```
//+------------------------------------------------------------------+
//|                                                C_Chart_Drawing.mqh |
//|                                                          M Wilson |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "M Wilson"
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
class C_Chart_Drawing
  {

public:
   //Constructor and Destructor
  C_Chart_Drawing();
  ~C_Chart_Drawing();
  //Public functions
   void RefreshChart(int intSleep = 100);
   bool CreateRectangle(const long chart_ID=0,const string name="Rectangle",const int
sub_window=0, datetime time1=0,double price1=0, datetime time2=0,double price2=0, const
color clr=clrRed,const ENUM_LINE_STYLE style=STYLE_SOLID,const int width=1, const bool
fill=false,const bool back=false,const bool selection=true, const bool hidden=true,const
 long z_order=0);
   bool DeleteRectangle(const long   chart_ID=0,const string name="Rectangle");
   bool CreateRectangleLabel(const long chart_ID=0,const string name="RectLabel",const
int sub_window=0,const int x=0,const int y=0,const int width=50,const int height=18,
const color back_clr=C'236,233,216', const ENUM_BORDER_TYPE border=BORDER_SUNKEN,const
ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER,const color clr=clrRed,const ENUM_LINE_STYLE
style=STYLE_SOLID,const int line_width=1,const bool back=false,const bool selection=
false,const bool hidden=true,const long z_order=0);
   bool DeleteRectangleLable(const long chart_ID=0,const string name="RectLabel");
   bool CreateLabel(const long chart_ID=0,const string name="Label",const int sub_window
=0,const int x=0,const int y=0,const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER,const
string text="Label", const string font="Arial",const int font_size=10, const color clr=
clrRed,const double angle=0.0, const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER,const
bool back=false, const bool selection=false,const bool hidden=true,const long z_order=0
);
   bool DeleteLabel(const long chart_ID=0,const string name="Label");
   bool ChangeLabelText(const long chart_ID=0,const string name="Label",const string
text="Text");
   bool CreateVerticalLine(const long chart_ID=0,const string name="VLine",const int
sub_window=0,datetime time=0,const color clr=clrRed,const ENUM_LINE_STYLE style=
STYLE_SOLID,const int width=1,const bool back=false,const bool selection=true,const bool
 hidden=false,const long z_order=0);
   bool DeleteLine(const long   chart_ID=0, const string name="VLine_TLine", bool
boolReportError = True);
   bool DeleteAllVerticleLines(const long chart_ID=0, bool boolReportError = True);
   bool CreateHorizontalLine(const long chart_ID=0, const string name="HLine", const int
 sub_window=0,double price=0,const color clr=clrRed, const ENUM_LINE_STYLE style=
STYLE_SOLID, const int width=1,const bool back=false,const bool selection=true, const
bool hidden=false, const long z_order=0);
   bool DeleteAllHorizontalLines(const long chart_ID=0, bool boolReportError = True);
   bool CreateTrendLine(const long chart_ID=0,const string name="TrendLine", const int
sub_window=0,datetime time1=0,double price1=0, datetime time2=0,double price2=0,const
color clr=clrRed,const ENUM_LINE_STYLE style=STYLE_SOLID,const int width=1,const bool
back=false,const bool selection=true,const bool ray_right=false,const bool hidden=false,
const long z_order=0);
   bool DeleteTrendLine(const long chart_ID=0, const string name="TrendLine", bool
boolReportError = True);
   bool DeleteAllTrendLines(const long chart_ID=0, bool boolReportError = True);
   bool CreateText(const long chart_ID=0, const string name="Text",const int sub_window=
0,const datetime atTime=0, const double atValue=0, const double atAngle=90, const bool
anchorAbove=True, const string strTextValue="?", const color clr=clrWhite);
   bool DeleteText(const long   chart_ID=0, const string name="Text", bool
boolReportError = True);
   bool DeleteAllText(const long chart_ID=0, bool boolReportError = True);
   bool CreateUpArrow(const long chart_ID=0, const string name="ArrowUp", const int
sub_window=0,datetime time=0,double price=0, const ENUM_ARROW_ANCHOR anchor=
ANCHOR_BOTTOM,const color clr=clrRed, const ENUM_LINE_STYLE   style=STYLE_SOLID, const
int width=3,const bool back=false, const bool selection=true, const bool hidden=False,
const long z_order=0);
```

```
   bool CreateDownArrow(const long chart_ID=0, const string name="ArrowDown", const int
sub_window=0,datetime time=0,double price=0, const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP,
const color clr=clrRed, const ENUM_LINE_STYLE   style=STYLE_SOLID, const int width=3,
const bool back=false, const bool selection=true, const bool hidden=False, const long
z_order=0);
   bool DeleteAllArrows(const long chart_ID=0, bool boolReportError = True);
   bool DeleteArrow(const long chart_ID=0, const string name="Arrow", bool
boolReportError = True);
   bool CreateFibonacci(const long chart_ID=0, const string name="Fibo",const int
sub_window=0,datetime time1=0,double price1=0,datetime time2=0, double price2=0,const
color clr=clrRed, const ENUM_LINE_STYLE style=STYLE_SOLID, const int width=1,const bool
back=True, const bool selection=true,const bool ray_right=false,const bool hidden=False,
const long z_order=0);
   bool DeleteFibonacci(const long chart_ID=0,const string name="Fib");
   bool DeleteAllFibonacci(const long   chart_ID=0, bool boolReportError = True);
