ＣＲＴ画面帳票作成ツール

SimpleMaster

テストプログラム　説明書

株式会社 FUJI フジテクニカ

目次

ページ

1　メニュー画面　---------- 1
2　テスト画面(1) 図形　---------- 2
3　テスト画面(2) ボタン･文字列　---------- 3
4　テスト画面(3) キー入力　---------- 4
5　テスト画面(4) グラフ　---------- 9
6　テスト画面(5) マウス操作　---------- 11
7　メンテナンス画面　---------- 13

Windows 版 と Linux 版の相違点

レイアウト作成プログラム SimpleMaster ではダイアログボックスの表示形式などに相違点がありますが、
設定する項目、内容は同一です。
作成したアプリケーションプログラムの実行内容は同一です。
その他、明確な相違点がある場合には説明書に記述しています。

改訂記録

2004/01/15　V1.01　新規作成
2004/06/25　V1.11　スプレッドシート機能追加 ／ Linux 対応

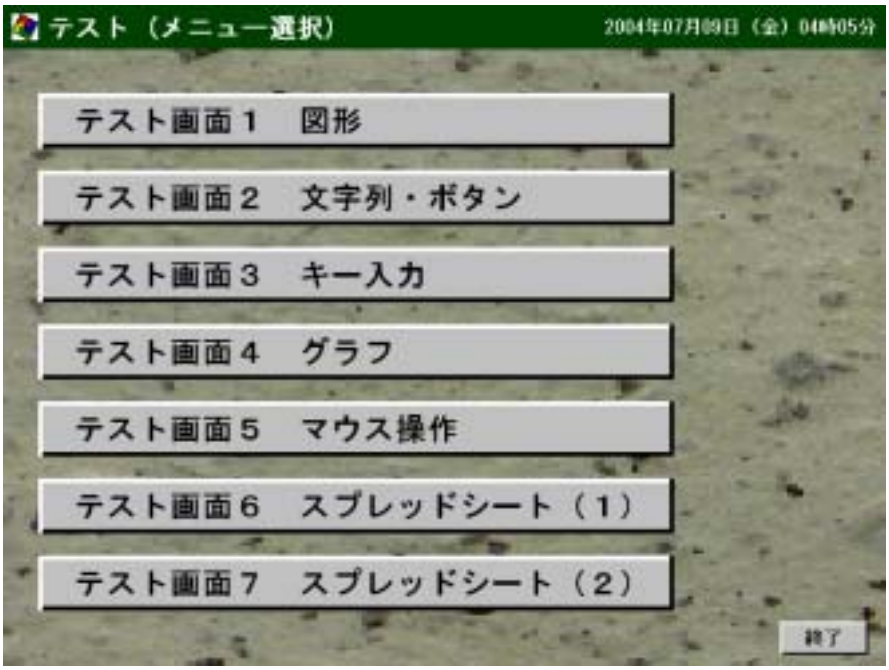
テストプログラムは SimpleMaster の機能確認のために使用しています。
ソースプログラムを添付しましたのでアプリケーションプログラムの開発に参照してください。

