルは IBM ILOG Views データ・ファイルではありません。

Not a valid IlogViews message database file (無効な IlogViews メッセージ・データベース・ファイル)

`IlvMessageDatabase::read` could not convert this file contents into a database format (IlvMessageDatabase::read このファイル・コンテンツをデータベース・フォーマットに変換できませんでした)

Not a XPM format (XPM フォーマットではありません)

有効な XPM フォーマットではありません。IBM ILOG Views は XPM 2 フォーマットおよび C でコーディングされたフォーマットを読み込めます。

PolyPoints with zero points (ポリポイントの点がありません)

空のポリポイント・オブジェクトを作成しようとしています。

ReadAsciiColorBitmap: couldn't open file xxx (ReadAsciiColorBitmap: xxx を開けません)

読み込み用にファイルを開けませんでした。

ReadMonochromeX11Bitmap: couldn't read bitmap. Data=xxx (ReadMonochromeX11Bitmap: bitmap を読み込めませんでした。データは xxx です。)

Microsoft Windows バージョンのみで発生します。XBM ビットマップ・ファイルを読み込めませんでした。

Unknown bitmap format: xxx (不明なビットマップ・フォーマット : xxx)

IlvBitmap::read に、不明なビットマップ・フォーマットがあります。

Unknown event type: xxx (イベント・タイプが見つかりません : xxx)

イベント・ファイルの読み込み : 既知のイベント・タイプが見つかりませんでした。

Unknown requested type xxx in isSubtypeOf (isSubtypeOf に不明な要求タイプ xxx があります)

IlvGraphic::isSubtypeOf へのパラメータは既知のクラスではありません。

Unknown proposed type xxx in isSubtypeOf (isSubtypeOf に不明な提案タイプ xxx があります)

isSubtypeOf を呼び出したオブジェクトのクラス名が無効です。

警告

(<<IlvPattern*): Pattern has no name. Using 'noname' (パターンには名前がありません。noname を使用します。)

(<<IlvColorPattern*): Pattern has no name. Using 'noname' (パターンには名前がありません。noname を使用します。)

パターンまたは色のパターンの保存時に、パターンのビットマップに名前が設定されていません。ビットマップを保存する前に名前を設定する必要があります。設定しないと、正しくロードされません。

CreateBitmapCell: bitmap xxx not found using default (CreateBitmapCell: デフォルトを使用してビットマップ xxx が見つかりませんでした)

ビットマップ・セルの作成時に、ビットマップ名が内部で見つかりませんでした。指定されたビットマップ・ファイルを事前に読み込む必要があるかもしれません。

Found object xxx without IlvPalette (IlvPalette のないオブジェクト xxx が見つかりました)

オブジェクトの保存時に、パレットが置き換えられています。つまり、保存時にオブジェクトに修正を加えたことを意味します。

Icon bitmap has no name. Using 'noname' (アイコンのビットマップには名前がありません。noname を使用します。)

透明アイコンの保存時に、ビットマップに内部名が設定されていないため、適切にロードされません。

IlvButton::read: could not find bitmap xxx. Using default (IlvButton::read: ビットマップ xxx が見つかりません。デフォルトを使用します。)

ビットマップ・ボタンの作成時に、ビットマップ名が内部で見つかりませんでした。指定されたビットマップ・ファイルを事前に読み込む必要があるかもしれません。

IlvDisplay::copyStretchedBitmap: can't stretch from pixmap to bitmap (IlvDisplay::copyStretchedBitmap: ピクセル・マップからビットマップに伸張できません)
bitmap

カラー・ビットマップをモノクロの宛先デバイスに伸張しようとしています。

IlvGadgetContainer::read: couldn't allocate background color (IlvGadgetContainer::read: 背景色を割り当てられません)

データ・ファイルに保存されたコンテナ背景色を割り当てられません。一部の色をシステムに対して解放してください。

IlvGrapher::duplicate: object selection not removed (IlvGrapher::duplicate: オブジェクト選択が解除されていません)

オブジェクトの選択を解除するときにエラーが発生しました。

IlvGrapher::duplicate: object not found (IlvGrapher::duplicate: オブジェクトが見つかりません)

グラファに格納されていないオブジェクトを複製しようとしています。

IlvIcon::read: could not find bitmap xxx. Using default (IlvIcon::read: ビットマップ xxx が見つかりません。デフォルトを使用します。)

アイコンの作成時に、ビットマップ名が内部で見つかりませんでした。指定されたビットマップ・ファイルを事前に読み込む必要があるかもしれません。

IlvIcon::write: no name. Using 'noname'... (IlvIcon::write: 名前がありません。noname を使用します。)

アイコンのビットマップの保存時に、ビットマップに名前が設定されていません。ビットマップを保存する前に名前を設定する必要があります。設定しないと、正しくロードされません。

IlvManager::align: invalid value for align : xxx (IlvManager::align: 整列する値が無効です : xxx)

Invalid direction parameter for `IlvManager::align` (IlvManager::align の方向パラメータが無効です)

IlvManager::cleanObj: no properties (IlvManager::cleanObj: プロパティがありません)

マネージャに格納されていないオブジェクトをクリーンアップしようとしています。このオブジェクトを 2 回除去したか、またはマネージャが削除する前にオブジェクトが削除された可能性があります。

IlvManager::duplicate: object not found (IlvManager::duplicate: オブジェクトが見つかりません)

このマネージャに格納されていないオブジェクトを複製しようとしています。

IlvManager::reshapeObject: no properties (IlvManager::reshapeObject: プロパティがありません)

このマネージャに格納されていないオブジェクトの形状を変更しようとしています。

IlvManager::translateObject: no properties (IlvManager::translateObject: プロパティがありません)

このマネージャに格納されていないオブジェクトを変換しようとしています。

IlvManager::zoomView: invalid transformer (IlvManager::zoomView: 無効なトランスフォーマ)

要求されたズーム操作の結果、トランスフォーマが反転不可能になる可能性があります。

IlvReadPBMBitmap: bad values (IlvReadPBMBitmap: 不正な値)

イメージの記述が正しくありません。

IlvSetLanguage: locale not supported by Xlib (IlvSetLanguage: Xlib でサポートされないロケールです)

アプリケーションのリンク先の X11 ライブラリが、現在のロケールをサポートしません。このロケールをサポートする共有バージョンの libX11 へリンクし直す必要があるかもしれません。

IlvTransformer::inverse(IlvPoint&): bad transformer xxx
(IlvTransformer::inverse(IlvPoint&): 不正なトランスフォーマ xxx)

IlvTransformer::inverse(IlvFloatPoint&): bad transformer xxx
(IlvTransformer::inverse(IlvFloatPoint&): 不正なトランスフォーマ xxx)

IlvTransformer::inverse(IlvRect&): bad transformer xxx
(IlvTransformer::inverse(IlvRect&): 不正なトランスフォーマ xxx)

このトランスフォーマは反転不可能なので反転コールを実行できません。指定された値はトランスフォーマのアドレスで、デバッグ目的で提供されています。トランスフォーマの値を確認してください。

IlvTransparentIcon::read: could not find bitmap xxx. Using default
(IlvTransparentIcon::read: ビットマップ xxx が見つかりません。デフォルトを使用します。)

透明アイコンの読み込み時に、名前が既知のビットマップと一致しません。関連するビットマップを事前にロードする必要があるかもしれません。

IlvZoomableIcon::read: could not find bitmap xxx. Using default
(IlvZoomableIcon::read: ビットマップ xxx が見つかりません。デフォルトを使用します。)

ズーム可能なアイコンのビットマップ名が既知のビットマップと一致しません。

Object not removed xxx (オブジェクトは削除されていません xxx)

`IlvIndexedSet::removeObject` で、オブジェクトがこのインデックス付きセットに格納されていません。

Quadtree::add: xxx [bbox] Already in quadtree (Quadtree::add: xxx [bbox] 既にクワッドツリーにあります)

オブジェクトはマネージャに 2 回格納されます。オブジェクト・タイプとそのバウンディング・ボックスが提供されます。

Quadtree::remove: object xxx [bbox] not in quadtree (Quadtree::remove: オブジェクト xxx [bbox] はクワッドツリーにありません)

オブジェクトがマネージャから削除されますが、これはマネージャに格納されていませんでした。

ReadBitmap: bitmap xxx not found using default (ReadBitmap: ビットマップ xxx が見つかりません。デフォルトを使用します。)

ReadColorPattern: Pattern xxx not found! (ReadColorPattern: パターン xxx が見つかりません)

ReadPattern: Pattern xxx not found! Using 'solid' (ReadPattern: パターン xxx が見つかりません。solid を使用します。)

ビットマップを読み込む際に、これがパターンで使用されていた可能性があります。ビットマップ名が内部で見つかりませんでした。指定されたビットマップ・ファイルを事前に読み込む必要があるかもしれません。

ReadLineStyle: LineStyle xxx not found! Using 'solid' (ReadLineStyle: 線の種類 xxx が見つかりません。solid を使用します。)

線の種類を読み込む際に、指定された線の種類 ID が見つかりませんでした。

Too many colors. We'll keep xxx (色が多すぎます。xxx を維持します。)

色の割り当てリクエストが失敗しました。IBM ILOG Views はビットマップを完成するために、もっとも近い既存の色を探します。

WriteBitmap: Bitmap has no name using 'noname' (WriteBitmap: ビットマップには名前がありません。noname を使用します。)

ビットマップの保存時に、ビットマップに名前が設定されていません。ビットマップを保存する前に名前を設定する必要があります。設定しないと、正しくロードされません。

F

IBM ILOG Script 2.0 言語リファレンス

このリファレンスでは、IBM ILOG Script の構文を扱います。IBM ILOG Script は、Netscape Communications Corporation の JavaScript™ スクリプト言語をアイログが実装したものです。

