

# eSignal Formula Script (EFS) Tutorial Series

## INTRODUCTORY TUTORIAL 3

---

### Introduction to preMain() and main()

**Summary:** This tutorial introduces the specific purpose and usage of the preMain() and main() user-defined functions.

#### Topic List:

1. main()
2. preMain() - Study Properties
3. preMain() - Study Formatting
4. preMain() - Study Parameters (FunctionParameter Class)

## 3.1 main()

The **main()** function is where the study logic begins and ends. It is this function that does all the work based on your custom designs. As we previously discussed in the first tutorial (section 1.4 EFS Formula Execution), we explained when and how an EFS formula gets executed. To be more specific, each time an execution is triggered, it is the **main()** function that gets executed. The code within **main()** will be executed line by line starting from the top of the function. All of your code could be contained within **main()** or you could call other user-defined functions from within **main()** to recycle various code routines or to organize your study logic. We covered an example of this in section 2.4.3, The **function** Statement, in the previous tutorial. When the processing reaches the end of **main()** you have the option to return various series of information produced by your formula to the Advanced Chart. The purpose of this section is to explain how **main()** is used to return this information to the Advanced Chart.

### 3.1.1 The **return** statement in **main()**

Like any other user-defined function, **main()** also uses the **return** statement. When a number data type is returned from **main** with this statement, EFS will plot these values in the Advanced Chart. How this plot appears visually is determined by the settings you establish in the **preMain()** function, which we'll cover in depth in the following sections. You've already seen many code examples of **main()**'s return statement in action in the previous tutorials. For this tutorial we'll start with a basic example where we are returning the result of a simple moving average with no **preMain()** configurations (see figure 1).



As you can see, the plot defaults to a non-price study with a blue line that has a line thickness of 1.

### 3.1.2 Returning Arrays

The previous section showed you how to return a single data series to the chart. However, many studies need to return multiple data series to the chart. This is accomplished by returning an array of values from the return statement in **main()**. In figure 2, we've added another simple moving average study and return these from the **return** statement as an array.



This can also be done by creating an Array Object and returning the identifier of the object name of this array rather than using the **new Array(x, x);** syntax. The code example in figure 3 would plot the exact same result as in figure 2 above.

There are two important notes regarding the return statement in main(). First, you cannot return a multi-dimensional array (an array where each element in the array contains another array) from main(). Secondly, an EFS formula may only return data series to either the price window or the non-price window, not both simultaneously.

### 3.1.3 Returning Strings

A string data type may also be returned from **main()**. However, the values will not be plotted on the chart but they will be visible in the cursor window. Using this method can be helpful in situations where you may want to know what the value of a data series is but

```

5 var myArray = new Array();
6
7 function main() {
8     myArray[0] = sma(10);
9     myArray[1] = sma(20);
10
11     return myArray;
12 }
  
```

Figure 3.

Published by eSignal (www.esignal.com)



do not want to plot it on the chart. To convert a number to a string, you can simply concatenate an empty string to the number using `+` `""` or the `.toFixed(n)` method where `n` is the number of decimals to display. The `.toFixed()` method converts the result to a string by default. In figure 4, we have converted the result of the second moving average from our previous code example to a string value.



Notice that in the cursor window you can still see the value of the second moving average, but it does not appear on the chart.

A note of importance on returning strings is that they may also be words that indicate a custom state, such as "Long" or "Short."

### 3.1.4 Returning Booleans

Boolean (true/false) data may also be returned from `main()`. What appears in the cursor window will be the string of **On** or **Off** where the values returned from `main()` are **true** or **false**, respectively. This is not a frequently used process, but is worth mentioning.

### 3.1.5 Commenting //main()

There is one small programming idiosyncrasy with **main()** that you need to be aware of. When a formula is first applied to an Advanced Chart, the EFS code is interpreted into memory and referenced from that point forward. When the EFS engine is interpreting the formula code it searches for the first instance of the string; “main()” within the code. If you have this string on a commented line or within a comment block above your active function for main(), the study parameters (if any are used) will not be properly interpreted. The result is that the dialog options will not be present when you use the Edit Studies option to make an adjustment to a study parameter. Study Parameters are covered in depth later in this tutorial. To avoid this problem, simply eliminate using the string of “main()” inside any commented areas above the main() function that you intend to be the active function. Alternatively, you can simply remove the function call operators (i.e. **()**) that appear after “main()” in the commented area.

## 3.2 preMain() – Study Properties

The purpose of **preMain()** is to tell the EFS environment what specific properties or behaviors the study will use. This function is only executed once by the EFS engine when a study is initially applied to the Advanced Chart. The EFS functions that control the study properties determine if the study will be applied to the price pane or indicator pane, which can also be referred to as price studies and non-price studies, respectively. The property functions also set characteristics such as the study title and cursor label names, among others. An important programming note to remember when adding or modifying preMain() functions, is that in most cases your code changes will not be reflected upon a manual reload (Chart Options→Reload) of the study. You will need to remove (Chart Options→Remove) the study and reapply it to the Advanced Chart to ensure all preMain() changes take affect. This has to be performed because some of the properties can only be processed when the study is being interpreted on the initial load by the EFS engine. A manual reload does not destruct and re-interpret all of the preMain() settings. This section will cover the most commonly used collection of EFS properties that may be configured through EFS functions.

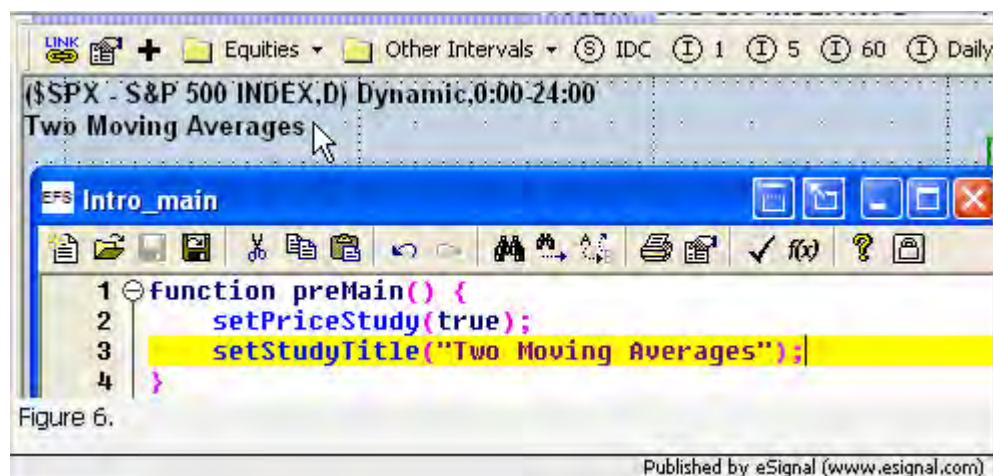
### 3.2.1 setPriceStudy(Boolean)

This function is used to determine where a study will display its returned data series. If the function is not used, the study will be a non-price study by default as seen in the previous section. The function accepts one Boolean (true/false) parameter. Specifying **true** will make the study plot as a price study in the price window of the Advanced Chart. Figure 5 shows an example of a price study that builds onto our two moving averages study from the previous section.



### 3.2.2 [setStudyTitle\(sTitle\)](#)

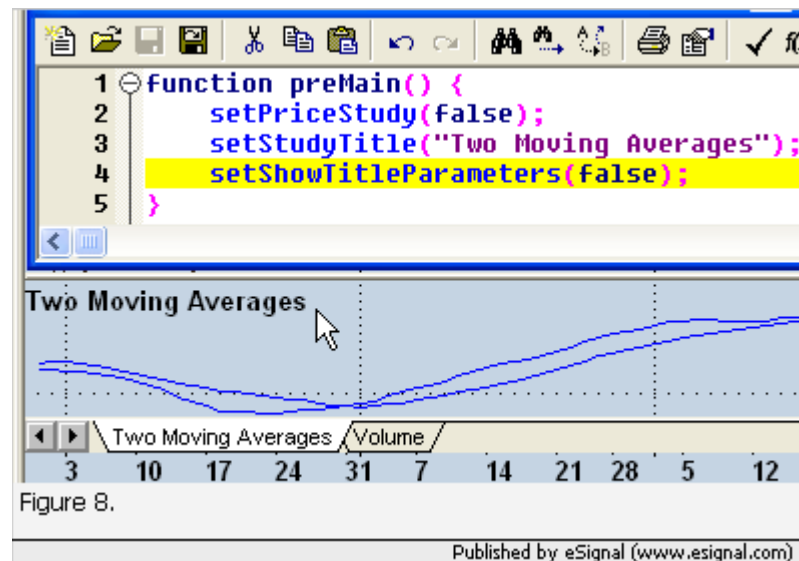
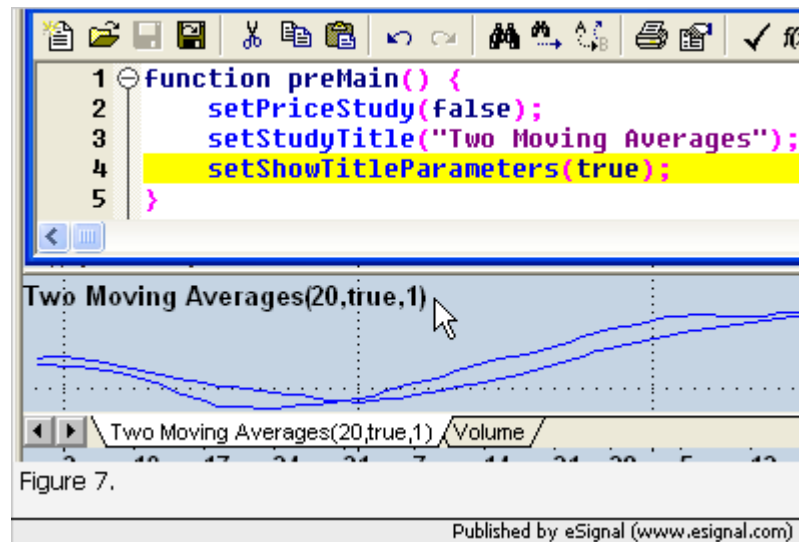
This function sets the title for the study that will appear in the top left corner of the pane where the study is operating. The function accepts one parameter as a string as seen in figure 6.





### 3.2.3 [setShowTitleParameters\(Boolean\)](#)

This function determines whether or not to display the current settings or values for the study parameters if they are used by the study. We will cover study parameters in detail later in this tutorial. If the function is not used or is set to **true**, any study parameters will be displayed in parentheses after the study title. Specifying **false** will prevent them from being displayed. This function is especially helpful if you prefer to use the Stack Studies option from the Chart Options menu for non-price studies. Figure 7 shows an example without using this function and Figure 8 shows the result after applying this function.



### 3.2.4 [setCursorLabelName\(sName, \[nIndex\]\)](#)

This function sets the name for a returned data series in the cursor window of the Advanced Chart. The function accepts two parameters. The first parameter is a string that represents the name for the series. The second parameter is optional, which is why it is surrounded by brackets. The nIndex parameter is used to specify which data series the label corresponds to based on the order in which it is returned by the **return** statement in **main()**. The first data series corresponds to the index value of 0. The second data series corresponds to the index value of 1 and so on and so forth. The index values start with 0 because arrays in JavaScript are zero-based, which is what is being returned in **main()**



(i.e. **return new Array(...)** ). Any of the preMain() functions that set a property or formatting option for a data series will have the optional index parameter. If a study only returns a single data series, the index parameter is not necessary. Figure 9 shows an example where we have added cursor labels for the two moving averages in the cursor window.