   bool SetLevelsFibonacci(double &values[], color &colors[], ENUM_LINE_STYLE &styles[],
 int &widths[], const bool boolIncLevelText=True, const long chart_ID=0,const string
name="FiboLevels");
   bool CreateTimeFibonacci(const long chart_ID=0,const string name="Fibo",const int
sub_window=0,datetime time1=0,double price1=0,datetime time2=0,double price2=0,const
color clr=clrRed,const ENUM_LINE_STYLE style=STYLE_SOLID,const int width=1,const bool
back=True,const bool selection=true,const bool hidden=False,const long z_order=0);

   bool DeleteTimeFibonacci(const long chart_ID=0,const string name="Fib");
   bool DeleteAllTimeFibonacci(const long   chart_ID=0, bool boolReportError = True);
   bool SetLevelsTimeFibonacci(double &values[], color &colors[], ENUM_LINE_STYLE &
styles[], int &widths[], const long chart_ID=0,const string name="FiboLevels");
private:
   //Private functions
   void InitiateTrendEmptyPoints(datetime &time1,double &price1, datetime &time2,double
&price2);
   void InitiateRectangleEmptyPoints(datetime &time1,double &price1,datetime &time2,
double &price2);
   void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,datetime &time2,
double &price2);
   void ChangeTimeFiboLevelsEmptyPoints(datetime &time1,double &price1,datetime &time2,
double &price2);
   void ChangeArrowEmptyPoint(datetime &time,double &price);
  };
//+------------------------------------------------------------------+
//|  Constructor                                                     |
//+------------------------------------------------------------------+
C_Chart_Drawing::C_Chart_Drawing()
  {
  }
//+------------------------------------------------------------------+
//|  Destructor                                                      |
//+------------------------------------------------------------------+
C_Chart_Drawing::~C_Chart_Drawing()
  {
  }
//+------------------------------------------------------------------+
//+------------------------------------------------------------------+
//|  Public Functions                                                |
//+------------------------------------------------------------------+
void C_Chart_Drawing::RefreshChart(int intSleep = 100)
{
      ChartRedraw();
      Sleep(intSleep);
}
//|--Rectangle ------------------------------------------------------|
bool C_Chart_Drawing::CreateRectangle(const long            chart_ID=0,
// chart's ID
                      const string          name="Rectangle",  // rectangle name
                      const int             sub_window=0,      // subwindow index
                      datetime              time1=0,           // first point time
                      double                price1=0,          // first point price
                      datetime              time2=0,           // second point time
                      double                price2=0,          // second point price
                      const color           clr=clrRed,        // rectangle color
                      const ENUM_LINE_STYLE style=STYLE_SOLID,
// style of rectangle lines
```

```
                     const int             width=1,
// width of rectangle lines
                     const bool            fill=false,
// filling rectangle with color
                     const bool            back=false,        // in the background
                     const bool            selection=true,    // highlight to move
                     const bool            hidden=true,
// hidden in the object list
                     const long            z_order=0)
// priority for mouse click
  {
   //--- set anchor points' coordinates if they are not set
   InitiateRectangleEmptyPoints(time1,price1,time2,price2);
   //--- reset the error value
   ResetLastError();
   //--- create a rectangle by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
   {
    Print(__FUNCTION__,": failed to create a rectangle! Error code = ",GetLastError());
    return(false);
   }
   //--- set rectangle color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   //--- set the style of rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   //--- set width of the rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
   //--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- enable (true) or disable (false) the mode of highlighting the rectangle for moving

//--- when creating a graphical object using ObjectCreate function, the object cannot be
   //--- highlighted and moved by default. Inside this method, selection parameter
   //--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   //--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   //--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
   //--- successful execution
   return(true);
}
bool C_Chart_Drawing::DeleteRectangle(const long   chart_ID=0,       // chart's ID
                     const string name="Rectangle") // rectangle name
{
   //--- reset the error value
   ResetLastError();

   //--- delete rectangle
   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         Print(__FUNCTION__,": failed to delete rectangle! Error code = ",GetLastError
());
         return(false);
      }
   }

   //--- successful execution
   return(true);
}
//+------------------------------------------------------------------+
//| Check the values of rectangle's anchor points and set default    |
//| values for empty ones                                            |
