IBM ILOG Views

Maps 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

前書き　**本書について 10**

書体の規則 11

命名規則 11

第 1 章　**IBM ILOG Views Maps の概要 14**

IBM ILOG Views Maps とは 15

IBM ILOG Views Maps のインストール 16

第 2 章　**IBM ILOG Views Maps のスタートアップ 18**

例で使用するデータ 18

地図の作成 19

IBM ILOG Views マップ・ビルダの実行 20

沿岸線情報が含まれるファイルのロード 21

IBM ILOG Views マップ・ビルダのツールバーを使用する 25

道路を地図に読み込む 25

町の読み込み 26

大きな町を地図に読み込む 28

レイヤの編集 29

ファイルの保存 32

ロード・オン・デマンドで地図を作成する 32

基本構造マップと CADRG ファイルのロード 33

Oracle Spatial で地図を作成する 36
Oracle Spatial データベースのレイヤの作成 36
マップ・ビルダによる Oracle Spatial レイヤの表示 39
ロード・オン・デマンドで Oracle Spatial レイヤを表示する 42
情報の永続性をテストする 42

第 3 章 IBM ILOG Views Maps リーダ・フレームワーク 44

地図作成用のクラス : 概要 45
地図機能 46
地図機能のジオメトリ 47
地図機能のアトリビュート 47
アトリビュートをグラフィック・オブジェクトに付加する 48
レンダラ 49
レンダラの概要 49
色付き線のレンダラを作成する 51
レンダラの永続化 53
既存レンダラの拡張 54
機能イテレータ 55
IlvMapFeatureIterator の概要 56
新しいリーダの作成 57
ターゲット投影図法を選択する 64
IBM ILOG Views に地図をロードする 65
IBM ILOG Views 形式の地図をロードする 65
マップ・ローダー 66
縮尺フィルタ 70

第 4 章 ロード・オン・デマンドの使用 72

ロード・オン・デマンドの機能 73
タイリング・グリッドの構造とサイズ 78
タイリング・グリッドの構造 80
タイリング・グリッドのサイズ 81
タイル・ステータスの表示 81

ロード・オン・デマンドの制御 83
ビジビリティ・フィルタを使って、ロード・オン・デマンドを制御する 84
API を介してタイルをロードする 85
エラーおよびロード・オン・デマンド・イベントの管理 85
タイルのキャッシュ処理 88
タイル・レイヤの保存 88
新しいキャッシュ・アルゴリズムを記述する 89
新しいデータ・ソースにロード・オン・デマンドを実装する 91

第 5 章 定義済みリーダ 94

形状ファイル・リーダ 95
形状ファイル形式の概要 95
形状ファイル形式を読み込むためのクラス 95
形状ファイルのロード・オン・デマンド 98
DTED ファイル・リーダ 103
DTED 形式の概要 104
DTED 形式を読み込むためのクラス 104
デジタル地形モデルのグラフィカル・レンダリング 104
CADRG ファイル・リーダ 105
CADRG 形式を読み込むためのクラス 105
イメージ・ファイル・リーダ 107
IlvImageReader クラス 107
IlvImageTileLoader クラス 107
例 108
IlvImageLayer クラス 109
GeoTIFF リーダ 109
GeoTIFF 形式 109
IlvGeoTIFFReader クラス 110
IlvGeoTIFFTileLoader クラス 110
IlvGeoTIFFLayer クラス 111
IlvGeoTIFFTiler クラス 111
Oracle Spatial 機能 111

リレーショナル・モデル・クラス 112
オブジェクト・モデル・クラス 115
S57 マップ・リーダ 122
S57 形式を読み込むためのクラス 123
スタイル、色およびアイコンの設定 124

第 6 章 地図投影図法 126

地図投影図法の概要 127
円筒図法 127
円錐図法 128
方位図法 129
正積図法または正角図法 129
投影データ 例 130
サンプル・アプリケーションの実行 131
投影図法宣言を含める 131
Main 関数 131
投影データの作成 132
データの投影 132
投影結果の出力 133
逆投影の計算 133
地理座標の印刷 134
サンプル・コード一式 134
投影メソッドとパラメータ 135
順関数と逆関数 135
投影図法パラメータ 136
ユーティリティ 137
楕円 137
楕円の概要 137
投影図法に楕円を関連付ける 138
新しい楕円の定義 138
定義済みの楕円 139
単位コンバータ 141

単位コンバータの直接使用 141
投影図法でコンバータを使用する 142
単位コンバータの定義 142
定義済み単位コンバータの使用 142
測地原点が異なる座標間の変換 144
測地系の水平移動 144
インポートした地図の上にグラフィック・オブジェクトを追加する 145
サンプル・アプリケーションの実行 145
Sample クラス、Main 関数、コンストラクタの定義 146
地図情報の取得 149
マウス位置の表示 150
新しい投影図法の作成 151
ステップ 1 新しい投影図法の定義 152
投影図法の定義 153
順投影を書く 154
逆投影を書く 155
ステップ 2 新しい投影図法の定義 157
ステップ 3 新しい投影図法の定義 160

第 7 章 地図データ 162

推奨無料ソース 162

索引 166

本書について

*このユーザ・マニュアル*では、*IBM® ILOG® Views Maps 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 Maps 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 Maps の根底となる基礎概念を説明するとともに、地図オブジェクトの作成方法と使用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 Maps の概要* では、IBM ILOG Views Maps の概略を説明します。
- 2 章 *IBM ILOG Views Maps のスタートアップ*では、IBM ILOG Views Maps のマッピング機能について、単純なチュートリアルが提供されます。
- 3 章 *IBM ILOG Views Maps リーダ・フレームワーク*では、IBM ILOG Views に地図データを読み込む API について説明します。
- 4 章 *ロード・オン・デマンドの使用*では、ロード・オン・デマンド機構について説明します。
- 5 章 *定義済みリーダ*では、IBM ILOG Views Maps に備わる定義済みリーダを紹介します。
- 6 章 *地図投影図法* では、投影図法の使い方を説明します。
- 7 章 *地図データ*では、ダウンロードできる地図データの推奨無料ソースの一覧を提供します。

表記法

書体の規則

以下の書体に関する規則は、このマニュアル全体に適用されます。

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

命名規則

以下の命名規則は、マニュアル全体を通して API に適用されます。

- IBM ILOG Views Foundation ライブラリで定義されている型、クラス、関数、マクロの名前は `Ilv` で始まります。
- クラス名、およびグローバル関数は、最初の文字が大文字で表された連結語として記載されます。

  class `IlvDrawingView`;

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

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

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

章

1

IBM ILOG Views Maps の概要

この章では、IBM® ILOG® Views Maps パッケージの概要を説明します。
IBM ILOG Views Maps は、地図背景の使用が必要な高性能アプリケーションを構築するための、フル機能 C++ クラス・ライブラリで構成されています。

このモジュールは、IBM ILOG Views で作成されたすべての 2D グラフィック・アプリケーションに基本機能を提供する、IBM ILOG Views Graphics Framework の上に構築されています。

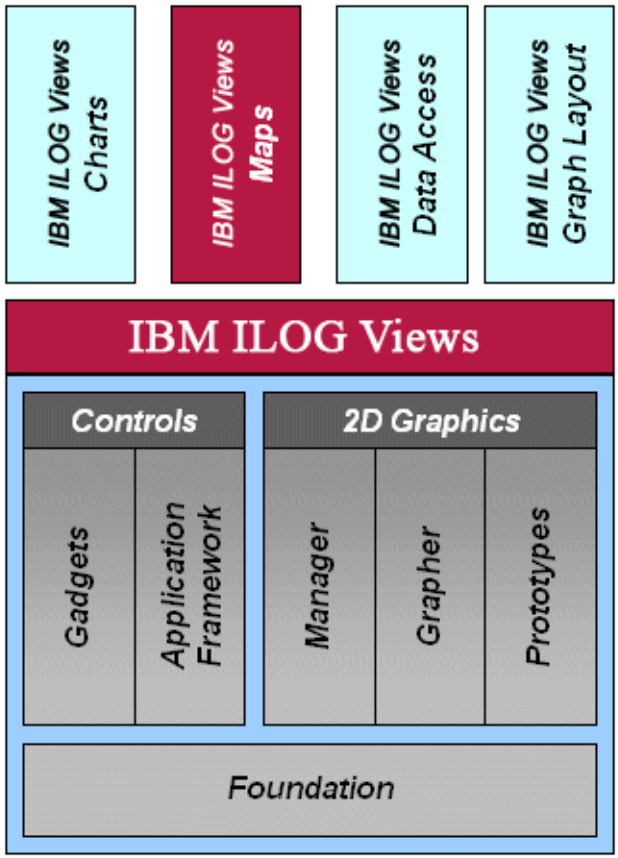


図1.1　Maps パッケージと IBM ILOG Views *コンポーネント・スイート*

IBM ILOG Views Maps とは

IBM® ILOG® Views Maps は、以下のコンポーネントで構成されます。

- マップ・ビルダ
- クラス・ライブラリ

マップ・ビルダ

IBM ILOG Views Maps には、実行中のアプリケーションに読み込み可能な地図を作成できるエディタ、マップ・ビルダが備わっています。マップ・ビルダで作成した地図では、膨大なデータを処理できる最新のロード・オン・デマンド機構など、IBM ILOG Views Maps のほとんどの機能を使用できます。

マップ・ビルダの使用方法については、2 章 *IBM ILOG Views Maps のスタートアップ*で説明します。

クラス・ライブラリ

IBM ILOG Views Maps はクラス・ライブラリで、地図作成の必要があるアプリケーションの構築で時間を節約することができます。このクラス・ライブラリの API は、アプリケーションのニーズに応じて容易に完全なカスタマイズが可能で、また拡張もできます。

ライブラリは、以下の内容で構成されています。

- 地図データを読み込み、IBM ILOG Views マネージャでグラフィック・オブジェクトとして表示するためのクラス。3 章 *IBM ILOG Views Maps リーダ・フレームワーク*を参照してください。
- 大量のデータを管理するためのロード・オン・デマンド機能を実行するクラス。4 章 *ロード・オン・デマンドの使用*を参照してください。
- CADRG、DTED または Oracle Spatial など、広く採用されているさまざまな地図形式に対応する定義済みリーダを実行するクラス。5 章*定義済みリーダ*を参照してください。
- グラフィック座標の地図参照および再投影を管理するためのクラス。6 章*地図投影図法*を参照してください。

> ***メモ:*** *このマニュアルでは、ヘッダー・ファイルがパッケージと同じ名前のディレクトリにあるクラスを、パッケージと呼びます。たとえば、パッケージ projection は、projection ディレクトリにあります。*

IBM ILOG Views Maps のインストール

この製品を使用するためには、次のソフトウェアが必要です。

- IBM® ILOG® Views Maps

- IBM ILOG Views、IBM ILOG Views 2D Graphics Standard または IBM ILOG Views 2D Graphics Professional

IBM ILOG Views Maps の Oracle Spatial リーダを使用する場合は、IBM ILOG DB リンク のインストールも必要です。IBM ILOG Views Maps のライセンスで IBM ILOG DB リンク も使用することができます。

章

2

IBM ILOG Views Maps のスタートアップ

この章では IBM® ILOG® Views Maps の主な機能を、IBM ILOG Views のマップ・ビルダでの地図作成を介して説明します。さまざまな例を取り上げて、このビルダで使用可能な定義済みマップ・リーダの効率や Oracle Spatial データベースへの簡単な接続、その他の機能について説明します。

この章では、以下の例について説明します。

- *地図の作成*
- *ロード・オン・デマンドで地図を作成する*
- *Oracle Spatial で地図を作成する*

例で使用するデータ

この章では、次の一覧で説明されているようにデモ目的でデータを使用します。ダウンロード可能な推奨地図データの無償ソースの一覧は、*地図データ*を参照してください。

- **地図の作成 :** フィリピンの ESRI 形状ファイル
- **ロード・オン・デマンドで地図を作成する :**CADRG ファイルおよび `waco.ivl` ファイル。

- **Oracle Spatial で地図を作成する：**世界の ESRI 形状ファイル

これらの例では、データは次のディレクトリにダウンロードされています。

- `<shapedir>` (フィリピンおよび世界の ESRI 形状ファイル)
- `<wacodir>` (`waco.ivl` ファイル)
- `<cadrgdir>` (DADRG ファイル・データ)

地図の作成

> ***メモ：****このセクションでは、適切なデータが既にワークステーションにダウンロードされているものとみなします。ダウンロード可能な推奨地図データの無償ソースの一覧は、地図データを参照してください。この例では、ESRI 形状ファイル・データを使用します。*

南太平洋のフィリピン諸島の地図を作成し、沿岸線、町、道路を表示してください。そのためには、ファイル形式、または地図作成サーバのデータ・リクエスト形式による地図データが必要です。いずれの場合も、ILOGIBM® ILOG® Views マップ・ビルダの定義済みリーダを使うことで、さまざまなソースからデータを簡単にロードすることができます。データがマップ・ビルダにロードされたら、IBM ILOG Views Maps 内での作業を開始します。

最初の例では、地図データ・ファイルを使用します。

> ***メモ：****マップ・ビルダにファイルをロードする際、ロードされる情報にさまざまなパラメータを設定します。これらのパラメータ、レンダラ、ターゲットの投影図法などについての詳細は、さまざまなファイル形式と併せ、以下の各章で説明します。*

ファイルのロードが終わると、地図が次のように表示されます。

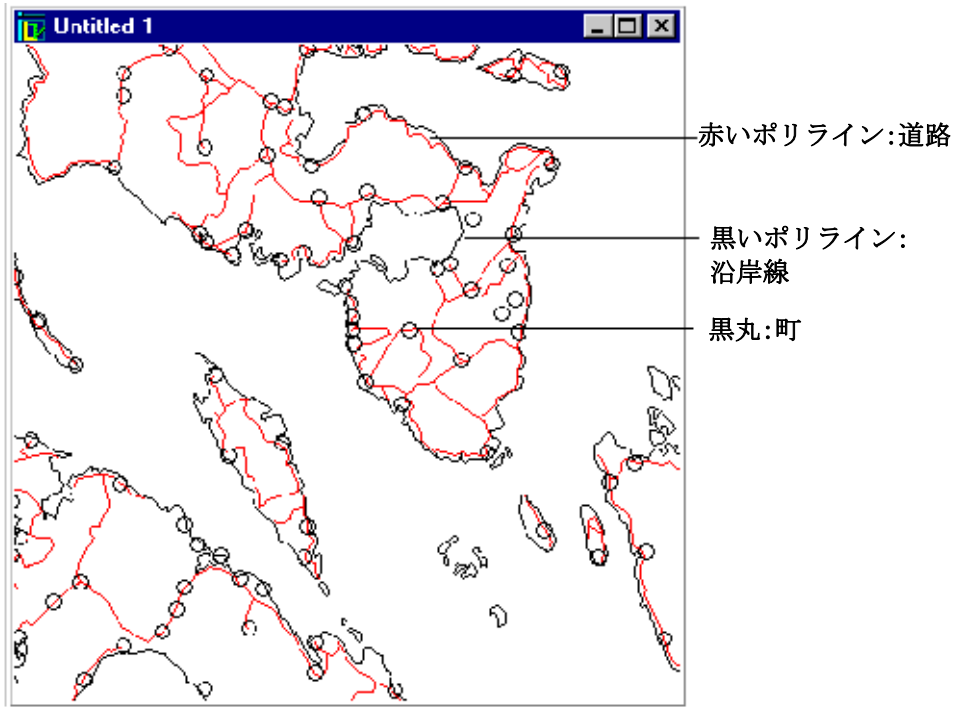


図2.1　フィリピン諸島の地図

この例では、以下の方法を学びます。

- IBM ILOG Views マップ・ビルダによるファイル (この例では形状ファイル) のロード
- レンダラの選択
- 縮尺フィルタの使用
- オブジェクト・アトリビュートの表示
- レイヤ名の編集
- IBM ILOG Views ファイルの保存

IBM ILOG Views マップ・ビルダの実行

IBM® ILOG® Views マップ・ビルダを起動する

- UNIX® システム上 : `<installdir>/bin` ディレクトリの、`mapbuilder` 実行可能ファイルを使用します。

◆ Microsoft® Windows® システム上 : `mapbuilder.exe` ファイルを使います。

IBM ILOG Views マップ・ビルダを起動すると、次のメイン・ウィンドウが表示されます。

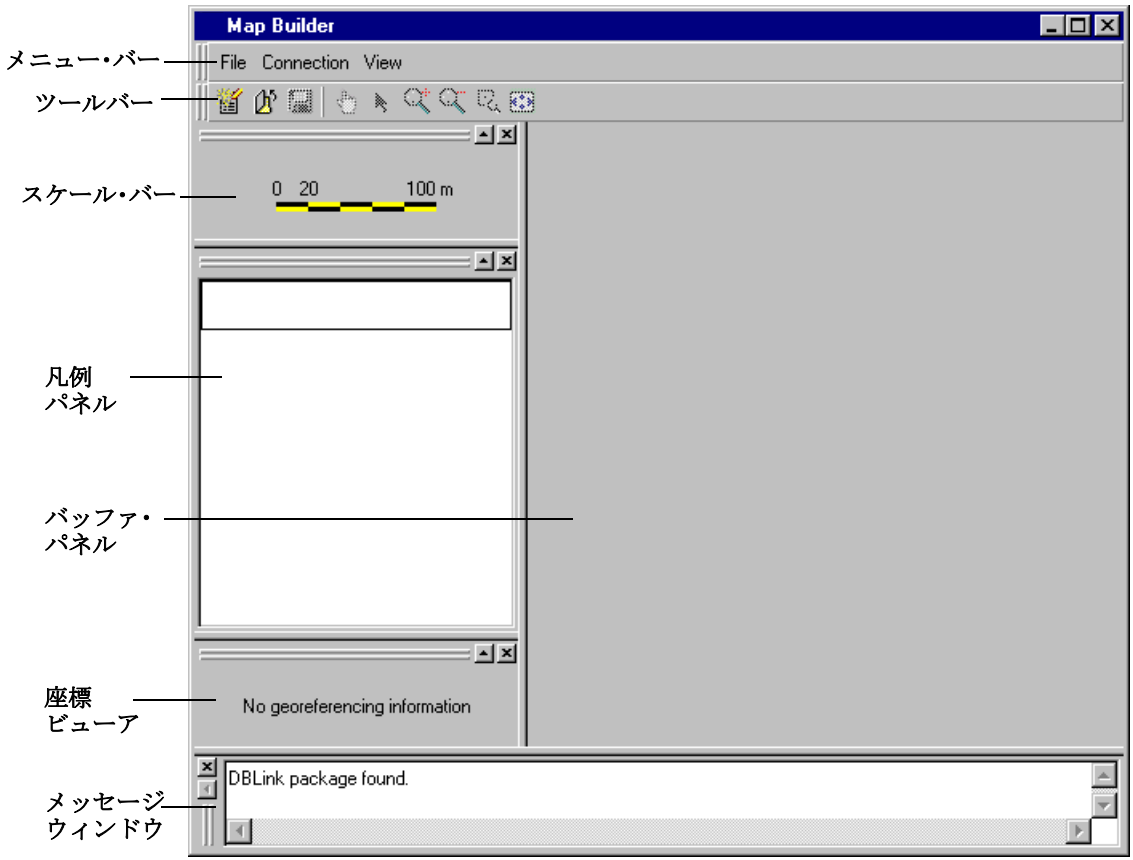


図 2.2 IBM ILOG Views *マップ・ビルダ・メイン・ウィンドウ*

沿岸線情報が含まれるファイルのロード

例えば、ロードする最初の形状ファイル `Poline.shp` には、フィリピンの沿岸線情報が含まれています。各形状ファイルには、道路、町、沿岸線などの「テーマ」が 1 つと、同じタイプ (線、点、多角形など) のファイルのオブジェクトがすべて含まれています。このファイルのテーマは沿岸線です。

このセクションでは、ファイルのロード方法と、ツールバーの使い方について説明します。

次のようにファイルをロードします。

1. [ファイル (file)] メニューから [ファイルをロード (Load file)] を選択するか、ツールバーの対応するアイコンをクリックします。
2. `<shapedir>` を探して、`Poline.shp` ファイルを開きます。

   [パラメータ (Source file parameters)] ウィンドウが表示されます。

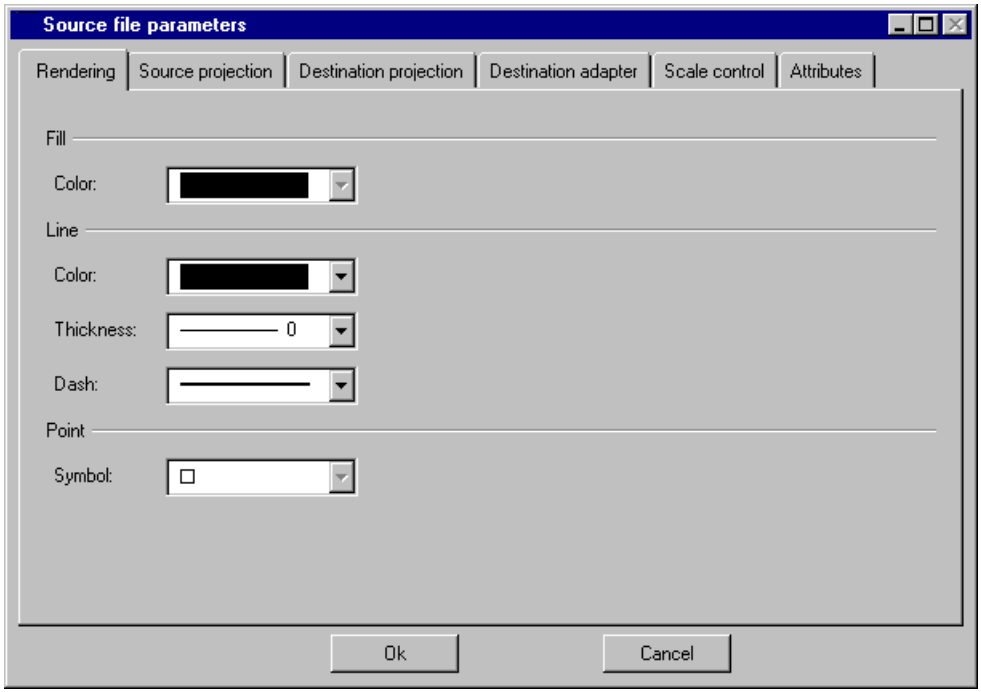
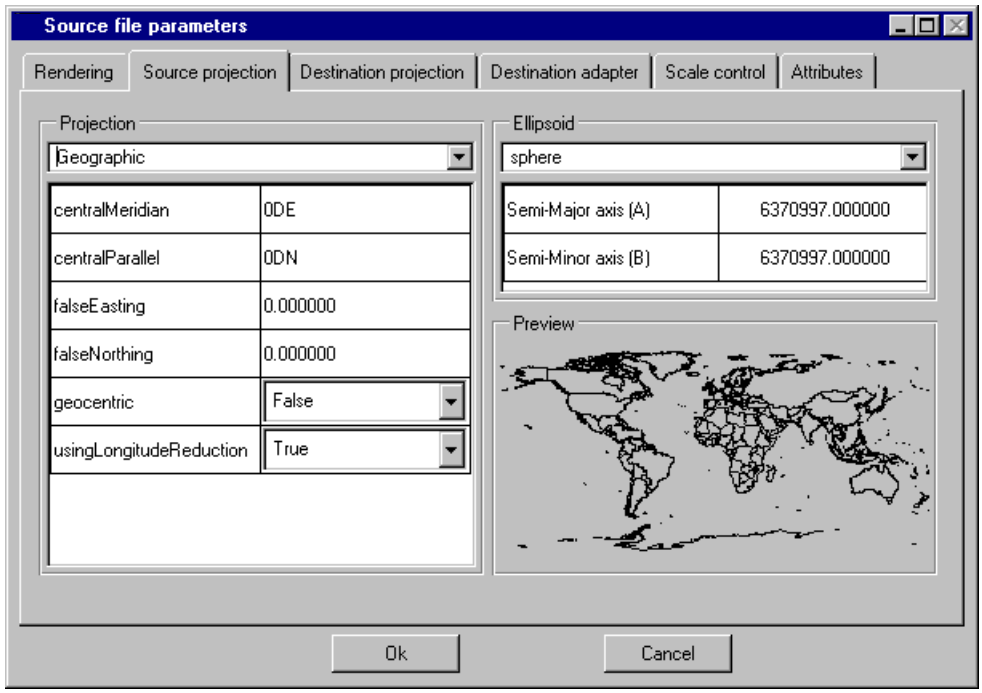


図 2.3 [パラメータ] ウィンドウ

3. [パラメータ] ウィンドウで、[レンダリング (Rendering)] ページのデフォルトのパラメータをそのまま使用します。

   これらのデフォルト・パラメータに基づいて、沿岸線が黒の実線でレンダリングされます。
4. [ソース (Source projection)] ページで、ロードするファイルのソース投影図法を指定できます (地理参照されていない地図ファイルをロードすると、このページが表示されます)。このシナリオのファイルは、地理投影図法によるものです。



> ***メモ:*** *多くの場合、形状ファイルの座標系は地理投影図法であるため、点の座標は、度数で表される緯度と経度になります。*

5. [ターゲット投影図法 (Destination projection)] ページで、ファイルのターゲット投影図法を指定できます。IBM® ILOG® Views Maps では、異なる投影図法のジオメトリを、形状ファイルの読み込み時に変換できます。このサンプルの場合は、Geographic を選択する必要があります。

> ***メモ:*** *地図に読み込まれるファイルは、すべて同じ投影図法にする必要があります。*IBM ILOG Views *マップ・ビルダでは、[パラメータ] ウインドウの [ターゲット投影図法] タブを使って、地図の投影図法を設定します。*
> IBM ILOG Views *形式ではない複数のファイルをバッファにロードする場合、[ターゲット投影図法] タブは最初のファイルのみに表示されます。後続のファイルは、これをターゲットの投影図法としてロードされます。*

6. [アトリビュート (Attributes)] ページで、*地図機能*のアトリビュートを読み込む場所が指定できます。沿岸線を含むこのファイルでは、アトリビュートは何も情報を伝えないため、このオプションはクリアすることができます。これにより、メモリ・リソースの節約になります。

7. [ターゲット・アダプタ (Destination adapter)] ページでは、地図の精度であるグラフィック座標アダプタを指定できます。1m のデフォルト値をそのままにできます。

8. [スケール・コントロール (Scale Control)] ページで、ロードしたデータを含むレイヤにビジビリティ・フィルタが指定できます。このファイルでは、沿岸線をすべての縮尺で表示する必要があるため、設定を [No Limit] (制限なし) のままにします。

9. ロードするファイルのパラメータをすべて設定したら、[OK] をクリックします。

   ファイルがロードされ、フィリピン諸島の沿岸線が表示されます。

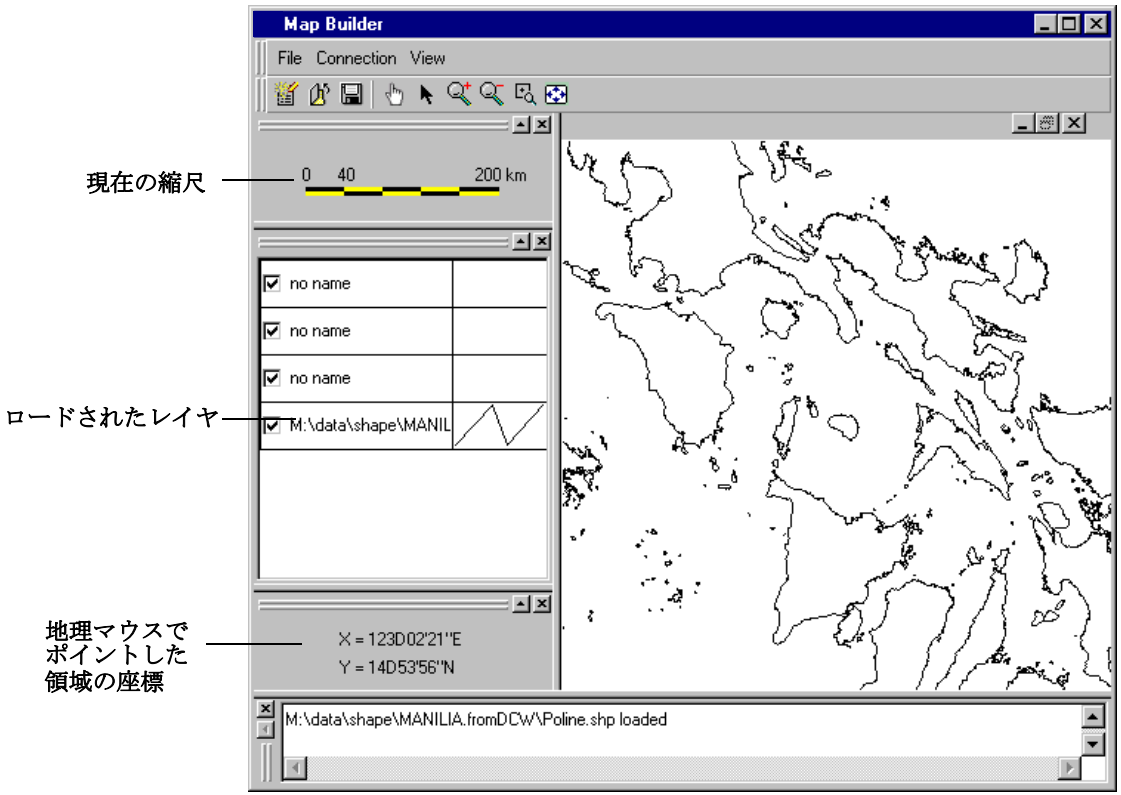


図 2.4 *地図ファイルがロードされた* IBM ILOG Views *マップ・ビルダ*

IBM ILOG Views マップ・ビルダのツールバーを使用する

地図を読み込むと、マップ・ビルダのツールバーに次のコマンドが表示されます。

- [選択 (Selection)] アイコン でバッファのオブジェクトを選択します。
- [拡大 (Zoom In)] アイコン および [縮小 (Zoom Out)] アイコン 。
- [ズーム・ウィンドウ (Zoom window)] アイコンで、ズームする領域に矩形をドラッグできます。
- [内容に合わせる (Show all)] アイコン で、地図全体を表示します。
- [パン (Pan current map)] アイコン で、バッファ・パネル内の地図を移動します。

これらのコマンド使用を終了する場合は、[内容に合わせる] アイコンをクリックして地図全体を表示します。

道路を地図に読み込む

ここでは、レンダラの選択方法と縮尺フィルタの使用方法について説明します。

例えば、前のセクションで使用したバッファに、道路が含まれる `Rdline.shp` ファイルをロードします。

1. ［ファイル (File)］メニューから、［地図ファイルの挿入 (Insert Map)］を選択します。

   バッファが新規作成されてしまうため、[ロード] は選択しないでください。

2. `<shapedir>` を探して、例えば、`Rdline.shp` ファイルを開きます。

3. [パラメータ (Source file parameters)] ウィンドウの [レンダリング (Rendering)] ページで、地図の道路が赤色で表示されるように線の色を変更します。

   変更できる 3 つのレンダリング・パラメータがあります。オブジェクトの塗りつぶし色、線の色、マーカー・シンボルです。地図のロード時に変更できるのは、適用可能なパラメータだけです。この場合は、線のレンダリング・スタイルのみ変更できます。

4. [ソース投影図法 (Source projection)] を [Geographic] (地理) に設定します。

5. デフォルトのマップ・アダプタはそのままにします。

6. [アトリビュートのロード (Load attributes)] チェック・ボックスをオフにします。

7. [スケール・コントロール (Scale control)] ページで、[小縮尺限度 (Small scale limit)] を 1/5,000,000 に設定します。これで、1/5,000,000 よりも大きな縮尺 (たとえば 1/2,500,000) を選択すると道路が表示されるようになります。

8. [OK] をクリックします。

次の手順で設定をテストします。

1. [内容に合わせる (Show all)] アイコン  をクリックして、地図全体を表示します。
2. 地図を拡大して、縮尺が 1/5,000,000 よりも大きい場合に道路が表示されることを確認します。

町の読み込み

このセクションでは、アトリビュートの付加とその表示方法について説明します。

ここで、地図に町を読み込みます。この情報は、`Pppoint.shp` ファイルにあります。

1. ［ファイル (File)］メニューから、［地図ファイルの挿入 (Insert map file)］を選択します。
2. `<shapedir>` を探して、例えば、`Pppoint.shp` ファイルを開きます。

   [パラメータ (Source file parameters)] ウィンドウが表示されます。
3. [点 (Point)] レンダリング・スタイルを [円 (Circle)] に変更します。
4. [ソース投影図法 (Source projection)] ページで、ソース投影図法を Geographic に設定します。
5. [アトリビュート (Attributes)] ページで、[アトリビュートのロード (Load attributes)] チェック・ボックスをオンにし、地図にアトリビュートを読み込みます。

   町の名前がアトリビュートに含まれているため、この情報を地図に表示する必要があります。
6. [スケール・コントロール (Scale control)] ページで、[小縮尺限度 (Small scale limit)] を 1/2,500,000 に設定します。町の記号を表示するためには、この縮尺まで拡大する必要があります。
7. [OK] をクリックします。

   町が地図に小さな円で表示されます。

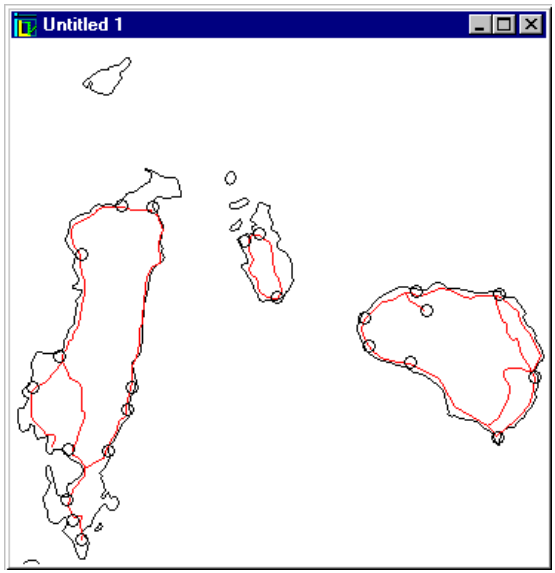


図 2.5　地図に読み込まれた町

付加されたアトリビュートの表示

ここで、付加した町のアトリビュートが表示されるかどうかを確認できます。

1. [表示 (View)] メニューから [アトリビュート (Attributes)] を選択して、[アトリビュート] ウィンドウを開きます。
2. [選択 (Selection)] アイコン をクリックして、バッファの選択モードをアクティブにします。
3. 地図上の町をいくつか選択します。

   ロードしたファイルに情報のある、町のアトリビュートが表示されます。町の名前がファイル情報に含まれている場合、[PPPTNAME] プロパティ・フィールドの右の [値] フィールドに表示されます。

