○「New MQL4；標準クラスを使ってみる（初めの二歩目)」　2014.09.01

・アメンボです、

クラスを使いこなすのは難しい！（アメンボの感想)

と、言うわけで「標準クラス」の活用に焦点を絞っているアメンボは、

当面の間は「MQL5 の遺産を掘り起こし」ては NewMQL4 で使えるか否かを試しています。

・なぜ、標準クラス・ライブラリに拘（こだわ）るか、

チョット複雑な処理のコードを書いてみると判ります。

標準関数で書くより、標準クラスを使った方が圧倒的に簡単で短いコードで済むからです。

（でも、なんて初心者用の資料が少ないのでしょう！）

・ところで、NewMQL4（Build600 以降）ですが、

つい数週間前には MQL4_Reference に載っているのにコンパイル・エラーになった関数が、

新しいバージョンでは使えるようになっている。と、言う状況に気が付きましたか？

「MQL4_Reference」に載っている内容は最終的に全てサポートされる、と推測しています。

＜本稿で使用した MQL4 コード＞

※※使用コードと「textdisplay.mqh」ファイルを添付しました。

「new_mql4_2014_09_01.zip；New MQL4 標準クラス（二歩目)」(ZIP 形式圧縮)

※本稿は「MT4；version 4.00 Build670」「MetaEditor；version 5.00 Buid934」にて確認済み。

目次：

１．標準クラスの調査進捗表

※New_MQL4 の資料が余りに少ないので、MQL5 の過去の資料を掘り起こしながら進めています。

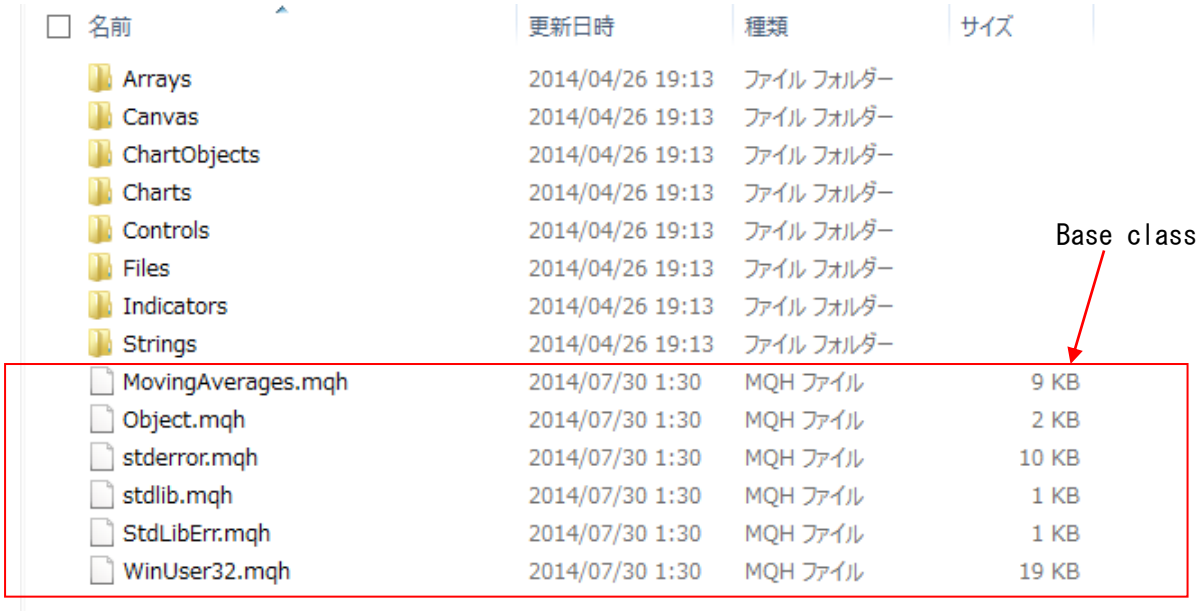
＜全体像；クラスを利用するための「.mqh」ファイルが在る場所＞

MQL5；ライブラリの種類	MQL5； Include¥ **	NewMQL4； Include¥ **	New_MQL4；Location 内容と「ホルダ有無」
Base class	¥	¥	本稿で一部使用・解説
Classes of data	¥Arrays¥	¥Arrays¥	本稿で一部使用・解説
Classes for file operations	¥Files¥	¥Files¥	未確認
Classes for string operations	¥Strings¥	¥Strings¥	未確認
Classes for graphic objects	¥Objects¥	¥Object.mqh	「フォルダ無い！？」※１
(Classes for chart objects)	無い！	¥ChartObjects¥	本稿で一部使用・解説※２
Class for creating custom graphics	¥Canvas¥	¥Canvas¥	済；チョット使ってみた
Class for working with chart	¥Charts¥	¥Charts¥	未確認
Technical indicators	¥Indicators¥	¥Indicators¥	未確認
Trade classes	¥Trade¥	－見当たらず－	「無い！？」
Trading strategy classes	¥Expert¥	－見当たらず－	「無い！？」
Classes for creation of control panels and dialogs	¥Controls¥	¥Controls¥	未確認

※１；MQL5 に比較すると、専用フォルダーは無くなって、「Object.mqh」ファイルが
　　Base_class として存在してる。(本稿のコードでも継承しています)

※２；「¥ChartObjects フォルダ」は MQL5 では「¥Objects¥」の Base_class として収録されている。

＜New_MQL4 の Include 時ベース・フォルダ「MQL4¥Include」内の構成＞

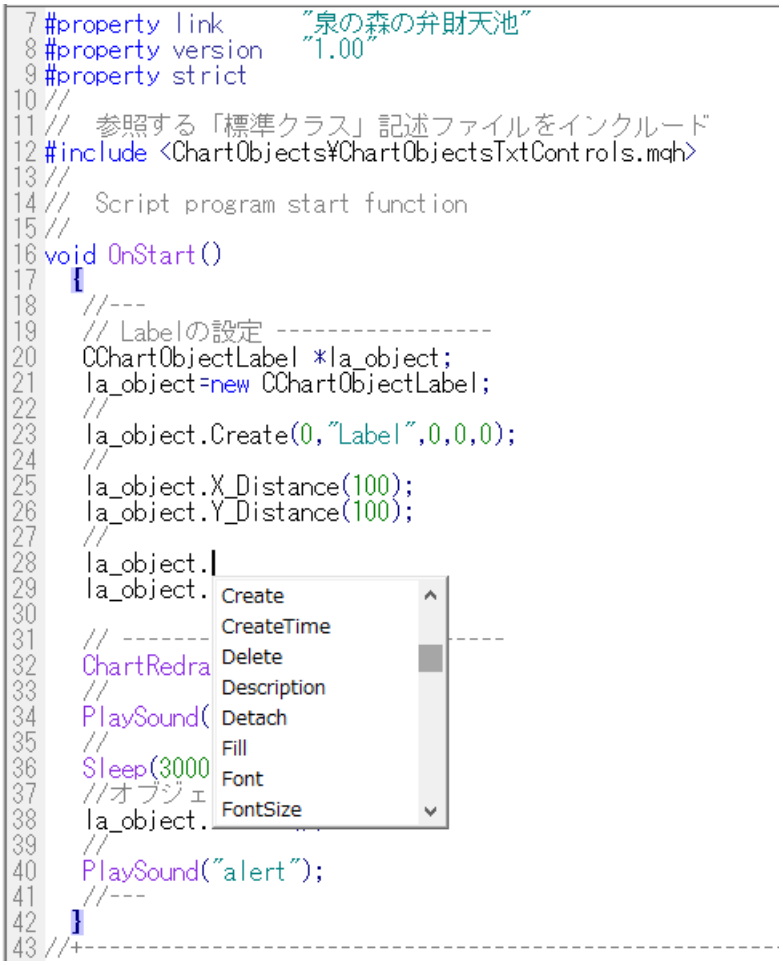


２．基礎の解析（小さな一歩）；スクリプトで試す

・標準クラスの内容は膨大です、本稿ではその中の一部を解説します。（コツコツ解析中の成果）
#include 時のベース・フォルダである「MQL4¥Include」の中にある、
<ChartObjects¥ChartObjectsTxtControls.mqh>ファイルを覗いてみると、以下のような
標準クラスが定義されています。