//+------------------------------------------------------------------+

//|--Rectangle Labels-------------------------------------------------|
```

```
bool C_Chart_Drawing::CreateRectangleLabel(const long     chart_ID=0,
// chart's ID
                                   const string           name="RectLabel",
// label name
                                   const int              sub_window=0,
// subwindow index
                                   const int              x=0,
// X coordinate
                                   const int              y=0,
// Y coordinate
                                   const int              width=50,
// width
                                   const int              height=18,
// height
                                   const color            back_clr=C'236,233,216',
// background color
                                   const ENUM_BORDER_TYPE border=BORDER_SUNKEN,
// border type
                                   const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER,
// chart corner for anchoring
                                   const color            clr=clrRed,
// flat border color (Flat)
                                   const ENUM_LINE_STYLE  style=STYLE_SOLID,
// flat border style
                                   const int              line_width=1,
// flat border width
                                   const bool             back=false,
// in the background
                                   const bool             selection=false,
// highlight to move
                                   const bool             hidden=true,
// hidden in the object list
                                   const long             z_order=0)
// priority for mouse click
{

   //Reset Last Error
   ResetLastError();

   //Create Rectangle Label
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
   {
      Print(__FUNCTION__, ": failed to create a rectangle label! Error code = ",
GetLastError());
      return(false);
   }

   //Set X/Y Co-ordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
   //Set Width/Height
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
   //Background Color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
   //Border Type
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
   //set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
   //set flat border color (in Flat mode)
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   //set flat border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   //set flat border width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
   //display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
   //enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   //hide (true) or display (false) graphical object name in the object list
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   //set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);
}

bool C_Chart_Drawing::DeleteRectangleLable(const long   chart_ID=0,       // chart's ID
                     const string name="RectLabel") // label name
{
   //reset the error value
   ResetLastError();

   //delete the label
   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         Print(__FUNCTION__,": failed to delete a rectangle label! Error code = ",
GetLastError());
         return(false);
      }
   }

   return(true);
}
//|---Labels-----------------------------------------------------------|
bool C_Chart_Drawing::CreateLabel(const long              chart_ID=0,
// chart's ID
                                  const string            name="Label",
// label name
                                  const int               sub_window=0,
// subwindow index
                                  const int               x=0,
// X coordinate
                                  const int               y=0,
// Y coordinate
                                  const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER,
// chart corner for anchoring
                                  const string            text="Label",             // text
                                  const string            font="Arial",             // font
                                  const int               font_size=10,
// font size
                                  const color             clr=clrRed,
// color
                                  const double            angle=0.0,
// text slope
                                  const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER,
// anchor type
                                  const bool              back=false,
// in the background
                                  const bool              selection=false,
// highlight to move
                                  const bool              hidden=true,
// hidden in the object list
                                  const long              z_order=0)
// priority for mouse click
{
   //reset the error value
   ResetLastError();

   //create a text label
   if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
   {
      Print(__FUNCTION__, ": failed to create text label! Error code = ",GetLastError
());
      return(false);
    }
   //set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
```

```
   //set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
   //set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
   //set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
