

EFS is the scripting language for the eSignal Advanced Charts product. It is based on Javascript 1.5 (*e.g.*, includes all of the core Javascript 1.5 objects) and provides extensions that are specific to charting, trading and the manipulation of financial data. With EFS you can create custom indicators, trading systems and function libraries. This document is intended to be a reference for users/developers who are interested in creating new EFS scripts or extending the functionality of existing scripts.

EFS2 is the most recent incarnation of the EFS scripting language and it contains a host of new features, not the least of which are enhanced multi-timeframe support and the ability to create custom series and custom function libraries. Note that in this reference guide, all of the functions that are new in EFS2 are designated with a small red star in the topic icon.

Chris Kryza  
Divergence Software, Inc.  
[ckryza@sr-analyst.com](mailto:ckryza@sr-analyst.com)  
[www.sr-analyst.com](http://www.sr-analyst.com)



## EFS Development Resources

The best way to learn EFS is to look at scripts that others have created. A good place to start is in your eSignal Formula Directory on your hard drive! eSignal comes installed with a large number of scripts that can be loaded into your eSignal Formula Editor for review and testing purposes. Don't be intimidated by the large number of functions and objects presented in this reference guide. Even the most exotic EFS scripts typically use only a small fraction of the available functionality.

Another excellent resource is the [eSignalCentral](#) web site. The FileShare area contains a large number of groups that maintain EFS scripts that perform a variety of different tasks and most of these scripts are free for the taking. The Bulletin Board area is where you can go to get your questions answered. Be sure to use the Search feature to search through existing posts. If you have an EFS question, chances are it has been asked (and answered) before.

If you have a script that you need developed but you don't have the time or know-how to code it yourself, post your request up in the Bulletin Board area. If no one there can help you then check out the [3rd Party Developer](#) list for commercial support.

- [eSignalCentral File Share](#)
- [eSignalCentral Bulletin Board](#)
- [eSignal - Latest Version Download](#)

## Javascript Resources

Javascript is a popular language and it is used in many different business applications and environments. There are many free Javascript tutorials and resources available on the web and more are being added every day. If you are interested in learning more about Javascript, just pull up your web browser and do a search for "Javascript Tutorial" or "Javascript Function Reference" and you will find plenty of informational web sites to visit.

## General Information

- EFS is an extension of the Javascript 1.5 language.
- EFS is an interpreted language.
- EFS is a case-sensitive language, so *myData* and *MyData* would be treated as two separate variable objects.
- EFS scripts can be created with any text editor, although it is recommended that you use the built-in eSignal Formula Editor for your programming work.
- EFS scripts are contained in one file (although with the advent of EFS2 it is now possible to call functions stored in an external Function Library).
- EFS scripts are intended to be loaded into eSignal Advanced Charts and, as such, they must exist in a subdirectory of the eSignal Formula Directory, which is generally c:\Program Files\esignal\formulas
- EFS can be used to create indicators that run in the Price Pane or in a separate indicator pane. It can also be used to create function libraries that are, in turn, called by other EFS scripts.
- EFS scripts can be delivered as plain text files or they can be encrypted to protect intellectual property. Entitlement features are also available that allow developers to restrict script usage and/or set up subscription-based activations.

## Layout of a Basic EFS Script

An EFS script generally consists of the following parts:

|                           |  |
|---------------------------|--|
| <i>External variables</i> | variables that are available to all functions within the EFS script. They are essentially global variables (since they are global in scope as far as the script is concerned) however, in EFS the term 'global variable' is reserved for special variables that are used to share information/data between running scripts.  |
| <i>Functions</i>          | these would consist of the required preMain() and main() functions as well as any user-defined functions created as part of the project.   |
| <i>Local variables</i>    | variables that are local in scope to the function in which they are declared.  |
| <i>Comments</i>           | comments and in-line documentation are an important, but often overlooked, programming practice.   |
| <i>Indentation</i>        | proper indentation is yet another very important, but often overlooked, programming practice.  |
| <i>preMain()</i>          | this is the initialization function for any EFS script and is required. The preMain function is generally used to set the script title, define the display properties of any values that will be plotted by the script, define the script parameter menu, check for authorization and entitlement, and set the display characteristics of the script in general. The preMain function is only called once, when the script is first loaded (or refreshed). |
| <i>main()</i>             | this is the workhorse function in any EFS script and is required. It is called by the EFS engine as frequently as each new tick (or as infrequently as each new bar). This is where the core logic of the script is defined and this function also returns the value (or values) back to the chart to be plotted.  |

## Example Script

The following script is an example from the EFS2 folder that you will find in your eSignal Formula Directory. As you can see, it contains all of the pieces/parts described above.

```
*****
Alexis C. Montenegro © January 2005
Use and/or modify this code freely. If you redistribute it
please include this and/or any other comment blocks and a
description of any changes you make.
*****
```

```
var vATR = null;
var fpArray = new Array();

function preMain() {

    var x;

    setStudyTitle("ATR");
    setCursorLabelName("ATR", 0);
    setDefaultBarFgColor(Color.blue, 0);
    setPlotType(PLOTTYPE_LINE,0);
    setDefaultBarThickness(1,0);

    x=0;
    fpArray[x] = new FunctionParameter("Length",
FunctionParameter.NUMBER);
    with(fpArray[x]){
        setLowerLimit(1);
        setDefault(14);
    }
    x++;
    fpArray[x] = new FunctionParameter("Symbol",
FunctionParameter.NUMBER);
    with(fpArray[x]){


```

```

        setDefault("");
    }
    x++;
    fpArray[x] = new FunctionParameter("Interval",
FunctionParameter.NUMBER);
    with(fpArray[x]){
        setDefault("");
    }
}

function main(Length,Symbol,Interval) {

    if(getBuildNumber()<689)
        return;

    if(Symbol==null) Symbol=getSymbol();
    if(Interval==null) Interval=getInterval();
    vSymbol=Symbol+","+Interval;

    if (vATR == null) vATR = atr(Length,sym(vSymbol));

*****
Insert your code following this text block
Use vATR.getValue(0) for your code
*****

    return vATR.getValue(0);
}

```

This section describes the built-in study functions that are available in eSignal EFS:

|   |             |
|---|-------------|
| <a href="#">Accumulation/Distribution</a> | new in EFS2 |
| <a href="#">ADX/DM</a>                    | new in EFS2 |
| <a href="#">Average True Range</a>        | new in EFS2 |
| <a href="#">Bollinger Bands</a>           | new in EFS2 |
| <a href="#">Commodity Channel Index</a>   | new in EFS2 |
| <a href="#">Chop</a>                      | new in EFS2 |
| <a href="#">Donchian Channels</a>         | new in EFS2 |
| <a href="#">Envelope</a>                  | new in EFS2 |
| <a href="#">Linear Regression</a>         | new in EFS2 |
| <a href="#">MACD</a>                      | new in EFS2 |
| <a href="#">Momentum</a>                  | new in EFS2 |
| <a href="#">Money Flow</a>                | new in EFS2 |
| <a href="#">Moving Average</a>            | new in EFS2 |
| <a href="#">On Balance Volume</a>         | new in EFS2 |
| <a href="#">Parabolic SAR</a>             | new in EFS2 |
| <a href="#">Percent R</a>                 | new in EFS2 |
| <a href="#">Price Oscillator</a>          | new in EFS2 |
| <a href="#">Rate of Change</a>            | new in EFS2 |
| <a href="#">RSI</a>                       | new in EFS2 |
| <a href="#">Stochastic</a>                | new in EFS2 |

Note that you can 'nest' studies in EFS2. So you can create constructs such as:

```
myVar = cci( 14, ema( 20, rsi( 5, close() ) ) );
```

which would return the value of a 14-period CCI of a 20-period EMA of a 5-period RSI of the close.

### **accDist( [sym() | inv()] [, barIndex] )**

Accumulation/Distribution looks at both price and volume action to try and determine if more people are buying than selling or vice versa. When there are more buyers than sellers the stock is under accumulation, but when there are more sellers than buyers the stock is under distribution.

#### Parameters

|                       |  |                            |
|-----------------------|--|----------------------------|
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol    |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval  |
| barIndex              | optional. bar index of value to retrieve | default: most recent value |

#### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = accDist();  
        bInit = true;  
  
    }  
}
```

```
//retrieve the current value  
myVar = myStudy.getValue(0);  
  
return( myVar );  
  
}
```

## To Retrieve a Single Value

```
function main() {  
  
    ...  
    ...  
    myVar = accDist();  
  
    //do something with the value in myVar  
  
}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded  
myStudy = accDist( sym( "IBM" ) );  
  
//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = accDist( inv(15) );  
  
//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = accDist( inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded.
myStudy = accDist( sym( "MSFT,30" ) );
```

## [offsetSeries\(\)](#)

**adx( length, smoothing [, sym() | inv()] [, barIndex] )**  
**pdi( length, smoothing [, sym() | inv()] [, barIndex] )**  
**ndi( length, smoothing [, sym() | inv()] [, barIndex] )**

ADX indicates the strength of the overall trend. Typically readings below 20-25 indicate a weak trend. +DI and -DI show the strength of the up trend (+DI) and the strength of the downtrend (-DI) respectively.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| smoothing             | the period to use for the smoothing      |                           |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy1 = null;
var myStudy2 = null;
var myStudy3 = null;
var bInit = false;
```

```
function main() {
var myVar1;
var myVar2;
var myVar3;
```

```

if ( bInit == false ) {

    myStudy1 = adx( 10, 5 );
    myStudy2 = pdi( 10, 12 );
    myStudy3 = ndi( 10, 12 );

    bInit = true;

}

//retrieve the current values
myVar1 = myStudy1.getValue(0);
myVar2 = myStudy2.getValue(0);
myVar3 = myStudy3.getValue(0);

return new Array( myVar1, myVar2, myVar3 );

}

```

## To Retrieve a Single Value

```

function main() {

    ...
    ...
    myVar = adx( 10, 5 );

//do something with the value in myVar

}

```

## Calling Examples

*//create a study that uses IBM as the symbol along with whatever bar*

```
//interval you are using in the chart where the script is loaded
myStudy = adx( 10, 5, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = pdi( 12, 4, inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = ndi( 10, 5, inv("D" ) );

//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded.
myStudy = adx( 21, 5, sym( "MSFT,30" ) );
```

## [offsetSeries\(\)](#)

## atr( length [, sym() | inv()] [, barIndex] )

The Average True Range (ATR) is a measure of volatility. It was introduced by Welles Wilder in his book New Concepts in Technical Trading Systems and has since been used as a component of many indicators and trading systems.