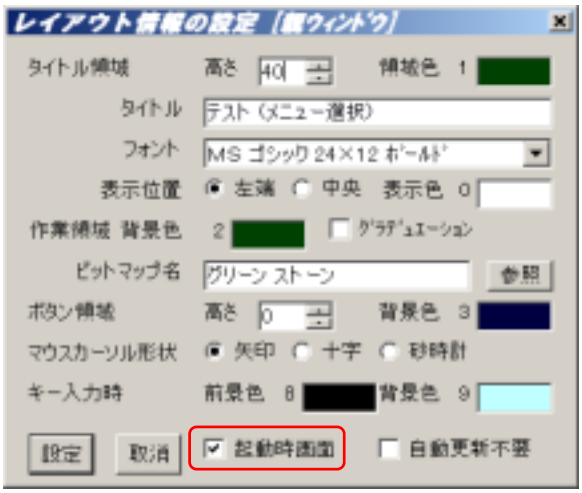
1　メニュー画面

この画面は起動時の画面に設定されています。

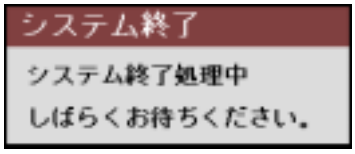
背景にビットマップを表示しています。

テスト画面6/7 スプレッドシートは　取り扱い説明書 を参照してください。





［終了］ボタンを押すと終了時画面が表示されます。



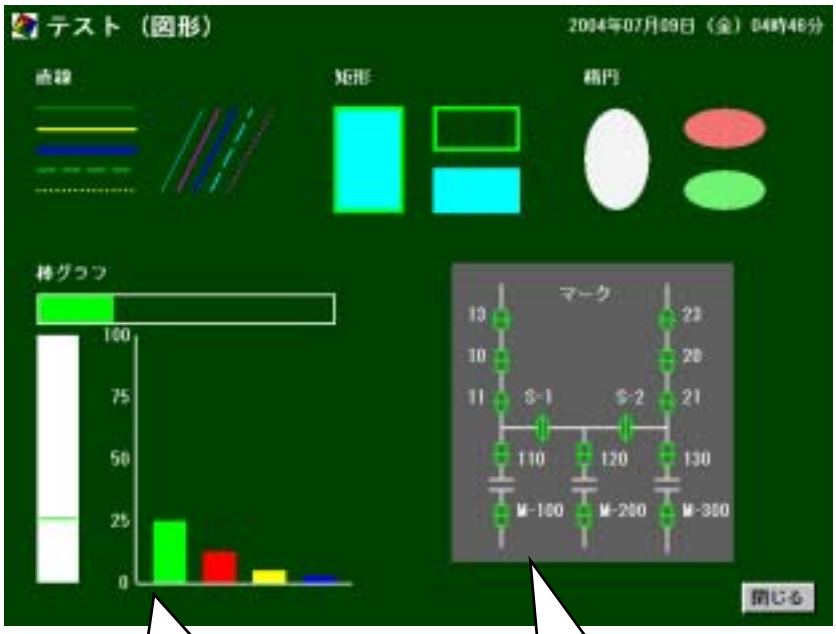
2　テスト画面 (1)　　図形

ウィンドウ処理クラス名　　CPtrnTest1　(TestDisp.cpp)
オーバライドされる仮想関数　　PtrnInitFunc, PtrnTimrFunc, PtrnMesgFunc

直線 / 矩形
5秒周期で表示色を更新しています。
(TestMain.cpp / TestThread)

棒グラフ
1秒周期で数値データを 0～99 の範囲で更新しています。
(TestMain.cpp / TestThread)

マーク
マウス左ボタンのクリックで反転、点滅表示を開始します。
(PtrnMesgFunc)
5秒経過後、通常表示に戻します。 (PtrnTimrFunc)



上下限値を 0～100, 0～200, 0～500, 0～1000 に設定することでスケールを変化させています。

マウス左ボタンでマークをクリックすると反転、点滅 (5秒間) します。

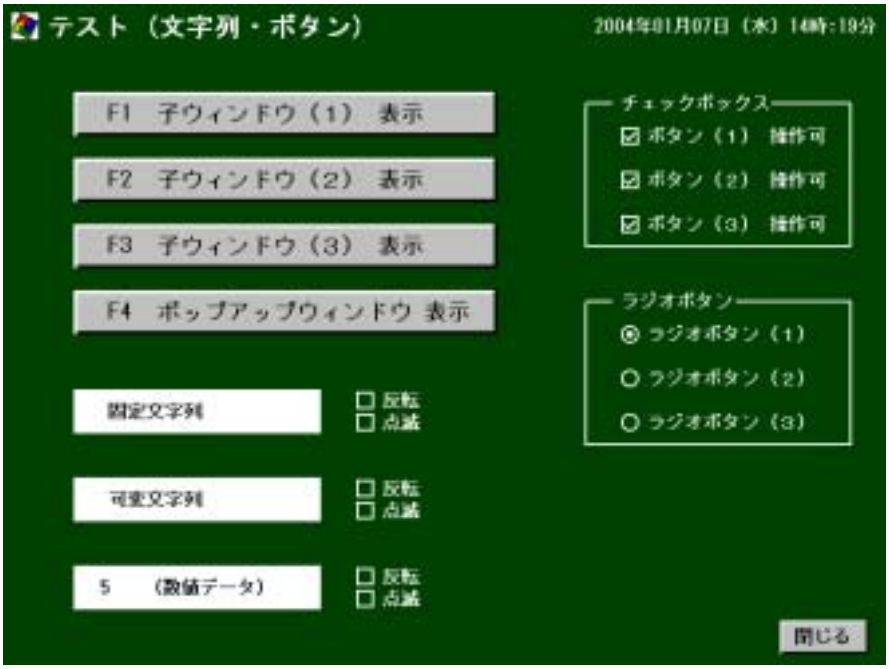
楕円
50msec 周期で表示色を更新しています。
(TestMain.cpp / TestThread)