   図 2.6 では、アルカンタラの町が選択され、町の名前が [PPPTNAME] 値フィールドに表示されています。

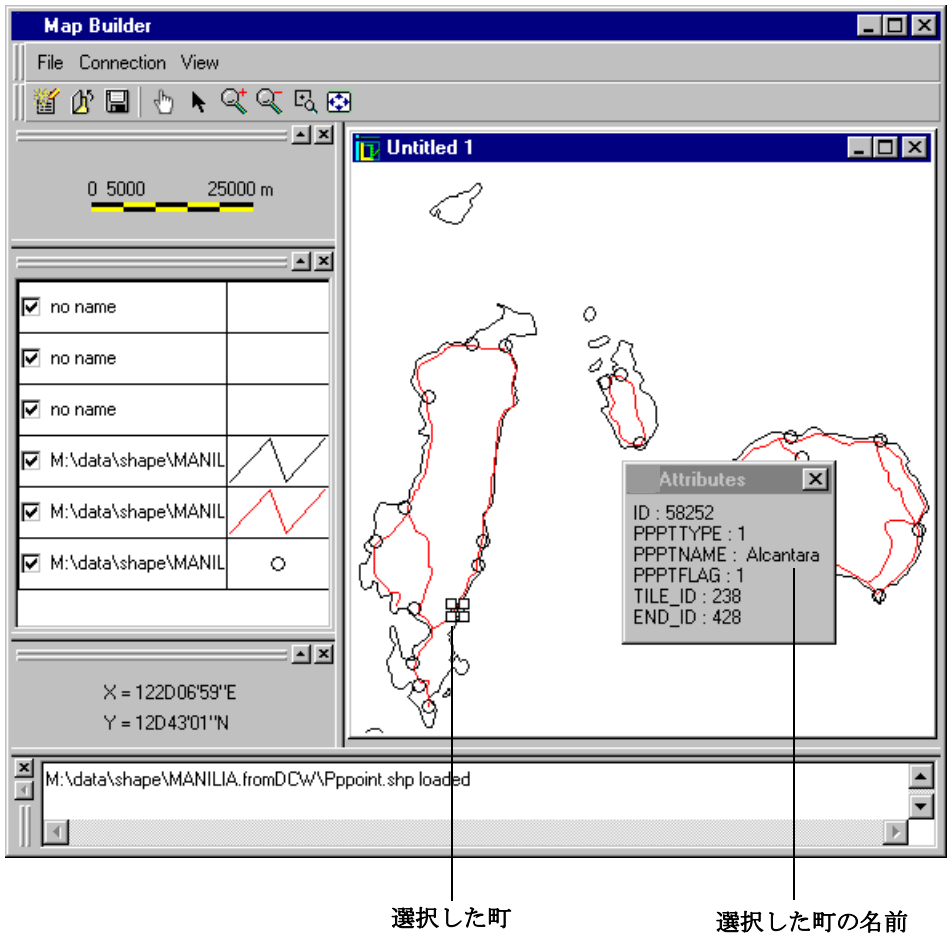


図2.6 [アトリビュート] ウィンドウ

大きな町を地図に読み込む

このセクションでは、レンダラ・プロパティの設定、レイヤの編集、IBM® ILOG® Views 地図の保存、などの方法について説明します。

大きな町の情報は、`<shapedir>` ディレクトリの `Pppoly.shp` ファイルにあります。

1. ［ファイル (File)］メニューから、［地図ファイルの挿入 (Insert Map)］を選択します。
2. 例えば、`Pppoly.shp` ファイルを探して開きます。

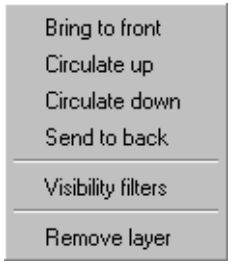
3. [レンダリング (Rendering)] ページで、塗りつぶしの色の値を濃い灰色に設定します。そのためには、色フィールドをクリックしてから灰色を選択します。
4. 線の色を、たとえば黒に設定します。

> ***メモ:*** *境界線の描画が不要な場合は、[色] コンボ・ボックスで [None] (なし) を選択します。*

5. [アトリビュート (Attributes)] ページで、[アトリビュートのロード (Load attributes)] チェック・ボックスのチェックをオフにします。
6. [OK] をクリックします。

レイヤの編集

レイヤの順序とビジビリティ・フィルタは変更できます。変更する場合は、[凡例 (Legend)] ウィンドウのレイヤ名を右クリックします。次のメニューが表示されます。



レイヤの位置を変更する

[凡例 (Legend)] ウィンドウの [レイヤ] リスト・ボックスのもっとも下にあるレイヤが、ビューで最初に表示されるレイヤです。リスト・ボックスの下から 2 番目のレイヤは、2 番目に表示されるレイヤで、その内容は先頭のレイヤのオブジェクトで隠れる場合があります。バッファにある地図の場合、道路レイヤは大きな町のレイヤの下にあります。

道路を表示する方法

1. 地図の縮尺を 1/500,000 に設定します。

   この縮尺ならば、境界線が濃い灰色で塗りつぶされた大きな町を容易に見つけることができます

2. [パン] アイコン で地図をパンして、大きな町を見つけます。

   赤い道路の描線を覆う、灰色の大きな町が表示されています。

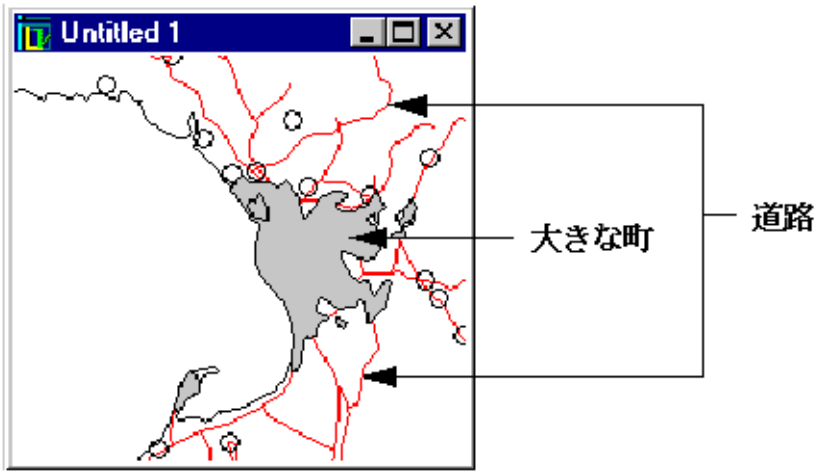


図 2.7　道路レイヤを隠す大きな町のレイヤ

次のように、道路レイヤを大きな町のレイヤの上に移動します。

1. [凡例 (Legend)] ウィンドウで、大きな町のオブジェクトが含まれるレイヤを右クリックします。
2. [レイヤのポップアップ・メニューから [一番後ろへ移動 (Send to back)] オプションを選択して、そのレイヤをリストの一番上へ移動します。

以上で道路レイヤが大きな町のレイヤの一番上に置かれ、大きな町の境界線内に表示されるようになりました。

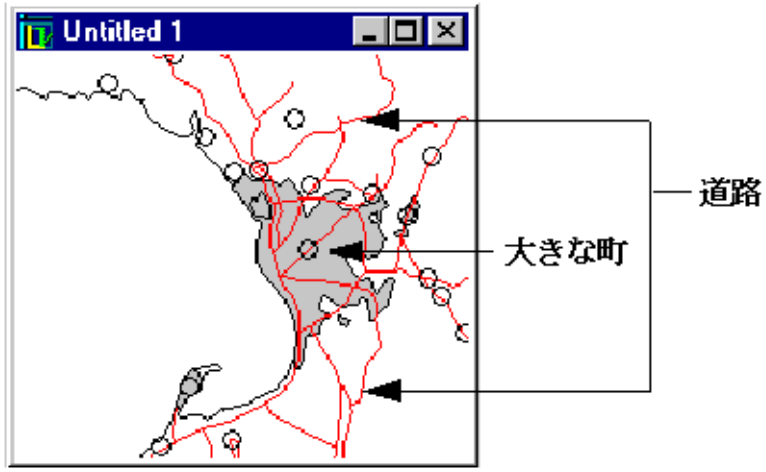


図2.8　道路レイヤの下に位置が変更された大きな町のレイヤ

スケール・コントロール・パラメータ

地図作成時に、特定タイプの情報を表示するレイヤが、特定の縮尺でのみ表示されるようにする場合があります。たとえば、道路インフラストラクチャについて、地図を特定の縮尺に拡大したときだけ二次道路を表示したい場合があります。[レイヤのポップアップ・メニューから [ビジビリティ・フィルタ (Visibility filters)] オプションを選択することで、これが指定できます。

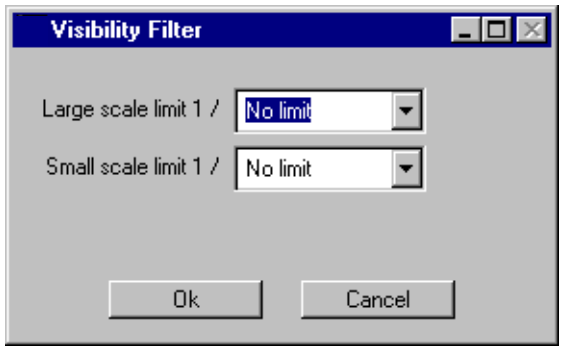


図2.9　[スケール・ビジビリティ] パネル

ビジビリティ・フィルタには、2 つの限度値を設定できることが特徴です。[大縮尺限度 (Large scale limit)] と [小縮尺限度 (Small scale limit)] です。これら 2 つの限度内縮尺で、指定のレイヤが表示されます。たとえば、[大縮尺限度] を 100,000、[小縮尺限度] を 500,000 に設定すると、指定のレイヤは 1/100,000 から 1/500,000 までの縮尺でのみ表示されます。

ファイルの保存

ここで、ファイルを保存します。IBM® ILOG® Views マップ・ビルダでは、.ilv 形式のみでファイルを保存します。ファイル名には .ilv 拡張子が自動的に追加されます。これで、このフォーマットのファイルであることが容易に特定できます。

ファイルを保存するには、次の手順に従ってください。

1. [ファイル (File)] メニューから [名前を付けて保存 (Save as)] を選択します。
2. ファイル保存先のディレクトリを指定します。
3. Philippines.ilv ファイルに名前を付けます。
4. [保存] をクリックします。

ロード・オン・デマンドで地図を作成する

> ***メモ:*** *この例を開始する前に、必要なファイルを* <wacodir> *ディレクトリおよび、* <cadrgdir> *ディレクトリにダウンロードしたことを確認してください。*

この例では、ロード・オン・デマンド機能を持つ地図を作成します。サンプルの地理領域は、米国合衆国テキサス州のウェイコ地域です。いくつかのファイルをロードします。

- 最初の地図ファイル waco.ilv は、ベクトル・ファイルです。このファイルには、地域の町と道路が含まれます。この地図ファイルはあらゆる縮尺で表示され、地図の基本構造を提供します。
- 2 番目のファイルは、ロード・オン・デマンド機能を表示します。サンプルのテキサス州ウェイコ地域をスキャンした CADRG (Compressed ARC Digitized Raster Graphics) 地図ファイルを使用します。このファイル形式は、特にロード・オン・デマンド機構に適したもので、Views Maps でもサポートされています。情報は、設定した縮尺に基づいて表示されます。
- 3 番目のファイルは非常に大きなファイルですが、ロード・オン・デマンドによる地図の操作では優れた応答時間を示します。

CADRG フォーマットの詳細については、105 ページの *CADRG ファイル・リーダ* を参照してください。

この例では、以下の方法を紹介します。

- ロード・オン・デマンド機能を使って地図を作成する方法

- ロード・オン・デマンド機能によって、IBM® ILOG® Views ファイルのパフォーマンスを向上させる方法
- ロード・オン・デマンドのパフォーマンス

基本構造マップと CADRG ファイルのロード

この例に必要なファイルをロードするため、現在開いているバッファを閉じます。次に、以下の手順で `waco.ivl` ファイルをロードします。

1. [ファイル (File)] メニューから [ファイルのロード (Load file)] を選択します。
2. 次の場所を探します。`<wacodir>` へ移動したら `waco.ivl` ファイルを開きます。

   ウェイコ地域がバッファに表示されます。

事前に作成されたウェイコ地図に含まれる情報がこれらのレイヤで表示される

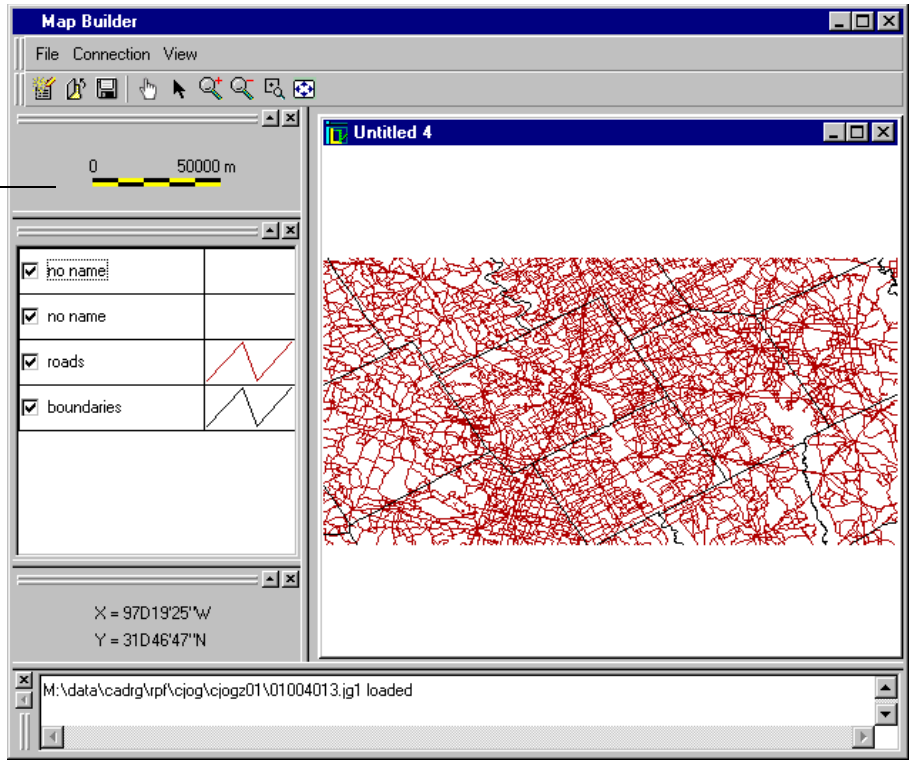


図 2.10　バッファに読み込まれたテキサス州ウェイコの地図

次にロードするファイルは、ロード・オン・デマンド・モードでスキャンされた CADRG 地図ファイルです。

1. ［ファイル (File)］メニューから、［地図ファイルの挿入 (Insert Map)］を選択します。
2. 次の場所を探します。`<cadrgdir>/cadrg/rpf/cjog/cjogz01` へ移動したら、例えば `01004013.jgl` ファイルを開きます。
3. [パラメータ (Source file parameters)] ウィンドウで、[ロード・オン・デマンド (Load On Demand)] ページの [ロード・オン・デマンド] を選択します。

   この設定を行うと、「ズーム」関連、「パン」、「内容に合わせる」コマンドを使用する際にユーザが要求する項目に基づいて、地図全体を構成する CARDRG ディレクトリのさまざまな部分がロードされます。ロード・オン・デマンド機能は、IBM ILOG Views マップ・ビルダのロード・オン・デマンド・パラメータで設定します。
4. [スケール・コントロール (Scale control)] ページで、[大縮尺限度 (Large scale limit)] を 1/100,000 、[小縮尺限度 (Small scale limit)] を 1/500,000 に設定します。
5. [OK] をクリックします。

CADRG レイヤは、その縮尺がこのビジビリティ・フィルタの限度内になるまで表示されません。[内容に合わせる] アイコンをクリックして地図全体を表示してから、CADRG レイヤが表示されるまで拡大します。このレイヤが表示されると、`waco.ilv` を含むレイヤの上に CADRG 地図が表示されます。CADRG レイヤの上に道路を表示するには、レイヤの順番を変更する必要があります。

レイヤの位置を変更する

ここでレイヤの位置を変更して、ベクトル地図のレイヤがスキャンした地図の上に表示されるようにします。

1. [凡例 (Legend)] メニューで、CADRG レイヤを右クリックして、レイヤ・ビジビリティのメニューを開きます。
2. [背後へ移動 (Bring to Back)] を選択します。これで CADRG レイヤが背後に移動され、道路が表示されます。

地図の重ね合わせ

テキサス州ウェイコ地域の道路と町は、完全に重ね合わされています。それを確かめるため、マップ・ビルダのメイン・ウィンドウの左 ([凡例] パネル) に表示される、[roads] (道路レイヤ) のチェック・ボックスをオフにして、それを非表示にします。オフにすると同時に、ベクトル地図 (基本構造地図) の道路が下になります。

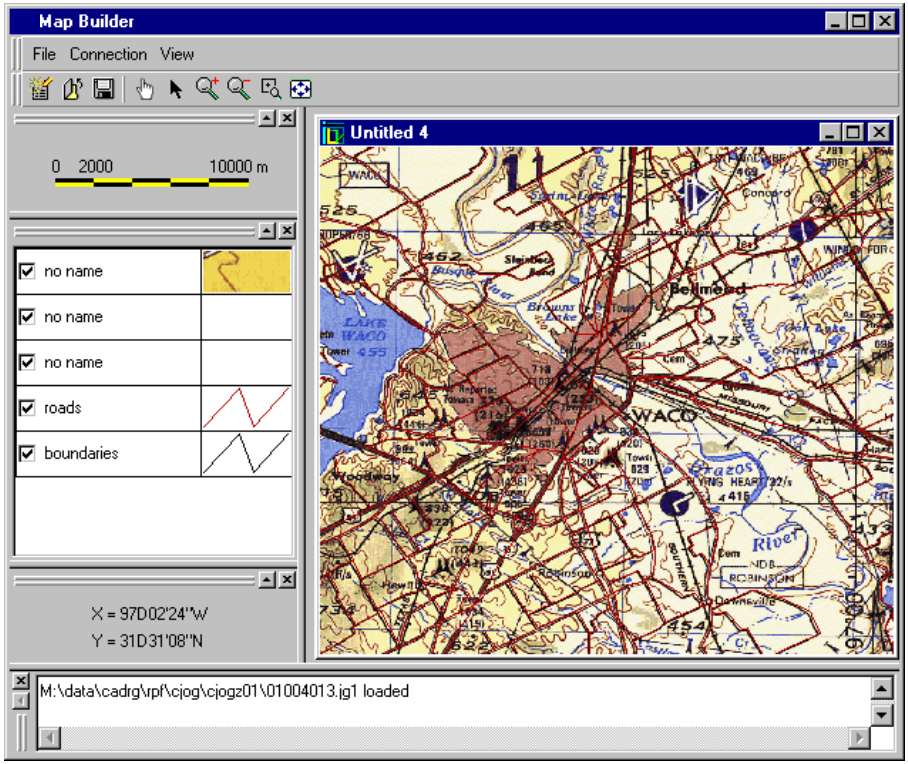


図 2.11 *地図の重ね合わせ*

異なった縮尺でロード・オン・デマンドを監視する

ロード・オン・デマンド機能を監視するため、圧縮形式でディスクに格納された数メガバイトのデータで構成されるファイルをロードします。ロード・オン・デマンド機構によって、長い応答時間を要することなく地図全体の操作ができます。

1. ［ファイル (File)］メニューから、［地図ファイルの挿入 (Insert map file)］を選択します。
2. 次の場所を探します。`<cadrgdir>/cadrg/rpf/cltm50/ct50z01` へ移動したら、例えば `Oqen2013.tl1` ファイルを開きます。

   ロード・オン・デマンドで表示される最初の地域が、これによって単純に指示されます。
3. [パラメータ (Source file parameter)] ウィンドウで、[ロード・オン・デマンド (Load On Demand)] ページの [ロード・オン・デマンド] を選択します。
4. [スケール・コントール (Scale control)] ページで、[小縮尺限度 (Small scale limit)] を 100,000 に設定します。

5. [OK] をクリックします。

   ロードされたレイヤが、描画スタックの一番上に置かれます。*レイヤの位置を変更する*で示した手順によって、そのレイヤを背後へ移動します。

地図の縮尺を変更することによって、または異なる地図のセクションを表示するビューで地図を移動させることによって、ロード・オン・デマンド機能が観察できるようになります。そのために、ツールバーの「内容に合わせる」、「ズーム」関連、「パン」などのコマンドを実行します。ロード・オン・デマンドが迅速に応答し、見たい情報がほとんど遅延なく表示されます。

Oracle Spatial で地図を作成する

> ***メモ:*** *この例を開始する前に、ESRI 形状ファイルが* `<shapedir>` *ディレクトリにダウンロードされていることを確認します。*

この例では、Oracle Spatial データベースに含まれるデータを表示するために、IBM® ILOG® Views Maps を使用する方法について説明します。この例の実行にあたっては、次が必要です。

- マシンにインストールされ、アクセス可能な Oracle 8.1.5 (または 8.1.6) サーバ
- マシンにインストールされた Oracle 8.1.5 (または 8.1.6) クライアント
- IBM ILOG DB リンク 4.1
- IBM ILOG Views Maps からアクセスできる、`<installdir>/samples/maps/oracle` ディレクトリの Oracle サンプル

この例では、以下の方法を示します。

- Oracle Spatial データベースに保存する単純なレイヤの作成
- マップ・ビルダによるそのレイヤの表示
- マップ・ビルダのロード・オン・デマンドによるそのレイヤの表示

Oracle Spatial データベースのレイヤの作成

このセクションでは、サンプルで使用するデータベースを作成します。データベースの作成には、Oracle Sample を使用します。

Oracle Sample がコンパイルされていない場合、`<installdir>/samples/maps/oracle/index.html` の指示に従ってコンパイルします。

サンプル・データベースの作成は、次の手順に従います。

1. Oracle Sample アプリケーションを実行します。
2. ［ファイル (File)］メニューから、［地図ファイルのロード (Insert map file)］を選択します。
3. 表示されるテキスト・フィールドに、データ・ファイル名を入力します。この例では、形状ファイル・データを使用します。`<shapedir>/world.shp` を使用します。

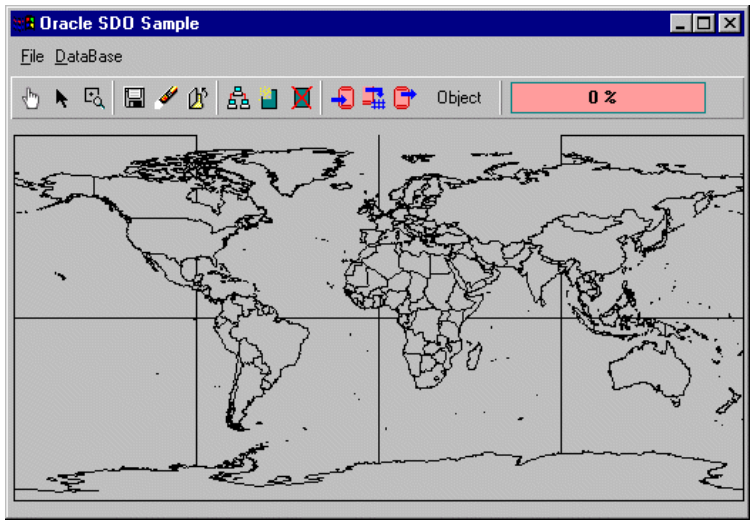


図 2.12　ロードされた形状ファイルの SDO サンプル

4. データベースのレイヤを作成するために、サンプルのツールバーにあるデータベース・モードのトグル表示が [Object] になっていることを確認します。次に、[データベース (DataBase)] メニューから [レイヤの作成 (Create SDD layer)] を選択します。データベースに接続されていない場合は、デフォルトの接続パネルが表示されて接続パラメータの入力を求められます。

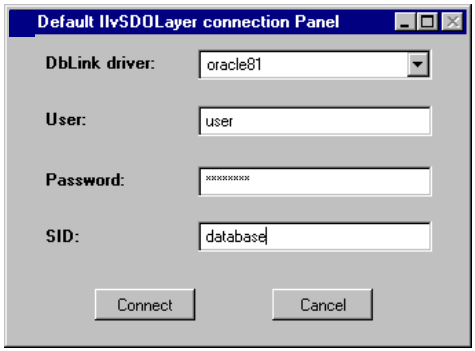


図 2.13 *デフォルトのデータベース接続パネル*

オブジェクト・リレーショナル・モデルを使用するので、DBLink Driver コンボ・ボックスには、［oracle81］または ［oracle8］とドライバ名を入力します。

その他のドライバとしては、［oracle73］のみがリレーショナル・モデル用として有効です。

データベース・アカウントと SID については、データベース管理者にお問い合わせください。

5. データベースに接続されたら、作成するレイヤ名を入力します。このサンプルの場合、レイヤ名は `WORLD` です。
6. [OK] をクリックします。

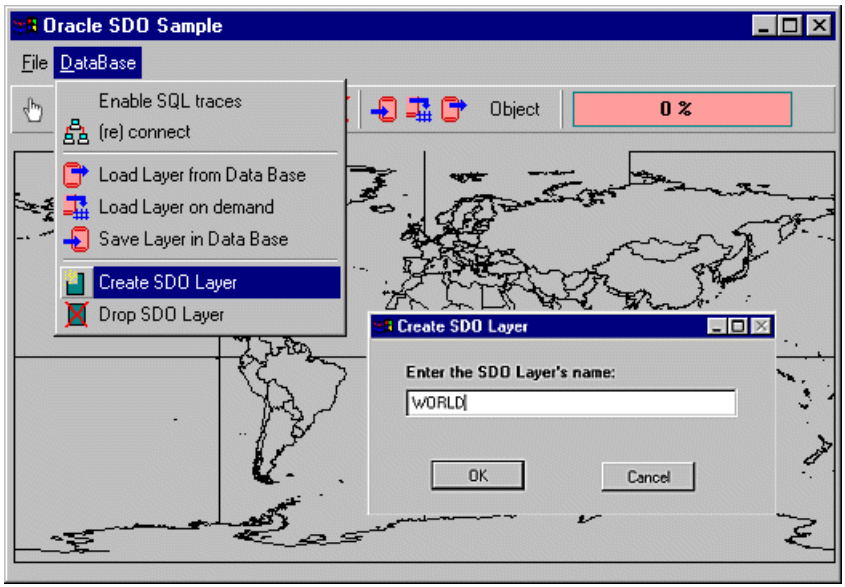


図 2.14　SDO レイヤの作成

7. [データベース (DataBase)] メニューの [データベースにレイヤを保存 (Save Layer in DataBase)] オプションを選択すると、このレイヤのデータがデータベースに保存されます。リストから [WORLD.GEOMETRY] レイヤを選択して、保存を確認します。

8. データが書き込まれると、タイリング・レベルの指定を求めるダイアログ・ボックスが表示されます。このサンプルには十分なレベルの [8] と入力します。この値は Oracle Spatial 変数で、実行されるデータのタイリング法を制御します。詳細については、Oracle Spatial のマニュアルを参照してください。

この章の復習で取り組む練習問題のデータが、これで使用できるようになります。

マップ・ビルダによる Oracle Spatial レイヤの表示

このセクションでは、Oracle データベースへの単純なリクエストを実行します。データベースへの接続と問い合わせ方法、データ・ウィンドウの表示方法について説明します。前のセクションでデータベースにロードされたデータを使用しますが、Oracle Spatial データベースにある他の任意のデータ・セットが使用できます。

1. IBM® ILOG® Views マップ・ビルダを実行します。

2. データベースに接続するには、空のバッファを作成する必要があります。ツールバーの [新規地図 (New map)] アイコンをクリックするか、[ファイル (File)] メニューの [新規地図バッファ (New map buffer)] を選択します。

3. 地図作成で使用するバッファを選択します。
4. [接続 (Connection)] メニューで、[Oracle オブジェクト・モデル (Oracle Object Model)] を選択します。デフォルトの接続パネルが表示されます。
5. データベース接続パラメータを入力します。

   ファイルからデータを読み込むときに使用する場合と同様の、[パラメータ (Source file parameters)] ウィンドウが表示されます。このウィンドウによって、Oracle 関連パラメータが [パラメータ] ウィンドウに追加されます。
6. [レンダリング (Rendering)] ページで、使用する色を選択します。
7. [ターゲット投影図法 (Destination projection)] ページでは、格納するデータの投影図法を選択します。サンプルの [WORLD] 地図の場合、投影図法は [Geographic] です。
8. [ターゲット・アダプタ (Destination adapter)] ページでは、デフォルトのパラメータ (1m) をそのままにします。
9. [ロード・オン・デマンド (Load On Demand)] ページでは、次の 2 つのモードでリクエストを実行できます。

   - 1 つは対象領域 (AOI) を指定する方法です：指定した AOI のデータが現在の地図バッファに読み込まれます。
   - もう 1 つはロード・オン・デマンドを使用する方法です。

   このセクションでは、[プレビュー (Preview)] モードを使用します。[プレビュー] トグルをクリックしてオンにすると、リクエスト・データの配列が選択できます。すべてのフィールドを 0 (ゼロ) のままにすると、レイヤ全体がロードされます。

> ***メモ:*** *このサンプルでは、小さなデータ・セットを使用します。大きなデータ・セットで AOI によるリクエストを実行する場合は、選択した AOI が十分に小さいかどうか、読み込まれるデータ量が妥当かどうかに注意する必要があります。*

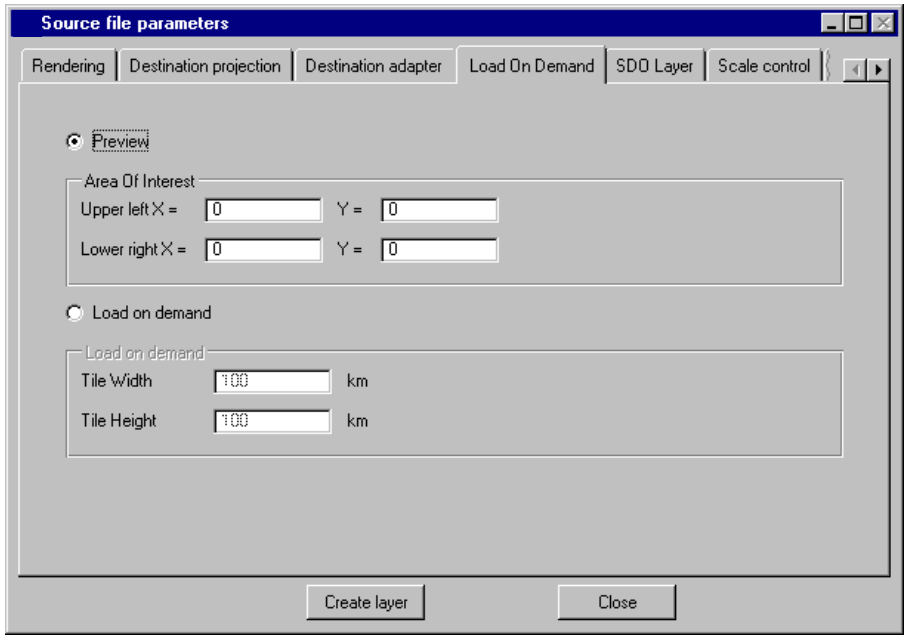


図2.15　Oracle 接続パネルの [ロード・オン・デマンド] ページ

10. [SDO レイヤ (SDO Layer)] ページには、データベースで使用できるレイヤのすべてが一覧表示されます。レイヤ名は、レイヤが含まれる表名、ピリオド '.'、データのジオメトリが含まれる表の列名、で構成されます。このサンプルでは、少なくとも WORLD.GEOMETRY という名前のレイヤが表示されます。このレイヤを選択します。

11. [スケール・コントロール (Scale control)] ページでは、作成したレイヤにビジビリティ・フィルタを設定できます。ただし、このサンプルではスケール・ビジビリティ・フィルタを設定する必要はありません。

12. [アトリビュート (Attributes)] ページでは、オブジェクトに関連するアトリビュートをロードできます。アトリビュートのロードを選択すると、ジオメトリではない列のすべてがアトリビュートとして、作成したグラフィック・オブジェクトに付加されます。詳細については、Oracle リーダのリファレンス・マニュアルを参照してください。ただし、このサンプルではオブジェクトのアトリビュートをロードする必要はありません。

13. さまざまなページでオプションを選択した後で、[レイヤの作成 (Create Layer)] ボタンをクリックします。

データ全体を含むレイヤが、これで作成されました。

ロード・オン・デマンドで Oracle Spatial レイヤを表示する

このセクションでは、前のセクションと同じレイヤを、ロード・オン・デマンド機能によってロードします。

1. 前のセクションの 1 から 7 のステップを繰り返します。
2. [ロード・オン・デマンド] ページで [ロード・オン・デマンド] を選択し、タイル・サイズを指定します。

   このサンプルでは、WORLD レイヤの有効範囲は 1 X 1 度です。たとえば、500km をタイル・サイズとして指定できます。
3. [SDO レイヤ (SDO Layer)] ページには、データベースで使用できるレイヤのすべてが一覧表示されます。レイヤ名は、レイヤが含まれる表名、ピリオド '.'、データのジオメトリが含まれる表の列名、で構成されます。このサンプルでは、少なくとも WORLD.GEOMETRY という名前のレイヤが表示されます。このレイヤを選択します。
4. [スケール・コントロール (Scale control)] ページでは、作成したレイヤにビジビリティ・フィルタを設定できます。ただし、このサンプルではスケール・ビジビリティ・フィルタを設定する必要はありません。
5. [アトリビュート (Attributes)] ページでは、オブジェクトに関連するアトリビュートをロードできます。アトリビュートのロードを選択すると、ジオメトリではない列のすべてがアトリビュートとして、作成したグラフィック・オブジェクトに付加されます。詳細については、Oracle リーダのリファレンス・マニュアルを参照してください。ただし、このサンプルではオブジェクトのアトリビュートをロードする必要はありません。
6. さまざまなページでオプションを選択した後で、[レイヤの作成] ボタンをクリックします。

   データ全体を含むレイヤが、これで作成されました。

情報の永続性をテストする

地図情報の永続性をテストするには、次の手順に従います。

7. ファイルを保存するために、[ファイル (File)] メニューから [名前を付けて保存 (Save as)] を選択します。
8. 次にバッファを閉じるために、[ファイル (File)] メニューから [閉じる (Close)] を選択します。
9. ファイルを開くには、[ファイル] メニューの [ロード (Load file)] を選択します。

レイヤの復元に特定の情報 (たとえば、Oracle Spatial レイヤ、ログイン用のパスワードなど) が必要なレイヤがいくつかある場合は、ファイルを開くときに情報の提供を求められます。

章

3

IBM ILOG Views Maps リーダ・フレームワーク

この章では、IBM® ILOG® Views Maps リーダ・フレームワークを使って、地図データを表示する IBM ILOG Views アプリケーションを作成する方法について説明します。

IBM ILOG Views Maps では、航空写真、デジタル地図、多角形、線およびラベルなどの地図データをインポートしたり、`IlvManager` や `IlvGrapher` のような `IlvManager` のサブクラス内にある `IlvGraphic` オブジェクト群を使って、このようなデータ表現ができます。IBM ILOG Viewss Graphics Framework には、定義済みの豊富なグラフィック・オブジェクトが含まれており、それらのオブジェクトを使って、*地図機能*を表現することができます。このグラフィック・オブジェクト群は、コーディングによって、または Prototypes パッケージ付きの IBM ILOG Views Studio を使って強化することができます。

地図データは、さまざまな形式のある多様なデータ・ソースから派生しているため、IBM ILOG Views Maps では、元の形式に関わらず、データ表現のグラフィック・オブジェクトが容易に生成できる、高レベル・リーダ・フレームワークを提供しています。

この章は、次の 2 つの部分から構成されています。

- 最初の部分では、IBM ILOG Views で地図データを表現する方法について説明します。ロードからグラフィカル・レンダリングまでプロセス全体について扱います。ここでは、以下のトピックを取り上げます。
  - *地図作成用のクラス : 概要* では、リーダ・フレームワークの主要クラスを紹介します。
  - *地図機能* では、地図機能について説明します。地図機能は、ソース・ファイルから読み込んだときに地図データを表示するオブジェクトです。
  - *レンダラ* では、レンダラと、レンダラの拡張方法について説明します。レンダラは、地図機能を IBM ILOG Views グラフィック・オブジェクトに変換するために、使用するオブジェクトです。
  - *機能イテレータ* では、機能イテレータと簡単なリーダのプログラミング法について説明します。機能イテレータは、地図データの読み込みに使用するオブジェクトです。
  - *ターゲット投影図法を選択する* では、地図投影図法をマネージャに関連付ける方法について説明します。
  - *マップ・ローダー* では、`.ilv` ファイルのロード方法について説明し、マップ・ローダーについて説明します。マップ・ローダーは、定義済み形式の地図データを IBM ILOG Views に非常に簡単にインポートできるクラスで、その拡張方法についても紹介します。
- 後半では、以下のトピックを取り上げます。
  - *縮尺フィルタ* では、縮尺フィルタを使って、地図の縮尺セットに応じて情報を表示したり、非表示にする方法について説明します。