Wilder has found that the high ATR values often occur at market bottoms following a "panic" sell-off. Low ATR values are often found during extended sideways periods, such as those found at tops and after consolidation periods.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;
```

```
function main() {  
    var myVar;  
  
    if ( bInit == false ) {
```

```

myStudy = atr( 12 );
bInit = true;

}

//retrieve the current value
myVar = myStudy.getValue(0);

return( myVar );

}

```

## To Retrieve a Single Value

```

function main() {

    ...
    ...
    myVar = atr( 7 );

//do something with the value in myVar

}

```

## Calling Examples

```

//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = atr( 50, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = atr( 10, inv(15) );

```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = atr( 10, inv("D") );

//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded.
myStudy = atr( 15, sym( "MSFT,30" ) );
```

## offsetSeries()

**upperBB( length, stdDev [, source | sym() | inv()] [, barIndex] )**

**middleBB( length, stdDev [, source | sym() | inv()] [, barIndex] )**

**lowerBB( length, stdDev [, source | sym() | inv()] [, barIndex] )**

Bollinger Bands (created by John Bollinger) are similar to moving average envelopes. The difference between Bollinger Bands and envelopes is that envelopes are plotted at a fixed percentage above and below a moving average, whereas Bollinger Bands are plotted at standard deviation levels above and below a moving average. Since standard deviation is a measure of volatility, the bands are self-adjusting, widen during volatile markets and contract during calmer periods.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| stdDev                | the number of standard deviations        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy1 = null;
```

```

var myStudy2 = null;
var myStudy3 = null;
var bInit = false;

function main() {
  var myVar1;
  var myVar2;
  var myVar3;

  if ( bInit == false ) {

    myStudy1 = upperBB( 20, 2 );
    myStudy2 = middleBB( 20, 2 );
    myStudy3 = lowerBB( 20, 2 );

    bInit = true;
  }

  //retrieve the current values
  myVar1 = myStudy1.getValue(0);
  myVar2 = myStudy2.getValue(0);
  myVar3 = myStudy3.getValue(0);

  return new Array( myVar1, myVar2, myVar3 );
}

}

```

## To Retrieve a Single Value

```

function main() {

  ...
  ...
  myVar = middleBB( 10, 1.5 );
}

```

```
//do something with the value in myVar  
}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded  
myStudy = middleBB( 20, 2, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = upperBB( 20, 2, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = lowerBB( 10, 1.5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded  
myStudy = middleBB( 20, 1, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.  
myStudy = middleBB( 10, 2, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless  
//of the bar interval of the chart in which the script is loaded  
myStudy = upperBB( 12, 2, close( inv(15) ) );
```

## offsetSeries()

## cci( length [, source | sym() | inv()] [, barIndex] )

The Commodity Channel Index (CCI) is a price momentum indicator to measure the price excursions from the mean price as a statistical variation. It is used to detect the beginnings and endings of trends.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = cci( 12 );  
        bInit = true;  
    }  
  
    //retrieve the current value  
    myVar = myStudy.getValue(0);
```

```
    return( myVar );  
  
}
```

## To Retrieve a Single Value

```
function main() {  
  
    ...  
    ...  
    myVar = cci( 7 );  
  
    //do something with the value in myVar  
  
}
```

## Calling Examples

//create a study that uses IBM as the symbol along with whatever bar

//interval you are using in the chart where the script is loaded

```
myStudy = cci( 15, sym( "IBM" ) );
```

//create a study that uses the 15-min bar interval, regardless of the bar

//interval of the chart in which the script is loaded

```
myStudy = cci( 15, inv(15) );
```

//create a study that uses the Daily bar interval, regardless of the bar

//interval of the chart in which the script is loaded

```
myStudy = cci( 5, inv("D") );
```

//create a study that will be based on MSFT 30-min bars, regardless of

//the symbol/bar interval of the chart in which the script is loaded

```
myStudy = cci( 14, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.
myStudy = cci( 15, high() );

//create a study that will be based on the close, using 15-min bars, regardless
//of the bar interval of the chart in which the script is loaded
myStudy = cci( 20, close( inv(15) ) );
```

## offsetSeries()

## **chop( length [, source | sym() | inv()] [, barIndex] )**

Choppiness is a function of market direction. A market that is trending has a low choppiness number while a trend less market has a high choppiness number. The Choppiness Index ranges between 0 and 100, higher the index the choppier the price action is and the lower the index the more trending the price action.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = chop( 12 );  
        bInit = true;  
  
    }  
}
```

```
//retrieve the current value  
myVar = myStudy.getValue(0);  
  
return( myVar );  
  
}
```

## To Retrieve a Single Value

```
function main() {  
  
    ...  
    ...  
    myVar = chop( 7 );  
  
    //do something with the value in myVar  
  
}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded  
myStudy = chop( 15, sym( "IBM" ) );  
  
//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = chop( 15, inv(15) );  
  
//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = chop( 5, inv("D") );  
  
//create a study that will be based on MSFT 30-min bars, regardless of
```

```
//the symbol/bar interval of the chart in which the script is loaded
myStudy = chop( 14, sym( "MSFT,30" ) );

//create a study that will be based on the high.
myStudy = chop( 15, high() );

//create a study that will be based on the close, using 15-min bars, regardless
//the bar interval of the chart in which the script is loaded
myStudy = chop( 20, close( inv(15) ) );
```

## offsetSeries()

**upperDonchian( length [, source | sym() | inv()] [, barIndex] )**

**middleDonchian( length [, source | sym() | inv()] [, barIndex] )**

**lowerDonchian( length [, source | sym() | inv()] [, barIndex] )**

Channels are lines that surround a stock price movement, which help in projecting future price action. The Donchian Channel indicator uses the highest high and the lowest low of a period of time to plot the channel.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| source                | optional. input series for the study     | default: high and low     |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy1 = null;  
var myStudy2 = null;  
var myStudy3 = null;  
var bInit = false;
```

```
function main() {  
    var myVar1;  
    var myVar2;
```

```

var myVar3;

if ( bInit == false ) {

    myStudy1 = upperDonchian( 20 );
    myStudy2 = middleDonchian( 20 );
    myStudy3 = lowerDonchian( 20 );

    bInit = true;

}

//retrieve the current values
myVar1 = myStudy1.getValue(0);
myVar2 = myStudy2.getValue(0);
myVar3 = myStudy3.getValue(0);

return new Array( myVar1, myVar2, myVar3 );

}

```

## To Retrieve a Single Value

```

function main() {

    ...

    ...

    myVar = middleDonchian( 10 );

    //do something with the value in myVar

}

```

## Calling Examples

*//create a study that uses IBM as the symbol along with whatever bar interval you are using in the chart where the script is loaded*  
myStudy = upperDonchian( 20, sym( "IBM" ) );

*//create a study that uses the 15-min bar interval, regardless of the bar interval of the chart in which the script is loaded*  
myStudy = middleDonchian( 10, inv(15) );

*//create a study that uses the Daily bar interval, regardless of the bar interval of the chart in which the script is loaded*  
myStudy = lowerDonchian( 5, inv("D") );

*//create a study that will be based on MSFT 30-min bars, regardless of the symbol/bar interval of the chart in which the script is loaded*  
myStudy = middleDonchian( 14, sym( "MSFT,30" ) );

*//create a study that will be based on the high.*  
myStudy = lowerDonchian( 15, high() );

*//create a study that will be based on the close, using 15-min bars, regardless of the bar interval of the chart in which the script is loaded*  
myStudy = middleDonchian( 20, close( inv(15) ) );

## offsetSeries()

**upperEnv( length, bExponential, percentage [, source | sym() | inv()] [, barIndex] )**

**middleEnv( length, bExponential, percentage [, source | sym() | inv()] [, barIndex])**

**lowerEnv( length, bExponential, percentage [, source | sym() | inv()] [, barIndex] )**

An envelope is composed of two moving averages. One moving average is shifted upward and the second moving average is shifted downward. Envelopes define the upper and lower boundaries of a security's normal trading range. A sell signal is generated when the security reaches the upper band, whereas a buy signal is generated at the lower band. The optimum percentage shift depends on the volatility of the security—the more volatile, the larger the percentage.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| bExponential          | create an exponential average            |                           |
| percentage            | envelope width expressed as a percentage |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```

var myStudy1 = null;
var myStudy2 = null;
var myStudy3 = null;
var bInit = false;

function main() {
  var myVar1;
  var myVar2;
  var myVar3;

  if ( bInit == false ) {

    myStudy1 = upperEnv( 20, false, 10 );
    myStudy2 = middleEnv( 20, false, 10 );
    myStudy3 = lowerEnv( 20, false, 10 );

    bInit = true;
  }

  //retrieve the current values
  myVar1 = myStudy1.getValue(0);
  myVar2 = myStudy2.getValue(0);
  myVar3 = myStudy3.getValue(0);

  return new Array( myVar1, myVar2, myVar3 );
}

}

```

## To Retrieve a Single Value

```
function main() {
```

...

...

```
myVar = middleEnv( 20, false, 10 );  
  
//do something with the value in myVar  
  
}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded  
myStudy = middleEnv( 20, false, 10, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = upperEnv( 20, false, 10, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = lowerEnv( 10, true, 5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded  
myStudy = middleEnv( 12, false, 4, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.  
myStudy = upperEnv( 10, false, 3, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless  
//of the bar interval of the chart in which the script is loaded  
myStudy = lowerEnv( 20, true, 7, close( inv(15) ) );
```

## offsetSeries()

**upperLinearReg( length, stdDev [, source | sym() | inv(), barIndex] )**

**middleLinearReg( length, stdDev [, source | sym() | inv(), barIndex] )**

**lowerLinearReg( length, stdDev [, source | sym() | inv(), barIndex] )**

Linear regression is a statistical tool used to predict future values from past values. In the case of security prices, it is commonly used to determine when prices are overextended. A linear regression trendline uses the least squares method to plot a straight line through prices so as to minimize the distances between the prices and resulting trendline.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| stdDev                | the number of standard deviations        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy1 = null;  
var myStudy2 = null;  
var myStudy3 = null;  
var bInit = false;
```

```

function main() {
  var myVar1;
  var myVar2;
  var myVar3;

  if ( bInit == false ) {

    myStudy1 = upperLinearReg( 20, 2 );
    myStudy2 = middleLinearReg( 20, 2 );
    myStudy3 = lowerLinearReg( 20, 2 );

    bInit = true;

  }

  //retrieve the current values
  myVar1 = myStudy1.getValue(0);
  myVar2 = myStudy2.getValue(0);
  myVar3 = myStudy3.getValue(0);

  return new Array( myVar1, myVar2, myVar3 );

}


```

## To Retrieve a Single Value

```

function main() {

  ...
  ...
  myVar = middleLinearReg( 20, 2 );

  //do something with the value in myVar

```

}

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = middleLinearReg( 20, 2, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = upperLinearReg( 20, 2, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = lowerLinearReg( 10, 1.5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
myStudy = middleLinearReg( 20, 1, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.
myStudy = middleLinearReg( 10, 2, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless
//of the bar interval of the chart in which the script is loaded
myStudy = upperLinearReg( 12, 2, close( inv(15) ) );
```

## offsetSeries()

**sma( length [, source | sym() | inv()] [, barIndex] )**  
**ema( length [, source | sym() | inv()] [, barIndex] )**  
**wma( length [, source | sym() | inv()] [, barIndex] )**  
**vwma( length [, source | sym() | inv()] [, barIndex] )**

A Moving Average is an indicator that shows the average value of a security's price over a period of time. When calculating a moving average, a mathematical analysis of the security's average value over a predetermined time period is made. As the security's price changes, its average price moves up or down.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | the period to use for the calculation    |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy1 = null;  
var myStudy2 = null;  
var myStudy3 = null;  
var bInit = false;  
  
function main() {  
    var myVar1;
```

```

var myVar2;
var myVar3;

if ( bInit == false ) {

    myStudy1 = ema( 20 );
    myStudy2 = sma( 10 );
    myStudy3 = wma( 40 );

    bInit = true;

}

//retrieve the current values
myVar1 = myStudy1.getValue(0);
myVar2 = myStudy2.getValue(0);
myVar3 = myStudy3.getValue(0);

return new Array( myVar1, myVar2, myVar3 );

}

```

## To Retrieve a Single Value

```

function main() {

    ...
    ...
    myVar = sma( 10 );

    //do something with the value in myVar

}

