Ｃ＋＋グラフィック・クラスライブラリＪＧＲの使い方
(ＪＧＲ ｖｅｒ．３．ｘ準拠)

平成 27 年 12 月 8 日版　西井 淳

目 次

1 はじめに

1.1 jgr-3.x の変更点

以下は jgr-2.x から 3.x への主な変更点です。

- EPS ルーチンの制御がコンパイルオプションで与えられていたのを関数による制御に変更。これによりグラフのどの部分を EPS に出力するか等細かな調整が可能になった。
- 関数名の命名規則を統一した。ただし jgr-2.x までの関数名でも利用出来る。
- 座標軸の目盛りのフォントとテキストのフォントを独立に設定可能に (setScaleFont,setTextFont)
- 座標軸の色とグラフ等の図形の色独立に設定可能に
- ウィンドウの背景色設定用関数追加 (setBgColor)
- サンプルプログラムの修正と追加

1.2 jgr の特徴

JGR は X Window 上でグラフィックを描くための C++ クラスライブラリです。複数のウインドウを管理でき, また, 1 つのウインドウ内に複数のグラフを管理できます。プログラムの実行中に随時計算結果をどんどんプロットしていきたい時に便利です。

通常、グラフを描く時には X 軸の座標の単位系（例えば m,s) における値とグラフック画面上での値（ドット）が違いますが、このグラフィック・ツールでは X、Y 軸の長さ、実際の値（m,s など）と画面上での値（ドット）の関係などをはじめに与えておくことによって、あとは自動的に実際の値から座標変換を行なって点や線を描いてくれます。Xlib の関数を使ってますが, Xlib を意識せずに使用できます。表示結果は EPS 形式で出力することもできます。

1.3 動作環境

C++をコンパイル出来, X Window が動いてるシステム上なら動作すると思います。jgr-3.x は Vine Linux 4.x と Mac OS X(Leopard) 上で動作確認しています。

1.4 謝辞

PS 関連ルーチン (jgrPS.cc) は荻原剛志氏の XeasyGraphic をベースにして開発しています。大変感謝です。

1.5 ライセンス

PS 関連ルーチン (jgrPS.cc) は XeasyGraphic-Copyright に、それ以外は LGPL とします。

1.6 その他の注意

バグレポート・パッチその他，西井 (nishii@bcl.sci.yamaguchi-u.ac.jp) まで御連絡頂けますととても嬉しいです。

2 jgr の入手とインストール

2.1 jgr の入手

jgr はhttp://bcl.sci.yamaguchi-u.ac.jp/~jun/?JGRから入手出来ます。Vine Linux で使う場合は rpm 形式のパッケージを，他の環境で使う場合はソースを入手してください。

2.2 jgr のインストール

2.2.1 Vine Linux の場合

rpm コマンドでインストールしてください。ダウンロードしたパッケージ名がjgr-3.0-0vl1.i386.rpmならば以下を実行します.

```
# rpm -ivh jgr-3.0-0vl1.i386.rpm
```

サンプルプログラムやドキュメントは/usr/share/doc/jgr-3.0 以下にインストールされます。インストールされたファイル一覧は以下で確認できます。

```
$ rpm -ql jgr
```

2.2.2 ソースから make する場合

Linux 上の場合は以下のようにします。

```
# ln -sf Makefile.linux Makefile
# make
# make -C doc
# make install
```

Mac OSX の場合は以下の通り。

```
# ln -sf Makefile.mac Makefile
# 以下は Linux の場合と同じ
```

3 jgr の使い方

3.1 ブログラミング時の注意

- jgr は C プログラムで使えますが，コンパイルには C++コンパイラ g++ を使って下さい[1]。C のプログラムのヘッダに C++の標準入出力ライブラリ (stream.h) を追加すれば，g++で C プログラムもコンパイル出来ます。

  ```
  #include <stream.h>
  ```

- jgr を使うには、以下のヘッダファイルを include して下さい。

  ```
  #include <jkit/jgr.h>
  ```

- コンパイル時には libX11, libjgr をリンクして下さい。g++のコンパイルオプションは、-lX11 -ljgr になります

具体的なコンパイルの仕方は後の例を見て下さい.