地図作成用のクラス : 概要

IBM® ILOG® Views Maps には、さまざまな地図データ・ソース (ファイル、データベース、マップ・サーバなど) からデータを読み込んだり、地図機能を作成する、さらにレンダラを使ってその機能を IBM ILOG Views グラフィック・オブジェクトに変換したり、それらのオブジェクトを既存の地図に配置する際に使用可能なクラスが用意されています。

- `IlvMapFeature` クラスは地図機能のアイテム・クラスです。地図機能は、ソース・ファイルから読み込まれた地図データを表すオブジェクトです。道路の一部、俯瞰イメージ、丘の頂上、または数値地形モデル (DTM) などになります。それぞれの地図機能には、次に示す 3 つの主な情報フィールドがあります。*ジオメトリ*、ジオメトリを表現する *投影図法*、およびそのアトリビュートです。たとえば、地図機能が町の場合、そのアトリビュートは町の名前と居住者数になります。地図機能は、アプリケーションのグラフィック表現方法とはまったく独立しています。したがって、丘の頂上を示すポイントなどは、十字、円、

またはアイコンなどのさまざまなグラフィック・オブジェクトで、正確に表現した方がわかりやすくなります。このクラスの詳細については、55 ページの*機能イテレータ*を参照してください。

- `IlvFeatureRenderer` 抽象クラスは、地図機能をグラフィック・オブジェクトに変換し、`IlvManager` に追加するために使用されます。機能レンダラを使うと、特定の地図機能に関連付け、必要に応じてターゲット・アプリケーションの投影図法に、そのジオメトリを再投影するグラフィック表現を指定できます。
- `IlvMapFeatureIterator` 抽象クラスは、リーダの共通インターフェースです。この抽象クラスを実装するすべてのクラスは、元の形式に関わらず、地図データを読み込む際に使用できます。このインターフェースの詳細については、55 ページの*機能イテレータ*を参照してください。IBM ILOG Views Maps には、このインターフェースをすべて実装するさまざまな定義済みリーダが用意されています。これらのリーダについては、5 章*定義済みリーダ*で説明します。
- 使用パッケージ `projection` の `IlvProjection` クラスは、多数ある定義済み投影図法クラスのベース・クラスです。投影図法については、64 ページの*ターゲット投影図法を選択する*で紹介します。投影図法の詳細については、6 章*地図投影図法*を参照してください。

次の図は、地図データを IBM ILOG Views にロードするプロセスを示したものです。

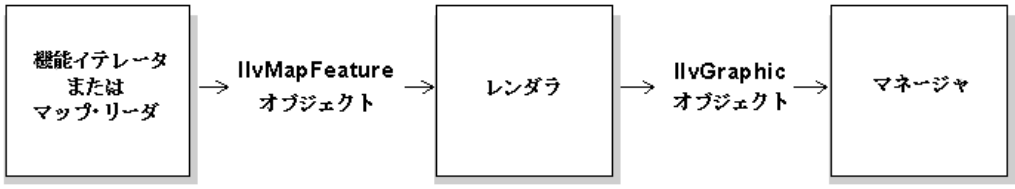


図 3.1 *外部地図データを IBM ILOG Views にロードする*

- パッケージ `format` の `IlvMapLoader` クラスは、ILOG Views Maps に用意されているすべての定義済みリーダで、この全プロセスを自動的に実行します。定義済みリーダの詳細については、5 章*定義済みリーダ*を参照してください。

地図機能

IBM® ILOG® Views に地図を読み込む際、地図データは `IlvMapFeature` オブジェクトとしてロードされます。地図機能は、ソース・ファイルから読み込まれた地図データを表示するオブジェクトです。道路の一部、俯瞰イメージ、丘の頂上、また

は数値地形モデル (DTM) などになります。地図機能には、以下の情報が含まれています。

- 機能のジオメトリ
- アトリビュート
- ジオメトリを表現する投影図法

  投影図法については、このセクションでは扱いません。詳細は、*ターゲット投影図法を選択すると* 6 章*地図投影図法*を参照してください。

地図機能のジオメトリ

各地図機能にはジオメトリがあります。地図機能のジオメトリとは、形状や位置に関する情報です。

IBM® ILOG® Views Maps では、地図機能のジオメトリは、`IlvMapGeometry` クラスで定義されます。パッケージ `geometry` では、GIS (Geographic Information Systems) 間の相互運用性を確保するために、OpenGIS Consortium が定めた「Simple Map Features」ジオメトリで作られた、さまざまな定義済みジオメトリが提供されています。ただし、このパッケージのクラスは、機能性の面でこのモデルを厳密に遵守しているわけではありません。このパッケージは、単純な機能を提供しており、主に描画を目的としています。しかし、このクラスを使用すると、Oracle Spatial などのマップ・サーバのデータを変換する際に非常に便利です。

また、このパッケージには、イメージ、ラスターおよびテキストをより簡単に処理するための追加ジオメトリが含まれており、新しいジオメトリで拡張することもできます。

地図機能のアトリビュート

各地図機能にはアトリビュートもあります。地図機能が町の場合、そのアトリビュートはその町の名前、または居住者数になります。たとえば、アトリビュートはグラフィカル・レンダリングに使用できます。51 ページの *色付き線のレンダラを作成する*では、地図上で等高線を表すポリラインの色は、標高アトリビュートで定義されます。

アトリビュートは、クラス `IlvFeatureAttribute` に属します。それらのアトリビュートは、次に示す 2 つのクラスに格納されます。

- `IlvFeatureAttributeInfo`、名前、タイプ、必須条件または任意文字などのアトリビュート・プロパティを定義します。
- `IlvFeatureAttributeProperty`、これらのアトリビュートの値を含みます。

以下のコード・サンプルでは、`IlvMapFeature` オブジェクトのアトリビュートをリストアップし、画面に表示します。

```
void
dumpAttributes(const IlvMapFeature* feature)
{
    const IlvFeatureAttributeProperty* attributes =
        feature->getAttributes();
    if(!attributes)
        return;
    const IlvFeatureAttributeInfo* info =
        attributes->getInfo();

    if(info) {
        IlvUInt count;

        count = info->getAttributesCount();
        for(IlvUInt i = 0; i < count; i++) {

            const char* name = info->getAttributeName(i);

            const IlvMapClassInfo* clsinfo = info->getAttributeClass(i);
            const IlvFeatureAttribute* fa = attributes->getAttribute(i);
            if(clsinfo->isSubtypeOf(IlvStringAttribute::ClassInfo())) {
                const char *str = ((IlvStringAttribute*)fa)->getValue();
                    IlvPrint("%s %s", name ? name : "", str ? str : "");
            } else if(clsinfo->isSubtypeOf(IlvIntegerAttribute::ClassInfo())){
                int in = ((IlvIntegerAttribute*)fa)->getValue();
                IlvPrint("%s %d", name ? name : "", in);
            } else if(clsinfo->isSubtypeOf(IlvDoubleAttribute::ClassInfo())){
                double dbl = ((IlvDoubleAttribute*)fa)->getValue();
                IlvPrint("%s %g", name ? name : "", dbl);
            } else if(clsinfo->isSubtypeOf(IlvBooleanAttribute::ClassInfo())){
                IlvBoolean bo = ((IlvBooleanAttribute*)fa)->getValue();
                IlvPrint("%s %s", name ? name : "",
                        bo ? "true" : "false");
            }
        }
    }
}
```

整数、小数点値、文字列などを表すかどうかによって、アトリビュートのタイプは異なります。IlvFeatureAttribute クラスのすべての定義済みアトリビュートは、attribute パッケージにあります。

アトリビュートをグラフィック・オブジェクトに付加する

IBM® ILOG® Views で、クラス IlvNamedProperty を使って、プロパティを IlvGraphic オブジェクトに付加できます。これで、.ilv ファイルのプロパティを関連オブジェクトと一緒に保存できます。

IlvFeatureAttributeProperty クラスは、IlvNamedProperty クラスから継承した地図機能のすべてのアトリビュートを格納し、あらゆるグラフィック・オブジェクトに付加できます。

次のコード・サンプルでは、IlvFeatureAttributeProperty オブジェクトを IlvGraphic クラスのオブジェクトに付加します。

```
const IlvFeatureAttributeProperty* attributeProperty;
attributeProperty = feature->getAttributes();
graphic->setNamedProperty(attributeProperty->copy());
```

この例では、アトリビュートのプロパティをコピーしました。これは、ジオメトリやアトリビュートを伴う地図機能が揮発性で、他の地図機能を読み込んだときに失われてしまうためです。地図機能の揮発性についての詳細は、56 ページの *IlvMapFeatureIterator の概要*を参照してください。

以下のコードを使って、グラフィック・オブジェクトに付加したアトリビュートにアクセスできます。

```
IlvNamedProperty* namedProperty;
const IlvSymbol* symbol = IlvFeatureAttributeProperty::GetName();
namedProperty = graphic->getNamedProperty(symbol);
```

*IBM ILOG Views Foundation ユーザ・マニュアル*の「名前付きプロパティ」に説明があるとおり、特別な名前付きプロパティを書くことで、maps パッケージが提供する定義済みの名前付きプロパティを使って保存できない、アプリケーション特有の情報を保存することができます。

レンダラ

このセクションでは、マップ・レンダラを紹介します。以下のトピックから構成されています。

- *レンダラの概要*では、IlvFeatureRenderer 抽象クラスと、レンダラを実装するライブラリのクラスについて説明します。
- *色付き線のレンダラを作成する*では、新しいレンダラの書き方について説明します。
- *レンダラの永続化*では、レンダラに永続性を持たせる方法について説明します。
- *既存レンダラの拡張*では、既存のレンダラの拡張方法について説明します。

レンダラの概要

レンダラとは、地図機能をクラス IlvGraphic や、そのいずれかのサブクラスのグラフィック・オブジェクトに変換する際に使用するオブジェクトです。

レンダラは、IlvFeatureRenderer 抽象クラスを実装する必要があります。特定の地図機能をグラフィック・オブジェクトに変換するために、makeGraphic メソッドを使用できます。

```
IlvGraphic* makeGraphic(const IlvMapFeature& feature,
                        const IlvMapInfo& targetMapInfo,
                        IlvMapsError& status);
```

2 番目の引数 targetMapInfo によって、ターゲット投影図法とアダプタを指定できます。アダプタは、地理座標 (通常は垂直軸が上向きの座標系の浮動小数点値で表現される) を、IlvManager で使用するポイント座標 (垂直軸が下向きの座標系の整数値で表現される) に変換するコンバータです。

IlvMapInfo の投影図法を 0 か、または IlvUnknownProjection のインスタンスに設定すると、ターゲット投影図法が IlvMapFeature のソース投影図法と同じとみなされます。この場合、投影変換は行われません。

投影図法とアダプタについては、64 ページの *ターゲット投影図法を選択する* および 6 章 *地図投影図法* を参照してください。

IBM® ILOG® Views Maps には、ライブラリで使用可能な各ジオメトリ・タイプにデフォルトのレンダラ一式が含まれます。これらのレンダラは、パッケージ rendering にあります。たとえば、IlvDefaultPointRenderer は、ポイントのジオメトリがある地図機能をタイプ IlvMarker のオブジェクトに変換します。また、ライブラリは、タイプ IlvDefaultFeatureRenderer のグローバル・デフォルト・レンダラを提供しています。これを使って、定義済みジオメトリの地図機能を変換することができます。レンダラのアトリビュートの一部は、レンダリング・スタイルの指定によってカスタマイズできます。たとえば、IlvMapLineRenderingStyle は、線の幅、色、線の種類のカスタマイズに使用します。

次のコード・サンプルは、ジオメトリがタイプ IlvMapCurve の地図機能を、緑の曲線に変換する方法を示します。これらのポリラインは、国の境界線の一部などになります。

```
IlvFeatureRenderer* renderer = new IlvDefaultCurveRenderer(_display);

IlvMapLineRenderingStyle *lrs = new IlvMapLineRenderingStyle(_display);

lrs->setForeground("green");
lrs->setLineWidth(2);
renderer->setLineRenderingStyle(lrs);

IlvGraphic* graphic = renderer->makeGraphic(feature, mapInfo, status);

if (graphic) {
    _manager->addObject(graphic);
} else {
    IlvWarning("This renderer can't translate the map feature");
    if(status != IlvMaps::NoError())
        IlvWarning(IlvMaps::GetErrorMessage(status, _display));
}
```

色付き線のレンダラを作成する

このセクションでは、既知名のあるアトリビュートの数値から色付きのポリラインを表示する新しいレンダラの書き方について説明します。

このレンダラのサンプル・ソース・コード一式は、以下のファイルにあります。

```
<installdir>/samples/maps/userman/src/colorLineRenderer.cpp
```

標高線を色別表示すると仮定します。簡単な方法は、`ColorModel` で標高値を使って、色を索引付けする方法です。クラスである ColorModel によって、整数値から RGB カラーへの表示に依存しないマッピングができます。より一般的には、地図機能に関連付けられたいずれかのアトリビュートを使用して、線、多角形またはテキストなどのグラフィック・オブジェクトに色を適用するレンダラ・クラスがあると役に立ちます。

ColorLineRenderer クラスの makeGraphic メソッドは、IlvMapGeneralPath グラフィック・オブジェクトを作成します。

```
IlvGraphic*
ColorLineRenderer::makeGraphic(const IlvMapFeature& feature,
                               const IlvMapInfo& targetMapInfo,
                               IlvMapsError& status) const {

    const IlvMapGeometry* geometry = feature.getGeometry();
    if (!geometry) {
        status = IlvMaps::IllegalArgument();
        return 0;
    }
    if (geometry->getClassInfo() != IlvMapLineString::ClassInfo()) {
        status = IlvMaps::ClassError();
        return 0;
    }

    const IlvMapLineString* lineStr = (const IlvMapLineString*) geometry;

    int segCount = lineStr->getSegmentCount();
    if (segCount == 0)
        return 0;

    IlvMapGeneralPath* genPath = new IlvMapGeneralPath(getDisplay());

    const IlvMapSegment *segment;
    IlvCoordinate c;
    IlvPoint p;

    segment = lineStr->getSegment(0);
    c = segment -> getStartPoint();
    status = targetMapInfo.toViews(c, feature.getProjection(), p);
    genPath->moveTo(p);

    for (int i = 0; i < segCount ; i++) {
        c = segment -> getEndPoint();
        status = targetMapInfo.toViews(c, feature.getProjection(), p);
        genPath->lineTo(p);
    }
```

地図機能の座標は、マネージャの座標系に変換する必要があります。この変換により、マネージャ座標系では下向きになるのに対し、地図データ座標系には上向きの y 軸の正数部分があるため、y 軸の方向が変更されます。これはまた、投影図法の変更も意味します。この例では、toViews メソッドは、投影図法に従って両方の座標を変換し、必要に応じて y 軸の方向を変更します。特に、ソース・データが地理参照されていない場合 (すなわち、投影図法が不明な場合)、または再投影の必要がない場合は、ソース (feature::getProjection) とターゲットの投影図法を 0 に設定できることに留意してください。投影図法の詳細については、64 ページの *ターゲット投影図法を選択する*および 6 章*地図投影図法*を参照してください。

グラフィック・オブジェクトを作成すると、下記のように、色モデルを使って線を着色するためのアトリビュート値を取得します。

```
    IlvInt colorIndex = 0;

    const IlvFeatureAttributeProperty* attributeList =
        feature.getAttributes();

    const IlvFeatureAttribute* fa =
        attributeList->getAttribute(_attributeName);

    const IlvMapClassInfo* clsinfo =
        fa->getClassInfo();

    if(clsinfo->isSubtypeOf(IlvIntegerAttribute::ClassInfo()))
        colorIndex = ((IlvIntegerAttribute*)fa)->getValue();
    else if(clsinfo->isSubtypeOf(IlvDoubleAttribute::ClassInfo()))
        colorIndex = (IlvInt)((IlvDoubleAttribute*)fa)->getValue();

    IlvColor* color = getDisplay()->
        getColor((IlvUShort)_colorModel->getRed(colorIndex),
                 (IlvUShort)_colorModel->getGreen(colorIndex),
                 (IlvUShort)_colorModel->getBlue(colorIndex));

    genPath->setForeground(color);

    return genPath;
}
```

レンダラの永続化

場合によっては、レンダラを保存する必要があります。たとえば、ロード・オン・デマンド・モードで作業を行っている場合、オブジェクト自体ではなく、レイヤのグラフィック・オブジェクトをロードする際に必要なパラメータのみを保存するような場合です。ロード・オン・デマンドについては、4 章*ロード・オン・デマンドの使用*を参照してください。

特定のレンダラを使ってグラフィック・オブジェクトを作成している場合は、オブジェクトを描画するレンダラを次にロードする場合と同じ方法で保存する必要があります。たとえば、クラス `IlvSDOLayer` を使うと、レイヤに保存するレンダラを指定できます。*リファレンス・マニュアル*の `IlvSDOLayer::setFeatureRenderer` を参照してください。

前のセクションで説明した `ColorLineRenderer` は、`IlvFeatureRenderer` 抽象クラスから派生します。このクラスは、永続性に関連する 3 つのメソッドを指定します。`isPersistent` メソッドは、レンダラに永続性があるかどうかを指定し、`write` メソッドはレンダラを `IlvOutputFile` に書き込み、`save` メソッドはクラス

とレンダラを保存します。永続性のあるレンダラを実装する場合は、isPersistent メソッドと write メソッドの上書きが必要です。

```
IlvBoolean
ColorLineRenderer::isPersistent()
{
    return IlvTrue;
}

void
ColorLineRenderer::write(IlvOutputFile& output) const
{
    IlvWriteString(output, _attributeName);

    if(_colorModel->isPersistent()) {
        output.getStream() << 1 << IlvSpc();
        _colorModel->write(output);
    } else {
        output.getStream() << 1 << IlvSpc();
        IlvWarning("colormodel not saved");
    }
}
```

ただし、他の任意の色モデルを使って、永続性のないレンダラも作成できます。その場合、色モデルは保存されず、write メソッドが警告を生成します。

レンダラとともに色モデルが保存されない場合は、レンダラの読み込み時にデフォルトの色モデルが作成されます。

```
ColorLineRenderer::ColorLineRenderer(IlvInputFile& stream)
    :IlvFeatureRenderer(stream.getDisplay(), IlvTrue)
{
    char* s = IlvReadString(stream);
    if(s)
        _attributeName = strcpy(new char[strlen(s)+1], s);

    IlvInt hasColorModel;
    stream.getStream() >> hasColorModel;
    if(hasColorModel) {
        _colorModel = (IlvMapColorModel*)
            IlvMapClassInfo::ReadObject(IlvMapColorModel::ClassInfo(),
                                        stream,
                                        0);
    } else {
        _colorModel = IlvIntervalColorModel::MakeElevationColorModel();
    }
}
```

既存レンダラの拡張

ほとんどの場合、レンダラを最初から書く必要はありません。パッケージで提供されているいずれかのデフォルトのレンダラを使って、必要に応じて変更できます。

このセクションでは、IlvDefaultPointRenderer を拡張して、レンダリングする機能ポイントの横に、タイプ IlvLabel のテキストを追加する方法について説明します。

このセクションに示す例のソース・コード一式は、以下のファイルにあります。

<installdir>/samples/maps/userman/src/markerTextRenderer.cpp

テキストは、クラス MarkerTextRenderer に提供された名前のあるアトリビュートに格納されます。このテキストがある場合、テキストはレンダラのスーパークラスで作成するマーカーと一緒に返されるか、またはマーカーのみが返されます。

```
IlvGraphic*
MarkerTextRenderer::makeGraphic( const IlvMapFeature& feature,
                                 const IlvMapInfo& targetMapInfo,
                                 IlvMapsError& status) const
{
    IlvMarker *marker =
        (IlvMarker *)IlvDefaultPointRenderer::makeGraphic(feature,
                                                          targetMapInfo,
                                                          status);
    if(!marker)
        return 0;

    IlvLabel* label = 0;

    const IlvFeatureAttributeProperty* attributeList =
        feature.getAttributes();

    const IlvFeatureAttribute* attribute =
        attributeList->getAttribute(_attributeName);

    if(attribute)
        label = new  IlvLabel (getDisplay(),
                               marker->getPoint(),
                               ToString(attribute));

    if (!label)
        return marker;

    label->setForeground(marker->getForeground());
    IlvGraphicSet* set = new IlvGraphicSet();
    set->addObject(marker);
    set->addObject(label);
    return set;
}
```

機能イテレータ

機能イテレータは、地図オブジェクトの読み込みに使用するオブジェクトです。IlvMapFeatureIterator 抽象クラスによって、各種ソース (ファイル、データ

ベース、マップ・サーバなど) の地図データを、ソースにかかわらず一定の方法で読み込むことができます。

IBM® ILOG® Views Maps で提供されているすべての定義済みリーダは、このインターフェースを実装します。リーダは、パッケージ・フォーマットで提供されます。これらのリーダの詳細については、5 章*定義済みリーダ*を参照してください。

このセクションでは、以下のトピックを取り上げます。

- *IlvMapFeatureIterator の概要*では、クラスと主なメソッドについて説明します。
- *新しいリーダの作成*では、例を使って、特定形式の地図ファイルのリーダをプログラミングする方法について説明します。この例を通じて、インターフェースが提供するほとんどのメソッドを使う機能イテレータの基本的な実装を学びます。

IlvMapFeatureIterator の概要

地図機能イテレータには、次の 3 つの主要メソッドがあります。

- `IlvMapFeatureIterator::getNextFeature` メソッドを使うと、ファイルから読み込んだり、データベースやマップ・サーバから問い合わせを行なった一連の地図オブジェクトについて反復処理ができます。このメソッドは、最後の地図機能を読み込んだ場合に null ポインタを返します。`IlvMapFeatureIterator` 抽象クラスを実装する IBM ILOG Views の定義済みリーダは、必ず `IlvMapFeature` の同じインスタンスを返します。したがって、`getNextFeature` メソッドが呼び出されるたびに、新たに読み込んだ地図機能は前の地図機能に置き換えられ、前の地図機能はジオメトリやアトリビュートとともに永久的に失われます。その結果、`getNextFeature` メソッドをもう一度呼び出す前に、この地図機能を必ず使うか、または地図機能を保持する場合は、その地図機能のコピーを取る必要があります。これは、48 ページの*アトリビュートをグラフィック・オブジェクトに付加する*の例で実行したものです。

  地図機能が揮発性なのは、新しい機能を読み込むたびにメモリが割り当てられ、パフォーマンスが失われるのを回避するためです。初めて地図機能を読み込んだときに保存用にメモリを割り当て、必要な場合のみ更新する方が効率的です。

- `getDefaultFeatureRenderer` メソッドは、読み込んだ地図機能にそれぞれ適切なレンダラを返します。49 ページの*レンダラ*を参照してください。

- データ・ソースが地理参照されている場合、`getProjection` メソッドは null 以外の `IlvProjection` を返し、その他の場合は null 投影図法を返します。データに投影図法が記述されている場合に、そのデータ・ソースは地理参照されていると言います。投影図法については、64 ページの*ターゲット投影図法を選択する* および 6 章*地図投影図法* を参照してください。また、67 ページの*非地理参照ファイルをロードする*も参照してください。

以下のコード例では、イテレータで読み込む地図機能をすべてロードし、それらの機能をマネージャに置きます。

```
IlvFeatureRenderer* renderer =
featureIterator->getDefaultFeatureRenderer(display);

for (const IlvMapFeature* feature = featureIterator->getNextFeature(status);
     feature;
     feature = featureIterator->getNextFeature(status)) {

    if (status != IlvMaps::NoError()) {
        IlvPrint(IlvMaps::GetErrorMessage(status, display));
        return;
    }

    IlvGraphic* graphic = renderer->makeGraphic(feature,
                                                mapinfo,
                                                status);
    if(graphic)
        manager->addObject(graphic, IlvFalse, layerIndex);
    else if(status != IlvMaps::NoError())
        IlvPrint(IlvMaps::GetErrorMessage(status, display));
}
```

新しいリーダの作成

このセクションには `IlvMapFeatureIterator` の例があり、これを使って ASCII ファイルに保存されているポリラインを読み込むことができます。

> ***メモ:*** `IlvMapFeatureIterator` *抽象クラスを実装するクラスは、必ずしもファイル・リーダとは限りません。たとえば、マップ・サーバへの問い合わせ結果について反復処理が行えます。*

読み込むファイル

読み込む ASCII ファイルは、特にこの例のために作成されています。その形式は非常に単純で、仕様は以下のとおりです。

- このファイルには、形式を特定するヘッダがあります。
- 1 行ごとに 1 組の座標 (緯度と経度) があります。これらの座標は度単位で表されます。
- 行にコメントが含まれている場合があります。コメントがある場合、それらのコメントはマージされてアトリビュートになります。
- ポリラインは空白行で分離されています。
- ファイルには、`.pol` 拡張子が付いています。

以下は ASCII ファイルです。

```
#ascii polylines
-1.0 40.0   A 1x1 degree rectangle centered on the
 1.0 40.0    (0,39) point
 1.0 38.0
-1.0 38.0
-1.0 40.0

0.0  90.0   A meridian extending from the North pole to the South pole
0.0 -90.0
```

リーダ

このセクションでは、このポリライン・ファイルの読み込みに使用可能なリーダについて説明します。

この例のソース・コード一式は、以下のファイルにあります。

<installdir>/samples/maps/userman/src/simplePolylineReader.cpp

メモ: *ここには、説明の必要なコード部分のみを示します。*

以下のように、SimplePolylineReader は、IlvMapFeatureIterator 抽象クラスを実装します。

```
#ifdef IL_STD
#include <fstream>
using namespace std;
#else
#include <fstream.h>
#endif

#include <ilviews/maps/mapfeature.h>
#include <ilviews/maps/format/mapinput.h>
#include <ilviews/maps/rendering/curverdr.h>

class ILVMAPSEXPORTED SimplePolylineReader
    :public IlvMapFeatureIterator
{
public:
    SimplePolylineReader(IlvDisplay* display,
                         const char*  filename);
    virtual ~SimplePolylineReader();

    virtual IlvMapsError getInitStatus() const;

    virtual const IlvMapFeature* getNextFeature(IlvMapsError& status);

    virtual IlvBoolean getLowerRightCorner(IlvCoordinate& coordinate) const;

    virtual IlvBoolean getUpperLeftCorner(IlvCoordinate&) const;

    IlvFeatureRenderer* getDefaultFeatureRenderer(IlvDisplay *);

    const IlvProjection* getProjection() const;
```

```
    IlvBoolean emptyLine(const char* line);

    IlvBoolean readHeader(const char* line);

    int parseLine(const char* line,
                  IlvDouble* lng,
                  IlvDouble* lat,
                  char* comment);

    IlvFeatureAttributeProperty* attributes(IlString& buffer,
                                            IlvMapsError& status);

    IlvMapFeatureIteratorDeclareClassInfo();

private:
    static IlvMapsError _formatError;
    ifstream _stream;
    IlvMapLineString* _geometry;
    IlvMapFeature* _feature;
    IlvProjection* _projection;

    IlvDisplay* _display;
    IlvMapsError _status;
    static IlvMapsError FormatError();
    static void Init();
    const IlvMapFeature* readPolyline(IlvMapsError& status);
};
```

IlvClassInfo の登録

(IlvMapLoader::makeFeatureIterator メソッドが返すような)
IlvMapFeatureIterator のクラスを決定するためには、IlvClassInfo 登録が必要です。新しいリーダ・クラスを登録するには、次のような定義済みマクロを使用します。

```
IlvMapFeatureIteratorDefineClassInfo(className,superClassName);
```

ここで、className は新しいリーダ・クラスの名前 (現在の例では SimplePolygonReader) で、superClassName はスーパークラスの名前 (現在の例では IlvMapFeatureIterator) です。

IlvMapFeatureIterator クラスは、次の方法で確認できます。

```
IlvMapFeatureIterator* reader;
......
IlvClassInfo* readerClass = reader->getClassInfo();
if (readerClass->isSubtypeOf(SimplePolygonReader::ClassInfo())) {
...
}
```

地理参照メソッド

ポリライン・ファイルの緯度と経度が度単位で表されていることから、地理投影図法であることがわかります。getProjection メソッドが

IlvGeographicProjection を返すからです。getProjection メソッドは、読み込むファイルの投影図法が不明な場合に、null を返す場合があります。

```
SimplePolylineReader::SimplePolylineReader(IlvDisplay* display,
                                           const char* filename)
    :_display(display),
     _status(IlvMaps::NoError()),
     _stream(filename),
     _geometry(0),
     _feature(0),
     _projection(new IlvGeographicProjection())
{
    char line[1024];
    _stream.getline(line, 1024);
    if(readHeader(line) == IlvFalse)
        _status = _formatError;
}
const IlvProjection*
SimplePolylineReader::getProjection() const {
    return _projection;
}
```

バウンディング・ボックス・メソッド

データ形式のために、すべてのデータが読み込まれるまでポリラインのバウンディング・ボックスを取得できません。ここで、メソッド getUpperLeftCorner と getLowerRightCorner は、IlvFalse を返し、これらのポイントが不明であることを示します。また、すべてのデータを読み込んで配列に置き、バウンディング・ボックスを計算する方法もあります。

```
IlvBoolean
SimplePolylineReader::getLowerRightCorner(IlvCoordinate& c) const {
    return IlvFalse;
}

IlvBoolean
SimplePolylineReader::getUpperLeftCorner(IlvCoordinate& c) const {
    return IlvFalse;
}
```

レンダリング・メソッド

getDefaultFeatureRenderer メソッドは、すべての地図機能がこの機能イテレータで読み込まれるグラフィック・オブジェクトに変換可能なレンダラを返す必要があります。IlvDefaultCurveRenderer は、タイプ IlvMapLineString のジオメトリがある地図機能を処理できます。

```
IlvFeatureRenderer*
SimplePolylineReader::getDefaultFeatureRenderer(IlvDisplay *display) {
    return new IlvDefaultCurveRenderer(display);
}
```

返された地図機能のジオメトリが定義されておらず、代わりに派生クラスのインスタンスである場合、または地図機能のアトリビュートが色や線の幅などのレンダリング処理に使用する描画パラメータを格納する場合、これらのアトリビュートや派生ジオメトリを処理できるレンダリング・スタイルのレンダラを提供する必要があります。51 ページの*色付き線のレンダラを作成する*を参照してください。

getNextFeature メソッド

`IlvMapFeatureIterator::getNextFeature` メソッドは、地図機能のジオメトリを読み込み、ジオメトリの処理に必要なすべての情報を保持する `IlvMapFeature` オブジェクトを作成します。次のコード例で読み込むジオメトリは、ポリライン・ジオメトリを定義するクラスの `IlvMapLineString` です。

```
 const IlvMapFeature*
SimplePolylineReader::getNextFeature(IlvMapsError& status) {
    return readPolyline(status);
}
```

ポリラインのポイントは、プライベート・メソッド `readPolyline` で読み込まれます。このメソッドはファイルの各行を読み込み、ポイントと関連するコメントの座標がある場合にそれを抽出します。

詳細は、次のようになります。

1. タイプ `IlvMapLineString` のジオメトリを作成またはリセットし、地図機能に関連付けます。パフォーマンス向上のため、リーダが常に `IlvMapFeature` の同じインスタンスを返すことに注意してください。ジオメトリの割り当ても、それらポイントが最初に読み込まれる時点の 1 回だけで、`getNextFeaure` を呼び出すたびに空になります。`getNextFeature` メソッドが返した地図機能は揮発性なので、メソッドを再び呼び出す前にジオメトリとアトリビュートを使用する必要があります。`IlvMapFeatureIterator` インターフェースを実装する、IBM ILOG Views Maps ライブラリのすべてのリーダはこのように動作します。

```
const IlvMapFeature*
SimplePolylineReader::readPolyline(IlvMapsError& status) {

    if(_stream.eof())
        return 0;

    if(!_geometry)
        _geometry = new IlvMapLineString();
    else
        _geometry->removeAll();

    if(!_feature) {
        _feature = new IlvMapFeature();
        _feature->setGeometry(_geometry);
        const IlvProjection* proj = getProjection();
        if(proj)
            _feature->setProjection(proj->copy());
    }
```

2. 次に線が読み込まれます。ファイルの末尾 (EOF)、または空白行に達すると、読み込んだ最終機能を返します。

```
char line[1024];
int first = 1;
IlString buffer;

IlvCoordinate c;

while (1) {
    _stream.getline(line, 1024);

    if(_stream.eof() || emptyLine(line)) {
        status = IlvMaps::NoError();
        IlvFeatureAttributeProperty* prop =
            attributes(buffer, status);
        if(status != IlvMaps::NoError())
            IlvWarning(IlvMaps::GetErrorMessage(status, _display));
        _feature->setAttributes(prop);
        buffer = 0;
        return _feature;
    }
```

3. 次に示すコードでは、parseLine メソッドによって、経度ポイントと緯度ポイントが読み込まれます。コメントがあると、それをコメント・バッファに追加します。このバッファは、後述する attributes メソッドによって処理されます。

```
double x, y;
char comment[1024];
int i = parseLine(line, &x, &y, comment);

if(i < 2) {
    status = _formatError;
    return 0;
}

c.x(x);
c.y(y);

if(i == 3)
    buffer += comment;
```

4. ファイルから読み込まれた各ポイントを、行文字列のジオメトリに追加します。

```
        if(first) {
            first = 0;
           _geometry->setStartPoint(c);
        } else {
            _geometry->lineTo(c);
        }
    }
}
```

5. 地図機能に関連付けるフォーム・アトリビュートへコメントを抽出します。これは、attributes メソッドです。

```
   IlvFeatureAttributeProperty*
SimplePolylineReader::attributes(IlString& buffer, IlvMapsError& status)
{
    if(buffer.getLength() == 0)
        return 0;

    IlvUInt count = 1;

    const char* name = "Comment";

    const IlvMapClassInfo* attClass = IlvStringAttribute::ClassInfo();

    const IlvBoolean nullable = IlvTrue;

    IlvFeatureAttributeInfo* info =
        new IlvFeatureAttributeInfo( count,
                                     &name,
                                     &attClass,
                                     &nullable);

    const char* val = buffer.getValue();
    IlvStringAttribute* att =
        new IlvStringAttribute();
    att->setValue(val);

    IlvFeatureAttributeProperty* prop =
        new IlvFeatureAttributeProperty(info,
                                        (IlvFeatureAttribute**)&att,
                                        status);

    return prop;
}
```

ターゲット投影図法を選択する

地図は常に、特定の投影図法内で表現されます。異なるソースからのデータを同じマネージャ内にマージする場合は、ターゲット地図での各位置がソースの投影図法におけるそれぞれの位置を正確に反映するように、再投影する必要があります。また、スケール、コンパス、座標ビューアなどの、IBM® ILOG® Views Maps グラフィカル・ユーザ・インターフェース (GUI) コンポーネントをアプリケーションに含める場合は、それらのコンポーネントで参照投影図法を正しく操作できる必要があります。地図作成用に設計された IBM ILOG Views コンポーネントは、GUI パッケージに含まれます。

投影図法を `IlvManager` に関連付ける場合は、`IlvMapInfo` クラスを使用します。このクラスは、`IlvProjection` および `IlvMapAdapter` を保持し、投影図法の定義とともに地理座標とマネージャ座標のマッピングを定義します。

次のサンプルでは、メルカトル図法をマネージャに関連付けます。

```
IlvMercatorProjection* proj = new IlvMercatorProjection();
IlvMapInfo* mapInfo = new IlvMapInfo(proj, 0, IlvFalse);
mapInfo->attach(_manager);
```

