IBM ILOG Views

Prototypes V5.3

チュートリアル

2009 年 6 月

著作権の告知

商標

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

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

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

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

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

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

告知

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

目次

このチュートリアルについて

このチュートリアルでは、IBM® ILOG® Views Studio のプロトタイプ拡張機能を使ったビジネス・グラフィック・オブジェクトの作成法ついて紹介します。

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

このマニュアルは、以下の章で構成されています。

- *IBM ILOG Views Studio でプロトタイプを作成する*
- *プロトタイプをアプリケーション・オブジェクトへリンクする*

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 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 Studio でプロトタイプを作成する

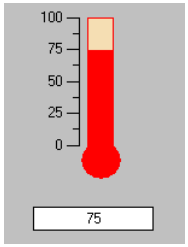
このセクションは、プロトタイプを使った作成法と使用法についての簡単なチュートリアルになっています。チュートリアルを完了するのには約 20 分かかります。以下のセクションが含まれます。

- *プロトタイプの作成*.
- *プロトタイプのアプリケーション・インターフェースを定義する*.
- *プロトタイプで視覚的な表現を定義する*.
- *プロトタイプのグラフィックの振る舞いを定義する*.
- *プロトタイプのインスタンスを作成する*.
- *プロトタイプの対話的な振る舞いを定義する*.
- *プロトタイプのインスタンスを接続する*.

はじめに

このチュートリアルでは、次に示す「温度」と呼ぶオブジェクトのグラフィカル・インターフェースを作成します。温度オブジェクトは、温度のレベルを示す目盛りとゲージのある温度計に似ています。温度が 30 度を超えると、ゲージは赤で表示

されます。温度が 30 度未満の場合、ゲージは青で表示されます。温度計の下にあるテキスト・フィールドに、温度が数字で示されます。



温度オブジェクトには 2 つのアトリビュートがあります。温度アトリビュートと閾値アトリビュートです。温度が閾値の上か下かを視覚的に区別できるフィードバックが必要です。そのため、これら 2 つのアトリビュートを記述することで、温度プロトタイプのアプリケーション・インターフェースを定義します。次に、IBM ILOG Views Studio のグラフィカル・エディタによって、このオブジェクトのグラフィック表現を作成します。最後に、グラフィカルで対話的な振る舞いを作成します。すなわち、アトリビュート値をどのように表示し、温度アトリビュートをどのように編集するかです。

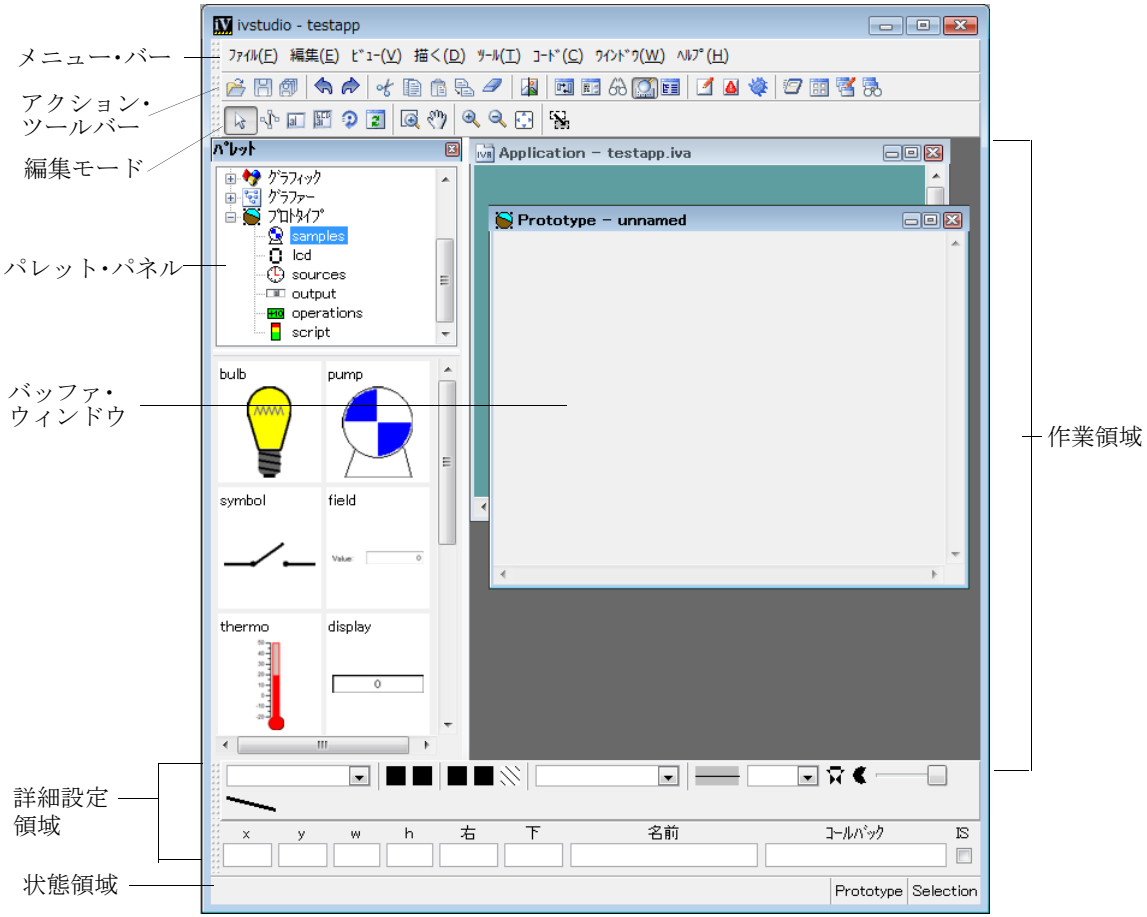
プロトタイプのインスタンスを C++ で書かれた実際のアプリケーションに接続し、データを提供するアプリケーション統合の最終ステップについては、次のチュートリアルで説明します。

プロトタイプ拡張機能付き Studio の起動

プロトタイプ拡張機能付き Studio を、次の手順に従って起動します。

1. IBM ILOG Views の `$ILVHOME/studio/<system>` ディレクトリへ移動します。
2. `ivfstudio` と入力するか、[Studio] アイコンをダブルクリックしてアプリケーションを起動します。

   Studio のメイン・ウィンドウが表示されます。最初にメイン・ウィンドウを開くと、空の 2D Graphics バッファ・ウィンドウが表示されます。パレット・パネルには定義済みのプロトタイプ・ライブラリが、IBM ILOG Views Studio の他の定義済みライブラリとともに含まれます。



プロトタイプの作成

Prototype バッファ・ウィンドウでプロトタイプを作成します。後でライブラリから呼び出して再使用するため、まず格納するライブラリを作成する必要があります。

1. 新しいプロトタイプ・ライブラリを作成するため、[ファイル] メニューから [新規] を選択します。次に、表示されるサブメニューから [プロトタイプ・ライブラリ] を選びます。

   [名前を付けて保存] ダイアログ・ボックスが表示されます。

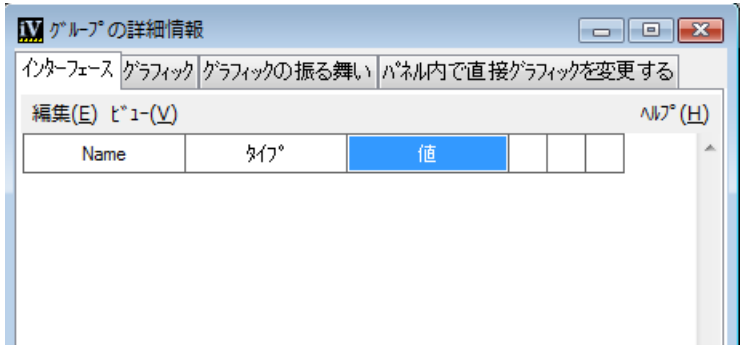
2. 書き込み権限があるディレクトリを選択します。ファイル名 `myLib.ipl` をタイプして、[保存] をクリックします。.