[1] Intel の C++コンパイラでも動作確認しています。

3.2 サンプルプログラム

このプログラム例は、sample ディレクトリに tiny.cc という名前で置いてあります。サンプルプログラムのあるディレクトリで以下を実行すればコンパイル出来ます。

```
make tiny
```

単に引数無しで make を実行するとサンプルプログラム全部がコンパイルされます。
g++を使うなら以下のようにします。

```
g++ -o tiny -lm -ljgr -L /usr/X11R6/lib -lX11 tiny.cc
```

以下は tiny.cc のソースです。

```
#include <stream.h>
#include <iostream.h>
#include <jkit/jgr.h>    //jgr クラスを使うのに必要

JWindow win;             //開くウィンドウを宣言
Graph   g;               //ウィンドウに描画するグラフを宣言

main(){
  extern JWindow win;             //開くウィンドウを宣言
  extern Graph   g;               //ウィンドウに描画するグラフを宣言
  double xmax, ymax;
  char *msg="Click the Mouse, Please!!";

  // ウィンドウと、そこに描くグラフを登録-----------------------
  win.open(10,10)  // ウィンドウを (10,10) の位置に開く宣言
    .graph(g.open( 500, 50, 200, 40 ))
    // 開いたウィンドウに、x 軸の正負方向が (500,50)、
    //y 軸の正負方向 (200,40) のグラフを登録
    .map()             // ウィンドウを画面上にマップする。
    .mouseWait();    // マウス・クリックの入力を待つ

  //ここからは、グラフ g の属性設定-------------------------------
  xmax = 10.0;  ymax = 100.0;
  g.setRatio( xmax, ymax, 500.0, 200.0 )
    // xmax, ymax の値をそれぞれ、画面上では 500dot, 200dot の長さで描画
    .axis() // X 軸, Y 軸を描画
    .text( 5, -15, msg )  // メッセージをグラフ上の (5,-15) の位置に出力
    .flush()           // グラフィック・バッファにたまってるデータをフラッシュ
    .mouseWait();    // マウス・クリックの入力を待つ

  //メインルーチン：グラフ g 上に線を書く.....
  for( double i=1.0; i<=xmax; i+=0.1 ){
    g.color("blue") // グラフ g の描画色を青にする
      .line( (double)i, 0.0, 0.0, (xmax-(double)i)*ymax/xmax )
      // (i,0.0) から (0.0, (xmax-(double)i)*ymax/xmax) に線を引く
      .flush();
    mouseBreak();  // マウス・クリックがあったら for ループから抜ける
  }

  g.color("black")
    .text( 5, -15, msg) // テキスト表示
    .mouseWait();      // マウス・クリックを待つ
  Xwin.close();          // 終了の儀式
}
```

なんとなく tiny.cc がわかったら他のサンプルプログラムも見てみてください。以下はサンプルプログラムのリストです。

tiny.cc	簡単なグラフを書く
sample.cc	tiny.cc に円や四角などの描画のルーチンを増やしたもの
sin.cc	複数の窓を開き、一つの窓には複数のグラフ描画。EPS 出力も行う
mouse.cc	マウスのクリック情報を取得
font.cc	各種フォントの表示
tinymenu.cc	メニューの利用例
menu.cc	メニューの利用例 2

プログラムに用いられる各関数の詳細は以降の節で紹介します。

4　基本的な関数

この節では、ウィンドウやそこに描画するグラフを初期化する関数について、ほぼプログラム内で宣言する順番で紹介して行きます。

4.1　ウィンドウを開く

```
JWindow& open( unsigned int sx, unsigned int sy,
               unsigned int width, unsigned int height, int bsflag=True );
JWindow& open( unsigned int sx, unsigned int sy, int bsflag=True );
```

- (sx,sy): 左上の角の座標、原点は画面の左上の角
- (width,height): ウィンドウの幅と高さ
- bsflag: バッキングストアを使うかどうか (指定がなければ” 使う”)

ウィンドウのクラス名は JWindow です。例えば 2 つウィンドウを開きたいときには、

```
JWindow win1, win2;
```

と宣言します。さらに、このウィンドウの大きさや位置を以下のように指定します。

- 大きさと位置両方を指定するとき、

  ```
  win1.open(10,10,200,100); //座標 (10,10) を左上の角とし、幅 200dot, 高さ 100dot
  ```

- 位置のみ指定し、大きさは登録するグラフの大きさに応じて自動的に決める時。

  ```
  win1.open(10,10); //座標 (10,10) を左上の角とする
  ```

最後の引数の bsflag は、バッキングストアの指定を付け加えたいときに True を，不要な時には False を指定します。(よくわからなければこの引数は不要です)

4.2 グラフの初期化

```
Graph&  open(int sx, int sy, int x_pos, int x_neg, int y_pos, int y_neg,
           char* clr="black" );
Graph&  open( int x_pos, int x_neg, int y_pos, int y_neg,
           char* clr="black" );
Graph&  open( Graph& g, char* clr="black" );
Graph&  child( Graph& g );
Graph&  child( Graph& g, char* clr );
```

- (sx,sy): グラフの左上の角の座標、原点はウィンドウの左上の角
- x_pos,x_neg: X 軸の正方向・負方向の長さ
- y_pos,y_neg: Y 軸の正方向・負方向の長さ
- clr: グラフの描画色 (”red”,”blue” など X window で使える色名を指定)。指定しなければ黒に設定される。