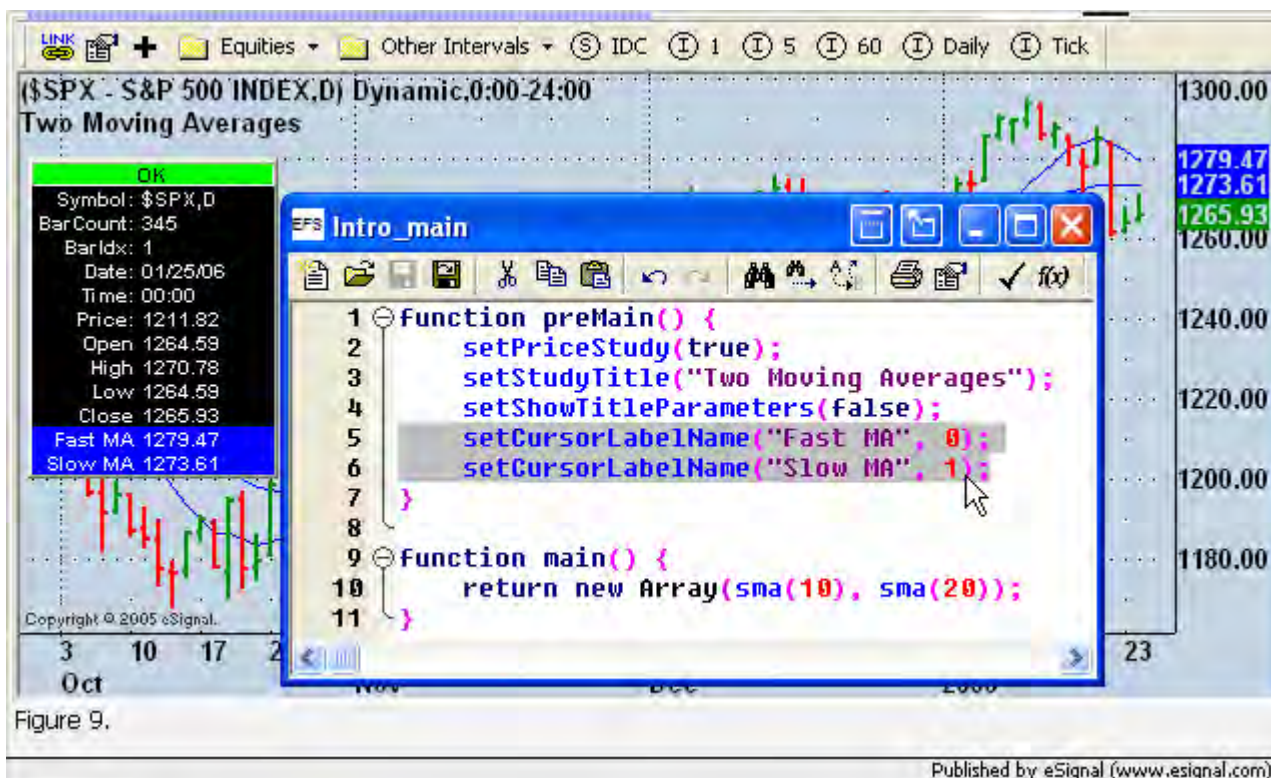
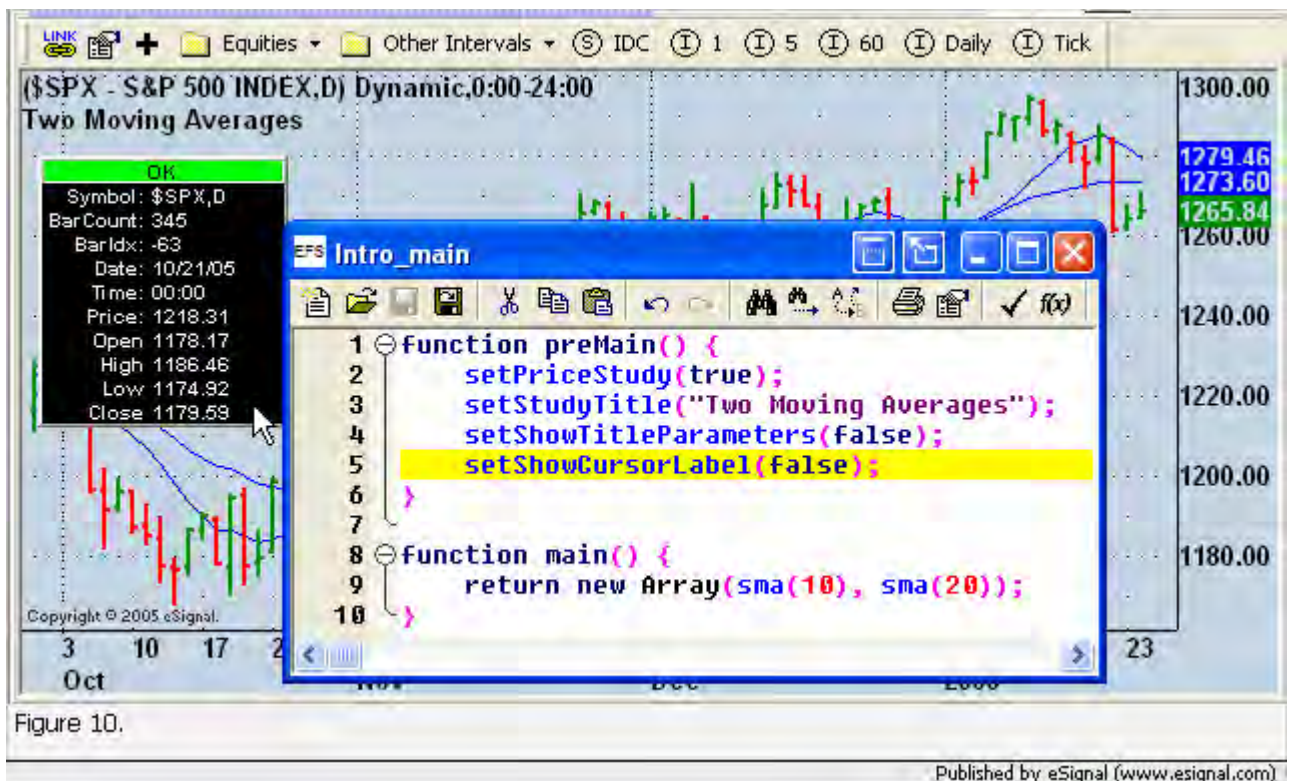


Figure 9.

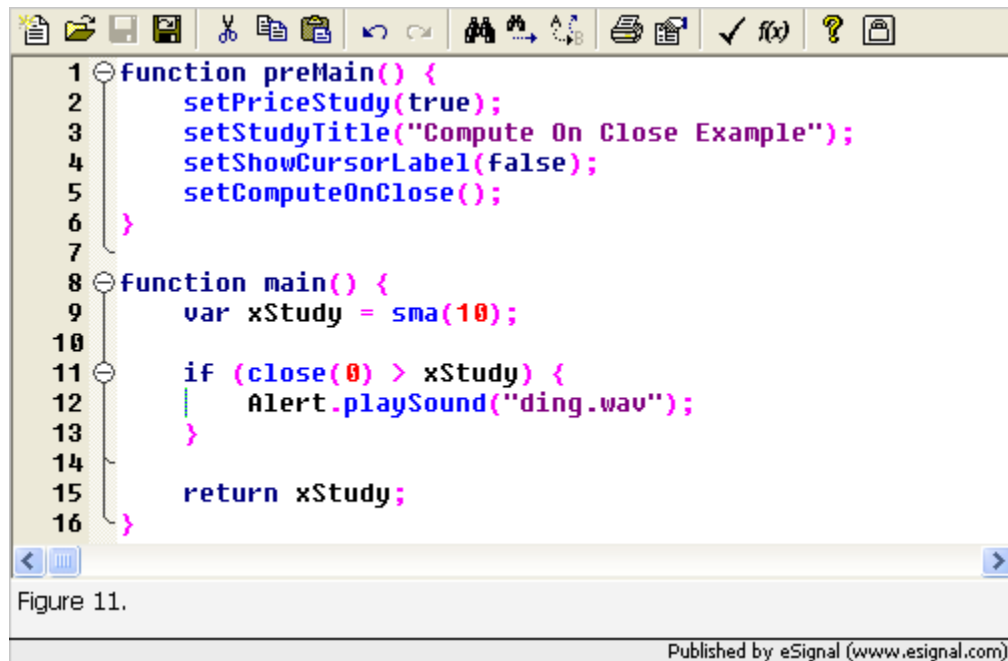
### 3.2.5 [setShowCursorLabel\(Boolean\)](#)

This function is used to turn off the display of the cursor labels in the cursor window. The function accepts one Boolean (true/false) parameter. Specifying **false**, turns off the display of the labels. Please note that this function does not have the optional index parameter. This function either shows all or none. Turning off the display of only certain labels is not currently possible. Figure 10 shows example with the cursor labels for the two moving averages turned off.



### 3.2.6 [setComputeOnClose\(\)](#)

This function determines if a study is to be executed on each trade update in real time or only on the completion of a price bar. This function does not require any parameters. It has a default state of true. EFS has a global setting (Tools→EFS→Settings) to force all studies to compute on close rather than tick by tick. However, you may only want certain studies to perform under this condition on intraday intervals. Adding this function to **preMain()** will force this behavior for that study only. This function is useful for simplifying the code logic where a study has conditions that you only want to evaluate upon the completion of a price bar to avoid receiving false signals during real time if the study was allowed to process each trade. For example, if you want to generate an alert only when a bar closes above a moving average you will need to prevent the alert from occurring when the last price (i.e. **close(0)**) crosses above the average during the interval. This is because it is possible that the last price may occur below the moving average when the interval closes. This would result in the alert condition evaluating to false at the close of the price bar. Figure 11, is a code example that you can try with some real time data on any symbol of your choosing. Click on this link to download the formula, [Intro COC.efs](#). Save it to a folder within your \eSignal\Formulas\ folder.



After running this study as is, make note that the study does not plot the moving average value on the current bar (bar index of 0). The **setComputeOnClose()** function prevents the study from executing until the current bar closes, therefore it does not have any data to plot on bar 0. Now comment out the **setComputeOnClose()** function with two forward slashes (i.e. `//`) and then save the study. Then remove the study from your chart and reapply it to see the difference while processing each trade rather than at the close of the price bars. You will see that the alert occurs on each trade that occurs above the moving average and the current bar will also have a plotted value for the moving average series.

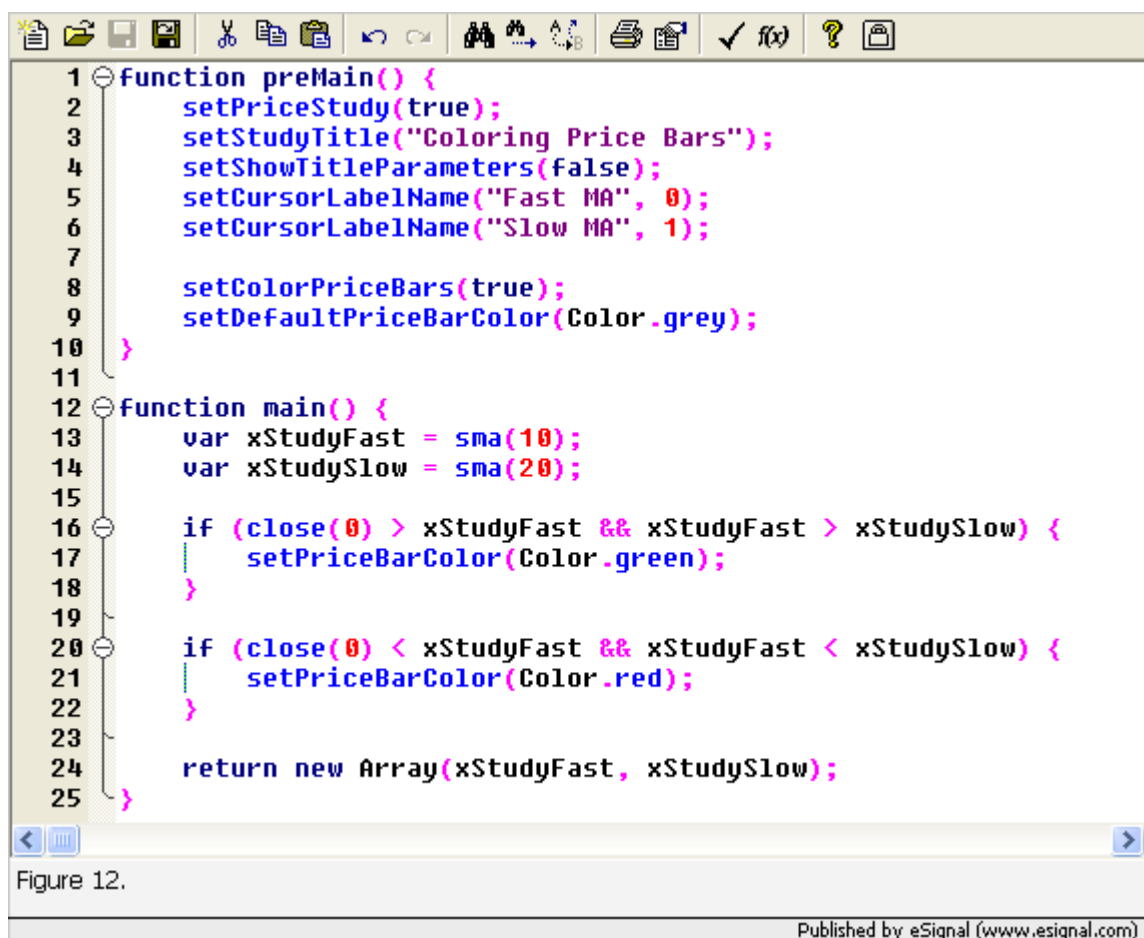
### 3.3 preMain() – Study Formatting

The EFS formatting functions control the appearance of the study's returned, or plotted, data series on the Advanced Chart. The following section will explore the available formatting options. Although these functions are included here under the **preMain()** sections of this tutorial, they may also be executed within **main()** or any other user-defined function with the only exception being **setColorPriceBars()**. They are primarily used in **preMain()** because they only need to be executed once to take affect. When they are used in **main()**, they are typically used within a routine that is only allowed to be executed once. Many studies are developed to allow you to change some of the formatting options through study parameters. When this action is performed, the study gets reinitialized so that any changes will also be reflected on the historical bars. This event can be detected inside **main()** so that the default formatting options can be set programmatically within the EFS code. We'll cover some code examples of this process in the Study Parameters section.

#### 3.3.1 [setColorPriceBars\(Boolean\)](#) and [setDefaultPriceBarColor\(Color\)](#)

These functions enable price bar coloring and sets the default price bar color. Both price and non-price studies may color the price bars using these functions. The **setColorPriceBars()** function accepts one Boolean (true/false) parameter where **true**

turns this feature on and **false** turns it off. The **setDefaultPriceBarColor()** function is used in tandem with **setColorPriceBars()**. If used by itself, the function will have no effect on the price bar coloring. The **setDefaultPriceBarColor()** function accepts a single color parameter. The color parameter is expected to be a color from the EFS **Color** object. The following link is a list of the available [color options](#). These two **preMain()** functions are only required if you want the default price bar color to be any color other than black. If your study logic sets the price bar color for every bar that appears on your chart there is no need to set a default in **preMain()**. The function used in your study logic to color a price bar is **setPriceBarColor(Color)**. The following code example in Figure 12 ([Intro\\_ColorPriceBars.efs](#)) will color the price bars green when the fast moving average is above the slow moving average and the close of the bar is also above the fast moving average. The price bars are colored red in the opposite condition.



```

1 function preMain() {
2     setPriceStudy(true);
3     setStudyTitle("Coloring Price Bars");
4     setShowTitleParameters(false);
5     setCursorLabelName("Fast MA", 0);
6     setCursorLabelName("Slow MA", 1);
7
8     setColorPriceBars(true);
9     setDefaultPriceBarColor(Color.grey);
10 }
11
12 function main() {
13     var xStudyFast = sma(10);
14     var xStudySlow = sma(20);
15
16     if (close(0) > xStudyFast && xStudyFast > xStudySlow) {
17         setPriceBarColor(Color.green);
18     }
19
20     if (close(0) < xStudyFast && xStudyFast < xStudySlow) {
21         setPriceBarColor(Color.red);
22     }
23
24     return new Array(xStudyFast, xStudySlow);
25 }

```

Figure 12.