   //set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
   //set the slope angle of the text
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
   //set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
   //set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   //display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
   //enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   //hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   //set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);
}
bool C_Chart_Drawing::DeleteLabel(const long   chart_ID=0,   // chart's ID
                  const string name="Label") // label name
  {
   //reset the error value
   ResetLastError();

   //delete the label
   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         Print(__FUNCTION__,": failed to delete a text label! Error code = ",
GetLastError());
         return(false);
      }
   }

   return(true);
  }
bool C_Chart_Drawing::ChangeLabelText(const long   chart_ID=0,   // chart's ID
                    const string name="Label", // object name
                    const string text="Text")  // text
  {
   //reset the error value
   ResetLastError();

   //change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,": failed to change the text! Error code = ",GetLastError());
      return(false);
     }

   //successful execution
   return(true);
  }
//|---Vertical Lines-------------------------------------------------|
bool C_Chart_Drawing::CreateVerticalLine(const long     chart_ID=0,        // chart's ID
                  const string          name="VLine",      // line name
                  const int             sub_window=0,      // subwindow index
                  datetime              time=0,            // line time
                  const color           clr=clrRed,        // line color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                  const int             width=1,           // line width
                  const bool            back=false,        // in the background
```

```
                  const bool            selection=true,    // highlight to move
                  const bool            hidden=false,       // hidden in the object list
                  const long            z_order=0)         // priority for mouse click
{

   if(!time)   time=TimeCurrent();

   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
   {
      Print(__FUNCTION__, ": failed to create a vertical line! Error code = ",
GetLastError());
      return(false);
   }

   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
}
bool C_Chart_Drawing::DeleteLine(const long   chart_ID=0, const string name=
"VLine_TLine", bool boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         if(boolReportError)  Print(__FUNCTION__, ": failed to delete the line: ",name,
" Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);
}

bool C_Chart_Drawing::DeleteAllVerticleLines(const long chart_ID=0, bool boolReportError
 = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_VLINE)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY,OBJ_VLINE))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all verticle lines from chart, Error code = ",GetLastError());
         return(false);
      }
   }
```

```
   return(true);
}
//|---Horizontal Lines-----------------------------------------------|
bool C_Chart_Drawing::CreateHorizontalLine(const long     chart_ID=0,
// chart's ID
                  const string          name="HLine",      // line name
                  const int             sub_window=0,      // subwindow index
                  double                price=0,            // line price
                  const color           clr=clrRed,        // line color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                  const int             width=1,           // line width
                  const bool            back=false,        // in the background
                  const bool            selection=true,    // highlight to move
                  const bool            hidden=false,       // hidden in the object list
                  const long            z_order=0)         // priority for mouse click
{

   if(!price) price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
   {
      Print(__FUNCTION__, ": failed to create a horizontal line! Error code = ",
GetLastError());
      return(false);
   }

   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
}
bool C_Chart_Drawing::DeleteAllHorizontalLines(const long chart_ID=0, bool
boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_HLINE)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY,OBJ_HLINE))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all horizontal lines from chart, Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);
}
//|---Trend Lines----------------------------------------------------|
bool C_Chart_Drawing::CreateTrendLine(const long            chart_ID=0,
// chart's ID
                     const string          name="TrendLine",  // line name
                     const int             sub_window=0,      // subwindow index
```

```
                     datetime              time1=0,           // first point time
                     double                price1=0,          // first point price
                     datetime              time2=0,           // second point time
                     double                price2=0,          // second point price