直線/矩形ではレイアウト作成時にあらかじめ設定されている4系統の表示色を切り替えています。

楕円では表示色番号は固定で表示色そのものを入れ替えています。
右図 の例では表示色番号60に SetColorData 関数を使用して表示色を設定しています。

3　テスト画面 (2)　　ボタン･文字列

ウィンドウ処理クラス名　　CPtrnTest2　(TestDisp.cpp)
オーバライドされる仮想関数　　PtrnInitFunc, PtrnMesgFunc



子ウィンドウ表示ボタン
　ファンクションキー F1/F2/F3 で開くことができます。

ポップアップウィンドウ表示ボタン
　ファンクションキー F4 で開くことができます。
　ボタンを操作するとボタンの右側にポップアップウィンドウが表示されます。

チェックボックス
　OFF にすると左側の子ウィンドウを開くボタンが操作不可 (灰色表示) となります。

固定文字列 ／ 可変文字列 ／ 数値データ
　それぞれチェックボックスの操作により、反転、点滅表示を行うことができます。
　“可変文字列”は文字列データファイル (MesgData.xls) で設定しています。

4　テスト画面(3)　　キー入力

ウィンドウ処理クラス名　　CHaisenhyo　(haisenhyo.cpp)
オーバライドされる仮想関数　　PtrnInitFunc, PtrnExitFunc, PtrnMesgFunc

4－1　　画面構成

テスト画面6／7 ｽﾌﾟﾚｯﾄﾞｼｰﾄの表示に使用しています 配線表データ を1レコード単位で表示し、キーボード入力処理を行います。

キーボード入力が可能な可変文字列/数値データ、およびその背景となる矩形枠には同一の識別コードが設定されています。
文字列以外の矩形領域をマウス左ボタンでｸﾘｯｸすると、自動的に同一識別コードのキーボード入力可能な可変文字列 または 数値データ へのキーボード入力が開始されます。

4－2　プログラムの説明

表示開始時処理

```
void  CHaisenhyo :: PtrnInitFunc ()                    // ウィンドウの初期化
{
①  InitHaisenhyo();                                    // 配線表初期化

②  DispDataSet(1);                                     // 先頭ページ設定
③  KeyInputStart(SYSTEMNO);                            // キーボード入力開始
}
```

① 配線表ファイルを読込みます。
② 1レコード分の配線表データを共有データ領域へ設定します。
引数には行番号（レコード番号）を指定します。
③ キーボード入力を開始します、
引数にはキーボード入力を行う項目の識別コードを指定します。

表示終了時処理

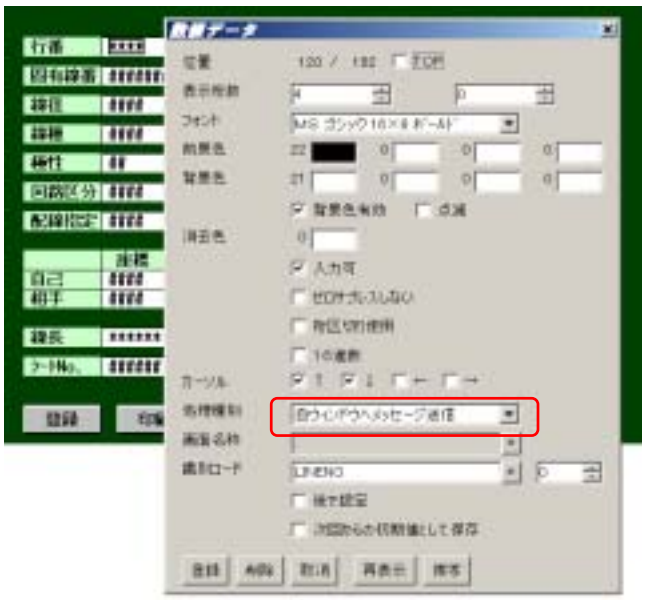
```
void  CHaisenhyo :: PtrnExitFunc ()                    // ウィンドウの終了処理
{
     if (HaisenData != NULL) delete [] HaisenData;     // 作業領域破棄
     HaisenData = NULL;
}
```

配線表読込み用に確保した領域を開放します。

マウス操作時処理

```
void  CHaisenhyo :: PtrnMesgFunc (int code)            // マウス操作による通知
{
     if (code == LINENO) {                             // 行番
          int line = Get_iData(LINENO);
          if ((line >= 1) && (line <= HaisenHedr.datacount)) {
          DispDataGet();
          DispDataSet(line);
     }
```