```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = sma( 20, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = ema( 10, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = wma( 5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
myStudy = vwma( 14, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.
myStudy = ema( 15, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless
//the bar interval of the chart in which the script is loaded
myStudy = sma( 20, close( inv(15) ) );
```

## offsetSeries()

**macd( fastLength, slowLength, smoothing [, source |  
sym() | inv()] [, barIndex] )**  
**macdSignal( fastLength, slowLength, smoothing [, sou  
| sym() | inv()] [, barIndex] )**  
**macdHist( fastLength, slowLength, smoothing [, sourc  
sym() | inv()] [, barIndex] )**

MACD is short for Moving Average Convergence Divergence. The MACD looks at the difference between a short-term moving average and a long-term moving average. A third moving average is taken of the difference and used as a signal line.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| fastLength            | the fast MACD period                     |                           |
| slowLength            | the slow MACD period                     |                           |
| smoothing             | the MACD smoothing period                |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy1 = null;  
var myStudy2 = null;  
var myStudy3 = null;  
var bInit = false;
```

```

function main() {
    var myVar1;
    var myVar2;
    var myVar3;

    if ( bInit == false ) {

        myStudy1 = macd( 12, 26, 9 );
        myStudy2 = macdSignal( 12, 26, 9 );
        myStudy3 = macdHist( 12, 26, 9 );

        bInit = true;

    }

    //retrieve the current values
    myVar1 = myStudy1.getValue(0);
    myVar2 = myStudy2.getValue(0);
    myVar3 = myStudy3.getValue(0);

    return new Array( myVar1, myVar2, myVar3 );

}

```

## To Retrieve a Single Value

```

function main() {

    ...

    ...

    myVar = macd( 12, 26, 9 );

    //do something with the value in myVar
}

```

}

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = macd( 12, 26, 9, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = macdSignal( 12, 26, 9, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = macdHist( 10, 18, 5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
myStudy = macd( 12, 26, 9, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.
myStudy = macd( 10, 20, 4, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless
//of the bar interval of the chart in which the script is loaded
myStudy = macdHist( 12, 26, 9, close( inv(15) ) );
```

## offsetSeries()

## moneyFlow( length [, source | sym() | inv()] [, barIndex]

The Money Flow Index (MFI) is a momentum indicator that measures strength of money flowing in and out of a security. It is related to the Relative Strength Index (RSI) but whereas the RSI incorporates only prices, the MFI accounts for volume. Instead of using "up closes" versus "down closes," money flow compares the current interval's average price to the previous interval's average price and then weighs the average price by volume to calculate money flow. The ratio of the summed positive and negative money flows is then normalized to a scale of 0 to 100.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: hlc3             |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = moneyFlow( 20 );  
        bInit = true;  
    }  
}
```

```
}

//retrieve the current value
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...
    ...
    myVar = moneyFlow( 14 );

    //do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = moneyFlow( 15, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = moneyFlow( 15, inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
```

```
myStudy = moneyFlow( 5, inv("D") );  
  
//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded  
myStudy = moneyFlow( 14, sym( "MSFT,30" ) );  
  
//create a study that will be based on the high.  
myStudy = moneyFlow( 15, high() );  
  
//create a study that will be based on the close, using 15-min bars, regardless  
//the bar interval of the chart in which the script is loaded  
myStudy = moneyFlow( 20, close( inv(15) ) );
```

## [offsetSeries\(\)](#)

## **mom( length [, source | sym() | inv()] [, barIndex] )**

Calculated by subtracting a price for a given interval in the past from the current price. You determine how many intervals to go back for the price that is subtracted. The momentum study is an indication of overbuying or overselling. A positive value indicates overbuying, while a negative value indicates overselling.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = mom( 20 );  
        bInit = true;  
  
    }  
}
```

```
//retrieve the current value  
myVar = myStudy.getValue(0);  
  
return( myVar );  
  
}
```

## To Retrieve a Single Value

```
function main() {  
  
    ...  
    ...  
    myVar = mom( 14 );  
  
    //do something with the value in myVar  
  
}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded  
myStudy = mom( 15, sym( "IBM" ) );  
  
//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = mom( 15, inv(15) );  
  
//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded  
myStudy = mom( 5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
myStudy = mom( 14, sym( "MSFT,30" ) );
```

```
//create a study that will be based on the high.
myStudy = mom( 15, high() );
```

```
//create a study that will be based on the close, using 15-min bars, regardless
//the bar interval of the chart in which the script is loaded
myStudy = mom( 20, close( inv(15) ) );
```

## [offsetSeries\(\)](#)

## **obv( [, sym() | inv()] [, barIndex] )**

On Balance Volume (OBV) is a cumulative total of volume. It shows if volume is flowing into or out of a security. When an issue closes higher than its previous close, all of the day's volume is considered to be up-volume. When the issue closes lower than the previous close, all of the day's volume is considered to be down-volume.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = obv();  
        bInit = true;  
  
    }  
  
    //retrieve the current value
```

```
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...

    ...

    myVar = obv();

    //do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = obv( sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = obv( inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = obv( inv("D") );

//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded.
```

```
myStudy = obv( sym( "MSFT,30" ) );
```

## offsetSeries()

**osc( fastLength, slowLength, bExponential [, source | sym() | inv()] [, barIndex] )**

The Price Oscillator displays the difference between two moving averages of a security's price. The difference between the moving averages can be expressed in either points or percentages.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| fastLength            | fast period for the oscillator           |                           |
| slowLength            | slow period for the oscillator           |                           |
| bExponential          | create an exponential average            |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = osc( 10, 21, false );  
        bInit = true;  
    }  
}
```

```
}

//retrieve the current value
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...
    ...
    myVar = osc( 10, 21, false );

//do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = osc( 10, 21, false, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = osc( 5, 10, false, inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
```

```
myStudy = osc( 15, 31, true, inv("D") );  
  
//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded  
myStudy = osc( 10, 21, false, sym( "MSFT,30" ) );  
  
//create a study that will be based on the high.  
myStudy = osc( 10, 21, false, high() );  
  
//create a study that will be based on the close, using 15-min bars, regardless  
//the bar interval of the chart in which the script is loaded  
myStudy = osc( 10, 21, true, close( inv(15) ) );
```

## offsetSeries()

**sar( start, increment, max [, source | sym() | inv()] [, barIndex] )**

Parabolic SAR is a display of "Stop and Reverse" points for a particular market. When the market touches or crosses a point, this indicates you should reverse your position. If you are long, for example, go short. If you are short, go long. The Parabolic SAR assumes that you are always in the market.

## Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| start                 | starting sar value                       |                           |
| increment             | acceleration factor                      |                           |
| max                   | max sar value                            |                           |
| source                | optional. input series for the study     | default: high and low     |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

## To Create a Series

```
var myStudy1 = null;  
var bInit = false;
```

```
function main() {  
    var myVar;  
  
    if ( bInit == false ) {
```

```

myStudy1 = sar( 0.02, 0.02, 0.2 );

bInit = true;

}

//retrieve the current value
myVar = myStudy1.getValue(0);

return( myVar );
}

```

## To Retrieve a Single Value

```

function main() {

    ...
    ...
    myVar = sar( 0.02, 0.02, 0.2 );

    //do something with the value in myVar

}

```

## Calling Examples

```

//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = sar( 0.02, 0.02, 0.2, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = sar( 0.02, 0.02, 0.2, inv(15) );

```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = sar( 0.02, 0.02, 0.2, inv("D") );

//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
myStudy = sar( 0.02, 0.02, 0.2, sym( "MSFT,30" ) );

//create a study that will be based on the high.
myStudy = sar( 0.02, 0.02, 0.2, high() );

//create a study that will be based on the close, using 15-min bars, regardless
//of the bar interval of the chart in which the script is loaded
myStudy = sar( 0.02, 0.02, 0.2, close( inv(15) ) );
```

## offsetSeries()

## **percentR( length [, source | sym() | inv()] [, barIndex] )**

Williams' %R (pronounced "percent R") is a momentum indicator that measures overbought/oversold levels. It was developed by Larry Williams.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = percentR( 14 );  
        bInit = true;  
  
    }  
  
    //retrieve the current value
```

```
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...

    ...

    myVar = percentR( 14 );

    //do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = percentR( 15, sym( "IBM" ) );
```

```
//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = percentR( 15, inv(15) );
```

```
//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = percentR( 5, inv("D") );
```

```
//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
```

```
myStudy = percentR( 14, sym( "MSFT,30" ) );  
  
//create a study that will be based on the high.  
myStudy = percentR( 15, high() );  
  
//create a study that will be based on the close, using 15-min bars, regardless  
//of the bar interval of the chart in which the script is loaded  
myStudy = percentR( 20, close( inv(15) ) );
```

## offsetSeries()

## roc( length [, source | sym() | inv()] [, barIndex] )

The Rate-of-Change (ROC) indicator displays the difference between the current price and the price  $x$  time periods ago. The difference can be displayed either in points or as a percentage. The momentum indicator displays the same information but expresses it as a ratio.

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = roc( 20 );  
        bInit = true;  
  
    }  
  
    //retrieve the current value
```

```
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...

    ...

    myVar = roc( 14 );

    //do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = roc( 15, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = roc( 15, inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = roc( 5, inv("D") );

//create a study that will be based on MSFT 30-min bars, regardless of
//the symbol/bar interval of the chart in which the script is loaded
```

```
myStudy = roc( 14, sym( "MSFT,30" ) );  
  
//create a study that will be based on the high.  
myStudy = roc( 15, high() );  
  
//create a study that will be based on the close, using 15-min bars, regardless  
//of the bar interval of the chart in which the script is loaded  
myStudy = roc( 20, close( inv(15) ) );
```

## offsetSeries()

## rsi( length [, source | sym() | inv()] [, barIndex] )

The Relative Strength Index (RSI) is designed to indicate a market's current strength or weakness depending on where prices close during a given period. It is based on the premise that higher closes indicate strong markets and lower closes indicate weak markets. The RSI is displayed as three lines, the RSI and two moving averages of the RSI. The RSI is calculated by finding the percentage of positive closes (the current close is higher than the previous close) to negative closes (the current close is lower than the previous close).

### Parameters

|                       |  |                           |
|-----------------------|--|---------------------------|
| length                | period to use for the calculation        |                           |
| source                | optional. input series for the study     | default: close            |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol   |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval |
| barIndex              | optional. bar index of value to retrieve | default: current value    |

### To Create a Series

```
var myStudy = null;  
var bInit = false;  
  
function main() {  
    var myVar;  
  
    if ( bInit == false ) {  
  
        myStudy = rsi( 14 );  
        bInit = true;  
    }  
}
```

```
}

//retrieve the current value
myVar = myStudy.getValue(0);

return( myVar );

}
```

## To Retrieve a Single Value

```
function main() {

    ...
    ...
    myVar = rsi( 14 );

//do something with the value in myVar

}
```

## Calling Examples

```
//create a study that uses IBM as the symbol along with whatever bar
//interval you are using in the chart where the script is loaded
myStudy = rsi( 15, sym( "IBM" ) );

//create a study that uses the 15-min bar interval, regardless of the bar
//interval of the chart in which the script is loaded
myStudy = rsi( 15, inv(15) );

//create a study that uses the Daily bar interval, regardless of the bar
//interval of the chart in which the script is loaded
```

```
myStudy = rsi( 5, inv("D") );  
  
//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded  
myStudy = rsi( 14, sym( "MSFT,30" ) );  
  
//create a study that will be based on the high.  
myStudy = rsi( 15, high() );  
  
//create a study that will be based on the close, using 15-min bars, regardless  
//the bar interval of the chart in which the script is loaded  
myStudy = rsi( 20, close( inv(15) ) );
```

## [offsetSeries\(\)](#)

**stochK( kLength, kSmoothing, dLength [, source | sym  
inv()] [, barIndex] )**

**stochD( kLength, kSmoothing, dLength [, source | sym  
inv()] [, barIndex] )**

The Stochastic is designed to indicate when the market is overbought or oversold. It is based on the premise that when a market's price increases, the closing prices tend to move toward the daily highs and, conversely, when a market's price decreases, the closing prices move toward the daily lows. A Stochastic displays two lines, %K and %D. %K is calculated by finding the highest and lowest point in a trading period and then finding where the current close is in relation to that trading range, %K is then smoothed with a moving average. %D is a moving average of %K.