言語構造

- *構文*
- *式*
- *ステートメント*

組み込まれた値と機能

- *数値*
- *文字列*
- *ブール型*
- *配列*
- *オブジェクト*
- *日付*
- *ヌル値*
- *未定義の値*

- *関数*
- *その他*

構文

本セクションは、次のような構成になっています。

- *IBM ILOG Script のプログラム構文*
- *複合ステートメント*
- *コメント*
- *識別子の構文*

IBM ILOG Script のプログラム構文

IBM ILOG Script プログラムは一連の*ステートメント*で構成されます。ステートメントには、条件ステートメント、ループ、関数定義、ローカル変数宣言などが含まれます。また、*式*は、値が無視され、その二次作用だけが考えられるステートメントが予測されるときにいつでも使うことができます。式には、代入、関数の呼び出し、プロパティ・アクセスなどが含まれます。

セミコロン (;) で区切られていると、一行に複数のステートメントまたは式が表示されることがあります。たとえば、次の 2 つのプログラムは等しくなります。

プログラム 1:

```
writeln("Hello, world")
x = x+1
if (x > 10) writeln("Too big")
```

プログラム 2:

```
writeln("Hello, World"); X = X+1; If (X > 10) Writeln("Too Big")
```

複合ステートメント

*複合ステートメント*とは、波括弧 ({}) で囲まれた一連のステートメントおよび式です。このステートメントを使って、1 つのステートメントが予測されるときにいつでも複数のタスクを実行することができます。たとえば、次の条件ステートメントでは、条件 a > b が true のときに波括弧で囲まれた 3 つのステートメントと式が実行されます。

```
if (a > b) {
var c = a
a = b
b = c
```

```
}
```

閉じ波括弧の前の最後のステートメントまたは式は、同じ行にある場合でもセミコロンを前に置く必要はありません。たとえば、次のプログラムは構文的に正しく、先述のプログラムと同じになります。

```
if (a > b) { var c = a; a = b; b = c }
```

コメント

IBM ILOG Script では、次の 2 種類のコメントをサポートしています。

- 一行コメント。一行コメントは // で始まり、その行の末尾で終わります。
  例 :

```
x = x+1 // Increment x,
y = y-1 // then decrement y.
```

- 複数行コメント。複数行コメントは /* で始まり、*/ で終わります。コメントは、複数行に及ぶこともあります。複数行コメントのネストはできません。
  例 :

```
/* The following statement
increments x. */
x = x+1
/* The following statement
decrements y. */
y = y /* A comment can be inserted here */ -1
```

識別子の構文

IBM ILOG Script では、識別子を使って変数や関数に名前を付けます。識別子は文字またはアンダーバーで始まり、その後に一連の文字、数字およびアンダーバーが続きます。

以下に識別子の例をいくつか示します。

```
car
x12
main_window
_foo
```

IBM ILOG Script では大文字と小文字を区別するため、大文字の A ～ Z と小文字の a ～ z は別の文字として認識されます。たとえば、「car」と「Car」は異なる識別子として扱われます。

以下の表の名前は予約されているため、識別子に使うことはできません。これらの名前の中には、IBM ILOG Script で使われるキーワードもあれば、将来使うために予約されているものもあります。

abstract	else	int	switch
boolean	extends	interface	synchronized
break	false	long	this
byte	final	native	throw
case	finally	new	throws
catch	float	null	transient
char	for	package	true
class	function	private	try
const	goto	protected	typeof
continue	if	public	var
default	implements	return	void
delete	import	short	while
do	in	static	with
double	instanceof	super	

式

本セクションは、次のような構成になっています。

- *IBM ILOG Script の式*
- *リテラル*
- *変数リファレンス*
- *プロパティ・アクセス*
- *代入演算子*
- *関数呼び出し*
- *特殊キーワード*
- *特殊演算子*
- *その他の演算子*

IBM ILOG Script の式

式は、リテラル、変数、特殊キーワードおよび演算子の組み合わせです。

> ***メモ:*** C++ プログラマの方へ : *IBM ILOG Script 式の構文は、C/C++ 構文と非常に似ています。*

演算子の先行によって、式を評価する際に適用される演算子の順番が決まります。演算子先行は、括弧を使って変更することができます。

以下の表では、IBM ILOG Script の演算子を先行が低い方から順番に列挙します。

表 F.1 *IBM ILOG Script 演算子先行*

カテゴリ	演算子
シーケンス	,
代入	= += -= *= /= %= <<= >>= >>>= &= ^= |=
条件	?:
論理和	||
論理積	&&
ビット論理和	|
ビット排他的論理和	^
ビット論理積	&
等価	== !=
関係	< <= > >=
ビット・シフト	<< >> >>>
加算、減算	+ -
乗算、除算	* / %
否定、インクリメント、typeof	! ~ - ++ -- typeof
呼び出し	( )
新規	new
プロパティ	. [ ]

リテラル

リテラルは、次を表現します。

- *数値*、例：`12 14.5 1.7e-100`

- *文字列*、例 : `"Ford" "Hello world\n"`
- *ブール型*、`true` または `false`。
- *ヌル値* : `null`。

数値と文字列のリテラル構文の詳細については、331 ページの *数値リテラル構文* および 337 ページの *文字列リテラル構文* を参照してください。

変数リファレンス

変数リファレンスの構文を、以下の表にまとめます。

表 F.2 *IBM ILOG Script の変数構文*

構文	効果
変数	*変数*の値を返します。変数の構文については、312 ページの *識別子の構文* を参照してください。 *変数*が存在しない場合はエラーが表示されます。これは、値が、有効で未定義の値を返す 未定義の値 の既存の変数への参照と同じではありません。 `with` ステートメントの本文で使った場合、変数リファレンスは現在のデフォルト値のプロパティとして検索されます。

プロパティ・アクセス

値プロパティにアクセスするための構文には、以下の 2 つがあります。

表 F.3 *IBM ILOG Script のプロパティ・アクセス構文*

構文	効果

値 . *名前*	*値*の*名前*プロパティの値、またはこのプロパティが定義されていない場合は 未定義の値 を返します。*名前* の構文については、312 ページの*識別子の構文* を参照してください。
	例：
	`str.length`
	
	`getCar().name`
	*名前*は有効な識別子でなければならないため、有効な識別子構文のないプロパティにアクセスするためにこの形式を使うことはできません。たとえば、配列 の数値プロパティに次のようにアクセスすることはできません。
	`myArray.10 // Illegal syntax`
	これらのプロパティでは、2 つ目の構文を使います。

値 [*名前*]	この場合の*名前*がプロパティ名を呼び出す評価された式であることを除いて、先述の構文と同じです。
	例：
	`str["length"] // Same as str.length`
	`getCar()[getPropertyName()]`
	`myArray[10]`
	`myArray[i+1]`

代入演算子

= 演算子を使って、変数またはプロパティに新しい値を代入することができます。

表 F.4 *IBM ILOG Script の代入演算子構文*

構文	効果
変数 = *式*	*式*の値を*変数*に代入します。*変数*が存在しない場合、グローバル変数として作成されます。 例： `x = y+1` 式全体で *expression* の値を返します。

値 . *名前* = *式*
値 [*名前*] = *式*
	*式*の値を特定のプロパティに代入します。
	*値*にそのようなプロパティがなく、プロパティが 配列 または オブジェクト の場合はそのプロパティが作成されます。そうでない場合はエラーになります。
	例：
	`car.name = "Ford"`
	`myArray[i] = myArray[i]+1`
	式全体で *式* の値を返します。

以下の簡略演算子も定義されています。

表 F.5 *簡略演算子*

構文	対象の演算子
++*X*	*X* = *X*+1
X++	++*X* と同じですが、新しい値ではなく *X* の初期値を返します。
--*X*	*X* = *X*-1
X--	--*X* と同じですが、新しい値ではなく *X* の初期値を返します。
X += *Y*	*X* = *X* + *Y*
X -= *Y*	*X* = *X* - *Y*
X *= *Y*	*X* = *X* * *Y*
X /= *Y*	*X* = *X* / *Y*
X %= *Y*	*X* = *X* % *Y*
X <<= *Y*	*X* = *X* << *Y*

表 F.5 簡略演算子 （続き）

構文	対象の演算子
X >>= *Y*	*X* = *X* >> *Y*
X >>>= *Y*	*X* = *X* >>> *Y*
X &= *Y*	*X* = *X* & *Y*
X ^= *Y*	*X* = *X* ^ *Y*
X |= *Y*	*X* = *X* | *Y*

関数呼び出し

関数を呼び出すための演算子の構文は以下のとおりです。

表 F.6 IBM ILOG Script の関数呼び出し構文

構文	効果

関数（*引数 1*, ..., *引数 n*）	特定の引数を使って*関数*を呼び出し、呼び出し結果を返します。
	例：
	`parseInt(field)`
	`writeln("Hello ", name)`
	`doAction()`
	`str.substring(start, start+length)`
	通常、*関数*は変数リファレンスまたはプロパティ・アクセスのいずれかですが、任意の式の場合もあります。式は 関数値 を呼び出すか、またはエラーになります。
	例：
	`// Calls the function in callbacks[i]`
	`callbacks[i](arg)`
	`// Error:a string is not a function`
	`"foo"()`

特殊キーワード

使用可能な特殊キーワードは以下のとおりです。

表 F.7 IBM ILOG Script の特殊キーワード

構文	効果
`this`	メソッドで参照される場合、現在の呼び出しオブジェクトを返します。コンストラクタで参照される場合は、初期化中のオブジェクトを返します。その他の場合はグローバル・オブジェクトを返します。例は、352 ページの*オブジェクト* を参照してください。

`arguments`	現在の関数の引数が含まれる 配列 を返します。関数の外で使われるとエラーになります。
	たとえば、次の関数はすべての引数の合計を返します。
	`function sum() {`
	`  var res = 0`
	`  for (var i=0; i<arguments.length; i++)`
	`  res = res+arguments[i]`
	`  return res`
	`}`
	コール `sum(1, 3, 5)` は 9 を返します。

特殊演算子

特殊演算子は以下のとおりです。

表 F.8 IBM ILOG Script の特殊演算子構文

構文	効果

`new` *コンストラクタ* (*引数1*, ..., *引数n*)	特定の*引数* を使って *コンストラクタ* を呼び出し、作成した値を返します。
	例 :
	`new Array()`
	`new MyCar("Ford", 1975)`
	通常、*コンストラクタ*は変数リファレンスですが、任意の式の場合もあります。
	例 :
	`new ctors[i](arg) // Invokes constructor ctors[i]`