パレット・パネルの上側ペインに表示されるツリーの定義済みプロトタイプ・ライブラリの下に、myLib という新しいツリー・アイテムが追加されます。これで、プロトタイプを格納する新規プロトタイプ・ライブラリが作成されました。

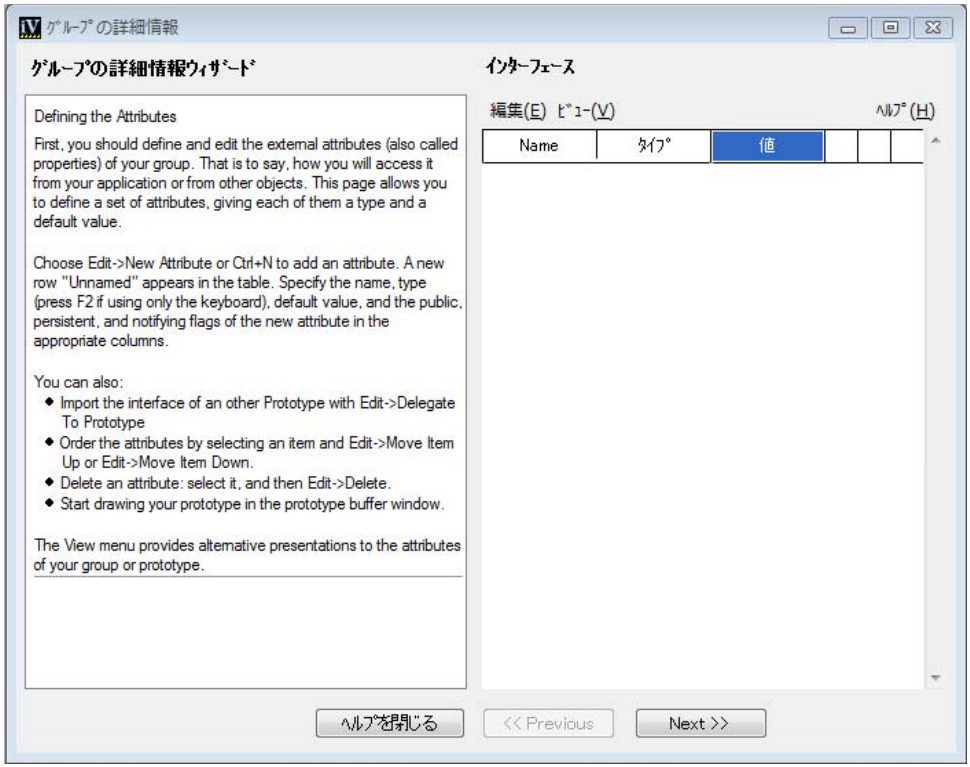
3. 新しい Prototype バッファ・ウィンドウを開くため、[ファイル] メニューから [新規] を選択します。次に、表示されるサブメニューから [プロトタイプ] を選びます。

   [Prototype - unnamed] という Prototype バッファ・ウィンドウが、メイン・ウィンドウに表示されます。

   [グループの詳細情報] パネルも表示されます。



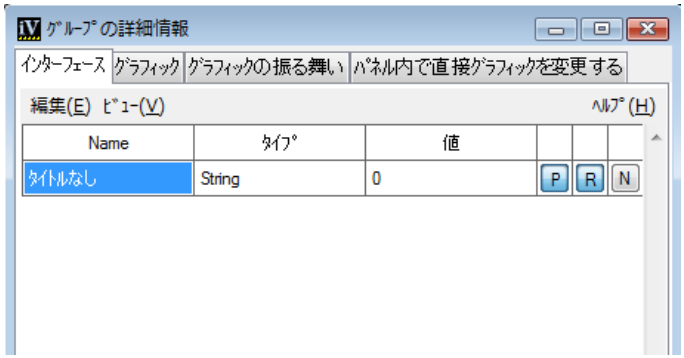
   [ヘルプ] ボタンをクリックすると、プロトタイプの作成プロセスをガイドする文脈依存型ハイパーテキスト・ページが開きます。



プロトタイプのアプリケーション・インターフェースを定義する

アプリケーション・インターフェースは、アプリケーションに設定しそこから取得できるアトリビュート・セットです。これらのアトリビュートによって、プロトタイプの外観が決まります。[グループの詳細情報] パネルの [インターフェース] ページでアトリビュートを追加して、これらを定義します。

1. [編集] > [新規アトリビュート] を選択します。「タイトルなし」という新しいアトリビュートが表示されます。



2. Unnamed (タイトルなし) をクリックしてから、threshold と入力します。これによってアトリビュートの名前が from Unnamed (タイトルなし) から threshold に変わります。

3. [タイプ] 列の String を 2 度クリックするか、F2 キーを押します。アトリビュートのタイプを選択するコンボ・ボックスが表示されます。[値] > [Float] を選択して、アトリビュートのタイプを浮動小数点に指定します。

4. [値] 列にデフォルト値 30 を設定するため、threshold アトリビュート行の 0 をクリックしてから、30 と入力します。

   最後の 3 列は、デフォルト設定のままにします。P を押すとアトリビュートはパブリックになり、R を押すとアトリビュートは永続的になります。N を押すとアトリビュートは他への変更通知を行いません。

5. もう 1 つの、temperature という名前のアトリビュートを作成するため、[編集] > [新規アトリビュート] を選択します。Unnamed (タイトルなし) から temperature に名前を変更します。

6. [タイプ] 列でコンボ・ボックスを開き、そのタイプを Int にするために、[値] > [Int] を選択します。

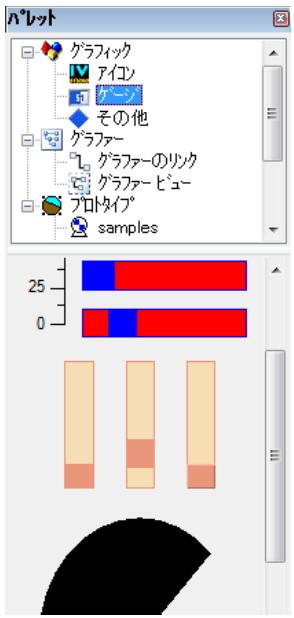
7. 最後に、この temperature アトリビュートで変更を他に通知しないようにします。最終列の N ボタンをクリックして押し、設定します。

> *メモ: 温度アトリビュートの前に閾値アトリビュートを定義してください。* `temperature` *の振る舞いは、* `threshold` *の値によって左右されるからです。*

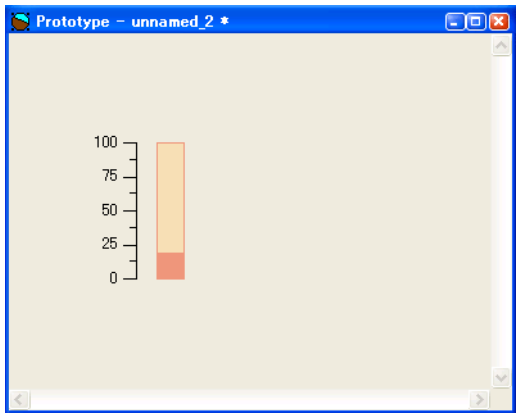
プロトタイプで視覚的な表現を定義する

プロトタイプの視覚的な表現を作成します。

1. 最初に、[グループの詳細情報] パネルの次のタブ (グラフィック・ペ - ジ) を選択します。グラフィック・ページが表示されます。
2. IBM ILOG Views Studio のメイン・ウィンドウで [Prototype - unnamed] をクリックして、Prototype バッファ・ウィンドウを開きます。
3. パレット・パネルの上側ペインで、画面右側のスクロール・バーを使ってパレットのリスト内にグラフィック・アイテム が表示されるまで下にスクロールします。グラフィック・アイテムの下の [ゲージ] を選択すると、ゲージと目盛りの含まれるパレットが表示されます。

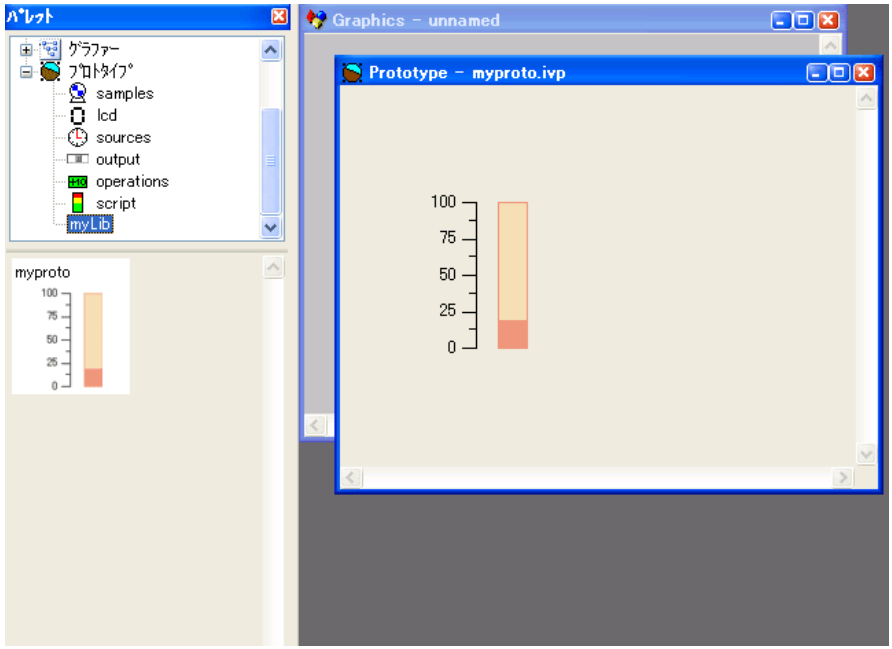


4. 垂直矩形スケール (`IlvRectangularScale`) をパレットから、[Prototype - unnamed] ウィンドウへドラッグします。次に、垂直矩形ゲージ (`IlvRectangularGauge`) を同様に、スケール (目盛り) の隣へドラッグします。ゲージを垂直に拡大し、目盛りに高さを合わせます。



5. [ファイル] メニューから [名前を付けて保存] を選択して、作成したプロトタイプをプロトタイプ・ライブラリに保存します。
6. プロトタイプをライブラリに保存するか尋ねるメッセージ・ダイアログ・ボックスが表示されます。[はい] をクリックします。
7. 新規作成したプロトタイプの保存ライブラリを選択するダイアログ・ボックスが表示されます。ツリー表示のライブラリ・リストで、`myLib` が選択されていること確認します。リストになければ、テキスト・フィールドに `myLib` と入力します。[適用] をクリックします。
8. プロトタイプの名前を尋ねる、新しいダイアログ・ボックスが表示されます。`myproto` を入力して [適用] をクリックします。