グラフのクラス名は Graph です。それぞれのグラフについて宣言を以下のように行います。例えば2つグラフを書きたいときには (1 つのウィンドウ上であっても、3 つのウィンドウそれぞれに書くのであっても)、

```
Graph grp1, grp2 ,grp3;
```

と宣言します。次に、これらのグラフの初期化を行います。

- ウィンドウ中でのグラフの位置、大きさを指定するとき
  例えば、ウィンドウ中、(10,10) の位置に X 軸の正負の方向の長さがそれぞれ 100,10, Y 軸の正負の方向の長さがそれぞれ 50,10 のグラフを書きたいときには

  ```
  grp1.open(10,10,100,10,50,10);
  ```

  とします

- ウィンドウ中でのグラフの大きさだけ指定して、位置は自動で決める場合には以下のようにします。

  ```
  grp2.open(100,10,50,10);   //大きさは grp1 と同じ場合の例
  ```

- あるグラフと同じ大きさのグラフを書きたい時には、

  ```
  grp3.open(grp2);
  ```

  とします。この例では grp3 の大きさは grp2 と同じになり、位置は自動で決められます.

いずれの場合にも、描画色の指定をしたいときには、さらに"red", "blue"などを指定した引数を加えてください。この指定は

```
Graph&  color( char* clr="black" );
```

を使って、いつでも変更できます。色指定についての詳細は 5.5節を見てください。

さらに、あるグラフ (親グラフ) と**同じ位置に重ねてグラフ (子グラフ, child graph) を書きたいとき**には、子グラフの初期化に child を使います。(1 つのグラフに複数のデータ曲線を書く時などに使います)。例えば、子グラフ grp3 を 親グラフ grp2 と同じ位置に描く時、

```
grp3.child(grp2);
```

とします。子グラフの描画色は、それぞれ異なる色が自動的に選ばれます。自分で色の設定をしたいときには、

```
grp3.child(grp2,"blue");
```

と、色の指定を加えます。

4.3 ウィンドウにグラフを登録

```
JWindow& graph( Graph& g );
```

前節で初期化したグラフを、描画するウィンドウに登録します。例えば、グラフ grp1 をウィンドウー win1 に描きたい場合には、

```
win1.graph(grp1);
```

とします。また、grp1 の初期化とあわせて、

```
win1.graph(grp1.open(100,10,50,10));
```

などとすることもできます。

4.4 ウィンドウの描画

```
JWindow& map( void );
```

ウィンドウにグラフの登録がすんだところで、ウィンドウをいよいよ X ウィンドウ上に描画します。単に

```
win1.map();
```

とするだけです。ただし、これだけだと X window の処理の問題上データがグラフィックバッファに入るだけですぐに描画されない時があります。そのときには、以下の命令を map コマンドのあとに入れてください。

```
win1.flush();
```

これで、バッファにたまっているデータがすべて描写されます。なおこの命令は、jgr のデフォルトで定義される画面制御のためのクラス Xwin の関数 (**??**節) により、

```
Xwin.flush();
```

を用いるのと等価です。

4.5 グラフ描画に関する初期設定

```
Graph& setRatio( double realx, double realy, double scrx, double scry );
Graph& setRatio( Graph& g );
Graph& setRatio( void );
```

- (realx, realy): 実際の数値
- (scrx, scry): スクリーン上での値 (ドット)
- jgr-2.x での関数名: `set_ratio`

計算で得られる数値の大きさと、画面上での大きさの関係を設定します。例えば、グラフ grp1 上で、X 軸の時間 10s を画面上で 100dot、Y 軸の速度 5m/s を画面上で 60dot で表したい時には、

```
grp1.setRatio(10,5,100,60);
```

とします。また grp2 での設定は grp1 と同じにしたい時には、

```
grp1.setRatio(grp2);
```

さらに、グラフを重ねて描画している child グラフの場合には単に

```
grp3.setRatio();
```

とすれば、親グラフと同じ設定になります。

この関数の設定後、線などを描画する時の座標の入力は、スクリーン上での大きさを気にすることなく、ただ、(時間、速度) といった実際の値を入力すれば描画ができます。

なお、この関数を宣言しない時の、スクリーン上の値と実際の値の比は 1:1 に設定されています。

4.6 その他初期化に関する関数

```
JWindow& addLeftMargin(long xmargin);
```

ウィンドウに位置の自動割当でグラフを書いたとき、グラフの左にマージンがもうすこし欲しい場合に使う クラス JWindow の関数です. ウィンドウの map 前に指定してください。