表 F.8 *IBM ILOG Script の特殊演算子構文 （続き）*

<table>
<tr><th>構文</th><th colspan="2">効果</th></tr>
<tr><td>typeof value</td><td colspan="2">以下のように、値のタイプを表す文字列を返します。</td></tr>
<tr><td></td><td>値のタイプ</td><td>typeof value の結果</td></tr>
<tr><td></td><td>配列</td><td>" オブジェクト "</td></tr>
<tr><td></td><td>ブール型</td><td>" ブール型 "</td></tr>
<tr><td></td><td>日付</td><td>" 日付 "</td></tr>
<tr><td></td><td>関数</td><td>" 関数 "</td></tr>
<tr><td></td><td>ヌル</td><td>" オブジェクト "</td></tr>
<tr><td></td><td>数字</td><td>" 数値 "</td></tr>
<tr><td></td><td>オブジェクト</td><td>" オブジェクト "</td></tr>
<tr><td></td><td>文字列</td><td>" 文字列 "</td></tr>
<tr><td></td><td>未定義</td><td>" 未定義 "</td></tr>
<tr><td>delete 変数</td><td colspan="2">グローバル変数である変数を削除します。変数 の値は削除されませんが、グローバル環境から変数 が除去されます。
例 :
<pre> myVar = "Hello, world" // Create the
global variable myVar
  delete myVar
  writeln(myVar) // Signals an error
because myVar is undefined</pre>変数がローカル変数の場合、エラーになります。変数 が既知の変数でない場合は何も起きません。
式全体で true 値 を返します。
C/C++ プログラマの方へ : この演算子は、変数やプロパティではなくオブジェクトを削除するのに使われる C++ の演算子とはまったく違う意味になります。</td></tr>
</table>

表 F.8 *IBM ILOG Script の特殊演算子構文 （続き）*

構文	効果
delete *値* . *名前* delete *値* [*名前*]	オブジェクト *値*からプロパティである*名前*を削除します。*値*に*名前*プロパティが含まれない場合、この式は何も行いません。このプロパティが存在しない場合は、削除できずエラーになります。*値*がオブジェクトでない場合、エラーになります。 式全体で true 値 を返します。

式1 、*式2*	*式1* と *式2* を逐次評価し、*式2* の値を返します。*式1* の値は無視されます。
	一般的に、この演算子は for ループ内で、単一式が予測される複数の式を評価するのによく使われます。
	`for (var i=0, j=0; i<10; i++, j+=2) {`
	`  writeln(j, " is twice as big as ", i);`
	`}`

その他の演算子

その他の演算子については、それらが機能するデータ・タイプのセクションで説明します。その他の演算子は以下のとおりです。

表 F.9 *IBM ILOG Script のその他の演算子*

構文	効果
- *X* *X* + *Y* *X* - *Y* *X* * *Y* *X* / *Y* *X* % *Y*	算術演算子 これらの演算子では、一般的な算術演算を実行します。また、+ 演算子を使って文字列を連結することもできます。335 ページの *数値演算子* および 343 ページの *文字列演算子* を参照してください。

X == *Y*
X != *Y*
	等価演算子
	これらの演算子を使って、数値と文字列を比較することができます。335 ページの *数値演算子* および 343 ページの *文字列演算子* を参照してください。
	日付、配列、オブジェクト などの他のタイプの値については、*X* と *Y* が同一の場合のみ == 演算子が true を返します。
	例 :
	`new Array(10) == new Array(10) –> false`
	`var a = new Array(10); a == a –> true`

表 F.9 *IBM ILOG Script のその他の演算子 （続き）*

構文	効果
X > *Y* *X* >= *Y* *X* < *Y* *X* <= *Y*	関係演算子 これらの演算子を使って、数値と文字列を比較することができます。335 ページの *数値演算子* および 343 ページの *文字列演算子* を参照してください。
~ *X* *X* & *Y* *X* | *Y* *X* ^ *Y* *X* << *Y* *X* >> *Y* *X* >>> *Y*	ビット演算子 参照：335 ページの *数値演算子*
! *X* *X* || *Y* *X* && *Y* *condition* ?*X* :*Y*	論理演算子 参照：346 ページの *論理演算子*

ステートメント

本セクションは、次のような構成になっています。

- *条件ステートメント* (if)
- *ループ* (while、for、for..in、break、continue)
- *変数の宣言* (var)
- *関数定義* (function、return)
- *デフォルト値* (with)

条件ステートメント

条件 (if) ステートメントには、以下の構文があります。

表 F.10 *IBM ILOG Script の条件ステートメント構文*

構文	効果

if（*式*）
　ステートメント1
[else *ステートメント2*]
	*式*を評価します。true の場合、*ステートメント1* を実行し、そうでない場合は *ステートメント2* が提供されたときに *ステートメント2* を実行します。
	*式*が非ブール型値を返す場合、この値はブール型に変換されます。
	例：
	`if (a == b) writeln("They are equal")`
	`  else writeln("They are not equal")`
	
	`if (s.indexOf("a") < 0) {`
	`  write("The string ", s)`
	`  writeln(" doesn't contains the letter a")`
	`}`

ループ

ループ・ステートメントには以下の構文があります。

表 F.11 *IBM ILOG Script のループ・ステートメント構文*

構文	効果

while (*式*)
ステートメント
	*式*が true である限り、*ステートメント* を繰り返し実行します。*ステートメント*の各実行の前にテストが行われます。
	*式*が非ブール型値を返す場合、この値はブール型に変換されます。
	例 :
	`while (a*a < b) a = a+1`
	
	`while (s.length) {`
	`r = s.charAt(0)+r`
	`s = s.substring(1)`
	`}`

for （ [*初期化*];
[*条件*] ;
[*更新*])
ステートメント
*条件*と *更新*は式、*初期化*は式または以下の形式です。
var *変数* = *式*
	存在する場合に、*初期化* を一度評価します。値は無視されます。次の形式がある場合 :
	var *変数* = *式*
	*変数*がローカル変数として宣言され、変数ステートメント内で初期化されます。
	次に、*条件*が true である限り、*ステートメント*を繰り返し実行します。*条件*が省略されている場合、true とみなされて無限ループになります。*条件*が非ブール型値を返す場合、この値はブール型に変換されます。
	存在する場合、*ステートメント*の後、*条件*の前にループを通過するたびに*更新*が評価されます。値は無視されます。
	例 :
	`for (var i=0; i < a.length; i++) {`
	`sum = sum+a[i]`
	`prod = prod*a[i]`
	`}`

表 F.11 *IBM ILOG Script のループ・ステートメント構文 （続き）*

構文	効果

`for ([ var ]` *変数* `in` *式*`)` *ステートメント*	*式*の値のプロパティで反復します。各プロパティについて、*変数*は、このプロパティを表す文字列に設定され、*statement* が一度実行されます。
	キーワードがある場合、*変数*は、var ステートメントのようにローカル変数として宣言されます。
	たとえば、次の関数では任意の値を取り、すべてのプロパティとそれらの値を表示します。
	`function printProperties(v) {`
	`  for (var p in v)`
	`  writeln(p, " -> ", v[p])`
	`}`
	for..in ステートメントによって列挙されるプロパティには、値が関数値の通常のプロパティであるメソッド・プロパティが含まれます。たとえば、呼び出し `printProperties("foo")` では以下を表示します。
	`length -> 3`
	`toString -> [primitive method toString]`
	`substring -> [primitive method substring]`
	`charAt -> [primitive method charAt]`
	`  etc.`
	配列 の数値プロパティだけは、*for..in* ループによって列挙されません。

`break`	現在の *while*、*for* または *for..in* ループを終了し、そのループの直後にステートメントの実行を続けます。このステートメントはループの外では使用できません。
	例 :
	`while (i < a.length) {`
	`  if (a[i] == "foo") {`
	`  foundFoo = true`
	`  break`
	`  }`
	`  i = i+1`
	`}`
	`// Execution continues here`

`continue`	現在の *while*、*for* または *for..in* ループを停止し、次の反復を使ってループの実行を続けます。このステートメントはループの外では使用できません。
	例 :
	`for (var i=0; i < a.length; i++) {`
	`  if (a[i] < 0) continue`
	`  writeln("A positive number:", a[i])`
	`}`

変数の宣言

変数の宣言には以下の構文があります。

***表 F.12** IBM ILOG Script の変数宣言構文*

構文	効果

`var` *decl1*, ..., *decln*
各 *decli* には次の形式があります。
変数[= *式*]
	各*変数*をローカル変数として宣言します。*式*が与えられている場合、その式は評価され、値が変数に初期値として代入されます。そうでない場合、変数は未定義値に設定されます。
	例：
	`var x`
	`var name = "Joe"`
	`var average = (a+b)/2, sum, message="Hello"`

***表 F.12** IBM ILOG Script の変数宣言構文 （続き）*

構文	効果

関数定義内の var	関数定義内で var を使う場合、宣言された変数は関数に対してローカルで、それらの変数は同じ名前を持つグローバル変数を隠します。実際に、それらの変数は関数引数と同じステータスを持ちます。
	たとえば、次のプログラムにおいて、変数 sum および res は、引数 a および b と同様に、average 関数に対してローカルになります。したがって、average が呼び出されると、同じ名前を持つグローバル変数がある場合、そのグローバル変数は関数が終了するまで一時的に隠されます。
	`function average(a, b) {`
	`  var sum = a+b`
	`  var res = sum/2`
	`  return res`
	`}`
	関数本体の*任意*の場所で var を使って宣言された変数には、関数*全体*の範囲があります。これは、C または C++ のローカル変数範囲とは異なります。
	たとえば、以下の関数で、if ステートメントの最初の分岐で宣言された変数 *res* は、他の分岐および return ステートメントで使われます。
	`function max(x, y) {`
	`  if (x > y) {`
	`  var res = x`
	`  } else {`
	`  res = y`
	`  }`
	`  return res`
	`}`