### Parameters

|                       |  |                              |
|-----------------------|--|------------------------------|
| kLength               | period for fastK calculation             |                              |
| kSmoothing            | %K smoothing period                      |                              |
| dLength               | period for fastD calculation             |                              |
| source                | optional. input series for the study     | default: high, low and close |
| <a href="#">sym()</a> | optional. specify a symbol to use        | default: current symbol      |
| <a href="#">inv()</a> | optional. specify a bar interval to use  | default: current interval    |
| barIndex              | optional. bar index of value to retrieve | default: current interval    |

### To Create a Series

```
var myStudy1 = null;
var myStudy2 = null;
```

```

var bInit = false;

function main() {
  var myVar1;
  var myVar2;

  if ( bInit == false ) {

    myStudy1 = stochK( 14, 1, 3 );
    myStudy2 = stochD( 14, 1, 3 );

    bInit = true;

  }

  //retrieve the current values
  myVar1 = myStudy1.getValue(0);
  myVar2 = myStudy2.getValue(0);

  return new Array( myVar1, myVar2 );

}

}

```

## To Retrieve a Single Value

```

function main() {

  ...

  ...

  myVar = stochD( 14, 3, 5 );

  //do something with the value in myVar

}

}

```

## Calling Examples

//create a study that uses IBM as the symbol along with whatever bar  
//interval you are using in the chart where the script is loaded

```
myStudy = stochasticK( 14, 1, 3, sym( "IBM" ) );
```

//create a study that uses the 15-min bar interval, regardless of the bar  
//interval of the chart in which the script is loaded

```
myStudy = stochasticD( 14, 1, 3, inv(15) );
```

//create a study that uses the Daily bar interval, regardless of the bar  
//interval of the chart in which the script is loaded

```
myStudy = stochasticK( 5, 3, 3, inv("D") );
```

//create a study that will be based on MSFT 30-min bars, regardless of  
//the symbol/bar interval of the chart in which the script is loaded

```
myStudy = stochasticK( 14, 2, 5, sym( "MSFT,30" ) );
```

//create a study that will be based on the high.

```
myStudy = stochasticD( 15, 3, 5, high() );
```

//create a study that will be based on the close, using 15-min bars, regardless  
//the bar interval of the chart in which the script is loaded

```
myStudy = stochasticK( 14, 1, 3, close( inv(15) ) );
```

## offsetSeries()

This section describes the series manipulation functions that are available in eSignal EFS:

[sym\(\)](#) new in EFS2

[inv\(\)](#) new in EFS2

[efs\(\)](#) new in EFS2

[efsInternal\(\)](#) new in EFS2

[efsExternal\(\)](#) new in EFS2

[offsetSeries\(\)](#) new in EFS2

[getSeries\(\)](#) new in EFS2

[setIntervalsBackfill\(\)](#) new in EFS2

[open\(\)](#) revised in EFS2

[high\(\)](#) revised in EFS2

[low\(\)](#) revised in EFS2

[close\(\)](#) revised in EFS2

[oi\(\)](#) revised in EFS2

[volume\(\)](#) revised in EFS2

[year\(\)](#) revised in EFS2

[month\(\)](#) revised in EFS2

[day\(\)](#) revised in EFS2

[hour\(\)](#) revised in EFS2

[minute\(\)](#) revised in EFS2

[second\(\)](#) revised in EFS2

[rawtime\(\)](#) new in EFS2

[getYear\(\)](#)

[getMonth\(\)](#)

[getDay\(\)](#)

[getHour\(\)](#)

[getMinute\(\)](#)

[getSecond\(\)](#)

[getValue\(\)](#)

[getValueAbsolute\(\)](#)

[getMostRecentAsk\(\)](#)

[getMostRecentAskSize\(\)](#)

[getMostRecentBid\(\)](#)

[getMostRecentBidSize\(\)](#)

[getMostRecentTrade\(\)](#)

[getMostRecentTradeSize\(\)](#)

[getBarState\(\)](#)

[getBarStateInterval\(\)](#) new in EFS2

[getBarStateSymbol\(\)](#) new in EFS2

[getCurrentBarIndex\(\)](#)

[getNewestBarIndex\(\)](#)

[getOldestBarIndex\(\)](#)

[getFirstBarIndexOfDay\(\)](#)

[getPreviousTradingDay\(\)](#)

[getNextTradingDay\(\)](#)

[isLastBarOnChart\(\)](#)

[getNumBars\(\)](#)

[getInterval\(\)](#)

[getSymbol\(\)](#)

|                                   |             |
|-----------------------------------|-------------|
| <a href="#"><u>highest()</u></a>  | new in EFS2 |
| <a href="#"><u>lowest()</u></a>   | new in EFS2 |
| <a href="#"><u>hhv()</u></a>      | new in EFS2 |
| <a href="#"><u>llv()</u></a>      | new in EFS2 |
| <a href="#"><u>midpoint()</u></a> | new in EFS2 |

## **getMostRecentAsk()**

Returns the most recent ask price. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the most recent ask price  
    myVar = getMostRecentAsk();  
  
}
```

## **getMostRecentAskSize()**

Returns the size traded at the most recent ask price. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the size at the most recent Ask price  
    myVar = getMostRecentAskSize();  
  
}
```

## **getMostRecentBid()**

Returns the most recent bid price. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    ...  
    ...  
    //retrieve the most recent bid price  
    myVar = getMostRecentBid();  
  
}
```

## **getMostRecentBidSize()**

Returns the size traded at the most recent bid price. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the size at the most recent Bid price  
    myVar = getMostRecentBidSize();  
  
}
```

## **getMostRecentTrade()**

Returns the most recent trade price. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    ...  
    ...  
    //retrieve the value of the most recent trade  
    myVar = getMostRecentTrade();  
}
```

## **getMostRecentTradeSize()**

Returns the size traded at the most recent trade. Note that this function can only be used in realtime. It will NOT return historical values.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the size of the most recent trade  
    myVar = getMostRecentTradeSize();  
  
}
```

## **sym( symbol )**

New in EFS2. Converts string representation of symbol or symbol,interval into a Series object. It is used in conjunction with all other series-type functions and is an important aspect of EFS2's multi-timeframe support.

### Parameters

|        |  |
|--------|--|
| symbol | the symbol or symbol,interval to convert |
|--------|--|

### Usage

```
function main() {
var myVar;

...
...

//get the open price from the prior bar (at the current bar interval)
//for QQQ
myVar = open( -1, sym("QQQ") );

//get the current close from the IBM 15-min series
myVar = close( 0, sym("IBM,15") );

//get the ema value (at the current bar interval) from the MSFT series
myVar = ema( 10, sym("MSFT") );

//get the atr from the SP Emini 5-min series
```

```
myVar = attr( 5, sym("ES #F,5" ) );
```

```
}
```

## inv( interval )

New in EFS2. Converts a bar interval into a Series object. It is used in conjunction with all other series-type functions and is an important aspect of EFS2's multi-timeframe support.

### Parameters

|          |                         |
|----------|-------------------------|
| interval | the interval to convert |
|----------|-------------------------|

### Usage

Note that when passing a non-numeric interval value to inv(), you must enclose the value in quotes. Examples would be inv("D") or inv("W").

```
function main() {
    var myVar;

    ...
    ...

    //get the open price from the daily series of the symbol
    //we are charting
    myVar = open( 0, inv("D") );

    //get the 25-bar ema value from the 5-min bar interval of the
    //symbol we are charting
    myVar = ema( 25, inv(5) );
```

```
//calculate a 10-bar ema of the 14-bar RSI from the 1-min
//bar interval of the symbol we are charting
myVar = ema( 10, rsi( 14, inv(1) ) );
}

}
```

## **efs( pathToEFS, [, seriesIndex] [, sym/inv] [, parameters] )**

New in EFS2. Returns a series object representing a specific seriesIndex value as calculated from the given external EFS script.

### Parameters

|             |  |
|-------------|--|
| pathToEFS   | the path and filename of the EFS script to call  |
| seriesIndex | optional. index into array of return values. defaults to 0.  |
| sym/inv     | optional. if used, the external EFS will be loaded into the specified symbol or interval's context |
| parameters  | optional. parameters, if any, required by the EFS script you are calling                           |

### Usage

```
function main() {
```

```
var myVar;
```

```
...
```

```
...
```

```
//call an external EFS called "myCustomEFS.efs" and load it into  
//the context of IBM 15-min bars. we also pass two parameters  
//to the script.
```

```
myVar = efs( "myCustomEFS.efs", 0, sym( "IBM,15" ), 20, 5 );
```

}

[efsExternal\(\)](#)

## **efsInternal( functionName [, parameters... ] )**

New in EFS2. Returns a series object representing the values calculated from a given function name. The function specified will be executed as if it were a main() function being called in an external EFS. This is conceptually the same as the efs() function, the difference being that instead of calling an external file's main() function, the engine will call the user-defined function instead.

**Note:** If you pass a series object to the efsInternal() function, be sure to pass the series as the last parameter. This will force efsInternal() to execute in the series' sym/inv context.

### Parameters

|              |   |
|--------------|---|
| functionName | name of user-defined function to call   |
| parameters   | optional. parameters, if any, required by the user-defined function you are calling |

### Usage

```
function main() {  
    var myVar;
```

...

...

*//call a user-defined function as an input source to an ema*

```
myVar = ema( 20, efsInternal( "myCustomFunction", 5, close() ) );  
  
}  
  
//user-defined function. our function accepts two parameters and  
//note that the series is passed as the last parameter.  
function myCustomFunction( length, source ) {  
    return( source.getValue(0) - source.getValue(-length) );  
}
```

## **efsExternal( pathToEFS [, sym/inv] [, parameters] )**

New in EFS2. Returns a series object representing the values as calculated from the given external EFS script. You would then use the getSeries() function to extract a specific series value if multiple series are returned.

### Parameters

|            |  |
|------------|--|
| pathToEFS  | the path and filename of the EFS script to call  |
| sym/inv    | optional. if used, the external EFS will be loaded into the specified symbol or interval's context |
| parameters | optional. parameters, if any, required by the EFS script you are calling                           |

### Usage

```
function main() {  
    var myVar;  
    var myPlot;
```

...

...

```
//call an external EFS called "myCustomEFS.efs" and load it into  
//the context of IBM 15-min bars. we also pass two parameters  
//to the script.  
myVar = efsExternal( "myCustomEFS.efs", sym( "IBM,15" ), 20, 5 );
```

*//assuming that "myyCustomEFS.efs" returned 3 values, we would use  
//the following logic to retrieve just the 2nd return value. The first  
//value returned would be seriesIndex 0, second value would be  
//seriesIndex 1, etc.*

```
var myPlot = getSeries( myVar, 1 );
```

```
}
```

efs()

## **offsetSeries( series, offset )**

New in EFS2. This function is used to offset into the past (negative offset) or into the future (positive offset). It is used primarily in conjunction with the [built-in study functions](#). Note that the offset will be applied in the context of the bar interval used in the chart rather than the bar interval being used in the study being offset.