   メイン・ウィンドウで、Prototype バッファ・ウィンドウの名前が `myproto.ivp` に変わり、`myproto` というラベルの新しいプロトタイプ・アイコンが `myLib` ページのプロトタイプ・パレットに表示されます。



9. 必要に応じて、メイン・ウィンドウの [ツール] メニューから [グループの詳細情報] を選択し、さらにグラフィック・ページをクリックして、[グループの詳細情報] パネルの [グラフィック・ページ] タブを一番前へ移動します。

   プロトタイプ構造と各ノードのアトリビュートを示す階層シートが、グラフィック・ページに表示されます。

10. グラフィック・ノードの `IlvRectangularScale` を選択します。オブジェクトの名前が良く見えるようにウィンドウを拡大することも、マウス・ポインタを名前の上に移動してそのオブジェクトのフル・ネームが含まれるツールチップを表示させることもできます。

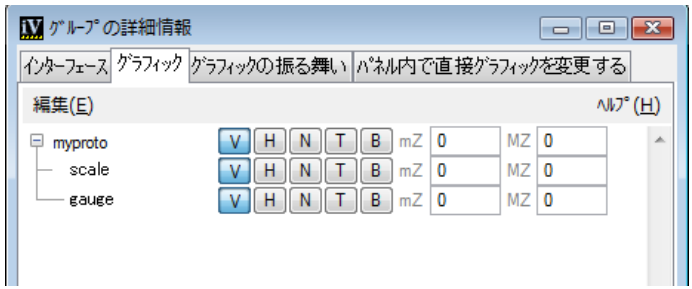
    目盛りはメイン・ウィンドウでも選択できます。

11. フィールドを編集可能にするため再びクリックしてから、`scale` と入力します。Enter キーを押します。

    ツリー・ガジェットの `IlvRectangularScale` を `scale` に変更します。

12. ステップ 8 と 9 を繰り返して、`IlvRectangularGauge` ノードの名前を `gauge` に変更します。

    この結果、[グループの詳細情報] のグラフィック・ページは以下のようになります。



13. メイン・ウィンドウの [ファイル] メニューから [保存] を選択して、プロトタイプを保存します。

プロトタイプのグラフィックの振る舞いを定義する

プロトタイプを構成するインターフェースとグラフィック・オブジェクトの定義が終われば、振る舞いの追加によってそれらのリンクを開始できます。振る舞いによって、プロトタイプ・アトリビュートの変更に伴うグラフィック・オブジェクトの変化を定義します。

1. [グループの詳細情報] の [グラフィックの振る舞い] タブに移動します。
2. アトリビュート・ツリーで `temperature` アトリビュートを選択します。
3. [コントロール] > [代入] を選択します。

   新しい振る舞いが追加されます。次に、この振る舞いのパラメータを指定する必要があります。

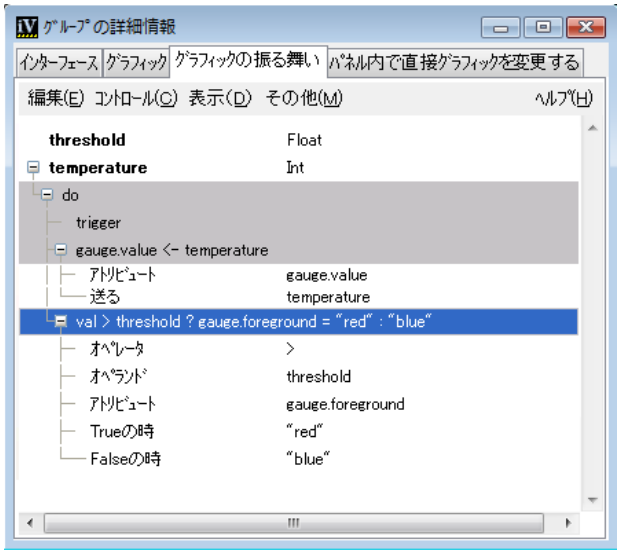
4. 新しく割り当てられた振る舞いの `Attribute` パラメータの右側の列を 2 度クリックして、コンボ・ボックスを開きます。
5. アイテム `gauge` > `value` を選択するか、またはテキスト・フィールドに `gauge.value` と入力して、Enter キーを押します。

6. 割り当てられた振る舞いの `Send` パラメータの右側の列を 2 度クリックして、コンボ・ボックスを開きます。
7. [All Types] を選択するとすべてのパラメータが表示されます。アイテム `temperature` を選択するか、`temperature` とタイプして Enter キーを押します。

この振る舞いを割り当てることにより、温度が変更されるとゲージ・アトリビュートの値が現在の温度の値に設定されます。

次に、2 番目のアクションを `temperature` アクセサに追加します。このアクションでは、温度が閾値を超えるとゲージが赤で表示され、温度が閾値未満の場合ゲージは青で表示されるように指定します。

1. まだ選択されていない場合は、`temperature` アトリビュートを選択します。
2. [コントロール] > [条件] を選択します。これによって新しい振る舞いが、前に割り当てた振る舞いの下に追加されます。
3. この振る舞いには、設定するパラメータが 5 つあります。上で実行したのと同じ手順で右側列を 2 度クリックしてから、表示されるコンボ・ボックスで次に示す値を各パラメータに設定します ([ ] 内は値設定の前に選択するメニュー・オプションです)。
   - Operator : `>`
   - オペランド : [All Types] > [threshold]
   - Attribute : [gauge] > [foreground]
   - True の時 : [Immediate value] を選択してから、テキスト・フィールドに `red` と入力します。(コンボ・ボックスで [選択 ...] を選択してから、色セレクタで色を選択することもできます。)
   - False の時 : [Immediate value] を選択してから、色の入力フィールドに `blue` とタイプします。



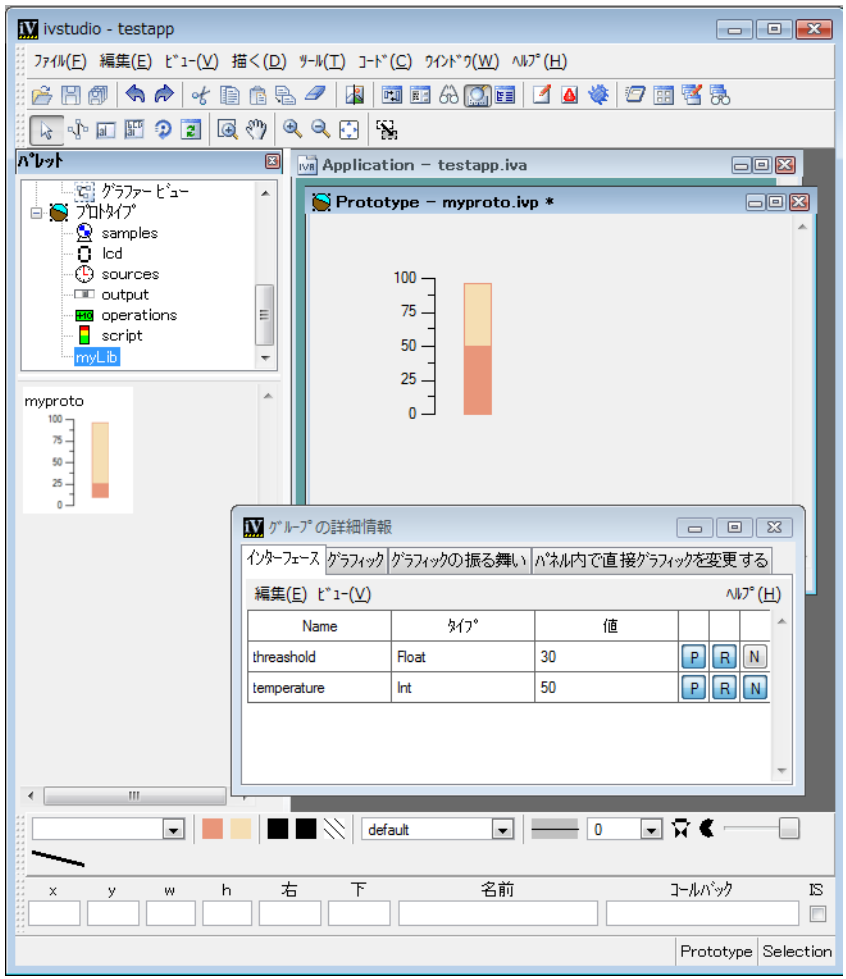
temperature の値の変更は、2 つのアクションを伴います。まず、ゲージの実際のレベルを変更します。次に、条件をテストしてゲージの色をそれに応じて設定します。

プロトタイプ値の変更

次に、プロトタイプの振る舞いをテストするため、temperature アクセサの値を変更します。

1. [グループの詳細情報] パネルの [インターフェース] タブをクリックして、ノートブック・ページを前面に表示します。表示されるアトリビュートのリストでは、temperature の値が 0 となっています。
2. [値] 列をクリックして 0 を 50 に変更します。Enter キーを押します。