                     const color           clr=clrRed,        // line color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                     const int             width=1,           // line width
                     const bool            back=false,        // in the background
                     const bool            selection=true,    // highlight to move
                     const bool            ray_right=false,
// line's continuation to the right
                     const bool            hidden=false,
// hidden in the object list
                     const long            z_order=0)
// priority for mouse click
{
//--- set anchor points' coordinates if they are not set
   InitiateTrendEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a trend line by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__, ": failed to create a trend line! Error code = ",GetLastError
());
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- enable (true) or disable (false) the mode of continuation of the line's display to the
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
}
bool C_Chart_Drawing::DeleteTrendLine(const long chart_ID=0, const string name=
"TrendLine", bool boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectFind(chart_ID,name)>=0)
   {
     if(!ObjectDelete(chart_ID,name))
     {
         if(boolReportError)  Print(__FUNCTION__, ": failed to delete text: ",name,
" Error code = ",GetLastError());
         return(false);
     }
   }

   return(true);
}
bool C_Chart_Drawing::DeleteAllTrendLines(const long chart_ID=0, bool boolReportError =
True)
```

```
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_TREND)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY,OBJ_TREND))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all trend lines from chart, Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);
}
//|---Text-------------------------------------------------------|
bool C_Chart_Drawing::CreateText(const long     chart_ID=0,         // chart's ID
                  const string          name="Text",       // text name
                  const int             sub_window=0,       // subwindow index
                  const datetime        atTime=0,           // text time
                  const double          atValue=0,
// Value of the text (how high it is)
                  const double          atAngle=90,         // Angle of Text
                  const bool            anchorAbove=True,
                  const string          strTextValue="?",   // Value of text displayed
                  const color           clr=clrWhite,       // line color
                   )          // priority for mouse click
{

   ResetLastError();

   ObjectCreate(name, OBJ_TEXT, 0, 0, 0, 0, 0);
   ObjectSetText(name, strTextValue,8.0,"Arial");
   if(anchorAbove)
   {
      ObjectSet(name,OBJPROP_ANCHOR,ANCHOR_LEFT);
   }
   else
   {
      ObjectSet(name,OBJPROP_ANCHOR,ANCHOR_RIGHT);
   }
   ObjectSet(name,OBJPROP_COLOR,clr);
   ObjectSet(name, OBJPROP_TIME1, atTime);
   ObjectSet(name, OBJPROP_PRICE1, atValue);
   ObjectSet(name,OBJPROP_ANGLE,atAngle);
   ObjectSet(name,OBJPROP_FONTSIZE,8.0);

//--- successful execution
   return(true);
}

bool C_Chart_Drawing::DeleteText(const long   chart_ID=0, const string name="Text", bool
 boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectFind(chart_ID,name)>=0)
   {
     if(!ObjectDelete(chart_ID,name))
     {
         if(boolReportError)  Print(__FUNCTION__, ": failed to delete text: ",name,
" Error code = ",GetLastError());
         return(false);
     }
   }

   return(true);
}
```

```
bool C_Chart_Drawing::DeleteAllText(const long chart_ID=0, bool boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_TEXT)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY, OBJ_TEXT))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all text from chart, Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);
}
//|---Arrow-----------------------------------------------------|
bool C_Chart_Drawing::CreateUpArrow(const long              chart_ID=0,
// chart's ID
                    const string            name="ArrowUp",        // sign name
                    const int               sub_window=0,          // subwindow index
                    datetime                time=0,                // anchor point time
                    double                  price=0,               // anchor point price
                    const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                    const color             clr=clrRed,            // sign color
                    const ENUM_LINE_STYLE   style=STYLE_SOLID,     // border line style
                    const int               width=3,               // sign size
                    const bool              back=false,            // in the background
                    const bool              selection=true,        // highlight to move
                    const bool              hidden=False,
// hidden in the object list
                    const long              z_order=0)
// priority for mouse click
{

   ChangeArrowEmptyPoint(time,price);

   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
   {
      Print(__FUNCTION__,": failed to create \"Arrow Up\" sign! Error code = ",
GetLastError());
      return(false);
  }

   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);
}