	MQL4¥Include¥ChartObjects フォルダ内			
Object.mqh	ChartObject.mqh	ChartObjectsTxtControls.mqh		備考
曽祖父クラス	祖父クラス	親クラス	子クラス	
CObject	CChartObject	CChartObjectText	CChartObjectLabel	本稿で解説
			CChartObjectEdit	
			CChartObjectButton	
			CChartObjectRectLabel	本稿で解説
	Delete()と Description()等は ここに在る			

※継承関係を理解しようとすると「頭が痛くなります」が、コード作成時は「使える関数」は、
コード入力時に候補として表示されます。（便利！）
例えば「CChartObjectLabel」クラスを使う場合には、このクラス内で定義されたもの以外にも、
祖父クラスで設定された「Delete()」や「Description()」なども候補として表示されますし、
コンパイルも通り、ちゃんと動作します。（#include を忘れたらダメです）
例；（候補表示）



（1）「Label クラス；CChartObjectLabel」を使ってみる

－1．コード

```
//+------------------------------------------------------------------+
//|                                                Class_Label_01.mq4 |
//|                                                           amenbo |
//|                                                 泉の森の弁財天池 |
//+------------------------------------------------------------------+
#property copyright "amenbo"
#property link      "泉の森の弁財天池"
#property version   "1.00"
#property strict
//
//  参照する「標準クラス」記述ファイルをインクルード
#include <ChartObjects¥ChartObjectsTxtControls.mqh>
//
//  Script program start function
//
void OnStart()
  {
   //---
   // Label の設定 -----------------
   CChartObjectLabel *la_object;
   la_object=new CChartObjectLabel;
   //
   la_object.Create(0,"Label",0,0,0);
   //
   la_object.X_Distance(100);
   la_object.Y_Distance(100);
   //
   la_object.Description("GOOD!---WOW");
   la_object.Color(Blue);

   // ------------------------------
   ChartRedraw();
   //
   PlaySound("alert2");
   //
   Sleep(3000);
   //オブジェクトを削除
   la_object.Delete();
   //
   PlaySound("alert");
   //---
  }
//+------------------------------------------------------------------+
```

－2．結果

※「３秒」後に表示は消えます

（2）「RectLabel クラス；CChartObjectRectLabel」と「Label クラス」を一緒に使ってみる

－１．コード

```
//+------------------------------------------------------------------+
//|                                                Class_Label_03.mq4 |
//|                                                            amenbo |
//|                                                  泉の森の弁財天池 |
//+------------------------------------------------------------------+
#property copyright "amenbo"
#property link      "泉の森の弁財天池"
#property version   "1.00"
#property strict
//
//  参照する「標準クラス」記述ファイルをインクルード
#include <ChartObjects¥ChartObjectsTxtControls.mqh>
//
//  Script program start function
//
void OnStart()
  {
   //---
   // RectLabel の設定   ----------
   CChartObjectRectLabel *rla_object;
   rla_object=new CChartObjectRectLabel;
   //
   rla_object.Create(0,"Label1",0,0,0,0,0);
   //
   rla_object.X_Distance(100);
   rla_object.Y_Distance(80);
   //
   rla_object.X_Size(100);
   rla_object.Y_Size(50);
   //
   rla_object.BackColor(clrLightCyan);
   //

   // Label の設定 -----------------
   CChartObjectLabel *la_object;
   la_object=new CChartObjectLabel;
   //
   la_object.Create(0,"Label2",0,0,0);
   //
   la_object.X_Distance(100);
   la_object.Y_Distance(100);
   //
   la_object.Description("GOOD!---WOW");
   la_object.Color(Red);

   // ------------------------------
   ChartRedraw();
   //
   PlaySound("alert2");
   //
   Sleep(3000);
   //オブジェクトを削除
   rla_object.Delete();
   la_object.Delete();
   //
   PlaySound("alert");
   //---
  }
//+------------------------------------------------------------------+
// 注記；
//「.Description」は「CChartObjct」（２つ先の mqh ファイル）から継承
```

```
//「.Color」も「CChartObjct」（２つ先）から継承
//２つ先から継承したものでも「コード」の候補としてコード入力時に候補として表示される！
/*
   調査中；
   CChartObjectRectLabel は四角いラベルを表示する。
   この上に「文字」を置くには CChartObjectLabel を使うしかないのか？
*/
```

※チョット、判らないことが残っているので「コード」中にメモしておきました。

－２．結果

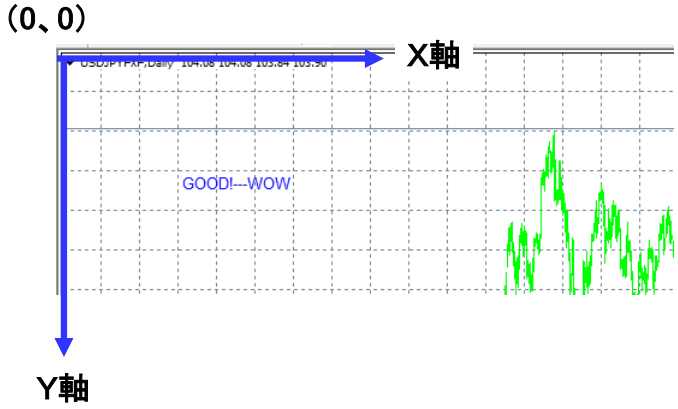
※やはり、「３秒」後に消えます。

（３）コード解説

※「クラスの使い方」に関する基礎知識の解説は省略します。（参照可能な資料は山の様にあります）
本稿では、NewMQL4 標準クラス・ライブラリの使い方を中心に解説します。

－１．座標系

標準クラスで扱うオブジェクトの配置では、デフォルトでは座標系を採用しています。
チャートの左上を「原点；(0、0)」とし、下記に示す方向にＸ軸座標とＹ軸座標を
設定します。（単位は通常ピクセルです）



この座標系、つまり原点（0,0）を「左上・左下・右上・右下」の何所に設定するかは
変更が可能です。

－２．クラスを動的に確保する

Ｃ＋＋では良く見かけるように、new 関数で動的に確保して使うのが一般的です。

```
CChartObjectLabel *la_object;
la_object=new CChartObjectLabel;
```

ただ、通常のＣ＋＋言語と違うのが、new 関数で動的に確保したクラスのデータや
メソッド（クラス内関数）を使うのに、NewMQL4（MQL5）では「アロー演算子；"->"」
ではなく、「ドット演算子；"."」でアクセスするのが、一般的？のように観えることです。
（MQL5 のコードを観るかぎりですが）

```
la_object.Create(0,"Label",0,0,0);
```

－３．Label クラスのメソッド（関数）

※「Class_Label_01.mq4」で使用したもののみ解説します、
その他のメソッド（関数）については「ChartObjectsTxtControls.mqh」、または、
「ChartObject.mqh」を参照ください。