表 F.12 IBM ILOG Script の変数宣言構文 （続き）

構文	効果

関数定義外の var	var が関数定義の外、つまり関数定義と同じレベルで使われる場合、宣言された変数は現在の*プログラム・ユニット*に対してローカルになります。プログラム・ユニットとは、全体とみなされるステートメントのグループです。ただし、プログラム・ユニットの正確な定義は、IBM ILOG Script が組み込まれているアプリケーションによって異なります。通常、アプリケーションによってロードされたスクリプト・ファイルは、プログラム・ユニットとして扱われます。この場合、ファイルのトップ・レベルで var を使って宣言された変数は、このファイルに対してローカルになり、それらの変数は同じ名前のグローバル変数をすべて隠します。たとえば、ファイルに以下のプログラムが含まれる場合を考えてみましょう。
	`var count = 0`
	
	`function NextNumber() {`
	`  count = count+1`
	`  return count`
	`}`
	このファイルがロードされると、関数 NextNumber はアプリケーション全体に対して表示可能になります。また、count はロードしたプログラム・ユニットに対してローカルになり、内部のみで表示可能になります。
	同じ範囲で同じローカル変数を宣言するとエラーになります。たとえば、以下のプログラムは、res が 2 回宣言されているため正しくありません。
	`function max(x, y) {`
	`  if (x > y) {`
	`  var res = x`
	`  } else {`
	`  var res = y // Error`
	`  }`
	`  return res`
	`}`

関数定義

関数定義には、以下の構文があります。

***表 F.13** IBM ILOG Script の関数定義*

構文	効果

[static]
function *名前* (*v1*,...,*vn*)
{ *ステートメント* }
	特定のパラメータおよび本体を持つ関数の*名前*を定義します。関数定義はトップ・レベルのみで実行されるため、関数定義のネストはできません。
	関数が呼び出されると、変数 *v1*、...、*vn* は対応する引数値に設定されます。次に、*ステートメント*が実行されます。return ステートメントに達すると、関数は特定の値を返します。または、*ステートメント*が実行された後に、関数は 未定義 値を返します。
	実際の引数の数とパラメータの数は一致する必要はありません。パラメータよりも引数の方が少ない場合、残りのパラメータは未定義値に設定されます。パラメータよりも引数の方が多い場合、余った引数は無視されます。
	パラメータのメカニズムとは無関係に、引数キーワードを使って関数引数を読み出すことができます。
	関数の*名前*の定義は、特定の関数値を変数の*名前*に割り当てるのと操作上同じです。したがって、関数の定義は以下に等しくなります。
	`var` *名前* `=` *特定の関数値*
	関数値は、変数から取得したり、他のタイプの値のように操作することができます。たとえば、以下のプログラムでは関数 add を定義し、その値を変数 sum に代入します。これにより、add と sum は同じ関数の類義語になります。
	`function add(a, b) {`
	`  return a+b`
	`}`
	
	`sum = add`
	static キーワードを使わない場合、定義した関数はグローバルで、アプリケーション全体からアクセスすることができます。static キーワードを使う場合、var キーワードを使って *名前*を宣言したときのように、関数は現在のプログラム・ユニットに対してローカルになります。
	`var` *名前* `=` *特定の関数値*

return [*式*]	現在の関数から*式*の値を返します。*式*が省略されている場合は、未定義 値を返します。return ステートメントは、関数本体のみで使用できます。

デフォルト値

デフォルト値は、以下の構文で使用します。

表 F.14 *IBM ILOG Script のデフォルト値*

構文	効果

`with` (*式*)
  ステートメント
	*式*を評価してから、デフォルト値として一時的に設定した *式*の値を使って *ステートメント*を実行します。
	*ステートメント*の識別子の *名前*の参照を評価する際、この識別子では最初にデフォルト値のプロパティを検索します。デフォルト値にそのようなプロパティがない場合、*名前*は通常の変数として扱われます。
	たとえば、以下のプログラムでは、識別子 `length` が文字列 `"abc"` の `length` プロパティとしてみなされるため、"The length is 3 (長さは 3 です)" と表示されます。
	`with ("abc") {`
	`  writeln("The length is ", length)`
	`}`
	`With` ステートメントはネストできます。この場合、識別子の参照は、連続するデフォルト値で一番内側から一番外側の `with` ステートメントに向かって検索されます。

数値

本セクションは、次のような構成になっています。

- *数値リテラル構文*
- *特殊数値*
- *数値への自動変換*
- *数値メソッド*
- *数値関数*
- *数値定数*
- *数値演算子*

数値リテラル構文

数値は、10 進数 (底 10)、16 進数 (底 16)、または 8 進数 (底 8) で表すことができます。

> ***メモ:*** C++ プログラマの方へ : *数値は、C および C++ の整数および倍精度と同じ構文を持ちます。それらの数値は、64 ビット倍精度浮動小数点数として表されます。*

小数は、一連の数字の後に任意の端数、任意の指数と続きます。端数は、小数点 (.) の後に一連の数字が続きます。指数は、`e` または `E` の後に、任意の `+` または `-` 符号と一連の数字が続きます。小数には 1 桁以上が必要です。

以下に、小数リテラルの例をいくつか示します。

```
15
3.14
4e100
.25
5.25e-10
```

16 進数は、`0x` または `0X` プレフィックスの後に数字 (`0-9`) および文字 `a-f` または `A-F` を含む一連の 16 進数字が続きます。例 :

```
0x3ff
0x0
```

8 進数は、`0` の後に、数字 `0-7` を含む一連の 8 進数字が続きます。例 :

```
0123
0777
```

特殊数値

特殊数値には、*NaN* (非数)、*Infinity* (正の無限) および *-Infinity* (負の無限) の 3 つがあります。

特殊数値 NaN は、数値操作におけるエラーを表すのに使います。たとえば、負数に適用された平方根関数 `Math.sqrt` は NaN を返します。数値リテラルとしての NaN の表現はありませんが、グローバル変数 `NaN` にその値が含まれます。

NaN 値は伝染するため、NaN を伴う数値操作は常に NaN を返します。NaN を伴う比較操作は、NaN == NaN 比較の場合でも常に false を返します。

例 :

```
Math.sqrt(-1) –> NaN
```

```
Math.sqrt(NaN) –> NaN
NaN + 3 –> NaN
NaN == NaN –> false
NaN <= 3 –> false
NaN >= 3 –> false
```

特殊数字 Infinity および 蜂 nfinity は、算術演算で無限値とオーバーフローを示すのに使われます。グローバル変数 `Infinity` には正の無限が含まれます。負の無限は、負の演算子 (*-Infinity*) を使って計算することができます。

例：

```
1/0 –> Infinity
-1/0 –> -Infinity
1/Infinity –> 0
Infinity == Infinity –> true
```

数値への自動変換

引数の 1 つに数値を予測する関数またはメソッドに非数値が渡された場合、その関数またはメソッドでは、次のルールを使ってその値を数値に変換します。

- 文字列 は数値リテラルとして解析されます。文字列が有効な数値リテラルを表していないときは、変換によって NaN が返されます。
- ブール型 *true* は、数値 1 を返します。
- ブール型 *false* は、数値 0 を返します。
- ヌル 値は、数値 0 を提供します。
- 日付 は、1970 年 1 月 1 日 00：00：00 UTC (世界協定時刻) 以降の対応する数値を、ミリ秒単位で返します。

たとえば、`Math.sqrt` 関数に文字列が渡された場合、この文字列はそれを表す数値に変換されます。

```
Math.sqrt("25")  –> 5
```

同様に、数値オペランドを取る演算子は、すべての非数オペランドを数値に変換しようとします。

```
"3" * "4" –> 12
```

+ などの文字列と数値の両方を取ることができる演算子では、数値の変換よりも文字列への変換の方が優先されます (338 ページの *文字列への自動変換* 参照)。したがって、1 つ以上のオペランドが文字列の場合、他のオペランドは文字列に変換さ

れます。また、オペランドに文字列がない場合、そのオペランドは両方数値に変換されます。例:

```
"3" + true –> "3true"
```

```
3 + true  –> 4
```

== や >= などの比較演算子では、文字列への変換よりも数値への変換の方が優先されます。したがって、1 つ以上のオペランドが数値の場合、他のオペランドは数値に変換されます。両方のオペランドが文字列の場合は、文字列に対して比較が行われます。例:

```
"10" > "2" –> false
```

```
"10" > 2 –> true
```

数値メソッド

次に示すのは、唯一の数値メソッドです。

***表 F.15** IBM ILOG Script の数値メソッド*

構文	効果
number.toString( )	数値をリテラルとして表す文字列を返します。 例: (14.3e2).toString() –> "1430"

数値関数

以下の数値関数が定義されています。

> ***メモ:*** C/C++ プログラマの方へ: *これらの関数のほとんどは、標準ライブラリ関数をまとめたものです。*

***表 F.16** IBM ILOG Script の数値関数*

構文	効果
Math.abs(*x*)	*x* の絶対値を返します。
Math.max(*x*, y) Math.min(*x*, y)	Math.max(*x*, *y*) は *x* および *y* の大きい方の値を返し、Math.min(*x*, *y*) は両者のうちで小さい方を返します。
Math.random( )	0、包含的、および 1、排他的の中から擬似ランダム数値を返します。

表 F.16 IBM ILOG Script の数値関数 （続き）

構文	効果

`Math.ceil(x)`
`Math.floor(x)`
`Math.round(x)`
	`Math.ceil(x)` は、*x* に等しいかまたはそれよりも大きい最小整数値を返します。`Math.floor(x)` は、*x* に等しいかまたはそれよりも小さい最大整数値を返します。
	`Math.round(x)` は、*x* の値に最も近い整数値を返します。