   Prototype バッファ・ウィンドウでは、ゲージのレベルが値 50 を示すように変更になり、色が赤に変わります。



3. ゲージの値を 50 から 20 に変更して、Enter キーを押します。

   ゲージのレベルが値 20 を示し、色が青に変わります。

4. [ファイル] メニューから [保存] を選択して、最新の変更を保存します。

これで、プロトタイプの最初のバージョンが作成できました。次に、アプリケーション・バッファ・ウィンドウで、プロトタイプをインスタンス化する方法を学習します。

プロトタイプのインスタンスを作成する

プロトタイプが作成されると、アプリケーション・バッファ・ウィンドウでそれをインスタンス化できます。[グループの詳細情報] パネルを読み込むと、アプリケーションでプロトタイプのアトリビュートにアクセスできます。または、アプリケーション・オブジェクトを示すインスタンスが動的に追加できます。

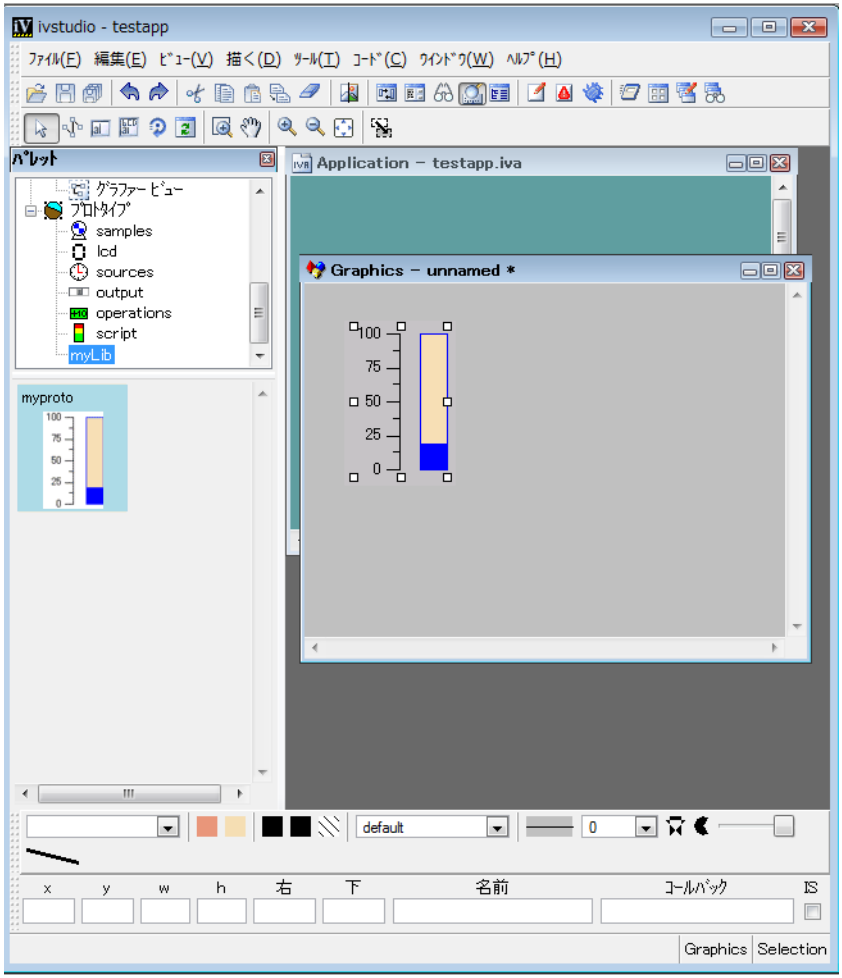
アプリケーション・バッファ・ウィンドウで、プロトタイプのインスタンスを作成します。

1. バッファ・ウィンドウを開くには、[ファイル] メニューから [新規] を選択して、表示されるサブメニューから [2D グラフィック] を選択します。

   メイン・ウィンドウにバッファ・ウィンドウが表示されます。このバッファ・ウィンドウに、プロトタイプまたは他のグラフィック・オブジェクトを保持できます。

2. `myproto` プロトタイプ・アイコンを、`myLib` プロトタイプ・パレットからメイン・ウィンドウへドラッグします。

   プロトタイプのインスタンスが作成され、選択されます。



3. [グループの詳細情報] パネルに移動します。

   インターフェース・ページと、グラフィック・ページだけが有効になっています。他のノートブック・ページは無効なので、振る舞いの追加やプロトタイプ・インスタンスのノード変更はできません。ただし、温度および閾値の値は変更できます。また、ノードのグローバル・プロパティの一部も変更できます。たとえば、拡大 / 縮小機能などです (作成したインスタンスは、メイン・ウィンドウで選択されたままにします)。さらに、プロトタイプのインスタンスをいくつか作成して、それぞれに異なる値を与えることもできます。

   作成したプロトタイプのインスタンスにアクセスできるように、作成したインスタンスに意味のある名前を付けます。これは、次の手順に従います。

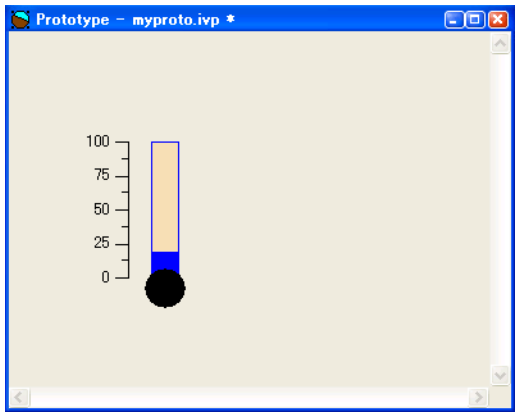
4. 選択されていない場合は、作成したばかりのインスタンスを選択します。
5. [グループの詳細情報] パネルのグラフィック・ページで、[名前] フィールドに myThermometer と入力します。

   IlvGroupHolder クラスの getGroup(“myThermometer”) メソッドを使うと、C++ プログラム内でそれを取得できます。
6. [ファイル] メニューから [名前を付けて保存] を選択し、ファイル名 myPanel.ilv でこのウィンドウを保存します。

プロトタイプの変更

温度計に外見を似せるため、作成したプロトタイプを変更します。

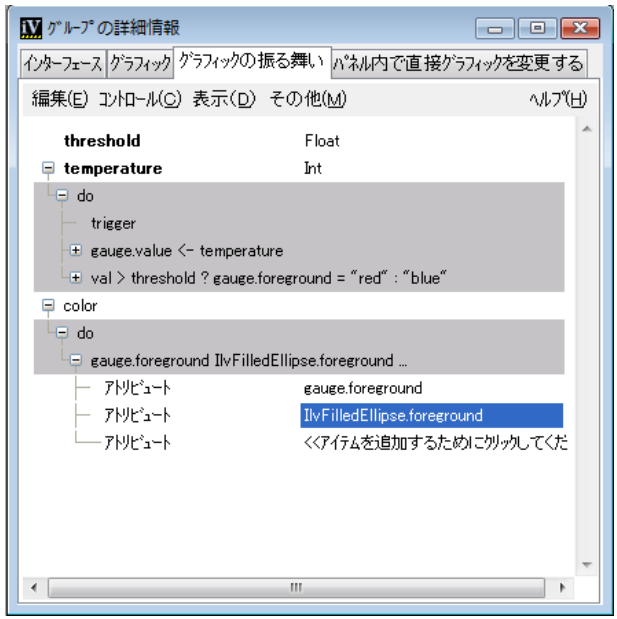
1. メイン・ウィンドウの [ウィンドウ] メニューから、myproto を選択し、Prototype バッファ・ウィンドウを前面に表示します。
2. パレット・パネルの上側ペインで、ツリーを下へスクロールし、Graphics をクリックします。
3. 下側ペインに表示される [グラフィック] パレットから、塗りつぶし楕円 (IlvFilledEllipse) を myproto.ivp ウィンドウへドラッグします。
4. 楕円の形状とサイズを変更します。楕円をゲージの下に置き、2 つのグラフィック・オブジェクトの組み合わせて温度計に見えるようにします。



5. [グループの詳細情報] パネルの [グラフィックの振る舞い] タブをクリックして、対応するノートブック・ページを前面に表示します。

6. 現在、選択されているアトリビュートがないことを確認します。必要ならば、[編集] > [解除] を選択するか、またはマウスの中央ボタンで選択されているアトリビュートをクリックして選択を解除します。
7. [コントロール] > [複数] を選択します。これによって、[Action] という新しい中間アトリビュートが作成されます。
8. `Action` というラベルをクリックしてから `color` とタイプして、このアトリビュートの名前を設定します。Enter キーを押します。
9. 最初の `Attribute` パラメータの右側列を 2 度クリックしてから、[All Types] を選択します。コンボ・ボックスのメニューから [gauge] > [foreground] を選択します。代わりに直接、`gauge.foreground` と入力して、Enter キーを押すこともできます。
10. 「<< アイテムを追加するためにクリックしてください >>」というラベルが付いた 2 番目のパラメータを 2 度クリックします。[All Types] を選んで、コンボ・ボックスのメニューから `IlvFilledEllipse` > `foreground` を選択します。

ゲージと楕円両方の前景色を設定された色に対応させる、`color` 中間アトリビュートがこれで追加できました。



`gauge.foreground` アトリビュートの代わりに `color` アクセサを使うには、`temperature.Condition` アクセサの編集が必要です。

