SNS TABLE OBJECT

SNS MT4 DEVELOPER TOOLS SERIES

Version 1.0 11/30/2009

By Steven N. Scott
(snscott @ www.kreslik.com forum)
Steven.n.scott@gmail.com

SNS TABLE OBJECT

This MT4 library file is ***not*** a "stand alone" indicator.

It is an "***importable***" MQL4 **LIBRARY** that serves as a utility for developers to allow them to create "data table" or "gauge" indicators more easily. By itself, it does nothing. However, MQL4 programmers can use it to quickly add a data table structure to their indicators without having to manually hard-code the creation of all the individual LABEL objects that usually go into the creation of such a feature.

As a developer, you have complete control over the ROW and COLUMN coordinates of your data table's individual cells. You can specify the font, font size, and font color for each cell and can also fine-adjust the positioning of text within a given cell using ROW and COLUMN offset values (this is most useful when you have text of different sizes on the same row and would like for the text in the cells to align more aesthetically up and down within the row).

OBJECT ORIENTED PROGRAMMING

While MQL4 is ***not*** an "object oriented" programming language like JAVA or C#, I have tried to maintain the spirit of OOP in how I created the API for this utility/tool.

The convention I have ***tried*** to consistently follow, in an attempt to make my MQL4 code "feel" more like an object oriented language, is to use naming for variables and functions like this:

```
<object name>.<variable or function>
```

For example:

```
Table.SetText(int row, int col, string text);
```

MQL4 INCLUDE HEADER FILES

MQL4 allows you to use the #INCLUDE directive to insert other files into your code, which is quite useful for encapsulating the #IMPORT code necessary for referencing functions in LIBRARY files.

NOTE

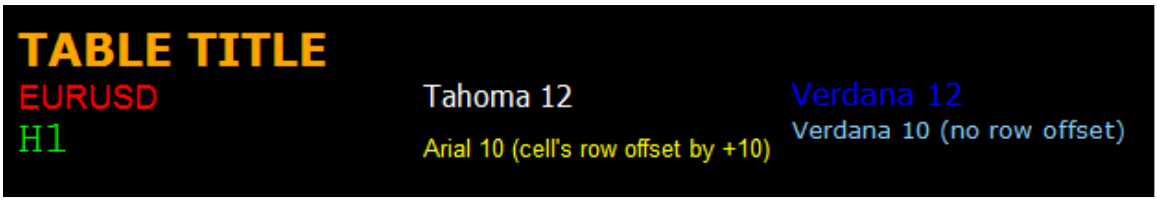
To use the examples or the SNS_TABLE_OBJECT in your own indicators, you need to place the SNS_TABLE_OBJEXT.ex4 file in your MetaTrader EXPERTS\LIBRARIES folder.

AND you need the SNS_OBJECTLIBRARY.EX4 file in your libraries folder as well.

Because of an inconsistent quirk in how MQL4 searches for #INCLUDED files, you will probably need to keep a copy of these MQH files in ***both*** your **LIBRARIES** folder ***and*** your **INDICATORS** folder.

EXAMPLES

The example table created in **SNS_#TABLE** is ***very*** simple and looks like this:



SNS_#TABLE does not do anything really useful and is meant ***only*** as a means of easily demonstrating how to set up and use a table object in your own indicator.

SNS_#TABLE

This is a simple "test bed" indicator that uses the SNS_TABLE_OBJECT to create a (meaningless) data table containing three rows with three columns in each row. The first row, or row 0, is used as a chart header or title area and then rows 1 and 2 are used to display "data".

The code in **SNS_#TABLE** shows how to set up a table object and how to expose whatever settings you would like to end users. For example, you could expose font settings to your users. Then it shows how to put data into the table for display.

Putting data into the table is as simple as:

```
Table.SetText(1,1,"Some data);
```

or

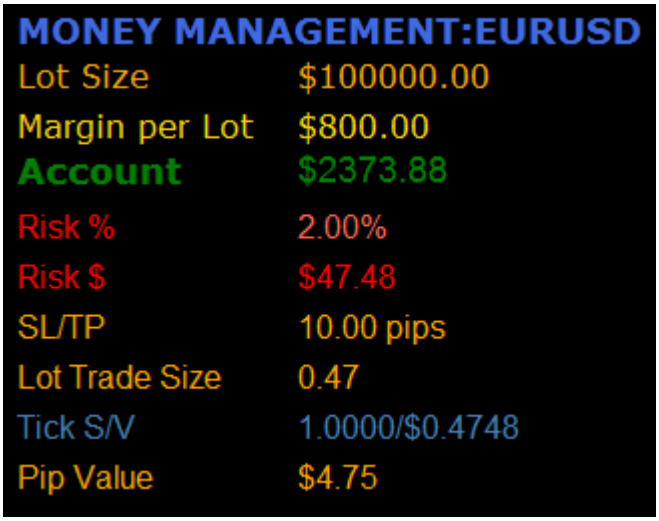
```
Table.SetTextOverride(1,2,"More Stuff","Arial",12,CLR_NONE);
```

Where the first two parameters to **SetText** are the ROW/COLUMN (Y/X) coordinates of the cell, the third parameter is the data or text to display in that cell. With **SetTextOverride** there are three additional parameters allowing you to override the font, size, and color for that cell (or leave any of them as the default as in the example above using the MQL4 predefined constant **CLR_NONE** to not override the assigned color for that cell. Passing in an empty string for the font name, or zero for the size will likewise leave those values set to whatever the cell is currently configured for). **SetTextOverride** does ***NOT*** change the default settings for the cell.