`Math.sqrt(x)`	*x* の平方根を返します。

`Math.sin(x)`
`Math.cos(x)`
`Math.tan(x)`
`Math.asin(x)`
`Math.acos(x)`
`Math.atan(x)`
`Math.atan2(y, x)`
	`Math.sin(x)`、`Math.cos(x)` および `Math.tan(x)` は、ラジアン引数の三角関数を返します。
	`Math.asin(x)` は、範囲 -PI/2 ～ PI/2 の *x* の逆正弦関数を返します。
	`Math.acos(x)` は、範囲 0 ～ PI の *x* の逆余弦関数を返します。
	`Math.atan(x)` は、範囲 -PI/2 ～ PI/2 の *x* の逆正接関数を返します。
	`Math.atan2(y, x)` は、*a* を範囲 -PI ～ PI の逆正接関数 *y*/*x* として計算して、直角座標 (*x*, *y*) を極 (*r*, *a*) に変換します。

`Math.exp(x)`
`Math.log(x)`
`Math.pow(x, y)`
	`Math.exp(x)` は、指数関数 ex を計算します。
	`Math.log(x)` は、*ex* の自然対数を計算します。
	`Math.pow(x, y)` は、*y* 乗した *x* を計算します。

数値定数

以下の数値定数が定義されています。

表 F.17 IBM ILOG Script の数値定数

構文	値
`NaN`	NaN 値を含みます。
`Infinity`	Infinity 値を含みます。
`Number.NaN`	NaN と同じ。
`Number.MAX_VALUE`	表現可能な最大数は、約 1.79E+308 です。
`Number.MIN_VALUE`	表現可能な最小正数は、約 2.22E-308 です。
`Math.E`	オイラーの定数および自然対数の底は、約 2.718 です。

表 F.17 IBM ILOG Script の数値定数 （続き）

構文	値
`Math.LN10`	10 の自然対数は、約 2.302 です。
`Math.LN2`	2 の自然対数は、約 0.693 です。
`Math.LOG2E`	e の底 2 の対数は、約 1.442 です。
`Math.LOG10E`	e の底 10 の対数は、約 0.434 です。
`Math.PI`	円の直径に対する円周の比率 (円周率) は、約 3.142 です。
`Math.SQRT1_2`	2 分の 1 の平方根は、約 0.707 です。
`Math.SQRT2`	2 の平方根は、約 1.414 です。

数値演算子

以下の数値演算子を使用できます。

> ***メモ :*** C/C++ プログラマの方へ : *これらの演算子は、C および C++ のものと同じです。*

表 F.18 IBM ILOG Script の数値演算子

構文	効果
x + *y* *x* - *y* *x* * *y* *x* / *y*	通常の算術演算 例 : `3 + 4.2` –> 7.2 `100 - 120` –> -20 `4 * 7.1` –> 28.4 `6 / 5` –> 1.2
- *x*	否定 例 : `- 142` –> -142
x % *y*	*x* ÷ *y* の浮動小数点の余りを返します。 例 : `12 % 5` –> 2 `12.5 % 5` –> 2.5

表 F.18 *IBM ILOG Script の数値演算子 （続き）*

構文	効果
x == *y* *x* != *y*	演算子 == は、*x* および *y* が等しい場合に *true* を、そうでない場合に *false* を返します。演算子 != は、== の逆です。 例： `12 == 12` –> *true* `12 == 12.1` –> *false* `12 != 12.1` –> *true*
x < *y* *x* <= *y* *x* > *y* *x* >= *y*	演算子 < は、*x* が *y* よりも小さい場合に *true* を、そうでない場合に *false* を返します。演算子 <= は、*x* が *y* よりも小さいか等しい場合に *true* を、そうでない場合に *false* を返します。 例： `-1 < 0` –> *true* `1 < 1` –> *false* `1 <= 1` –> *true*
x & *y* *x* | *y* *x* ^ *y*	ビット演算 AND、OR および XOR。*x* および *y* は、-2**32+1 ～ 2**32-1 (-2147483647 ～ 2147483647) の整数でなければなりません。 例： `14 & 9` –> 8 (1110 & 1001 –> 1000) `14 | 9` –> 15 (1110 | 1001 –> 1111) `14 ^ 9` –> 7 (1110 ^ 1001 –> 111)
~ *x*	ビット NOT*x* は、-2**32+1 to 2**32-1 (-2147483647 ～ 2147483647) の整数でなければなりません。 例： `~ 14` –> 1 (~ 1110 –> 0001)
x << *y* *x* >> *y* *x* >>> *y*	2 進数シフト演算。*x* および *y* は、-2**32+1 ～ 2**32-1 (-2147483647 ～ 2147483647) の整数でなければなりません。演算子 << は左にシフトし、>> は右にシフトします (符号ビットは残ります)。また、>>> は、左からゼロにシフトしながら右にシフトします。 例： `9 << 2` –> 36 (1001 << 2 –> 100100) `9 >> 2` –> 2 (1001 >> 2 –> 10) `-9 >> 2` –> -2 (1..11001 >> 2 –> 1..11110) `-9 >>> 2` –> 1073741821 (1..11001 >>> 2 –> 01..11110)

文字列

- *文字列リテラル構文*
- *文字列への自動変換*
- *文字列プロパティ*
- *文字列メソッド*
- *文字列関数*
- *文字列演算子*

文字列リテラル構文

文字列リテラルとは、二重引用符 (") または一重引用符 (') で囲まれた 0 個以上の文字です。

> ***メモ:*** C++ プログラマの方へ: *一重引用符の使用を除いて、文字列リテラルは C および C++ と同じ構文を持ちます。*

以下は文字列リテラルの例です。

```
"My name is Hal"
'My name is Hal'
'"Hi there", he said'
"3.14"
"Hello, world\n"
```

これらの例の 1 つ目と 2 つ目の文字列は同じです。

円記号文字 (¥) を使って、文字列リテラルで直接表現できない文字を表すエスケープ・シーケンスを取り入れることができます。文字列で使用可能なエスケープ・シーケンスは以下のとおりです。

***表 F.19** IBM ILOG Script のエスケープ・シーケンス*

エスケープ・シーケンス	意味
¥n	改行
¥t	タブ
¥¥	円記号文字 (¥)

表 F.19 *IBM ILOG Script のエスケープ・シーケンス （続き）*

エスケープ・シーケンス	意味
¥"	二重引用符 (")
¥'	一重引用符 (')
¥b	バックスペース
¥f	フォーム・フィード
¥r	キャリッジ・リターン
¥x*hh*	ASCII コードが *hh* の文字。*hh* は、2 つの 16 進数字のシーケンスです。
¥*ooo*	ASCII コードが *ooo* の文字。*ooo* は、1 つ、2 つまたは 3 つの 8 進数字のシーケンスです。

以下はエスケープ・シーケンスを使った文字列リテラルの例です。

表 F.20 *IBM ILOG Script のエスケープ・シーケンス例*

文字列リテラル	意味
`"Read ¥"The Black Bean¥""`	"The Black Bean" をお読みください (Read "The Black Bean")
`'¥'Hello¥', he said'`	彼は「こんにちは」と言った ('Hello', he said)
`"c:¥¥temp"`	c:¥temp
`"First line¥nSecond line¥nThird line"`	1 行目 2 行目 3 行目
`"¥xA9 1995-1997"`	© 1995-1997

文字列が 数字に変換 の場合、その文字列を数値リテラルとして解析しようと試みます。文字列が有効な数値リテラルを表していないときは、変換によって NaN が返されます。

文字列への自動変換

引数の 1 つに文字列を予測する関数またはメソッドに非文字列値が渡された場合、その値は文字列に自動的に変換されます。たとえば、文字列メソッド indexOf に最初の引数として数値が渡された場合、この数値は文字列表現のように扱われます。

```
"The 10 commandments".indexOf(10) –> 4
```

同様に、文字列オペランドを取る演算子は、すべての非文字列オペランドを文字列に自動的に変換します。

```
"The " + 10 + " commandments" –> "The 10 commandments"
```

文字列への変換では、特定の値の `toString` メソッドを使用します。組み込まれたすべての値には `toString` メソッドがあります。

文字列プロパティ

文字列には以下のプロパティがあります。

***表 F.21** IBM ILOG Script の文字列プロパティ*

構文	値
文字列 `.length`	*文字列*の文字数。これは読み取り専用のプロパティです。 例 `"abc".length` –> 3 `"".length` –> 0

文字列メソッド

文字列の文字は、左から右にインデックス化されます。文字列 *文字列* の最初の文字のインデックスは 0 で、最後の文字のインデックスは *文字列* .length-1 です。

文字列には以下のメソッドがあります。

***表 F.22** IBM ILOG Script の文字列メソッド*

構文	効果
文字列 .substring (*開始* [, *end*])	インデックス *開始* で始まり、インデックス *終了*-1 で終わる *文字列*のサブ文字列を返します。*終了*が省略されている場合、*文字列* の末尾を返します。 例： "0123456".substring(0, 3) –> "012" "0123456".substring(2, 4) –> "23" "0123456".substring(2) –> "23456"
文字列 .charAt (*インデックス*)	*文字列*の特定のインデックスの文字が含まれる 1 文字文字列を返します。*インデックス*が範囲外の場合は、空の文字列を返します。 例： "abcdef".charAt(0) –> "a" "abcdef".charAt(3) –> "d" "abcdef".charAt(100) –> ""
文字列 .charCodeAt (*インデックス*)	*文字列*の特定のインデックスの文字の ASCII コードを返します。*インデックス*が範囲外の場合は、NaN を返します。 例： "abcdef".charCodeAt(0) –> 97 "abcdef".charCodeAt(3) –> 100 "abcdef".charCodeAt(100) –> NaN
文字列 .indexOf (*サブ文字列* [, *インデックス*])	最初の *サブ文字列*の *文字列*のインデックスを返します。*文字列*は、*インデックス*を起点に検索されます。*インデックス*が省略されている場合、*文字列* は最初から検索されます。このメソッドでは、*サブ文字列*が見つからない場合に ñ1 を返します。 例： "abcdabcd".indexOf("bc") –> 1 "abcdabcd".indexOf("bc", 1) –> 1 "abcdabcd".indexOf("bc", 2) –> 5 "abcdabcd".indexOf("bc", 10) –> -1 "abcdabcd".indexOf("foo") –> -1 "abcdabcd".indexOf("BC") –> -1