マネージャを (`IlvManager::save` メソッドの呼び出しによって) `.ilv` ファイルに保存すると、地図情報が自動的に保存されます。`.ilv` ファイルから地図を読み込む場合は、地図の作成に使われた投影図法の種類を知るためにマネージャに含まれる投影図法が検索できます。例:

```
IlvManager* manager = new IlvManager(display);
manager->read(fileName);
IlvMapInfo* mapInfo = IlvMapInfo::Get(manager);
if(!mapInfo) {
    IlvPrint("No IlvMapInfo was saved in this file");
} else {
    const IlvProjection* projection = mapInfo->getProjection();
    if(!projection)
        IlvPrint("No projection was saved in this file");
    else
        IlvPrint("The %s projection was saved in this file",
                 projection->getClassInfo()->getProjectionName());
}
```

パッケージ `projection` では、すべて `IlvProjection` ベース・クラスから継承する多くの定義済み投影図法が提供されます。これらの投影図法は、6 章*地図投影図法*で説明されています。また、67 ページの*非地理参照ファイルをロードする*も参照してください。

IBM ILOG Views に地図をロードする

IBM® ILOG® Views に地図データをロードする操作は非常に簡単です。このセクションでは、IBM ILOG Views に地図をインポートする方法について説明します。以下のトピックから構成されています。

- *IBM ILOG Views 形式の地図をロードする*
- *マップ・ローダー*

IBM ILOG Views 形式の地図をロードする

IBM ILOG Views ファイル (`.ilv`) には、さまざまな種類の情報が含まれます。それらの情報は、他の IBM ILOG Views パッケージ (Graph Layout、Prototypes) や、エンドユーザが設計した IBM ILOG Views アプリケーションからの情報です。また、これらのファイルには、たとえばマップ・ビルダで作成された地図データ、地図の

投影図法、使用する測定単位、グラフィック・オブジェクトやそれらのアトリビュートを含むマネージャ・レイヤ、*タイル・レイヤ*、レイヤのビジビリティ・フィルタなども含まれます。

.ilv 形式の地図は、IlvManager クラスのマネージャや、IlvGrapher などのいずれかのサブクラスにロードされます。これらのクラスは、IBM ILOG Views Manager パッケージの一部です。

.ilv 形式の地図をロードするには、以下に示すとおり、IlvManager::read メソッドを使用します。

```
manager->read("filename.ilv");
```

マップ・ローダー

パッケージ format では、クラス IlvMapLoader を提供しています。このクラスを使って、IBM ILOG Views が定義済みリーダ (形状ファイル、DTED、CADRG など) を提供しているあらゆるファイル形式が、非常に単純な方法でロードできます。これら定義済みリーダの詳細については、5 章*定義済みリーダ*を参照してください。

このセクションでは、以下のトピックを取り上げます。

- *定義済みの地図形式をロードする*
- *非地理参照ファイルをロードする*
- *レンダラの指定*
- *IlvMapLoader クラスの拡張*
- *IlvMapLoader クラスの拡張*

定義済みの地図形式をロードする

定義済みの地図形式をロードするには、次のメソッドを使います。

```
load(const char* fileName, IlvMapsError& status)
```

このメソッドでは、さまざまな形式の仕様に設定された命名規則セットに基づき、まずファイル形式を特定し、適切なリーダを初期化します。その後、マップ・ローダーに関連付けられているマネージャに地図をロードします。

以下の例では、マップ・ローダーを使って、地図ファイルを IBM ILOG Views マネージャにインポートする方法を示します。ファイルは、形状ファイル、CADRG ファイル、または DTED ファイルです。

```
IlvMapLoader* loader = new IlvMapLoader(&manager);
IlvMapsError status = IlvMaps::NoError();
loader->load(filename, status);
if(status != IlvMaps::NoError())
   IlvWarning(IlvMaps::GetErrorMessage(status, display));
```

非地理参照ファイルをロードする

マップ・ローダーを使って地図を IBM ILOG Views マネージャにロードすると、ソース・データ形式が地理参照されている限り、この地図はマネージャに関連付けられた投影図法で自動的に表示されます。`IlvMapFeatureIterator` 抽象クラスには、`getProjection` メソッドがあり、このメソッドでファイルが地理参照されているかどうかを確認できます。`getProjection` メソッドが 0 または `IlvUnknownProjection` を返せば、ファイルは地理参照されていません。その他の場合、ファイルは地理参照されています。詳細については、「機能イテレータ」のセクションを参照してください。

地図ファイルのほとんどは地理参照されています。これは、DTED および CADRG 形式の場合です。形状ファイルなど他の地図形式の中には地理参照されないものもあります。

地理参照されていない、他に何も指示がないファイルからデータをロードする場合、マップ・ローダーはターゲットの投影図法 (マネージャに関連付けられている投影図法) にソース・データを再投影できません。64 ページの *ターゲット投影図法を選択する*を参照してください。

> ***メモ:*** *同じマネージャで、投影図法が不明な形状ファイル形式の複数のソース・ファイルをロードする場合、データが同じ座標系で表現されていればオブジェクトは適切に配置されます。ただし、マネージャで他の形式のデータをインポートすると、異なるソース形式から派生するオブジェクトの相対位置が不正確になります。*

形式が地理参照されていないファイルを読み込む場合でも、データが表現されている投影図法がわかっていれば、`IlvMapLoader::setDefaultSourceProjection` メソッドを使って、この情報を `IlvMapLoader` に提供できます。

以下の例は、投影図法が地理投影図法であることがわかっている形状ファイルを、メルカトル図法のマネージャにインポートする方法を示しています。

```
// Initialize the manager for the mercator projection.
IlvProjection* projection = new IlvMercatorProjection();
IlvMapInfo* mapinfo = new IlvMapInfo(projection, 0, IlvFalse);
mapinfo->attach(manager);

// Create a map loader.
IlvMapLoader mapLoader = new IlvMapLoader(manager);

// Load other data.
....

// Load a shape file that is in the geographic projection.
IlvProjection* geographic = new IlvGeographicProjection();
mapLoader->setDefaultSourceProjection(geographic);
mapLoader->load("myShapeFile.shp");
```

レンダラの指定

IlvMapLoader オブジェクトで特定のレンダラを使用する場合、以下のとおり、そのオブジェクトを load(IlvMapFeatureIterator* featureIterator, IlvFeatureRenderer* renderer, IlvMapsError& status) メソッドの 2 番目の引数に指定します。

```
IlvMapLoader* loader = new IlvMapLoader(&manager);
IlvMapsError status = IlvMaps::NoError();
IlvMapFeatureIterator* iterator =
    loader->makeFeatureIterator(filename);
IlvDefaultCurveRenderer* renderer =
    new IlvDefaultCurveRenderer(display);
IlvMapLineRenderingStyle* style =
    new IlvMapLineRenderingStyle(display);
style->setForeground(display->getColor("green"));
renderer->setLineRenderingStyle(style);
loader->load(iterator, renderer, status);
if(status != IlvMaps::NoError())
    IlvWarning(IlvMaps::GetErrorMessage(status, display));
```

IlvMapLoader クラスの拡張

このセクションでは、IlvMapLoader クラスをサブタイプ化して、IBM ILOG Views Maps の定義済み形式以外のファイル形式を認識できるようにする方法を説明します。

メソッド IlvMapLoader::makeFeatureIterator では、パラメータで指定されたファイルの形式を認識するリーダを作成します。ここでは、IlvMapLoader クラスを取得して、57 ページの*新しいリーダの作成*セクションに記載されているポリライン・ファイルの形式が認識できるようにこのメソッドを変更し、適切なリーダを初期化します。

ファイルに .pol 拡張子があると仮定します。

```
#include <strings.h>

#include <ilviews/maps/format/maploader.h>

#include "polread.h"

class MyMapLoader
    :public IlvMapLoader
{
public:
    MyMapLoader(IlvDisplay* display,
                IlvManager* manager);

    /**
     * Overrides the makeFeatureIterator method from super class.
     */
    virtual IlvMapFeatureIterator* makeFeatureIterator(const char* fileName);
private:
    IlvDisplay* _display;
};

MyMapLoader::MyMapLoader(IlvDisplay* display,
                         IlvManager* manager)
    :IlvMapLoader(manager),
     _display(display)
{
}

IlvMapFeatureIterator*
MyMapLoader::makeFeatureIterator(const char* fileName)
{
    // Does superclass know the format of provided file?
    IlvMapFeatureIterator* result =
        IlvMapLoader::makeFeatureIterator(fileName);
    // If not, try with the polygon reader.
    if (!result) {
        // test extension
        int length = strlen(fileName);
        // .pol are polylines files.
        if (length > 4) {
            char* ptr = strrchr(fileName, '.');
            if(ptr)
                if(strcasecmp(ptr, ".pol") == 0)
                    return new SimplePolylineReader(_display, fileName);
        }
    }
    return result;
}
```

makeFeatureIterator メソッドでは、まずスーパークラスから IlvMapFeatureIterator を取得します。ファイルが認識されない場合、指定のファイル拡張子 (この例では .pol) が読み込むファイルの拡張子に一致するかどう

かを判定します。テストの結果が成功ならば、適切なリーダが作成されます。ここでは、57 ページの*新しいリーダの作成*で作成したリーダです。

ファイルにヘッダが含まれていない場合、メソッドは null ポインタを返し、ファイル形式を特定できなかったことを示します。

縮尺フィルタ

地図作成時に、一定の縮尺でのみ特定タイプの情報が表示されるマネージャ・レイヤを表示する必要が生じる場合があります。たとえば、道路インフラストラクチャについて、地図を小縮尺でオーバーロードされないように十分ズームした場合のみ、二次道路を表示する必要のある場合があります。

IBM® ILOG® Views マップ・ビルダのレイヤ・ウィンドウには、縮尺フィルタ・コマンドが用意されています。ただし、`IlvScaleVisibilityFilter` クラス によっても、縮尺フィルタを実行できます。このクラスは、地図用にカスタマイズした IBM ILOG Views 4.0 の一般クラス `IlvLayerVisibilityFilter` の拡張機能です。

以下は、縮尺が 1/100,000 ～ 1/500,000 の場合のみ表示するレイヤを指定するコードの例です。

```
IlvScaleVisibilityFilter* filter =
  new IlvScaleVisibilityFilter(1./500000., 1./100000.);
layer->addVisibilityFilter(filter);
```

`IlvScaleVisibilityFilter` コンストラクタは引数として、2 つの倍率をとります。マネージャ・レイヤの縮尺がこれら 2 つの値の中間にあれば、そのレイヤは表示されます。この限界チェックを破棄する場合は、倍率として `IlvScaleVisibilityFilter::NoLimit()` の値を渡すこともできます。マネージャ・レイヤの縮尺は、マネージャに付加されたマネージャ座標の表現に使用される測定単位 (m、cm、ヤードその他) を示す `IlvMapInfo` によって算出されます。縮尺は、レイヤを保持するマネージャの測定単位に従って計算されます。

以下のコード例では、レイヤが 1f/500,000 未満の縮尺のみで表示されるように指示します。

```
 IlvScaleVisibilityFilter* scaleFilter =
    new IlvScaleVisibilityFilter(IlvScaleVisibilityFilter::NoLimit(),
                                 1./500000.);
layer->addVisibilityFilter(scaleFilter);
```

レイヤに付加されているビジビリティ・フィルタは、`.ilv` ファイルに自動的に保存されます。

章

4

ロード・オン・デマンドの使用

この章では、マネージャ・ビューに表示するデータのみをメモリにロードする*ロード・オン・デマンド*機構の使い方、および不要になったデータをアンロードする方法について説明します。非常に大きな地図を使用する場合、この機構は特に役に立ちます。1/25,000 の縮尺で全世界の地図を保存するデータベースを考えてみましょう。これらの地図を 300 DPI の解像度でスキャンする場合、必要なデータベース領域は次のようになります。

- 1 平方キロメートル当たり 654 キロバイト
- 100 平方キロメートル当たり 64 メガバイト
- 全世界で約 310 テラバイト

このデータ量の大きさを考えると、関心がある地図の部分のみを読み込んで表示するロード・オン・デマンド機構があることはきわめて重要です。

この章では、以下のトピックを取り上げます。

- *ロード・オン・デマンドの機能*では、ロード・オン・デマンド・クラスと各クラスの関係について紹介します。
- *タイリング・グリッドの構造とサイズ*では、タイリング・パラメータについて説明します。
- *タイル・ステータスの表示*では、新しい形式にロード・オン・デマンドを実装した際に、デバッグ・ビューを使ってタイルのステータスを表示する方法について説明します。

- 次のコードは、IlvManagerMagViewInteractor をデバッグ・ビューに関連付けます。ロード・オン・デマンドの制御に使用できるさまざまな方法について説明します。
- *エラーとロード・オン・デマンド・イベントの管理*では、ロード・オン・デマンドに関するエラーやイベントの管理方法を説明します。
- *タイルのキャッシュ処理*では、タイルのキャッシュ機構を紹介します。
- *タイル・レイヤの保存*では、タイル・レイヤの保存に関する問題点を明らかにします。
- *新しいキャッシュ・アルゴリズムを記述する*では、カスタマイズされたキャッシュ・アルゴリズムの記述方法について例を挙げて説明します。
- *新しいデータ・ソースにロード・オン・デマンドを実装する*では、新しいタイル・ローダーの実装方法について説明します。

定義済みのマップ・リーダーへのロード・オン・デマンドの実装については、5 章 *定義済みリーダ*で説明します。

ロード・オン・デマンドの機能

ロード・オン・デマンド機構では、マネージャ・レイヤは*タイル*と呼ばれる同じサイズの矩形群に分割されます。タイルに含まれるグラフィック・オブジェクトは、タイルがいずれかのマネージャ・ビューで表示される場合のみ、アプリケーションにロードされます。タイルが任意のマネージャ・ビューで表示されなくなると、アンロードされます。

地図の移動、拡大 / 縮小、*縮小フィルタ*のオン / オフ、ウィンドウのサイズ変更など、ユーザのアクションに従ってタイルは表示、非表示になります。

タイル群に分割するには、レイヤは IlvManagerLayer のサブタイプであるタイプ IlvTiledLayer である必要があります。タイルは、タイル・ローダー (IlvTileLoader) およびタイル・キャッシュ (IlvTileCache) に関連付けられたタイル・コントローラ (IlvTileController) で管理されます。

図 4.1 は、ロード・オン・デマンドの関連クラスと、それらの関係を示しています。

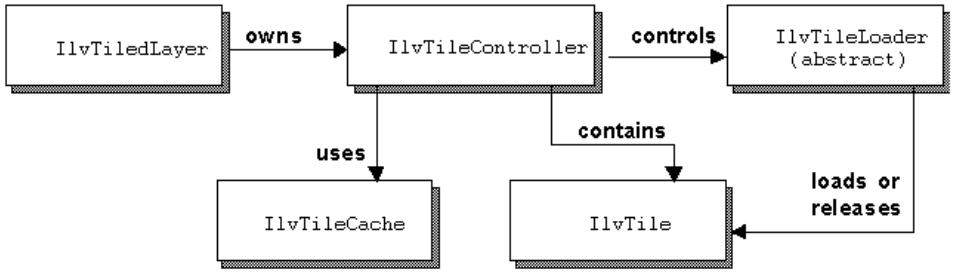


図 4.1 *ロード・オン・デマンド・クラス*

いずれかのマネージャ・ビューでタイルが表示されると、必ずタイル・コントローラに通知され、タイル・ロック・カウンタが増加します。タイルが表示されていない場合は、タイル・ローダーが起動し、タイルのデータはメモリにロードされます。いずれかのマネージャ・ビューでタイルが非表示になると、タイル・ロック・カウンタが減少します。カウンタがゼロになると、タイル・コントローラは関連するキャッシュにタイルを配置します。タイル・コントローラで新しいタイルのロードが必要になると、この処理のキャッシュに通知し、キャッシュの 1 つ以上のタイルが空いているメモリにアンロードされます。キャッシュは、様々なマネージャに属す複数のレイヤに割り当てられているタイルを処理できます。

ロード・オン・デマンド機構を有効にするためには、タイル・レイヤをマネージャに追加し、その `start` メソッドを呼び出す必要があります。

`IlvTiledLayer` クラスは、IBM® ILOG® Views Maps が CADRG、DTED、および Oracle Spatial という定義済みリーダを提供している地図形式でロード・オン・デマンドをサポートするための拡張機能です。クラス `IlvCADRGLayer`、`IlvDTEDLayer`、および `IlvSDOLayer` については、5 章 *定義済みリーダ* で説明します。

`IlvMapTileLoader` は、ロード・オン・デマンド用のもう 1 つの便利なクラスです。このクラスを使うと、定義済みのタイル・ローダーにマップ・リーダ (`IlvMapFeatureIterator`) を統合できます。このクラスの最も重要なメソッドは、`IlvMapTileLoader::load()` メソッドです。

```
IlvMapsError
IlvMapTileLoader::load(IlvTile* tile)
{
 // Get the feature iterator.
 IlvMapFeatureIterator* iterator = getFeatureIterator(tile);
 ...
 // Check if the iterator implements IlvLookAheadFeatureIterator.
 ...
 // Parameters for rendering.
 ...
 IlvFeatureRenderer* renderer = getFeatureRenderer(tile->getDisplay());
 ...
 do {
    // Case of look ahead feature iterator.
    // Check if the next feature ID corresponds to an object
    // already in the manager (skip the next feature in this
    // case and continue).
    ...
    // Process the feature itself.
    ...
    feature = iterator->getNextFeature();
    ...
    // Ask the renderer to make the IlvGraphic.
    ...
    // Attach the attributes to the graphic if necessary
    // and add the graphic to the tile.
   } while (feature);
 ...
}
```

このコード・サンプルには、IlvMapFeatureIterator クラスの IlvLookAheadFeatureIterator クラスに関する最適化の一部が示されています。このクラスをサブクラス化する機能イテレータは、次の機能の ID を取り出し、getNextFeature メソッドから返る次の機能をスキップします。例えば、機能に独自 ID があり、タイル・グループに属する場合 (ジオメトリが大きい場合)、これは役立ちます。これらのタイルの 1 つが初めて表示されると、この機能が読み込まれ、レンダリングされて、対応する IlvGraphic が IlvTile::addObject(IlvGraphic*, IlvMapFeatureId*) メソッドによって IlvTile に追加されます。次に、このグループの別のタイルが表示されると、IlvMapTileLoader の load メソッドが IlvLookAheadFeatureIterator::getNextFeatureId() メソッドをチェックして、返される ID が既存の IlvGraphic に対応するかどうか調べ、複数タイルに属するこの機能がロードされ、レンダリングされた後、タイルに一度だけ追加されます。

独自のタイトル・ローダーを定義するために、特定の IlvTile に合わせたマップ・リーダを返すように、IlvMapTileLoader をサブクラス化し、その IlvMapTileLoader::getFeatureIterator() メソッドをオーバーライドすることができます。

次のコード・サンプルは、IlvMapTileLoader の実装を示します。クラス MyTileLoader によって、IlvManager にある同じイメージのモザイクを、それが IlvTiledLayer に関連付けられたときにロードできます。

```
class MyTileLoader:public IlvMapTileLoader
{
    IlvDisplay* _display;
    const char* _filename; // The filename that corresponds to the image
                           // its format should be known by IBM ILOG Views.
    IlvProjection* _projection;
    IlvMapInfo* _info;
    IlvDim _imageWidth;
    IlvDim _imageHeight;

public:
    MyTileLoader(IlvDisplay* display, const char* filename)
      _display(display),
      _filename(IlvMapDataPathManager::ResolvePath(filename)),
      _projection(new IlvGeographicProjection()),
      _info(0),
      _imageWidth(0),
      _imageHeight(0)
    {
        //Creation of the IlvBitmap corresponding to the given filename.
        IlvBitmap* bitmap =
           display->readBitmap(IlvMapDataPathManager::ResolvePath(filename));
        if(bitmap) {
            _imageWidth = bitmap->width();
            _imageHeight = bitmap->height();
        }
    }

    ~MyTileLoader()
    {
        if(_info)
            delete _info;
    }

    IlBoolean isPersistent() const {
        return IlFalse;
    }

    IlvMapFeatureIterator* getFeatureIterator(IlvTile* tile)
    {
        IlvRect  rect;
        tile->boundingBox(rect);
        IlvMapInfo* info = getMapInfo();
        IlvCoordinate ul;
        IlvCoordinate lr;
        IlvPoint p1(rect.x(), rect.y());
        IlvPoint p2(rect.x() + rect.w(), rect.y() - rect.h());
        ul = info->getAdapter()->fromViews(p1);
        lr = info->getAdapter()->fromViews(p2);
        return new IlvImageReader(_display, _filename, ul, lr);
    }
```

```
    IlvFeatureRenderer* getDefaultFeatureRenderer(IlvDisplay* display)
    {
        return new IlvDefaultFeatureRenderer(display);
    }

    IlvMapInfo* getMapInfo()
    {
        if (!_info)
            _info = new IlvMapInfo(_projection);
        return _info;
    }

    IlvRect getTileOrigin()
    {
        return IlvRect(0, 0, _imageWidth, _imageHeight);
    }
};
```

このコード・サンプルでは、以下の行に注目してください。

```
IlvMapDataPathManager::ResolvePath(filename)
```

このデータ・パス管理機能によって、相対パス名を解決できます。

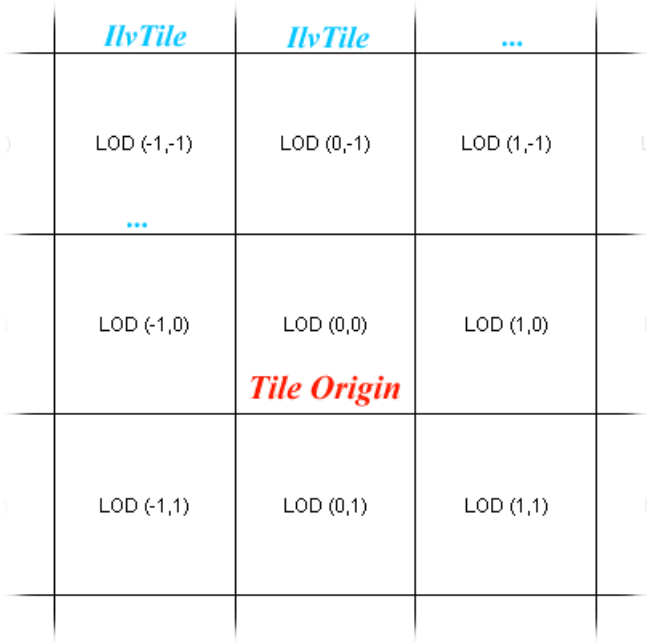
例えば、.ilv ファイルには (形状ファイル、GeoTIFF ファイルなどの) ファイルを参照するタイル・ローダが含まれる場合があります。これらの参照が相対パスの場合でも、デフォルト・リゾルバがあれば、容易に解決できます。例えば、タイル・ローダに必要なファイルが c:\data にあるとしても、必要なのは次のリゾルバを呼び出すことだけです。

```
IlvDefaultDataPathResolver* resolver =
                        new IlvDefaultDataPathResolver("c:\\data");
IlvMapDataPathManager::AddDataPathResolver(resolver);
```

.ilv ファイルの読み込み前に、呼び出すことのできるスタティック関数です。もちろん、.ilv ファイルに保存されたタイル・ローダーが、パス名解決のため IlvMapDataPathManager::ResolvePath(const char* filename) を使用する場合のみ機能します (上記のサンプル・コードを参照してください)。

タイリング・グリッドの構造とサイズ

タイル・レイヤは、マネージャ・レイヤの特定のタイプであり、特にロード・オン・デマンドをサポートするために設計されている。タイル・レイヤは、タイルするグリッドを形成する同サイズの矩形タイルのセットに分けられる。



このセクションでは、以下のトピックを取り上げます。

- *タイリング・グリッドの構造*
- *タイリング・グリッドのサイズ*

タイリング・グリッドの構造

タイリング・グリッドは、index 0 の行と列の交点にある原点のタイルによって定義されます。

グリッドの他のタイルは、原点のタイルから始まる他の列と行番号で識別されます。以下のコード例は、列 `col` と行 `row` の交点にあるタイルのステータスを表示しています。

```
void
SimpleLod::displayTileStatus(int row, int col)
{
    IlvTile* tile = _tiledLayer->getTileController()->getTile(col, row);
    if (!tile)
       IlvPrint("The tile  %d %d is not yet loaded",
                col, row);
    else {
        IlvTileStatus status;
        status = tile->getStatus();
        switch(status) {
        case IlvTileEmpty:
            IlvPrint("The tile %d %d is empty", col, row);
            break;
        case IlvTileLocked:
            IlvPrint("The tile  %d %d is locked", col, row);
            break;
        case IlvTileCached:
            IlvPrint("The tile  %d %d is cached", col, row);
            break;
        case IlvTileDeleted:
            IlvPrint("The tile  %d %d is deleted", col, row);
            break;
        }
    }
}
```

上記のコード例で、`IlvTileController::getTile` method メソッドは値を返す場合があります。タイルの番号は非常に大きくなり (実際に無限に近くなることもあるため)、`IlvTile` オブジェクトはタイルがロードされるか、キャッシュにある場合にのみ割り当てられます。

タイリング・グリッドのサイズ

次のメソッドを使って、タイリング・グリッドのサイズを設定できます。

```
IlvTileController::setSize(IlvRect)
```

タイリング・グリッドを制限する矩形は、マネージャ座標に表示されます。この矩形と交差するタイルのみを読み込むことができます。デバッグ・ビューで定義されたサイズを持つタイリング・グリッドの例は、96 ページの図 4.2 を参照してください。

> ***メモ:*** *このセクションのタイリング・パラメータ (タイルのサイズ、原点のタイル、タイリング・グリッドのサイズ) は、マネージャの各レイヤ・タイルに設定できます。これで、小縮尺で表示されるオブジェクトと含む大きなタイル・レイヤと、大縮尺で表示されるオブジェクトを含む小さなタイル・レイヤを同じマネージャに配置できます。*

タイル・ステータスの表示

デバッグ・ビューを作成して、次の方法でタイル・レイヤのタイルのステータスを表示できます。

```
IlvTiledLayer::setDebugView(IlvView* view,
                            IlvColor* borderColor = 0,
                            IlvColor* lockedTilesColor = 0,
                            IlvColor* cachedTilesColor = 0)
```

名前のとおり、このデバッグ・ビューは、新しい地図形式にロード・オン・デマンドを実装する際のデバッグ処理に特に便利です。

タイルには、4 つの異なるステータスを与えることができます。

- 空 : オブジェクトがメモリにロードされていない。
- ロック済み : オブジェクトがメモリにロードされ表示されている。
- キャッシュ済み : オブジェクトがメモリにロードされているが、表示されていない。
- 削除済み : オブジェクトがキャッシュから削除されている。

デバッグ・ビューが `IlvView` タイプで、タイル・レイヤに付加する必要があります。デバッグ・ビューでは、タイル・ロック・カウンタの増減は行いません。これは、タイル・コントローラの役割です。デバッグ・ビューは、図 4.2 のように、ステータスに従ってタイルを色表示するだけです。

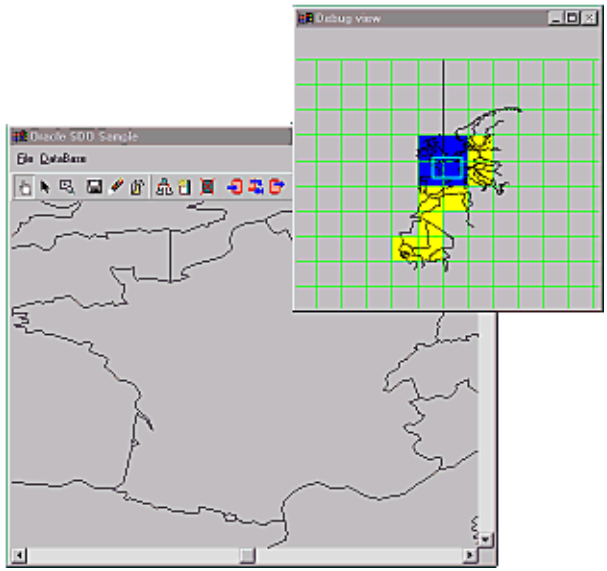


図4.2　ロード・オン・デマンド・デバッグ・ビュー

1 より大きいロック・カウンタを持つタイルは、青色で表示されます。それらのタイルは、1 つ以上のビューで表示されます。カウンタが 0 のタイルは、黄色で表示されます。それらのタイルはキャッシュされています。白色のタイルは、ロードされていません。

前の例では、`IlvManagerMagViewInteractor` はデバッグ・ビューに関連付けられていました。青色のタイルの中に小さい黄色の正方形を表示するのは、このインタラクタです。正方形はメイン・ビューで表示されている地域を表します。

以下は、レイヤのデバッグ・ビューを作成する例です。

```
void
SimpleLod::debugView() {
    _debug = new IlvView(_display,
                         "DebugView",
                         "DebugView",
                         IlvRect(450, 0, 100, 100),
                         IlvFalse, IlvFalse);
    _manager->addView(_debug);
    _tiledLayer->setDebugView(_debug);
}
```

次のコードは、IlvManagerMagViewInteractor をデバッグ・ビューに関連付けます。

```
void
SimpleLod::magView()
{
    _magvint = new IlvManagerMagViewInteractor(_manager, _debug, _view);

    _magvint->setAutoZooming(IlvTrue);
    _magvint->setAutoTranslating(IlvTrue);
    _magvint->setResizingAllowed(IlvTrue);

    _manager->setInteractor(_magvint, _debug);
}
```

ロード・オン・デマンドの制御

このセクションでは、ロード・オン・デマンドの様々な制御方法を紹介します。以下のトピックから構成されています。

- *ビジビリティ・フィルタを使って、ロード・オン・デマンドを制御する*
- *API を介してタイルをロードする*

ビジビリティ・フィルタを使って、ロード・オン・デマンドを制御する

スケール・ビジビリティ・フィルタを使って、マネージャ・レイヤでのタイルの表示を制御できます。ビジビリティ・フィルタの詳細は、71 ページにある *スケール・フィルタ*のセクションを参照してください。

スケール・ビジビリティ・フィルタがタイル・レイヤに設定されている場合、このレイヤは、マネージャ・ビューの倍率設定が指定限度内の場合に表示されます。そうでない場合は、このレイヤは非表示になります。マネージャ・ビューの倍率によって、タイル・レイヤが表示されると、表示タイルは、自動的にロックされメモリに読み込まれます。同様に、ビューの倍率が特定の値を超えているためにタイル・レイヤが非表示になると、表示タイルのすべてのロック設定は解除され、それらのタイルは削除されます。

通常、スケール・ビジビリティ・フィルタを使って、最小スケール値から最大スケール値までの倍率でロード・オン・デマンドを有効にします。1/1,000,000 の縮尺でスキャンした地図の場合を考えてみると、レイヤを 1/2,500,000 ～ 1/500,00 の範囲の縮尺で表示するように、ビジビリティ・フィルタを設定できます。

API を介してタイルをロードする

ロード・オン・デマンドはイベント駆動型の機構で、ズームやパンなどのユーザ操作にしたがって地図データをロード / アンロードします。ただし、lockTile メソッドを使うこともできます。例えば、頻繁に表示する地図領域に対応するタイルをプレロードしたり、タイルをアンロードできないようにできます。

下記の lockTile メソッドでは、タイル・ロック・カウンタを増加します。

```
IlvTileController::lockTile(IlvInt column,
                            IlvInt row,
                            IlvAny source)
```

タイル・ロック・コントローラが 0 の場合、タイルはメモリにロードされ、以下のメソッドでロックが解除されるまでアンロードされません。

```
IlvTileController::unlockTile(IlvInt row,
                               IlvInt column,
                               IlvAny source)
```

エラーおよびロード・オン・デマンド・イベントの管理

ロード・オン・デマンド機構では、メモリに問題がある、データがない、サーバへの接続が失われているなどの理由で、地図をすべてロードできない場合にエラーを生成することがあります。アプリケーションは、これらのエラーを取得し、ユーザに表示中の地図が完全なものでないことを通知する必要があります。

これらのイベントや、ロード・オン・デマンドに関する他のイベントをすべて通知するために、IlvTileListener を IlvTileController クラスのインスタンスに設定できます。このリスナには、タイルへの変更がすべて通知され、タイルのステータスを変更するたびに tileChanged メソッドが呼び出されます。キャッシュ機構に関連するイベントは、IlvTileCache クラスをサブタイプ化することで使用できます。

- `void IlvTileCache::tileAboutToLoad(IlvTile* tile)`

  指定した tile のロード直前に呼び出されます。

- `void IlvTileCache::tileCached(IlvTile* tile)`

  指定した tile をキャッシュに挿入する場合に、呼び出されます。

- `void IlvTileCache::tileRetrieved(IlvTile* tile)`

  指定したタイルをキャッシュから削除する場合に、呼び出されます。

次の例では、ロード・オン・デマンドに関するイベントをすべて表示します。

```
static const char*
toString(IlvTileStatus status) {
    switch(status) {
    case IlvTileEmpty:
        return "IlvTileEmpty";
    case IlvTileLocked:
        return "IlvTileLocked";
    case IlvTileCached:
        return "IlvTileCached";
    case IlvTileDeleted:
        return "IlvTileDeleted";
    }
}

void
Listener::tileChanged(IlvTile* tile,
                          IlvTileStatus oldStatus,
                          IlvTileStatus newStatus)
{
    IlvPrint("tile %d %d status changed from %s to %s",
             tile->getRow(), tile->getColumn(),
             toString(oldStatus),
             toString(newStatus));
}

void
SimpleLod::listener()
{
    _listener = new Listener(_manager, _view);
    _tiledLayer->getTileController()->addTileListener(_listener);
}
```

タイル・リスナには、次のような様々なタイプのイベントが送信されます。

- `IlvTileEmpty` - タイルの初期ステータス (タイルに含まれるオブジェクトはありません)。
- `IlvTileLocked` - メモリにタイルをロードしたときにトリガされます。
- `IlvTileCached` - 未使用のタイルをキャッシュに格納したときにトリガされます。
- `IlvTileDeleted` - タイルを完全に解放したときにトリガされます。

ビューのイベントがタイル上でアクションを実行すると、タイル・コントローラはアクションのタイトル・リスナに通知します。このイベントが一連の移行イベントをトリガすると、それらの移行イベントはグループでリスナに送信されます。したがって、縮尺を変更すると、新しいタイルはロードされ、他のタイルはキャッシュされます。イベントをグループ化すると、生成されたすべての移行イベントが完了した場合のみアクションを実行できます。イベント・グループの開始は下記の `IlvTileListener::beginGroupedNotification` メソッドの呼び出しによって通

知され、終了は下記の IlvTileListener::endGroupedNotification メソッドの呼び出しによって通知されます。

タイルのキャッシュ処理

タイルのキャッシュは、ロック・カウンタが 0 に戻ったタイルの格納場所です。キャッシュ内のタイルは、新しいタイルの読み込みにメモリが必要な場合はアンロード可能です。

キャッシュは、複数のレイヤで共有できます。したがって、レイヤに 1 つのタイルをロードすると、他のレイヤで 1 つのタイルがアンロードされる可能性があります。

IlvDefaultTileCache クラスは、最長時間未使用キャッシュにアクセスしたタイルを最初にアンロードする、簡単な LRU (最長時間未使用) 構造で構成されるキャッシュ・アルゴリズムを実装します。

ただし、アプリケーション特性を考えて、効率のよい、その他のアルゴリズムを実行することもできます。次に、新しいキャッシュ・アルゴリズムを実装する際に考慮すべき条件を示します。

- 現在の場所から様々なパンやズーム操作にアクセスする必要のあるタイルを最初にアンロードすること。
- ロードに最も時間を要したタイルを最初にアンロードすること。
- 最も多くのグラフィック・オブジェクトを含むタイルを最初にアンロードすること。