メソッド（関数）／機能	引数とその内容
bool Create(long chart_id,const string name,const int window,const int X,const int Y);	
ラベル・オブジェクトを作成する	・long chart_id　　；オブジェクトを配置（表示）する為替ペアのチャート No、現在表示中の為替チャートなら「0」 ・const string name；オブジェクトの「名前」 ・const int window　；オブジェクトを設定するウインドウ番号、メインウインドウなら「0」、１個目のサブウインドならば「1」を設定 ・const int X　　；オブジェクトを配置する「Ｘ座標」 ・const int Y　　；オブジェクトを配置する「Ｙ座標」 ※X、Y の単位はピクセル
bool X_Distance(const int X);	
bool Y_Distance(const int Y);	
	オブジェクトの配置座標を（X、Y）に設定する
※以下の３つは、「ChartObject.mqh の CChartObject クラス」から継承しているメソッド	
bool Description(const string new_text);	
	・const string new_text；テキストを記述する
bool Color(const color new_color);	
	・ラベルテキストのカラーを設定する
bool Delete(void)	
	・オブジェクトを削除する

※同じ名前のメソッドで、引数が「void」の場合は「その値を取得する」と、言うものが結構あります。
例；「int X_Distance(void)」は X 座標を取得します。

－４．RectLabel クラスのメソッド（関数）

※「Class_Label_01.mq4」と比較し、設定方法が異なるメソッド、および
「Class_Label_03.mq4」で新たに使用したメソッドのみを解説します。

メソッド（関数）／機能	引数とその内容
bool Create (long chart_id,const string name,const int window,const int X,const int Y, const int sizeX, const int sizeY);	
四角ラベル・オブジェクトを作成する	・long chart_id　　；オブジェクトを配置（表示）する為替ペアのチャートNo、現在表示中の為替チャートなら「0」 ・const string name；オブジェクトの「名前」 ・const int window ；オブジェクトを設定するウインドウ番号、メインウインドウなら「0」、１個目のサブウインドならば「1」を設定 ・const int X　　；オブジェクトを配置する「X座標」 ・const int Y　　；オブジェクトを配置する「Y座標」 ・const int sizeX　；四角ラベルの幅 ・const int sizeY　；四角ラベルの高さ ※X、Y の単位はピクセル
bool X_Size(const int X);	
bool Y_Size(const int Y);	
	・四角ラベルの大きさ（縦横）を設定する
bool BackColor(const color new_color);	
	・四角ラベルのバックカラーを設定する

※注意；

RectLabel クラスでは「Create(0,"Label",0,0,0,0,0)」と記述するべきところを
「Create(0,"Label",0,0,0)」と記述すると、エラーは出ませんが成城には動きません。

※名前が同じで、引数（数やデータ型）が異なるメソッド（関数）がゴロゴロ存在していて頭が痛いです！（オーバーロードも使い過ぎると、訳わからん！）

※当たり前ですが、「標準クラス」で実現できることは、全て NewMQL4 の「標準関数」を使って実現できることばかりです。
でも、「標準クラス」を覚えて使う方が、コード作成が遥かに楽になります。（本当！です）

３．インディケータに拡張するには（ヒントのみ）

※スクリプトからインディケータに拡張するためのヒントとなるコードとその結果を示します、
解析と更なる拡張は読者にお任せします。(「２.」を読んだ読者には難しくないはず)

（1）「Label クラス」＋「Button クラス」の場合

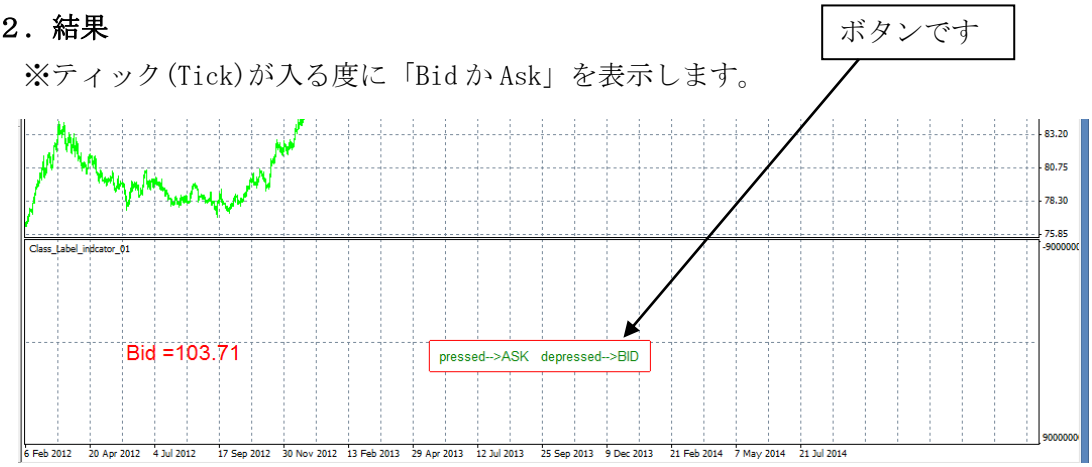
－１．コード

```
//+------------------------------------------------------------------+
//|                                    Class_Label_indcator_01.mq4 |
//|                                                         amenbo |
//|                                                泉の森の弁財天池 |
//+------------------------------------------------------------------+
#property copyright "amenbo"
#property link      "泉の森の弁財天池"
#property version   "1.00"
#property strict
#property indicator_separate_window
//  参照する「標準クラス」記述ファイルをインクルード
#include <ChartObjects¥ChartObjectsTxtControls.mqh>
//
   CChartObjectLabel *la_object;
   CChartObjectButton *la_button;
//
//  Custom indicator initialization function
//
int OnInit()
  {
   //--- indicator buffers mapping
   // Label の設定 ------------------
   la_object=new CChartObjectLabel;
   //la_object.Create(0,"Label",0,0,0);
   la_object.Create(0,"Label",1,0,0);
   la_object.X_Distance(100);
   la_object.Y_Distance(100);
   la_object.FontSize(16);
   //
   // Button の設定 --------------
   la_button=new CChartObjectButton;
   la_button.Create(0,"Button",1,0,0,0,0);
   la_button.X_Distance(400);
   la_button.Y_Distance(100);
   la_button.X_Size(220);
   la_button.Y_Size(30);
   la_button.BackColor(White);
   la_button.BorderColor(Red);
   la_button.Description("pressed-->ASK   depressed-->BID");
   la_button.Color(Green);
   la_button.FontSize(10);
   //---
   return(INIT_SUCCEEDED);
  }
//
void OnDeinit(const int _reason)
  {
   //オブジェクトを削除
   la_object.Delete();
   la_button.Delete();
   //
   PlaySound("alert2");
  }
//
//  Custom indicator iteration function
//
int OnCalculate(const int rates_total,
```