表 F.22 IBM ILOG Script の文字列メソッド（続き）

構文	効果
文字列.lastIndexOf (*サブ文字列* [, *インデックス*])	最後の *サブ文字列* の *文字列* のインデックスを返します。*文字列* は、*インデックス* から後方に検索されます。*インデックス* が省略されている場合、*文字列* は末尾から検索されます。このメソッドでは、*サブ文字列* が見つからない場合に ñ1 を返します。 例： "abcdabcd".lastIndexOf("bc") –> 5 "abcdabcd".lastIndexOf("bc", 5) –> 5 "abcdabcd".lastIndexOf("bc", 4) –> 1 "abcdabcd".lastIndexOf("bc", 0) –> -1 "abcdabcd".lastIndexOf("foo") –> -1 "abcdabcd".lastIndexOf("BC") –> -1
文字列 .toLowerCase( )	小文字に変換された *文字列* を返します。 例： "Hello, World".toLowerCase() –> "hello, world"
文字列 .toUpperCase( )	大文字に変換された *文字列* を返します。 例： "Hello, World".toUpperCase() –> "HELLO, WORLD"
文字列 .split (*区切り文字*)	*区切り文字* で区切られた *文字列* のサブ文字列を含む文字列の配列を返します。配列メソッドの join (結合) も参照してください。 例： "first name,last name,age".split(",") –> an array *a* such that a.length is 3, a[0] is "first name", a[1] is "last name", and a[2] is "age". *文字列* に *区切り文字* が含まれない場合、*文字列* 全体を含む 1 つの要素を持つ配列が返されます。 例： "hello".split(",") –> an array *a* such that a.length is 1 and a[0] is "hello",
文字列 .toString( )	文字列自体を返します。

文字列関数

以下の関数は文字列で機能します。

表 F.23 IBM ILOG Script の文字列関数

構文	効果
`String.fromCharCode(` *コード*`)`	特定の ASCII コードを持つ文字が含まれる 1 つの文字列を返します。 例： `String.fromCharCode(65)` –> "A" `String.fromCharCode(0xA9)` –> "©"
`parseInt(` *文字列* `[ , ` *基数* `] )`	*文字列*を特定の基数に書き込まれた整数として解析し、その値を返します。文字列が有効な整数でない場合は、NaN が返されます。 先頭の空白文字は無視されます。`parseInt` が特定の基数の数字でない文字を見つけると、その文字とその後の文字をすべて無視して、そこまでに解析された整数値を返します。 *基数*が省略されている場合、底 16 で解析されるときに*文字列*が `0x` または `0X` で始まらない限り、または底 8 で解析されるときは `0` で始まらない限り、10 と見なされます。 例： `parseInt("123")` –> -123 `parseInt("-123")` –> -123 `parseInt("123.45")` –> 123 `parseInt("1001010010110", 2)` –> 4758 `parseInt("a9", 16)` –> 169 `parseInt("0xa9")` –> 169 `parseInt("010")` –> 8 `parseInt("123 poodles")` –> 123 `parseInt("a lot of poodles")` –> NaN
`parseFloat(` *文字列*`)`	*文字列*を浮動小数として解析し、その値を返します。文字列が有効な数値でない場合は、NaN が返されます。 先頭の空白文字は無視されます。文字列は、最初の認識されない文字まで解析されます。数字が何も認識されない場合、関数は NaN を返します。 例： `parseFloat("-3.14e-15")` –> -3.14e-15 `parseFloat("-3.14e-15 poodles")` –> -3.14e-15 `parseFloat("a fraction of a poodle")` –> NaN

文字列演算子

以下の演算子を使って文字列を操作することができます。

表 F.24 *IBM ILOG Script の文字列演算子*

構文	効果
文字列1 + *文字列2*	*文字列1* および *文字列2* の連結を含む文字列を返します。 例： `"Hello," + " world"` –> "Hello, world" 演算子 + を使って文字列を非文字列値に追加すると、まず非文字列値が文字列に変換されます。 例： `"Your age is " + 23` –> "Your age is 23" `23 + " is your age"` –> "23 is your age"

表 F.24 *IBM ILOG Script の文字列演算子 （続き）*

構文	効果
文字列1 `==` *文字列2* *文字列1* `!=` *文字列2*	演算子 `==` は、*文字列1* および *文字列2* が同じ場合にブール値 *true* を、そうでない場合に *false* を返します。2 は長さが同じで同じ文字列のシーケンスが含まれる場合に同一と見なされます。演算子 `!=` は、`==` の逆です。 例： `"a string" == "a string"` –> *true* `"a string" == "another string"` –> *false* `"a string" == "A STRING"` –> *false* `"a string" != "a string"` –> *false* `"a string" != "another string"` –> *true* 演算子 `==` および `!=` が文字列と数値の比較に使われる場合、文字列はまず数字に変換され、2 つの数値を比較します。 例： `"12" == "+12"` –> *false* `12 == "+12"` –> *true*

表 F.24 *IBM ILOG Script の文字列演算子 （続き）*

構文	効果
文字列1 `<` *文字列2* *文字列1* `<=` *文字列2* *文字列1* `>` *文字列2* *文字列1* `>=` *文字列2*	演算子 `<` は、*文字列1* が *文字列2* よりも辞書式分類上、厳密に優先される場合に *true* を、そうでない場合に *false* を返します。演算子 `<=` は、*文字列1* が *文字列2* よりも辞書式分類上、厳密に優先されるかまたは等しい場合に *true* を、そうでない場合に *false* を返します。 例： `"abc" < "xyz"` –> *true* `"a" < "abc"` –> *true* `"xyz" < "abc"` –> *false* `"abc" < "abc"` –> *false* `"abc" > "xyz"` –> *false* `"a" > "abc"` –> *false* `"xyz" > "abc"` –> *true* その他 これらのいずれかの演算子を使って文字列と非文字列値を比較すると、まず非文字列値が文字列に変換されます。 例： `"2" >= 123` –> *true* `123 < "2"` –> *false* これらいずれかの演算子を使って文字列と数値を比較すると、文字列が数字に変換されてから、2 つの数値を比較します。 例： `"10" > "2"` –> *false* `10 > "2"` –> *true*

ブール型

- *ブール型リテラル構文*
- *ブール型への自動変換*
- *ブール型メソッド*
- *論理演算子*

ブール型リテラル構文

ブール型リテラルには、以下の 2 つがあります。ブール型 *true* を表す `true`、およびブール型 *false* を表す `false`。

数字に変換 される場合、*true* は 1 を、*false* は 0 を返します。

ブール型への自動変換

引数の 1 つにブール型を予測する関数またはメソッドに非ブール型が渡された場合、その値は以下のようにブール型に自動的に変換されます。

- 数字 0 は *false* を返します。
- 空の 文字列 "" は *false* を返します。
- ヌル値は、*false* を返します。
- 未定義値は、*false* を返します。
- 他のすべての非ブール型値は、*true* を返します。

例 :

```
if ("") writeln("True"); else writeln("False");
if (123) writeln("True"); else writeln("False");
```

displays "False", then "True".

ブール型メソッド

次に示すのは、唯一のブール型メソッドです。

***表 F.25** IBM ILOG Script のブール型メソッド*

構文	効果
ブール型 .toString( )	ブール型を表す文字列を "true" または "false" で返します。 例 : true.toString –> "true" false.toString –> "false"

論理演算子

以下のブール型演算子を使用できます。

> ***メモ :*** C/C++ プログラマの方へ *: これらの演算子は、C および C++ のものと同じです。*

表 F.26 *IBM ILOG Script の論理演算子*

構文	効果
! *ブール型*	論理否定 例： `!` `true` –> *false* `!` `false` –> *true*
式1 `&&` *式2*	*式1* および *式2* の両方のブール型式が true の場合、*true* を返します。そうでない場合は、*false* を返します。 *式1* が false の場合、この式は *式2* を評価せずにただちに *false* を返します。したがって、*式2* の二次作用は考慮されません。 例： `true && true` –> *true* `true && false` –> *false* `false &&` *whatever* –> *false*; *whatever* は評価されません。
式1 `||` *式2*	*式1* または *式2* のいずれかのブール型式が true の場合、*true* を返します。そうでない場合は、*false* を返します。 *式1* が true の場合、この式は *式2* を評価せずにただちに *true* を返します。したがって、*式2* の二次作用は考慮されません。 例： `false ||` `true` –> *true* `false ||` `false` –> *false* `true ||` *whatever* –> *true*; *whatever* は評価されません。
条件 `?` *式1* `:` *式2*	*条件*が true の場合、この式は *式1* を返します。そうでない場合は、*式2* を返します。 *条件*が true の場合、式 *式2* は評価されません。したがって、含まれる可能性のある二次作用は考慮されません。同様に、*条件* が false の場合、*式1* は評価されません。 例： `true ?3.14 :` *whatever* –> 3.14 `false ?` *whatever* `:"Hello"` –> "Hello"

配列

本セクションは、次のような構成になっています。

- *IBM ILOG Script の配列*
- *配列コンストラクタ*
- *配列のプロパティ*
- *配列メソッド*

IBM ILOG Script の配列

配列では、0 で始まるインデックスを通じて参照され、順番に並べられた値のセットを操作するための方法を提供します。他の言語の配列と異なり、IBM ILOG Script の配列には固定サイズがなく、新しい要素が追加されると自動的に拡張されます。たとえば、次のプログラムでは、空の配列を作成してから新しい要素を追加します。

```
a = new Array() // Create an empty array
a[0] = "first" // Set the element 0
a[1] = "second" // Set the element 1
a[2] = "third" // Set the element 2
```

配列は内部で分散オブジェクトとして表現されます。したがって、要素 0 および要素 10000 だけが設定されている配列は 0 ～ 10000 の間の 9999 の要素ではなく、それら 2 つの要素を格納するのに十分なメモリを占有します。