5 いろいろな関数

各ウィンドウの操作には JWindow クラスのメンバー関数を使います。画面上に作ったウィンドウには複数のグラフを登録して描画することができ，各グラフの操作には Graph クラスのメンバー関数を使います。

また，すべてのウィンドウの操作のために、XController というクラスがデフォルトで定義されています。以下の宣言が `jgr.h` でされており，変数 Xwin に対してメンバー関数を呼び出す事で，全てのウィンドウに対するいくつかの操作を行うことができます。

```
XController Xwin;
```

以下では JWindow,Graph クラスや，変数 Xwin に対して使えるいろいろなメンバー関数を紹介します。各関数がどのクラスや変数に対するものかは各関数名の前に [] で囲って記しています。複数のクラスに対して同じ名前のメンバー関数がある場合，特に明記していない限りその関数は，どのクラスに対しても同じ動作をします。

5.1 ウィンドウ操作

```
[JWindow] JWindow& map( void );
[JWindow] JWindow& unmap( void );
```

- ウィンドウを画面上にマップ (表示) する (map)，もしくはアンマップ (非表示) する (unmap)。

```
[Xwin] void close( void );
```

- ウィンドウ全てを閉じる。**jgr 終了時にかならず実行してください。**

```
[JWindow] JWindow& clear( void );
[Graph]   Graph&   clear( void );
```

- ウィンドウ内を消去し，ウィンドウ上の全てのグラフのラストポイントを消去する。

```
[JWindow] JWindow& flush( void );
[Graph]   Graph&   flush( void );
[Xwin]    XController& flush( void );
```

- ウィンドウ用のメモリバッファをフラッシュ(メモリバッファ上のデータを画面上に出力) する。

```
[Xwin] XController& sync( void );
```

- ウィンドウをシンクする。

5.2 マウス操作

```
[JWindow,Graph,Xwin] int mousePress( void );
```

- Xwin に対して呼び出した場合はいずれかのウィンドウに対して，JWindow もしくは Graph に対して呼び出した場合はそのウィンドウに対してマウスクリックを検出したとき True を，それ以外のとき False を返す。
- JGR-2.x での関数名: `mouse_press;`

```
[JWindow,Graph] int mousePress( XEvent &ev );
```

- グラフのあるウィンドウへのマウス入力があると True を返す。また、取得したマウスイベント情報を変数 ev に代入する
- `XEvent` は Xlib に用意されている構造体です。
- JGR-2.x での関数名: `mouse_press;`

```
[Xwin,JWindow,Graph,NONE] double ev2button(XEvent &ev);
```

- mousePress() で取得したマウスイベント変数 ev を引数として、マウスのどのボタンがクリックされたかを返す。
- ev2button は Xwin, JWindow, Graph の メンバ関数として呼びだすことも, 単独の関数として呼び出すこともできます。

```
[Graph] double ev2x(XEvent &ev);
[Graph] double ev2y(XEvent &ev);
```

- マウスイベント変数 ev を引数として、マウスクリックがクリックされた座標値 x,y を返す。

```
[JWindow] JWindow&     mouseWait();
[Graph]   Graph&       mouseWait();
[Xwin]    XController& mouseWait();
```

- JWindow もしくは Graph に対して呼び出した時はそのウィンドウへ，Xwin に対して呼び出した時はいずれかのウィンドウへのマウス入力があるまで待つ。
- JGR-2.x での関数名: `mouse_wait();`

以下はマウス関数の使用例です。

```
JWindow win;
Graph g;
XEvent ev;

win.open(10,10)
   .graph(g.open(...));
...

// マウスがクリックされるまで待つ。win.mouseWait(ev); としても等価
g.mouseWait(ev);

//どのボタンがクリックかを表示。以下のように呼び出すこともできる。
//Xwin.ev2button(ev); win.ev2button(ev); g.ev2button(ev);
printf("押されたボタンは %d\n",ev2button(ev));

printf("クリック位置は (%lf,%lf)\n", g.ev2x(ev), g.ev2y(ev));
```

サンプルプログラム sample.cc も見てください。

```
  [JWindow] JWindow&     mousePosition(XEvent &ev);
[Graph]   Graph&       mousePosition(XEvent &ev);
```

- 現在のマウスカーソルの位置 (pixel) を取得する。JWindow でよびだしたときには，ウィンドウ上の位置が，Graph でよびだしたときには，グラフ上の位置を取得できる。

```
  [JWindow] JWindow&     hideCursor(int mode=True);
[Graph]   Graph&       hideCursor(int mode=True);
```

- マウスカーソルを消す。再表示するには引数に False を指定する。

```
  [JWindow] JWindow&     setCursor(unsigned int mouse_shape);
```

- マウスカーソルの形を指定する。
- 指定は，Xlib のフォントカーソルから指定する。例えば，https://tronche.com/gui/x/xlib/appendix/b/ 参照のこと。

```
[JWindow] JWindow&     unsetCursor(void);
```

