

20091220 StockChartX Settings

Malcolm Moore

© 20-Dec-2009

Contents

Background	1
Working with StockChartX	1
The Quality Process of Running StockChartX.....	1
Registering the Object	2
Inserting StockChartX into a VB6 Project	3
Position StockchartX.....	4
Twips in a Nutshell	4
Check the Positioning.....	4
Understanding Panels.....	4
Adding a Panel.....	5

Background

StockChartX is a complex object that provides a powerful stock market graphing ability. It was developed by a software company called Modulusfe in the USA and the website is as <http://www.modulusfe.com/stockchartx/> If you are a regular programmer then the Modulusfe documentation that comes with the product is somewhat helpful if you are a regular programmer. Like most software guides it severely lacks the necessary detail, and so this document was written as a running sheet to provide the extra detail as necessary to make StockChartX work – in this case with **Visual Basic 6**. My understanding is that StockChartX will work with VB.Net, though I have not tried it – but I have seen the demo working with VB.Net. Experience has shown that StockChartX has difficulty working with VB 2010.

This StockChartX object comes with about 200 different attribute controls and feeds to and from it, so it makes sense to break these feeds and controls into groups that they can be handled separately. The second advantage of this approach is that there is a procedure in managing the feeds and controls associated with StockChartX that brings the processing down to a very manageable level and greatly assists rapid software development.

Working with StockChartX

StockChartX is a rather powerful and complex ActiveX object. In 2008 it was being continually updated with bugs being fixed on an almost weekly basis. Since then the StockChartX object has stabilised with almost all the bugs removed. Unfortunately it is not a simple matter of downloading and saving the object, as the object has references in the Registry, and it is necessary to clean the Registry of these references before loading the newly updated object. When this was written in 2009, I believe this product is now rather stable.

The Quality Process of Running StockChartX

Now that we have a basic appreciation of the StockchartX object, the next step is to use the Quality process to ensure that when a the StockChartX object is loaded into a VB6 program, the associated data can be loads and presented through the StockChartX object in a minimum of time and without any problems. There are three main steps in getting the StockChartX object in and functioning properly and these steps are to:

- Have the object registered so that it can be used in the VB project.

- Position StockChartX in the form
- Copy the Trade Data into StockChartX
- Activate one or more Indicators in StockChartX
- Run the Presentation routine

If this sequence is followed, then the development time taken to get StockChartX up and functioning properly will in most cases be reduced from several weeks to a few hours.

The difficulty here is that you need to have a concept of what you want the final outcome (graphs / charts) to look like before you start, because this dictates what data you will arrange to feed into the StockChartX object, and how that data will be presented.

The following areas step through the process of the above dot points and include code snippets to assist with the structure of the programming. The StockChartX object has two compiled HTTP documents that provide assistance of the StockChartX object itself (stock.chm) and a large number of Technical Analysis tools /indicators (TA.chm) that are already part of the StockChartX object. These documents are useful but getting started is rather difficult without firstly working through this document.

Registering the Object

Register StockchartX and ensure that its' Icon is in the left-hand side panel.

Files that are created as [self-registering](#) have information that need to be stored in the Windows [registry](#) in order for the file to be available and useable to applications on the machine. Usually DLLs, OCXs, and EXEs are self-registering and require to be registered on the machine.

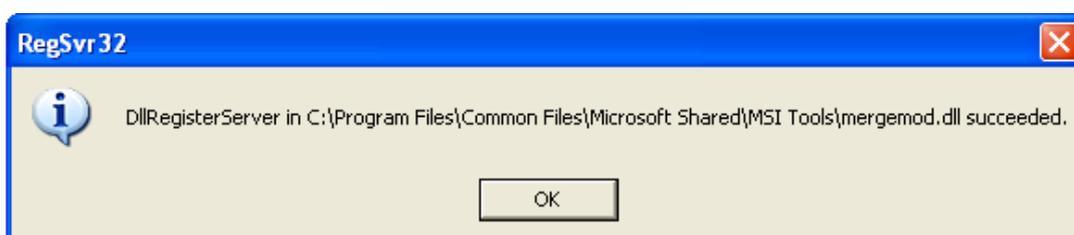
Follow these steps to register DLLs or OCXs:

On the Taskbar, select Start > Run.

In the Open field, type `regsvr32 "<path>\NameOfFile.extension"`, where <path> is the directory where the file is located on your computer and NameOfFile.extension is the name of the file. In the case of the stockchartx.ocx object the line will be as follows:

```
regsvr32 "C:\Windows\System32\stockchartx.ocx"
```

When you then press enter the operating system will try to register this object in the registry, and when successful will come up with a message like the one below to show that the process was successful:



In a very similar fashion when it comes to registering “.exe” files the programming line is rather different. Follow these steps to register an executable, or .exe, file:

On the Taskbar, select Start > Run.

In the Open field, type "<path>\NameOfFile.exe" /REGSERVER, where <path> is the directory where the file is located on your computer and NameOfFile.exe is the name of the file. For example:

```
"C:\Program Files\MyAppLocation\MyApp.exe" /REGSERVER
```

Press Enter.

Most executables do not display any message boxes when registered. It depends on how the application is created. Some applications when registered may open on your computer.

Inserting StockChartX into a VB6 Project

After loading the StockChartX object into the project through the Properties toolbar icon, the associated icon will appear in the Toolbox as a faint 'pair of leaves'. Having chosen the document that you wish to put StockChartX into; firstly click on the StockChartX icon and then insert the object like you would with a command button or picture.