行番号へのキーボード入力、
または スピンボタンの操作が
行われた時のメッセージを受け取り、
現在表示中のデータを格納後
（キーボード入力が行われている
場合のため）、行番号に相当する
配線表の1レコードを表示のため
共有データ領域へ設定します。



行番号入力欄は他の項目と異なり
処理種別コードに 自ウィンドウへ
メッセージ送信 が設定されてい
ます。 スピンボタンの操作と区別
することなく行番号の変化を検出
することができます、

```
else if (code == TOUROKU) {                          // 登録 ボタン
          DispDataGet();
          CFileWrite  HaisenHyo(ProjPath, "Haisenhyo");     // 配線表ファイル書込み
          HaisenHyo.WriteBlock(&HaisenHedr, sizeof(HAISENHEDR));
          for (int i=0; i<HaisenHedr.datacount; ++i) {
               strcpy(HaisenData[i].kyoku, "0");
               HaisenHyo.WriteBlock(&HaisenData[i], sizeof(HAISENDATA));
          }
     }
```

登録 ボタン が押された時、配線表データをディスクに書込みます。

```
     else if (code == INSATSU) {                     // 印刷 ボタン
          DispDataGet();
          PtrnSetCursor(2);                          // カーソル：砂時計
          PrintHaisenhyo();
          PtrnSetCursor(0);                          // カーソル：矢印
     }
```

印刷 ボタンを押した時の処理を記述します。
印刷処理の詳細は 4－3 配線表の印刷 を参照してください。

```
else if (code == HENKAN ) {                          // 変換 ボタン
          DispDataGet();
          PtrnSetCursor(2);                          // カーソル：砂時計
          ConvertHaisenhyo();
          PtrnSetCursor(0);                          // カーソル：矢印
          InitHaisenhyo();                           // 配線表再読込み
          DispDataSet(1);                            // 先頭ページ設定
     }
}
```

変換 ボタン を押された時の処理を記述します。
変換処理の詳細は 4－4 配線表の変換 を参照してください。

4－3　配線表の印刷　(関数 PrintHaisenhyo)

親ウィンドウで 印刷 ボタンを押された時、配線表の印刷を行います。

```
void  CPtrnTest3 :: PrintHaisenhyo ()                                    // 配線表印刷
{
①   CPrintDraw  DC(this, "HAISENHYO");                                   // プリンターデバイスの作成
②   CSize       PrntSize = DC.PrintGetSize();                            // 印字領域の大きさ

③   CPtrnData   PtrnData;                                                // 印字情報の読込み
    if (PrntSize.cx >= PrntSize.cy) PtrnData.LoadPtrnData("帳票 (横)");  // 横書き
    else                            PtrnData.LoadPtrnData("帳票 (縦)");  // 縦書き

④   int  Line = (PrntSize.cx >= PrntSize.cy) ? PRINT_LINE1 : PRINT_LINE2; // 印字行数
    int  Page = 1 + (HaisenHedr.datacount-1) / Line;                     // 総ページ数
⑤   Page = 1;                                                            // (For Test)

⑥   if (DC.PrintOpen("配線表", PtrnData.PtrnHedr.Parm1[1],               // 印字開始処理
                           PtrnData.PtrnHedr.Parm1[2]) == TRUE) {
        for (int page=0; page<Page; ++page) {
            Set_sData(P_SYSTEMNO,    HaisenHedr.systemno);               // システムＮｏ
            Set_sData(P_NOUNYUUSAKI, HaisenHedr.nousaki );               // 納入先

⑦           （途中省略）


            }
⑧           PtrnData.DispPtrnData(&DC);                                  // 印刷

⑨           if (Page == (page+1)) break;                                 // 最終ページ
            else DC.PrintNewPage();                                      // 改ページ
        }

⑩       DC.PrintClose();                                                 // 印刷終了
    }
}
```

① 印字用デバイスコンテキストを作成します。
環境設定ファイル (Test.ini)、または ［プリンター設定］ダイアログボックスで印字情報を設定します。
② 印字に使用する用紙の大きさを取得します。
③ レイアウト情報を処理するクラスを生成し、横書き または 縦書き用のレイアウト情報を読込みます。
④ 1ページ当たりの印字行数と配線表のレコード数より印字を行うページ数を計算します。
⑤ 印字するページ数を 1ページ のみに限定します。 (テスト用なので)
⑥ 印刷処理を開始します。 引数でドキュメント名称とレイアウトの大きさを指定します。
印字用デバイスコンテキストでは用紙の大きさとレイアウトの大きさにより拡大／縮小印字を行います。
⑦ 1ページ分の印字データを共有データ領域へ設定します。
⑧ 1ページ分の印字を行います。
⑨ 最終ページでなければ 改ページ 処理を行い印字を継続します。
⑩ 最終ページの場合印刷処理を終了します。