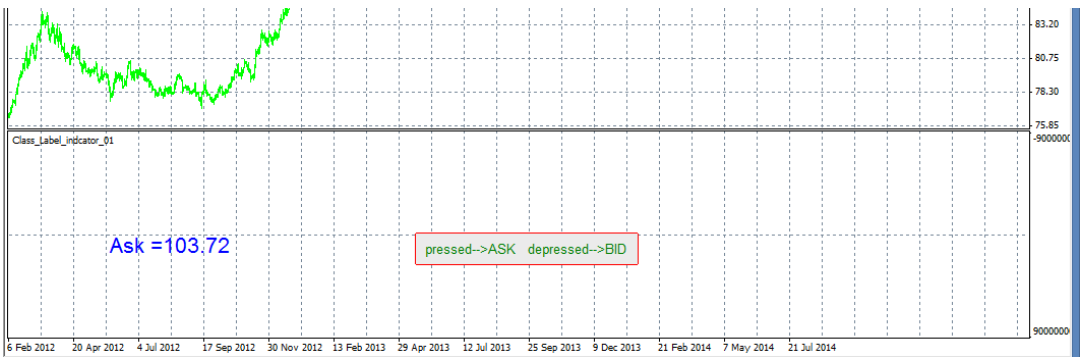
```
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   //--- return value of prev_calculated for next call
   if(la_button.State()==true)
   {
          string price_1="Ask ="+DoubleToString(Ask,2);
          la_object.Description(price_1);
          la_object.Color(Blue);
   }
   //
   if(la_button.State()==false)
   {
          string price_2="Bid ="+DoubleToString(Bid,2);
          la_object.Description(price_2);
          la_object.Color(Red);
   }
   //
   PlaySound("alert");
   ChartRedraw();
   //
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

－２．結果

※ティック(Tick)が入る度に「Bid か Ask」を表示します。



※ティックが入った時、ボタンが OFF 状態では「Bid」を、ON 状態では「Ask」を表示します。
　ボタンをクリックする度に「ON⇔OFF」が切替わります。



４．実用的な応用例（MQL5 の遺産から）

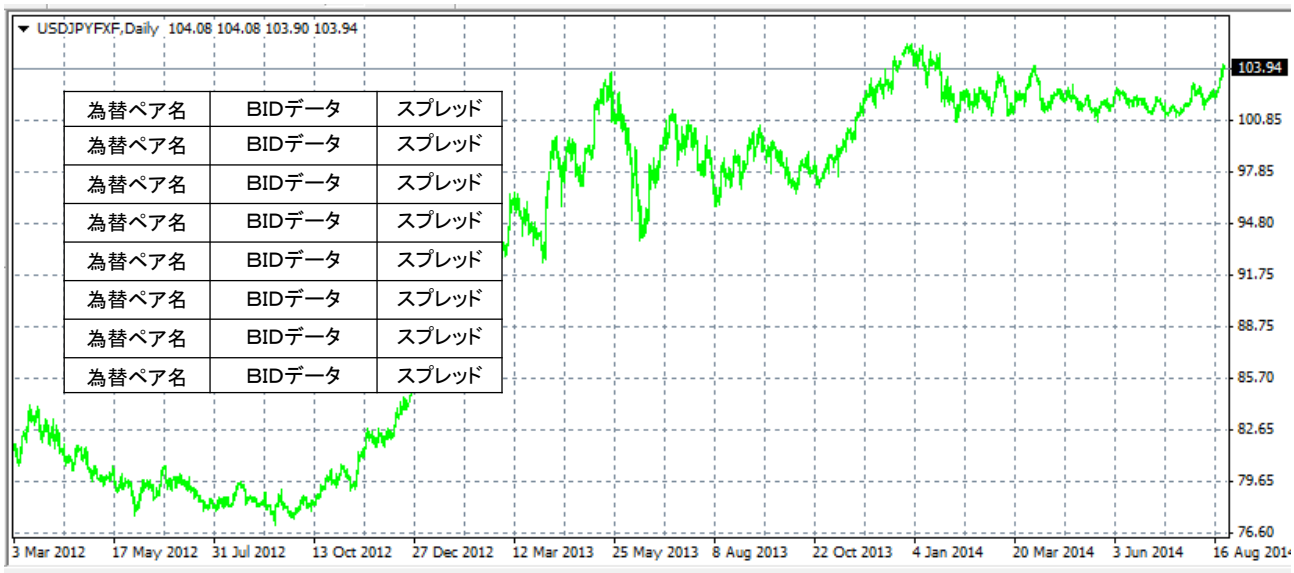
※出典；「Create your own Market Watch using the Standard Library Classes」
上記資料は MQL5 Site 上にあります、A4 で 28 枚の分量（多くは無い）ですが、
解読するのに結構時間がかかります、筆者も未だ解析中です。
（判りやすい資料が余りにも少ないので、トレジャー・ハンターの気分です）

・上記資料には２つの「.mqh」ファイルが添付されており、当然「標準クラス」ではないのですが、
その内の「textdisplay.mqh」が准汎用タイプとして結構使えるので、本稿で使い方を紹介します。

※「２．基礎の解析」で解説した内容から、かなり飛躍しますが、
「（出典）資料内容の例」は非常に実用的なので、解説することにいたします。

（１）本稿で実現する内容（実現仕様）

・出典元の資料にあった表示例の１つです、ただ資料ではコードが汎用性の盛り込み過ぎで
「読み取り難い」こと、及びコードの解説が「初心者向きで無い」（省略しすぎ）なので、
本稿では「コードの単純化」と「解説の盛り込み」に注力しました。
・下記が「実現する表示」です。
左が通貨ペア、真ん中がその値、右端がスプレッドです



※なお、チャートの画面サイズを変えても、表示位置は変化しません。

（２）準備（ヘッダファイルの設定）

①本稿で使用する「textdisplay.mqh」ファイルは、出典「資料」に添付されています。
②上記の「.mqh」ファイルは「MQL4¥Include」フォルダ内（ベース・フォルダ）に入れてください。
※「textdisplay.mqh」は、添付の「.ZIP」ファイルに入れておきました。

※このベースホルダに入れると、「#include <テキスト名>」で取り込めます。

※「textdisplay.mqh」には、コード中で使用する各種のクラスの定義が記載されています。

（3）使用コード

```
//+------------------------------------------------------------------+
//|                                                PriceList_03.mq4 |
//|                                                          amenbo |
//|                                                泉の森の弁財天池 |
//+------------------------------------------------------------------+
#property copyright "amenbo"
#property link      "泉の森の弁財天池"
#property version   "1.00"
#property strict
#property indicator_chart_window
//
// ------------ Custom indicator -------------
//      Include libraries
#include        <TextDisplay.mqh>
//      External input parameters
input int   Y_=2;//Y方向のオフセット
input int   X_=1;//X方向のオフセット
//---------------------------------------
TableDisplay      Table1;
//---------------------------------------
#define NUMBER  8
//---------------------------------------
string    names[NUMBER]={ "EURUSDFXF","GBPUSDFXF","AUDUSDFXF","NZDUSDFXF","USDCHFFXF",
                          "USDCADFXF","USDJPYFXF","EURJPYFXF" };
int       coord_y[ NUMBER ] = { 0, 1, 2, 3, 4, 5, 6, 7 };
//---------------------------------------
double     rates[NUMBER];
datetime   times[NUMBER];
MqlTick    tick;
//---------------------------------------
int OnInit()
  {
   ArrayInitialize(times,0);
   ArrayInitialize(rates,0);
   //   ----Create table----
   //   Set coordinates ------
   Table1.SetParams(0,0,CORNER_LEFT_UPPER);
```

```
   //   Show headers ------
   for(int i1=0; i1<NUMBER; i1++)
     {
       Table1.AddTitleObject(40,40,X_,Y_+coord_y[i1],names[i1]+" :",Blue);
     }
   //   Show prices ------
   for(int i2=0; i2<NUMBER; i2++)
     {
       Table1.AddFieldObject(40,40,X_+2,Y_+coord_y[i2],Blue);
     }
   //   Show spreads ------
   for(int i3=0; i3<NUMBER; i3++)
     {
       Table1.AddFieldObject(40,40,X_+4,Y_+coord_y[i3],Blue);
     }

   //
   RefreshInfo();
   ChartRedraw(0);
   EventSetTimer(1);
   //
   return(0);
  }
//  Custom indicator iteration function ----------------
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   //--- return value of prev_calculated for next call
   return(rates_total);
  }
//      OnTimer event handler ---------------------
void OnTimer()
  {
   RefreshInfo();
   ChartRedraw(0);
  }
//      OnDeinit event handler --------------------
void OnDeinit(const int _reason)
  {
   EventKillTimer();
//      Delete table
   Table1.Clear();
  }
//      Refresh info ------------------------------
void RefreshInfo()
  {
   for(int i4=0; i4<NUMBER; i4++)
     {
      //        Get price data
    bool tick_=SymbolInfoTick(names[i4],tick);//「MqlTick tick」に最新データをセットする処理
```