Published by eSignal (www.esignal.com)

The chart image in figure 13 illustrates the results from this code example.





### 3.3.2 [setDefaultBarBgColor\(Color, \[nIndex\], \[yMin\], \[yMax\]\)](#)

This function sets the default bar background color for a data series. The function has one required parameter, `Color`, and three optional parameters, `nIndex`, `yMin` and `yMax`. The `nIndex` parameter assigns the coloring to the specified data series. However, there is only one background color that can be shown on the chart as each data series shares the same background. If no value is specified for `nIndex`, the default is 0, or the first data series from the return array in `main()`. If you specify the default background color for multiple data series, only the last one executed will be visible because it will be in the foreground. The following code example ([Intro\\_ColorBarBg.efs](#)) in Figure 14 uses our two moving average example and colors the background of the bars rather than the price bar colors as we did in the previous formula example. Also, just like in the previous section, notice that we have a corresponding function used in `main` to set the current bar's background color, `setBarBgColor()`, which has the same set of parameters. Also make note that we are not using the `nIndex` parameter in `preMain()` or `main()`.

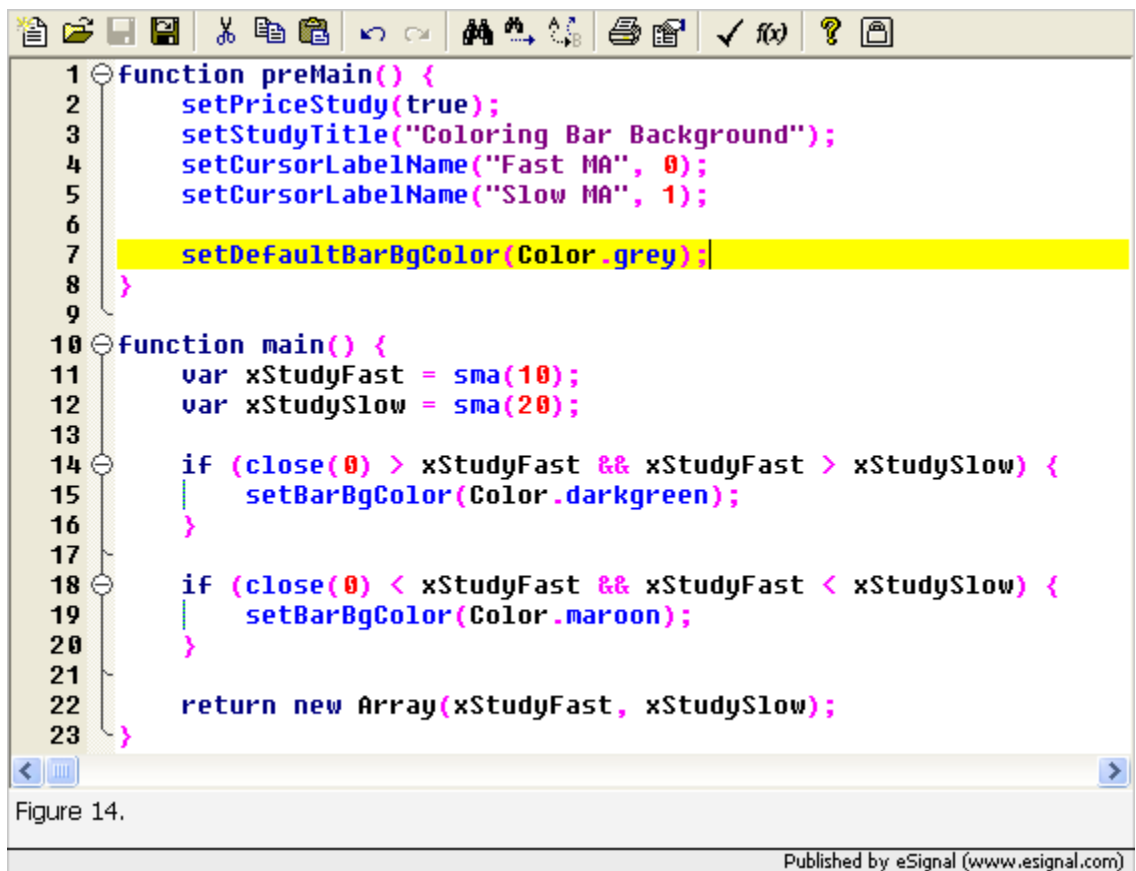
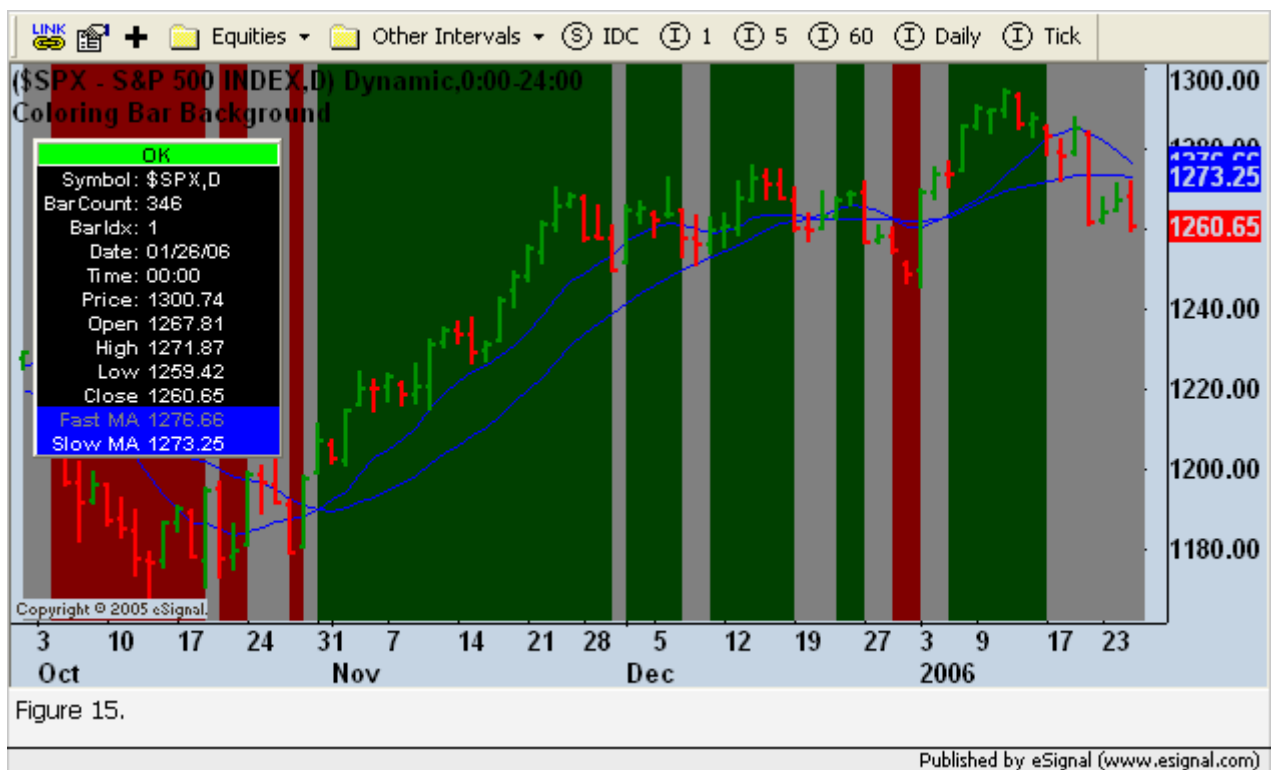
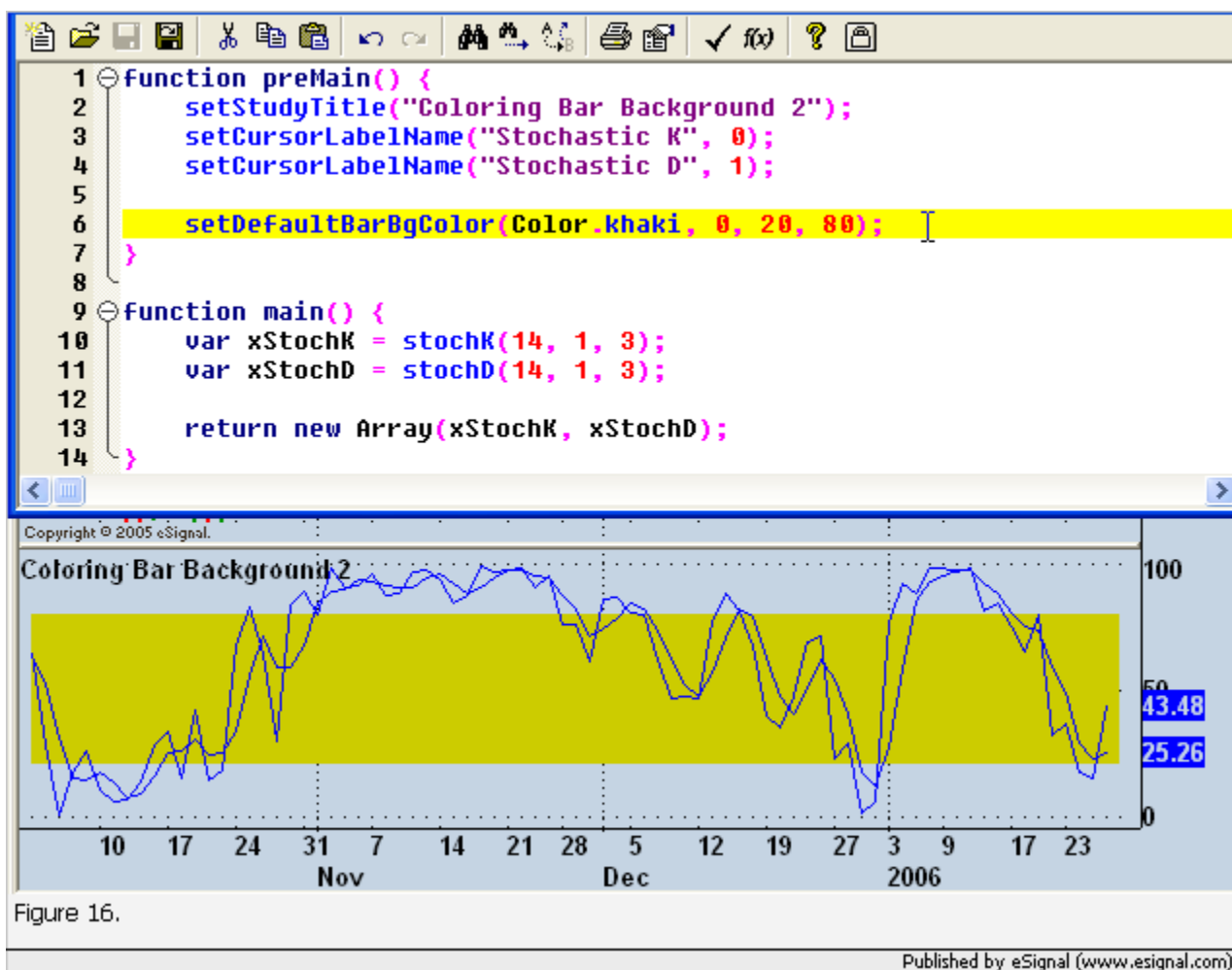


Figure 15 is a chart example of the formula results.



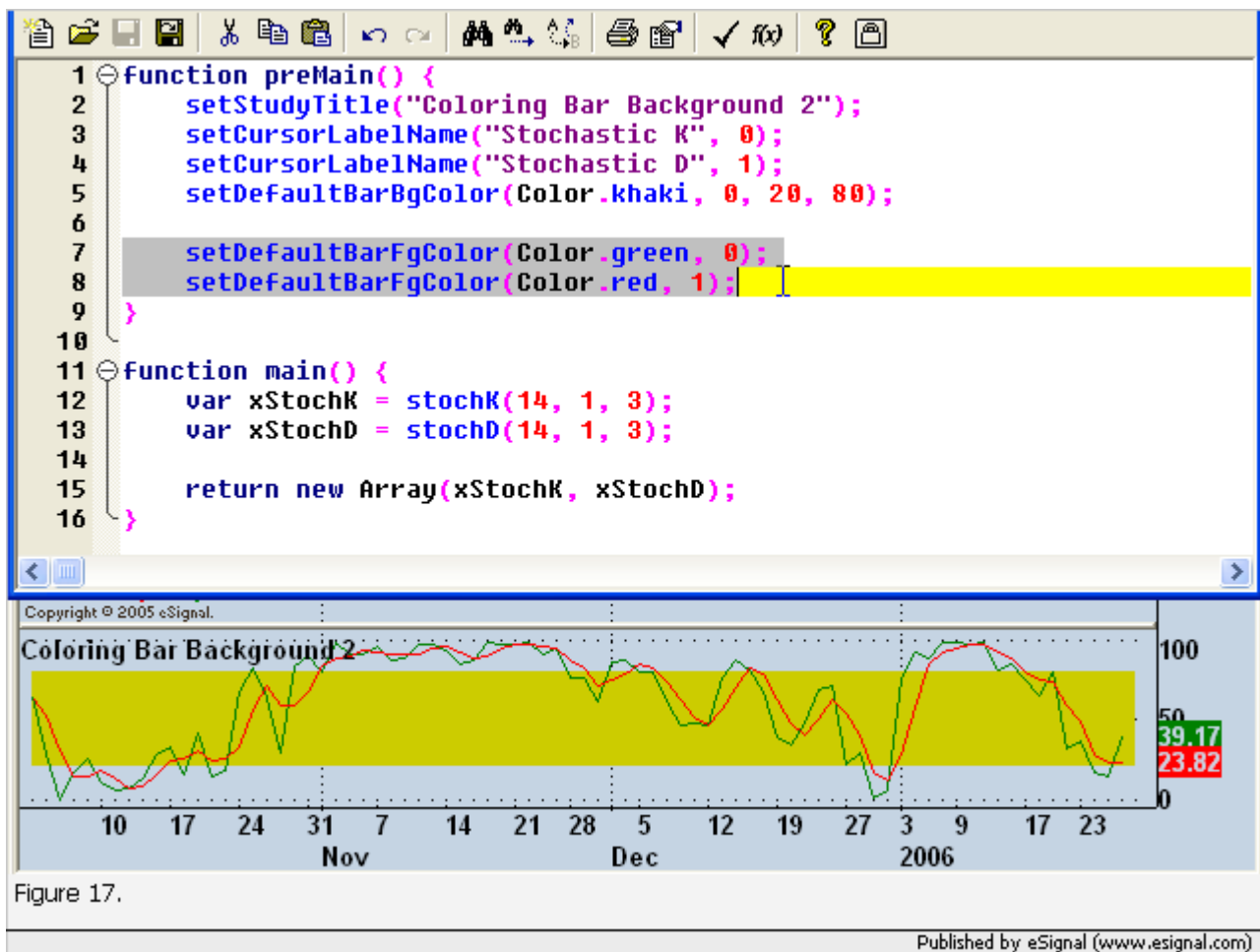
Notice also the color of the text in the cursor window for the first data series, Fast MA, is colored the same as our specified color we used by the functions `setDefaultBarBgColor()` and `setBarBgColor()`.