1. [グラフィックの振る舞い] タブのツリーで、temperature > Condition アクセサを、> threshold ? gauge.foreground =... 行の左にある [+] ボタンのクリックにより展開します。

   gauge.foreground パラメータがページ右側の行列内に表示され、編集できるようになります。

2. このパラメータを 2 度クリックするとコンボ・ボックスが開くので、color を選択します。

3. これによって以前のアクセサ定義が、温度を変更すると楕円とゲージ両方の前景色が変わる新しい定義に置き換わります。

4. メイン・ウィンドウの [ファイル] メニューから [保存] を選択して、プロトタイプを保存します。

[インターフェース] ページに戻って温度値を変更してプロトタイプの新しい振る舞いをテストし、この条件付き振る舞いが楕円とゲージの両方に影響することを確認できます。

最後に、これらの温度アトリビュートを簡略化します。この温度は常に内部の gauge.value アトリビュートに等しいので、タイプを変更できます。単純な Int 値とする代わりに、この温度アトリビュートに直接 gauge.value を参照させることができます。これを行うには、次の手順に従います。

1. [グループの詳細情報] の [グラフィックの振る舞い] タブに移動します。

2. temperature アトリビュートの右側にある、タイプを示すラベル Int を 2 度クリックします。コンボ・ボックスが表示されます。

3. コンボ・ボックスのメニューで、[リファレンス] > [gauge] > [value] と選択します。

   Int タイプが ^gauge.value に置き換えられます。これは、温度の変更が直接 gauge.value サブ・アトリビュートの設定に、同等の値で反映されることを意味します。

4. アトリビュート・ツリーの gauge.value=temperature 行を選択します。次に、[編集] > [削除] を選びます。

   これは、冗長な割り当てを削除する効果があります。

プロトタイプの対話的な振る舞いを定義する

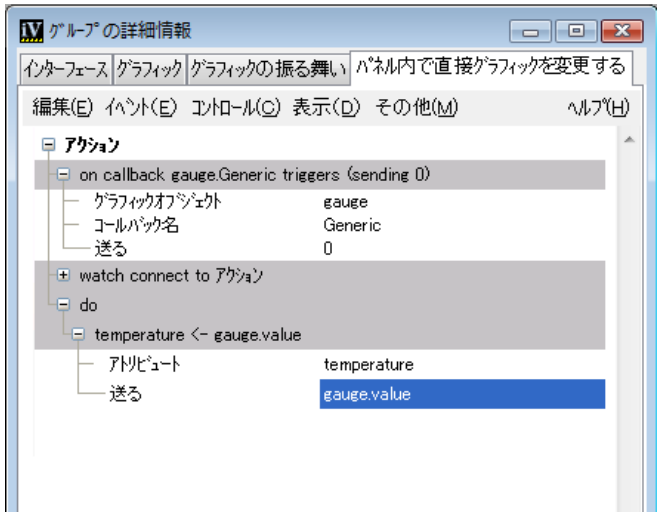
プロトタイプの対話的な振る舞いを定義します。これは、[グループの詳細情報] の最後のタブ、[パネル内で直接グラフィックを変更する] タブで定義するイベントの振る舞いを追加して行います。

1. メイン・ウィンドウの [ウィンドウ] メニューから、myproto を選択し、Prototype バッファ・ウィンドウを前面に表示します。
2. [グループの詳細情報] パネルの [パネル内で直接グラフィックを変更する] タブをクリックします。
3. メニューから [イベント] > [コールバック] を選択します。

   Callback の振る舞いおよび Watch の振る舞いとともに、アクションという名前の新しいアトリビュートが作成されます。
4. ページ右側の行列内で、グラフィック・オブジェクト・パラメータを gauge に設定します。コールバック名と送るパラメータの値は、それぞれデフォルト設定の Generic および 0 のままにします。

   Callback アクセサでは、(ユーザ入力によって) ゲージの Generic コールバックが呼び出されると温度値が問い合わされ、その戻り値が Action 値の変更をリスンするオブジェクトのすべてに送信されます。これらの「リスナ」は、同じプロトタイプ、他のプロトタイプ、またはアプリケーション・オブジェクトの他のアクセサという場合があります。デフォルトでは、Callback の振る舞いの追加は、それ自体に「Watch」の振る舞いを追加します。
5. メニュー・バーから、[コントロール] > [代入] を選択します。

   新しく割り当てた振る舞いが、Action の振る舞いの最後に追加されます。
6. 右側の行列で、パラメータの値を次のとおり選択します。
   - アトリビュート : temperature
   - 送る : gauge > value



ユーザのアクションを介してコールバックがトリガされると、温度アトリビュートがゲージ値に一致するように調整されます。

7. メイン・ウィンドウの [ファイル] メニューから [保存] を選択して、プロトタイプを保存します。

プロトタイプの対話的な振る舞いは、次の手順でテストできます。

1. マーキング・メニュー・コマンド (right, then left) を発行するか、編集モードのツールバーの [アクティブ] アイコン  をクリックして、Prototype バッファ・ウィンドウでアクティブ・モードを選択します。
2. ゲージ・オブジェクト内をクリックして、レベルを上または下へドラッグします。ゲージが更新されるとともに、閾値の前後で色も変化します。[グループの詳細情報] の [インターフェース] タブでは、インタラクションを終了するたびに温度が更新されます。

プロトタイプの編集

プロトタイプを作成して保存したら、パレットでプロトタイプを選択し、[描く] > [グループの編集] (Ctrl+E) の順にクリックして選択したプロトタイプを編集することができます。

変更を保存するとき、すべての変更が作成したインスタンスに伝播されます。

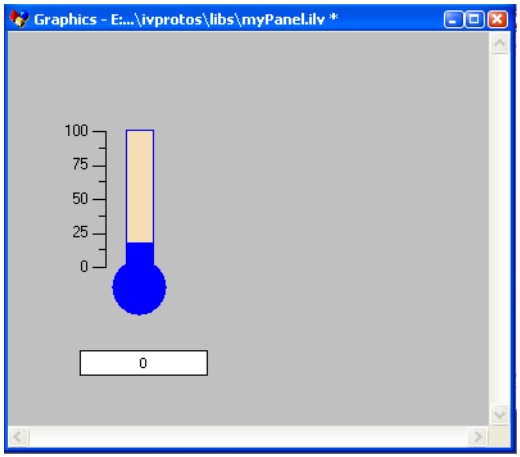
たとえば、sample パレットで bulb を選択し、アイコン上でダブルクリックすると、プロトタイプの編集バッファが開き、[グループの詳細情報] ページでインターフェースとグラフィックの振る舞いを編集することができます。

プロトタイプのインスタンスを接続する

異なる 2 つのプロトタイプのインスタンスを接続する方法を学習します。

1. メイン・ウィンドウの [ウィンドウ] メニューから、`myPanel` を選択します。
2. 2D Graphics バッファ・ウィンドウには、このチュートリアルで作成したプロトタイプのインスタンスが含まれているはずです。含まれていない場合は、プロトタイプ・パレットから `myproto` アイコンをバッファ・ウィンドウにドラッグしてこのプロトタイプの新しいインスタンスを作成します。
3. パレット・パネルで、プロトタイプ・パレットの `samples` を選択します。
4. Display アイコンをバッファ・ウィンドウにドラッグします。

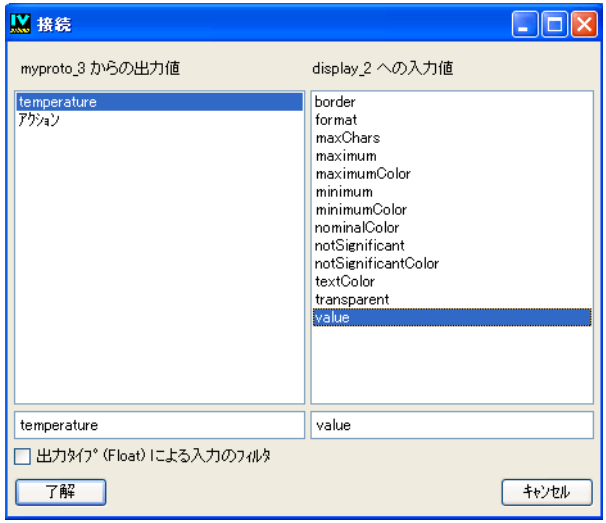
   これで、`myproto` プロトタイプ・インスタンスと同じパネルに `display` プロトタイプのインスタンスが作成されます。



5. [編集モード] ツールバーから、[グループ接続] インタラクタ  を選択します。

6. myproto プロトタイプ・インスタンス内をクリックします。myproto インスタンスから display インスタンスへラインをマウスでドラッグします。ドラッグを開始する際は、空白スペースからではなくグラフィック・オブジェクトからドラッグします。

   マウスのボタンを放すと [接続] ダイアログ・ボックスが表示され、myproto インスタンスのどの出力値を display プロトタイプのどの入力値に接続するのか尋ねられます。



7. myproto でのトリガ値は temperature です。左ペインで temperature を、右ペインで value を選択します。[OK] ボタンをクリックします。

   myproto インスタンスから display インスタンスへ向かう緑の矢印が表示され、接続の完了を伝えます。

   温度が変わるたびに、ユーザまたはアプリケーションのどちらかによって display インスタンスの value アクセサに通知され、その値に応じてディスプレイの数値が変化します。