4－4　配線表の変換　（関数 ConvertHaisenhyo）

親ウィンドウで 変換 ボタンを押された時、Microsoft Excel 形式のファイルを読込み配線表を作成します。

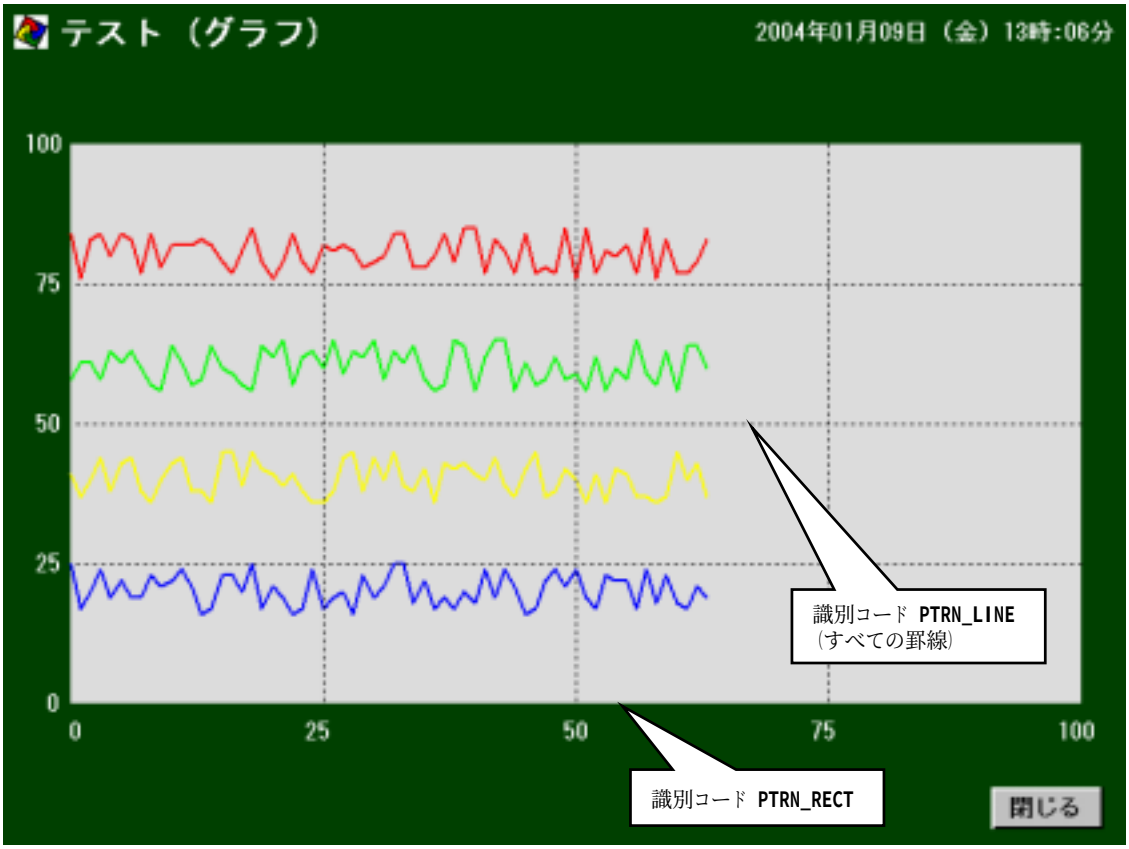
```
static  HAISENHEDR  Hedr;                                          // 配線表 （ヘッダー部）
static  HAISENDATA  Data[500];                                     // 配線表 （データ部）

void  CPtrnTest3 :: ConvertHaisenhyo ()                            // 配線表変換
{
     ::ZeroMemory(&Hedr, sizeof(Hedr));
     ::ZeroMemory( Data, sizeof(Data));

①        if (DdeExcelInit(0) == 0) {                               // ＤＤＥ接続の開始
②        if (DdeExcelOpen (ProjPath, "Haisenhyo.XLS") == TRUE) {   // Excel ﾌｧｲﾙのｵｰﾌﾟﾝ
③             DdeExcelSheet("Sheet1");

④             DdeExcelReadAsc(1, 2,  Hedr.systemno);               // ヘッダー部
              DdeExcelReadAsc(2, 2,  Hedr.nousaki );
              DdeExcelReadAsc(3, 2,  Hedr.sakuban );
              DdeExcelReadAsc(4, 2,  Hedr.banno   );
              DdeExcelReadAsc(5, 2,  Hedr.comment );

              for (int i=0;; ++i) {                                // データ部
                   Data[i].gyoban = DdeExcelReadInt(i+9,  1);
                   Data[i].length = DdeExcelReadInt(i+9, 16);

                   DdeExcelReadAsc(i+9,  2,  Data[i].senban);
                   DdeExcelReadAsc(i+9,  3,  Data[i].senkei);
                   DdeExcelReadAsc(i+9,  4,  Data[i].sensyu);

                    （途中省略）

                   DdeExcelReadAsc(i+9, 17,  Data[i].sheet );
                   if (Data[i].gyoban == 0) break;                 // 最終行
              }
              Hedr.datacount = i;                                  // データ数
⑤             DdeExcelClose(0);                                    // Excel ﾌｧｲﾙｸﾛｰｽﾞ

⑥             CFileWrite  HaisenHyo(ProjPath, "Haisenhyo");        // ファイル書込み
              HaisenHyo.WriteBlock(&Hedr, sizeof(HAISENHEDR));
              for (i=0; i<Hedr.datacount; ++i)
                   HaisenHyo.WriteBlock(&Data[i], sizeof(HAISENDATA));
         }
⑦        DdeExcelExit();                                           // ＤＤＥ切断
     }
}
```