101 ページのセクション*新しいキャッシュ・アルゴリズムを記述する*に、簡単なキャッシュ・アルゴリズムの例を示します。

タイル・レイヤの保存

レイヤには、様々な関連パラメータがあります。名前付きプロパティ、ビジビリティ・フィルタ、および名前などの一部のパラメータは、タイル・レイヤか通常のレイヤかに関わらず、すべてのレイヤに共通しています。他のパラメータは、タイル・レイヤ特有のものです。これらのパラメータは、前項で説明したタイリング・パラメータ、およびタイル・ローダーです。

通常のレイヤと異なり、タイル・レイヤを .ivl ファイルに保存すると、付加されているパラメータのみが、保持しているオブジェクトではなくレイヤに格納されます。

新しいキャッシュ・アルゴリズムを記述する

このセクションでは、アプリケーション特有の要件を満たすようにカスタマイズされた、キャッシュ・アルゴリズムを記述する方法について説明します。このセクションの例は、IBM® ILOG® Views Maps ライブラリに用意されているクラス IlvDefaultTileCache の簡易版です。この例のソース・コード一式は、以下のファイルにあります。

クラス SimpleTileCache は、クラス IlvTileCache を拡張します。

```
class TileCache
    :public IlvTileCache
{
public:
    TileCache(int size);
    TileCache::TileCache(IlvInputFile& file);
    virtual void tileAboutToLoad(IlvTile *);
    virtual void tileCached(IlvTile *);
    virtual void tileRetrieved(class IlvTile *);
    virtual void controllerDeleted(IlvTileController *);
    virtual void write(IlvOutputFile& output) const;
private:
    int _size;
    IlList _tiles;
};
```

_size では、キャッシュに格納可能なタイルの最大数を定義します。

```
TileCache::TileCache(int size)
    :IlvTileCache(),
     _size(size)
{
}
```

TileCache コンストラクタでは、特定のサイズでデフォルト・キャッシュのインスタンスを作成します。

以下のコンストラクタでは、提供された入力ストリームからキャッシュを読み込みます。このコンストラクタを使って作成したキャッシュには永続性があるため、IlvTiledLayer オブジェクトに保存されます。

```
TileCache::TileCache(IlvInputFile& file)
    :IlvTileCache()
{
    file.getStream() >> _size;
}
```

write メソッドでは、出力ストリームのキャッシュを書き込みます。

```
void
TileCache::write(IlvOutputFile& output)  const
{
    output.getStream() << _size;
}
```

以下のメソッドは、IlvTileCache インターフェースに属します。このメソッドは、タイルのキャッシュ時に呼び出されます。この実装では、タイルは内部タイル・リストの末尾に追加されます。

```
void
TileCache::tileCached(IlvTile *tile)
{
    _tiles.append(tile);
}
```

以下のメソッドは、IlvTileCache インターフェースに属します。このメソッドは、タイルをキャッシュから削除し、再びロックすると呼び出されます。この実装では、タイルは内部タイル・リストから削除されます。

```
void
TileCache::tileRetrieved(IlvTile* tile)
{
    _tiles.remove(tile);
}
```

以下のメソッドは、IlvTileCache 抽象クラスに属します。このメソッドは、タイルのロード直前に呼び出されます。この実装では、キャッシュのタイル数がキャッシュ・サイズを超えた場合に、内部タイル・リストの先頭にある LRU タイルがアンロードされ、新規タイルの領域を確保します。

```
void
TileCache::tileAboutToLoad(IlvTile *tile)
{
    int toRemove = _tiles.length() - _size;
    if (toRemove <= 0)
      return;
    for (int i = toRemove; i > 0; i--) {
        IlvTile* current = (IlvTile*)_tiles[0];
        _tiles.remove(current);
        releaseTile(current);
    }
}
```

以下のメソッドは、IlvTileCache 抽象クラスに属します。このメソッドは、タイル・レイヤをマネージャから取り出し、タイル・コントローラが管理するタイルをキャッシュから削除すると呼び出されます。

```
void
TileCache::controllerDeleted(IlvTileController* controller)
{
    IlvLink* l;
    l = _tiles.getFirst();
    IlvTile* tile;
    while (l) {
        tile = (IlvTile*) l->getValue();
        l = l->getNext();
        if (tile->getController() == controller)
            _tiles.remove(tile);
    }
}
```

新しいデータ・ソースにロード・オン・デマンドを実装する

新しいデータ・ソースにロード・オン・デマンドを実装する場合に必要となるのは、IlvTileLoader 抽象クラスを実装する特定のタイル・ローダーを記述することだけです。ただし、CADRG などの IBM® ILOG® Views に備わる定義済みの地図形式では、ロード・オン・デマンドは、タイル・ローダー (プライベート・クラスとして) と、タイリング・パラメータの両方を関連する適切な形式に定義する IlvTiledLayer のサブクラスで実装されている点に注意してください。CADRG の場合、ロード・オン・デマンドは、タイルがフレームと位置合わせされるように実装されています。定義済みの形式については、5 章 *定義済みリーダ*で説明します。

タイル・ローダーの効率を最大にするためには、以下の要件を満たす必要があります。

- タイルにロードするオブジェクトを決定できること。これらのオブジェクトは、既知の名前を持つファイルから読み込み可能であるもの、または地図データベースへの問い合わせ結果になります。
- データへのダイレクト・ランダム・アクセスが可能なこと。
- 読み込むデータ・サイズはタイル・サイズに比例し、高速ロードが可能なこと。例えば、100x100 のサイズのラスター・イメージは、6000x6000 のサイズのイメージよりも速くロードされます。

タイル・ローダーの以下の例では、2 つのグラフィック・オブジェクト、矩形およびラベルのロードをシミュレートします。

load メソッドは、タイルをパラメータとしてロードします。このメソッドでは、タイルに表示するグラフィック・オブジェクトを生成し、IlvTile::addObject メソッドを呼び出し、それらのオブジェクトをタイル・レイヤに追加します。ロードが完了すると、IlvTile::loadComplete メソッドを呼び出し、タイルのデータが使用可能であることをリスナに通知します。

```
class TileLoader
    :public IlvTileLoader
{
public:
    TileLoader(IlvDisplay*);
    virtual IlvMapsError load(IlvTile* tile);
    virtual void release(IlvTile* tile);
    virtual IlvBoolean isPersistent() const;
private:
    IlvDisplay* _display;
};

IlvMapsError
TileLoader::load(IlvTile* tile)
{
    IlvRect rbbox;
    tile->boundingBox(rbbox);
    IlvRectangle *rect = new IlvRectangle(_display, rbbox);
    tile->addObject(rect);

    IlString str;
    str += ?(?;
    str += tile->getColumn();
    str += ?, ?;
    str += tile->getRow();
    str += ?)?;

    IlvMapLabel* label = new IlvMapLabel(_display,
                                         IlvPoint(),
                                         IlvPoint(),
                                         IlvCenter,
                                         10,
                                         str.getValue());
    IlvRect lbbox;
    label->boundingBox(lbbox);
    IlvPos cx = rbbox.x() + rbbox.w() / 2;
    IlvPos cy = rbbox.y() + rbbox.h() / 2;
    label->move(cx - lbbox.w() / 2,
                cy - lbbox.h() / 2);
    tile->addObject(label);

    tile->loadComplete();
    return IlvMaps::NoError();
}
```

release メソッドは、タイル・キャッシュがタイルを解放すると呼び出されます。tile.deleteAll メソッドでは、タイルを消去します。

```
void
TileLoader::release(IlvTile* tile) {
    tile->deleteAll();
}
```

章

5

定義済みリーダ

この章では、IBM® ILOG® Views Maps で提供されている定義済みリーダの概要を説明します。

- *形状ファイル・リーダ*
- *DTED ファイル・リーダ*
- *CADRG ファイル・リーダ*
- *イメージ・ファイル・リーダ*
- *GeoTIFF リーダ*
- *Oracle Spatial 機能*
- *S57 マップ・リーダ*

形状ファイル・リーダ

このセクションでは、形状ファイル形式のファイルの読み込みを可能にする IlvMaps ライブラリのクラスについて説明します。

以下のトピックから構成されています。

- *形状ファイル形式の概要*
- *形状ファイル形式を読み込むためのクラス*
- *形状ファイルのロード・オン・デマンド*

形状ファイル形式の概要

形状ファイル形式は、ESRI (Environmental Systems Research Institute) のベクトル地図用の交換形式です。この形式では多角形、円弧、線、点がサポートされます。各形状ファイルにはそれぞれ 1 つのテーマが含まれます。つまり、ファイルに含まれるオブジェクトは、すべて同じタイプ (線、点、多角形、または他のタイプのオブジェクト) になります。形状ファイル形式のテーマは次の 4 つの異なるファイルに記述されています。

- 形状ファイル (.shp) には、オブジェクトのジオメトリが含まれます。
- Dbase ファイル (.dbf) には、オブジェクトのアトリビュートが含まれます。このファイルはオプションです。
- インデックス・ファイル (.shx) には、データへのアクセスを容易にする .shp ファイルにあるオブジェクトのインデックスが含まれます。このファイルはオプションで、ロード・オン・デマンド処理によるジオメトリのランダム・アクセスを実行するために使用されます。
- 空間インデックス・ファイル (.idx) には、タイリング情報が含まれます。このファイルは Maps モジュール特有で、形状ファイルでロード・オン・デマンドを実行する際に使用されます。このファイルはオプションで、ロード・オン・デマンド処理のタイリング情報を格納するために使用されます。

この形式には、グラフィック・オブジェクトの位置の参照に使用する座標系に関する情報は含まれません。形状ファイルのオブジェクトはしばしば地理投影 (IlvGeographicProjection) 内に位置付けられますが、常にそうなるわけではありません。

形状ファイル形式を読み込むためのクラス

次に示すクラスは、形状ファイル形式の読み込みに使用されます。

- IlvShapeFileReader

このクラスは、IlvMapFeatureIterator のサブクラスで、これによって .shp ファイル、.dbf ファイル、および .shx ファイルが読み込めます。形状ファイルでは使用する投影図法に関する情報がないため、このリーダは地理参照されません。55 ページの*機能イテレータ*および 67 ページの*非地理参照ファイルをロードする*を参照してください。

- IlvShapeSHPReader

  このクラスは、IlvMapFeatureIterator のサブクラスです。このリーダは、.shp ファイルのみを読み込みます。

- IlvShapeDBFReader

  このクラスは、.dbf ファイルのみを読み込みます。

- IlvShapeFileIndex

  このクラスは、.shx ファイルを読み込みます。詳細については、100 ページの *IlvShapeFileIndex クラス*を参照してください。

- IlvShapeSpatialIndex

  このクラスは、空間インデックス・ファイル (.idx ファイル) を読み込みます。詳細については、101 ページの *IlvShapeSpatialIndex クラス*を参照してください。

IlvShapeFileReader

このリーダは、ジオメトリを格納する .shp ファイルと、アトリビュートを格納する .dbf ファイルを同時に読み込み、情報を 1 つの IlvMapFeature オブジェクトにマージします。このリーダは、ジオメトリへのランダム・アクセスを提供するために、オプションの shx ファイルも読み込めます。

次のいずれかの方法でインスタンス化できます。

- .dbf ファイル、.shp ファイル、オプションの .shx ファイルの名前を指定する。
- IlvShapeDBFReader、IlvShapeSHPReader、オプションの IlvShapeFileIndex オブジェクトを直接指定する。

これは、たとえば派生した IlvShapeSHPReader オブジェクトを使用する場合などに役立ちます。

いずれの場合も、.shx ファイルを読み込めるリーダが作成されると、IlvShapeFileReader::getFeatureAt メソッドを介したジオメトリへのランダム・アクセスが可能になります。

このリーダは、3 つの専用リーダを使用します。IlvShapeSHPReader、IlvShapeDBFReader、および IlvShapeFileIndex。getNextFeature メソッドは、これらの専用リーダで生成した情報を 1 つの*地図機能*にマージします。

例については、次の場所にある形状ファイルのサンプルを参照してください。

<installdir>/samples/maps/shapefile

IlvShapeSHPReader

このリーダは、.shp ファイルのみを読み込みますが、.shp ファイルおよび .shx ファイル兼用として作成された場合は、ジオメトリへのランダム・アクセスを可能にします。

形状ファイルに格納されるジオメトリは、必ずしも 2D オブジェクトとは限りません。形状オブジェクトを構成する各点は、測地データ、または測地データと標高に関連付けることができます。

測地データは、IlvAttributeArray タイプのアトリビュートに格納されます。

以下は、測地データに関連付けられる形状タイプです。

- POINTZ
- POLYLINEZ
- POLYGONZ
- MULTIPOINTZ
- POINTM
- POLYLINEM
- POLYGONM
- MULTIPOINTM

標高は、IlvAttributeArray タイプのアトリビュートに格納されます。

以下は、測地データと標高に関連付けられる形状タイプです。

- POINTZ
- POLYLINEZ
- POLYGONZ
- MULTIPOINTZ

IBM ILOG Views Maps には、3D のレンダリングに不可欠なタイプ MULTIPATCH の形状オブジェクトを描画する定義済みのジオメトリがないため、これらは無視されます。ただし、クラス IlvShapeSHPReader をサブタイプ化してこの振る舞いを変更できます。形状オブジェクトは保護されたメソッドで読み込まれるため、リーダを修正して新しいジオメトリを含める場合も、最小限の作業ですみます。

IlvShapeDBFReader

このリーダは、.dbf 形式のファイルを読み込む場合にのみ使用します。また、次のようなファイルの反復処理に使用できます。

```
IlvShapeDBFReader* reader = new IlvShapeDBFReader("myFile.dbf");
IlvFeatureAttributeProperty* attributes = reader->getNextRecord();
while (attributes) {
   // process attributes
    ...
   attributes = reader->getNextRecord(...);
}
```

.dbf ファイルはアトリビュートを .shp のオブジェクトに逐次関連付けるため、記録番号を直接指定することで地図機能のアトリビュートにアクセスできます。

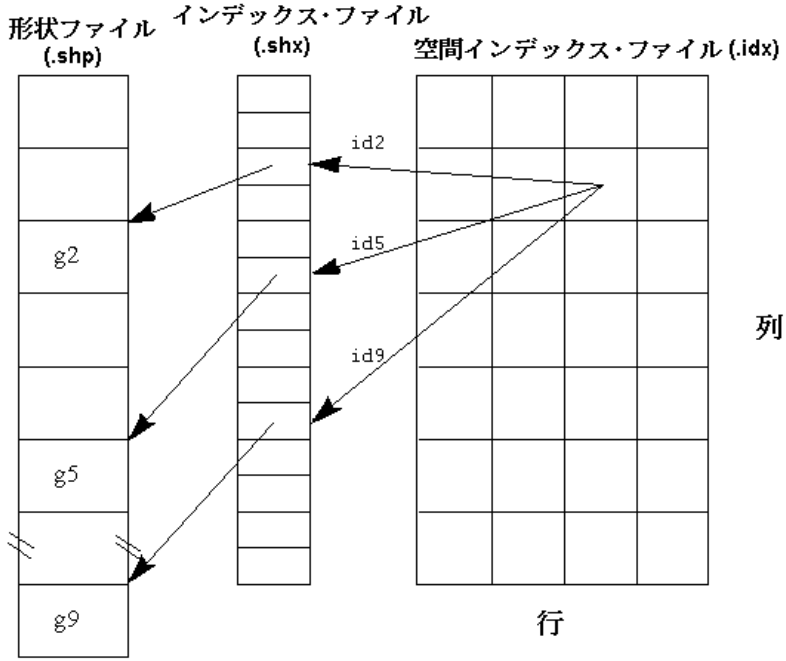
```
reader->readRecord(index, error);
```

形状ファイルのロード・オン・デマンド

Maps モジュールでは、形状ファイルでロード・オン・デマンドを実行するためのクラスが提供されています。ロード・オン・デマンドは、特定の空間インデックス・ファイルを使用して実行します。通常、.idx 拡張子を持つこれらのファイルは、タイルとそれらタイルに属するオブジェクトの識別子間の関係を格納します。用意されたクラスとツールの例を使って、これらの空間インデックス・ファイルを生成することができます。

ロード・オン・デマンド機構では、形状リーダと dbf リーダだけでなく、IlvShapeFileIndex と IlvShapeSpatialIndex の 2 つのクラスも必要です。そして、特定の形状ファイルに空間インデックスを作成するために、IlvShapeFileTiler クラスというユーティリティ・クラスも用意されています。

次の図に、オブジェクトをタイルごとに格納、取得するための機構を示します。



空間インデックス・ファイルには、各タイルのオブジェクト識別子が含まれます。オブジェクトの識別子には、インデックス・ファイル内の順序が含まれます。ジオメトリは、インデックス・ファイルを使って形状ファイルから取得します。上の図では、タイル [2, 1] (0 で始まるタイル・インデックス) には、ジオメトリ g2、g5、g9 を参照する識別子 2、5 および 9 が含まれます。

形状ファイルでロード・オン・デマンドを実行する際に使用するクラスは、以下のとおりです。

- *IlvShapeFileIndex クラス*
- *IlvShapeSpatialIndex クラス*
- *IlvShapeFileTiler クラス*
- *IlvShapeFileTileLoader クラス*
- *IlvShapeLayer クラス*

IlvShapeFileIndex クラス

このクラスを使うと、形状ファイルのジオメトリに直接アクセスできます。空間インデックスと形状ファイルは、同一テーマに対応する必要があります。

```
IlvMapsError status;
// Open the index file.
IlvShapeFileIndex* index =
    new IlvShapeFileIndex(shxFileName);
status = index->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
// Open the corresponding Shapefile.
IlvShapeSHPReader* shape =
    new IlvShapeSHPReader(shpFileName);
status = shape->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
// Construct a reader from the Shapefile reader and the Shapefile index.
IlvShapeFileReader* reader =
    new IlvShapeFileReader(shape, 0, index);
status = reader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;

// Retrieve the feature for each index.
IlInt count = index->getRecordCount();
for(IlInt i = 0; i < count; i++) {
    const IlvMapFeature* feature = reader->getFeatureAt(i, status);
    if(status != IlvMaps::NoError())
        return status;
}
```

IlvShapeSpatialIndex クラス

このクラスは、次のようなタイル情報を格納します。すなわち、タイルのサイズと数、各タイルに属するオブジェクトの識別子情報です。`getIdArray()` メソッドを使って、行と列で指定したタイルからオブジェクトを読み取ります。

```
// Create a reader with the Shapefile name and the index file name.
IlvShapeFileReader* reader =
    new IlvShapeFileReader(shpFilename, 0, shxFilename);
// Open the spatial index.
IlvShapeSpatialIndex* spindex =
    new IlvShapeSpatialIndex(idxFileName);
status = spindex->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
// Loop on all columns and rows.
for(int c = 0; c < spindex->getColumnCount(); c++) {
    for(int r = 0; r < spindex->getRowCount(); r++) {
        IlInt *ret;
        IlUInt size;
        // Retrieve the IDs of objects belonging to the tile
        // at column 'c' and row 'w'.
        status = spindex->getIdArray(c, r, ret, size);
        if(status != IlvMaps::NoError())
            return status;
        // Loop on these IDs and retrieve the corresponding map feature.
        for(int i = 0; i < size; i++) {
            const IlvMapFeature* feature =
                reader->getFeatureAt(ret[i], status);
        if(status != IlvMaps::NoError())
            return status;
      }
      // Free allocated array.
      if(ret)
         delete[] ret;
    }
}
```

また、このクラスを使って、特定の形状ファイルに独自のタイリング情報を生成することもできます。

```
IlvShapeSpatialIndex* tilerIndex = new
    IlvShapeSpatialIndex(getColumnCount(),
                         getRowCount(),
                         getOrigin(),
                         getTileWidth(),
                         getTileHeight());
status = tilerIndex->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvMapFeature* feature = (IlvMapFeature*)reader->getNextFeature(status);
if(status != IlvMaps::NoError())
    return status;
int id = 0;
while(feature) {
    // Determine to which tile(s) the object must belong.
    int row = getRow(feature);
    int col = getColumn(feature);
    // Add it to the spatial index.
    tilerIndex->add(row, col, id);
    feature = (IlvMapFeature*)reader->getNextFeature(status);
    if(status != IlvMaps::NoError())
        return status;
    id++;
}
// Write the spatial index.
tilerIndex->save("spatialIndex.idx");
```

IlvShapeFileTiler クラス

このクラスを使って特定の形状ファイルのタイリング情報を生成します。このクラスを使うには、タイルの形状ファイル、書き込む空間インデックス・ファイル、およびタイル・サイズを指定する必要があります

```
IlvShapeFileTiler::CreateShapeSpatialIndex("example.shp",
                                           "example.idx",
                                            (IlDouble)5, (IlDouble)10);
```

上に抜粋したサンプル・コードでは、幅 5、高さ 10 のタイル・サイズを持つ、example.idx という名前の空間インデックス・ファイルが生成されます。たとえば example.shp ファイルの左上隅が (x = -5、y = 20) で、右下隅が (x = 35、y = -30) ならば、結果のタイル配列は 8 列 X 5 行です。

```
IlvShapeFileTiler::CreateShapeSpatialIndex("example.shp",
                                           "example.idx",
                                           (IlInt)20, (IlInt)30);
```

上のコード抜粋では、600 タイル、20 列、30 行の空間インデックス・ファイルが作成されます。

IlvShapeFileTileLoader クラス

このクラスは、タイル状の形状ファイルにロード・オン・デマンドを実装します。IlvTiledLayer に関連付けると、形状ファイルのファイル名、インデックス・ファイルのファイル名、および空間インデックス・ファイルのファイル名を指定した場合、このクラスによってタイルが自動的にロードされます。また、オプションで Dbase のファイル名を指定して、オブジェクトのアトリビュートを読み込むこともできます。

```
IlvMapAdapter a(0.001);
IlvShapeFileTileLoader* tileLoader =
    new IlvShapeFileTileLoader(shpFileName,
                               dbfFileName, // or null if attribute
                                            // loading is not wanted.
                               shxFileName,
                               idxFileName,
                               &a);
IlvTiledLayer* tiledLayer = new IlvTiledLayer(getTileOrigin());
tiledLayer->setTileLoader(tileLoader);
```

IlvShapeLayer クラス

IlvShapeFileLayer クラスは、1 つの IlvTiledLayer レイヤで、.ilv ファイルに保存できます。特に、このレイヤが .ilv ファイルから読み込まれると、ロード・オン・デマンド・レイヤを正しく再スタートさせるためのあらゆる機構を処理します。

```
IlvMapAdapter a(0.001);
IlvShapeFileTileLoader* tileLoader =
    new IlvShapeFileTileLoader(shpFileName,
                               dbfFileName, // or null if attribute
                                            // loading is not wanted.
                               shxFileName,
                               idxFileName,
                               &a);
IlvShapeFileLayer* shapeLayer = new IlvShapeFileLayer(tileLoader);
IlvManager* manager = new IlvManager(0);
manager->addLayer(shapeLayer);
manager->save(ofstream("out.ilv"));
```

DTED ファイル・リーダ

このセクションでは、DTED 形式のファイルの読み込みを可能にする IlvMaps ライブラリのクラスについて説明します。

以下のトピックから構成されています。

- *DTED 形式の概要*
- *DTED 形式を読み込むためのクラス*

- *デジタル地形モデルのグラフィカル・レンダリング*

DTED 形式の概要

DTED (Digital Terrain Elevation Data) 形式は、米国 NIMA (National Imagery and Mapping Agency) が発行した標高用の地図形式です。DTED ファイルには、デジタル地形モデルがラスターとして含まれています。ラスターは、それぞれのセルに値を含む地図参照されたグリッドです。通常、DTED やデジタル地形モデルでは、値はセルの平均標高を示します。ただし、この値は他の任意のアトリビュート、すなわち 地表温度、地上気圧、土壌の窒素率などを示すこともできます。

DTED ファイルには、1 x 1 度の地域を対象とするデジタル地形モデル・ラスターが含まれます。ラスターのセル・サイズは、DTED レベルによって異なります。

- DTED0 は、生データを提供します (1 ファイルで約 30 ～ 40KB)。
- DTED1 は、より詳細なデータを提供します。
- DTED2 は、もっとも精度の高いレベルです。DTED2 には、DTED1 の表面セルの 9 倍小さい表面セルがあります。この精度の DTED ファイルは非常に大きくなります (数メガバイト)。

DTED 形式を読み込むためのクラス

DTED 形式を読み込むためのクラスは `IlvDTEDReader` です。このクラスは、`IlvMapFeatureIterator` のサブクラスで、ファイルに保存されているデジタル地形モデル (DTM) に対応するラスターの `IlvMapFeature` オブジェクトを 1 つだけ返します。この地図機能のジオメトリは、タイプ `IlvMapRaster` です。この地図機能にアトリビュートはありません。リーダの投影図法は、DTED データ、つまり地理投影のソースの投影図法になります。

`IlvDTEDLayer` クラスは、DTED 形式のロード・オン・デマンドを定義します。ロード・オン・デマンドは、対応するファイル名の DTED レベルに基づいて実装されます。言い換えれば、IBM ILOG Views Maps タイル・レイヤのタイル・サイズは、DTED タイルのサイズに一致します。このロード・オン・デマンドの特有の実装は、地理投影で描画された地図でのみ機能します。例については、次の場所にある DTED のデモを参照してください。

```
<installdir>/samples/maps/dted
```

詳細は 4 章 *ロード・オン・デマンドの使用*を参照してください。

デジタル地形モデルのグラフィカル・レンダリング

`IlvDTEDReader` クラスのデフォルトの *レンダラ*は、標高値にそれぞれ対応する色でラスターをイメージとして表示する `IlvDefaultRasterRenderer` です。このレンダラのデフォルト・パラメータでは、標高表示のため特別に設計された色モデ

ル (IlvIntervalColorModel::MakeElevationColorModel()) を使用します。null の地形標高はクリア・ブルーで描画され、海抜以下は藍色で表示されます。海抜 1m までは黄色で表示されます。その後は低い方から緑色、茶色へと変化し、もっとも標高の高い部分は白で表示されます。

CADRG ファイル・リーダ

このセクションでは、CADRG 形式のファイルを読み込むことができる IlvMaps ライブラリのクラスについて説明します。

CADRG (Compressed ARC Digitized Raster Graphics) 形式は、米国 NIMA (National Imagery and Mapping Agency) が発行したデジタル地図の地図形式です。この地図形式は、デジタル地図作成の条件を満たすように設計されています。この形式の構造は、特にロード・オン・デマンドに適しており、特定の表示縮尺にもっともよく適合する有効範囲を選択することができます。

CADRG のボリュームは、通常 rpf ディレクトリに保存されています。このディレクトリは、有効範囲に対応するサブディレクトリで構成されています。有効範囲ディレクトリには、フレームにそれぞれ対応するファイル群が含まれます。有効範囲は、特定の縮尺の地図に対応し、その縮尺の地図の領域に対応する矩形フレームに分割されています。また、rpf ディレクトリには a.toc という名前の目次ファイルがあり、ボリュームの要素がすべて一覧表示されています。CADRG ボリュームには、有効範囲に示される領域の概要や 1 つ以上の凡例ファイルなど、他の一般情報も含まれます。

通常、CADRG 有効範囲は、正距方位図法の方が適切な北極、南極以外は地理投影範囲にあります。

CADRG 形式を読み込むためのクラス

- IlvCADRGTocReader: このクラスでは、CADRG ボリュームの目次を読み込みます。
- IlvCADRGFrameReader: このクラスは CADRG フレームを読み込みます。
- IlvCADRGLayer: このクラスは、CADRG 形式にロード・オン・デマンドを実装します。

IlvCADRTocReader と CADRG モデル

このクラスを使って、ファイル (a.toc ファイル) の目次を読み込むことができます。以下のオブジェクト・モデルに基づいて、CADRG ボリュームの要素にアクセスできるようになります。

- CADRG 有効範囲は、クラス IlvCADRGCoverage のインスタンスで表します。このクラスは、CADRG 目次で説明した CADRG 有効範囲に関する情報を格納します。
- CADRG フレームは、クラス IlvCADRGFrame のインスタンスで表します。

以下の例は、CADRG ボリュームの目次を表示するものです。

```
IlvCADRGTocReader* tocReader = new IlvCADRGTocReader(fileName);
IlvUShort count;
const IlvCADRGFrame* const* frames = tocReader->getOverViewFrames(count);
for (int i = 0; i < count; i++) {
   IlvMapFeatureIterator* iterator = frames[i]->makeReader();
   mapLoader->load(iterator);
}
```

この例で、mapLoader は IlvMapLoader クラスのインスタンスです。マップ・ローダーの詳細については、66 ページの *マップ・ローダー* を参照してください。

IlvCADRGFrameReader

IlvCADRGFrameReader クラスを使うと、CADRG フレームを直接読み込むことができます。IlvMapFeatureIterator インターフェースを実装します。

次のいずれかの方法で、IlvCADRGFrameReader オブジェクトを作成できます。

- 読み込むフレームの名前で makeReader メソッドを呼び出す。上記の例を参照してください。
- クラス・コンストラクタに、読み込むフレームの名前を指定する。

このクラスは、各 CADRG サブフレーム (CADRG フレームは、6x6 の 36 サブフレームで構成されます) の地図機能を返します。これらのサブフレームのジオメトリは IlvMapImage オブジェクトです。この地図機能にアトリビュートはありません。デフォルトのレンダラは IlvDefaultImageRenderer オブジェクトです。

> ***メモ:*** *このレンダラはイメージを再投影できません。*

IlvCADRGLayer

このクラスは、CADRG 有効範囲にロード・オン・デマンドを実装します。これは IlvCADRGCoverage クラスのインスタンスから作成されます。タイルのサイズは、CADRG フレームのサイズに一致します。タイル・レイヤの実装は、CADRG の極地以外の地理投影範囲内でのみ動作します。4 章 *ロード・オン・デマンドの使用* を参照してください。

イメージ・ファイル・リーダ

このセクションでは、一般的なイメージ・ファイル・リーダについて説明します。このリーダが処理するイメージ形式は、IBM ILOG Views によってサポートされているものです いずれかの形式でコード化されるイメージには、地理参照情報が含まれないため、この種類のイメージを読み込む前に同情報を確認する必要があります。IlvImageReader クラスでは、左上隅と右下隅の座標の指定により、地図にイメージを正しく位置付けできます。

以下のセクションでは、IlvImageReader クラスと IlvImageTileLoader クラスについて説明します。

IlvImageReader クラス

このクラスは、IlvMapFeatureIterator 抽象クラスのサブクラスで、ファイルに格納されたイメージの IlvMapFeature オブジェクトを 1 つだけ返します。この地図機能のジオメトリは、タイプ IlvMapImage です。この地図機能にアトリビュートはありません。このリーダを使用するには、このイメージのファイル名と座標を指定する必要があります。

```
IlvMapsError status;
// The image is known to be at 77 degrees 30 seconds east
// and 10 degrees north for the upper-left corner.
// Lower-right corner is at 82 degrees 30 seconds east
// and 5 degrees north.
IlvCoordinate ul(77.5, 10);
IlvCoordinate lr(82.5, 5);
IlvImageReader* reader = new IlvImageReader(display, fileName, ul, lr);
status = reader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvMapFeature* feature = (IlvMapFeature*)reader->getNextFeature(status);
if(status != IlvMaps::NoError())
    return status;
IlvFeatureRenderer* renderer = reader->getDefaultFeatureRenderer(display);
// Image is known to be in the geographic projection.
IlvGeographicProjection* projection = new IlvGeographicProjection();
IlvMapInfo* mapInfo = new IlvMapInfo(projection);
feature->setProjection(projection);
IlvGraphic* g = renderer->makeGraphic(*feature, *mapInfo, status);
if(status != IlvMaps::NoError())
    return status;
IlvManager* manager = new IlvManager(display);
manager->addObject(g, IlFalse);
return IlvMaps::NoError();
```

IlvImageTileLoader クラス

このクラスを使用して、1 つの大きなイメージの部分にあたるイメージ群を読み込みます。このタイル・ローダーを使うと、特定の時間に表示されるタイルに対応す

るそれぞれのイメージだけをアプリケーションで読み込むことができます。各ファイルには名前を付けて、対応するタイルの行インデックスと列インデックスがわかるようなファイル名を構成する必要があります。このタイル・ローダーを使うには、特定のタイルのファイル名を再構成する際に必要な情報を提供する必要があります。必要な情報は、ファイルの命名方法と 2 つのフォーマット文字列を一致させるパターンです。

```
IlvImageTileLoader loader =
    new IlvMapImageTileLoader(IlvDisplay* display,
                              const char* pattern,
                              const char* rowFormatString,
                              const char* colFormatString,
                              IlvMapAdapter* adapter);
```

pattern 引数には、"%r" と "%c" 交換指定子が必ず 1 つずつ含まれます。%c 変換指定子でタイルの行インデックスを、%c 変換指定子で列インデックスをそれぞれ変換します。これらの変換パラメータは、rowFormatString および colFormatString パラメータにそれぞれ置き換わります。これらのフォーマット用文字列は、列指定子および行指定子として使用される通常の C 関数 printf として解釈されます。行列番号に基づかない命名方式では、IlvImageTileLoader をサブクラス化して、getFileName メソッドをオーバーライドできます。

例

以下のコード抜粋で、列 10 および行 20 にあるタイルをロードすると、タイル・ローダーによって tiles/tile_010_20.jpg というファイルが検索されます。

```
IlvImageTileLoader loader =
   new IlvImageTileLoader(display,
                          "tiles/tile_%c_%r.jpg",
                          "%03d",
                          "%02d",
                          adapter);
```

命名方式が決まると、IlvTiledLayer に関連付けて、タイル・ローダーに IlvImageTileLoader を使うことができます。

```
IlvImageTileLoader* loader =
    new IlvImageTileLoader(display,
                           "tiles/tile_%c_%r.jpg",
                           "%03d",
                           "%02d",
                            adapter);
status = loader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvTiledLayer* tiledLayer = new IlvTiledLayer(getTileOrigin());
tiledLayer->setTileLoader(loader);
```

IlvImageLayer クラス

IlvImageLayer クラスは、1 つの IlvTiledLayer レイヤで、.ilv ファイルに保存できます。特に、このレイヤが .ilv ファイルから読み込まれると、ロード・オン・デマンド・レイヤを正しく再スタートさせるためのあらゆる機構を処理します。

```
IlvImageTileLoader* loader =
    new IlvImageTileLoader(display,
                           "tiles/tile_%c_%r.jpg",
                           "%03d",
                           "%02d",
                           adapter);
IlvMapsError status = loader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvImageLayer* layer = new IlvImageLayer(loader, getTileOrigin());
IlvManager* manager = new IlvManager(display);
manager->addLayer(layer);
manager->save(ofstream("out.ilv"));
```

GeoTIFF リーダ

このセクションでは、GeoTIFF ファイルの読み込みを可能にするクラスについて説明します。このリーダは、IBM ILOG Views Foundation パッケージの IlvTIFFStreamer を使用します。

GeoTIFF 形式

GeoTIFF 形式は、TIFF (Tagged Image File Format) 形式を拡張したものです。TIFF 形式は、ファイルにタグを追加できるイメージ・ファイル形式です。タグは、解像度、ピクセルごとのサンプル数など、ファイルに含まれるイメージに関する情報になります。GeoTIFF 拡張形式では、イメージが表される座標系、その座標系のイメージの場所など、ファイルに含まれるイメージに関する地理情報を提供する特定の地図作成用タグを追加します。