- マウスカーソルの形をデフォルトにもどす。

5.3 座標軸関係の描画

```
[Graph] Graph&  setXScaleFull( int flag=True );
[Graph] Graph&  setYScaleFull( int flag=True );
[Graph] Graph&  setScaleFull( int flag=True );
```

- x 軸 (setXScaleFull), y 軸 (setYScaleFull), もしくは両者 (setScaleFull) の目盛刻みを高さ一杯にする
- jgr-2.x での関数名: set_xscale_full, set_yscale_full, set_scale_full

```
[Graph] Graph&  setAxisWithDigit( int flag=True );
```

- 座標軸には数値付き目盛をつける
- jgr-2.x での関数名: set_axis_with_digit

```
[Graph] Graph&  axis( void ); [JWindow] JWindow& axis( void );
```

- 座標軸を書く
- Jwindow に対して適用した場合はウィンドウ上の全てのグラフの座標軸を描く

```
[Graph] Graph&  axisBox( void );
```

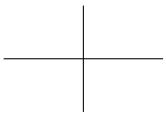
- 枠で囲んだ座標軸を書く
- jgr-2.x での関数名: axis_box(void);

```
[Graph] Graph&  xaxis( double blank );
[Graph] Graph&  yaxis( double blank );
[Graph] Graph&  xaxis( void );
[Graph] Graph&  yaxis( void );
```

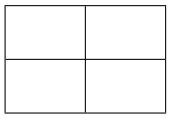
- xaxis() は x 座標軸を，yaxis() は y 座標軸を書く
- 間隔 blank で座標軸の刻みを書く
- 間隔 blank を指定しなければ間隔は自動で決まる

もし、グラフ grp に関して

```
grp.axis();
```

とすると、  という感じの座標軸が書かれます。また、

```
grp.axisBox();
```

とすると、  となります。

X 軸だけ、もしくは Y 軸だけ書きたいときには、

```
grp.xaxis( 1.0 ); //目盛の刻み幅を 1.0 にして、
grp.yaxis( 0.1 ); //目盛の刻み幅を 0.1 にして、Y 軸を書く
grp.xaxis();      // X 軸を書く
grp.yaxis();      // Y 軸を書く
```

などとします。目盛の刻み幅の指定をしていないときには、目盛は自動設定になります。目盛をグラフ一杯の長さにしたい時には、setScaleFull() や setXScaleFull()、setYSCaleFull() を使って、

```
grp.setXScaleFull()
   .axis();
```

などとします。
目盛に数値を表示したいときには、

```
grp.setAxisWithDigit()
   .axis();
```

と、setAxisWithDigit() を使います。

5.4 グラフ・図形描画関数

5.4.1 直線を描く

```
[Graph] Graph&  line(double x, double y, double x2, double y2);
[Graph] Graph&  dashLine( double x, double y, double x2, double y2 );
```

- (x,y) と (x2,y2) を直線もしくは点線で結ぶ
- Vine-2.x での関数名: dash_line();

```
[Graph] Graph&  line(double x, double y);
[Graph] Graph&  dashLine( double x2, double y2 );
```

- ラストポイントと (x,y) を直線もしくは点線で結ぶ
- ラストポイントが設定されて無い時は (x,y) をラストポイントとする
- Vine-2.x での関数名: dash_line();

各グラフで最後に描いた点はラストポイントとして記憶されています。折れ線グラフを描く時にはこれを利用すると楽です。例えば (1,1), (2,1), (3,4) を結ぶ線を描く方法です。

```
Graph g;
....
g.line(1,1,2,1).line(2,1,3,4); // ラストポイントを使わない場合
g.line(1,1).(2,1).(3,4);       // ラストポイントを使った場合
```

$y = x^2$ のグラフは以下のように描くことができます。

```
for(x=-10;x<=10;x++) g.line(x,x*x);
```

ラストポイントを消去したい時には以下の関数を使います。

```
[Graph] Graph& forgetLastPoint(void);
[Graph] Graph& forget(void);
```

- ラストポイントの消去 (forget は fotgetLastPoint の別名)

直線の属性を変更したい時には以下の関数を使います。

```
[Graph] Graph& setLineSolid( int width=0 );
```

- 直線の属性を実線に変更
- 引数で線の太さを指定できる (指定しなかったら，一番細い線になる)
- jgr-2.x での関数名: `set_solid_line();`

```
[Graph] Graph& setLineDash( int width=0 );
[Graph] Graph& setLineDDash( int width=0 );
```

- 直線の属性を点線もしくは二点鎖線に変更
- 引数で線の太さを指定できる (指定しなかったら，一番細い線になる)
- jgr-2.x での関数名: `set_dash_line();`, `set_double_dash_line();`