SNS_MONEY_MANAGEMENT

This example is a modified version of the **TRO_DRAGON_MONEY_MGMT** indicator that I originally modified to display just the information I was particularly interested in.

I have now refactored this code to use the SNS_TABLE_OBJECT as a "real world" demonstration of how to use the table in a real indicator.



This indicator uses the table object to display this data on a chart. It uses a table with two columns and 10 rows (0-9). Row 0, Cell 0 is used for the table name or title (Row 0, Cell 1 is ignored and "overwritten" by the large font header in Cell 0)

NOTE that nothing prevents your data from "overflowing" past the right side of a "cell". All a CELL really is an encapsulated MQL4 LABEL OBJECT (position and font information is encapsulated).

BOTH of the example indicators provided use **EXTERN**ally exposed properties that allow an end user to change the row and column information, including individual cell font settings. As MQL4 does not allow for EXTERN arrays, what I did is allow the user to enter lists of data, such as cell font names, as semi-colon delimitated values which the code then shows how to transpose into the two-dimensional CELL property arrays.

EXTERN PROPERTIES EXAMPLE:

```
// List Column (X) and Row (Y) values as string with values separated by semi-colons
extern string  Table.Columns  =  "0;140";
extern string  Table.Rows     =  "0;25;50;70;100;125;150;175;200;225";

// List Cell Font attributes as a string with values separated by semi-colons - any left off are set to the DEFAULT values in Table.Initialize()
extern string  Cell.Fonts     =       "Verdana Bold;Verdana;Verdana;Verdana;Verdana;Verdana;Verdana Bold";
extern string  Cell.FontSizes =       "14;0;12;12;12;12;14;14";
extern string      Cell.FontColors =
"RoyalBlue;RoyalBlue;Orange;Orange;Gold;Gold;White;White;Red;Tomato;Red;Red;Orange;Orange;Orange;Orange;SteelBlue;SteelBlue;Orange;Orange";
```

EXAMPLE table initialization using above extern properties:

```
void Table.Initialize(string name){

        string f[],fonts[ROWS][COLUMNS],fcolors[];
        color colors[ROWS][COLUMNS];
        int cols[],rows[],fsizes[],sizes[ROWS][COLUMNS];

        String.ToIntArray(cols,Table.Columns);
        String.ToIntArray(rows,Table.Rows);

        Table.SetCorner(Table.Corner);

        if (Table.Init(name,rows,cols)<0) Print("TABLE FAILED TO INITIALIZE");
         else Table.Initialized=true;

        // convert EXTERN string "arrays" into "real" arrays (because MT4 does not provide for EXTERN of array variables)
        int fas = String.ToArray(f,Cell.Fonts);
        int fss = String.ToIntArray(fsizes,Cell.FontSizes);
        int fcs = String.ToArray(fcolors,Cell.FontColors);


        //Plug values from one-dimensional arrays into corresponding two-dimensional cell arrays
        int i=0;
        for (int r=0; r<ROWS; r++) {

                for(int c=0; c<COLUMNS; c++) {

                        if(i<fas) fonts[r,c]=f[i];
                        if(i<fss) sizes[r,c]=fsizes[i];
                        if(i<fcs) colors[r,c]=String.ToColor(fcolors[i]);
                        i++;

                }

        }
        Table.SetFonts(fonts);
        Table.SetFontColors(colors);
        Table.SetFontSizes(sizes);

        // Now run through cells and find any that were not set (missing from EXTERN string) and set their properties to these DEFAULT values:
        Table.SetDefaultFont("Verdana",12,White);

}
```

You can easily copy and paste this code into your own indicators.

Note that by executing **SetDefaultFonts** at the ***end*** of the initialization process, you insure the default font settings for any cells which did not have corresponding property settings included in the EXTERN strings.

In the code example above, for example, I did not list a font name for EVERY cell in the table, but I did include colors for every one. **SetDefaultFonts** makes sure the cells I did not include font names for will be set to “Verdana”.

All (or any) Table settings do not necessarily need to be exposed to end users. You could, for example, allow the user to provide only a row spacing value that you would then use programmatically to calculate each rows' relative position to the preceding row. Personally, I like to be able to tweak just about any setting possible in my indicators so I tend to expose everything.