① Microsoft Excel をアイコンの状態で起動し、DDE 接続を開始します。
② Microsoft Excel に ”Haisenhyo.xls” を読込むように指示します。
③ Microsoft Excel に ”Sheet1” を選択するように指示します。
④ 行／桁番号を指定して Excel のセルに設定されたデータを読込みます。
⑤ Microsoft Excel に ”Haisenhyo.xls” を保存せずに閉じるように指示します。
⑥ 読込んだデータをディスクファイル Haisenhyo に格納します。
⑦ Microsoft Excel を終了し、DDE 接続を切断します。

関数 DdeExcelxxx は Microsoft Excel 形式のファイルを DDE 接続により読書きすることができる関数で、スタティックライブラリ SimplePack に含まれています。

5　テスト画面 (4)　　グラフ

ウィンドウ処理クラス名　　CPtrnTest4　(TestDisp.cpp)
オーバライドされる仮想関数　　PtrnInitFunc, PtrnDispFunc



SimpleMaster で準備された描画機能では作成することができない画面の表示例です。

```
void  CPtrnTest4 :: PtrnInitFunc ()                          // ウィンドウの初期化
{
     PTRNDATA  Ptrn;                                         // 矩形領域表示情報
     PtrnReadData(GRAPH_RECT, PTRN_RECT, &Ptrn);
     rect = Ptrn.Rect;
     timr = cont = 0;
}
```

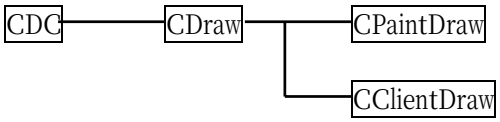
画面表示開始時に、グラフを描画する矩形領域の座標を取込み、保存します。

仮想関数 PtrnDispFunc は CRT 画面初期表示 (WM_PAINT メッセージ)、および 100msec タイマー処理時 (WM_TIMER メッセージ) で呼ばれます。

```
void  CPtrnTest4 :: PtrnDispFunc (CDraw* pDraw, int mode)          // 画面再表示
{
①   if ((mode != 0) && (++timr >= 5)) {                           // 500msec 周期
        timr = 0;
②       PtrnDispData(pDraw, GRAPH_RECT, PTRN_RECT);              // 矩形塗り潰し

③       data1[cont] = 15 + (rand() % 10);
        data2[cont] = 35 + (rand() % 10);
        data3[cont] = 55 + (rand() % 10);
        data4[cont] = 75 + (rand() % 10);
        if (++cont == 100) cont = 0;
    }

    if ((mode == 0) || (timr == 0)) {
        CPoint  point[100];                                      // グラフ計算
④       for (int i=0; i<4; ++i) {
            for (int j=0; j<cont; ++j) {
                int  data;
                if      (i == 0) data = data1[j];
                else if (i == 1) data = data2[j];
                else if (i == 2) data = data3[j];
                else             data = data4[j];
                point[j].x= rect.left + j    * (rect.right  - rect.left) / 100;
                point[j].y= rect.top  + data * (rect.bottom - rect.top ) / 100;
            }

⑤           CPen  pen1(PS_SOLID, 2, GetColorData(50+i));         // グラフ描画
            CPen* pen2 = pDraw->SelectObject(&pen1);
            pDraw->SetBkMode(TRANSPARENT);
            pDraw->Polyline(point, cont);
            pDraw->SelectObject(pen2);
        }

⑥       PtrnDispData(pDraw, GRAPH_LINE, PTRN_LINE);              // 罫線再表示
    }
}
```