TIFF の正式な仕様は、次のサイトを参照してください。

http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf

GeoTIFF 形式に関する詳細は、次のサイトを参照してください。

http://www.remotesensing.org/geotiff/geotiff.html

IBM ILOG Views Maps に実装された GeoTIFF リーダによって、GeoTIFF ファイルから次の情報を取得できます。

- ピクセル解像度

- イメージの左上隅と右下隅
- イメージ・サイズ
- タイル・サイズ

座標系および投影タグは処理しません。

IlvGeoTIFFReader クラス

IlvGeoTIFFReader クラスは、IlvFeatureIterator 抽象クラスを実装します。getNextFeature メソッドは、IlvMapImage ジオメトリを含む IlvMapFeature を返します。その後、IlvDefaultImageRenderer で TIFF イメージをレンダリングして IlvIcon を生成できます。TIFF リーダでは、引数としてパラメータ、TIFF ファイル名を取ります。

このリーダは、Maps リーダ・フレームワークに準拠するリーダとして使用可能です。

```
IlvMapsError status;
IlvGeoTIFFReader* reader =
    new IlvGeoTIFFReader(filename);
status = reader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
const IlvMapFeature* feature =
    reader->getNextFeature(status);
if(status != IlvMaps::NoError())
    return status;
IlvFeatureRenderer* renderer = reader->getDefaultFeatureRenderer(display);
// Image is known to be in the geographic projection.
IlvGeographicProjection* projection = new IlvGeographicProjection();
IlvMapInfo info(projection);
IlvGraphic* graphic = renderer->makeGraphic(*feature,
                                            info,
                                            status);
IlvManager* manager = new IlvManager(display);
manager->addObject(graphic, IlFalse);
return IlvMaps::NoError();
```

IlvGeoTIFFTileLoader クラス

IlvGeoTIFFTileLoader クラスを使って、必要に応じてタイル状の TIFF ファイルをロードできます。つまり、タイル状の TIFF ファイルの表示部分のみがロードされ、指定時間に表示されます。このタイル・ローダーは IlvTiledLayer と連動し、

任意のタイル・ローダーとして動作します。TIFF リーダではタイル原点などの必要なタイル情報を提供し、特定のタイルを IlvBitmapData として取得できます。

```
IlvMapAdapter adapter(0.001);
IlvGeoTIFFTileLoader* loader =
    new IlvGeoTIFFTileLoader(fileName,
                             &adapter);
IlvMapsError status = loader->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvTiledLayer* tiledLayer = new IlvTiledLayer(getTileOrigin());
tiledLayer->setTileLoader(loader);
```

IlvGeoTIFFLayer クラス

IlvGeoTIFFLayer クラスは、1 つの IlvTiledLayer レイヤで、.ilv ファイルに保存できます。特に、このレイヤが .ilv ファイルから読み込まれると、ロード・オン・デマンド・レイヤを正しく再スタートさせるためのあらゆる機構を処理します。

```
IlvGeoTIFFLayer* layer = new IlvGeoTIFFLayer(loader);
IlvManager* manager = new IlvManager(display);
manager->addLayer(layer);
manager->save(ofstream("out.ilv"));
```

IlvGeoTIFFTiler クラス

IlvGeoTIFFTiler クラスは、既存の TIFF ファイルをタイル状 GeoTIFF ファイルに再エンコードするために使用されます。タイラーの使用法を次の例に示します。

```
IlvGeoTIFFTiler* tiler =
    new IlvGeoTIFFTiler(filename,
                        "out.tif",
                         getTileWidth(),
                         getTileHeight());
IlvMapsError status = tiler->getInitStatus();
if(status != IlvMaps::NoError())
    return status;
IlvTIFFStreamer streamer;
status = tiler->performTiling(streamer);
return status;
```

Oracle Spatial 機能

このセクションでは、Oracle SDO リレーショナル・モデルと、同オブジェクト・モデルの処理が可能になる ilvdbmaps ライブラリの機能を説明します。

Oracle SDO または Oracle Spatial は、Oracle バージョン 7.3 の空間拡張です。この空間拡張は、バージョン 8.0 で Spatial Cartridge に改名され、バージョン 8i でさらに Oracle Spatial に改名されました。

Oracle Spatial を使うと、Oracle データベースに地理参照されているオブジェクトを格納したり、特定の多角形に交差するオブジェクトの一覧の取得などの空間問い合わせを実行できます。

Oracle では、次の 2 つの Oracle Spatial の実装を行っています。

- Oracle 7.3 以降採用されているリレーショナル・テーブルに基づく実装。
- Oracle 8.i 以降採用され、Oracle Spatial のリレーショナル定義も含むオブジェクト・モデルに基づく実装。

`ilvdbmaps` ライブラリは、異なる 2 つのクラス・パッケージを介して、SDO のリレーショナル・オブジェクト・モデルにおけるデータの読み込み / 書き込みをサポートします。

アプリケーションでこれらのユーティリティを使用するには、次が必要です。

- Spatial パッケージを用いた Oracle サーバへのアクセス。さらに、データベース環境の正しいインストールが必要です。たとえば、ORACLE_HOME 変数の設定が不可欠です。
- データを取得し、Oracle データベースに書き込む (ライタ・クラスを使用する場合) ための特定の権限。たとえば、SDO パッケージへの読み書きアクセス権限のある、特別なアカウントも必要です。
- このライブラリの実装では、データベース・アクセス用の IBM ILOG DB Link を使用するため、それに精通している必要があります 特に `IldDbms`、`IldRequest`、`IldNewDbms`、および `IldErrorReporter` クラスを熟知していなければなりません)。
- インストール済みの IBM ILOG DB Link 製品。`ilvdbmaps` ライブラリは IBM ILOG DB Link の動的ロード機能を使用するため、`dbora8(1)` ライブラリの共有バージョンをインストールする必要があります。動的ロード機能の詳細については、IBM ILOG DB Link のユーザ・マニュアルを参照してください。IBM ILOG Views Maps のライセンスで、IBM ILOG DB Link も使用できます。
- SDO を特定用途で使用する場合は、Oracle と IBM ILOG DB Link、特に SQL 言語に精通している必要があります。

リレーショナル・モデル・クラス

このセクションでは、以下のトピックを取り上げます。

- *Oracle Spatial データベースからデータを読み込むためのクラス*
- *Oracle Spatial データベースにデータを書き込むためのクラス*

Oracle Spatial データベースからデータを読み込むためのクラス

Oracle SDO リレーショナル・モデルのリーダ・クラスは次のとおりです。

- Oracle Spatial レイヤ・データを IlvSDOFeatureIterator オブジェクトに変換するための IlvMapFeature オブジェクト。
- リレーショナル Oracle Spatial データにロード・オン・デマンドを実装するための IlvSDOLayer。
- IlvSDOLayer に Oracle 問い合わせを定義するための IlvSDOTileLoader 抽象クラス。サブクラスの IlvDefaultSDOTileLoader (最適化されている) は、IlvSDOLayer によって使用されます。

IlvSDOFeatureIterator

このクラスは、リレーショナル Oracle Spatial レイヤへの SQL 問い合わせの結果からデータを読み込み、それらのデータを IlvMapFeature オブジェクトに変換します。IBM ILOG Views Maps アプリケーションは、このクラスを使って Oracle Spatial データを透過的に処理できます。以下の C++ コードの例では、ROADS_SDOGEOM という Oracle Spatial レイヤからデータを読み込むための問い合わせを、(IBM ILOG DB Link を使用して) 実行します。

```
IlString query = IlString("SELECT * FROM ROADS_SDOGEOM ORDER BY 1, 2, 4");
                                  //keep always the ORDER BY statement
IldDbms* myDbms = IldNewDbms("oracle8", "scott/tiger@myDomain");
IldRequest* resultSet = myDbms->getFreeRequest();
resultSet->execute(query.getValue());
```

問い合わせの結果は、次の 3 つの基準を使って次に示す順序通りに並べられます。

1. GID (Geometric ID)
2. ESEQ (要素シーケンス)
3. SEQ (行シーケンス)

> ***メモ:*** *この順序は、*IlvSDOFeatureIterator *を正常に動作させるために必要です。*

Oracle Spatial レイヤへの問い合わせの ResultSet を使って、IlvSDOFeatureIterator を初期化できますが、SDO のすべての列が resultSet (GID、ESEQ、ETYPE、SEQ およびそれらの座標を定義する列) にある必要があります。

このイテレータが返す機能にアトリビュートはありません。ただし、Oracle Spatial ジオメトリの GID を各機能の識別子として使うと、この識別子を使ってデータベースのその他のアトリビュートを取得できます。IlvMapFeature::getId() メソッドを参照してください。

IlvSDOLayer

このクラスは、リレーショナル Oracle Spatial データ・ソースにロード・オン・デマンドを実装します。デフォルトの実装は、空間インデックスが実行された Oracle Spatial レイヤを使って、Oracle Spatial タイリングと同等のタイリングで内容を読み込みます。

以下の例では、ROADS_SDOGEOM という Oracle Spatial レイヤで IlvSDOLayer を作成します。

```
IldDbms* myDbms = IldNewDbms("oracle8", "scott/tiger@myDomain");
IlvMapAdapter* adapter = new IlvMapAdapter(0.5);
                         // Create an adapter that fits your data.
IlvSDOLayer* layer = new IlvSDOLayer(adapter, myDbms, "ROADS_SDOGEOM");
manager->addLayer(layer);
```

IlvSDOTileLoader

このクラスは、Oracle Spatial データベースからデータを取得する際に追加機能を提供します。これらの機能には、IlvSDOLayer のデフォルトの振る舞いを補完する役割があります。たとえば、Oracle のタイリングとは異なるタイリング定義を行う場合があります。

IlvDefaultSDOTileLoader

このクラスは、IlvSDOTileLoader のサブクラスで、IlvSDOLayer で使用します。このクラスは部分的に最適化されています。たとえば、メソッド setTileGroupingCount() を使って、データベースへの単一問い合わせにグループ化されるタイル数を設定できます。実際に、各タイルは空間問い合わせに対応します。ロード・オン・デマンドを実行するたびに平均 n のタイルがある場合は、すべての n の問い合わせが、データベースに送られる単一問い合わせにグループ化される setTileGroupingCount(n) を使用する必要があります。

> ***メモ:*** *IlvSDOLayer レイヤのロード・オン・デマンドで取得した各 IlvMapFeature で特別な操作を処理する場合、IlvDefaultSDOTileLoader をサブクラス化して getFeatureIterator メソッドをオーバーライドする必要があります。このメソッドでは、getNextFeature メソッドをオーバーライドした IlvSDOFeatureIterator のサブクラスのインスタンスを返す必要があります(この中で、レイヤから返された各 IlvMapFeature で特別な処理を実行できます)。最後に、IlvDefaultSDOTileLoader のサブクラスをレイヤのタイル・ローダーとして設定する必要があります。*

Oracle Spatial データベースにデータを書き込むためのクラス

このセクションでは、リレーショナル Oracle Spatial データベースに地図機能を書き込む IlvSDOWriter クラスについて説明します。

IlvSDOWriter

IlvSDOWriter クラスは、Oracle Spatial (ベクトル・ジオメトリ) のリレーショナル・モデルでサポートされているジオメトリが、機能に含まれる IlvMapFeatureIterator を記述し、それらを次の例のようにデータベースに書き込むことができます。

```
IldDbms* myDbms = IldNewDbms("oracle81", "scott/tiger@myDomain");
IlvSDOWriter* writer = new IlvSDOWriter(myDbms, "MyLayer", 135);
// Create a source feature iterator.
IlvShapeFileReader* reader = new IlvShapeFileReader("foo.shp", 0);
IlvInt geomCount;
// Dump its content to the Oracle layer.
writer->writeFeatureIterator(reader, geomCount);
```

IlvSDOWriter の write メソッドは、機能のアトリビュートを書き込みません。機能のアトリビュートを書き込む場合は、親クラスの writeFeature(feature) メソッドを呼び出した後、IlvSDOWriter の writeFeature メソッドをサブタイプ化できます。

> ***メモ:*** *Oracle Spatial ライタでサポートしているジオメトリは、次のとおりです。* IlvMapPoint、IlvMapLineString、IlvMapPolygon、IlvMapMultiPoint、*および* IlvMapMultiCurve *(複数行文字列用)*、IlvMapMultiArea *(複数多角形用)*。

オブジェクト・モデル・クラス

このセクションでは、以下のトピックを取り上げます。

- *Oracle Spatial データベースからデータを読み込むためのクラス*
- *Oracle Spatial データベースにデータを書き込むためのクラス*

Oracle Spatial データベースからデータを読み込むためのクラス

Oracle SDO オブジェクト・モデルの、リーダ・クラスは次のとおりです。

- IlvObjectSDOFeatureIterator

  Oracle Spatial レイヤ・データを IlvMapFeature オブジェクトに変換する。
- IlvObjectSDOLayer

  オブジェクト Oracle Spatial データにロード・オン・デマンドを実装する。
- IlvSDOTileLoader

  IlvObjectSDOLayer の Oracle 問い合わせを定義する抽象クラス。サブクラスの IlvDefaultObjectSDOTileLoader (最適化されている) は、IlvObjectSDOLayer によって使用されます。

IlvObjectSDOFeatureIterator

このクラスは、リレーショナル Oracle Spatial レイヤへの SQL 問い合わせの結果からデータを読み込み、それらのデータを `IlvMapFeature` オブジェクトに変換します。IBM ILOG Views Maps アプリケーションは、このクラスを使って Oracle Spatial データを透過的に処理できます。以下の C++ コードの例では、ROADS という名前の Oracle Spatial レイヤからデータをロードするための問い合わせを (IBM ILOG DB Link を使用して) 実行します。

```
IlString query = IlString("SELECT * FROM ROADS");
IldDbms* myDbms = IldNewDbms("oracle81", "scott/tiger@myDomain");
IlvObjectSDOFeatureIterator* iterator =
          new IlvObjectSDOFeatureIterator(myDbms,
                                          "SELECT * FROM ROADS",
                                          // the name of the geometries column
                                          "Geometry",
                                          // no Key ID
                                          0,
                                          // the name of the x ordinates column
                                          "X",
                                          // the name of the y ordinates column
                                          "Y"
                                          );
```

Oracle Spatial レイヤへの問い合わせの結果セットを使って `IlvObjectSDOFeatureIterator` を初期化できますが、ジオメトリを含む列がその結果セットの一部として必要です。

このイテレータから返る機能のアトリビュートは、`IlvMapFeature::getAttributes()` メソッドを介して取得できます。実際に、文字列、浮動値または整数として解釈可能なレイヤの列はアトリビュートに変換され、返された地図機能に設定されます。さらに、ID 名で機能イテレータをインスタンス化する場合、その列の値を各地図機能の識別子に設定し、それら機能の識別子を使ってデータベースから追加アトリビュート (該当する場合) が取得できます。`IlvMapFeature::getId()` メソッドを参照してください。この ID は、本ライブラリでは最適化目的で使用されます。イテレータのインスタンス化時に ID 名を指定すると、複数のタイルに関連する「大きな」ジオメトリが 1 度だけ読み込まれます。ID 名を指定しないと、関連する各タイルの `load` メソッドが「大きな」ジオメトリをすべて読み込んでしまいます。

IlvObjectSDOLayer

このクラスは、オブジェクト Oracle Spatial データ・ソースにロード・オン・デマンドを実装します。デフォルトの実装は、空間インデックスが実行された Oracle Spatial レイヤを使って、その内容を読み込みます。

以下の例では、ROADS という Oracle Spatial レイヤで IlvObjectSDOLayer を作成します。

```
IldDbms* myDbms = IldNewDbms("oracle81", "scott/tiger@myDomain");
 // You should create an adapter that fits your data.
IlvMapAdapter* adapter = new IlvMapAdapter(0.5);
IlvObjectSDOLayer* layer =
     new IlvObjectSDOLayer(adapter,
                           myDbms,
                           // The name of the SDO layer
                           "ROADS".
                           // Assume that the layer has only one
                           // geometry column.
                           0,
                           // Width of a tile in the database
                           // coordinate system.
                           1500,
                           // height of a tile in the database
                           // coordinate system.
                           1500,
                           // The name of the x-ordinates column
                           "X".
                           // The name of the y-ordinates column
                           "Y".
                           );
manager->addLayer(layer);
```

x 列名および y 列名の無視により、レイヤを作成することもできます。x および y のデフォルト値は、いずれも 0 です。

```
IlvObjectSDOLayer* layer =
     new IlvObjectSDOLayer(adapter, myDbms, "ROADS", 0, 1500, 1500);
```

この方法ではレイヤの前提として、メタデータ・テーブル (Oracle 8.1.5 では SDO_GEOM_METADATA 、Oracle 8.1.6 では USER_SDO_GEOM_METADATA) の、ROADS レイヤに対応するエントリが次の形状を持つと仮定されます。

```
'ROADS', 'GEOMETRY', SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', -180, 180,
0), SDO_DIM_ELEMENT('Y', -90, 90, 0));
```

また、最初の SDO_DIM_ELEMENT が X 要素、2 番目が Y 要素と仮定されます。

IlvDefaultObjectSDOTileLoader

このクラスは IlvSDOTileLoader のサブクラスで、IlvObjectSDOLayer で使用します。このクラスは部分的に最適化されています。たとえば、メソッド setTileGroupingCount() を使って、データベースへの単一問い合わせにグループ化されるタイル数を設定できます。実際に、タイルはそれぞれ空間問い合わせに対応し、ロード・オン・デマンドを行うたびに平均 n のタイルがある場合、すべ

ての n の問い合わせが、データベースに送られる単一問い合わせにグループ化される setTileGroupingCount(n) を使用する必要があります。

> ***メモ:*** *IlvObjectSDOLayer レイヤのロード・オン・デマンドで取得した各 IlvMapFeature で特別な操作を処理する場合、IlvDefaultObjectSDOTileLoader をサブクラス化して getFeatureIterator メソッドをオーバーライドする必要があります。このメソッドでは、getNextFeature メソッドをオーバーライドした IlvObjectSDOFeatureIterator のサブクラスのインスタンスを返す必要があります (この中で、レイヤから返された各 IlvMapFeature で特別な処理を実行できます)。最後に、IlvDefaultObjectSDOTileLoader のサブクラスをレイヤのタイル・ローダーとして設定する必要があります。*

このクラスのもう 1 つの興味深いメソッドは、setRequestParameters() メソッドです。

たとえば、このメソッドを使ってレイヤの問い合わせで使用する空間演算子を設定できます。デフォルトの演算子は SDO_FILTER です。

図 5.1 は、レベル 2 の固定タイリングを使用する空間レイヤを示します。赤色の矩形は、タイル・ローダーで問い合わせを行った領域です。SDO_FILTER 演算子を使うと (デフォルト)、その赤色の矩形に交差する Oracle Spatial タイルに属するすべてのジオメトリは要求を満たします。図 5.1 では、線、点、三角形、円および四角形などのタイル (2,2) ; (2,3); (3,2) および (3,3) に属するジオメトリをすべて取得します。

赤色の矩形に明示的に交差しないジオメトリ (たとえば、ここでは円形や四角形ジオメトリ) を取得したくない場合もあるかもしれません。その場合は、SDO_RELATE という Oracle の別な空間演算子を使用できます。この演算子は次に示すパラメータで使用します。"querytype=window mask=anyinteract". この場合、取得したジオメトリのすべては赤色の矩形と交差するジオメトリです。たとえば、図 5.1 に示す点、三角、線です。

最後に、SDO_RELATE 空間演算子は SDO_FILTER 演算子よりも遅くなるという点に注意してください。

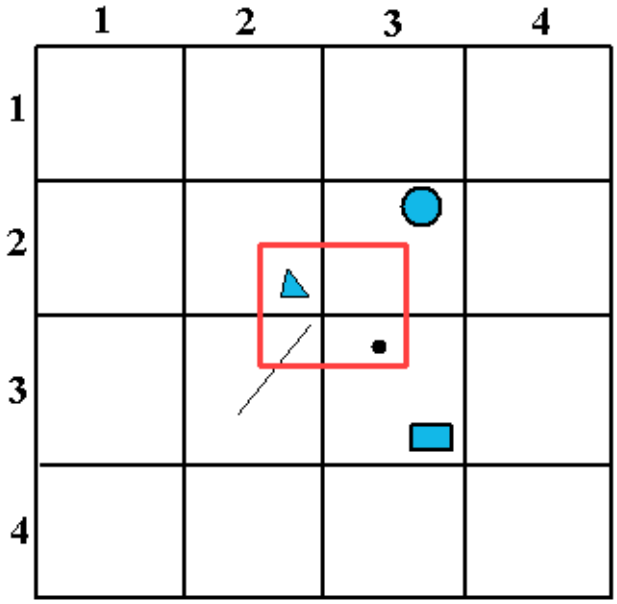


図5.1　タイル

Oracle Spatial データベースにデータを書き込むためのクラス

このセクションでは、オブジェクト Oracle Spatial データベースに地図機能を書き込む `IlvObjectSDOWriter` クラスについて説明します。

IlvObjectSDOWriter

クラス `IlvObjectSDOWriter` は、Oracle Spatial 8i (ベクトル・ジオメトリ) でサポートされているジオメトリを持つ機能の `IlvMapFeatureIterator` を記述し、それらを次の例のようにデータベースに書き込むことができます。

```
IldDbms* myDbms = IldNewDbms("oracle8", "scott/tiger@myDomain");
IlvObjectSDOWriter* writer =
    new IlvObjectSDOWriter(myDbms, "MyLayer", "GEOMETRY", "X", "Y", IlTrue);
// Create a source feature iterator.
IlvShapeFileReader* reader = new IlvShapeFileReader("foo.shp", 0);
// Dump its content to the Oracle layer.
IlvInt count;
writer->writeFeatureIterator(reader, count); // calls close()
```

> ***メモ:*** *Oracle Spatial オブジェクト・モデルの場合、ユーザ・メタデータ・テーブルなどの一部の補助テーブルを更新する必要があります。*`write()` *メソッドでデータを書き込んだときに、メソッド* `IlvObjectSDOWriter::close()` *を呼び出してデータベースを最新に保つようにしてください。*

また、`IlvObjectSDOWriter` の書き込みメソッドは、機能の*アトリビュートを書き込む*ことができます。

メソッド `IlvObjectSDOWriter::writeFeature(IlvMapFeature* feature, IlvBoolean saveAttributes)` の 2 番目の引数は、最初の引数である地図機能のアトリビュートを保存するため、`IlTrue` に設定できます。そのためには地図機能の、SDO レイヤ列名に一致するアトリビュートを表す `IlvFeatureAttributeInfo` が、正しく設定されていなければなりません。また、地図機能には、`IlvFeatureAttributeInfo` に適合し、正しい値を持つ `IlvFeatureAttributeProperty` も必要です。たとえばデータベースに、以下のような記述がある `ROADS` という SDO レイヤのある場合を考えます。

名前	Null?	タイプ
GEOMETRY		MDSYS.SDO_GEOMETRY
TYPE_DESC		VARCHAR2(512)

次に示すように、この方法で地図機能をデータベースに書き込むことができます。

```
IldDbms* myDbms = IldNewDbms("oracle81", "scott/tiger@myDomain");
IlvObjectSDOWriter* myWriter =
    new IlvObjectSDOWriter(myDbms, "ROADS", "GEOMETRY", "X", "Y", IlTrue);
IlvMapFeature* feature = new IlvMapFeature();
// Construction of the IlvFeatureAttributeInfo: it can be done just once.
IlvMapClassInfo** attributeClasses = new IlvMapClassInfo*[1];
IlvBoolean* nullable = new IlvBoolean[1];
nullable[0] = IlTrueIlTrue;
attributeClasses[0] = IlvStringAttribute::ClassInfo();
char** names = new char*[1];
names[0] = new char[10];
//Exactly the same name as the layer column name.
strcpy(names[0], "TYPE_DESC");
IlvFeatureAttributeInfo* info =
    new IlvFeatureAttributeInfo(1, names, attributeClasses, nullable);

// The writing itself.
IlvFeatureAttribute** attributes = new IlvFeatureAttribute*[1];
attributes[0] = new IlvStringAttribute("MY FOO TYPE");
IlvMapsError error;
IlvFeatureAttributeProperty* prop =
    new IlvFeatureAttributeProperty(info, attributes, error);
feature->setAttributeInfo(info);
feature->setAttributes(prop);
if (error == IlvMaps::NoError())
    error = myWriter->writeFeature(feature, IlTrue);
```

ライタは、SDO レイヤの行を更新できます。これは、指定のキーの値がある行を更新するキー機構をベースとしています。この更新は、次のメソッドで実行されます。

- `updateFeatureAttributes(IlvFeatureAttributeProperty* attributes, IlvUInt keyPos)` は、アトリビュート・プロパティに基づいていますが、そこではアトリビュート・リストにキーの場所を指定する必要があり、同時に複数の列を更新できます。
- `updateFeatureAttribute(const char* keyColumnName, IlvFeatureAttribute* keyAttribute, const char* attributeColumnName, IlvFeatureAttribute* attributeToUpdate)` では、キー・アトリビュートを指定した列 (新しい値は引数として渡された `attributeToUpdate`) を 1 つだけ更新できます。

> ***メモ:*** *オブジェクト・ライタでは、*`IlvMapText`、`IlvMapImage`、*および* `IlvMapRaster` *以外の* `IlvMapGeometry` *のサブクラスをすべてサポートしています。*

S57 マップ・リーダ

IBM ILOG Views Maps には、S57 ファイルの読み込みに使用するクラスが含まれています。*S57 形式*は国際水路機関 (IHO) が発行した基準である海里マップ用の数値地図形式です。詳細は、次のサイトをご覧ください。

`http://www.iho.shom.fr/`

このパッケージで提供される S57 リーダは、IHO TRANSFER STANDARD FOR DIGITAL HYDROGRAPHIC DATA Edition 3.1 に基づいています。

S57 リーダ・モジュールを使用すると、IHO S57 形式のファイル・セットのデータにアクセスすることができます。また、S57 リーダ・モジュールは、1 つ以上の関連する S57 データ・ファイル内で S57 機能を生成します。S57 データセットはディレクトリ、カタログまたは個々のデータ・ファイルになることができます。ディレクトリの場合、ディレクトリ内のすべての S57 ファイルが選択されます。カタログの場合はカタログから参照されている全ファイルが選択されます。S57 カタログは、海里データの領域を網羅します。カタログ・ファイル (.030 または .031) およびセル・ファイル (.000) の両方を含んだ単一のディレクトリから構成されています。通常、セルにはカタログのゾーン全体のサブゾーンのみを対象とするデータが含まれています。

S57 機能オブジェクトは、機能に変換されます。S57 ジオメトリ・オブジェクトは自動的に収集され、機能上のジオメトリに変換されます。

S57 シンボルは、オブジェクトのアトリビュートに関係なく、オブジェクト・タイプごとに定義済みのアイコンを使ってレンダリングされます。

デフォルトの S57 レンダラは定義済みのスタイルを使ってポリライン、多角形、および小さいアイコンとして地図機能を表します。これらのスタイル設定は、設定ファイルで行うことができます。たとえば、S57 コードに基づいて各機能の色、線の種類、アイコン、可視性などを指定することができます。独自のレンダラの実装を作成して、同じ S57 データを異なる形式で表示することもできます。

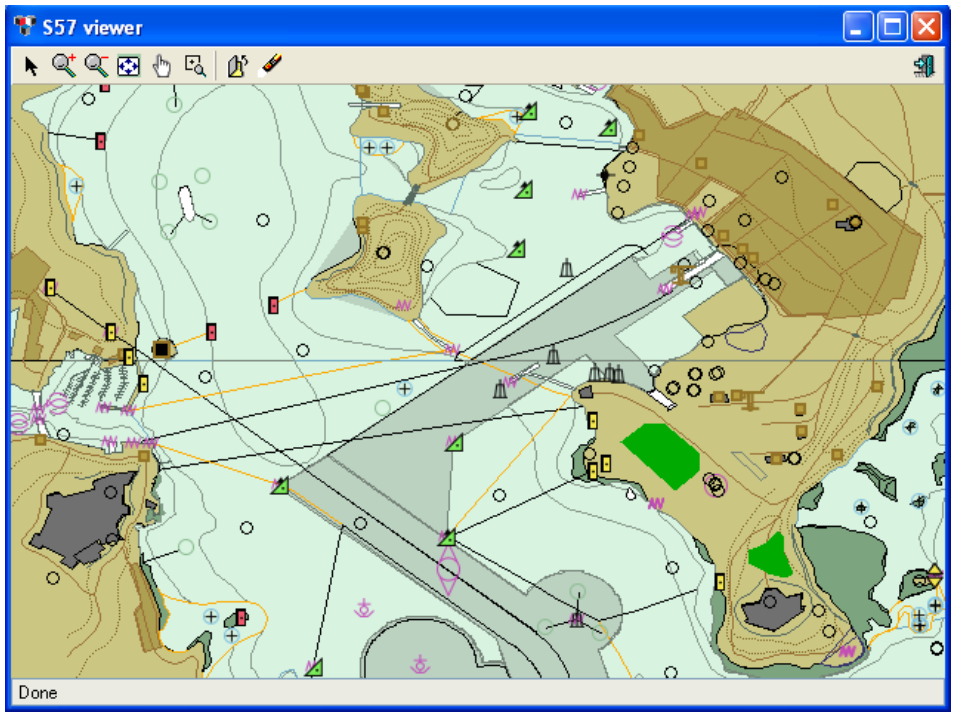


図5.2　S57 地図レンダリングの例

S57 形式を読み込むためのクラス

S57 形式を読み込むためのクラスは `IlvS57Loader` です。このクラスは `IlvMapFeatureIterator` のサブクラスで、各 S57 機能 (S57 FRID 記録) ごとに 1 つの `IlvMapFeature` オブジェクトを返します。

次のコードは、`IlvS57Loader` クラスを使用して S57 カタログ・ファイルを読み込む方法と、`IlvS57Renderer` クラスを使用して地図機能をグラフィック・オブジェクトに変換する方法について示しています。

```
IlvDisplay* display = ...;
IlvGraphic* graphic;
IlvManager* m = ...;
const IlvFeatureAttributeProperty* ap;
IlvMapInfo* mapInfo = ...;
Const char* filename = "catalog.030";
IlvMapsError status = IlvMaps::NoError();

IlvS57Loader reader(display);
reader.setFilename(filename);
IlvFeatureRenderer* renderer =
  reader.getDefaultFeatureRenderer(display);
```

```
for (const IlvMapFeature* f = reader.getNextFeature(status);
     status == IlvMaps::NoError() && f ;
     f = reader.getNextFeature(status)) {
      graphic = renderer->makeGraphic(*f, *mapInfo, status);
      if (graphic) {
          ap = f->getAttributes();
          m->addObject(graphic);
          if (ap)
              graphic->setNamedProperty(ap->copy());
      }

}
```

スタイル、色およびアイコンの設定

前述されているように、`S57Styles.txt` と呼ばれる設定ファイルを使って S57 コードに基づいて各地図機能を表すスタイルを指定することができます。例 :

```
1,Administration Area (Named),T,BBD2C1,,T,T,0,
2,Airport/airfield,T,aea052,997035,T,T,0,airare02.png
3,Anchor berth,T,,,T,T,0,achbrt07.png
4,Anchorage area,T,,,F,T,0,achare02.png
5,Beacon (cardinal),T,,,T,T,0,bcncar02.png
.
.
.
```

このファイルの各行は、コーディング・アトリビュートにしたがって特定の S57 機能を表す方法を示しています。各行にはコンマ区切りのフィールドが含まれており、次のような構造をしています。

- S57 コードの値
- S57 機能タイプの名前
- 可視性のアトリビュート (T: 表示、F: 非表示)
- 背景色 (16 進法の RGB)
- 前景色 (16 進法の RGB)
- 背景色で塗りつぶす領域のブール型 (T: True、F: False)
- 線を描画するブール型 (T: True、F: False)
- 線のスタイルを指定する整数
- ポイント機能のアイコン・ファイル名

利用できる線のスタイルは、次のとおりです。

```
0   solid
1   dot
```

```
2   dash
3   dashdot
4   dashdoubledot
5   alternate
6   doubledot
7   longdash
```

> ***メモ:*** *異なるスタイルの画像は、リファレンス・マニュアルの* `IlvLineStyle` *クラスをご覧ください。*

`<$ILVHOME>/data/maps/s57` に格納されているこのファイル (`S57Styles.txt`) は編集することができます。また、環境変数 `ILVMAPSS57STYLES` で指定する別のディレクトリに独自のバージョンの `S57Styles.txt` ファイルを作成することもできます。ただし、この方法を利用する場合は、新しいディレクトリに `S57Styles.txt` で定義されているすべての必要なビットマップが含まれている必要があります。構文については、元のファイルまたは上記を参照してください。

このように、この設定ファイルでシンプルでフレキシブルな構文を用いて全体的なレンダリングの設定を簡単にカスタマイズすることができます。独自のレンダリング・ストラテジを定義してさらに詳細な設定も行うことができます。この場合、標準 S57 レンダラ (`IlvS57Renderer` クラス) から派生することができる独自のレンダラを作成し、独自のロジックを実装して S57 情報を適切なグラフィックに変換する必要があります。たとえば、いくつかの S57 機能を特定の地図レイヤ上でまとめてグループ化したり、他の表示基準に基づいて詳細なレンダリングを実装したりすることができます。

章

6

地図投影図法

この章では、地図投影図法を紹介するとともに、地図作成アプリケーションによる IBM® ILOG® Views Maps 投影図法パッケージの使用法について説明します。

以下のトピックから構成されています。

- *地図投影図法の概要* では、地図投影図法の概略を説明します。
- *投影データ例* では例を挙げて、地理データをデカルト座標系に投影する方法を説明します。
- *投影メソッドとパラメータ* では、この地図ライブラリで使用できるデータ投影メソッドおよび関連パラメータについて説明します。
- *楕円* では楕円投影図法を紹介し、定義済みの楕円または特定の楕円を投影図法に関連付ける方法を説明します。
- *単位コンバータ* では、単位コンバータについて説明します。
- *測地原点が異なる座標間の変換*
- *インポートした地図の上にグラフィック・オブジェクトを追加する* では、IBM ILOG Views マネージャへの地図のインポート方法および、グラフィック・オブジェクトの追加方法を紹介します。
- *新しい投影図法の作成*では、新しい投影図法の作成方法について説明します。

地図投影図法の概要

地図は、地球や地球の一部を紙面上やコンピュータ画面上などの平面に投影したものです。地球は楕円形をしているため、よく「球体」と表現され、ポイントを平面に投影して地球を描写しようとすると、投影中心から離れた領域に必ずひずみが生じます。したがって、距離、形、方位など地球のすべての特徴を 1 つの地図に忠実に表すことは不可能です。ひずみを最小限に抑えるために、永年にわたってさまざまな種類の投影図法が開発されてきました。ある投影図法では距離を、他の投影図法では形や角度を忠実に再現します。地図を作成する際には、描写する地域や各自のアプリケーションの設計目的に最適な投影図法を選択する必要があります。

投影図法は、次の 3 つに分類することができます。

- *円筒図法*
- *円錐図法*
- *方位図法*

また、次のように分類することもできます。

- *正積図法または正角図法*

円筒図法

円筒図法は、地球に大きな平面を巻いて円筒を作ることで得られます。次の図では、円筒は赤道に接します。接平面に近いほど、ひずみが少なくなります。

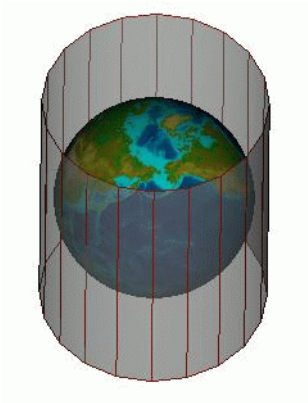


図 6.1 *円筒図法 (1)*

円筒の位置は変更可能です。たとえば、横軸円筒図法では、円筒は経線に接します。

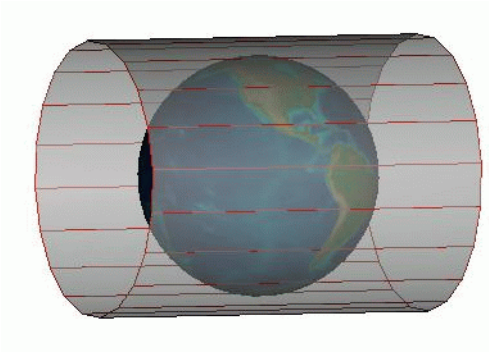
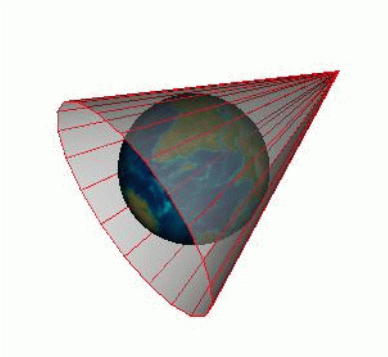