### Parameters

|        |  |
|--------|--|
| series | the series to offset                                   |
| offset | the positive or negative offset to apply to the series |

### Usage

```
var myStudy1 = null;  
var myStudy2 = null;  
var bInit = false;  
  
function main() {  
    var myVar1;  
    var myVar2;  
  
    if ( bInit == false ) {  
  
        //shift the ema 10 bars into the past  
        myStudy1 = offsetSeries( ema( 20 ), -10 );  
  
        //shift the sma 10 bars into the future  
        myStudy2 = offsetSeries( sma( 10 ), 10 );  
    }  
}
```

```
bInit = true;  
}  
  
//retrieve the current values  
myVar1 = myStudy1.getValue(0);  
myVar2 = myStudy2.getValue(0);  
  
return new Array( myVar1, myVar2 );  
}
```

## **getSeries( returnVal, seriesIndex )**

New in EFS2. This function converts the return value of a data series back into a series object.

### Parameters

|             |   |
|-------------|---|
| returnVal   | the return value to convert                                     |
| seriesIndex | optional. index into array of return values. default value is 0 |

### Usage

```
var myStudy1 = null;  
var myStudy2 = null;  
var bInit = false;  
  
function main() {  
  
    if ( bInit == false ) {  
  
        //create a 20-period EMA using 30-min bars  
        myStudy1 = ema( 20, inv(30) );  
  
        //create a 20-period EMA using 60-min bars  
        myStudy2 = ema( 20, inv(60) );  
  
        bInit = true;  
    }  
}
```

```
//use getSeries() to assure that the actual series is being returned, rather
than the values
return new Array( getSeries( myStudy1 ), getSeries( myStudy2 ) );
}
```

## **setIntervalsBackfill( boolean )**

New in EFS2. This function is used in preMain() in conjunction with multiple time-frame studies. When a custom time template is used and it has different intervals set to load a different number of days (or bars) of data, the study uses the number of days specified for the current chart interval. This amount of data could cover more days than the external interval being requested. The result may not plot the external interval-based indictor back to the beginning of the current chart's data. Using this function will force the study to use the same number of days as the main chart interval, which will backfill the plot for external interval-based indicator to the beginning of the chart's data.

### Parameters

|         |                                      |
|---------|--------------------------------------|
| boolean | <b>true</b> = on; <b>false</b> = off |
|---------|--------------------------------------|

### Usage

```
function preMain() {  
    setIntervalsBackfill( true );  
}
```

## **open( barIndex [, sym()] [, inv()] )**

Returns the open price at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = open(0);  
    //retrieve the value for the previous price bar  
    myVar = open(-1);  
    //retrieve the value for a specific symbol  
    myVar = open(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = open(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = open(0, inv("15") );  
  
}
```



## **high( barIndex [, sym()] [, inv()] )**

Returns the high price at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = high(0);  
    //retrieve the value for the previous price bar  
    myVar = high(-1);  
    //retrieve the value for a specific symbol  
    myVar = high(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = high(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = high(0, inv("15") );  
  
}
```



## **low( barIndex [, sym()] [, inv()] )**

Returns the low price at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = low(0);  
    //retrieve the value for the previous price bar  
    myVar = low(-1);  
    //retrieve the value for a specific symbol  
    myVar = low(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = low(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = low(0, inv("15"));  
  
}
```



## **close( barIndex [, sym()] [, inv()] )**

Returns the close price at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = close(0);  
    //retrieve the value for the previous price bar  
    myVar = close(-1);  
    //retrieve the value for a specific symbol  
    myVar = close(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = close(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = close(0, inv("15") );  
  
}
```



## **oi( barIndex [, sym()] [, inv()] )**

Returns the open interest at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = oi(0);  
    //retrieve the value for the previous price bar  
    myVar = oi(-1);  
    //retrieve the value for a specific symbol  
    myVar = oi(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = oi(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = oi(0, inv("15"));  
  
}
```



## **volume( barIndex [, sym()] [, inv()] )**

Returns the volume at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = volume(0);  
    //retrieve the value for the previous price bar  
    myVar = volume(-1);  
    //retrieve the value for a specific symbol  
    myVar = volume(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = volume(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = volume(0, inv("15"));  
  
}
```



## **year( barIndex [, sym()] [, inv()] )**

Returns the year at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = year(0);  
    //retrieve the value for the previous price bar  
    myVar = year(-1);  
    //retrieve the value for a specific symbol  
    myVar = year(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = year(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = year(0, inv("15") );  
  
}
```



## **month( barIndex [, sym()] [, inv()] )**

Returns the month at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = month(0);  
    //retrieve the value for the previous price bar  
    myVar = month(-1);  
    //retrieve the value for a specific symbol  
    myVar = month(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = month(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = month(0, inv("15"));  
  
}
```



## **day( barIndex [, sym()] [, inv()] )**

Returns the day at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = day(0);  
    //retrieve the value for the previous price bar  
    myVar = day(-1);  
    //retrieve the value for a specific symbol  
    myVar = day(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = day(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = day(0, inv("15"));  
  
}
```



## hour( barIndex [, sym()] [, inv()] )

Returns the hour at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = hour(0);  
    //retrieve the value for the previous price bar  
    myVar = hour(-1);  
    //retrieve the value for a specific symbol  
    myVar = hour(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = hour(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = hour(0, inv("15"));  
  
}
```



## **minute( barIndex [, sym()] [, inv()] )**

Returns the minute at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = minute(0);  
    //retrieve the value for the previous price bar  
    myVar = minute(-1);  
    //retrieve the value for a specific symbol  
    myVar = minute(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = minute(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = minute(0, inv("15") );  
  
}
```



## **second( barIndex [, sym()] [, inv()] )**

Returns the second at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = second(0);  
    //retrieve the value for the previous price bar  
    myVar = second(-1);  
    //retrieve the value for a specific symbol  
    myVar = second(0, sym("IBM") );  
    //retrieve the value for a specific symbol/interval  
    myVar = second(0, sym("IBM,5") );  
    //retrieve the value for a specific bar interval  
    myVar = second(0, inv("15") );  
  
}
```



## **rawtime( barIndex [, sym()] [, inv()] )**

Returns the rawtime (number of seconds elapsed since 01/01/1970) at the specified bar index.

### Parameters

|                       |   |
|-----------------------|---|
| barIndex              | bar index of series to retrieve         |
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = rawtime(0);  
    //retrieve the value for the previous price bar  
    myVar = rawtime(-1);  
    //retrieve the value for a specific symbol  
    myVar = rawtime(0, sym("IBM"));  
    //retrieve the value for a specific symbol/interval  
    myVar = rawtime(0, sym("IBM,5"));  
    //retrieve the value for a specific bar interval  
    myVar = rawtime(0, inv("15"));  
  
}
```



## hl2( [ , sym() ] [ , inv() ] )

This is a macro that can be passed as input to any function that expects an eSignal series. hl2() creates a series based on  $(\text{high}+\text{low})/2$ .

### Parameters

|                       |   |
|-----------------------|---|
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {
    var myVar;

    ...
    ...

    //return the value of an ema based on
    //(high+low)/2 from 10 bars ago.
    myVar = ema( 20, hl2(), -10 );

}
```

[hlc3\(\)](#)  
[ohlc4\(\)](#)

## hlc3( [, sym()] [, inv()] )

This is a macro that can be passed as input to any function that expects an eSignal series. hlc3() creates a series based on  $(\text{high}+\text{low}+\text{close})/3$ .

### Parameters

|                       |   |
|-----------------------|---|
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {
    var myVar;

    ...
    ...

    //return the value of an ema based on
    //(high+low+close)/3 from 10 bars ago.
    myVar = ema( 20, hlc3(), -10 );

}
```

[hl2\(\)](#)  
[ohlc4\(\)](#)

## **ohlc4( [, sym()] [, inv()] )**

This is a macro that can be passed as input to any function that expects an eSignal series. ohlc4() creates a series based on (open+high+low+close)/4.

### Parameters

|                       |   |
|-----------------------|---|
| <a href="#">sym()</a> | optional. specify a symbol to use       |
| <a href="#">inv()</a> | optional. specify a bar interval to use |

### Usage

```
function main() {
var myVar;

...
...

//return the value of an ema based on
//(open+high+low+close)/4 from 10 bars ago.
myVar = ema( 20, ohlc4(), -10 );

}
```

[hl2\(\)](#)

[hlc3\(\)](#)

## **getYear( barIndex [, numBars] [, symbol] )**

Returns the year at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getYear(0);  
    //retrieve the value for the previous price bar  
    myVar = getYear(-1);  
    //retrieve the value for a specific symbol  
    myVar = getYear(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getYear( 0, -20 );  
  
}
```

## **getMonth( barIndex [, numBars] [, symbol] )**

Returns the month at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getMonth(0);  
    //retrieve the value for the previous price bar  
    myVar = getMonth(-1);  
    //retrieve the value for a specific symbol  
    myVar = getMonth(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getMonth( 0, -20 );  
  
}
```

## **getDay( barIndex [, numBars] [, symbol] )**

Returns the day at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getDay(0);  
    //retrieve the value for the previous price bar  
    myVar = getDay(-1);  
    //retrieve the value for a specific symbol  
    myVar = getDay(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getDay( 0, -20 );  
  
}
```

## **getHour( barIndex [, numBars] [, symbol] )**

Returns the hour at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getHour(0);  
    //retrieve the value for the previous price bar  
    myVar = getHour(-1);  
    //retrieve the value for a specific symbol  
    myVar = getHour(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getHour( 0, -20 );  
  
}
```

## **getMinute( barIndex [, numBars] [, symbol] )**

Returns the minute at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getMinute(0);  
    //retrieve the value for the previous price bar  
    myVar = getMinute(-1);  
    //retrieve the value for a specific symbol  
    myVar = getMinute(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getMinute( 0, -20 );  
  
}
```

## **getSecond( barIndex [, numBars] [, symbol] )**

Returns the second at the specified bar index.

### Parameters

|          |  |
|----------|--|
| barIndex | bar index of series to retrieve            |
| numBars  | optional. number of bars of data to return |
| symbol   | optional. specify a symbol to use          |

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the value for the current price bar  
    myVar = getSecond(0);  
    //retrieve the value for the previous price bar  
    myVar = getSecond(-1);  
    //retrieve the value for a specific symbol  
    myVar = getSecond(0, "IBM" );  
    //retrieve the last 20 values into an array  
    myArray = getSecond( 0, -20 );  
  
}
```

## **getValue( barType, barIndex [, numBars] [, symbol] )**

Returns the value of the specified type at the specified bar index relative to the bar currently being processed.

### Parameters

|          |  |
|----------|--|
| barType  | type of value to retrieve: "open", "high", "low", "close", "time", "rawtime", "volume", "oi", "year", "month", "day", "hour", "minute", "second" |
| barIndex | bar index of series to retrieve  |
| numBars  | optional. number of bars of data to return   |
| symbol   | optional. specify a symbol to use  |

**Note:** using `getValue( "time" )` can be very processor-intensive and, for that reason, its use is not recommended. Instead you should use `getValue( "rawtime" )` which returns the number of seconds elapsed since 1/1/1970, or the [getYear\(\)](#), [getMonth\(\)](#), [getDay\(\)](#), [getHour\(\)](#), [getMinute\(\)](#), [getSecond\(\)](#) functions.

### Usage

```
function main() {  
  
    ...  
    ...  
    //retrieve the close from 10 bars ago  
    myVar = getValue( "close", -10 );
```

```
//retrieve the rawtime of the current bar  
myVar = getValue( "rawtime", 0 );  
//retrieve the most recent 20 high values into an array  
myArray = getValue( "high", 0, -20 );  
  
}
```

## **getValueAbsolute( barType, barIndex [, numBars] [, symbol] )**

Returns the value of the specified type at the specified bar index relative to the most recent bar in the series, regardless of the bar currently being processed.

### Parameters

|          |  |
|----------|--|
| barType  | type of value to retrieve: "open", "high", "low", "close", "time", "rawtime", "volume", "oi", "year", "month", "day", "hour", "minute", "second" |
| barIndex | bar index of series to retrieve  |
| numBars  | optional. number of bars of data to return   |
| symbol   | optional. specify a symbol to use  |

**Note:** using `getValue( "time" )` can be very processor-intensive and, for that reason, its use is not recommended. Instead you should use `getValue( "rawtime" )` which returns the number of seconds elapsed since 1/1/1970, or the [getYear\(\)](#), [getMonth\(\)](#), [getDay\(\)](#), [getHour\(\)](#), [getMinute\(\)](#), [getSecond\(\)](#) functions.

### Usage

```
function main() {
```

...

...