5.4.2 その他の図形

```
[Graph] Graph&  pset(double x, double y);
```

- (x,y) に点をうつ

```
[Graph] Graph&  rectangle( double x, double y, double width, double height );
[Graph] Graph&  fillRectangle( double x, double y, double width, double height );
```

- 左上が (x,y) になる位置に (width,height) の四角形を描く
- `fillRectangle` の場合は四角形の中を塗りつぶす
- Vine-2.x での関数名: `fill_rectangle`

```
[Graph] Graph&  cross( double x, double y, double r );
```

- 中心が (x,y) のクロスを描く。
- クロスの長さが r(x 軸方向の長さで指定)

```
[Graph] Graph&  circle( double x, double y, double r );
[Graph] Graph&  fillCircle( double x, double y, double r );
```

- 中心が (x,y)、半径 r(x 軸方向の長さで指定) の円を描く
- `filCircle` の場合は円の中を塗りつぶす
- Vine-2.x での関数名: `fill_circle`

```
[Graph] Graph&  poly( double x, double y );
[Graph] Graph&  closePoly( double x, double y );
[Graph] Graph&  fillPoly( double x, double y );
```

- 点 (x,y) を結ぶ多角形を描く
- 最後の点は fillPoly か closePoly で指定。fillPoly なら中はぬりつぶされる。

以下は poly を使って (0,0), (10,0), (5,5) を結んだ三角形を描く例です。

```
Graph g;
g.poly(0,0)
 .poly(10,0)
 .fillPoly(5,5); //塗りつぶさないなら closePoly を使う
```

5.5 色操作

```
[Graph] Graph&  setFGColor( char* clr="black" );
[Graph] Graph&  color( char* clr="black" );
```

- 前景色指定 (グラフ等の図形の絵の色)
- color は setFGcolor の別名です。
- 指定色は以下から選べます。

  ”black”, ”blue”, ”red”, ”yellow”, ”green”, ”cyan”,”magenta”, ”brown”, ”lightgray”, ”darkgray”, ”lightblue”, ”lightgreen”, ”lightcyan”, ”lightred”, ”lightmagenta”

- 例) `g.color("red")`

```
[Graph] Graph&  setAxisColor( char* clr="black" );
```

- 座標軸の色指定 (デフォルトは黒)

```
[Graph] Graph&  pushColor( char* clr="black" );
[Graph] Graph&  popColor();
```

- 前景色を変更し、古い色は保存しておく (pushColor)。
- popColor は前景色を元の色に戻す

```
[JWindow] JWindow& setBgColor(char *bgclr);
```

- ウィンドウの背景色を変更する。

色の設定例はサンプルプログラム `sin.cc` などを見てください。

5.6 フォント設定とテキスト操作

以下はフォント設定のための関数です。

```
[JWindow] JWindow&  setTextFont( int ftype, int fstyle, int size );
[Graph]   Graph&    setTextFont( int ftype, int fstyle, int size );
[JWindow] JWindow&  setScaleFont( int ftype, int fstyle, int size );
[Graph]   Graph&    setScaleFont( int ftype, int fstyle, int size );
```

- 関数 text で表示する文字フォントの設定 (`setTextFont`)，および座標軸ラベルのフォント設定 (`setScaleFont`)
- `ftype` はフォントの種類。以下の値を設定出来る。

  Courier, Helvetica, Times, Symbol

- `fstyle` はフォントのスタイル。以下の値を設定出来る。

  Roman, Bold, Italic, Oblique

- `size` はフォントの大きさ。以下の値を設定出来る。

  8,10,11,12,14,17,18,20,24,25,34

- JWindow に対して呼び出した場合には，ウィンドウ登録されるグラフのテキストフォントのデフォルト値を設定出来ます。この場合上記の関数はウィンドウの map より前に実行してください。設定値はウィンドウの map 時に各グラフのフォントに設定されます。
- Graph クラスに対して呼び出した場合は, 個々のグラフのフォントを個別に変更します。

以下はテキスト表示の関数です。

```
[Graph] Graph& text( double x, double y, char *msg, int position=Center);
```

- 文字列 msg を位置 (x,y) に書く。
- 位置 (x,y) は文字の中心位置がデフォルト。引数 position で指定位置を以下から選ぶことができる。

  Left, Center, Right, LeftTop, CenterTop, RightTop, RightCenter

以下はテキストの表示例です。

```
Graph g;
...
g.text(50,0,"Hello!");           //"Hello"と表示．文字列の中心位置が (50,0)
g.setTextFont(Courier,Roman,14) //フォントをサイズ 14 の Courier,Roman 体に
 .text(50,20,"Hello!",RightTop);//"Hello"と表示．文字列の右上が (50,20)
```

setTextFont で指定したフォントがシステム上に無い時には，利用出来るものが探索されます。サンプルプログラム font.cc のソースや実行例等も参考にしてください。