図 6.2 *円筒図法 (2)*

円錐図法

円錐図法では、図 6.3 のように、地表に交差または接するように地球のイメージを円錐に転移します。



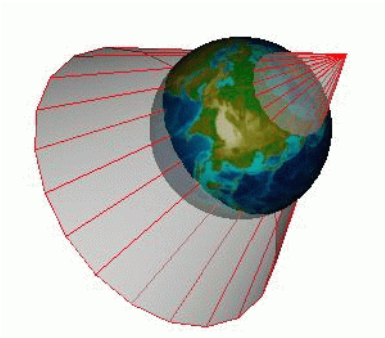


図 6.3　円錐図法の例

方位図法

2 次元投影図法とも呼ばれる方位図法では、球体の地球を平面に投影します。

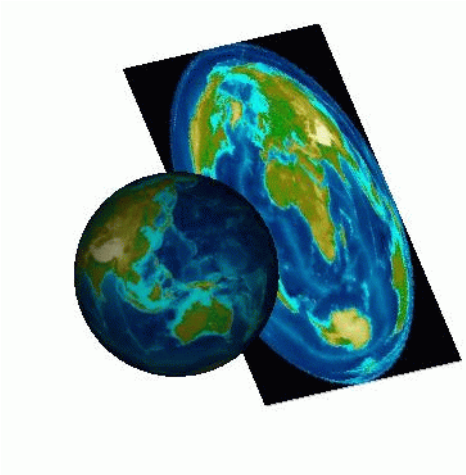


図 6.4　方位図法

正積図法または正角図法

どの地図投影図法でも、投影中心から離れた場所にひずみが生じます。使用する投影図法によりますが、角度、面積、形、寸法、距離、縮尺などでひずみが生じます。この点で、投影図法は主に正積または正角の 2 つに分類されます。

- 正積図法では、地図に表すさまざまな地域間の比率を忠実に保持します。
- 正角図法では、角度を保持し、局所的に形も保持します。

他の投影図法には、投影中心から測定した距離の保持 (正距方位図法) などの注目すべき特性があります。また、他の投影図法では、角度のひずみと面積のひずみをうまく解決しています。

したがって、投影図法は、描写する地域 (たとえば、メルカトル図法で極地は描写できません) と、投影図法の使用分野 (航海または航空路用アプリケーション、小縮尺または大縮尺地図など) に従って、設定し選択する必要があります。たとえば、航海用アプリケーションでは、通常、正角図法を使用します。

このパッケージで提供される投影図法は、Gerald I. Evenden による `proj` プログラムから派生しています。

投影図法の詳細については、次の参考文献があります。

- *Map Projections - A Working Manual* (Snyder, 1987)
- *An Album of Map Projection*s (Snyder and Voxland, 1989)

投影データ 例

ここでは、投影図法を使用する場合に必要な基本操作を示す簡単なサンプルを紹介します。投影図法の作成方法と、平面上にある 1 地点のイメージを計算する方法について説明します。

この例は、以下の手順で構成されます。

- *サンプル・アプリケーションの実行*
- *投影図法宣言を含める*
- *Main 関数*
- *投影図法の初期化*
- *投影データの作成*
- *データの投影*
- *投影結果の出力*
- *逆投影の計算*
- *地理座標の印刷*
- *サンプル・コード一式*

サンプル・コード一式

このセクションで基本としている例のコード一式は、以下のファイルにあります。

`<installdir>/samples/maps/userman/src/useproj.cpp`

サンプル・アプリケーションの実行

このサンプルの全体と、各部の詳細な説明は、134 ページの *サンプル・コード一式* を参照してください。

このアプリケーションのコードは、<installdir>/samples/maps/userman/src/useproj.cpp ファイルとして提供されます。コンパイルすればアプリケーションを実行できます。

サンプルをコンパイルする場合は、

4. ディレクトリ <installdir>/samples/maps/userman/<platform> まで移動します。
5. ILVHOME 変数を IBM® ILOG® Views インストール・ディレクトリに設定します。
6. make (UNIX® プラットフォームの場合) または nmake (Microsoft® Windows® プラットフォームの場合) を使ってコンパイルします。このユーザ・マニュアル用のサンプルすべてが、これによってコンパイルされます。
7. useproj アプリケーションを起動します。

プログラムの出力は次のようになります。

```
The projection of 45W 30N is
x = -5003769 m
y = 3499627 m
The inverse projection is
45DW 30DN
```

後続のセクションで、useproj.cpp から抽出したコードについて説明します。

投影図法宣言を含める

IBM® ILOG® Views Maps 投影図法ライブラリを使用するには、投影図法クラスを宣言するヘッダー・ファイルを含める必要があります。ここに挙げるサンプルでは、<ilviews/maps/projection/mercator.h> ヘッダー・ファイルで宣言するメルカトル図法クラスだけを使用します。

このライブラリで定義されたすべての投影図法クラスを含むヘッダー・ファイルもあります。

```
<ilviews/maps/projection/allprojs.h>
```

Main 関数

サンプル・プログラムの main 関数は、メルカトル図法を初期化し、順投影および逆投影が実行できる関数を呼び出します。

投影図法の初期化

サンプルでは、IlvMercatorProjection クラスのインスタンスを作成します。このタイプの投影図法は、航海用アプリケーションで多用されます。

このライブラリにある投影図法はすべて、IlvProjection 抽象クラスからの継承で、同じ API の利点が得られます。このため (次のサンプルに示すように)、任意の投影図法を引数として取り、IlvProjection クラスから継承するその投影図法が提供できる showProjection 関数を呼び出すことができます。この showProjection 関数は、ライブラリにある関数すべてのために呼び出せます。

```
IlvMercatorProjection projection;
showProjection (projection);
```

showProjection 関数の署名は次のとおりです。

```
void showProjection (const IlvProjection& projection);
```

投影データの作成

この投影図法ライブラリでは、倍精度で計算が実行されます。このため、2 つの座標のベクトルが含まれる IlvCoordinate クラスのインスタンスを作成する必要があります。このクラスの値は、緯度 lambda と経度 phi です。これらの値は、ラジアンで表されます。度数をラジアンに変換するため、スタティック変換関数 IlvMaps::DegreeToRadian を使用します。

```
double lambda = IlvMaps::DegreeToRadian(-45.);
double phi = IlvMaps::DegreeToRadian(30.);
IlvCoordinate ll(lambda, phi);
```

> ***危険:*** *コードの最終行にある 2 文字 (*ll*) は、アルファベットの小文字 (ll) で、数字 (11) ではありません。次のセクションに示すコードでも同様です。*

データの投影

データを投影すると、地理座標 (経度と緯度) は投影座標に変換されます。以下の理由のいずれかによって、この操作は不可欠です。

- 既存の地図またはそれ自体が投影済みのイメージに、グラフィック要素の位置を合わせるため
- 特に選択した投影図法のプロパティ (角度、面、中心点からの距離の保存) を利用するため

データを投影するため、選択した投影図法の forward メンバ関数を呼び出します。この関数は結果を、2 番目の引数 (サンプルでは xy) に置きます。一部の投影図法は全地表で使用できないため、この関数は考慮の必要なエラー・コードも返します。たとえばメルカトル図法では、北極および南極に接近したポイントの投影はできません。エラーが発生した場合、xy の値は使用できません。

```
IlvCoordinate xy;
IlvMapsError status = projection.forward(ll, xy);
    if(status != IlvMaps::NoError())
        IlvPrint("Projection exception for this data : %s",
                 IlvMaps::GetErrorMessageId(status));
```

エラーが発生した場合、エラー・メッセージは IlvMaps のエラー管理機能によって解釈できます。

ここでは、IBM ILOG Views メッセージを返す IlvMaps::GetErrorMessageId を使用します。

上のサンプルでは、メッセージの出力に IBM ILOG Views の IlvPrint 関数を使用しています。この関数は、純粋なグラフィック環境 (Microsoft® Windows® アプリケーションの場合) にも、またコンソール対応環境 (UNIX® アプリケーションの場合) にも移植可能な、メッセージの表示方法を提供します。

投影結果の出力

投影結果は xy 変数に格納されます。以下のサンプルでは、結果はデフォルト測定単位のメートルで表されています。座標は、投影中心からの距離を示します。この距離は、投影中心 (0E 0N) とポイント 45W 30N 間の実距離とは大幅に異なります。また、メルカトル図法では中心からの距離が保持されません。一方、正距方位図法では距離が保存されます。

```
    IlvPrint("The projection of 45W 30N is \n"
             " x = %d m\n"
             " y = %d m",
             (int) xy.x(),
             (int) xy.y());
```

逆投影の計算

逆投影では、投影データが経度と緯度に変換されます。この操作によって、たとえば、投影した地図にユーザがマウスで示したポイントの地理座標を決定できます。

このサンプルでは、計算した座標に逆投影を実行します。(ここで知りたいのは経度と緯度の初期値です)。

逆投影の計算では、正確を期してあらかじめ座標 ll を 0 にリセットした上で、選択した投影図法の inverse メンバ関数を呼び出します。

> ***注記:*** *コードにある 2 文字 (ll) は、アルファベットの小文字 (ll) で、数字 (11) ではありません。次のセクションに示すコードでも同様です。*

```
ll.move(0., 0.);
status = projection.inverse(xy, ll);
if(status != IlvMaps::NoError())
        IlvPrint("Projection exception for this data : %s",
        IlvMaps::GetErrorMessageId(status));
```

地理座標の印刷

地理座標を解釈可能な形式で印刷するために、IlvMaps クラスのスタティック関数 RadianToDMS を使用します。これによって、IlvCoordinate が、度、分、秒を含む文字列に変換されます。

```
    char buffer1[12];
    char buffer2[12];
    IlvPrint("The inverse projection is \n"
            " %s %s",
             IlvMaps::RadianToDMS(buffer1, ll.x(), IlFalse),
             IlvMaps::RadianToDMS(buffer2, ll.y(), IlTrue));
```

サンプル・コード一式

```
#include <ilviews/maps/projection/mercator.h>
void
showProjection(const IlvProjection& projection)
{
    double lambda = IlvMaps::DegreeToRadian(-45.);
    double phi = IlvMaps::DegreeToRadian(30.);
    IlvCoordinate ll(lambda, phi);
    IlvCoordinate xy;
    // Forward projection (from long/lat to x/y).
    IlvMapsError status = projection.forward(ll, xy);
    if(status != IlvMaps::NoError())
        IlvPrint("Projection exception for this data : %s",
                 IlvMaps::GetErrorMessageId(status));
    // Printing the result.
    IlvPrint("The projection of 45W 30N is \n"
             " x = %d m\n"
             " y = %d m",
             (int) xy.x(),
             (int) xy.y());
    // Resetting ll.
    ll.move(0., 0.);
    // Inverse projection (from x/y to long/lat).
    status = projection.inverse(xy, ll);
    if(status != IlvMaps::NoError())
        IlvPrint("Projection exception for this data : %s",
                  IlvMaps::GetErrorMessageId(status));
```

```
    // Printing the result.
    char buffer1[12];
    char buffer2[12];
    IlvPrint("The inverse projection is \n"
             " %s %s",
              IlvMaps::RadianToDMS(buffer1, ll.x(), IlFalse),
              IlvMaps::RadianToDMS(buffer2, ll.y(), IlTrue));}
int
main(int , char**)
{
    IlvMercatorProjection projection;
    showProjection(projection);
    return 0;
}
```

投影メソッドとパラメータ

このセクションでは、投影メソッドとパラメータについて説明します。以下のトピックから構成されています。

- *順関数と逆関数*
- *投影図法パラメータ*
- *ユーティリティ*

順関数と逆関数

投影図法は、`forward` および `inverse` 関数を使って実装します。

- `forward` 関数は、経度と緯度をデカルト座標に変換します。
- `inverse` 関数は、デカルト座標を緯度と経度に変換します。

これらの関数は、成功すると `IlvMaps::NoError()` コードを返します。その他の場合は、失敗の原因を示す適切なエラー・コードが返ります。

`IlvProjection::UnsupportedFeatureError()` は、実装されていない機能が呼び出された場合に返されます。これは以下の操作が原因です。

- 投影図法で非球形の楕円がサポートされていないのに、非球形の楕円で順投影を実行しようとした。例 :`IlvEquidistantCylindricalProjection`
- 逆にできない投影図法を、逆方向に実行しようとした。

使用中の投影図法にこれらの機能が実装されているかどうかを知るために、関数 `IlvProjection::isEllipsoidEnabled` および `IlvProjection::isInverseEnabled` を呼び出します。

その他のエラー・コードは、たとえば投影図法が定義されていない座標に対してこれらの関数が使用された場合に、計算中のエラー発生時などに返されます。エ

ラー・コードは、IlvMaps::GetErrorMessageId などの IlvMaps エラー管理関数によって解釈できます。

投影図法パラメータ

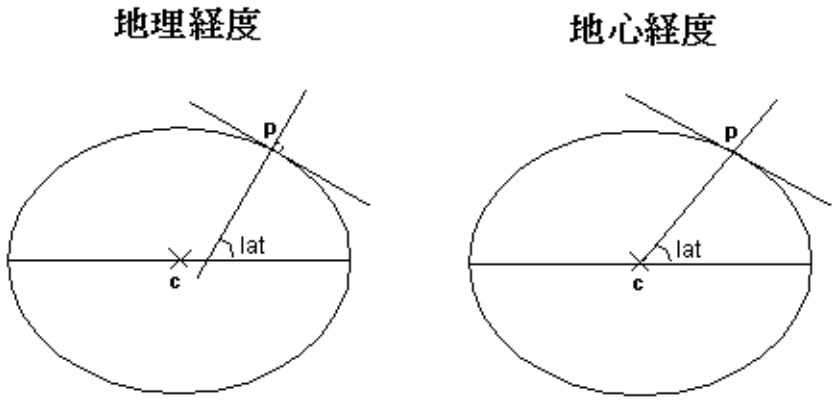
投影図法には、下記のパラメータを設定できます。

- 地球の形状を指定する楕円 楕円については、137 ページの*楕円*で説明します。
- デカルト座標の測定単位を指定する単位コンバータ 単位コンバータについては、141 ページの*単位コンバータの直接使用*で説明します。
- 投影図法の中央経線と中央緯線 これらのパラメータは、setLLCenter 関数で設定します。投影図法では中心に近いほどひずみが減ります。
- デカルト座標に適用され、false easting および false northing とも呼ばれるオフセット。これらのパラメータは、setXYOffset 関数で設定します。

次のような処理も行えます。

- setGeocentric 関数による、座標が地理 (デフォルト値) か地心かの指定。

  あるポイントの地心緯度は、地球と赤道面の中心にポイントをつなぐ線で作られる角度によって定義されます。一方、ポイントの測地 (または地理) 緯度は、このポイントと赤道面を通過する垂直線で作られる角度によって定義されます。地球が正確な球体ではなく、どちらかと言うと楕円であるため、これら 2 つの値は異なります。両方の緯度は、tan phiG = (1 - e ^ 2) tan phi 関係を通じて関連しています。e は、地球をかたどる際に使う楕円の偏りです。



アプリケーションで地心データを処理する場合、このパラメータを設定する必要があります。使用可能な地理データのほとんどは、地理緯度で表します。

- 投影図法で経度修正を使うかどうかを指定します。つまり、経度を範囲 [-PI;PI] 内に収めるか、または setUsingLongitudeReduction 関数を使って、任意の経度を受け入れるか指定します。

上記のパラメータは、すべての投影図法に共通のものです。これらのパラメータは、ライブラリにあるすべての投影図法の基本クラスである、IlvProjection クラスの API で設定します。また、一部の投影図法には、特定の追加パラメータがあります。たとえば、円錐図法を交差緯度に指定したり、実縮尺の緯度をほとんどの円筒図法に指定できます。詳細は、各投影図法のリファレンス・マニュアルを参照してください。

ユーティリティ

IlvMaps クラスでは、ラジアンを度数に変換したり、その逆変換を行う変換ユーティリティが提供されます。

楕円

このセクションでは楕円と、それが地図投影図法にどのように関係するかを説明します。IBM® ILOG® Views Maps 投影図法パッケージで提供されている、定義済み楕円の一覧表も掲載されています。

このセクションでは、以下のトピックを取り上げます。

- *楕円の概要*
- *投影図法に楕円を関連付ける*
- *新しい楕円の定義*
- *定義済みの楕円*

楕円の概要

楕円を使って地球の形を表します。多くのアプリケーション、特に小縮尺マッピングでは、地球を球形で表すことができます。本パッケージで提供されているほとんどの投影図法では、デフォルトで、地球を半径約 6371 キロメートルの球体としています。ただし、地球は自転しているため両極上では若干平らで、極軸を中心に回転する楕円で近似させています。楕円投影図法は、正確な大縮尺地図と明瞭な座標系用に使用します。ただし、非常に大きな縮尺の地図で大陸や地球全体を表す場合には、球面投影図法の使用をお勧めします。本パッケージのほとんどの投影図法で使う楕円は、投影中心周辺の数度の緯度や経度に対してのみ正確です。

投影図法に楕円を関連付ける

各投影図法には、楕円が関連付けられています。デフォルトで、ほとんどの投影図方が楕円 `IlvEllipsoid::SPHERE` を使用します。ユニバーサル横メルカトルやユニバーサル極ステレオ (UPS) といった一部の特定投影図法だけが、デフォルトで非球形の楕円を使用します。

適切な楕円を使うと、特に大縮尺地図に関する限り、より正確な投影図法が得られます。ただし、球形を使う場合よりも計算が複雑になり、動作が遅くなります。

投影図法に使用する楕円を指定するには、メソッド `IlvProjection::setEllipsoid` を使います。

```
IlvProjection* projection = new IlvMercatorProjection();
projection->setEllipsoid(*IlvEllipsoid::WGS84());
```

通常使用する多くの楕円が定義されている `IlvEllipsoid` クラスのスタティック・メンバと、`IlvEllipsoid::GetRegisteredEllipsoid()` スタティック・メソッドのどちらも、登録された楕円の取得に使用できます。また、次の 138 ページの*新しい楕円の定義*で説明するように、独自の楕円も作成できます。また、139 ページの*定義済みの楕円*にある一覧表の中から定義済み楕円の 1 つを選択して使用することもできます。

新しい楕円の定義

本パッケージにある楕円は、次の 2 つのパラメータで定義されています。

- 楕円の赤道半径と長半径
- 楕円の扁平率。

  楕円の扁平率が null の場合、楕円は球形です。

球面楕円の定義

パラメータを 1 つだけ指定した場合、楕円は球形であると想定されます。以下の例では、半径 10m の球形を定義します。

```
IlvEllipsoid ellipsoid = new IlvEllipsoid(10.0);
```

ほとんどの地図作成アプリケーションでは、地球の寸法に非常に近い球形を定義する楕円 `IlvEllipsoid::SPHERE()` を使用しています。通常、選択された楕円は、表す領域を可能な限り地球の実際の形に近づける必要があります。球形の半径はメートルで表します。

以下の例では、赤道半径 10m、扁平率 0.0067 の楕円を定義します。

```
IlvEllipsoid* ellipsoid = new IlvEllipsoid(10, 0.0067);
```

扁平率以外のパラメータを指定する場合、`IlvEllipsoid` クラスで提供されている変換メソッドを使用できます。

以下の例では、赤道半径 10m、極半径 9m の楕円を定義します。

```
IlvEllipsoid* ellipsoid =
    new IlvEllipsoid(10,IlvEllipsoid::ESFromPolarRadius(10.0,9.0));
```

極半径は、ESFromPolarRadius メソッドで扁平率に変換されます。

クラス IlvEllipsoid には、次に示す極半径と直線化の変換メソッドがあります。

- IlvEllipsoid::ESFromPolarRadius
- IlvEllipsoid::ESFromFlattening

定義済みの楕円

本パッケージには、定義済み楕円の一覧が含まれます。定義済み楕円はその名前で参照されます。定義済み楕円へのアクセスには、IlvEllipsoid クラスの GetRegisteredEllipsoid スタティック・メソッドを使用します。

例 :

```
const IlvEllipsoid* ellipsoid =
   IlvEllipsoid::GetRegisteredEllipsoid("clrk66");
IlvProjection* projection = new IlvMercatorProjection();
projection->setEllipsoid(*ellipsoid);
```

以下の表は、使用可能な定義済み楕円の一覧です。

名前	説明	長半径	扁平率
sphere	Sphere of 6370997 m	6370997.0	0.0
MERIT	MERIT 1983	6378137.0	0.0818192
SGS85	Soviet Geodetic System 85	6378136.0	0.0818192
GRS80	GRS 1980(IUGG, 1980)	6378137.0	0.0818192
IAU76	IAU 1976	6378140.0	0.0818192
airy	Airy 1830	6377563.396	0.0816734
APL4.9	Appl. Physics. 1965	6378137.0	0.0818202
NWL9D	Naval Weapons Lab., 1965	6378145.0	0.0818202
mod_airy	Modified Airy	6377340.189	0.0816734
andrae	Andrae 1876 (Den., Iclnd.)	6377104.43	0.0815816
aust_SA	Australian Natl & S. Amer. 1969	6378160.0	0.0818202
GRS67	GRS 67(IUGG 1967)	6378160.0	0.0818206

名前	説明	長半径	扁平率
bessel	Bessel 1841	6377397.155	0.0816968
bess_nam	Bessel 1841 (Namibia)	6377483.865	0.0816968
clrk66	Clarke 1866	6378206.4	0.0822719
clrk80	Clarke 1880 mod.	6378249.145	0.0824832
CPM	Commission des Poids et Mesures (1799)	6375738.7	0.0772909
delmbr	Delambre 1810 (Belgium)	6376428.0	0.080064
engelis	Engelis 1985	6378136.05	0.0818193
evrst30	Everest 1830	6377276.345	0.081473
evrst48	Everest 1948	6377304.063	0.081473
evrst56	Everest 1956	6377301.243	0.081473
evrst69	Everest 1969	6377295.664	0.081473
evrstSS	Everest (Sabah & Sarawak)	6377298.556	0.081473
fschr60	Fischer (Mercury Datum) 1960	6378166.0	0.0818133
fschr60m	Modified Fischer 1960	6378155.0	0.0818133
fschr68	Fischer 1968	6378150.0	0.0818133
helmert	Helmert 1906	6378200.0	0.0818133
hough	Hough	6378270.0	0.0819919
intl	International 1909 (Hayford)	6378388.0	0.0819919
krass	Krassovsky, 1942	6378245.0	0.0818133
kaula	Kaula 1961	6378163.0	0.0818215
lerch	Lerch 1979	6378139.0	0.0818192
mprts	Maupertius 1738	6397300.0	0.1021949
new_intl	New International 1967	6378157.5	0.0818202
plessis	Plessis 1817 (France)	6376523.0	0.0804333
SEasia	Southeast Asia	6378155.0	0.0818133
walbeck	Walbeck	6376896.0	0.0812068

名前	説明	長半径	扁平率
WGS60	WGS 60	6378165.0	0.0818133
WGS66	WGS 66	6378145.0	0.0818202
WGS72	WGS 72	6378135.0	0.0818188
WGS84	WGS 84	6378137.0	0.0818192

単位コンバータ

このセクションでは、投影図法で単位コンバータを使用する方法について説明します。以下のトピックから構成されています。

- *単位コンバータの直接使用*
- *単位コンバータの定義*
- *定義済み単位コンバータの使用*

単位コンバータの直接使用

メートルをフィートに変換する場合は、IlvUnitConverter クラスのスタティック・メンバが使用できます。

次のコードは、メートルをフィートに変換します。

```
IlvUnitConverter* converter = IlvUnitConverter::FT();
IlvDouble meters = 100;
IlvDouble feet= converter->fromMeters(meters);
IlvPrint("100 m = %f ft", feet);
feet = 100;
meters = converter->toMeters(feet);
IlvPrint("100 ft = %f m", meters);
```

IlvUnitConverter クラスの toMeters メソッドは、現在の測定単位をメートルに変換します。一方、fromMeters メソッドはメートルを、現在の測定単位に変換します。

通常使用するコンバータを定義する IlvUnitConverter クラスのスタティック・メンバを使用することも、142 ページの *単位コンバータの定義* で後述するように、独自のコンバータを作成することもできます。さらに、142 ページの *定義済み単位コンバータの使用* に示すリストから、定義済みコンバータの 1 つを選んで使用することもできます。

投影図法でコンバータを使用する

次の例で示すように、単位コンバータを投影図法に関連付けることができます。このコード・サンプルは、メートルの代わりにフィートを使用するように変更した useproj プログラムの修正版です。

```
#include <ilviews/maps/projection/mercator.h>
main()
{
    IlvUnitConverter* converter = IlvUnitConverter::FT();
    IlvMercatorProjection projection;
    projection.setUnitConverter(*converter);
    const double lambda  = IlvMaps::DegreeToRadian(-45.0);
    const double phi = IlvMaps::DegreeToRadian(30.0);
    IlvCoordinate ll(lambda, phi);
    IlvCoordinate xy;
    projection.forward(ll, xy);
    IlvPrint("The projection of 45W 30N is \n"
             " x = %f ft\n"
             " y = %f ft",
             xy.x(),
             xy.y());
    ll.moveTo(0, 0);
    projection.inverse(xy, ll);
    char buffer1[12];
    char buffer2[12];
    IlvPrint("The inverse projection is \n"
             " %s %s",
              IlvMaps::RadianToDMS(buffer1, ll.x(), IlFalseIlFalse),
              IlvMaps::RadianToDMS(buffer2, ll.y(), IlTrue));
}
```

setUnitConverter の呼び出しによって、測定単位としてフィートの使用を指定します。これによって、forward メソッドの出力がフィート単位になります。同様に、inverse メソッドの出力もフィート単位にする必要があります。

単位コンバータの定義

単位コンバータを定義するためには、測定単位名と単位当たりのメートル換算値を指定する必要があります。

次に示すのは、測定単位としてキロメートルを定義するコードです。

```
IlvUnitConverter converter(1000,"km");
```

定義済み単位コンバータの使用

このライブラリには、定義済み単位コンバータのリストが含まれます。定義済み単位コンバータはその名前で参照されます。定義済み単位コンバータへのアクセスには、IlvUnitConverter クラスの GetRegisteredConverter スタティック関数を使用します。

次に示すのは、メルカトル図法をインスタンス化して、その出力を国際海里に変換するコード・サンプルです。

```
IlvUnitConverter* converter =
                    IlvUnitConverter::GetRegisteredConverter("kmi");
IlvMercatorProjection projection;
projection.setUnitConverter(*converter);
```

定義済み単位コンバータのリスト

IBM ILOG Views Maps 投影図法ライブラリによって提供される、定義済み単位コンバータのリストを次に示します。

名前	説明	メートル換算値
km	キロメートル	1000.0
m	メートル	1.0
dm	デシメートル	0.1
cm	センチメートル	0.01
mm	ミリメートル	0.001
kmi	国際海里	1852.0
in	国際インチ	0.0254
ft	国際フィート	0.3048
yd	国際ヤード	0.9144
mi	国際法定マイル	1609.344
fath	国際尋	1.8288
ch	国際チェーン	20.1168
link	国際リンク	0.201168
us-in	米国測量局のインチ	0.025400050800101603
us-ft	米国測量局のフィート	0.304800609601219
us-yd	米国測量局のヤード	0.914401828803658
us-ch	米国測量局のチェーン	20.11684023368047
us-mi	米国測量局の法定マイル	1609.347218694437
ind-yd	インドのヤード	0.91439523

名前	説明	メートル換算値
ind-ft	インドのフィート	0.30479841
ind-ch	インドのチェーン	20.11669506

測地原点が異なる座標間の変換

異なる地図局によって作成された地図をマージする場合、同じポイントが数百メートル離れてしまう場合があります。この位置決めエラーの原因は、各地図局がそれぞれの座標系に異なる測地データを採用しているからです。

経緯度原点 (または水平方向の経緯度原点) は、ポイントの地理座標を緯度と経度で表す座標参照系です。永年にわたって、数百という異なる測地データが世界中の地図制作者によって使用されてきました。しかし、ジオイド表面が不規則なため、ほとんどの場合、違う 2 つの測地データを使って読み取った同じポイントの座標が異なってしまいます。ただし今日では、衛星測位システムや全地球測位システムの利用によって測地測量の精度はセンチメートル単位にまで向上しています。

測地系の不一致のため、異なるソースによる地図の統合ではデータの変換が必要です。

このセクションでは、IBM® ILOG® Views Maps における測地系変換の実行方法を説明するとともに、使用する変換メソッドを示します。

以下のトピックを取り上げます。

- 測地系の水平移動
- 測地系と投影図法

測地系の水平移動

IBM ILOG Views Maps がデフォルトで実装しているこの測地系変換メソッドは、使用測地系が楕円表面から読まれた緯度と経度によるもので、測地系を使用するエリアのジオイド表面に接するように、その中心が地心からわずかにずれていることを前提にしています。

この測地系は、`IlvHorizontalDatum` のサブクラスである、`IlvHorizontalShiftDatum` クラスによって実装します。移動パラメータを計算するための参照測地系は、WGS84 測地系です。米国 National Imagery and Mapping Agency (NIMA) から、大量データ処理用に、この参照測地系関連の移動パラメータが公開されています。

ここでは測地系 D1 から測地系 D2 への座標変換に、WGS84 測地系を使用します。D1 楕円表面の緯度と経度が、デカルト座標の X、Y、Z に変換されます。これらの

座標が、測地系 WGS84 の楕円表面に再投影されます。同じ操作を繰り返して、D2 楕円表面のポイントの緯度と経度を取得します。Molodensky 式のベースはこの変換原理です。`IlvMolodenskyConverter` クラスによって実装されます。この変換技術によって、データの精度は 10m 前後にまで向上します。

測地系と投影図法

投影図法のこうした測地系は、`IlvProjection::setDatum()` メソッドによって指定でき、`IlvProjection::getDatum()` によって取得できます。

インポートした地図のソースの投影図法とターゲットの投影図法が異なる測地データを持つ場合は、`IlvMapInfo::toViews()` メソッドによって測地系変換を実行します。

インポートした地図の上にグラフィック・オブジェクトを追加する

このセクションでは、投影図法が IBM® ILOG® Views マネージャにある `.ilv` 地図ファイルをインポートする方法と、その地図の上にグラフィック・オブジェクトを重ねる方法について説明します。ランベルト正積方位図法で投影した米国の地図をマネージャにローディングし、地図の上部に都市を追加する例に基づいて説明します。マウス・ポインタで示した地理座標と都市の名前は、ウィンドウ下部のテキスト・フィールドに表示されます。

サンプル・コード一式

この例のソース・コード一式は、以下のファイルにあります。

`<installdir>/samples/maps/userman/src/useviews.cpp`

次に示す各セクションで、このサンプル・コードについて説明します。

- サンプル・アプリケーションの実行
- Sample クラス、Main 関数、コンストラクタの定義
- 地図情報の取得
- 都市の追加
- マウス位置の表示

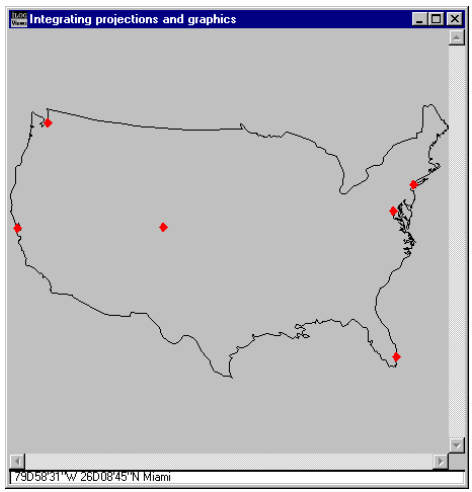
サンプル・アプリケーションの実行

サンプル・アプリケーションのコンパイルと実行は、次の手順に従います。

- ディレクトリ `<installdir>/samples/maps/userman/<platform>` まで移動します。

- ILVHOME 変数を IBM® ILOG® Views インストール・ディレクトリに設定します。
- make (UNIX® プラットフォームの場合) または nmake (Microsoft® Windows® プラットフォームの場合) を使ってコンパイルします。このユーザ・マニュアル用のサンプルすべてが、これによってコンパイルされます。
- useviews アプリケーションを起動します。

アプリケーションによって、一部の都市を伴う米国合衆国の地図が表示されます。



Sample クラス、Main 関数、コンストラクタの定義

クラス

このサンプルはクラス SimpleMapViewer として実装され、地図をロードし都市を作成します。

次のようなフィールドが含まれます。

```
private:
  IlvGadgetContainer* _container;
  IlvSCManagerRectangle* _managerRectangle;
  IlvMapInfo* _mapInfo;
  IlvTextField* _statusBar;
```

ここで、_container はこのアプリケーションのトップ・ウィンドウ、_managerRectangle は IlvManager と IlvView を組み合わせ、_mapinfo フィールドは IlvMapInfo インスタンスで、地図座標系からマネージャ座標系に座標を変換するための情報を格納します。地図情報は .ilv ファイルとともに保存されるため、地図ファイルをロードする際にこのフィールドが初期化されます。そのため、

地図上に都市を位置付けたり、地図上にマウスのポインタで示された位置に対応する地理座標を表示するために、このフィールドが使用できます。これらの地理座標は、ステータス・バーとして機能する IlvTextField に格納されます。

Main 関数

Main 関数は、IlvDisplay を初期化し、SimpleMapViewer クラスのインスタンスを作成してから、IBM ILOG Views のメイン・ループに入ります。

```
int main(int, char**)
{
    IlvDisplay* display = new IlvDisplay("Map Viewer");
    if (display->isBad()) {
        IlvPrint("Cannot create the display");
        return 1;
    }
    SimpleMapViewer*  viewer = new SimpleMapViewer(display,
                                                   "../data/usa.ilv");
    IlvMainLoop();
    return 0;
}
```

コンストラクタ

SimpleMapViewer のコンストラクタは、次の 2 つのアクションを実行します。クラスのインターフェース・コンポーネントを作成し、次に地図データをロードします。

```
SimpleMapViewer::SimpleMapViewer(IlvDisplay* display,
                                 const char* fileName)
    :_managerRectangle(0),
     _statusBar(0),
     _container(0),
     _mapInfo(0)

{
    createGUI(display);
    loadMap(fileName);
}
```

createGUI メソッドは、アプリケーションのトップ・ビューである IlvGadgetContainer のインスタンスを作成し、次に、地図を含むコンテナの ManagerRectangle を作成するメソッドおよび、ツールバーを作成するメソッドを呼び出します。

```
void
SimpleMapViewer::createGUI(IlvDisplay* display)
{
    _container = new IlvGadgetContainer(display,
                                        "SimpleMapViewer",
                                        "Integrating projections and graphics",
                                        IlvRect(50, 50, 450, 450),
                                        IlFalse);
    _container->setDestroyCallback(_exit, this);
    createManagerRectangle(_container);
    createStatusBar(_container);
```

```
}
```

createManagerRectangle メソッドは、地図を格納するマネージャを作成し、ビューを初期化します。

```
void
SimpleMapViewer::createManagerRectangle(IlvGadgetContainer* container)
{
    _managerRectangle = new IlvSCManagerRectangle(container->getDisplay(),
                                                  IlvRect(0, 0, 450, 435));
    container->addObject(_managerRectangle);

    // Attachments.
    container->getHolder()->attach(_managerRectangle, IlvHorizontal);
    container->getHolder()->attach(_managerRectangle, IlvVertical);

    IlvManager* manager = _managerRectangle->getManager();
    IlvView* view = _managerRectangle->getView();

    manager->setKeepingAspectRatio(view, IlTrue);
    manager->setDoubleBuffering(view, IlTrue);
}
```