The last two optional parameters, `yMin` and `yMax`, allow you color only a portion of the background. These parameters are typically used with non-price oscillators that fluctuate within a fixed range. They can also be used in the price window, but for our example in Figure 16 ([Intro\\_ColorBarBg2.efs](#)), we will use a Stochastic study and color only the portion of the background between 80 and 20.



### 3.3.3 `setDefaultBarFgColor(Color, [nIndex])`

This function sets the default bar foreground color for a data series. The function requires a color for the first parameter and has an optional `nIndex` parameter used for studies with multiple data series. When this function is not used, the default color will be blue as we have seen in all the code examples prior to this section. In Figure 17 we have added this function to our previous code example to color the Stochastic K green and the Stochastic D red.



Like the other setDefault... functions there is a corresponding function used to set the foreground color of the current bar used in main, **setBarFgColor(Color, [nIndex])**.

### 3.3.4 **setDefaultBarStyle(Style, [nIndex])**

This function sets the default bar style for a data series. The function requires a Style parameter and also has the optional nIndex parameter for studies with multiple data series. The Style parameter is expected to be one of five EFS style constants. Figure 18 shows a basic code example of how to use these constants in **preMain()**.



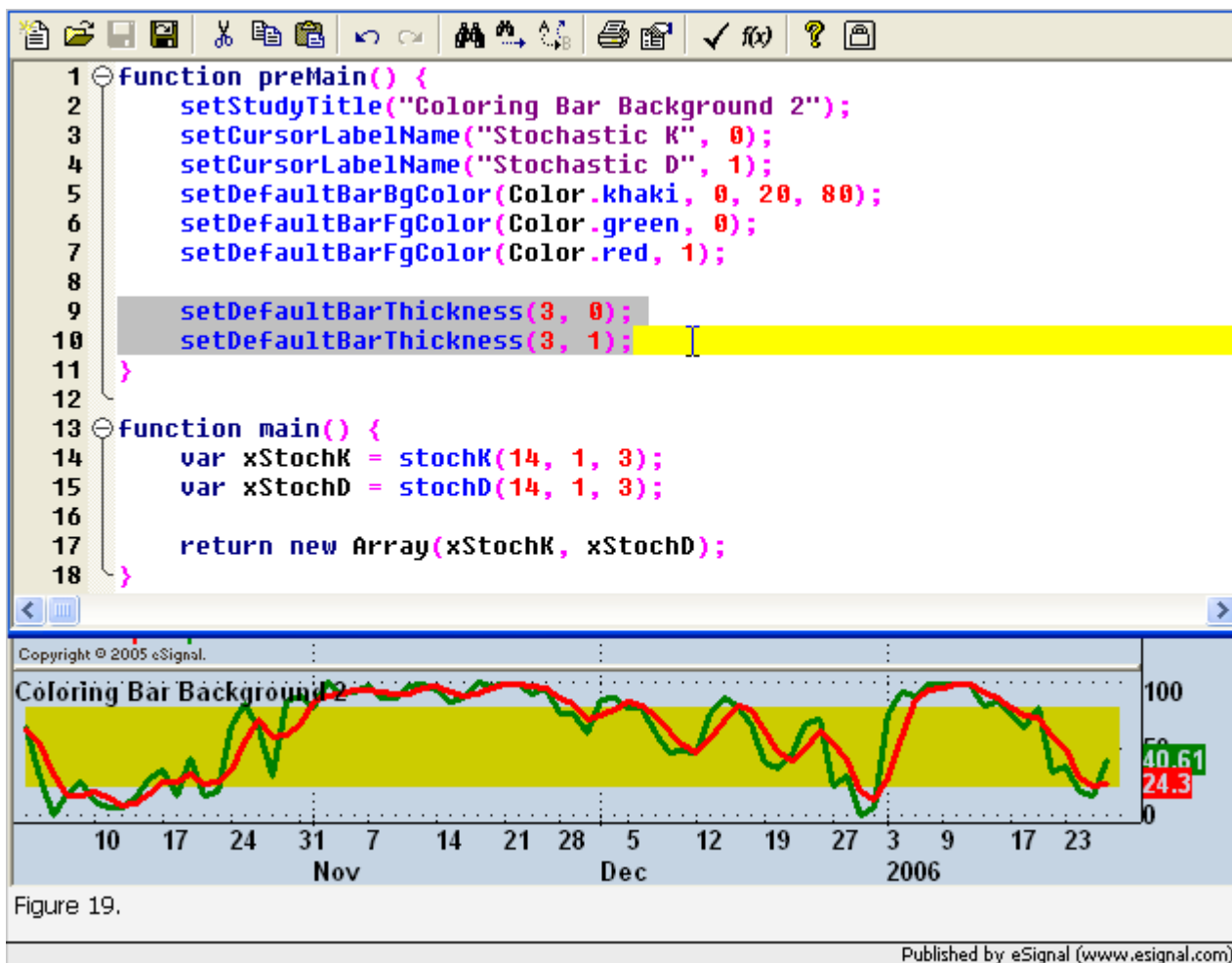
The following lists the five style constants with an example of their appearance.

- **PS\_SOLID** 
- **PS\_DOT** 
- **PS\_DASH** 
- **PS\_DASHDOT** 
- **PS\_DASHDOTDOT** 



### 3.3.5 [setDefaultBarThickness\(nThickness, \[nIndex\]\)](#)

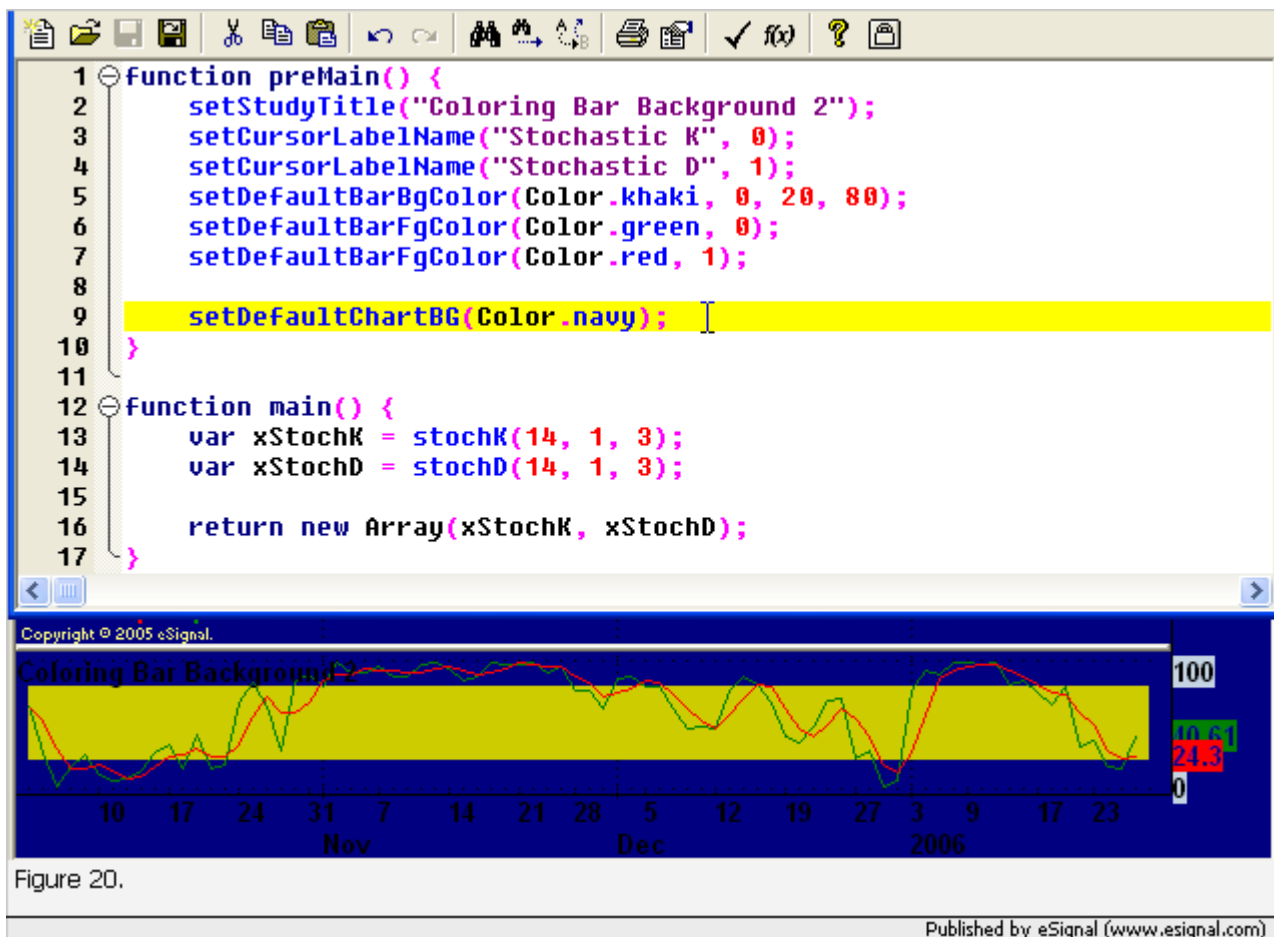
This function sets the default bar thickness for a data series. The function requires one parameter, `nThickness`, which is expected to be an integer value of 1 or greater. The function also has the optional `nIndex` parameter for studies that return multiple data series. If this function is not used in **preMain()** the default thickness will be set to 1. Figure 19 shows the functions with a thickness of 3 added to **preMain()** from our previous code example used in Figure 17 and the results in the accompanying chart.



Again, as with all other `setDefault...` functions, there is a corresponding function used in **main()** to set the thickness of the current bar, [setBarThickness\(Color, \[nIndex\]\)](#).

### 3.3.6 [setDefaultChartBG\(Color\)](#)

This function sets the default chart background color. The function has one required color parameter. This function will override the background color of an Advanced Chart that is set through the Chart Options→Properties window. If you have multiple formulas applied to the same chart that are setting the default background color, only the color set by the last formula applied to the chart will take affect. Figure 20 continues with our current code example and sets the chart background color to navy.



Once again, there is a corresponding function, **setChartBG(Color)**, that can be used in **main()** to change the color of the background while a study is processing in real time. This can be used as an indication of a custom state or simply to change the color based on a set of custom conditions.