Linux 版は下記参照

① CRT 画面初期表示 (mode = 0)、または 500msec 周期で表示を行います。
② グラフを表示する矩形領域を描画します。(塗りつぶし)
③ 乱数を使用して描画データを作成します。
④ 描画を行う矩形領域の座標に合わせて折れ線グラフの各点の座標を計算します。
⑤ 描画処理を行います。 (Linux 版は下記参照)
SlctObject, SetBkMode, PolyLine は ＭＦＣのデバイスコンテキストクラス ＣＤＣ のメンバ関数です。 SimplePack で定義している描画処理クラスは CDC の派生クラスとして作成しています。



仮想関数 PtrnDispFunc では、描画処理クラス CDraw、デバイスコンテキストクラス CDC のメンバ関数をすべて使用することができます。
⑥ 罫線を再表示します。(② 矩形領域の描画で塗りつぶされています。)

Linux 版では 共有ライブラリ GDK の関数を利用して描画を行います。

```
gdk_gc_set_foreground(pDraw->gc, color);
gdk_gc_set_line_attributes(pDraw->gc, 1, GDK_LINE_SOLID, GDK_CAP_BUTT, GDK_JOIN_MITER);
gdk_draw_lines(pDraw->window, pDraw->gc, point, cont);
```

6　テスト画面(5)　　マウス操作

ウィンドウ処理クラス名　　CPtrnTest5　(TestDisp.cpp)
オーバライドされる仮想関数　　PtrnInitFunc, PtrnMesgFunc, PtrnRbtnFunc, PtrnMouseFunc

マウス操作をアプリケーションプログラム側で横取りした例です。

仮想関数 PtrnMesgFunc　　マウス左ボタン操作時処理
仮想関数 PtrnRbtnFunc　　マウス右ボタン操作時処理

```
void  CPtrnTest5 :: PtrnMesgFunc (int code)                    // マウス操作による通知
{                                                              // 左ボタン
①   PtrnCloseWnd(1);                                          // 子ウィンドウを閉じる
②   if ((code >= MOUSE_RECT1) && (code <= MOUSE_RECT1+4))
         EOR_wData(code, 0x0001);

④   else if (code == MOUSE_SELECT  ) PtrnDispChild(1, "子ウィンドウ （1）");
    else if (code == MOUSE_SELECT+1) PtrnDispChild(1, "子ウィンドウ （2）");
    else if (code == MOUSE_SELECT+2) PtrnDispChild(1, "子ウィンドウ （3）");
}


void  CPtrnTest5 :: PtrnRbtnFunc (int code)                    // マウス操作による通知
{                                                              // 右ボタン
①   PtrnCloseWnd(1);
③   if ((code >= MOUSE_RECT2) && (code <= MOUSE_RECT2+4))
         EOR_wData(code, 0x0001);
}
```

①　　ポップアップメニューを閉じます。
　　ポップアップメニューが表示されていない場合は無視されます。
②、③　マウス左／右ボタンに対応した矩領域（A / B）をクリックされた時、表示色を変更します。
④　　ポップアップメニュー画面からのメッセージを受け取り、子ウィンドウを開きます。

仮想関数 PtrnMouseFunc　マウス操作時処理

```
BOOL CPtrnTest5 :: PtrnMouseFunc (CPoint point, int butn, int key)              // マウス操作横取り
{
①   PTRNDATA  Ptrn;
    if (PtrnTestData(point, &Ptrn) == TRUE) PtrnSetCursor(1);
    else                                    PtrnSetCursor(0);

②   if ((PtrnTestData(point, &Ptrn) == FALSE) && (butn == 2)
                                              && (key  == 0)) {
        PtrnDispPopup(1, "ポップアップメニュー", point, S_FILE, S_LINE);
        return TRUE;
    }

③   if (PtrnTestData(point, MOUSE_BUTTON1, PTRN_RECT, &Ptrn) == TRUE) {
        if ((butn == 0) && (key == 0)) Set_Color(MOUSE_BUTTON1, 1);
        return TRUE;
    }
    else Set_Color(MOUSE_BUTTON1, 0);

④   if (PtrnTestData(point, MOUSE_BUTTON2, PTRN_RECT, &Ptrn) == TRUE) {
        if ((butn == 0) && (key == 1)) Set_Color(MOUSE_BUTTON2, 1);
        return TRUE;
    }
    else Set_Color(MOUSE_BUTTON2, 0);

⑤   if (PtrnTestData(point, MOUSE_BUTTON3, PTRN_RECT, &Ptrn) == TRUE) {
        if ((butn == 1) && (key == 1)) Set_Color(MOUSE_BUTTON3, 1);
        return TRUE;
    }
    else Set_Color(MOUSE_BUTTON3, 0);

⑥   if (PtrnTestData(point, MOUSE_BUTTON4, PTRN_RECT, &Ptrn) == TRUE) {
        if ((butn == 2) && (key == 2)) Set_Color(MOUSE_BUTTON4, 1);
        return TRUE;
    }
    else Set_Color(MOUSE_BUTTON4, 0);

    return FALSE;
}
```