次に、createStatusBar メソッドが、ステータス・バーとして使用される IlvTextField を作成し、付加します。

```
void
SimpleMapViewer::createStatusBar(IlvGadgetContainer* container)
{
    _statusBar = new IlvTextField(container->getDisplay(),
                                  "",
                                  IlvRect(0, 435, 450, 15));
    container->addObject(_statusBar, IlTrue);

    _statusBar->setEditable(IlFalse);

    // Attachments.
    container->getHolder()->attach(_statusBar, IlvHorizontal);
    container->getHolder()->attach(_statusBar, IlvVertical, 1, 0, 0);
}
```

グラフィック・インターフェースの準備ができたら、地図をロードできます。

```
void
SimpleMapViewer::loadMap(const char* fileName)
{
    IlvManager* manager = _managerRectangle->getManager();
    IlvView* view = _managerRectangle->getView();
    manager->read(fileName);
    _mapInfo = IlvMapInfo::Get(manager);
    if (_mapInfo) {
        view->setInputCallback(_showMousePosition, this);
        addCities();
    }
    manager->fitTransformerToContents(view, IlTrue);
}
```

地図がロードされたら、.ilv ファイルに保存されていた地図情報を _mapinfo フィールドに格納し、入力コールバックのインストールによってマウスの位置を表示して都市を追加します。

地図情報の取得

IBM® ILOG® Views Maps では、IlvMapInfo クラスを使って地図情報をマネージャに付加できます。このクラスは、地図座標とマネージャ座標間の座標変換のために、IlvProjection と IlvMapAdapter をカプセル化します。

IlvManager に付加された IlvMapInfo を取得するには、サンプル・コードにあるように IlvMapInfo::Get スタティック関数を使用します。

```
    _mapInfo = IlvMapInfo::Get(manager);
```

都市の追加

addCities メソッドは、インポートした米国の地図上にいくつもの都市を追加します。

```
void
SimpleMapViewer::addCities()
{
    addCity("Washington", "39D11'N", "76D51W");
    addCity("New York", "40D59'N", "73D39'W");
    addCity("Miami", "25D58'N", "80D02'W");
    addCity("San Francisco", "37D44'N", "122D20'W");
    addCity("Seattle", "47D51'N", "122D01'W");
    addCity("Denvers", "39D50'N", "104D53'W");
}
```

addCity メソッドは最初に、表示する都市の緯度と経度を計算します。

```
void
SimpleMapViewer::addCity(const char* cityName,
                         const char* latString,
                         const char* longString)
  {
      double latitude;
      IlvMaps::DMSToRadian(latString, latitude);

      double longitude;
      IlvMaps::DMSToRadian(longString, longitude);

      IlvCoordinate c(longitude, latitude);
```

都市の地理座標が計算されると、次にこのメソッドは地図の _mapInfo を使ってそれらの座標をマネージャ座標に変換します。この変換は先に、選択した投影図法のデカルト座標に地理座標を効果的に変換してから、それらの座標をマネージャ単位に再変換します。投影図法座標系での座標変換にはエラーが生じやすいため、変換ステータスがテストされる点に留意してください。

```
     IlvMapsError status = IlvMaps::NoError();
```

```
    IlvPoint p;
    status = _mapInfo->forward(c, p);
```

座標が変換され、変換プロセスでのエラーがなければ、赤いマーカーとして都市を追加します。

```
  if (status == IlvMaps::NoError()) {
      IlvMarker* marker = new IlvMarker(_container->getDisplay(),
                                        p,
                                        IlvMarkerFilledDiamond);
      marker->setSize(4);
      marker->setForeground(_container->getDisplay()->getColor("red"));
      IlvManager* manager = _managerRectangle->getManager();
      manager->addObject(marker, 1, IlFalse);
      marker->setName(cityName);
  }
```

マウス位置の表示

データが地図にロードされたら、マウスの位置を表示するために入力コールバックを設定します。

```
if (_mapInfo) {
       view->setInputCallback(_showMousePosition, this);
    }
```

このコールバックは、ビューで入力イベントが発生するたびに showMousePosition メソッドを呼び出します。

```
void _showMousePosition(IlvView* view, IlvEvent& event, IlvAny arg)
{
    SimpleMapViewer* mapViewer = (SimpleMapViewer*) arg;
    mapViewer->showMousePosition(view, event);
}
```

このメソッドは最初に、情報を表示するため、いくつかのバッファを初期化します。

```
void
SimpleMapViewer::showMousePosition(IlvView* view, IlvEvent& event)
{
    char buf1[12];
    char buf2[12];
    char label[50];
}
```

次に、マウスの位置の下に最後にあったオブジェクトを取得します、さらに、都市名として表示するそのオブジェクトの名前があれば、それも取得します。

```
    IlvManager* manager = _mapInfo->getManager();

    const char* name = "";
    IlvPoint p(event.x(), event.y());
    IlvGraphic* g = manager->lastContains(p, view);
    if (g && g->getName())
```

```
        name = g->getName();
```

ここで再び、IlvMapInfo インスタンスによって、マウス位置のマネージャ座標を地理座標に変換します。

```
IlvCoordinate ll;
if (_mapInfo->inverse(event, view, ll) == IlvMaps::NoError())
    sprintf(label, "%s %s %s",
                   IlvMaps::RadianToDMS(buf1, ll.x(), IlFalse),
                   IlvMaps::RadianToDMS(buf2, ll.y(), IlTrue),
                   name);
else
    sprintf(label, "Unable to invert mouse position");
```

最後に、ステータス・バーにラベルを設定します。

```
 _statusBar->setLabel(label);
 _statusBar->reDraw();
```

新しい投影図法の作成

このセクションでは、IBM® ILOG® Views Maps の投影図法ライブラリを独自の投影図法で拡張する方法について説明します。このセクションで使用する例は、メルカトル図法の簡易版です。

サンプル・コードは 3 つのステップに分かれ、各ステップがそれぞれ本ライブラリの違う側面を示します。

- ステップ 1 では、投影図法をサブタイプするために必要な最低要件について説明します。具体的には、投影図法の順関数と逆関数の実装方法を紹介します。
- ステップ 2 では、投影図法へのパラメータの追加方法と、それら追加パラメータに対応する入出力関数の書き方について説明します。また、特定エラー・コードの作成方法にも触れます。
- ステップ 3 では、投影図法の特定パラメータに対応するアクセサの追加方法を説明します。

サンプルのソース・ファイルは、次のディレクトリにあります。

`<installdir>/samples/maps/userman/src`

(ソース・ファイルのディレクトリ)、ファイル名は `proj_step1.cpp`、`proj_step2.cpp`、`proj_step3.cpp`、また

`<installdir>/samples/maps/userman/include`

が対応するインクルード・ファイルのディレクトリ。

ステップ 1 新しい投影図法の定義

最初のステップでは、投影図法をサブタイプ化するために必要な最低要件について説明します。具体的には、投影図法の forward 関数と inverse 関数の実装方法を紹介します。

この投影図法のコード一式は、proj_step1.h ファイルと proj_step1.cpp ファイルに入っています。

クラス宣言

メルカトル図法は、proj_step1.h ファイルで宣言します。

この投影図法の基本クラス IlvProjection を宣言する <ilviews/maps/projection/project.h> ファイルを含める必要があります。

次に、実装する投影図法機能の宣言が必要です。

- sForward はもっとも単純なケース、すなわち地球を球体としてモデリングする場合の順投影を実装します。IlvProjection::sForward 関数は抽象関数なので、この機能の実装は必須です。

次に示す関数の宣言と実装は、必須ではありません。この新しい投影図法は単純に、実装されていない機能には対応せず、アプリケーションによってそれらが要求されるとエラー・コードを返します。

- sInverse は、地球が球体としてモデリングされる場合に逆投影を実装します。
- eForward は、地球が非球形の楕円としてモデリングされる場合に順投影を実装します。
- eInverse は、地球が非球形の楕円としてモデリングされる場合に逆投影を実装します。

投影図法宣言では、IlvMapsDeclareProjectionIO マクロを使用する必要があります。このマクロでは、入出力操作をサポートするために、クラスの IlvProjectionClassInfo といくつかの必須メンバを宣言します。

投影図法を宣言したら、ファイルに IlvMapsInitProjectionIO を追加する必要があります。このマクロによって、静的初期化中に IlvProjectionClassInfo の初期化が保証されます。

```
#include <ilviews/maps/projection/project.h>
class Mercator : public IlvProjection
{
public:
    Mercator();

protected:
    virtual IlvMapsError sForward(IlvCoordinate &) const;
    virtual IlvMapsError sInverse(IlvCoordinate &) const;
```

```
    virtual IlvMapsError eForward(IlvCoordinate &) const;
    virtual IlvMapsError eInverse(IlvCoordinate &) const;
    IlvMapsDeclareProjectionIO(Mercator);
};
// Enable IO initialization.
IlvMapsInitProjectionIO(Mercator);
```

投影図法の定義

投影図法は、proj_step1.cpp ファイルで定義されます。

クラスを定義するためには、IlvMapsDefineBasicProjectionIO マクロを使用します。このマクロは、入出力操作をサポートするのに必要な関数とスタティック・メンバを定義します。また、IlvProjectionClassInfo を初期化するコードも生成します。

```
#include "proj_step1.h"
IlvMapsDefineBasicProjectionIO(Mercator,
                               IlvProjection,
                               "My Mercator Implementation",
                               new Mercator(),
                               IlvMapsEmptyStatement());
```

- このマクロの最初の引数は、投影図法クラスの名前です。
- 2 番目の引数は、投影図法のスーパークラスの名前です。
- 3 番目の引数は投影図法の名前です。この名前は、IlvProjectionDictionary クラスなどで使用されます。
- 4 番目の引数は、このクラスの新しいインスタンスを作成するために使用されるステートメントです。
- 最後の引数は、投影図法クラスの初期化で呼び出されるステートメントです。このサンプル・コードでは指定がないので、IlvMapsEmptyStatement マクロが使用されます。最終パラメータは、ステップ 2 および 3 で使用されます。

ここで、投影図法コンストラクタの定義が必要になります。このコンストラクタは、その IlvProjection スーパークラスのコンストラクタを呼び出します。3 つの引数を取ります。

- 最初の引数は IlBoolean 値で、投影図法で非球形の楕円をサポートするかどうかを指定します。このサンプルでは、投影図法で非球形の楕円の方程式をサポートしているため、この引数は IlTrue に設定されています。
- 2 番目の引数は IlvBoolean 値で、逆関数をサポートするかどうかを示します。この例では、投影図法で逆関数をサポートしているため、この引数は IlTrue に設定されています。
- 3 番目の引数は enum 値で、投影図法のジオメトリ・プロパティを示します。この例では、メルカトル図法が正角のため、この引数は IlvConformalProjectionGeometricProperty です。

IlvMapsDeclareProjectionIO マクロで宣言されるコピー・コンストラクタを定義する必要もあります。

```
Mercator::Mercator()
:IlvProjection(IlTrue,
               IlTrue,
               IlvConformalProjectionGeometricProperty)
{
}
Mercator::Mercator(const Mercator& source)
:IlvProjection(source)
{
}
```

順投影を書く

メルカトル図法の順関数を書く前に、IlvProjection::forward 関数について理解しておく必要があります。

IlvProjection::forward 関数

IlvProjection::forward パブリック関数は、データの投影を行うユーザが呼び出します。この関数は投影計算のためにデータを用意し、適切に調整します。その後、呼び出しを投影サブクラス (この例ではメルカトル図法クラス) に定義された、eForward または sForward のいずれかの保護された関数に振り替えます。

IlvProjection::forward 関数 :

- 座標が地心の場合、緯度を調整します。
- 経度を投影図法の中央経線に合わせます。
- 経度換算を使っている場合に (デフォルト値)、経度を範囲 [-PI;PI] に合わせます。
- 地球を球体で表すか、楕円で表すかによって、sForward 関数か、または eForward 関数を呼び出します。次に、
- 投影済みデータを楕円の寸法とデカルト座標オフセットに調整して、指定した測定単位に変換します。

球形からデータを投影する

sForward 保護関数は、球形の投影図法を実装します。

IlvProjection::forward 関数によって適切なスケーリングが行われるため、sForward 関数では、常に球体の半径を 1 とします。

この例の場合、メルカトル図法は赤道に接する円筒上の球体の投影図法です。球体の半径を 1 と仮定するため、x 座標は経度 (ラジアンで表す) に等しくなります。この場合、x 値の ll は変更不要です。

メルカトル図法では、極地付近の領域を表示できないため、緯度が PI/2 に近すぎる場合は、エラー・コードを返します。

次の方程式を使って、投影済みデータの y 座標を計算します。

```
IlvMapsError
Mercator::sForward(IlvCoordinate& ll) const
{
    // Return an error if the point is close to a pole.
    if (fabs(fabs(ll.y()) - IlvMaps::Pi() / 2.) <= 1e-10)
     return ToleranceConditionError();

    ll.setY(log(tan(IlvMaps::Pi() / 4. + 0.5 * ll.y())));
    return IlvMaps::NoError();
```

楕円からデータを投影する

eForward 保護関数は、データが非球形の楕円から投影されている場合に、IlvProjection::forward 関数で呼び出します。

投影図法に eForward 関数を実装する必要はありません。非球形の楕円からデータを投影する場合で、使用する投影図法がこの種の楕円をサポートしない場合は、forward 関数が、IlvProjection::UnsupportedFeatureError() で与えられるエラー・コードを返します。この場合、球体の楕円を使うか、または IlvEllipsoid クラスの、適切な変換関数を使って相当する球形を作成することができます。

getEllipsoid()->getE() が 0 を返す場合、式の結果は同じですが、eForward 関数は sForward 関数よりも若干複雑になります。

```
IlvMapsError
Mercator::eForward(IlvCoordinate& ll) const
{
    // Return an error if the point is close to a pole.
    if (fabs(fabs(ll.y()) - IlvMaps::Pi() / 2.) <= 1e-10)
     return ToleranceConditionError();

    IlvDoublee = sqrt(getEllipsoid()->getES());
    IlvDouble   sinphi = e * sin(ll.y());

    ll.setY(tan(.5 * (IlvMaps::Pi() / 2. -
                ll.y())) /
            pow((1. - sinphi) / (1. +  sinphi),
                0.5 * e));
    ll.setY(-log(ll.y()));

    return IlvMaps::NoError();
}
```

逆投影を書く

メルカトル図法の inverse 関数を書くためには、IlvProjection::inverse 関数に精通している必要があります。

IlvProjection::inverse 関数

inverse 関数は、データを逆投影して適切なオフセットになるように処理します。

この関数は、以下の項目を実施します。

- デカルト座標で作成したオフセットを削除し、それらの座標をメートルに変換します。
- 座標を地理値に戻し、値 1 の長半径で標準の楕円に適用します。
- 楕円が球形かどうかによって、関数 sInverse または eInverse を呼び出します。
- 中央経線の値を経度に追加し、経度換算を使っている場合に (デフォルト値)、その経度を範囲 [-PI;PI] に合わせます。
- 座標が地心の場合、緯度を変換します。

球形への逆投影

球形への逆投影は、sInverse 関数を介して実行されます。

sInverse 関数を実装する必要はありません。逆関数をサポートしない投影図法で IlvProjection::inverse 関数を呼び出すと、エラー・コード IlvProjection::UnsupportedFeatureError() が返されます。

先に sForward 関数で見たとおり、この投影図法では x の値を変更しません。したがって、逆方程式は y 値にだけ適用されます。

```
IlvMapsError
MercatorProjection::sInverse(IlvCoordinate& xy) const
{
    xy.setY(IlvMaps::Pi()/2. - 2.*atan(exp(-xy.y())));
    return IlvMaps::NoError();
}
```

楕円への逆投影

楕円への逆投影は、eInverse 関数を介して実行されます。この関数では、楕円の長半径の値を 1 と仮定します。

メルカトル図法の特定のケースでは、この関数の楕円への実装は球形の場合よりも複雑になります。非球形の楕円からのメルカトル図法の単純で分析的な逆方程式がないため、繰り返し処理が必要となり、失敗する可能性があります。

```
IlvMapsError
Mercator::eInverse(IlvCoordinate& xy) const
{
    IlvDouble ts = exp(-xy.y());
    IlvDouble e = sqrt(getEllipsoid()->getES());
    IlvDouble eccnth = 0.5 * e;
    IlvDouble Phi = IlvMaps::Pi() / 2. -2. * atan(ts);

    int i = 15;
    IlvDoubledphi;
```

```
    do {
      IlvDouble con = e * sin(Phi);
      dphi = IlvMaps::Pi()/2. -
             2. * atan(ts * pow((1 - con)/(1 + con), eccnth)) - Phi;
      Phi += dphi;
    } while(fabs(dphi) > 1.e-10 && --i != 0);
    if(i <= 0)
      return ToleranceConditionError();

    xy.setY(Phi);

    return IlvMaps::NoError();
}
```

ステップ 2 新しい投影図法の定義

このステップでは、投影図法へのパラメータの追加方法と、それら追加パラメータに対応する入出力関数の書き方について説明します。また、特定エラー・コードの作成方法にも触れます。

この投影図法のコード一式は、proj_step2.h ファイルと proj_step2.cpp ファイルに入っています。

新規パラメータの定義

メルカトル図法では、距離は保持されません。メルカトル図法の場合、倍率が緯度によって変化します。さらに、ポイントが赤道から離れれば離れるほど倍率は大きくなります。ただし、実縮尺の緯度を指定できます。すなわち、その緯度を基準として投影したポイント間の距離が維持されます。

以下の例によって、この新規パラメータを紹介するとともに、その永続性の管理方法を示します。

新規エラー・コードの定義

メルカトル図法の前記ステップで見たとおり、ポイントの順投影では、ポイントが極に近付きすぎるとエラーが発生する場合があります。汎用エラー・コードの IlvProjection::ToleranceConditionError() を返す代わりに、この場合に返される特定のエラー・コードを作成します。

新規クラスの宣言

メルカトル・クラスの第 2 版を、ファイル proj_step2.h で宣言します。

サンプル・コードで追加したのは、次のものです。

- 実縮尺の緯度を設定するメソッド
- 実縮尺の緯度を取得するメソッド

- メルカトル図法の特定エラー・コードを取得するための、PolarZoneError スタティック・メソッド
- デフォルトの IlvProjection::write メソッドをオーバーライドする仮想 write メソッド
- 新規エラー・コードの割り当てに使用する、InitClass プライベート・スタティック・メソッド
- 実縮尺の緯度を格納するプライベート・フィールド
- エラー・コードを格納するプライベート・スタティック・フィールド

```
class Mercator : public IlvProjection
{
public:
    Mercator();

    void setLatitudeOfTrueScale(IlvDouble latitudeOfTrueScale)
    {_latitudeOfTrueScale = latitudeOfTrueScale;}

    IlvDouble getLatitudeOfTrueScale() const
    {return _latitudeOfTrueScale;}

    static IlvMapsError PolarZoneError() {return _polarZoneError;}

    virtual void write(IlvOutputFile&) const;

protected:
    virtual IlvMapsError sForward(IlvCoordinate &) const;
    virtual IlvMapsError sInverse(IlvCoordinate &) const;
    virtual IlvMapsError eForward(IlvCoordinate &) const;
    virtual IlvMapsError eInverse(IlvCoordinate &) const;

private:
    static void InitClass();

private:
    IlvDouble _latitudeOfTrueScale;
    static IlvMapsError _polarZoneError;

    IlvMapsDeclareProjectionIO(Mercator);
};
```

投影図法の定義

投影図法は、proj_step2.cpp ファイルで定義されます。

クラスを定義するためには、IlvMapsDefineProjectionIO マクロを使用します。このマクロは、追加パラメータの保存が必要な投影図法用の IlvMapsDefineBasicProjectionIO マクロの代わりに使用する必要があります。

このステップでは、エラー・コードを初期化する Mercator::InitClass() スタティック・プライベート関数に、マクロの初期化ステートメントを設定します。この場所でメルカトル・クラスのスタティック・プライベート関数を呼び出すこと

ができます。Mercator クラスの friend として、IlvMapsDeclareProjectionIO マクロによって宣言された用途で、IlvMapsDefineProjectionIO マクロが初期化ステートメントを生成するからです。

```
IlvMapsDefineProjectionIO(Mercator,
                          IlvProjection,
                          "My Mercator Implementation",
                          new Mercator(),
                          Mercator::InitClass());
```

エラー・コードの初期化

新規エラー・コードは InitClass メソッドによって呼び出されます。このメソッドは静的初期化の段階で、自動的に呼び出されます。

```
void
Mercator::InitClass()
{
    _polarZoneError =
         IlvMaps::CreateError("&MercatorPolarZoneError");
}
```

新規パラメータと新規エラー・コードの使用

実縮尺の緯度と新規エラー・コードは、順関数および逆関数で使用されます。次の例では、投影図法関数におけるそれらの使用法を示します。

```
IlvMapsError
Mercator::sForward(IlvCoordinate& ll) const
{
    if (fabs(fabs(ll.y()) - IlvMaps::Pi() / 2.) <= 1e-10)
       // Returning the specific error code.
       return PolarZoneError();

    IlvDouble k = cos(_latitudeOfTrueScale);
    ll.setY(k * log(tan(IlvMaps::Pi() / 4. + 0.5 * ll.y())));
    ll.setX(k * ll.x());
    return IlvMaps::NoError();
}
```

新規パラメータの入出力関数を書く

新規パラメータの完全な IO サポートを提供するためには、write メソッドを実装して、この追加パラメータを保存する必要があります。この write メソッドでは、何らかのデータを書き込む前に、スーパークラスの write メソッドを呼び出す必要があります。

```
void Mercator::write(IlvOutputFile &file) const
{
    IlvProjection::write(file);
    file.getStream() << _latitudeOfTrueScale << IlvSpc();
}
```

読み込みコンストラクタの実装も必要です。このコンストラクタは、そのスーパークラスの読み込みコンストラクタを呼び出してから、実縮尺の緯度を読み込みます。

```
Mercator::Mercator(IlvInputFile& file)
:IlvProjection(file)
{
    file.getStream() >> _latitudeOfTrueScale;
}
```

最後に、この新規パラメータのコピーを実装するために、コピー・コンストラクタを更新する必要があります。

```
Mercator::Mercator(const Mercator& source)
:IlvProjection(source),
 _latitudeOfTrueScale(source._latitudeOfTrueScale)
{
}
```

ステップ 3 新しい投影図法の定義

このステップでは、投影図法の特定パラメータに対応するアクセサの追加方法を説明します。

この投影図法のコード一式は、proj_step3.h ファイルと proj_step3.cpp ファイルに入っています。

アクセサ・サポートの追加

アクセサは、投影図法のパラメータに関するランタイム情報を提供します。それらは通常、地図アプリケーションの汎用投影図法エディタの構築に使用されます。

アクセサを投影図法に追加するためには、このパラメータのゲッタ関数とセッタ関数を書き、静的初期化の段階で、アクセサを IlvProjectionClassInfo に追加する必要があります。

```
// The getter for the latitudeOfTrueScale accessor.
static void _getter(const IlvProjection* p, IlvValue& v)
{
    Mercator* mercator = (Mercator*) p;
    char buffer[12];
    v = IlvMaps::RadianToDMS(buffer,
            mercator->getLatitudeOfTrueScale(),
            IlTrue);
}

// The setter for the latitudeOfTrueScale accessor.
static IlvBoolean _setter(IlvProjection* p, const IlvValue& v)
{
    Mercator* mercator = (Mercator*) p;
    IlvDouble value;
    IlvMapsError error = IlvMaps::DMSToRadian(v, value);
    if (error != IlvMaps::NoError())
```

```
      return IlFalse;
     mercator->setLatitudeOfTrueScale(value);
    return IlTrue;
}

void
Mercator::InitClass()
{
    _polarZoneError =
        IlvMaps::CreateError("&MercatorPolarZoneError");

    ClassInfo()->addAccessor(IlvGetSymbol("latitudeOfTrueScale"),
                             IlvValueStringType,
                             _getter,
                             _setter);
}
```

章

7

地図データ

推奨無料ソース

このセクションでは、ダウンロードできる地図データの推奨無料ソースの一覧を提供します。

DTED0

***表 7.1** DTED0*

範囲	Web リンク
世界標高	http://geoengine.nima.mil/muse-cgi-bin/rast_roam.cgi

ESRI shape

表 7.2 ESRI Shape

範囲	Web リンク	代替 Web リンク
世界地図	http://en.wikipedia.org/wiki/Global_Administrative_Unit_Layers_(GAUL)	http://www.bluemarblegeo.com/products/worldmapdata.php?op=download (低解像度地図)
米国タイム・ゾーン	http://www.nationalatlas.gov/mld/timeznp.html	
米国の州	http://www.nationalatlas.gov/mld/statesp.html	
米国地図	http://www.nationalatlas.gov/mld/countyp.html	http://www.census.gov/geo/www/cob/st2000.html#shp
米国の ZIP コード (郵便番号)	http://www.census.gov/geo/www/cob/z32000.html (3 桁)	http://www.census.gov/geo/www/cob/z52000.html (5 桁)

GeoTIFF、JPG、または PNG

表 7.3 GeoTIFF、JPG、または PNG

範囲	Web リンク	
世界	http://www.unearthedoutdoors.net/global_data/true_marble/download	
NASA からの世界衛星地図	http://earthobservatory.nasa.gov/Features/BlueMarble/ 次のミラー・サイトからダウンロード可能 http://mirrors.arsc.edu/nasa/world_500m/	http://neo.sci.gsfc.nasa.gov/Search.html (低解像度地図)

S57

表 7.4 S57

範囲	Web リンク
米国	http://www.nauticalcharts.noaa.gov/mcd/enc/download_agreement.htm

VMAP0

表 7.5 VMAP

範囲	Web リンク
世界	http://geoengine.nga.mil/geospatial/SW_TOOLS/NIMAMUSE/webinter/vmap0_legend.html

その他のデータ・ソース

`http://www.nationalatlas.gov/atlasftp.html`

`http://www.census.gov/geo/www/cob/bdy_files.html`

`http://data.geocomm.com/catalog/US/group21.html`

索引

C

C++
　前提条件 **10**
CADRG ファイル・リーダ **105**

D

DTED 形式
　概要 **104**
　ロード・オン・デマンド **104**
DTED ファイル・リーダ **103**

E

eInverse 関数 **156**
ESFromFlattening メソッド
　IlvEllipsoid クラス **139**
ESFromPolarRadius メソッド
　IlvEllipsoid クラス **139**

F

forward 関数 **135**, **154**

G

GeoTIFF ファイル **109**
getDefaultFeatureRenderer メソッド
　IlvMapFeatureIterator インターフェース **56**
getLowerRightCorner メソッド
　IlvMapFeatureIterator インターフェース **60**
getNextFeature メソッド
　IlvMapFeatureIterator インターフェース **56**, **61**
　IlvShapeReader クラス **96**
getProjection メソッド
　IlvMapFeatureIterator インターフェース **60**
GetRegisteredEllipsoid メソッド
　IlvEllipsoid クラス **139**
getTile メソッド
　IlvTileController クラス **80**
getUpperLeftCorner メソッド
　IlvMapFeatureIterator インターフェース **60**

I

IlvAttributeArray クラス **97**
IlvAttributeInfoProperty クラス **47**
IlvCADRGCoverage クラス **106**
IlvCADRGFrame クラス **106**
IlvCADRGFrameReader クラス **105**, **106**
IlvCADRGLayer クラス **106**
IlvCADRGTiledLayer クラス **105**, **106**
IlvCADRGTocReader クラス **105**
IlvCADRTocReader クラス **105**
IlvDefaultFeatureRenderer クラス **50**
IlvDefaultRasterRenderer クラス **104**
IlvDefaultTileCache クラス **88**
IlvDTEDLayer クラス **104**

IlvDTEDReader クラス **104**
IlvEllipsoid クラス
  ESFromFlattening メソッド **139**
  ESFromPolarRadius メソッド **139**
  GetRegisteredEllipsoid メソッド **139**
  SPHERE **138**
IlvFeatureAttribute クラス **47**
IlvFeatureAttributeProperty クラス **47**, **48**
IlvFeatureRenderer インターフェース **46**
  makeGraphic メソッド **49**
IlvGeoTIFFLayer クラス **111**
IlvGeoTIFFReader クラス **110**
IlvGeoTIFFTileLoader クラス **110**
IlvGeoTIFFTiler クラス **111**
IlvGraphic クラス **73**
IlvImageLayer クラス **109**
IlvImageReader クラス **107**
IlvImageTileLoader クラス **107**
IlvLayerVisibilityFilter クラス **70**
IlvManagerLayer クラス **73**
IlvManagerMagViewInteractor クラス **82**
IlvMapFeature オブジェクト **123**
IlvMapFeature クラス **45**, **46**, **47**
IlvMapFeatureIterator インターフェース **96**
  getDefaultFeatureRenderer メソッド **56**
  getLowerRightCorner メソッド **60**
  getNextFeature メソッド **56**, **61**
  getProjection メソッド **60**
  getUpperLeftCorner メソッド **60**
  isGeoreferenced メソッド **56**, **67**
IlvMapFeatureIterator オブジェクト **123**
IlvMapGeometry クラス **47**
IlvMapLineString クラス **50**
IlvMapLoader クラス **46**, **66**, **68**, **106**
  load メソッド **66**
  makeFeatureIterator メソッド **68**
  setDefaultSourceProjection メソッド **67**
IlvMapRaster クラス **104**, **107**
IlvMapTileLoader クラス
  load() : **74**
IlvNamedProperty クラス **48**
IlvObjectSDOFeatureIterator クラス **115**
IlvObjectSDOLayer クラス **115**
IlvProjection クラス **46**
  setEllipsoid メソッド **138**
IlvS57Loader オブジェクト **123**
IlvS57Renderer オブジェクト **123**, **125**
IlvScaleVisibilityFilter クラス **70**
IlvSDOFeatureIterator クラス **113**
IlvSDOLayer クラス **114**
IlvSDOTileLoader クラス **115**
IlvSDOWriter クラス **114**
IlvShapeDBFReader クラス **96**, **98**
  readRecord メソッド **98**
IlvShapeFileIndex クラス **100**
IlvShapeFileReader クラス **96**
IlvShapeFileTileLoader クラス **103**
IlvShapeFileTiler クラス **102**
IlvShapeLayer クラス **103**
IlvShapeReader クラス
  getNextFeature メソッド **96**
IlvShapeSHPReader クラス **97**
IlvShapeSpatialIndex クラス **101**
IlvSHPReader クラス **96**, **97**
IlvTileCache クラス **73**
IlvTileController クラス **73**
  getTile メソッド **80**
  lockTile メソッド **85**
  setSize メソッド **81**
IlvTiledLayer クラス **73**
  setDebugView メソッド **81**
IlvView クラス **81**
inverse 関数 **135**, **156**
isGeoreferenced メソッド
  IlvMapFeatureIterator インターフェース **67**

L

load メソッド
  IlvMapLoader クラス **66**
lockTile メソッド
  IlvTileController クラス **85**

M

makeFeatureIterator メソッド
  IlvMapLoader クラス **68**
makeGraphic メソッド
  IlvFeatureRenderer インターフェース **49**

O

OGDI
　地図作成 **36**
Oracle Spatial
　ライタ **114**

R

`readRecord` メソッド
　`IlvShapeDBFReader` クラス **98**

S

S57 **122**
　形式 **122**
　データ・ファイル **122**
S57 形式 **122**
`setDebugView` メソッド
　`IlvTiledLayer` クラス **81**
`setDefaultSourceProjection` メソッド
　`IlvMapLoader` クラス **67**
`setEllipsoid` メソッド
　`IlvProjection` クラス **138**
`setSize` メソッド
　`IlvTileController` クラス **81**
sInverse 関数 **156**

い

イメージ・ファイル・リーダ **107**

え

エラー管理 **133**
エラーの管理 **133**
エラーの処理 **133**
円錐図法 **128**
円筒図法 **127**

お

オフセット **136**

き

逆投影
　書く **155**
　球形への **156**
　計算 **133**
　楕円への **156**
逆投影を書く **155**
キャッシュ・アルゴリズム **88**, **89**
球面楕円
　定義 **138**

け

形式
　GeoTIFF **109**
形状ファイル形式
　概要 **95**
形状ファイル・リーダ **95**

こ

コンストラクタの引数 **153**

し

縮尺フィルタ **70**
順投影を書く **154**

せ

正角図法 **129**
正積図法 **129**

た

タイリング・グリッド **78**
　構造 **80**
　サイズ **81**
タイル
　API を介したロード **85**
　空 **81**
　キャッシュ **88**
　キャッシュ済み **81**
　スケール・ビジビリティ・フィルタ **84**

のステータス **81**
リスナ **85**
ロード済み **81**
ロック・カウンタ **74**, **81**
タイル・レイヤ **78**
保存 **88**
楕円
概要 **137**
球面 **138**
赤道半径 **138**
設定 **138**
定義済み **139**
パラメータ **138**
扁平率 **138**
単位コンバータ **142**

ち

地心座標 **136**
地図
読み込み **66**
地図機能
アトリビュート **47**
揮発性 **56**
ジオメトリ **47**
定義 **46**
地図機能イテレータ
概要 **56**
地図機能のアトリビュート
オブジェクトへの付加 **48**
保存 **48**
地図投影図法、例 **134**
地理座標 **136**
地理座標の印刷 **134**
地理参照されたデータ **67**

て

定義済み単位コンバータ **142**
定義済み単位コンバータのリスト **143**
データ
座標の変換 **144**
データ投影 **154**

と

投影結果の出力 **133**
投影図法
ILOG Views による統合 **145**
円錐 **128**
円筒 **127**
概要 **127**
結果の出力 **133**
正角 **129**
正積 **129**
選択 **64**
楕円 **138**
定義 **131**
方位 **129**
メルカトル **143**, **156**
メルカトル図法の例 **131**
例 **134**
投影図法に関連付けたコンバータ **142**
投影図法の作成 **132**
投影図法の定義 **131**
投影図法パラメータ **136**
投影データ **154**

ひ

非球形楕円からのデータの投影 **155**
表記法 **11**

ふ

フィルタ
縮尺 **70**

へ

変換ユーティリティ **137**, **142**
変換用ユーティリティ **137**, **142**

ほ

方位図法 **129**

ま

マップ・リーダ **56**
  `IlvShapeFileReader` **96**
  CADRG ファイル **105**
  DTED ファイル **103**
  `IlvShapeDBFReader` **98**
  `IlvShapeSHPReader` **97**
  `IlvSHPReader` **97**
  形状ファイル **95**
  作成 **57**
  定義済み **56**
マップ・ローダー **66**
  拡張 **68**
  地理参照されたデータ **67**
  および定義済みリーダ **66**
  特定のレンダラを使用する **68**
マニュアル
  構成 **10**
  表記法 **11**
  命名規則 **11**

め

命名規則 **11**
メルカトル図法 **131**, **143**, **156**

ら

ライタ
  Oracle Spatial **114**
ライブラリのインポート **131**

れ

例
  地図投影図法 **134**
  メルカトル図法 **131**
レンダラ
  永続化 **53**
  概要 **49**
  拡張 **54**
  作成 **51**
  デフォルト **50**

ろ

ロード・オン・デマンド
  CADRG 形式 **106**
  DTED 形式 **104**
  新しいデータ・ソースへの実装 **91**
  イベントの管理 **85**
  エラーの管理 **85**
  概要 **73**
  スケール・ビジビリティ・フィルタ **84**
  制御 **83**