8. このアクションをテストする場合は、[編集モード] ツールバーで [アクティブ] アイコン  をクリックしてアクティブ・モードに切り替えます。

   マウスのドラッグによってゲージのレベルを変えます。display インスタンスのレベルが、ゲージのレベルの変化に応じて変わります。

これで、最初のプロトタイプ・チュートリアルが完了しました。

まとめ

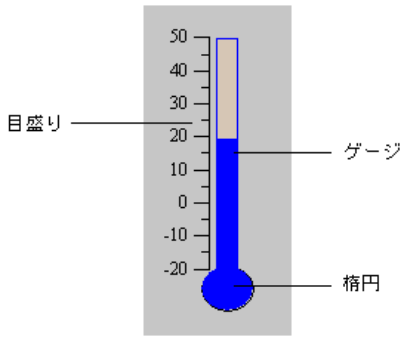
このチュートリアルで作成したプロトタイプは、サンプル・ライブラリにある定義済みプロトタイプの 1 つ、thermo プロトタイプの簡易バージョンです。thermo プロトタイプは (作成した myproto プロトタイプと同じく) 3 つのコンポーネントで構成されます。

- **アプリケーション・インターフェース** : パブリック値で構成され、アプリケーションによるプロトタイプの操作法を定義します。

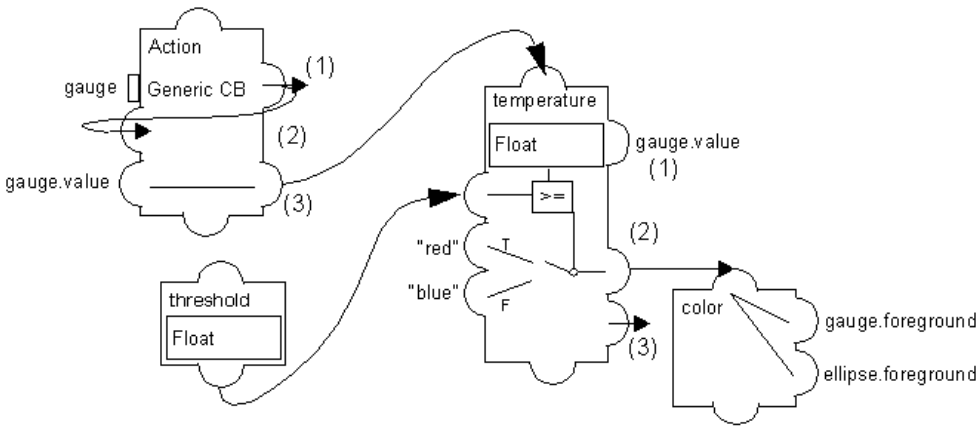
  Threshold: Float で、ユーザの注意を喚起するための警告温度です。

  Temperature: Int で、これが主要な値です。

- **グラフィック表現** : ユーザがアプリケーション・インターフェースの値を認識する方法です。3 つのオブジェクト、目盛り、ゲージ、楕円で構成されます。



- **グラフィックの振る舞いとインタラクティブな振る舞い**: 振る舞いは、アクセサ間の関係を示す次のデータフロー図によって表されます。



このデータフロー図の目的は、プロトタイプの振る舞いが定義されるアクセサ間の依存関係を概略的に示すことです。この時点では、アクセサを示すこの図の規則を理解する必要はありません。

プロトタイプの振る舞いが、以下の 4 つのアトリビュートによって定義されることがわかれば十分です。

- `threshold` - `temperature` 値の設定時に使用される浮動小数点を保持します。`temperature` に依存するため、このアトリビュートは `temperature` の前に定義することが重要です。プロトタイプが読み戻されると、`temperature` が評価される前に `threshold` はその値に設定されます。したがって条件アクセサは、最初の評価に当たって正しい値をテストします。機能する順序で振る舞いが実行されることを確認する代替手段は、`temperature=temperature` を `threshold` の振る舞いに割り当てる、もう 1 つの振る舞いを追加することです。これによって、`threshold` の変更時に `temperature` の再評価が強制されます。
- `temperature` - 次の振る舞いを保持します。

  (1) リファレンス・アクセサ : `gauge.value` 内部アトリビュートを参照します。`gauge.value` は、プロトタイプのサブアトリビュートを呼び出すこともできます。

  (2) 条件アクセサ : `s` 温度が閾値を超えると、`red` が色アトリビュートに送られます。そうでない場合、`blue` が送られます。

  (3) Notify アクセサ : 値を変更すると、このアトリビュートによって変更が他へ通知されます。
- `color` - 複数アクセサで、値を楕円とゲージの前景色へ送信します。

- `Action` - 次のアクセサを保持します。

  (1) コールバック・アクセサ : ユーザがゲージの Generic コールバックを (クリックまたはドラッグで) トリガするたびに、ゲージ値の変化を常に認識する必要のある他のアクセサすべてに値が送信されます。

  (2) Watch アクセサ : アクション・アトリビュート (それ自体) の変化を常に認識する必要があることを示します。

  (3) 代入アクセサ : `Action` アトリビュートを変更するたびに、`gauge.value` サブ・アトリビュートが必ず `temperature` 値に代入されることを示します。これはやや冗長ですが、`temperature` は `gauge.value` への参照でありながら、`temperature` アトリビュートの他の振る舞い (条件および Notify) を解釈する必要があります。

最後にプロトタイプのインスタンスを、データを提供する実際のアプリケーションに接続できます。これは、次のチュートリアルのトピックです。

また、振る舞いメカニズムのリソースをより詳しく検証したり、より複雑な振る舞いの作成法を調べることもできます。IBM ILOG Views Studio に付随するサンプル・ライブラリは、プロトタイプで作成できるさまざまな効果を詳細に調べるための格好な出発点を提供します。

定義済みライブラリ

IBM ILOG Views に含まれる定義済みプロトタイプは、次に示す各種のライブラリで一覧できます。Prototype Studio は起動時にこれらのライブラリを読み込みます。

ライブラリ名	説明
`samples`	起動時に読み込まれるサンプル・ライブラリ
`sources`	値ソースを含むプロトタイプ
`output`	ガジェットおよび定義する出力値を含むプロトタイプ
`lcd`	LCD ディスプレイ (1 桁および 4 桁)
`operations`	プロトタイプの接続およびこれらの値への操作の実行に使用されるプロトタイプ
`script`	スクリプト・アクセサを使用するプロトタイプ

これらのプロトタイプ・ライブラリの 1 つを開くには、パレット・パネルの上側ペインでプロトタイプ・パレット内の名前をクリックします。

プロトタイプ・アイコンをダブルクリックしてプロトタイプ定義を表示できます。これで Prototype バッファ・ウィンドウにプロトタイプが読み込まれ、グループの詳細情報パネルが開きます。プロトタイプ・インスタンスを含むパネル・ファイル

を読み込むとき、必要なプロトタイプ・ライブラリが自動的にプロトタイプ・パレットに読み込まれます。

実際のアプリケーションに近いサンプルは、ILOG Studio の `$ILVHOME/samples/protos` ディレクトリにあります。これらの各サンプルは、プロトタイプ・ライブラリ (通常、サンプル・データのサブディレクトリ内にあります) を使用するサンプル・プログラムのあるサブディレクトリに同梱されています。

完全な詳細については、ライブラリの `$ILVHOME/samples/protos` にある index.html ファイルを参照してください。

章

2

プロトタイプをアプリケーション・オブジェクトへリンクする

このチュートリアルでは、プロトタイプをアプリケーション・オブジェクトにリンクする手順を紹介します。チュートリアルを完了するのには、約 40 分かかります。次の基本タスクを実行する方法について学習します。

- プロトタイプのインスタンスが含まれるパネルを読み込みます。
- アプリケーション・データから駆動するインスタンスを取得します。
- ユーザ・イベントを扱うグループ・メディエータを作成し、アプリケーションに設定します。
- アプリケーション・オブジェクトにグループ・メディエータをリンクします。

はじめに

このチュートリアルは、次の 3 つのステップで構成されます。

- *ステップ1: プロトタイプのインスタンスを含むパネルの読み込み*
- *ステップ2: プロトタイプ・インスタンスの取得と変更*
- *ステップ3: ユーザ・フィードバックを処理するグループ・メディエータの作成*

チュートリアルを開始するには、$ILVHOME/doc/tutorials/protos/tutorial2 にあるファイルを書き込み権限のあるディレクトリにコピーします。これらのファイルを基本テンプレートとして、チュートリアルを進めていきます。各ステップについては、stepX ディレクトリ (X はステップの番号) 内に対応するファイルと、ユーザ環境に応じた指示を作成するファイルが用意されています。

最初に、プロトタイプ拡張機能付き IBM ILOG Views Studio を起動して、myPanel.ilv ファイルを開いて内容をチェックします。最初のチュートリアルの作成内容に似たプロトタイプのインスタンスがあります (*IBM ILOG Views Studio で プロトタイプを作成する* を参照してください)。このインスタンスのプロトタイプは myLib.ipl ファイルに保持され、ファイルを開くと自動的に表示されます。