配列コンストラクタ

配列コンストラクタには、以下の 2 つの明確な構文があります。

***表 F.27** IBM ILOG Script の配列コンストラクタ構文*

構文	効果
`new Array(`*長さ*`)`	ヌル に設定された 0 ～ *長さ*-1 の要素を持つ*長さ* の新しい配列を返します。 *長さ* が数値でなく、数字に変換 が NaN を返する場合、2 つ目の構文を使います。 例： `new Array(12)` –> an array *a* with length 12 and a[0] to a[11] containing *null*. `new Array("5")` –> an array *a* with length 5 and a[0] to a[4] containing *null*. `new Array("foo")` –> 2 番目の構文を参照。
`new Array` (*要素 1*, ..., *要素 n*)	*要素 1* を含む a[0]、*要素 2* を含む a[1] のある、長さ *n* の新しい配列 *a* を返します。引数が指定されていない場合、つまり *n*=0 の場合、空の配列が作られます。*n*=1 および *要素 1* が数値か、または数値に変換できる場合、最初の構文が使われます。 例： `new Array(327, "hello world")` –> an array *a* of length 2 with a[0] == 327 and a[1] == "hello world". `new Array()` –> 長さ 0 の配列。 `new Array("327")` –> 最初の構文を参照。

配列のプロパティ

配列のプロパティは以下のとおりです。

***表 F.28** IBM ILOG Script の配列プロパティ*

構文	効果

配列[*index*]	*インデックス*を 0 ～ 2e32-2 に変換できる場合 (332 ページの*数値への自動変換* 参照)、*配列*[*index*] は、配列の *index* 番目の要素の値です。そうでない場合は、標準プロパティ・アクセスと見なされます。この要素が設定されていない場合、*null* が返されます。
	例 : 配列 *a* が次のように作成されたと仮定しましょう。
	`a = new Array("foo", 12, true)`
	その場合、次のようになります。
	`a[0]` –> "foo"
	`a[1]` –> 12
	`a[2]` –> *true*
	`a[3]` –> *null*
	`a[1000]` –> *null*
	配列の要素が現在の配列の長さよりも多く設定されている場合、配列は自動的に拡張されます。
	`a[1000] = "bar"`
	`// the array is automatically expanded.`
	他のプロパティとは異なり、配列の数値プロパティは for..in ステートメントによって*列挙されません*。

配列 .length	*配列*に設定された要素のもっとも多いインデックスである *配列*の長さに 1 を足したもの。必ず 0 および 2e31-1 に含まれます。配列に新しい要素が設定され、そのインデックスが現在の配列の長さ以上の場合、length プロパティは自動的に増加します。
	例 : 配列 *a* が次のように作成されたと仮定しましょう。
	`a = new Array("a", "b", "c")`
	その場合、次のようになります。
	`a.length` –> 3
	`a[100] = "bar"`; `a.length` –> 101
	長さプロパティを設定して、配列の長さを変更することもできます。
	`a = new Array(); a[4] = "foo"; a[9] = "bar";`
	`a.length` –> 10
	`a.length` = 5
	`a.length` –> 5
	`a.length` –> 5
	`a[4]` –> "foo"
	`a[9]` –> null

配列メソッド

配列には以下のメソッドがあります。

表 F.29 IBM ILOG Script の配列メソッド

構文	効果
配列 .join ([*区切り文字*])	文字列に変換され、連結され、*区切り文字*で区切られた配列の要素が含まれる文字列を返します。*区切り文字*が省略されている場合は、"," と見なされます。初期化されていない要素は、空の文字列に変換されます。文字列メソッド split (分割) も参照してください。 例 : 配列 *a* が次のように作成されたと仮定しましょう。 `a = new Array("foo", 12, true)` その場合、次のようになります。 `a.join("//")` –> "foo//12//true" `a.join()` –> "foo,12,true"

配列 .sort ([*関数*])	配列をソートします。要素はその場所でソートされ、新しい配列は作成されません。
	*関数*が指定されていない場合、*配列*は辞書式分類的にソートされます。要素は、文字列に変換し、<= 演算子を使って比較されます。この順番では、"20" < "5" が true なため、数値 20 が数値 5 よりも先になります。
	*関数*が指定されている場合、配列はその関数の戻り値に従ってソートされます。この関数では、必ず *x* および *y* の 2 つの引数を取り、以下を返します。
	◆ *x* が *y* よりも小さい場合は ñ1。
	◆ *x* と *y* が等しい場合は 0。
	◆ *x* が *y* よりも大きい場合は 1。
	例 : 関数 compareLength が次のように定義されていると仮定しましょう。
	`function compareLength(x, y) {`
	`  if (x.length < y.length) return -1;`
	`  else if (x.length == y.length) return 0;`
	`  else return 1;`
	`}`
	また、配列 *a* が次のように作成されたと仮定しましょう。
	`a = new Array("giraffe", "rat", "brontosaurus")`
	`a.sort()` は、要素を次のように並べ替えます。
	`"brontosaurus" "rat" "giraffe"`
	`a.sort(compareLength)` は、要素を次のように並べ替えます。
	`"rat" "giraffe" "brontosaurus"`

表 F.29 IBM ILOG Script の配列メソッド（続き）

構文	効果
配列 .reverse( )	配列の要素の置き換え : 最初の要素は最後に、2 番目の要素は最後から 2 番目になります。要素はその場所で逆になります。新しい配列は作成されません。 例 : 配列 *a* が次のように作成されたと仮定しましょう。 `a = new Array("foo", 12, "hello", true, false)` Then a.reverse() changes *a* so that: `a[0]` –> *false* `a[1]` –> *true* `a[2]` –> "hello" `a[3]` –> 12 `a[4]` –> "foo"
配列.toString( )	文字列 "[object Object]" を返します。

オブジェクト

本セクションは、次のような構成になっています。

- *IBM ILOG Script のオブジェクト*
- *メソッドの定義*
- *this キーワード*
- *オブジェクト・コンストラクタ*
- *ユーザ定義のコンストラクタ*
- *組み込みメソッド*

IBM ILOG Script のオブジェクト

オブジェクトとは、定義済みのプロパティやメソッドを含まず (toString メャ b ドを除く)、新しいプロパティやメソッドを追加可能な値のことです。新しい空のオブジェクトは、オブジェクト・コンストラクタを使って作成することができます。たとえば、次のプログラムでは新しいオブジェクトを作成し、そのオブジェクトを変数 myCar に保存して、プロパティ 「名前」および「年」を追加します。

```
myCar = new Object() // o contains no properties
myCar.name = "Ford"
myCar.year = 1985
```

ここで

```
myCar.name -> "Ford"

myCar.year -> 1985
```

メソッドの定義

メソッドは 関数値 を含むプロパティのため、メソッドの定義は通常の関数の定義とその関数をプロパティに割り当てるだけです。

たとえば、次のプログラムでは、前のセクション *IBM ILOG Script のオブジェクト* に定義されている myCar オブジェクトにメソッド "start" を追加します。

```
function start_engine() {
  writeln("vroom vroom\n")
}

myCar.start = start_engine
```

ここで、式 myCar.start() は、start_engine として定義されている関数を呼び出します。関数やメソッドにさまざまな名前を使う理由は、混乱を避けるためです。以下のように書くことができます。

```
function start() {
  writeln("vroom vroom\n")
}

myCar.start = start
```

this キーワード

メソッド内で、this キーワードを使って呼び出しオブジェクトを参照することができます。たとえば、次のプログラムでは、呼び出しオブジェクトの名前プロパティの値を返すメソッド getName を定義して、このメソッドを *myCar* に追加します。

```
function get_name() {
  return this.name
}

myCar.getName = get_name
```

コンストラクタ内で、this はコンストラクタが作成するオブジェクトを参照します。非メソッド・コンテキストで使われる場合、this はグローバル・オブジェクトの参照を返します。グローバル・オブジェクトには、最上位で宣言された変数、組み込み関数、コンストラクタが含まれます。

オブジェクト・コンストラクタ

オブジェクトは、以下のコンストラクタを使って作成されます。

***表 F.30** IBM ILOG Script のオブジェクト・コンストラクタ*

構文	効果
`new Object()`	プロパティのない新しいオブジェクトを返します。

ユーザ定義のコンストラクタ

`Object` コンストラクタだけでなく、次の構文を使ってユーザ定義の関数をオブジェクト・コンストラクタとして使用することができます。

***表 F.31** IBM ILOG Script のユーザ定義のコンストラクタ*

構文	効果
`new` *関数* (*引数 1*, ..., *引数 n*)	新しいオブジェクトを作成し、*関数* (*引数 1*, ..., *引数 n*) を呼び出してオブジェクトを初期化します。

コンストラクタ内で、`this` キーワードを使って初期化中のオブジェクトを参照することができます。

たとえば、次のプログラムでは自動車のコンストラクタを定義します。

```
function Car(name, year) {
  this.name = name
  this.year = year
  this.start = start_engine
}
```

ここで、以下を呼び出します

```
new Car("Ford", "1985")
```

プロパティ `name` および `year` 、そして `start` メソッドのある新しいオブジェクトを作成します。

組み込みメソッド

以下は、オブジェクトの唯一の組み込みメソッドです。

***表 F.32** IBM ILOG Script の組み込みメソッド*

構文	効果
オブジェクト.toString( )	文字列 "[object Object]" を返します。このメソッドは、オブジェクトの toString プロパティを割り当ててオーバーライドすることができます。

日付

本セクションは、次のような構成になっています。

- *IBM ILOG Script の日付値*
- *日付コンストラクタ*
- *日付メソッド*
- *日付関数*
- *日付演算子*

IBM ILOG Script の日付値

日付値では、日付および時刻を操作する方法を提供します。日付は、1970 年 1 月 1 日 00：00：00 UTC 以降の対応数値をミリ秒数単位で内部表現したものとして理解することができます。1970 年以前の日付は負で表すことができます。

> ***メモ:*** C++ プログラマの方へ : *標準の C ライブラリで操作される日付とは異なり、日付値は、1970 ～ 2038 の範囲に制限されませんが、1970 年を中心に約 285,616 年に渡ります。*