```
      //
      if(tick.time>times[i4] || times[i4]==0)
        {

Table1.SetText(i4+NUMBER,DoubleToString(tick.bid,(int)(SymbolInfoInteger(names[i4],SYMBOL_D
IGITS))));
         if(tick.bid>rates[i4] && rates[i4]>0.1)
           {
            Table1.SetColor(i4+NUMBER,Lime);
           }
         else if(tick.bid<rates[i4] && rates[i4]>0.1)
           {
            Table1.SetColor(i4+NUMBER,Red);
           }
         else
           {
            Table1.SetColor(i4+NUMBER,Blue);
           }
       //
         rates[ i4 ] = tick.bid;
         times[ i4 ] = tick.time;
        }

Table1.SetText(i4+NUMBER+NUMBER,DoubleToString((tick.ask-tick.bid)/SymbolInfoDouble(names[i
4],SYMBOL_POINT),0));
      }
  }
//+------------------------------------------------------------------+
```

（4）実行結果

※真中の列の「BID 値」は、上昇が「ライム色」、下降が「赤色」、変わらずを「青色」にしました。

５．応用例の解説

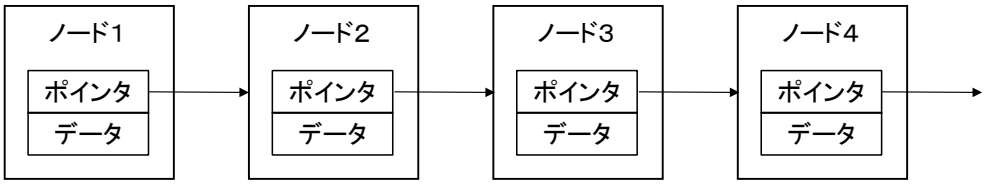
※基礎知識、応用方法、クラスの継承など「TextDisplay.mqh」に係る解説をいたします。

（１）基礎知識（TextDisplay.mqh を読み解くための）

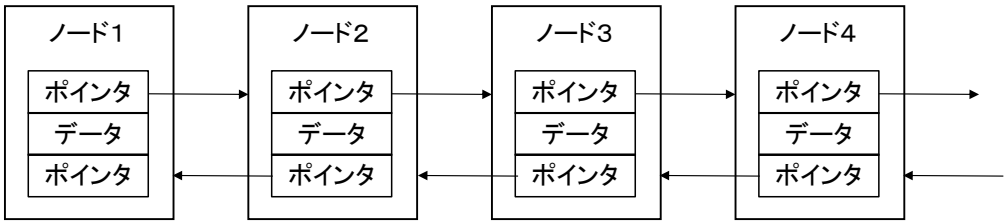
※「TextDisplay.mqh、List.mqh」で使われている線形リストの基本をおさらいします。
（と、言ってもＣ＋＋初心者の筆者にとって初めての概念でしたので、間違いがあればご容赦）

－１．「線形リスト」とは何か（大雑把に解説します）

・１次元配列と同様に、データの列を扱う
・線形リストは「データとポインタ」が入った「ノード」と呼ばれる要素を、
そのノード中の「ポインタ」でつないだもの。
・シングルリンク ;「ノード」の「ポインタ」は、次の要素（のアドレス）を指している。



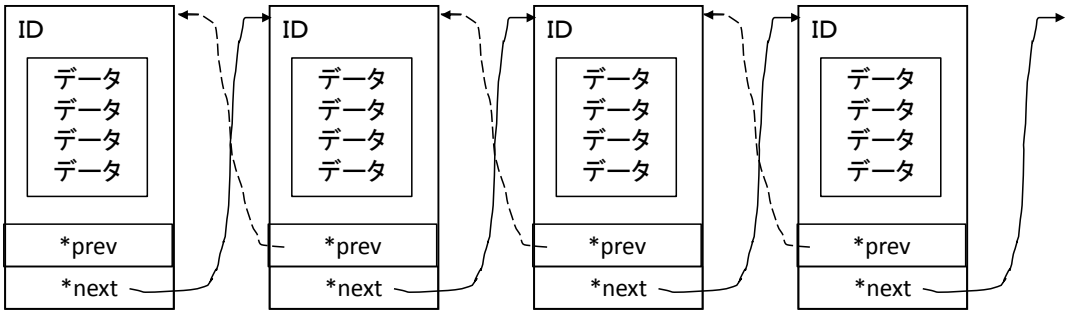
・ダブルリンク ;「ノード」は次の要素のアドレスと、前の要素のアドレスを持っている



・ポインタとは、その中に「アドレス」が格納されている領域です、
例；シングルリンクの場合、「ノード２」のポインタには「ノード３」の先頭アドレスが
格納されているので、次々に「ノード」を辿ってデータを得ることが出来ます。

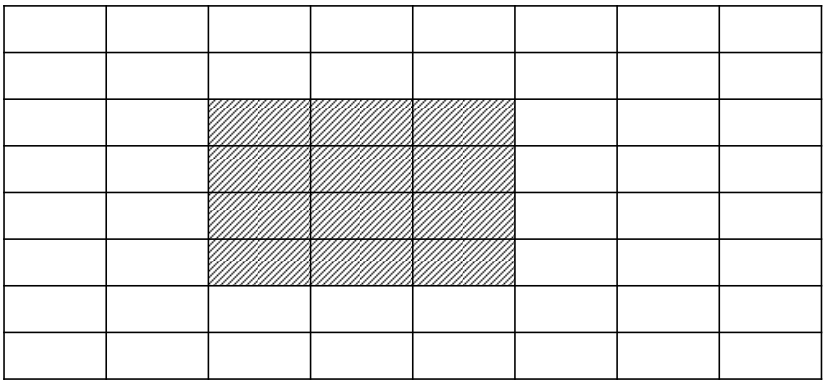
※より詳細な解説は、例えば以下の資料を参照ください。
「MQL5 Programming Basics: Lists」、「Using the Object Pointers in MQL5」
（少々難解なので、筆者も未だ一部を読んだだけです。）

・本稿では「ノード」１個に「データ」１個の「リスト」形式で使用していますが、
構造体をポインタでつないで「リンク構造」にした場合は、「ノード」１個に複数個の
データを格納できます。
以下に構造体をリンク構造にした場合のイメージ（上記の資料より抜粋）を示します。



－２．「テーブル」とは何か

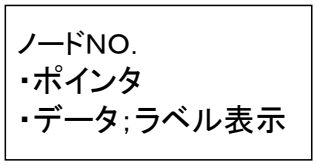
※本稿で利用した「TextDisplay.mqh」で採用している「テーブル」の概念を解説します。



テーブルとは；
「チャート表示領域全体」を任意個の升目に区切って、例えばその内の特定領域、「ハッチング部」をデータの表示（収納）領域として設定したものです。
（あくまで、本稿での定義ですが）

・テーブルの「１セグメント（１升目)」には、「１個のノード（ポインタとデータ)」が対応（収納）します。
　データはラベル（テキスト形式）に変換され、表示されます。
・テーブル上の特定セグメントをアクセスするときは、「行・列」（又は「列・行」）で指定します。

※１セグメントの拡大；



・そして、各「ノード」は線形リスト構造で繋がっています。
　上の例で言えば「１２個（＝４×３)」のノードがあり、それぞれに「データ；ラベル表示」が格納（表示）されています。

（２）応用クラスを使ってテーブルを作成する

※ここで言う「テーブル」とは、
　①特定のセグメント（＝ノード）にデータがラベルとして格納表示され、
　②且つセグメント（ノード）は線形リスト構造をしている、
ものである。

※応用クラスとは、「TextDisplay.mqh」で定義されているもの、として解説します。

－１．テーブルを作成する「応用クラス」には「２つ」のメソッドがあります

AddTitleObject(＊＊)と AddFieldObject(＊＊)ですが、両者の違いは下記のみです。

	ラベル表示データ	備　考
AddTitleObject()	引数で指定したテキスト	後で書換え可能
AddFieldObject()	""；つまり空白	後で書換え可能

※データを引数で渡せるか否か、だけの違い。

－２．メソッドの具体的な使い方

※２つのメソッドをまとめて解説します。（違いは殆どありません）

［座標系指定メソッド］　＜TableDisplay クラス＞