ステップ 1: プロトタイプのインスタンスを含むパネルの読み込み

step1/program.cpp ファイルには、すでにユーザにはなじみのある IBM® ILOG® Views プログラムの基本キャンバスが含まれます。

プロトタイプ・ライブラリを動作させるには、プロトタイプ・モジュール専用の次のファイルをインクルードする必要があります。

```
#include <ilviews/protos/proto.h>
#include <ilviews/protos/protogr.h>
#include <ilviews/protos/groupholder.h>
#include <ilviews/protos/allaccs.h>
```

> ***メモ:*** *マネージャなど、IBM ILOG Views の他の機能を使用する場合は、他のインクルード・ファイルが必要になります。*

必須ではありませんが、デフォルトのグラフィック・オブジェクトすべてをメイン・ファイルにインクルードしておくと便利です。これによって、個々のグラフィック・オブジェクトがインクルードされているかどうかそのたびに確認することなく、IBM ILOG Views Studio でプロトタイプを編集できます。プロトタイプの変更または拡張が終わったら、ランタイムのフットプリントを最適化するため、必要なオブジェクトだけを含めることができます。

```
#include <ilviews/graphics/all.h>
```

ここで扱う例では、IlvMessageLabel クラスを使用する display プロトタイプを読み込むために、次のファイルのインクルードも必要です。

```
#include <ilviews/gadgets/msglabel.h>
```

定義済みガジェットのすべてが含まれる IBM ILOG Views Controls をインストールしている場合は、プロトタイプでガジェットを使用できます。

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

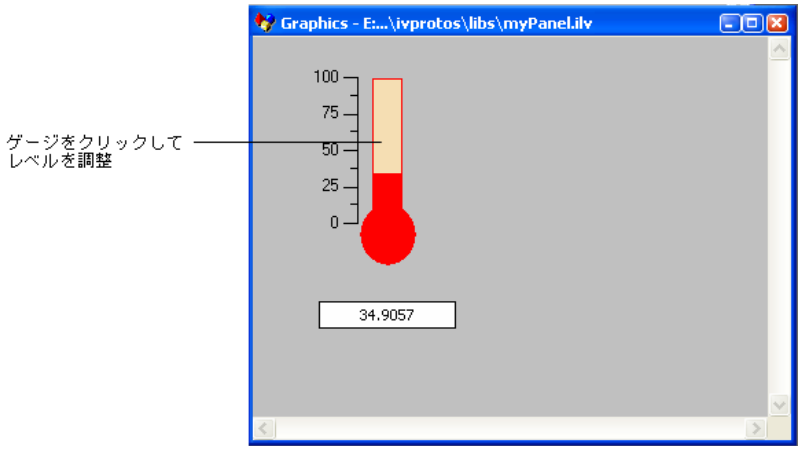
Prototype パッケージの呼び出しが含まれるプログラムの実行には、(少なくとも) プログラムを次に示すライブラリにリンクさせる必要があります。すなわち、ilvproto、ilvgrapher、ilvmgr、views、ilog および使用プラットフォームの専用ライブラリです。このうち、ilvproto ライブラリだけが Prototype パッケージ専用です。

> ***メモ:*** *提供される Makefile とプロジェクト・ファイルにはすでに、プログラムのコンパイル、リンク、実行に必要なすべてのライブラリが指定されています。*

サンプル・プログラムはまずディスプレイを開きます。次に、定義済みのプロトタイプが保持されるプロトタイプ・ライブラリ myLib.ipl の場所を示すため、ディスプレイ・パスに「..」を追加します。次に、プログラムはコンテナを作成し、プロトタイプのインスタンスが含まれる myPanel.ilv ファイルを読み込みます。これらのインスタンスは自動的に読み込まれます。最後に、コンテナを表示してから通常のすべての IBM ILOG Views プログラムと同様に、メイン・ループを開始します。

```
int
main(int argc, char* argv[])
{
  // Connect to the display system.
  IlvDisplay* display = new IlvDisplay("ViewsBGO", 0, argc, argv);
  if (!display || display->isBad()) {
    IlvFatalError("Couldn't open display");
    return -1;
  }
  // Prepare a window and a manager to display a scene.
  IlvManager* manager = new IlvManager(display);
  IlvView* view = new IlvView(display, "BGO tutorial 1",
        "BGO tutorial 1", IlvRect(0, 0, 400, 300));
  manager->addView(view);
  // Load a data file into the manager
  // Add the samples data directory to the display path
  // to be sure to find the proper file.
  IlString buf (getenv("ILVHOME"));
  buf+="/doc/tutorial2/step1";
  display->appendToPath(buf);
  display->appendToPath("..");
  manager->read("myPanel.ilv");
  // The prototype instances are automatically read.
  // Then run the main loop.
  manager->setDoubleBuffering(view, IlvTrue);
  view->show();
  IlvMainLoop();
  return 0;
}
```

このプログラム (step1/program.cpp) をコンパイルしてリンクし、パネルが読み込まれたことを確認できます。以下に、パネル内の操作を示します。



次のセクションで、最初のチュートリアルのパネルで作成した (*IBM ILOG Views Studio でプロトタイプを作成する*を参照) プロトタイプのインスタンス、`myThermometer` を取得するためにコードを追加します。

ステップ 2: プロトタイプ・インスタンスの取得と変更

パネル内のプロトタイプ・インスタンスに行った操作のすべては、コンテナであれマネージャであれ、(また、プロトタイプのインスタンスのスーパークラスである `IlvGroup` に対するものであれ)、すべて `IlvGroupHolder` を使用しています。このクラスによって、`IlvGroup` クラスのオブジェクトをパネルに追加することも、またパネルからオブジェクトを削除したり取得することもできます。

ファイルを読み込んだ後、マネージャに関連するグループ・ホルダを取得します。次に、グループ・ホルダからプロトタイプのインスタンス `myThermometer` を取得し、その温度アトリビュートを `0.0` に初期化します。

```
// Insert the following code after the line:
// "the prototype instances are automatically read."
  IlvGroupHolder* holder=IlvGroupHolder::Get(manager);
  IlvGroup* myThermometer = holder->getGroup("myThermometer");
  if (!myThermometer) {
    IlvFatalError("This program expects to find an IlvGroup "
                  "of name 'myThermometer' in the file 'myPanel.ilv'");
    return -1;
  }
  myThermometer->changeValue(IlvValue("temperature", 0.0));
```

さらに、アプリケーションによって提供される値から、温度値を定期的に更新する必要があります。このため、次の部分をプログラムの main() 関数の前 (include 命令の後) に挿入して、タイマ・ルーチンを作成します。

```
// Insert the following code before the main() body.
const IlUInt angleincrement = 4;
static void
TimerProc(IlvTimer*, IlvAny arg)
{
static IlUInt angle=0;
    IlvGroup* thermometer=(IlvGroup*) arg;
    IlDouble temperature=50.0+40.0*sin(degreesToRadians(angle));
    // Feed in temperature values.
    thermometer->changeValue(IlvValue("temperature", temperature));
    angle = (angle + angleincrement) % 360;
}
```

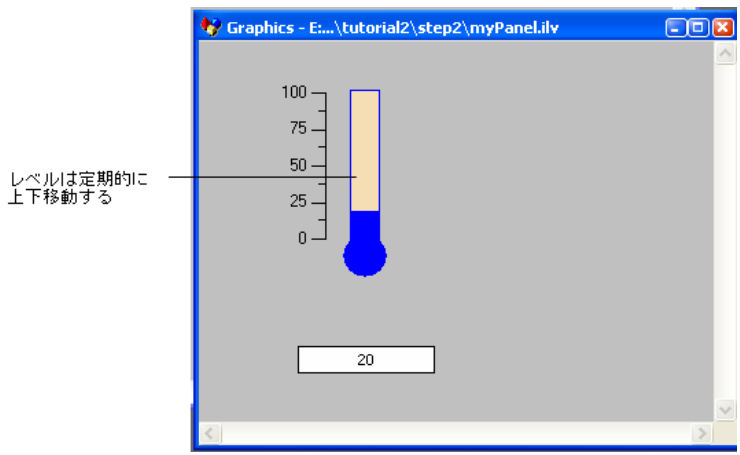
この機能によって、温度は値 10 から 90 の間でシヌソイド関数に従って上下します。次に、このタイマを 200ms ごとにウェイクアップし、パネルから取得した myThermometer プロトタイプのインスタンスを更新するように初期化します。次に示す行の後に、

```
myThermometer->changeValue(IlvValue("temperature", 0.0));
```

以下のコードを挿入します。

```
// start changing the values of the target group
IlvTimer* timer = new IlvTimer(display, 0, 200, TimerProc,
                               (IlvAny)myThermometer);
timer->run();
```

これによって定期的に温度計の値が更新されます。これで、このプログラムをコンパイルして実行できます。温度が規則的に上下して IBM ILOG Views Studio で前に設定した閾値を超えると、温度計が赤に変わるのを確認できます。



次の呼び出しで閾値を変更できます。

```
myThermometer->changeValue(IlvValue("threshold", 50.0));
```

次の呼び出しで閾値を取得できます。

```
IlvValue tval("threshold", (IlDouble)0.0);
IlDouble threshold = (IlDouble) myThermometer->queryValue(tval);
```