数字に変換 される場合、日付は、1970 年 1 月 1 日 00：00：00 UTC (世界協定時刻) 以降のミリ秒数を提供します。

日付コンストラクタ

日付コンストラクタには、以下の 4 つの異なる構文があります。

表 F.33 *IBM ILOG Script の日付コンストラクタ*

構文	効果
`new Date()`	現在の時間に対応する日付を返します。
`new Date(`*ミリ秒*`)`	1970 年 1 月 1 日 00：00：00 UTC に、*ミリ秒*を足した日付を返します。引数は負の値を使って 1970 年よりも前の日付を表すことができます。引数を数値に変換できない場合、3 つ目のコンストラクタ構文が使われます。 例： `new Date(0)` –> 1970 年 1 月 1 日 00：00：00 UTC を表す日付 `new Date(1000*60*60*24*20)` –> 1970 年 1 月 1 日 00：00：00 UTC より 20 日後を表す日付 `new Date(-1000*60*60*24*20)` –> 1970 年 1 月 1 日 00：00：00 UTC より 20 日前を表す日付

表 F.33 *IBM ILOG Script の日付コンストラクタ （続き）*

構文	効果
new Date(*文字列*)	以下の形式を持つ*文字列*によって表される日付を返します。 *月*/*日*/*年 時*:*分*:*秒 ミリ秒* *文字列*で表される日付は、現地時間が取得されます。 例 : new Date("12/25/1932 14:35:12 820") –> 1932 年 12 月 25 日 2 : 35 PM に 12 秒と 820 ミリ秒を足した日付 (現地時間)
new Date(*年*, *月*, [,*日* [,*時* [,*分* [,*秒* [,*ミリ秒*]]]]]]])	特定の年、月、日などを表す現地時間の新しい日付を返します。引数は以下のとおりです。 ◆ *年*: 任意の整数。 ◆ *月*: 範囲 0 ～ 11 (0= 1 月、1= 2 月など)。 ◆ *日*: 範囲 1 ～ 31、デフォルトは 1。 ◆ *時*: 範囲 0 ～ 23、デフォルトは 0。 ◆ *分*: 範囲 0 ～ 59、デフォルトは 0。 ◆ *秒*: 範囲 0 ～ 59、デフォルトは 0。 ◆ *ミリ秒*: 範囲 0 ～ 999、デフォルトは 0。 例 : new Date(1932, 11, 25, 14, 35, 12, 820) –> 1932 年 12 月 25 日 2 : 35 PM に 12 秒と 820 ミリ秒を足した日付 (現地時間) new Date(1932, 11, 25) –> 1932 年 12 月 25 日 00:00 を表す日付 (現地時間)

日付メソッド

日付には、以下のメソッドがあります。

表 F.34 *IBM ILOG Script の日付メソッド*

構文	効果
日付 .getTime( ) *日付* .setTime(*ミリ秒*)	1970 年 1 月 1 日 00 : 00 : 00 UTC (世界協定時刻) 以降のミリ秒数を返します (設定します)。 例 : 日付 ***d*** が次のように作成されたと仮定しましょう。 `d = new Date(3427)` その場合、次のようになります。 `d.getTime()` –> 3427
日付 .toLocaleString( ) *日付* .toUTCString( )	現地時間の日付を表す文字列を返します (UTC で)。 例 : 日付 ***d*** が次のように作成されたと仮定しましょう。 `d = new Date("3/12/1997 12:45:00 0")` その場合、次のようになります。 `d.toLocaleString()` –> "03/12/1997 12:45:00 000" `d.toUTCString()` –> "03/12/1997 10:45:00 000"、グリニッジ子午線に対して +2 時間の現地タイムゾーン・オフセットを仮定します。
日付 .getYear( ) *日付* .setYear(*年*)	*日付*の年を返します (設定します)。
日付 .getMonth( ) *日付* .setMonth(*月*)	*日付*の月を返します (設定します)。
日付 .getDate( ) *日付* .setDate(*日*)	*日付*の日を返します (設定します)。
日付 .getHours( ) *日付* .setHours(*日*)	*日付*の時を返します (設定します)。
日付 .getMinutes( ) *日付* .setMinutes(*日*)	*日付*の分を返します (設定します)。
日付 .getSeconds( ) *日付* .setSeconds(*日*)	*日付*の秒を返します (設定します)。
日付 .getMilliseconds( ) *日付* .setMilliseconds(*日*)	*日付*のミリ秒を返します (設定します)。
日付 .toString( )	同じ値を *日付* .toLocaleString() として返します。

日付関数

以下の関数では日付を操作します。

表 F.35 IBM ILOG ScriptIBM ILOG Script の日付関数

構文	効果
`Date.UTC(` *文字列*`)`	`new Date(` *文字列*`)` と同じですが、*文字列*は UTC で取得され、結果は日付オブジェクトではなく数値として返されます。
`Date.parse(` *文字列*`)`	`new Date(` *文字列*`)` と同じですが、結果は日付オブジェクトではなく数値として返されます。

日付演算子

日付を処理する特定の演算子はありませんが、数値演算子 は自動的に引数を数値に変換するため、これらの演算子を使って 2 つの日付の間の経過時間を計算したり、日付に特定の時間を追加することができます。例 :

日付 1 - *日付 2* –> *日付 1* および *日付 2* の間の経過時間 (ミリ秒)。

日付1 < *日付2* –> *日付1*が *日付2*よりも前の場合は true、そうでない場合は false。

`new Date(`*date*`+10000)` –> *日付*の 10000 ミリ秒後を表す日付。

次のプログラムでは、ステートメント <*do something*> の実行にかかった時間をミリ秒で表示します。

```
before = new Date()
<do something>
after = new Date()
writeln("Time for doing something:", after-before, " milliseconds.")
```

ヌル値

本セクションは、次のような構成になっています。

- *IBM ILOG Script のヌル値*
- *ヌルのメソッド*

IBM ILOG Script のヌル値

*ヌル*値は、情報がないことを特定するためにある場所で使われる特別な値です。たとえば、未設定の配列要素にはデフォルトの*ヌル*値があります。*ヌル*値は、あるコンテキストに情報がないことを特定する未定義値とは異なります。

*ヌル*値は、キーワード `null` を使ってプログラムで参照することができます。

`null` –> *ヌル*値。

数字に変換 の場合、*ヌル*は 0 を返します。

ヌルのメソッド

以下は、*ヌル*の唯一のメソッドです。

表 F.36 *IBM ILOG Script のヌル・メソッド*

構文	効果
ヌル `.toString( )`	文字列 "null" を返します。

未定義の値

本セクションは、次のような構成になっています。

- *IBM ILOG Script の未定義値*
- *未定義のメソッド*

IBM ILOG Script の未定義値

*未定義*値は、情報がないことを特定するためにある場所で使われる特別な値です。たとえば、定義されていない値のプロパティへのアクセス、または宣言されたものの初期化されていないローカル変数は、*未定義*値を返します。

プログラムの*未定義*値を参照する方法はありません。`typeof` 演算子を使うと、値が未定義値かどうかを確認することができます。

`typeof(` *値* `) == "undefined"` –> *値*が未定義の場合は ***true***、そうでない場合は ***false***。

未定義のメソッド

以下は、*未定義*の唯一のメソッドです。

***表 F.37** IBM ILOG Script の未定義メソッド*

構文	効果
未定義 .toString( )	文字列 "undefined" を返します。

関数

本セクションは、次のような構成になっています。

- *IBM ILOG Script の関数*
- *関数メソッド*

IBM ILOG Script の関数

IBM ILOG Script で、関数は他のタイプの値のように操作可能な通常の値です (「ファースト・クラス」値として知られます)。それらの値は関数に渡したり、関数によって返されたり、変数やオブジェクト・プロパティなどに保存されることがあります。

たとえば、関数 parseInt は、parseInt 変数に保存される実際の関数値です。

```
parseInt –> 関数値
```

たとえば、この関数値は、次のように他の変数に割り当てることができます。

```
myFunction = parseInt
```

次に、この変数を通じて呼び出されます。

```
myFunction("-25") –> -25
```

関数メソッド

以下は、関数の唯一のメソッドです。

***表 F.38** IBM ILOG Script の関数メソッド*

構文	効果
関数 .toString( )	関数に関する情報が含まれる文字列を返します。 例： "foo".substring.toString() –> "[primitive method substring]" eval.toString() –> "[primitive function eval]"

その他

その他の関数を以下の表にまとめます。

***表 F.39** その他の関数*

構文	効果
stop( )	現在のステートメントでプログラムの実行を停止します。デバッガが有効な場合、デバッグ・モードに入ります。
write(*引数 1*, ..., *引数 n*) writeln(*引数 1*, ..., *引数 n*)	引数を文字列に変換して、現在のデバッグ出力に印刷します。この実行は、IBM ILOG Script を搭載しているアプリケーションによって異なります。関数 writeln では出力の最後に新しい行を印刷しますが、write は行いません。

表 F.39 *その他の関数*

構文	効果
`loadFile(` *文字列*`)`	パスが*文字列*のスクリプト・ファイルを読み込みます。パスは絶対または相対です。このパスが既存のファイルを指定しない場合、IBM ILOG Script が搭載されているアプリケーションによって異なるメソッドを使ってファイルが検索されます。通常、*文字列*の名前のファイルは、アプリケーション設定で指定したディレクトリ・リストで検索されます。
`eval(` *文字列*`)`	プログラムとして*文字列*を実行し、最後に評価した式の値を返します。*文字列*のプログラムでは、関数を定義できないことを除いて、言語のすべての機能を使うことができます。つまり、関数ステートメントは、*文字列*では使用できません。 例： `eval("2*3")` –> 6 `eval("var i=0; for (var j=0; j<100; j++) i=i+j; i")` –> 4950 `n=25; eval("Math.sqrt(n)")` –> 5 `eval("function foo(x) { return x+1 }")` –> エラー

索引

E

F

G

H

I

L

M

N

P

R

S

T

U

V

け

こ

さ

し

す

せ

た

つ

て

と

な

に

ぬ

は

れ

ろ