5.7 EPS ファイルの出力

```
[Graph] Graph& EPSOn();
[Graph] Graph& EPSOff();
```

- EPS 出力のための情報を保存を開始 (EPSOn)， および停止 (EPSOff) する

```
[Graph] void resetEPS();
```

- EPS 出力のために保存されている情報を消去する。

```
[Graph] void EPSOut(char* epsfname,
                    int lmargin=LEFT_MARGIN, int bmargin=BOTTOM_MARGIN);
[Graph] void EPSAppend(ofstream& epsf,
                    int lmargin=LEFT_MARGIN, int bmargin=BOTTOM_MARGIN);
```

- 保存されているデータを EPS 形式で新規ファイルに書き込む (EPSOut)，もしくは既存ファイルに追加 (EPSAppend) する。
- 引数 epsfname (EPSOut) は出力ファイル名
- 引数 epsf (EPSAppen) は出力ファイルストリーム
- 必要に応じて左マージン (lmargin)， 下部マージン (bmargin) を dot 数で指定する。指定しなければデフォルト値として 25 がマージンとなる。

EPS ファイルの出力方法はサンプルプログラム sin.cc を参考にしてください。

5.8 発展的な操作のための関数

```
[JWindow] JWindow&     expose();
[Xwin]    XController& expose( void );
```

- XWin に対して用いた場合は，どれかのウィンドウが expose された時，JWindow に対して用いた場合は，そのウィンドウが expose された時に True を返す

```
[JWindow] int getIndex( void );
```

- ウィンドウが jgr で管理されている何番目のウィンドウであるか (インデックス) を返す
- JGR-2.x での関数名: `get_index();`

```
[JWindow] Window getWindow( void );
```

- ウィンドウの情報を Xlib の Window 構造体で返す
- JGR-2.x での関数名: `get_window();`

6 メニューについて

メニュー関連は上原さん (現東レ所属) の作成をしたものをもとに、手を加えたものです。感謝です。(注：以下の文書はとても古いもので、現状と一致しません。。。現在準備中です。)

6.1 メニューウィンドウ (Menu Class) について

class Menu のオブジェクトは、メニューの要素（アイテム) としてユーザー定義の処理関数とタイトルの組を受けとり、メニューウィンドウにタイトルを表示し、アイテムのタイトル部分がクリックされると該当する処理関数を呼びます。

メニューウィンドウはウィンドウマネージャーの支配を受けます。

Menu は、そのオブジェクトを宣言して直接メッセージを送ることにより使用します。

Menu 自身はその内部にイベント待ちループを持っていません。

タイトルは最大 32 文字で日本語は扱いません。処理関数としては `void (*func)(Menu&)` 型の関数をとります。アイテムの数に制限はありません。アイテムの数を登録する必要もありません。

ユーザーに公開するメソッドは以下の通りです。

・`Menu& open(int x=0,int y=0);`
　　働き　：メニューウィンドウを開く。最初に宣言。
　　引数　：メニューウィンドウの ルートでの位置 `x,y`
　　戻り値：当該オブジェクトの参照

・`Menu& addItem(const char* title, void (*handler)(Menu&));`
　　働き　：メニューにアイテムを登録する。
　　引数　：アイテムのタイトルの文字列へのポインタ `title`,
　　　　処理関数へのポインタ `handler`
　　戻り値：当該オブジェクトの参照

・`Menu& addSubmenu(const char* title, Menu& submenu, int type=Temporal));`
　　働き　：メニューにサブメニューを登録する。
　　引数　：アイテムのタイトルの文字列へのポインタ `title`,

サブメニューへのポインタ submenu,
サブメニューのタイプ type (Temporal もしくは Static)
戻り値：当該オブジェクトの参照

・Menu& addClose(void);
働き ：メニューをクローズするためのメニュー項目を追加する。
引数 ：なし
戻り値：当該オブジェクトの参照

・Menu& addExit(void);
働き ：プログラムを終了するためのメニュー項目を追加する。
引数 ：なし
戻り値：当該オブジェクトの参照

・Menu& setTitle(const char* str);
働き ：メニュー自体のタイトルを登録する。登録しなければタイトルは 'Menu' となる。
引数 ：タイトルの文字列へのポインタ str,
戻り値：当該オブジェクトの参照

・Menu& setType(int type=Static);
働き ：メニューのタイプを決める
引数 ：メニュー項目を選んでもメニューを閉じないとき Static、
メニュー項目のクリックがあったときメニューを閉じるなら
Temporal
戻り値：当該オブジェクトの参照