```
void  SetParams(long _chart_id,int _window,ENUM_BASE_CORNER _corner=CORNER_LEFT_UPPER);
```

※引数が「３個」の SetParams です。（同じ名前で、引数の数が違うものあり）

long _chart_id ;　　為替ペアのチャートＩＤ、現在表示しているものなら「0」。

int _window ;　　メインウインドウは「0」、次のサブウインドウなら「1」

ENUM_BASE_CORNER _corner=CORNER_LEFT_UPPER ;　座標系を左上原点（0,0）のものに設定

［テーブル作成メソッド］　＜TableDisplay クラス＞

＜TTitleDisplay クラス＞

```
int   AddTitleObject(int _cols,int _lines,int _col,int _row,string _title,color _color,
                     string _fontname="Arial",int _fontsize=10);
```

＜TFieldDisplay クラス＞

```
int  AddFieldObject(int _cols,int _lines,int _col,int _row,              color _color,
                     string _fontname="Arial",int _fontsize=10);
```

int _cols,int _lines ;　・チャートのフルサイズを任意個数の升目に分割します。
横（Ｘ軸方向）を「_cols」個に分割、
縦（Ｙ軸方向）を「_lines」個に分割。

int _col,int _row ;　・分割された升目の内、特定の升目（セグメント）を指定します。
「_col 行・_row 列」目の升目（セグメント）

string _title ;　AddTitleObject の場合
・引数に設定したデータ（string _title）を「指定した升目」に
①ラベルとして表示し、
②データは線形リストのノードに格納し、
③最後にノード番号（及びインデックス）を「＋１」します。

AddFieldObject の場合
・「指定した升目」には、
①常に「""；空白文字」データをラベルとして表示し、
②データは線形リストのノードに格納し、
③最後にノード番号（及びインデックス）を「＋１」します。

color _color, string _fontname="Arial",int _fontsize=10 ;
・全てラベルとして表示するテキストの属性指定です。
文字色、フォント、フォント・サイズ。

※AddTitleObject と AddFieldObject は、どちらも１回実行すると、
①ノードが１つ増えていき、
②インデックス（0 から始まる）も「＋１」されます
実行する度に「線形リスト」の長さが増えていきます。

［テーブル内容の書換えメソッド、その他］

※AddFieldObject(実は AddTitleObject も)で設定したデータは、下記メソッドで書換える事が出来ます。

＜TableDisplay クラス＞

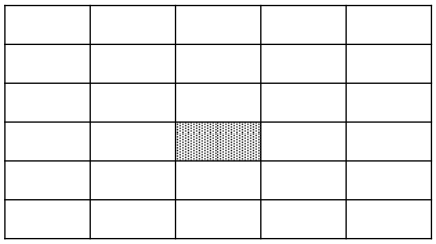
bool **SetText**(int _index,string _text);

int _index ;　　インデックス番号で、ノードを指定します、
インデックス番号は、下記の様に「0」から始まり、
AddFieldObject か AddTitleObject を実行する度に「＋１」されて行きます。
（線形リスト構造で、データが繋がって行くわけです）

実行順	実行メソッド	ノード番号	インデックス番号（_index）
1	AddTitleObject	1	0
2	AddTitleObject	2	1
3	AddFieldObject	3	2
4	AddFieldObject	4	3
5	AddTitleObject	5	4
6	AddFieldObject	6	5
7	AddFieldObject	7	6
8	AddFieldObject	8	7
9	AddFieldObject	9	8

※特定の「ノード」が、どの「升目（セグメント)」に対応するのかは、
「int _col,int _row」のセットで決めることに注意してください。

テーブル上の表示位置は、規則的に配置することもランダムに配置することも可能です



例；左のセグメントは、
_col＝４行、_row＝３列
で、指定される。

string _text ;　ノード内の「データ（テキスト形式)」を「_text」に書換えます。

bool **SetColor**(int _index,color _color) ;

int _index ;　　ノードを指定するインデックス番号

color _color ;　文字色を指定

（3）少し詳細に応用クラスを解説する

※読者の参考用として、また筆者の記憶用として「TextDisplay.mqh」に係るクラスと
その継承関係を解説・記録します。
（正直、かなり複雑なので、記録しておかないと忘れてしまいそうです）

－１．クラス継承規則など；

継承の書き方；

class 派生クラス名　：継承修飾子　基底クラス名

{　　};

継承修飾子の意味；

- ●public・・・ 基底クラスで設定したアクセス修飾子の設定をそのまま引き継ぐ
- ●protected・・基底クラスで public だったものを protected にして引き継ぐ。他はそのまま。
- ●private・・・基底クラスのメンバを全て private で引き継ぐ。

※通常は、「継承修飾子＝public」が殆ど。

クラス内メソッド（関数）の定義記載法；

class クラス名　：：メソッド（関数）名

{　　};

－２．「TextDisplay.mqh」にて定義したクラスの継承関係

＜継承関係＞

<table>
<tr><td colspan="4">標準クラス；「MQL4¥Include」フォルダ中に収録されてる</td><td>応用クラス</td></tr>
<tr><td>先祖ｸﾗｽ</td><td>曽祖父ｸﾗｽ</td><td>祖父ｸﾗｽ</td><td>親クラス</td><td>クラス／・関数</td></tr>
<tr><td>Object.mqh</td><td>ChartObjects¥ChartObject.mqh</td><td colspan="2">ChartObjects¥ChartObjectsTxtControls.mqh</td><td>textdisplay.mqh</td></tr>
<tr><td rowspan="4">CObject</td><td rowspan="2">CChartObject</td><td rowspan="2">CChartObjectText</td><td>CChartObjectLabel</td><td>TTitleDispiay</td></tr>
<tr><td>CChartObjectEdit</td><td>TFieldDisplay</td></tr>
<tr><td rowspan="2"></td><td colspan="2">Arrays¥List.mqh</td><td rowspan="2">TableDisplay
・SetParams()
・AddTitleObject()
・AddFieldObject()</td></tr>
<tr><td></td><td>CList</td></tr>
</table>

※「＊＊.mqh」はクラスが記述されているファイル名

※NewMQL4 の全てのオブジェクトのご先祖は、「CObject クラス」です。
そもそも NewMQL4 のオブジェクトは、「CObject クラス」の定義をみると判るのですが、
「ノード」による「リンク構造」を採れるように定義されている。
（「Object.mqh」は内容が１頁しかないので、読んでみてください）

－３．詳細なクラス継承関係と機能を解説します

※本節では、TableDisplay クラスの「**AddFieldObject クラス**」に焦点を絞り、理解するために必要な解説を試みます、と言っても、ただ列挙しているだけですが。

＜TextDisplay.mqh＞より ;