ソース・コード一式は、このチュートリアルにおける次のステップの出発点となる `step2/program.cpp` ファイルにあります。

ステップ 3: ユーザ・フィードバックを処理するグループ・メディエータの作成

ディスプレイを駆動するだけでなく、内部パラメータに影響するユーザ入力を処理するアプリケーションを作成する場合は、以下の手順に従います。
IBM® ILOG® Views Studio の `myproto` プロトタイプはわずかに拡張されて、スライダに似たオブジェクトが追加されています。ユーザはこのスライダを使って温度計のアニメーション速度を変更し、温度に応じて上下するレベルの動きを速くしたり遅くしたりできます。

また、IBM ILOG Views Studio の `myLib.ipl` ライブラリを開いて、この効果を達成するために追加されたアトリビュートと振る舞いを確認できます。

アプリケーション・オブジェクトとプロトタイプのインスタンスのリンクには、`IlvGroupMediator` クラスによって実装された `Mediator` デザイン・パターンを使用します。このクラスを使用するには次のファイルのインクルードが必要です。

```
#include <ilviews/protos/grouplin.h>
```

トークン・アプリケーション・オブジェクトの定義

このチュートリアルでは、BGO とともに実装されたユーザ・インターフェースを介して表示し、制御する単純なオブジェクトの温度計を仮定します。

```
#if defined(ILVSTD)
#include <cmath> // for sin() function
ILVSTDUSE
#else
#include <math.h>
#endif
struct Thermometer {
    IlFloat temperature;
    IlUInt acceleration;
    IlUInt curval;
    IlvTimer timer;
// The display argument is here to allow the use of a Views timer.
    Thermometer(IlvDisplay*);
    ~Thermometer() {}
    void adjust_temp();
static void TimerProc(IlvTimer*, IlAny);
};

Thermometer::Thermometer(IlvDisplay* dpy)
:temperature(20), acceleration(4), curval(0),
 timer(dpy, 0, 200, TimerProc, this)      //  see Note that follows
```

```
{
     timer.run();
}

void Thermometer::adjust_temp()
{
   temperature = 50.0 + (40.0 *
                  sin(degreesToRadians((IlDouble)curval)));
   curval = (curval + acceleration) % 360;
}
void Thermometer::TimerProc(IlvTimer*, IlAny arg)
{
   ((Thermometer*)arg)->adjust_temp();
}
```

> ***メモ:*** *一部のシステムでは、上の構文にある「this」の使用に関して警告メッセージが表示されることがあります。ただし、構文は安全なので警告は無視して構いません。*

adjust_temp メソッドは定期的に呼び出されてトグル・アニメーション・タイマを実装します。このクラスは、ユーザ・インターフェースの特定機能を一切提供しません。つまり、他のアプリケーション・オブジェクト（たとえばプラント・シミュレータなど）とともに動作できるスタンドアロンのソフトウェア・ピースとして機能することを意味しますが、その値の対話的な表示または編集を許可する機能は皆無です。本チュートリアルでは、このオブジェクトのユーザ・インターフェースを定義変更せずに作成します。

メイン・プログラムでディスプレイの作成後に、このアプリケーション・オブジェクトのインスタンスを作成します。

```
// create and initialize an application object.
Thermometer myThermometer(display);
```

アプリケーションのクラスの GroupMediator を定義する

Thermometer のインスタンスのユーザ・インターフェースを作成するため、この Thermometer クラスを処理する GroupMediator のサブクラスを定義します。

```
struct TemperatureWatcher: public IlvGroupMediator
{
    TemperatureWatcher(IlvGroup* g, Thermometer* a)
    : IlvGroupMediator(g, a) {
        if(!tempSymbol) tempSymbol=IlvGetSymbol("temperature");
        if(!accSymbol) accSymbol=IlvGetSymbol("acceleration");
    }
    IlBoolean changeValues(const IlvValue*, IlvUShort);
    void queryValues(IlvValue*, IlvUShort) const;
inline Thermometer* getThermometer() const {
            return (Thermometer*) getObject(); }
static IlvSymbol* tempSymbol;
```

```
static IlvSymbol* accSymbol;
};
IlvSymbol* TemperatureWatcher::tempSymbol;
IlvSymbol* TemperatureWatcher::accSymbol;
```

ユーザによる編集時に、Thermometer 値の更新方法を定義する必要があります。changeValues メソッドは、プロトタイプのインスタンスのアトリビュート値と Thermometer メンバ間に、1 対 1 の関係を確立します。

```
// Method handling with user input and updating the application.
IlBoolean
TemperatureWatcher::changeValues(const IlvValue* v, IlvUShort n)
{
    if(locked()) return IlFalse;
    for (IlUInt i=0;i<n;i++) {
        if (v[i].getName() == accSymbol)
           getThermometer()->acceleration = (IlUInt) v[i];
// You do not want the temperature value to be editable by the user.
// If this was the case, the following code would simply be added:
//       else if (v[i].getName() == tempSymbol)
//         getThermometer()->temperature = (IlFloat) v[i];
    }
    return IlTrue;
}
```

どのプロトタイプ・アトリビュートを介して温度計の値がユーザ・インターフェースに反映されるのか、指定する必要があります。ここでは queryValues メソッドが、Thermometer アトリビュートと、それを示すプロトタイプのインスタンス値の間に、1 対 1 の対応を確立します。

```
// Method that synchronizes the group to the application values.
void TemperatureWatcher::queryValues(IlvValue* v, IlvUShort n) const
{
    for (IlUInt i=0;i<n;i++) {
        if (v[i].getName() == tempSymbol)
            v[i] = getThermometer()->temperature;
        else if (v[i].getName() == accSymbol)
            v[i] = getThermometer()->acceleration;
    }
}
```

アプリケーション・オブジェクトを表示にリンクする

この温度計アプリケーションのユーザ・インターフェースをインスタンス化するため、それを監視する myThermometer のインスタンスを作成します。メイン・プログラムで次に示すコードを、グループ・ホルダから myThermometer グループの取得後に （ただしメイン・ウィンドウの表示前に）追加します。

```
TemperatureWatcher watch(tempRep, &myThermometer);
watch.update();
IlvTimer timer(display, 0, 200, TimerProc, &watch);
timer.run();
```

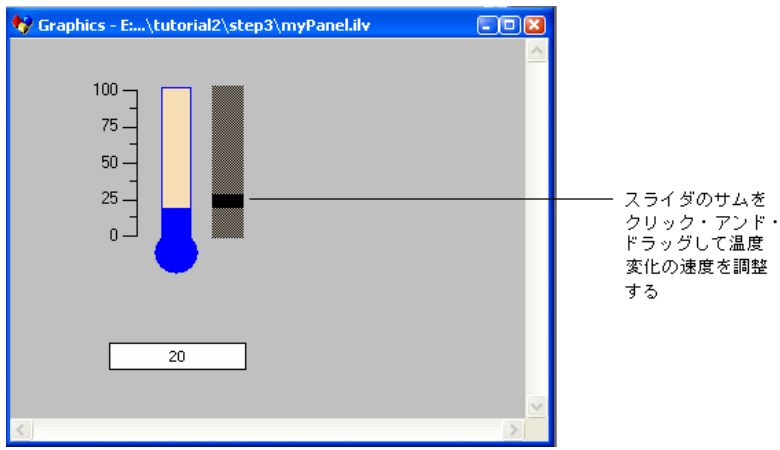
このタイマは定期的に温度計インスタンスの状態をチェックし、必要ならば表示を更新します。

アプリケーション・オブジェクトと表示の同期化

最後に、メイン・プログラムの前に非同期更新ルーチンを定義する必要があります。

```
static void
TimerProc(IlvTimer*, IlvAny arg)
{
    TemperatureWatcher* watcher=(TemperatureWatcher*) arg;
    watcher->update();
}
```

`TemperatureWatcher` は定期的に `Thermometer` オブジェクトの現在値に従って、それ自体を更新します。



> ***メモ:*** *Observer/Observable デザイン・パターンに従う Thermometer クラスの振る舞いを持たせたり、さらにその変更を他のオブジェクトに通知するなど、その他のメカニズムも実装できます。このメソッドの方がわずかに効率的です。*
>
> `$ILVHOME/samples/protos` *にあるサンプルは、このテクニックを実装しています。ただし、トリガされた非同期オブザーバの使用は、リアルタイム制約のあるアプリケーションの実装には役立ちます。厳しい時間的制約に合わせる必要がある場合は、ユーザ・インターフェースの更新頻度を調整できます。*

まとめ

ここで、このプログラムをコンパイルして実行し、その振る舞いをテストできます。オプションとして、step3/program.cpp ファイルをコンパイルして実行することもできます。このファイルはこれまでに述べたチュートリアルを実装した上でいくつかの特別なヒントも追加し、希望する場合は最適化も行えます。ステップ 2 の最後のプログラムと同様に温度のレベルが上下しますが、温度計の右にあるスライダに似たオブジェクトによって加速度を調整できます。

上記のメカニズムによってアプリケーション・オブジェクトの強力なカプセル化を保ちながら、ユーザ・インターフェースの容易な実装が可能になります。これが可能なのは、いったん共通インターフェースを決めてグループ・メディエータを定義すると、機能的なコアの実装をユーザ・インターフェースから完全に分離させ、別々に開発できるように設計されているためです。samples/protos ディレクトリには、BGO をアプリケーションに統合する他のメカニズムも用意されているため、ニーズに応じた最適な手段を見つけることができます。

索引