bool C_Chart_Drawing::CreateDownArrow(const long              chart_ID=0,
// chart's ID
                    const string            name="ArrowDown",        // sign name
                    const int               sub_window=0,          // subwindow index
                    datetime                time=0,                // anchor point time
                    double                  price=0,               // anchor point price
                    const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP, // anchor type
                    const color             clr=clrRed,            // sign color
```

```
                  const ENUM_LINE_STYLE   style=STYLE_SOLID,     // border line style
                  const int               width=3,               // sign size
                  const bool              back=false,            // in the background
                  const bool              selection=true,        // highlight to move
                  const bool              hidden=False,
// hidden in the object list
                  const long              z_order=0)
// priority for mouse click
{

   ChangeArrowEmptyPoint(time,price);

   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
   {
      Print(__FUNCTION__,": failed to create \"Arrow Down\" sign! Error code = ",
GetLastError());
      return(false);
  }

   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);
}
bool C_Chart_Drawing::DeleteArrow(const long chart_ID=0, const string name="Arrow", bool
 boolReportError = True)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectFind(chart_ID,name)>=0)
   {
     if(!ObjectDelete(chart_ID,name))
     {
         if(boolReportError)  Print(__FUNCTION__, ": failed to delete arrow: ",name,
" Error code = ",GetLastError());
         return(false);
     }
   }

   return(true);
}

bool C_Chart_Drawing::DeleteAllArrows(const long chart_ID=0, bool boolReportError = True
)
{
   //Delete's any kind of line, you just have to give it the right name
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_ARROW)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY,OBJ_ARROW))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all arrows from chart, Error code = ",GetLastError());
         return(false);
      }
   }
```

```
   return(true);
}
//|---Fibonacci-------------------------------------------------------|
bool C_Chart_Drawing::CreateFibonacci(const long            chart_ID=0,
// chart's ID
                      const string          name="Fibo",     // object name
                      const int             sub_window=0,      // subwindow index
                      datetime              time1=0,           // first point time
                      double                price1=0,          // first point price
                      datetime              time2=0,           // second point time
                      double                price2=0,          // second point price
                      const color           clr=clrRed,        // object color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                      const int             width=1,           // object line width
                      const bool            back=True,        // in the background
                      const bool            selection=true,    // highlight to move
                      const bool            ray_right=false,
// object's continuation to the right
                      const bool            hidden=False,
// hidden in the object list
                      const long            z_order=0)
// priority for mouse click
  {

   ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);


   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Retracement\"! Error code = ",GetLastError
());
      return(false);
     }

   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,clr);
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,style);
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);
  }
bool C_Chart_Drawing::DeleteFibonacci(const long chart_ID=0,const string name="Fib")
{
   //reset the error value
   ResetLastError();

   //delete the label
   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         Print(__FUNCTION__,": failed to delete a Fibonacci! Error code = ",GetLastError
());
         return(false);
      }
```

```
   }

   return(true);
}
bool C_Chart_Drawing::DeleteAllFibonacci(const long   chart_ID=0, bool boolReportError =
 True) // object name
  {
//--- reset the error value
   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_FIBO)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY, OBJ_FIBO))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all Fibo's from chart, Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);

  }
bool C_Chart_Drawing::SetLevelsFibonacci(double &values[], color &colors[],
ENUM_LINE_STYLE &styles[], int &widths[], const bool boolIncLevelText=True, const long
chart_ID=0,const string name="FiboLevels")
{
   //--- check array sizes
   int intSize = ArraySize(values);
   if(intSize!=ArraySize(colors) || intSize!=ArraySize(styles) || intSize!=ArraySize(
widths))
   {
      Print(__FUNCTION__,
": array size must be same for values, colors, styles does not correspond to the number of l
);
      return(false);
   }
   //--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,intSize);

   //--- set the properties of levels in the loop
   for(int i=0;i<intSize;i++)