```
//retrieve the close from 10 bars ago  
myVar = getValueAbsolute( "close", -10 );  
//retrieve the rawtime of the current bar  
myVar = getValueAbsolute( "rawtime", 0 );  
//retrieve the 20 most recent high values into an array  
myArray = getValueAbsolute( "high", 0, -20 );
```

}

## getBarState()

Returns a status flag from the EFS engine indicating the bar processing that is currently taking place.

### Parameters

|      |                                       |
|------|---------------------------------------|
| none | this function has no input parameters |
|------|---------------------------------------|

### Return Flags

|                     |   |
|---------------------|---|
| BARSTATE_ALLBARS    | script is initializing                    |
| BARSTATE_NEWBAR     | the first tick of a new bar has arrived   |
| BARSTATE_CURRENTBAR | a new tick has arrived in the current bar |

### Usage

```
function main() {
    var nState;

    ...
    ...
    nState = getBarState();
    if (nState == BARSTATE_ALLBARS) {
        //the bars are being loaded by the script. This happens when a script
        is first loaded
        debugPrint("Script is loading\n");
    }
    else if (nState == BARSTATE_NEWBAR) {
        //this flag is set when a new bar is coming in
        debugPrint("The first tick of a new bar has arrived\n");
    }
}
```

```

}

else if (nState == BARSTATE_CURRENTBAR) {
    //this flag is set as each new tick comes in
    debugPrint("A new tick has arrived\n");
}

}

var sSymbol = null;
var nInterval = null;
var myArray = new Array(50);

function main() {

    //perform some initialization actions when script is first loading
    if (getBarState() == BARSTATE_ALLBARS) {

        sSymbol = getSymbol(); //grab the name of the symbol being charted
        for later use
        nInterval = getInterval(); //grab the bar interval we are using for
        later use

        return;
    }

    ...
    ...
    //cycle an array when a new bar comes in....
    if (getBarState() == BARSTATE_NEWBAR) {

        myArray.unshift(0);
        myArray.pop();

    }
}

```



## **getBarStateInterval( interval )**

Returns a status flag from the EFS engine indicating the bar processing that is currently taking place. This function allows you to monitor bar activity in time intervals other than the one that the script is running under.

### Parameters

|          |   |
|----------|---|
| interval | a string identifying the interval to be monitored |
|----------|---|

### Return Flags

|                     |   |
|---------------------|---|
| BARSTATE_ALLBARS    | script is initializing                    |
| BARSTATE_NEWBAR     | the first tick of a new bar has arrived   |
| BARSTATE_CURRENTBAR | a new tick has arrived in the current bar |

### Usage

```
function main() {
var nState;

...
...

//we will monitor the daily bar interval while, for example, our script may
be
    //running in a 5-min chart.
nState = getBarStateInterval("D");
if (nState == BARSTATE_ALLBARS) {
    //the bars are being loaded by the script. This happens when a script
is first loaded
    debugPrint("Script is loading\n");
```

```

}

else if (nState == BARSTATE_NEWBAR) {
    //this flag is set when a new bar is coming in, in this case on the daily
    bar interval
    debugPrint("The first tick of a new bar has arrived\n");
}

else if (nState == BARSTATE_CURRENTBAR) {
    //this flag is set as each new tick comes in, again on the daily bar
    interval
    debugPrint("A new tick has arrived\n");
}

}

var sSymbol = null;
var nInterval = null;
var myArray = new Array(50);

function main() {

    //perform some initialization actions when script is first loading
    if (getBarState() == BARSTATE_ALLBARS) {

        sSymbol = getSymbol(); //grab the name of the symbol being charted
        for later use
        nInterval = getInterval(); //grab the bar interval we are using for
        later use
        return;

    }

    ...
    ...
    //cycle an array when a new bar comes in on the 60-minute bar interval
    regardless
    //of the interval we are currently viewing in our chart....
}

```

```
if (getBarStateInterval("60") == BARSTATE_NEWBAR) {  
  
    myArray.unshift(0);  
    myArray.pop();  
  
}  
  
}
```

## getBarStateSymbol()

[Top](#) [Previous](#) [Next](#)

### getBarStateSymbol( symbol/interval )

Returns a status flag from the EFS engine indicating the bar processing that is currently taking place. This function allows you to monitor bar activity in symbols other than the one that the script is running under.

#### Parameters

|                 |  |
|-----------------|--|
| symbol/interval | a string identifying the symbol or symbol/interval combination to be monitored |
|-----------------|--|

#### Return Flags

|                     |   |
|---------------------|---|
| BARSTATE_ALLBARS    | script is initializing                    |
| BARSTATE_NEWBAR     | the first tick of a new bar has arrived   |
| BARSTATE_CURRENTBAR | a new tick has arrived in the current bar |

#### Usage

```
function main() {
var nState;

...
...

//we will monitor the Dell daily bar interval while, for example, our script
may be

    //running in a MSFT 5-min chart.
nState = getBarStateSymbol("DELL,D");
if (nState == BARSTATE_ALLBARS) {

    //the bars are being loaded by the script. This happens when a script
is first loaded
```

```

        debugPrint("Script is loading\n");
    }

else if (nState == BARSTATE_NEWBAR) {
    //this flag is set when a new bar is coming in, in this case on the daily
    bar interval
    debugPrint("The first tick of a new bar has arrived\n");
}

else if (nState == BARSTATE_CURRENTBAR) {
    //this flag is set as each new tick comes in, again on the daily bar
interval
    debugPrint("A new tick has arrived\n");
}

}

var sSymbol = null;
var nInterval = null;
var myArray = new Array(50);

function main() {

    //perform some initialization actions when script is first loading
    if (getBarState() == BARSTATE_ALLBARS) {

        sSymbol = getSymbol(); //grab the name of the symbol being charted
        for later use
        nInterval = getInterval(); //grab the bar interval we are using for
        later use
        return;
    }

    ...
    ...
}

//cycle an array when a new bar comes in on the Dell 60-minute bar
interval regardless

```

*//of the symbol and/or interval we are currently viewing in our chart...*

**if** (getBarStateSymbol("DELL,60") == BARSTATE\_NEWBAR) {

    myArray.unshift(0);

    myArray.pop();

}

}

## getCurrentBarIndex()

Returns the current offset into the price series that is loaded in the chart.

### Parameters

|      |                                       |
|------|---------------------------------------|
| none | this function has no input parameters |
|------|---------------------------------------|

### Usage

```
function main() {  
    var nBarIndex;  
  
    ...  
    ...  
  
    nBarIndex = getCurrentBarIndex();  
    if ( nBarIndex < 0 ) {  
        //we are currently on a historical bar  
    }  
    else if ( nBarIndex == 0 ) {  
        //we are on the most current bar  
    }  
}
```

## getNewestBarIndex( [symbol] )

Returns the bar index of the most recent bar in the series. This will always return zero (0) unless no data exists for the given symbol, in which case it will return null. If symbol is omitted, the symbol for the current chart is used.

### Parameters

|        |                             |
|--------|-----------------------------|
| symbol | optional. the symbol to use |
|--------|-----------------------------|

### Usage

```
function main() {
var nBarIndex;

...
...

nBarIndex = getNewestBarIndex();
if ( nBarIndex == null ) {
    //there is no data for the currently-loaded symbol
}
else if ( nBarIndex == 0 ) {
    //we have data
}
}
```

## getOldestBarIndex( [symbol] )

Returns the bar index of the oldest bar in the series. This will always return a negative integer value unless no data exists for the given symbol, in which case it will return null. If symbol is omitted, the symbol for the current chart it used.

### Parameters

|        |                             |
|--------|-----------------------------|
| symbol | optional. the symbol to use |
|--------|-----------------------------|

### Usage

```
function main() {
var nBarIndex;

...
...

nBarIndex = getOldestBarIndex();
if ( nBarIndex == null ) {
    //there is no data for the currently-loaded symbol
}
else if ( nBarIndex < 0 ) {
    //we have data
    debugPrint( "There are currently " + Math.abs( nBarIndex ) + " bars
loaded in the chart.\n");
}
}
```



## **getFirstBarIndexOfDay( date [, symbol] )**

Returns the bar index to the first bar for the date specified.

### Parameters

|        |   |
|--------|---|
| date   | a valid date object (using "time" or "rawtime") |
| symbol | optional. the symbol to use                     |

### Usage

```
function main() {
  var nBarIndex;
  var nTime;
  var nPrice;

  ...

  ...

  //get the rawtime value for the current bar
  nTime = getValue( "rawtime", 0 );

  if ( nTime != null ) {

    //get the bar index to the first bar of the date in nTime
    nBarIndex = getFirstBarIndexOfDay( nTime );
    //grab the closing price from that bar
    nPrice = close( nBarIndex );

  }
}
```



## **getPreviousTradingDay( date [, symbol] )**

Returns a date object that contains the date of the prior trading day.

### Parameters

|        |   |
|--------|---|
| date   | a valid date object (using "time" or "rawtime") |
| symbol | optional. the symbol to use                     |

### Usage

```
function main() {
    var nBarIndex;
    var nTime;
    var nPriorDay;
    var nPrice;

    ...
    ...

    //get the rawtime value for the current bar
    nTime = getValue( "rawtime", 0 );

    if ( nTime != null ) {
        //get the date value for the prior trading day, if any
        nPriorDay = getPreviousTradingDay( nTime );

        if ( nPriorDay != null ) {

            //get the bar index to the first bar of the date in nTime
        }
    }
}
```

```
nBarIndex = getFirstBarIndexOfDay( nTime );
//grab the closing price from that bar
nPrice = close( nBarIndex );

}

}

}
```

## **getNextTradingDay( date [, symbol] )**

Returns a date object that contains the date of the next trading day when the current bar IS NOT in the most current trading day.

### Parameters

|        |   |
|--------|---|
| date   | a valid date object (using "time" or "rawtime") |
| symbol | optional. the symbol to use                     |

### Usage

```
function main() {
  var nBarIndex;
  var nTime;
  var nPriorDay;
  var nPrice;

  ...