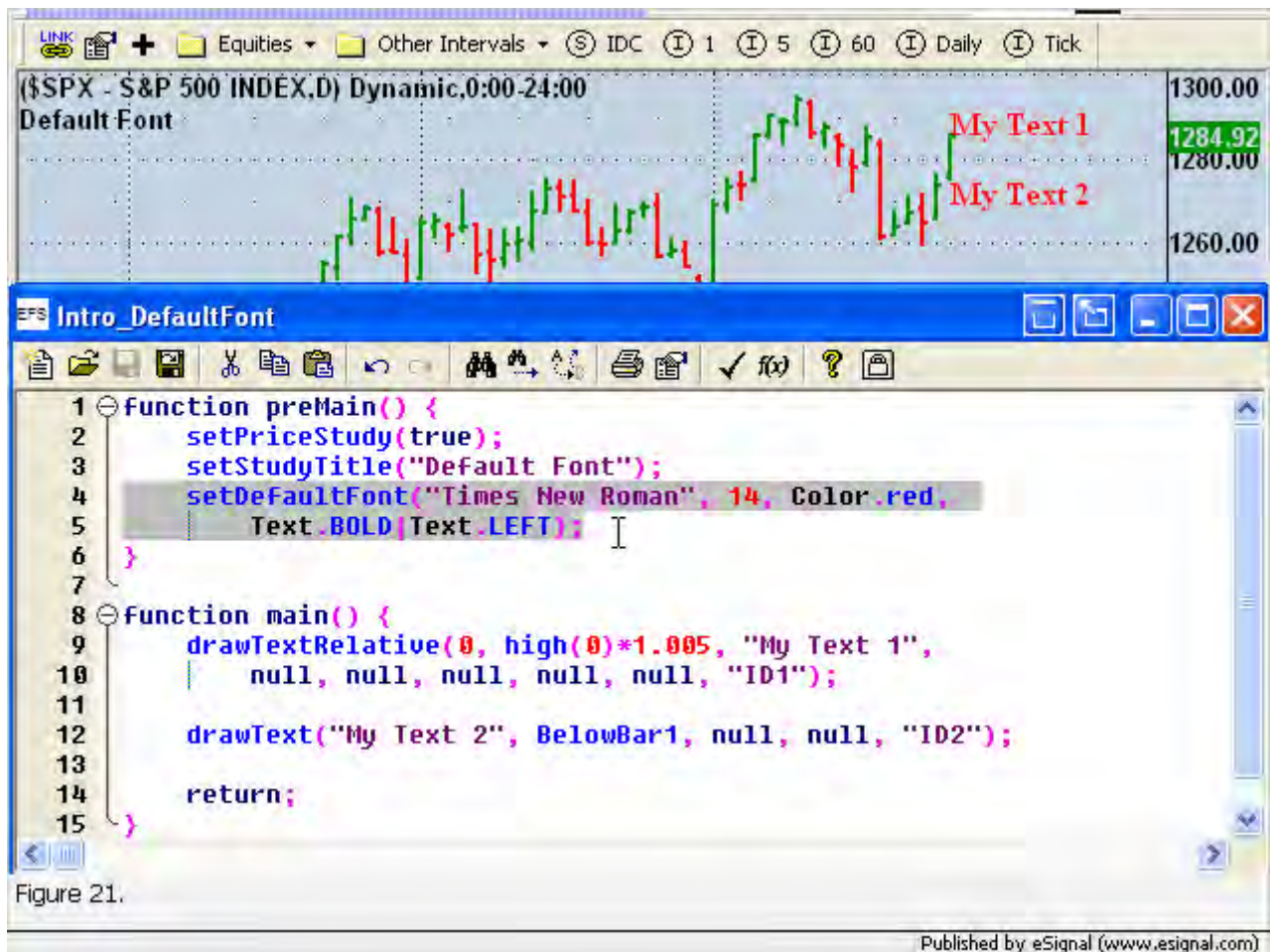
### 3.3.7 [setDefaultFont\(sFont, nSize, \[fgColor\], \[TextFlags\]\)](#)

This function sets the default font used with drawText... functions. The function requires the font type (sFont) as a string and the font size (nSize) as an integer. There are two optional parameters to set the color (fgColor) and any desired text flags (TextFlags). The font type can be any font that is available in your font list in the EFS Editor. To view the list, click on the properties icon in the EFS Editor and then click on the Font button. The text flags are a set of EFS constants that control the appearance of the text. The following is the list of available text flags that can be used with **setDefaultFont()**.

- **Text.BOLD** Display the text in bold.
- **Text.ITALIC** Display the text in italics.
- **Text.UNDERLINE** Underline the text.
- **Text.LEFT** Align the text to the left.
- **Text.RIGHT** Align the text to the right.
- **Text.TOP** Align the text to the top.
- **Text.BOTTOM** Align the text to the bottom.
- **Text.ONTOP** Draw the text on top of the study.
- **Text.FRAME** Draw a frame around the text using fgColor as the frame color.

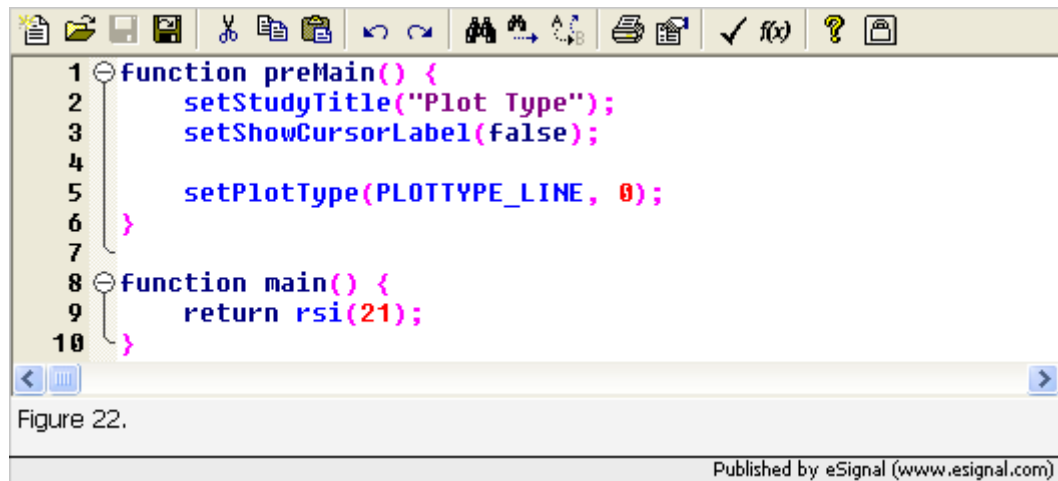
- **Text.BUTTON** Draws the text as a button (fgColor is ignored).
- **Text.CENTER** Horizontally center the text over the bar
- **Text.VCENTER** Vertically center the text
- **Text.PLAIN** Display plain text

The EFS text functions that utilize the default settings from `setDefaultFont()` are `drawText()` and `drawTextRelative()`. Both `drawText...` functions have parameters that can be specified within their own set of parameters. If these parameters are set within those functions, any defaults from `setDefaultFont()` will be ignored. The `setDefaultFont()` exists so that if you have multiple text objects being drawn by your EFS you can avoid have to set the parameters within the `drawText...` functions multiple times. In order for the `drawText...` functions to use the defaults set by `setDefaultFont()` you will need to enter **null** values for those parameters within the `drawText...` functions. Figure 21 ([Intro\\_DefaultFont.efs](#)) shows a basic example where both `drawText()` and `drawTextRelative()` are used in conjunction with `setDefaultFont()`.



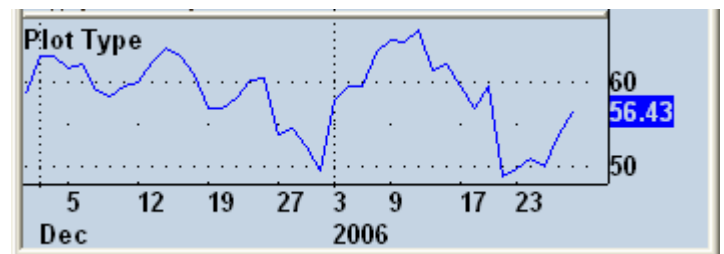
### 3.3.8 `setPlotType(nPlotType, [nIndex])`

This function sets the plot type for a data series. The function requires one of the EFS plot type constants and also has the optional `nIndex` parameter for studies that return multiple data series. If this function is not used, a data series will plot as a line by default. Figure 22 ([Intro\\_PlotType.efs](#)) is a basic code example that plots an RSI study as a line.

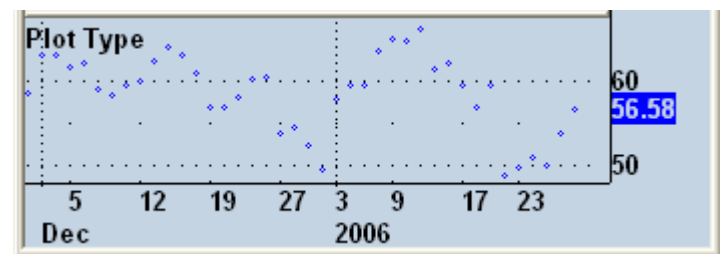


The following is a list of the available plot type constants with an image of their plotted result using the code example from Figure 22.

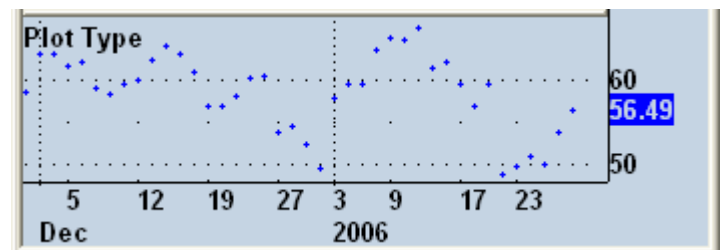
- **PLOTTYPE\_LINE**



- **PLOTTYPE\_CIRCLE**

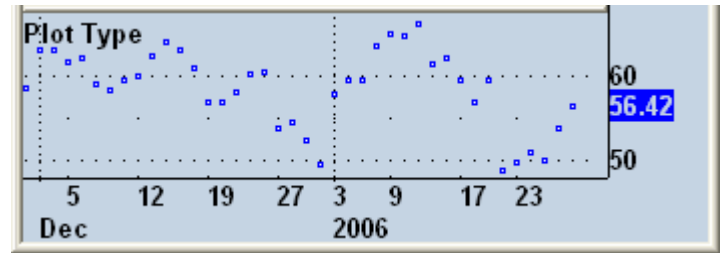


- **PLOTTYPE\_DOT**

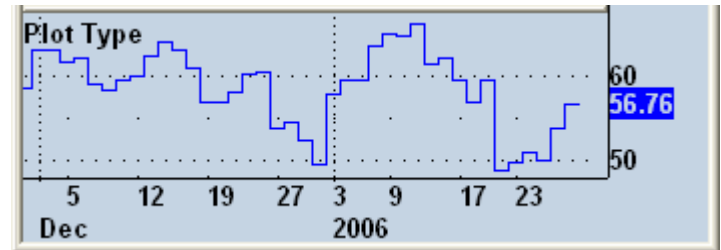


- **PLOTTYPE\_SQUARE**

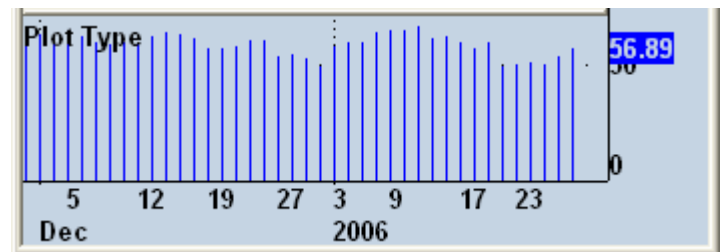




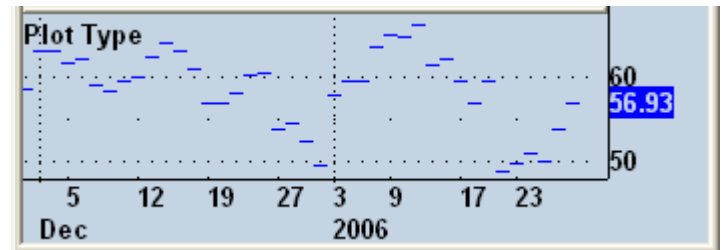
- **PLOTTYPE\_SQUAREWAVE**



- **PLOTTYPE\_HISTOGRAM**

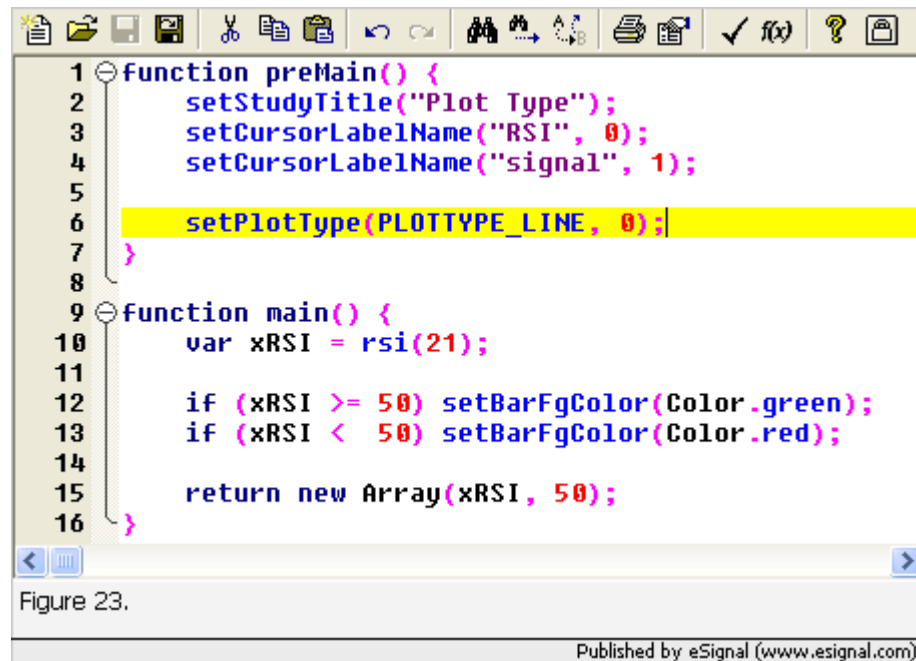


- **PLOTTYPE\_FLATLINES**



- **PLOTTYPE\_INSTANTCOLORLINE**

For this plot type we need to modify our code example to illustrate how it is used. Sometimes developers color indicator lines different colors to visually highlight a condition or signal. When the current bar's line segment is set to a different color you can see that the segment starts at the center of the current bar and goes to the center of the next bar by default. Figure 23 is the modified code where the RSI line is colored green when it is above 50 and red when below. Notice in this example that our plot type is set to **PLOTTYPE\_LINE**.



In Figure 24 below, notice that the bar highlighted with the red cursor tracking cross hairs (Chart Options→Cursor→Cursor Tracking) has a value for the RSI that is below 50. Due to the way the line segment are colored, its segment from that point to the next point to the right where the RSI value is above 50 is colored red.