```
//--------------------------------------------------------------------
//      Include libraries
//--------------------------------------------------------------------
#include        <ChartObjects¥ChartObjectsTxtControls.mqh>
#include        <Arrays¥List.mqh>
        ・・・・・・・・・・・・・
//--------------------------------------------------------------------
protected:
   long             chart_id;
   int              sub_window;
   long             chart_width;                        // chart width in pixels
   long             chart_height;                       // chart height in pixels
   long             chart_width_step;                   // chart width step
   long             chart_height_step;                  // chart height step
   int              columns_number;                     // number of columns
   int              lines_number;                       // number of rows
        ・・・・・・・・・・・・・・・・
//--------------------------------------------------------------------
//      TableDisplay class (List of objects)
//--------------------------------------------------------------------
class TableDisplay : public CList
  {
protected:
   long             chart_id;
   int              sub_window;
   ENUM_BASE_CORNER  corner;
public:
   void        SetParams(long _chart_id,int _window,ENUM_BASE_CORNER _corner=CORNER_LEFT_UPPER);
   int         AddTitleObject(int _cols,int _lines,int _col,int _row,string _title,
                              color _color,string _fontname="Arial",int _fontsize=10);
   int         AddFieldObject(int _cols,int _lines,int _col,int _row,
                              color _color,string _fontname="Arial",int _fontsize=10);
   bool             SetColor(int _index,color _color);
   bool             SetFont(int _index,string _fontname,int _fontsize);
   bool             SetText(int _index,string _text);
   bool             SetAnchor(int _index,ENUM_ANCHOR_POINT _anchor);
public:
   void             TableDisplay();
   void            ~TableDisplay();
  };
        ・・・・・・・・・・・・・・・・・・
//--------------------------------------------------------------------
//      Add Edit Field object
//--------------------------------------------------------------------
int TableDisplay::AddFieldObject(int _cols,int _lines,int _col,int _row,
                                 color _color,string _fontname,int _fontsize)
  {
① TFieldDisplay      *field=new TFieldDisplay();
② field.Create( this.chart_id, this.sub_window, _cols, _lines, _col, _row );
③ field.Description( "" );
   field.Color( _color );
   field.Font( _fontname );
   field.FontSize( _fontsize );
```

```
   field.Corner( this.corner );
④ return(this.Add(field));
  }
```

※「①②③④」以外の解釈は、読者への課題といたします。

・・・・・・・・・・・・・・・・・・・・・・・・・

①「TFieldDisplay　　*field=new TFieldDisplay();」の解説

機能；「field クラス」を「TFieldDisplay()クラス」から継承してダイナミックに作成

②「field.Create(this.chart_id, this.sub_window, _cols, _lines, _col, _row);」の解説

```
//------------------------------------------------------------------
//      Create object
//------------------------------------------------------------------
bool TFieldDisplay::Create(long _chart_id,int _window,int _cols,int _lines,int _col,int _row)
  {
       this.curr_column=_col;
       this.curr_row=_row;
  ②-1  SetParams(_chart_id,_window,_cols,_lines);
  ②-2
return(this.Create(this.chart_id,this.GetUniqName(),this.sub_window,(int)(_col*this.chart_width_
step),(int)(_row*this.chart_height_step)));
  }
```

・・・・・・・・・・・・

②-1　「SetParams(_chart_id,_window,_cols,_lines);」の解説

```
    //------------------------------------------------------------------
    //Set object parameters
    //------------------------------------------------------------------
    void TFieldDisplay::SetParams(long _chart_id,int _window,int _cols,int _lines)
      {
       this.chart_id=_chart_id;
       this.sub_window=_window;
       this.columns_number=_cols;  ※Ｘ軸方向の分割数
       this.limes_number=_lines;   ※Ｙ軸方向の分割数
    //get window width in pixels
      ※フルスクリーンの幅と高さをピクセル単位で取得
       this.chart_width=GetSystemMetrics(SM_CXFULLSCREEN);
       this.chart_height=GetSystemMetrics(SM_CYFULLSCREEN);
    //calculate steps
      ※分割した場合の升目「１セグメント」の幅と高さを取得
       this.chart_width_step=this.chart_width/_cols;
       this.chart_height_step=this.chart_height/_lines;
      }
```

②-2

「return

```
(this.Create(this.chart_id, this.GetUniqName(), this.sub_window,
            (int)(_col*this.chart_width_step), (int)(_row*this.chart_height_step)) );」の解説
```

引数；　５個ある（６個の引数がある「 ② Create()」とは異なる）

・(int)(_col*this.chart_width_step)、(int)(_row*this.chart_height_step)；

※「_col 行、_row 列」目の「セグメント」を指定している

機能；　指定する「セグメント」に「ラベル・オブジェクト」を１個配置する。

継承先；　「this.Create()」は＜ChartObjectsTxtControls.mqh＞から継承している

```
//+------------------------------------------------------------------+
//| Create object "Label"                                            |
//+------------------------------------------------------------------+
bool CChartObjectLabel::Create(long chart_id,const string name,const int window,
                                                          const int X,const int Y)
  {
   if(!ObjectCreate(chart_id,name,OBJ_LABEL,window,0,0.0))  ①-2-1；ラベルオブジェクト生成
      return(false);
   if(!Attach(chart_id,name,window,1))       ①-2-2；１個のアンカーでラベルを張り付ける
      return(false);
   if(!Description(name))
      return(false);
   if(!X_Distance(X) || !Y_Distance(Y))      ※ラベルを配置する座標の指定
      return(false);
   ・・・・・・・・・・・・・・・・・
```

②-2-1　「ObjectCreate(chart_id,name,OBJ_LABEL,window,0,0.0)」の解説

・ObjectCreate は MQL4(MQL5)の標準関数；

使用法１；　「time/price」座標の使用時

```
bool  ObjectCreate(
 long          chart_id,       // chart ID
 string        object_name,    // object name
 ENUM_OBJECT   object_type,    // object type
 int           sub_window,     // window index
 datetime      time1,          // time of the first anchor point
 double        price1,         // price of the first anchor point
 ...
 datetime      timeN=0,        // time of the N-th anchor point
 double        priceN=0        // price of the N-th anchor point
 );
```

使用法２；　「Ｘ／Ｙ」座標の使用時・・本節ではこちらを使用

```
bool  ObjectCreate(
 long          chart_id,       // chart ID
 string        object_name,    // object name
 ENUM_OBJECT   object_type,    // object type
 int           sub_window,     // window index
 int           x_ccordinate,
 int           y_ccordinate
 );
```

※本節の場合では「ObjectCreate(chart_id,name,OBJ_LABEL,window,0,0)」の様に、

(x,y) ＝ (0,0) と設定し、「ラベル・オブジェクト」を一旦生成後に、

「X_Distance(X 座標；ピクセル)、Y_Distance(Y 座標；ピクセル)で位置を再設定している。

②-2-2　「Attach(chart_id,name,window,1)」の解説

継承先；　「Attach()」は＜ChartObject.mqh＞から来ている

機能；生成したラベル・オブジェクトを「１個」のアンカーで指定位置に張り付ける。

```
protected:
   long              m_chart_id;          // identifier of chart the object belongs to
   int               m_window;            // number of subwindow (0 - main window)
   string            m_name;              // unique name object name
   int               m_num_points;        // number of anchor points of object
   ・・・・・・・・・・・・・
```