① マウス座標位置に識別コードが設定された描画要素が存在するとき（関数 PtrnTestData の戻り値が TRUE）、カーソルの形状を 十字 にします。 存在しないときには 矢印 に戻します。

② マウス座標位置に識別コードが設定された描画要素が存在しない場合に、マウス右ボタンが押され (butn == 2)、Shift、Ctrl キーが押されていないとき、ポップアップウィンドウを表示します。
表示位置は、マウス座標を基準にして自動調整が行われます。

③ マウス座標が C の矩形領域上にあり、マウスボタン、Shift キーが押されていないとき、
表示色を反転します。 マウス座標が C の矩形領域外となると表示色を元に戻します。

④ マウス座標が D の矩形領域上にあり、Shift キーが押されているとき、
表示色を反転します。 マウス座標が D の矩形領域外となると表示色を元に戻します。

⑤ マウス座標が E の矩形領域上にあり、マウス左ボタン、Shift キーが押されているとき、
表示色を反転します。 マウス座標が E の矩形領域外となると表示色を元に戻します。

⑥ マウス座標が F の矩形領域上にあり、マウス右ボタン、Ctrl キーが押されているとき、
表示色を反転します。 マウス座標が F の矩形領域外となると表示色を元に戻します。

7　メンテナンス画面

CRT 画面左上のロゴマークをマウス右ボタンでクリックするとメンテナンス画面を開くことができます。
メンテナンス画面ではプロジェクトで使用しているすべての識別コードの内容のモニタとデータの注入を行うことができます。

① 表示する識別コードを選択します。
② 選択された識別コードの内容が表示されます。（一定周期で表示内容は更新されます。）
③ チェックボックスを ON に設定すると、該当する識別コードをプログラム側で読書きした場合にログ情報が書込まれます。
④ データを注入したいデータ領域をマウス左ボタンでクリックし、キーボードよりデータを入力することができます。　WORD 型データは16進数4桁で入力します。

［一括設定］ ボタン　　すべての識別コードのチェックボックスを ON に設定します。
［一括解除］ ボタン　　すべての識別コードのチェックボックスを OFF に設定します。

［注意］ 識別コード読書き時のログ出力を有効にするためにはアプリケーションプログラム側での準備が必要です。

```
Set_Color(BUTTON, 0);                    ログ情報は出力されません。
Set_Color(BUTTON, 0, S_FILE, S_LINE);    ログ情報が出力されます。
```

パッケージプログラム SimplePack で公開しています関数の引数には、ソースファイル名と行番号を マクロ定数 S_FILE / S_LINE で指定することができます。 ログ出力はソースファイル名行番号が設定されている場合にのみ出力されます。
（エラー時には無条件に出力されます。）

ログ出力レベル

ログメッセージを出力するとき、各メッセージの重要度に応じて 0～9 の出力レベル（0:重要度 高）を付加します。
ログ出力プログラムでは、現在設定中の出力レベルとログメッセージに付加された出力レベルを比較し出力レベルが若いメッセージのみを保存します。（現在の出力レベルが 5 に設定されている時、0～5 の出力レベルが設定されたメッセージを保存します。）
ログ出力レベルの初期値は環境設定ファイル（INI ファイル）で指定し、メンテナンス画面で変更することができます。

SimpleMaster へのお問合せは下記へお願いします。

株式会社	フジテクニカ
住所	〒８１２－００１３
	福岡市博多区博多駅東２丁目９番２５号
TEL	092-431-0800
FAX	092-431-2018
URL	http://www.fuji-technica.jp/
E-Mail	master@fuji-technica.jp