  //get the rawtime value for the current bar
  nTime = getValue( "rawtime", 0 );

  if ( nTime != null ) {

    //get the date value for the next trading day, if any
    nPriorDay = getNextTradingDay( nTime );

    if ( nPriorDay != null ) {
```

```
//get the bar index to the first bar of the date in nTime  
nBarIndex = getFirstBarIndexOfDay( nTime );  
//grab the closing price from that bar  
nPrice = close( nBarIndex );  
  
}  
}  
  
}
```

## isLastBarOnChart()

Returns **true** if the bar currently being processed is the last bar on the chart.

### Parameters

|      |                                       |
|------|---------------------------------------|
| none | this function has no input parameters |
|------|---------------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
  
        //we are currently on the last bar in the chart so  
        //we can perform some action here  
  
    }  
  
}
```

## **getNumBars()**

Returns the total number of bars loaded into the current chart.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    var nNumBars;  
  
    ...  
    ...  
    //retrieve the number of bars loaded  
    nNumBars = getNumBars();  
  
}
```

## getInterval()

Returns the bar interval of the symbol currently being charted.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    var nInterval;  
  
    ...  
    ...  
    //get the bar interval of the current symbol  
    nInterval = getInterval();  
  
}
```

## getSymbol()

Returns the symbol name of the symbol currently being charted.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    var sSymbol;  
  
    ...  
    ...  
    //get the symbol name of the symbol currently being charted  
    sSymbol = getSymbol();  
  
}
```

## highest( numBars, series )

New in EFS2. This function will return the highest value found in a series within numBars number of bars.

### Parameters

|         |   |
|---------|---|
| numBars | the number of bars back to search           |
| series  | the series in which to search for the value |

### Usage

```
function main() {  
    var myVar1;  
  
    //find the highest-high over the last 50 bars  
    myVar1 = highest( 50, high() );  
  
}
```

## [hhv\(\)](#)

## **lowest( numBars, series )**

New in EFS2. This function will return the lowest value found in a series within numBars number of bars.

### Parameters

|         |   |
|---------|---|
| numBars | the number of bars back to search           |
| series  | the series in which to search for the value |

### Usage

```
function main() {  
    var myVar1;  
  
    //find the lowest-low over the last 50 bars  
    myVar1 = lowest( 50, low() );  
  
}
```

[llv\(\)](#)

## **hhv( numBars, series )**

New in EFS2. This function will return the highest value found in a series within numBars number of bars.

### Parameters

|         |   |
|---------|---|
| numBars | the number of bars back to search           |
| series  | the series in which to search for the value |

### Usage

```
function main() {  
    var myVar1;  
  
    //find the highest-high over the last 50 bars  
    myVar1 = hhv( 50, high() );  
  
}
```

[highest\(\)](#)

## **llv( numBars, series )**

New in EFS2. This function will return the lowest value found in a series within numBars number of bars.

### Parameters

|         |   |
|---------|---|
| numBars | the number of bars back to search           |
| series  | the series in which to search for the value |

### Usage

```
function main() {  
    var myVar1;  
  
    //find the lowest-low over the last 50 bars  
    myVar1 = llv( 50, low() );  
  
}
```

[lowest\(\)](#)

## **midpoint( numBars, series )**

New in EFS2. This function will return the midpoint of a series within numBars number of bars. It is implemented as ( highest-val-found + lowest-val-found ) divided by 2.

### Parameters

|         |   |
|---------|---|
| numBars | the number of bars back to search           |
| series  | the series in which to search for the value |

### Usage

```
function main() {  
    var myVar1;  
  
    //find the midpoint of the closes over the last 50 bars  
    myVar1 = midpoint( 50, close() );  
  
}
```

This section describes the drawing functions that are available in eSignal EFS:

[drawText\(\)](#) new in EFS2

[drawShape\(\)](#) new in EFS2

[setDefaultFont\(\)](#) new in EFS2

[addBand\(\)](#)

[clearBands\(\)](#)

[removeBand\(\)](#)

[drawTextRelative\(\)](#)

[drawTextAbsolute\(\)](#)

[drawTextPixel\(\)](#)

[clearText\(\)](#)

[removeText\(\)](#)

[drawShapeRelative\(\)](#)

[drawShapeAbsolute\(\)](#)

[clearShape\(\)](#)

[removeShape\(\)](#)

[drawImageRelative\(\)](#)

[drawImageAbsolute\(\)](#)

[clearImage\(\)](#)

[removeImage\(\)](#)

[drawLineRelative\(\)](#)

[drawLineAbsolute\(\)](#)

[clearLines\(\)](#)

[removeLine\(\)](#)

[addLineTool\(\)](#)

[clearLineTool\(\)](#)

[removeLineTool\(\)](#)

## **drawText( text, location [, fgColor] [, flags])**

The drawText() function is a simplified text drawing function that was introduced in EFS2. It accepts no barIndex parameter so it can only be used to draw text at the current bar offset.

### Parameters

|          |   |
|----------|---|
| text     | the symbol or text string to draw                     |
| location | see Location Flags below                              |
| fgColor  | optional. the foreground <a href="#">color</a> to use |
| flags    | optional. see Text Flags below                        |

### Location Flags

|            |  |
|------------|--|
| TopRow1    | draws at the upper margin of the chart   |
| TopRow2    | draws one row down from TopRow1          |
| TopRow3    | draws one row down from TopRow2          |
| TopRow4    | draws one row down from TopRow3          |
| AboveBar1  | draws directly above the high of the bar |
| AboveBar2  | draws one row above AboveBar1            |
| AboveBar3  | draws one row above AboveBar2            |
| AboveBar4  | draws one row above AboveBar3            |
| BelowBar1  | draws directly below the low of the bar  |
| BelowBar2  | draws one row below BelowBar1            |
| BelowBar3  | draws one row below BelowBar2            |
| BelowBar4  | draws one row below BelowBar3            |
| BottomRow1 | draws at the lower margin of the chart   |
| BottomRow2 | draws one row above BottomRow1           |
| BottomRow3 | draws one row above BottomRow2           |

BottomRow4 | draws one row above BottomRow3

## Text Flags

|  |                             |
|--|-----------------------------|
| Text.BOLD  | display the text in bold    |
| Text.ITALIC  | display the text in italics |
| Text.UNDERLINE   | underline the text          |
| Text.PLAIN   | display plain text          |
| Note: Text flags can be ORed together to combine various attributes (e.g., Text.ITALIC   Text.BOLD   Text.UNDERLINE) |                             |

## Usage

```
//draw the word "Buy" one row down from the low of the current bar  
drawText( "Buy", BottomRow2 );
```

```
//draw the word "Sell" right above the high of the current bar  
//color the text blue and display it with underline and bold attributes  
drawText( "Sell", AboveBar1, Color.blue, Text.UNDERLINE | Text.BOLD  
);
```

[drawTextRelative\(\)](#)  
[drawTextAbsolute\(\)](#)  
[drawTextPixel\(\)](#)



## **drawShape( shape, location [, fgColor] )**

The drawShape() function is a simplified shape drawing function that was introduced in EFS2. It accepts no barIndex parameter so it can only be used to draw shapes at the current bar offset.

### Parameters

|          |   |
|----------|---|
| shape    | see Shape Objects below                               |
| location | see Location Flags below                              |
| fgColor  | optional. the foreground <a href="#">color</a> to use |

### Location Flags

|            |  |
|------------|--|
| TopRow1    | draws at the upper margin of the chart   |
| TopRow2    | draws one row down from TopRow1          |
| TopRow3    | draws one row down from TopRow2          |
| TopRow4    | draws one row down from TopRow3          |
| AboveBar1  | draws directly above the high of the bar |
| AboveBar2  | draws one row above AboveBar1            |
| AboveBar3  | draws one row above AboveBar2            |
| AboveBar4  | draws one row above AboveBar3            |
| BelowBar1  | draws directly below the low of the bar  |
| BelowBar2  | draws one row below BelowBar1            |
| BelowBar3  | draws one row below BelowBar2            |
| BelowBar4  | draws one row below BelowBar3            |
| BottomRow1 | draws at the lower margin of the chart   |
| BottomRow2 | draws one row above BottomRow1           |
| BottomRow3 | draws one row above BottomRow2           |

BottomRow4 | draws one row above BottomRow3

## Shape Objects

|                     |                                    |
|---------------------|------------------------------------|
| Shape.CIRCLE        | draws a circle                     |
| Shape.SQUARE        | draws a square                     |
| Shape.TRIANGLE      | draws a triangle                   |
| Shape.DIAMOND       | draws a diamond                    |
| Shape.LEFTARROW     | draws a left arrow                 |
| Shape.RIGHTARROW    | draws a right arrow                |
| Shape.UPARROW       | draws a up arrow                   |
| Shape.DOWNARROW     | draws a down arrow                 |
| Shape.LEFTTRIANGLE  | draws a left-pointing triangle     |
| Shape.RIGHTTRIANGLE | draws a right-pointing triangle    |
| Shape.UPTRIANGLE    | draws a upward-pointing triangle   |
| Shape.DOWNTRIANGLE  | draws a downward-pointing triangle |

## Usage

```
//draw a circle above the high of the current bar  
drawShape( Shape.CIRCLE, AboveBar1 );
```

```
//draw a square below the low of the current bar and color it red  
drawShape( Shape.SQUARE, BelowBar1, Color.red );
```

[drawShapeRelative\(\)](#)  
[drawShapeAbsolute\(\)](#)

## **setFont( fontFace, size [, fgColor] [, flags])**

The `setFont()` function was introduced in EFS2 and is designed to work in conjunction with the `drawText()` function. It allows you to set a default font to be used in all drawing operations.

### Parameters

|          |   |
|----------|---|
| fontFace | the fontFace to use as the default                    |
| size     | the desired size of the font                          |
| fgColor  | optional. the foreground <a href="#">color</a> to use |
| flags    | optional. see Text Flags below                        |

### Text Flags

|   |                             |
|---|-----------------------------|
| Text.BOLD   | display the text in bold    |
| Text.ITALIC   | display the text in italics |
| Text.UNDERLINE  | underline the text          |
| Text.PLAIN  | display plain text          |
| <b>Note:</b> Text flags can be ORed together to combine various attributes (e.g., <code>Text.ITALIC   Text.BOLD   Text.UNDERLINE</code> ) |                             |

### Usage

```
//set the default font to Arial 14-pt, use magenta as the color and apply  
//bold and underline attributes  
setFont( "Arial", 14, Color.magenta, Text.BOLD |  
Text.UNDERLINE );
```

```
//set the default font to Courier New 10-pt
setDefaultFont( "Courier New", 10 );
```

### **addBand( value, style, thickness, color, tagID )**

This function will draw a horizontal line at the value specified.

#### Parameters

|           |  |
|-----------|--|
| value     | the series value at which to draw the band                 |
| style     | see Pen Styles below                                       |
| thickness | integer value assigning line thickness                     |
| color     | <a href="#">color</a> to use when drawing the band         |
| tagID     | optional. a unique identifier for this line object or null |

#### Pen Styles

|               |                          |
|---------------|--------------------------|
| PS_SOLID      | draw a solid line        |
| PS_DOT        | draw a dotted line       |
| PS_DASH       | draw a dashed line       |
| PS_DASHDOT    | draw a dash-dot line     |
| PS_DASHDOTDOT | draw a dash-dot-dot line |

#### Usage

```
//draw a navy-colored dotted band at the 50.0 price level and  
//assign a tagID of -10 so that we can remove it later  
addBand( 50.0, PS_DOT, 3, Color.navy, -10 );
```



## clearBands()

Remove all bands currently drawn on the chart by addBand().

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() = BARSTATE_NEWBAR ) {  
        //clear all band objects that may have been drawn during the  
        //creation of the last bar  
        clearBands();  
  
    }  
  
}
```

## **removeBand( tagID )**

Remove a specific band object originally drawn by addBand(). The object is identified by its tagID.