VB will insert this object as a unique instance with a number following (like 1) so the unique name for the first instance will be StockChartX1. The next instance in the same form would be StockchartX2.



Here is an example of the StockchartX1 object loaded onto a form.

The StockChartX1 object is publicly available on that form only, but not seen from any other form in a Multi Document Interface (MDI) project. If you want to bring up a StockChartX object in a form with particular attributes – for example with the details of a Symbol 's trade history, then save that symbol in a Public variable that is seen everywhere in the project. That way the lead-in coding that is run through when that form is opened will look at that public variable and conditionally load that data - or attributes - as per the code.

Position StockchartX

Put the object of StockchartX on the form. In the Form_Resize subroutine position StockchartX so that it is in the right place, and that this positioning is scalable. Do not use fixed coordinates for the width and height because if the program is used on another computer screen the object may extend past the screens boundaries. Here is an example of scalable positioning:

```
StockChartX1.Move 200, 2000, _  
    Me.ScaleWidth - 7800, Me.ScaleHeight - 3000  
  
StockChartX1.Visible = False
```

The StockChartX object is positioned 200 twips off the left hand border of the form (Me), 2000 twips down from the top border of the form, 7800 twips inside the right-hand border of the form and 3000 twips off the bottom border.

Twips in a Nutshell

The term “twips” is a measure of 1/20th of a printers “Point”, which in-effect makes the measures independent of the screen dimensions in terms of pixels (picture elements, or the “dots” on the display screens).

In more practical terms, there are approximately 1440 twips to a inch, approximately 567 twips to a centimetre or approximately 56.7 twips to a millimetre.

Check the Positioning

In the above snippet of programming the StockChartX object was made invisible until we need it.

When we need to show the StockChartX object, the object made visible in that form using the coding as follows:

```
StockChartX1.Visible = True
```

The white area needs to be where you want it to be and now is the time to get this right, and try scaling down the screen size from what it is to prove that the object will work on screens with a much lesser resolution.

For speed it is much better to keep the object invisible and update all the parameters and in the final stages of the presentation stage make the object visible. This substantially speeds the process and minimises flicker.

Understanding Panels

StockChartX1 visually appears as a rectangular window, with a number of horizontal panes, (called panels) that would “appear” to be numbered from the top as 0, 1, 2 etc. however this process is not quite straightforward.

When we start with a blank StockChartX object there are no panels, so the default no panel reference is -1.

When we add the first panel, it assumes the internal reference number 0. If we now do a "PanelCount", then the response is "1" for the first panel and it has the reference number "0". This snippet of code will read the panel count into an Integer Variable NumberOfPanels

```
NumberOfPanels = StockChartX1.PanelCount
```

This numbering practice is the standard numbering structure for all Windows objects. If we were to look at a ComboBox then the counting practice is identical as it is for ListBoxes, MSHFlexGrids etc.

Note that the property "PanelCount" is a read only parameter, so you can't tell the StockChartX object to count the panels, but you can direct various series of data to be associated with a particular panel number. This data series control property is particularly important. If the data is directed to the wrong panel then the data will be displayed insignificantly, causing the StockChartX object to look entirely wrong.

Adding a Panel

Before we put data into StockChartX1, we need to create, identify and nominate the correct panel. Here is an example of some code to create a new panel:

```
Dim PanelNo as Long    'This is the Panel Number
```

```
PanelNo = StockChartX1.AddChartPanel()
```

As this is the first panel to be added to the object in this example, the variable PanelNo will now assume the value 0, and the first panel has now been added to the current StockChartX1 object for display. This first panel will vertically fill almost all the available visual space (except for the Date / Time bar across the bottom of the StockChartX object).

If we were to now read the panel count, then in this case the "Number" would be "1" and the panel reference would be "0".

```
Number = StockChartX1.PanelCount
```

If we added another panel, by using the same programming line as below,

```
PanelNo = StockChartX1.AddChartPanel()
```

Then this second panel (1) will sit below the first panel and occupy 50% of the vertical space that was held by the first panel. If we added a third panel (2), there will be three panels on the screen, this third panel will occupy 50% of the vertical space that was allocated to the previous panel's vertical space and sit immediately below the first panel.

So, every time a new panel is added to the object, the vertical space for this panel will be half that of the last panel's vertical height, and the new panel will be located directly below the first panel.

There is a discontinuity in the way that StockChartX positions, numbers and manages these panels, as what really happens is that when the third panel (reference "2") is added, the data that was in panel 1 (on the bottom), moves into panel 2 (which is now on the bottom), leaving a clean panel number 1 below the price data and above the volume data.

If a fourth panel is programmed in then the (volume) data that was in panel 2 moves down into panel 3, leaving panel 2 above panel 3, and panel 2 is blank.

It appears that this repositioning of the volume data to the bottom of the screen has been programmed in intentionally so that the volume data always appears at the bottom of the screen (yet to be proven).

To further confuse the issue, if an activity references the zero (0) panel, then the panel stack is lost and StockChartX reverts back to panels 0 and 1, and the panels have to be added in again

Each panel has a unique graphing space and a vertical bar of calibration space (normally on the right hand side). This vertical bar is under common control and its control is described under "Positioning the Vertical Calibrations" in the presentation area.

Having now worked through the apparent internal automatic programming to keep the Volume data on the bottom of the visual screen, it also has to be understood that when the historical data is loaded into StockChartX, then that routine systematically adds the second panel (1), and puts the Volume data there. If there is need to insert indicators after panel 0 then these panels have to be addressable (i.e. these panels have to exist) before the indicators are inserted into those panels.