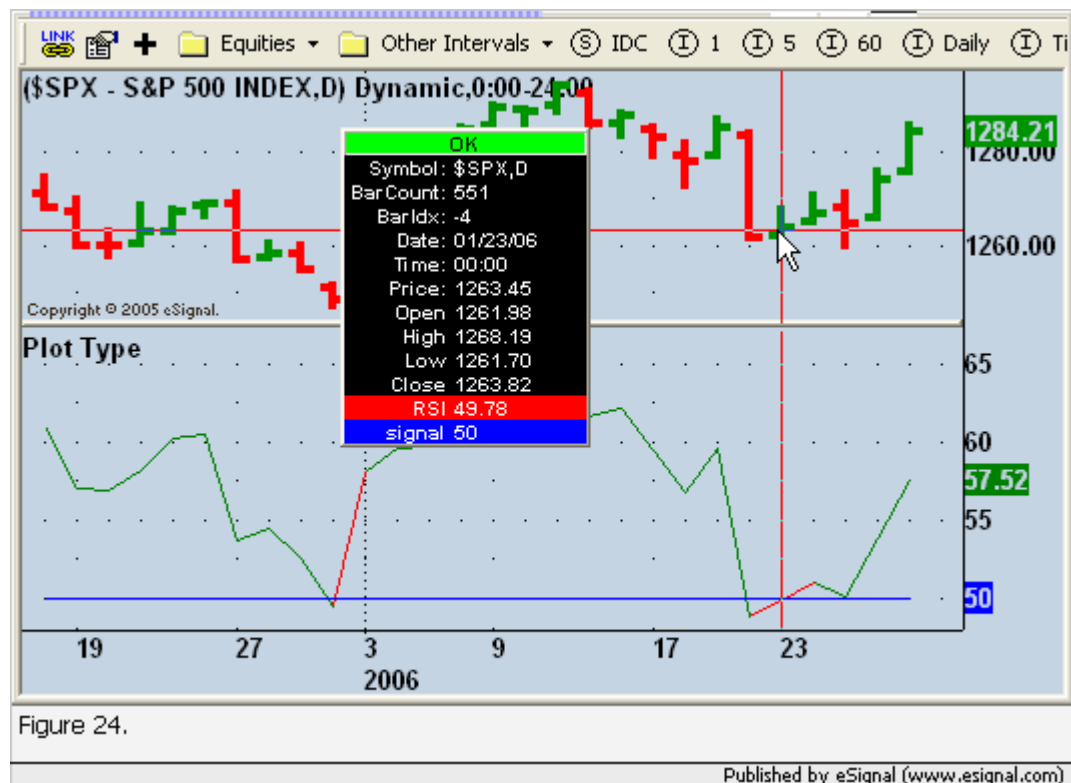
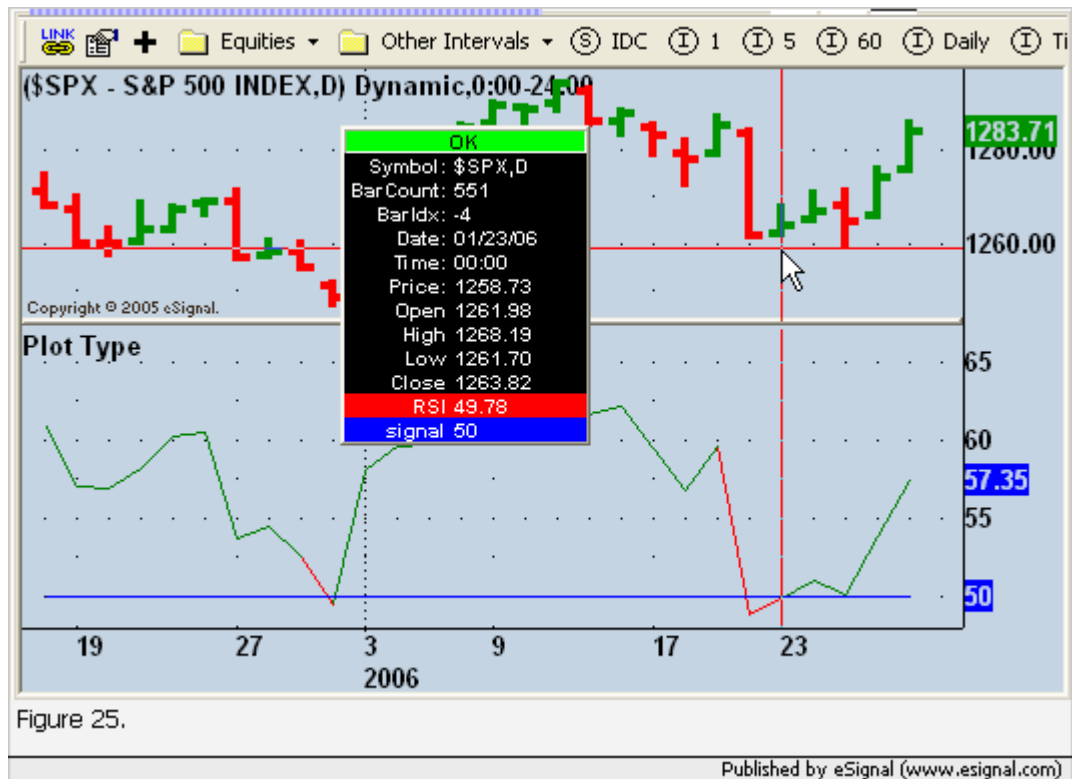


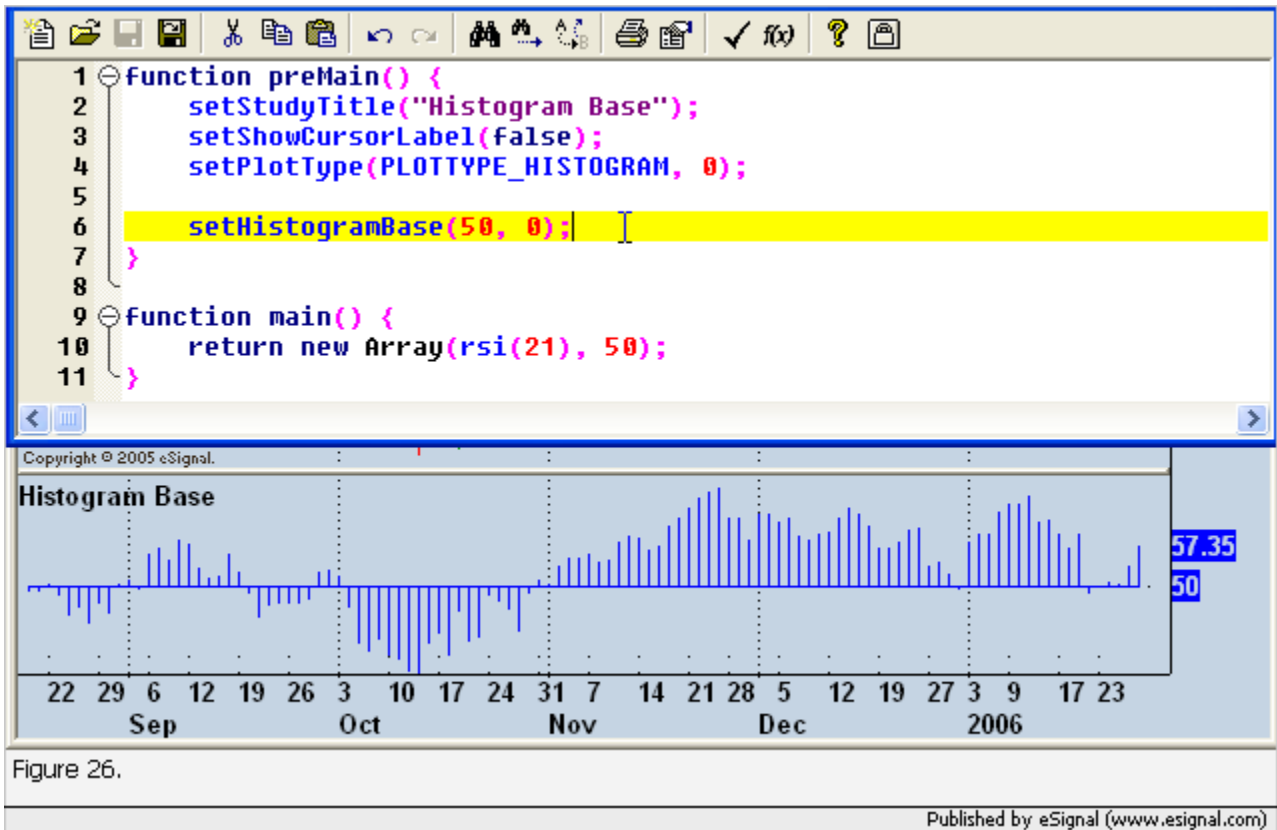
Figure 25 is the same study and chart after changing the plot type to **PLOTTTYPE\_INSTANTCOLORLINE**. Notice that our highlighted line segment is now green instead of red. The effect of this plot type changes the segment that

receives the color to the previous segment. The line segment for the highlighted bar now starts at the center of the previous bar to the center of the current bar.



### 3.3.9 [setHistogramBase\(nValue, \[nIndex\]\)](#)

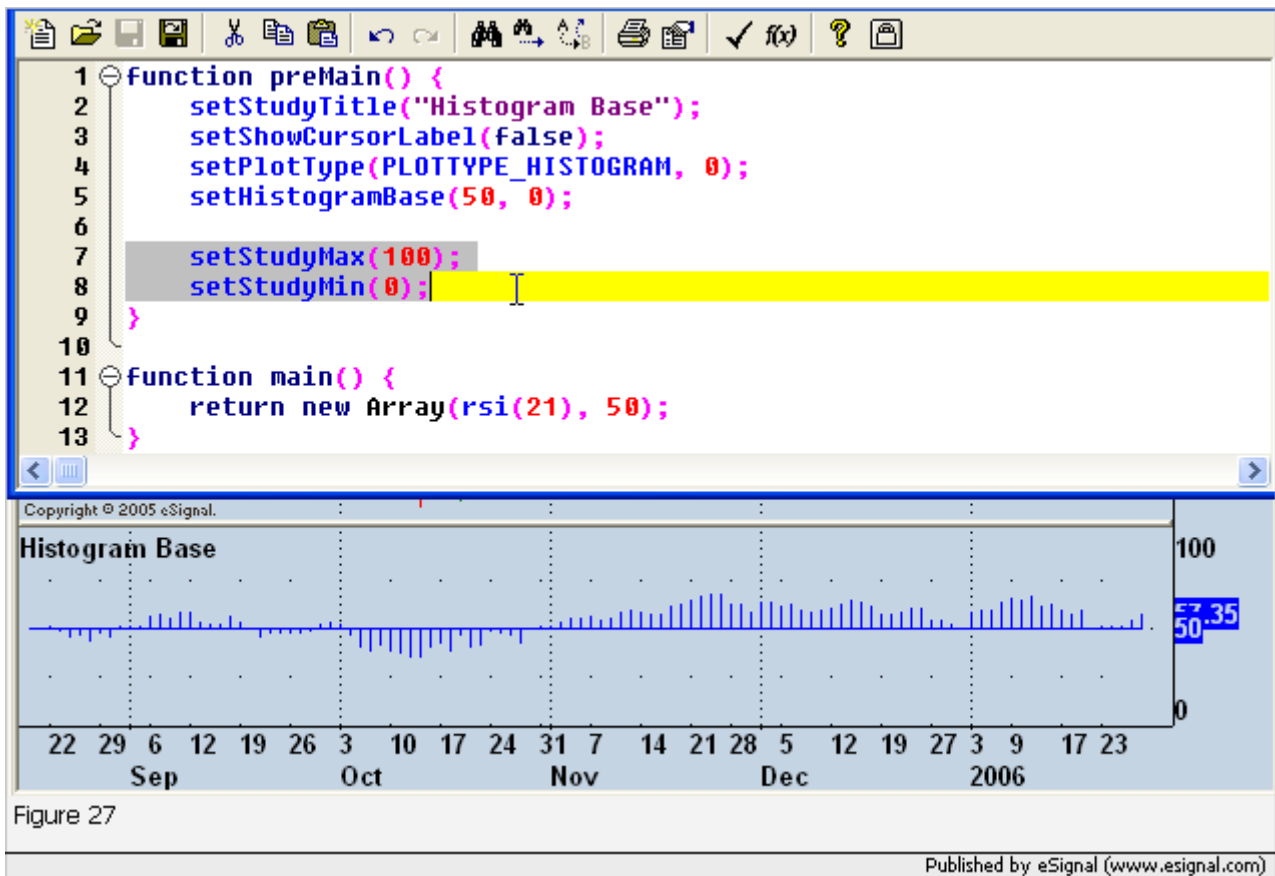
This function sets the histogram base used for data series that are set to plot as histograms. The function requires one parameter (nValue) as an integer and has the optional nIndex parameter for studies that return multiple data series as histograms. If this function is not used in **preMain()**, any histogram data series will use 0 for the histogram base. Figure 26 is a basic code example ([Intro\\_HistogramBase.efs](#)) of our RSI study that is using a value of 50 for the histogram base.



### 3.3.10 [setStudyMax\(nValue\)](#) and [setStudyMin\(nValue\)](#)

These functions set the maximum and minimum y-axis scale for a non-price study. They have one required parameter (nValue) that is expected to be an integer value. Note that these functions do not have the nIndex parameter as there can only be one maximum and minimum value used for a single study. When these functions are not used in **preMain()** the scale for the study will be controlled by the current scaling option selected for the Advanced Chart (Chart Options→Scaling). Continuing with our previous RSI example, Figure 27 sets the maximum and minimum scale to 100 and 0, respectively.





### 3.4 preMain() – Study Parameters (FunctionParameter Class)

In many EFS studies, there is at least one parameter that users would like to be able to customize or adjust without having to modify the parameter in the code through the EFS Editor. This feature is made possible with the use of the **FunctionParameter** object (or class) used in the **preMain()** function. This allows you to set up study parameters that can be edited by a user from the Edit Studies option (Chart Options→Edit Studies) in the Advanced Chart. This section covers the four types of parameters (String, Number, Color and Boolean) and the available configuration options for each type.