・Menu& setStyle(int style=Separate)
働き ：メニューの概観を決める
引数 ：メニュー項目の間にしきりをいれるなら Separate, いれな
いなら Flat を指定する。
戻り値：当該オブジェクトの参照

・Menu& map(); map(int x,int y);
働き ：メニューウィンドウをマップする。
open() 以降で使用する。
引数 ：メニューウィンドウの ルートでの位置 x,y
あるいは、なし
（以前にマップした位置にマップする）
戻り値：当該オブジェクトの参照

・Menu& unmap();
働き ：メニューウィンドウをアンマップする。
open() 以降で使用する。
引数 ：なし
戻り値：当該オブジェクトの参照

・Menu& act();
働き ：マウスクリックをチェックし、クリックがあれば処理関数を呼ぶ。
メインループ中で使用。
引数 ：なし
戻り値：当該オブジェクトの参照

・Menu& close();
働き ：メニューウィンドウを削除する。とくに明示的に呼ぶ必要はない。
open() を呼べば再びメニューウィンドウは使える。
引数 ：なし
戻り値：当該オブジェクトの参照

使い方の例は次節を参照して下さい。

6.2 入力ウィンドウについて

以下の関数を呼ぶことによって、入力ウィンドウをメニュー項目として登録し、ユーザーから文字列あるいは数値を取得することができます。

・Menu& addGetString(char* value, char* title, char* dflt);
働き ：文字変数 value に文字を入力する窓をメニュー項目としてつくる。
引数 ：文字変数 value, value のデフォルト値 dflt, メッセージ title
戻り値：当該オブジェクトの参照

・Menu& addGetValue(double& value, char* title, double dflt);
働き ：変数 value に数値を入力する窓をメニュー項目としてつくる。
引数 ：変数 value, value のデフォルト値 dflt, メッセージ title
戻り値：当該オブジェクトの参照

入力ウィンドウは、文字キー以外に、RET、BS、DEL、ESC、を受け付けます。ESC で入力ウィンドウをクリアします。マウスのボタンプレスは RET と同じ働きをします。
各文字列の最大長は Menu の MAX_STR_LEN と同じです。

6.3 簡単な例

以下に使い方の例を示します。

```
//
// メニューウィンドウ、入力ウィンドウの使い方の例
//
//    3つのアイテムを持つメインメニューを開く。
//    1つめのアイテムで入力を受け付けるための一時的なサブメニューを開く
//      サブメニューの1つめのアイテムで、文字列を受け付ける。
//      サブメニューの2つめのアイテムで、数値を受け付ける。
//    2つめのアイテムで変数を標準出力に表示するための静的なサブメニューを開く
//      サブメニューの1つめのアイテムで、文字列を標準出力に出す。
```

```
//      サブメニューの２つめのアイテムで、数値を標準出力に出す。
//    ３つめのアイテムで終了する。
//
//
#include <iostream.h>
#include <stdlib.h>
#include <jkit/jgr.h>

Menu main_menu(Horizontal),
        sub_menuA(Vertical,Static),
        sub_menuB(Vertical,Static);

char str[MAX_STR_LEN+1];
//char str[256];
double v;

////////////////////////// サブメニューのためのハンドラ ////////////////////

void handler_subB1(void* dummy){          //文字列を表示する
  cout << "str= " << str << "\n";
}
void handler_subB2(void* dummy){          //数値を表示する
  cout << "v= " << v << "\n";
}

////////////////////// メインルーチン ///////////////////////

int main(){
  sub_menuA.open()
          .addGetString(str,"Input string","aaaa")
          .addGetValue(v,"Input value",10.0)
          .addClose();

  sub_menuB.open()
          .addItem("string",handler_subB1)
          .addItem("value", handler_subB2)
          .addClose();

  main_menu.open(100,50)
          .addSubmenu("input",sub_menuA)
          .addSubmenu("print",sub_menuB)
          .addExit()
          .map();

  while(1) {main_menu.act();}                    // メインメニューを走らせる
}
```

このプログラム例は、sample ディレクトリに menu.cc という名前で置いてあります。

6.4 ウィンドウマネージャーに関する注意

メニューウィンドウと入力ウィンドウはウィンドウマネージャーの支配を受けます。これらが開かれる際、アプリケーションが指定するウィンドウ初期位置をウィンドウマネージャーに使用させるために、ウィンドウマネージャーの startup file で、そのことを指定しておかなければなりません。例えば、.twmrc で

```
UsePPosition "on"
```

という行を加えなければなりません。あるいは、

```
UsePPosition "non-zero"
```

としておけば、ウィンドウの位置を (0,0) と指定すると、open されてもすぐにはマップされず、マウスによって初期位置を決めることができます。

7 問題点

バグもあるかもしれませんが、あったら教えて下さい。

索 引