```
//+------------------------------------------------------------------+
//| Attach object                                                    |
//+------------------------------------------------------------------+
bool CChartObject::Attach(long chart_id,const string name,const int window,const int points)
  {
//--- check
   if(ObjectFind(chart_id,name)<0)
      return(false);
//--- attach
   if(chart_id==0)
      chart_id=ChartID();
   m_chart_id  =chart_id;
   m_window    =window;
   m_name      =name;
   m_num_points=points;
//--- successful
   return(true);
  }
```

③ 「field.Description("");」の解説

継承先；　「Description()」は＜ChartObject.mqh＞から来ている（継承する）

機能；　ラベルに「"";空白文字」を書き込みます

```
//+------------------------------------------------------------------+
//| Set comment of object                                            |
//+------------------------------------------------------------------+
bool CChartObject::Description(const string new_text) const
  {
//--- check
   if(m_chart_id==-1)
      return(false);
//--- tune
   if(new_text=="")
      return(ObjectSetString(m_chart_id,m_name,OBJPROP_TEXT," "));
//--- result
   return(ObjectSetString(m_chart_id,m_name,OBJPROP_TEXT,new_text));
  }
```

※「ObjectSetString()」は標準関数、③ではテキスト・オブジェクトを張り付けている。

```
bool  ObjectSetString(
   long    chart_id,          // chart identifier
   string  name,              // object name
   int     prop_id,           // property
   string  prop_value         // value
   );
```

④ 「return(this.Add(field));」の解説

継承先；　「Add(field)」は＜List.mqh＞から来ている（継承する）

機能；　「線形リスト」に新規「ノード」として継ぎ足します

```
//+------------------------------------------------------------------+
//| Adding a new element to the end of the list                      |
//+------------------------------------------------------------------+
int CList::Add(CObject *new_node)
  {
//--- check
   if(!CheckPointer(new_node))
      return(-1);
```

```
//--- add
   if(m_first_node==NULL)
      m_first_node=new_node;
   else
     {
      m_last_node.Next(new_node);
      new_node.Prev(m_last_node);
     }
   m_curr_node=new_node;
   m_curr_idx=m_data_total;
   m_last_node=new_node;
   m_data_sort=false;
//--- result
   return(m_data_total++);
```

その他（一部クラス）の補足解説；

「SetText()」の解説

```
//------------------------------------------------------------------------
//        Set text
//------------------------------------------------------------------------
bool TableDisplay::SetText(int _index,string _text)
  {
   CChartObjectText   *object=GetNodeAtIndex(_index);  A
   if(object==NULL)
     {
      return(false);
     }
   return(object.Description(_text));     B
  }
```

A部；

継承先；「GetNodeAtIndex(_index)」は＜List.mqh＞から来ている（継承する）

機能；インデックスから「ノード番号」を得て、その「ノード」内のデータとして、
テキスト・データを書き込む。

B部；

「③の解説」と同じ

（４）使用コード（PriceList_03.mq4）をブロック単位で把握する

※コードの理解に役立ちそうな「コメント」を入れておきます。

```
//+------------------------------------------------------------------+
//|                                                PriceList_03.mq4 |
//|                                                          amenbo |
//|                                                泉の森の弁財天池 |
//+------------------------------------------------------------------+
#property copyright "amenbo"
#property link      "泉の森の弁財天池"
#property version   "1.00"
#property strict
#property indicator_chart_window
//
// ------------ Custom indicator -------------
//      Include libraries
#include        <TextDisplay.mqh>
//      External input parameters
input int   Y_=2;//Ｙ方向のオフセット
input int   X_=1;//Ｘ方向のオフセット
//-----------------------------------------
TableDisplay      Table1;
//-----------------------------------------
#define NUMBER  8
//-----------------------------------------
string  names[NUMBER]={ "EURUSDFXF","GBPUSDFXF","AUDUSDFXF","NZDUSDFXF","USDCHFFXF",
                        "USDCADFXF","USDJPYFXF","EURJPYFXF" };
int      coord_y[ NUMBER ] = { 0, 1, 2, 3, 4, 5, 6, 7 };
//-----------------------------------------
double     rates[NUMBER];
datetime   times[NUMBER];
MqlTick    tick;
//-----------------------------------------
int OnInit()
  {
   ArrayInitialize(times,0);
   ArrayInitialize(rates,0);
   //   ----Create table----
   //   Set coordinates ------
    ※座標系を設定します「左上が原点（0,0)」
   Table1.SetParams(0,0,CORNER_LEFT_UPPER);
```

	インデックス No	ノード No
為替ペア名	0	1
		2
為替ペア名	7	8
BID 値	8	9
		10
BID 値	15	16
スプレッド	16	17
		18
スプレッド	23	24

処理内容；
１．２４個（８×３）のデータを「線形リスト」の「ノード」として、次々に繋げていく。
２．各「ノード」の表示位置（テーブル内の）を同時に設定しています。

```
   //   Show headers ------
   for(int i1=0; i1<NUMBER; i1++)
     {
         Table1.AddTitleObject(40,40,X_,Y_+coord_y[i1],names[i1]+" :",Blue);※為替ペア（８個）
     }
   //   Show prices ------

   for(int i2=0; i2<NUMBER; i2++)
     {
         Table1.AddFieldObject(40,40,X_+2,Y_+coord_y[i2],Blue);          ※BID データ（８個）
     }
   //   Show spreads ------
   for(int i3=0; i3<NUMBER; i3++)
     {
         Table1.AddFieldObject(40,40,X_+4,Y_+coord_y[i3],Blue);          ※スプレッド（８個）
     }
```

```
   //
   RefreshInfo();
   ChartRedraw(0);
   EventSetTimer(1);  ※タイマーを「１秒」にセット
   //
   return(0);
  }
```

```
//  Custom indicator iteration function ---------------
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   //--- return value of prev_calculated for next call
   return(rates_total);
  }
```

インディケータなので、
形式的に置いてあるだけ！
（何もしない）

```
//      OnTimer event handler --------------------
void OnTimer()
  {
   RefreshInfo();
   ChartRedraw(0);
  }
```

「１秒」ごとに、
BID 値とスプレッドを最新内容に
書換える

```
//      OnDeinit event handler --------------------
void OnDeinit(const int _reason)
  {
   EventKillTimer();
//      Delete table
   Table1.Clear();
  }
```

```
//      Refresh info ------------------------------
void RefreshInfo()
  {


   for(int i4=0; i4<NUMBER; i4++)
     {
      //        Get price data
    bool tick_=SymbolInfoTick(names[i4],tick);//「MqlTick tick」に最新データをセットする処理
      //
      if(tick.time>times[i4] || times[i4]==0)
        {

Table1.SetText(i4+NUMBER,DoubleToString(tick.bid,(int)(SymbolInfoInteger(names[i4],SYMBOL_D
IGITS))));

         if(tick.bid>rates[i4] && rates[i4]>0.1)
           {
            Table1.SetColor(i4+NUMBER,Lime);
           }
         else if(tick.bid<rates[i4] && rates[i4]>0.1)
           {
            Table1.SetColor(i4+NUMBER,Red);
           }
         else
           {
            Table1.SetColor(i4+NUMBER,Blue);
           }
       //
         rates[ i4 ] = tick.bid;
         times[ i4 ] = tick.time;

        }


Table1.SetText(i4+NUMBER+NUMBER,DoubleToString((tick.ask-tick.bid)/SymbolInfoDouble(names[i
4],SYMBOL_POINT),0));

     }


  }
//+------------------------------------------------------------------+
```

・直接に MQL4 形式の「Bid」を使って記述して良かったが、
MQL5 の資料からもってきたので、
MqlTick tick を使った。

・Bid 値を入れた「ノード」の番号は「９」から始まり、
インデックス No では「８」から始まることに注意。
・従って「Table1.SetText(i4+NUMBER,・・」となる。

・スプレッド値が格納された「ノード」の
インデックス No は「１６」から始まる

以　上