The **FunctionParameter** object requires two parameters. The first parameter is a string that is used as the variable name that is passed as an argument within the call operators of **main( arg )**. If a study uses multiple function parameters, the arguments are separated by commas. The order that they will appear in the Edit Studies window is based on their order within argument list for **main()**. The second parameter is expected to be one of the four constants that represent the respective types. The following is the syntax used to initialize an instance of the object. Note that because we are creating an object rather than a simple variable, the **new** operator is used.

```
var fp1 = new FunctionParameter( "paramName", paramType );
```

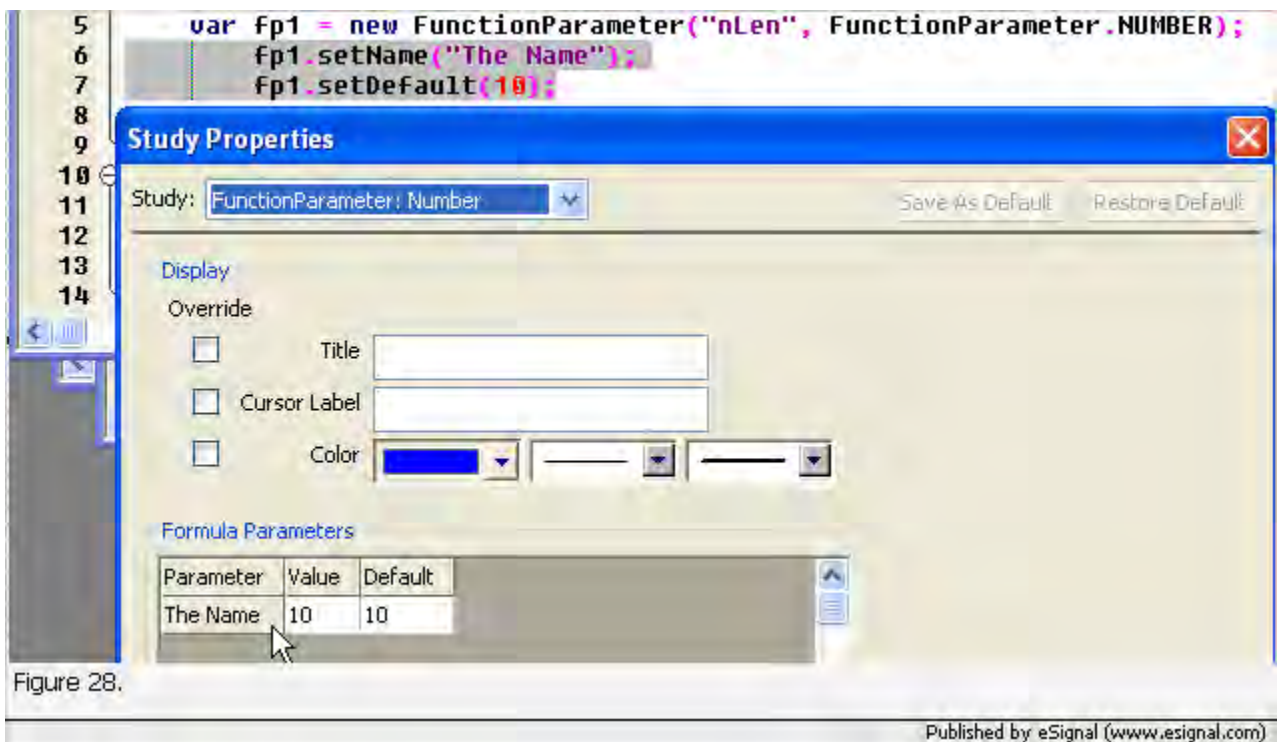
The following lists the available constants used to specify the parameter type.

- **FunctionParameter.NUMBER**
- **FunctionParameter.STRING**
- **FunctionParameter.COLOR**
- **FunctionParameter.BOOLEAN**

The **FunctionParameter** object has several methods that are used to configure the settings and options associated with each parameter type. The following two methods are common to all parameter types. Both methods are optional, however, their usage is highly recommended. The methods that are specific to each type will be covered in the following subsections.

- **setName( sName )** - Sets the description for the parameter that appears in the Edit Studies.
- **setDefault( DefaultValue )** - Sets the default value.

Figure 28 below shows a code example of these methods and what the result looks like in the Study Properties window (Edit Studies menu option).



The Parameter column in the Formula Parameters section lists the names of the formula parameters set by the setName() method. The Value column is where you would enter the new values you want the study to use for the parameters. The Default column displays the default values set by the setDefault() method. Changing the value in the Default column will have no effect on the formula or its parameter settings.

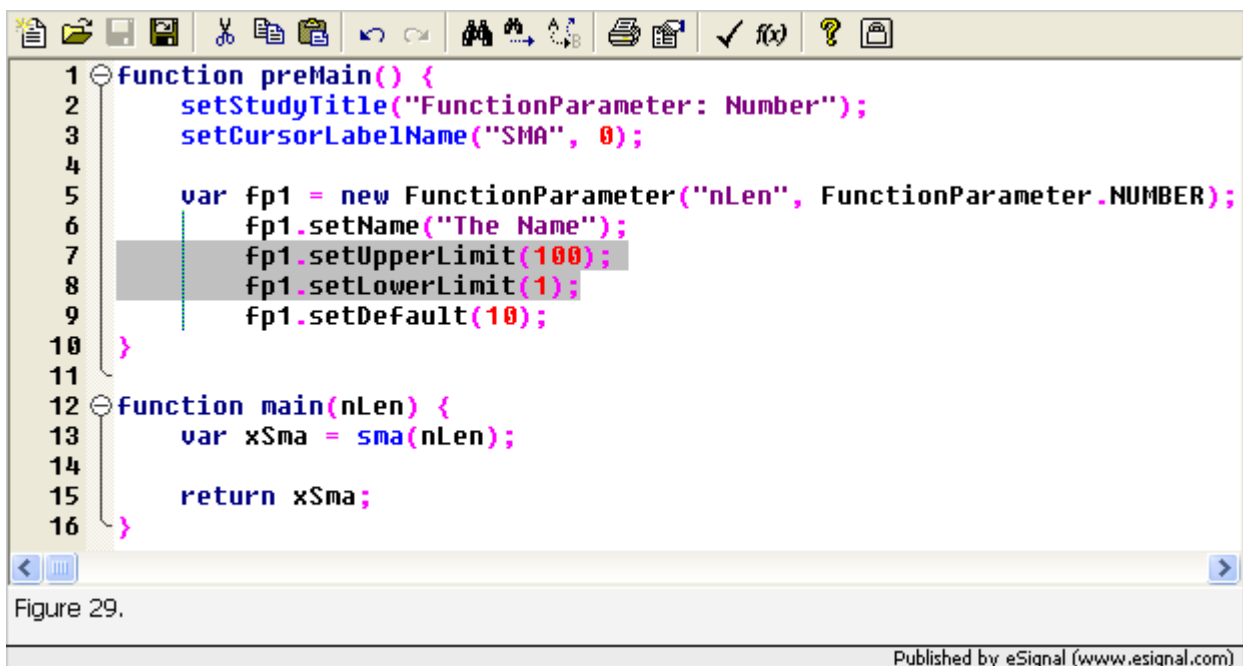
### 3.4.1 Number Parameter

This parameter type is used to pass numbers to **main()** that may be utilized to set any function parameter that requires a number, such as the length parameter for a simple

moving average or the default thickness of a line. Below are the additional methods for this parameter type.

- **setUpperLimit( nNumber )** - Sets the maximum limit that may be entered for the parameter.
- **setLowerLimit( nNumber )** - Sets the minimum limit that may be entered for the parameter.
- **addOption( vOption )** - Adds the specified option value to a drop-down list for the user to choose from.

Typically the limit methods are not used in conjunction with the addOption() method, but they certainly may. Figure 29 is a code example ([Intro\\_FP\\_number.efs](#)) that uses a number parameter to set the length of a moving average. The limit methods are being used to ensure that only a value between 1 and 100 can be entered through Edit Studies.



If a value outside the range set by the limit methods is entered into the Value column in Edit Studies, the following error window will appear indicating an invalid entry for the specified parameter.

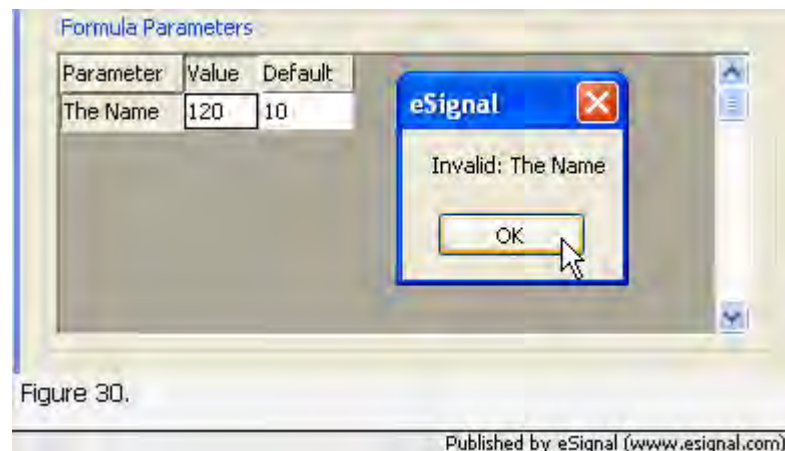


Figure 31 below is our previous code example modified to use the `addOption()` method that creates a list of choices, 10, 20 and 30. However, a value may still be entered manually to use a value other than one of the choices.

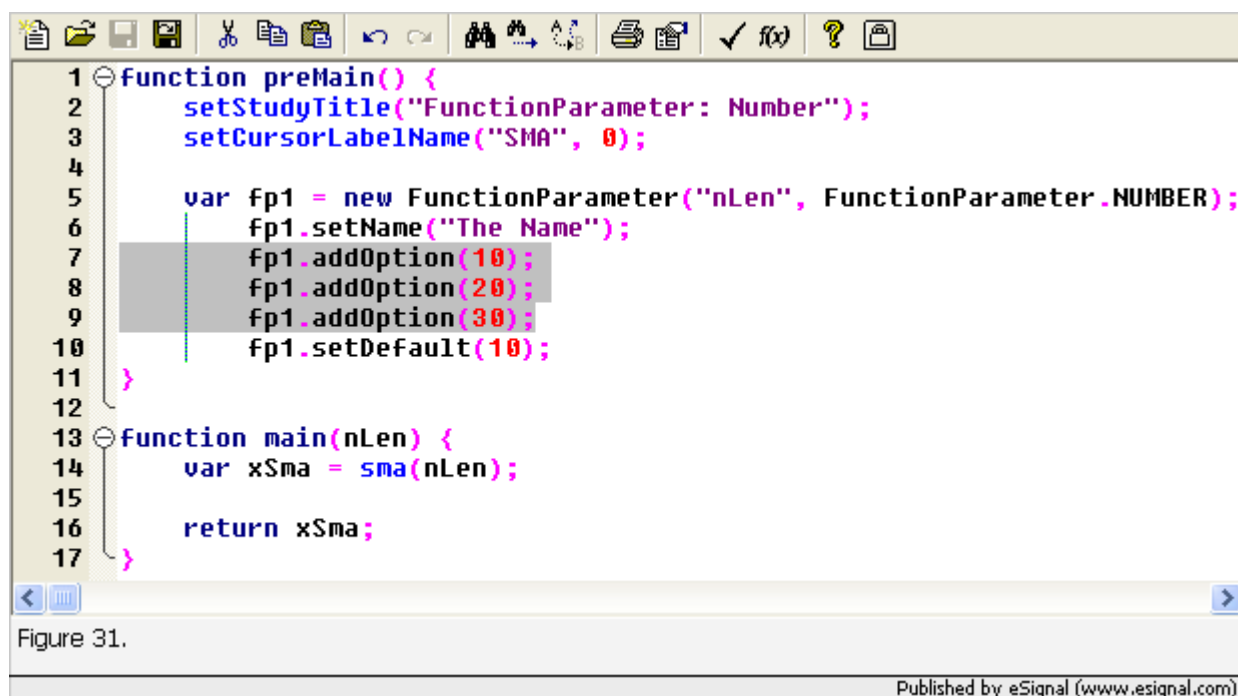
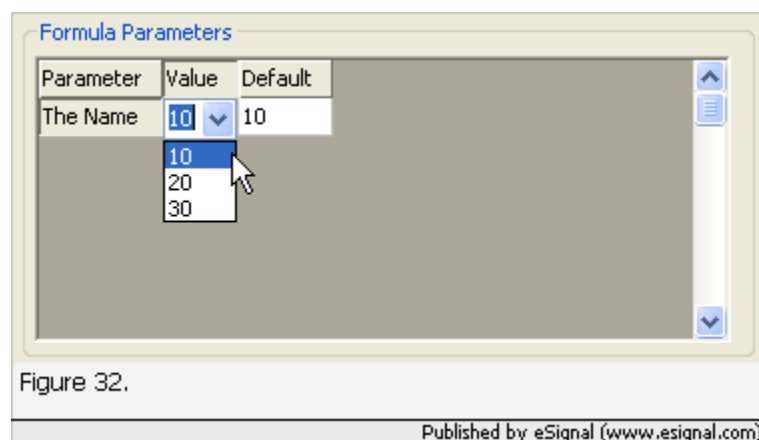
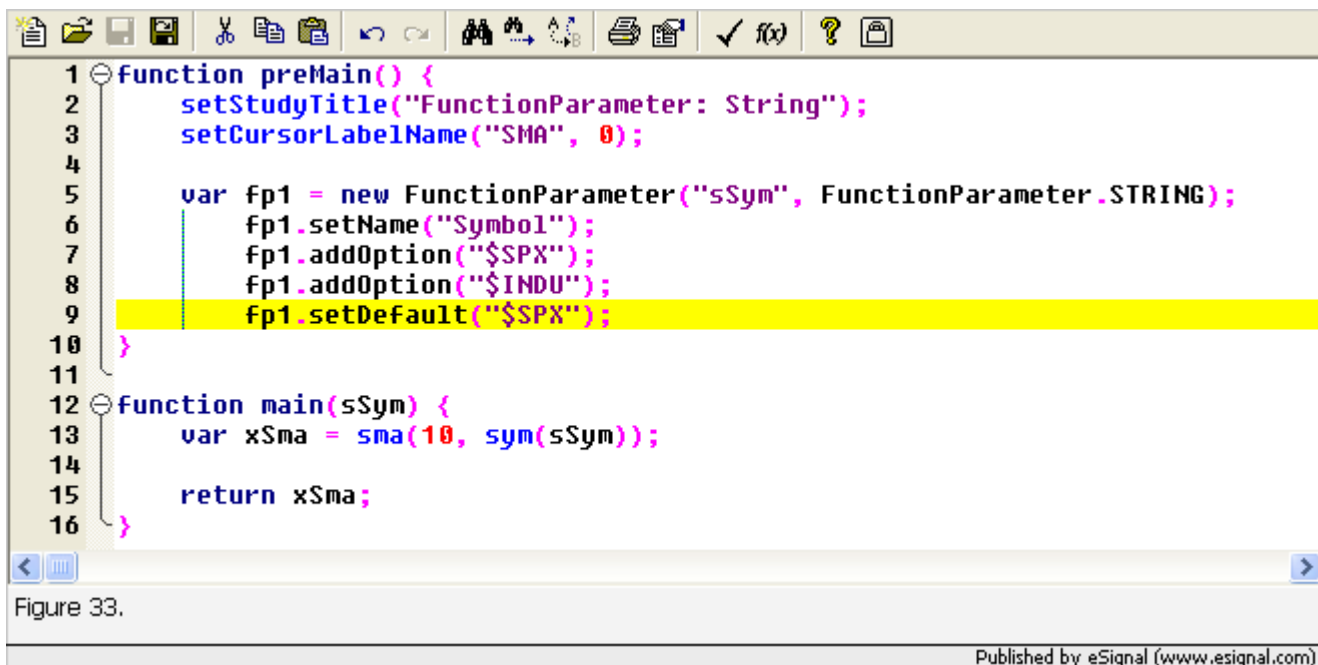