   {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      if(boolIncLevelText) ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"%$ ["+
DoubleToString(100*values[i],1)+"]");
   }

   return(true);
}
bool C_Chart_Drawing::CreateTimeFibonacci(const long            chart_ID=0,
// chart's ID
                        const string          name="Fibo",     // object name
                        const int             sub_window=0,       // subwindow index
                        datetime              time1=0,            // first point time
                        double                price1=0,           // first point price
                        datetime              time2=0,            // second point time
                        double                price2=0,           // second point price
                        const color           clr=clrRed,         // object color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                        const int             width=1,            // object line width
                        const bool            back=True,         // in the background
                        const bool            selection=true,     // highlight to move
```

```
                     const bool            hidden=False,
// hidden in the object list
                     const long            z_order=0)
// priority for mouse click
{

   this.ChangeTimeFiboLevelsEmptyPoints(time1,price1,time2,price2);

   ResetLastError();

   if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,time2,price2))
   {
      Print(__FUNCTION__,": failed to create \"Fibonacci Time Zones\"! Error code = ",
GetLastError());
      return(false);
   }

   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot be
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

   return(true);

}
bool C_Chart_Drawing::DeleteTimeFibonacci(const long chart_ID=0,const string name="Fib")

{
   //reset the error value
   ResetLastError();

   //delete the label
   if(ObjectFind(chart_ID,name)>=0)
   {
      if(!ObjectDelete(chart_ID,name))
      {
         Print(__FUNCTION__,": failed to delete a time fib! Error code = ",GetLastError
());
         return(false);
      }
   }

   return(true);
}
bool C_Chart_Drawing::DeleteAllTimeFibonacci(const long   chart_ID=0, bool
boolReportError = True) // object name
{

   ResetLastError();

   if(ObjectsTotal(chart_ID,-1,OBJ_FIBOTIMES)>0)
   {
      if(!ObjectsDeleteAll(chart_ID,EMPTY, OBJ_FIBOTIMES))
      {
         if(boolReportError)  Print(__FUNCTION__,
": failed to delete all Time Fibo's from chart, Error code = ",GetLastError());
         return(false);
      }
   }

   return(true);

}
```

```
bool C_Chart_Drawing::SetLevelsTimeFibonacci(double &values[], color &colors[],
ENUM_LINE_STYLE &styles[], int &widths[], const long chart_ID=0,const string name=
"FiboLevels")
{
   //--- check array sizes
   int intSize = ArraySize(values);
   if(intSize!=ArraySize(colors) || intSize!=ArraySize(styles) || intSize!=ArraySize(
widths))
   {
      Print(__FUNCTION__,
": array size must be same for values, colors, styles does not correspond to the number of l
);
      return(false);
   }
   //--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,intSize);

   //--- set the properties of levels in the loop
   for(int i=0;i<intSize;i++)
   {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
));
   }

   return(true);
}
//+------------------------------------------------------------------+
//|  Private Functions                                               |
//+------------------------------------------------------------------+
//+------------------------------------------------------------------+
//| Check the values of trend line's anchor points and set default   |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void C_Chart_Drawing::InitiateTrendEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)  time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1) price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- if the second point's time is not set, it is located 9 bars left from the second one
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
}

void C_Chart_Drawing::InitiateRectangleEmptyPoints(datetime &time1,double &price1,
                                   datetime &time2,double &price2)
{
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
```

```
//--- if the second point's time is not set, it is located 9 bars left from the second one
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }

//--- if the second point's price is not set, move it 300 points lower than the first one
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }

void C_Chart_Drawing::ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- if the first point's time is not set, it is located 9 bars left from the second one
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the second one
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
void C_Chart_Drawing::ChangeTimeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- if the second point's time is not set, it is located 2 bars left from the second one
   if(!time2)
     {
      //--- array for receiving the open time of the last 3 bars
      datetime temp[3];
      CopyTime(Symbol(),Period(),time1,3,temp);
      //--- set the first point 2 bars left from the second one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
  }
void C_Chart_Drawing::ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
```