Figure 32 below shows an example of the drop-down list of options this method creates.



### 3.4.2 String Parameter

This parameter type is used to pass strings to **main()** that may be utilized to set any function parameter that requires a string, such as the symbol parameter for studies based on external symbols. The `addOption()` method is the only other optional method for the string parameter type, which is used in the same manner as outlined in the previous section. The only difference is that the values used need to be strings. Figure 33 below is a code example that uses a string parameter ([Intro\\_FP\\_string.efs](#)) to pass a symbol to a moving average study. The result of this code example forces the moving average to be based on the symbol from the formula parameter rather than the current symbol in the Advanced Chart.



### 3.4.3 Color Parameter

This parameter type is used to pass colors to **main()** that may be utilized to set any function parameter that requires a color, such as the default color options previously covered in this tutorial. The **addOption()** method could also be used with this parameter type. Figure 34 below is a code example ([Intro\\_FP\\_color.efs](#)) that creates a formula parameter to allow you to change the default color of a moving average. This code is also using an initialization routine that allows the code within the **blnit** code block to only execute once, which occurs after parameter changes have been applied. Without this routine, the code within that code block would execute on every tick in real time, which is unnecessary. This type of initialization routine is heavily used in EFS development to improve code efficiency.



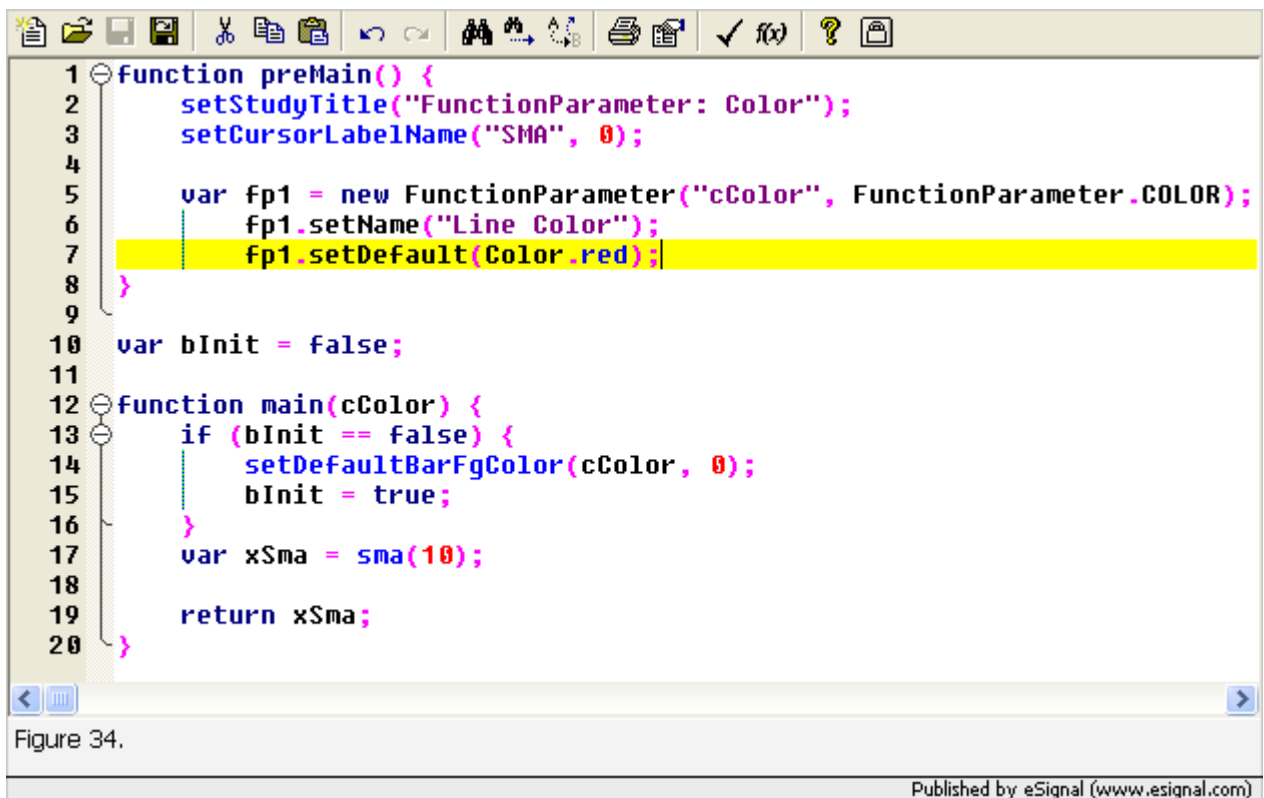
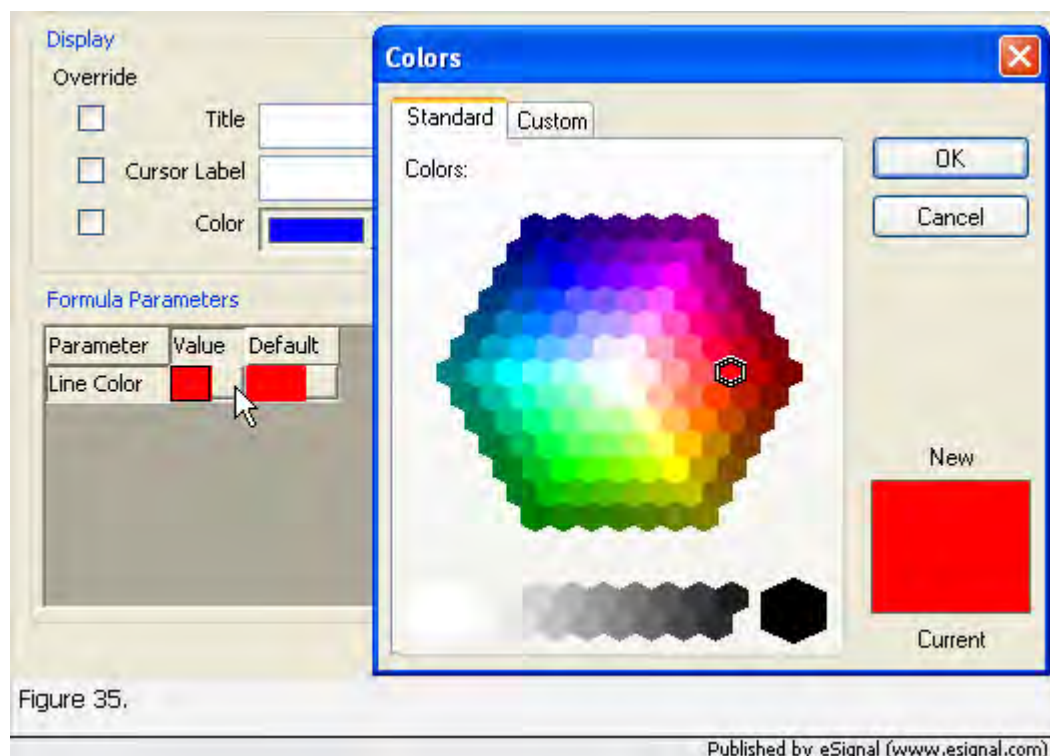


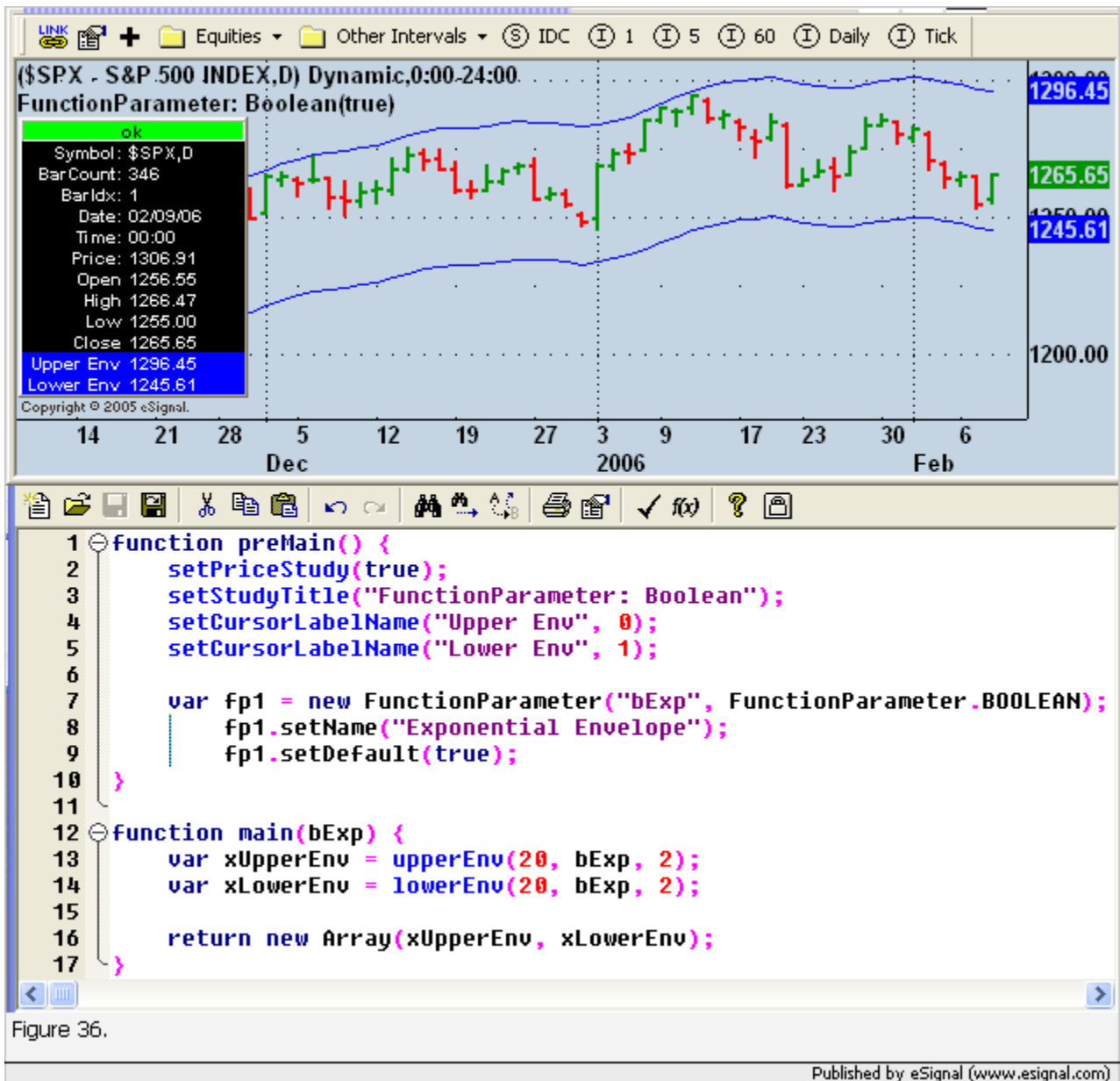
Figure 35 below shows an example of the available color options that appear after you click on the button in the Value column.



### 3.4.4 Boolean Parameter

This parameter type is used to pass Booleans to **main()** that may be utilized to set any function parameter that requires a true/false value, such as the bExponential parameter for the envelope study. This parameter type does not have any additional methods.

Figure 36 below is a code example ([Intro FP boolean.efs](#)) that uses a Boolean parameter to set the envelope study to use the exponential (**true**) or simple (**false**) calculation for the study.



## What's Next?

In the next tutorial, **Introductory Tutorial 4 – Understanding Bar State and Bar Indexing**, we will introduce the bar states and how they are used in formula development. We will also explain how the bars are indexed in the Advanced Chart.