### Parameters

|       |  |
|-------|--|
| tagID | the unique ID of the band object to remove |
|-------|--|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific band object from the chart. We we originally  
        //drew this band object, we gave it a tagID of 201 so that we  
        //could identify it later.  
        removeBand( 201 );  
  
    }  
}
```

**drawTextRelative( barIndex, yValue, text, fgColor, bgColor, Flags, fontName, fontSize [, tagID] [, cx] [, cy] )**

The drawTextRelative function allows you to draw text objects at any location on your chart. Any text object created with drawTextRelative will 'stick' to the location where it was originally anchored. So, a text object originally created at a bar index of -10 will be at a bar index of -20 after 10 new bars come in.

**Parameters**

|          |  |
|----------|--|
| barIndex | the bar index where the text object will be anchored (x-axis)    |
| yValue   | the series value where the text object will be anchored (y-axis) |
| text     | the symbol or text string to draw                                |
| fgColor  | optional. the foreground <a href="#">color</a> to use or null    |
| bgColor  | optional. the background <a href="#">color</a> to use or null    |
| Flags    | optional. see Text Flags below or use null                       |
| fontName | optional. the font to use when creating the text object or null  |
| fontSize | optional. the size of the font to use or null                    |
| tagID    | optional. a unique identifier for this text object or null       |
| cx       | optional. pixel spacing control - or omitted                     |
|          |  |

|    |  |
|----|--|
| cy | optional. pixel spacing control - or omitted |
|----|--|

## Text Flags

|   |   |
|---|---|
| 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  | draws a frame around the text using fgColor as the frame color  |
| Text.BUTTON   | draws the text as a button. fgColor and bgColor are ignored   |
| Text.RELATIVETOLEFT   | keeps the text fixed at a relative pixel distance from the left chart axis  |
| Text.RELATIVETOBOTTOM   | keeps the text fixed at a relative pixel distance from the bottom chart axis  |
| Text.RELATIVETOTOP  | keeps the text fixed at a relative pixel distance from the top chart axis   |
| Text.CENTER   | horizontally center the text over the bar   |
| Text.VCENTER  | vertically center the text at the series value  |
| Text.PRESET   | (new in EFS2) allows drawTextRelative to use the location flags described in the <a href="#">drawText</a> function. |
| <b>Note:</b> Text flags can be ORed together to combine various attributes (e.g., Text.ITALIC   Text.BOLD   Text.UNDERLINE) |   |

## Notes re the cx and cy parameters

- The cx and cy parameters control the width (cx) and height (cy) of

the text label. They are not very useful unless you are using them in conjunction with the text flags RELATIVETOTOP, RELATIVETOLEFT and RELATIVETOBOTTOM. Without these flags the parameters will be relative to the bar index for cx and the price scale for cy.

- Both cx and cy require whole numbers.
- You can pass positive or negative numbers to these parameters. If you use positive whole number then the size is based on that number of pixels:
  - cx of 15 will be the width of the text label of 15 pixels
  - cy of 15 will be the height of the text label of 15 pixels
- If you use negative whole numbers then the size is relative to the specified font size:
  - cx of -15 will be the approximate width of 15 characters of the specified font
  - cy of -2 will be 2 times the height of the specified font. -3 would be 3 times the height, etc.

## Usage

```
//draw the letter X at the 55.0 price level 10 bars back, color the text blue
//and display it in the Courier New font, center it over the bar
drawTextRelative( -10, 55.0, "X", Color.blue, null, Text.ONTOP |
Text.CENTER, "Courier New", 10 );
```

```
//use the Text.PRESET flag to draw the letter B above the high of the
current bar, color it red and
```

```
//display it with a bold attribute in a 12-pt fontsize.  
drawTextRelative( 0, AboveBar1, "B", Color.red, null, Text.PRESET |  
Text.BOLD, null, 12 );
```

## Special Characters

## **drawTextPixel( xVal, yVal, text, fgColor, bgColor, Flags, fontName, fontSize [, tagID] [, cx] [, cy] )**

The drawTextPixel function allows you to draw text objects at any location on your chart. Any text object created with drawTextAbsolute will remain at its designated relative pixel offset from the chart margins.

### Parameters

|          |   |
|----------|---|
| xVal     | pixel offset for the text object (x-axis)                       |
| yVal     | pixel offset for the text object (y-axis)                       |
| text     | the symbol or text string to draw                               |
| fgColor  | optional. the foreground <a href="#">color</a> to use or null   |
| bgColor  | optional. the background <a href="#">color</a> to use or null   |
| Flags    | optional. see Text Flags below or use null                      |
| fontName | optional. the font to use when creating the text object or null |
| fontSize | optional. the size of the font to use or null                   |
| tagID    | optional. a unique identifier for this text object or null      |
| cx       | optional. pixel spacing control - or omitted                    |
| cy       | optional. pixel spacing control - or omitted                    |

### Text Flags

---

|   |   |
|---|---|
| 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  | draws a frame around the text using fgColor as the frame color  |
| Text.BUTTON   | draws the text as a button. fgColor and bgColor are ignored   |
| Text.RELATIVETOLEFT   | keeps the text fixed at a relative pixel distance from the left chart axis  |
| Text.RELATIVETOBOTTOM   | keeps the text fixed at a relative pixel distance from the bottom chart axis  |
| Text.RELATIVETOTOP  | keeps the text fixed at a relative pixel distance from the top chart axis   |
| Text.CENTER   | horizontally center the text over the bar   |
| Text.VCENTER  | vertically center the text at the series value  |
| Text.PRESET   | (new in EFS2) allows drawTextRelative to use the location flags described in the <a href="#">drawText</a> function. |
| <b>Note:</b> Text flags can be ORed together to combine various attributes (e.g., Text.ITALIC   Text.BOLD   Text.UNDERLINE) |   |

### Notes re the cx and cy parameters

- The cx and cy parameters control the width (cx) and height (cy) of the text label. They are not very useful unless you are using them in conjunction with the text flags RELATIVETOTOP, RELATIVETOLEFT and RELATIVETOBOTTOM. Without these flags the parameters will be relative to the bar index for cx and the price scale for cy.

- Both cx and cy require whole numbers.
- You can pass positive or negative numbers to these parameters. If you use positive whole number then the size is based on that number of pixels:
  - cx of 15 will be the width of the text label of 15 pixels
  - cy of 15 will be the height of the text label of 15 pixels
- If you use negative whole numbers then the size is relative to the specified font size:
  - cx of -15 will be the approximate width of 15 characters of the specified font
  - cy of -2 will be 2 times the height of the specified font. -3 would be 3 times the height, etc.

## Usage

//draw the string "My Trading System" at the lower-left corner of the chart.  
 The text will remain fixed at a 1 pixel offset from the

//left margin and a 13-pixel offset from the bottom margin. We have a  
 assigned a unique tagID of -200 so that only one instance

//of this text object will be created (any new instances will automatically  
 delete the older instances).

```
drawTextPixel( 1, 13, "My Trading System", Color.blue, null,  

Text.RELATIVETOLEFT | Text.RELATIVETOBOTTOM, null, 12, -200 );
```

## Special Characters



**drawTextAbsolute( barIndex, yValue, text, fgColor, bgColor, Flags, fontName, fontSize [, tagID] [, cx] [, cy] )**

The drawTextAbsolute function allows you to draw text objects at any location on your chart. Any text object created with drawTextAbsolute will remain at its designated relative offset from the most current bar. So, a text object drawn with drawTextAbsolute at a bar index of -10 will always remain 10 bars back from the most current bar.

**Parameters**

|          |  |
|----------|--|
| barIndex | the bar index where the text object will be anchored (x-axis)    |
| yValue   | the series value where the text object will be anchored (y-axis) |
| text     | the symbol or text string to draw                                |
| fgColor  | optional. the foreground <a href="#">color</a> to use or null    |
| bgColor  | optional. the background <a href="#">color</a> to use or null    |
| Flags    | optional. see Text Flags below or use null                       |
| fontName | optional. the font to use when creating the text object or null  |
| fontSize | optional. the size of the font to use or null                    |
| tagID    | optional. a unique identifier for this text object or null       |
| cx       | optional. pixel spacing control - or                             |

|    |  |
|----|--|
|    | omitted                                      |
| cy | optional. pixel spacing control - or omitted |

## Text Flags

|   |   |
|---|---|
| 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  | draws a frame around the text using fgColor as the frame color  |
| Text.BUTTON   | draws the text as a button. fgColor and bgColor are ignored   |
| Text.RELATIVETOLEFT   | keeps the text fixed at a relative pixel distance from the left chart axis  |
| Text.RELATIVETOBOTTOM   | keeps the text fixed at a relative pixel distance from the bottom chart axis  |
| Text.RELATIVETOTOP  | keeps the text fixed at a relative pixel distance from the top chart axis   |
| Text.CENTER   | horizontally center the text over the bar   |
| Text.VCENTER  | vertically center the text at the series value  |
| Text.PRESET   | (new in EFS2) allows drawTextRelative to use the location flags described in the <a href="#">drawText</a> function. |
| <b>Note:</b> Text flags can be ORed together to combine various attributes (e.g., Text.ITALIC   Text.BOLD   Text.UNDERLINE) |   |

## Notes re the cx and cy parameters

- The cx and cy parameters control the width (cx) and height (cy) of the text label. They are not very useful unless you are using them in conjunction with the text flags RELATIVETOTOP, RELATIVETOLEFT and RELATIVETOBOTTOM. Without these flags the parameters will be relative to the bar index for cx and the price scale for cy.
- Both cx and cy require whole numbers.
- You can pass positive or negative numbers to these parameters. If you use positive whole number then the size is based on that number of pixels:
  - cx of 15 will be the width of the text label of 15 pixels
  - cy of 15 will be the height of the text label of 15 pixels
- If you use negative whole numbers then the size is relative to the specified font size:
  - cx of -15 will be the approximate width of 15 characters of the specified font
  - cy of -2 will be 2 times the height of the specified font. -3 would be 3 times the height, etc.

## Usage

```
//draw the letter X at the 55.0 price level 10 bars back, color the text blue
//and display it in the Courier New font, center it over the bar
drawTextAbsolute( -10, 55.0, "X", Color.blue, null, Text.ONTOP |
Text.CENTER, "Courier New", 10 );
```

```
//use the Text.PRESET flag to draw the letter B above the high of the  
current bar, color it red and  
//display it with a bold attribute in a 12-pt fontsize.  
drawTextAbsolute( 0, AboveBar1, "B", Color.red, null, Text.PRESET |  
Text.BOLD, null, 12 );
```

## Special Characters

## clearText()

Remove all text objects currently drawn on the chart by drawText(), drawTextRelative(), drawTextAbsolute() or drawTextPixel().

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() = BARSTATE_NEWBAR ) {  
        //clear all text objects that may have been drawn during the  
        //creation of the last bar  
        clearText();  
  
        //now we can start drawing new objects based on the current bar  
  
    }  
}
```

## **removeText( tagID )**

Remove a specific text object originally drawn by `drawText()`, `drawTextRelative()`, `drawTextAbsolute()` or `drawTextPixel()`. The object is identified by its tagID.

### Parameters

|       |  |
|-------|--|
| tagID | the unique ID of the text object to remove |
|-------|--|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific text object from the chart. We we originally  
        //drew this text object, we gave it a tagID of 201 so that we  
        //could identify it later.  
        removeText( 201 );  
  
    }  
  
}
```

## **drawShapeRelative( barIndex, yValue, shape, url, color, Flags [, tagID] )**

The drawShapeRelative function allows you to draw shape objects at any location on your chart. Any shape object created with drawShapeRelative will 'stick' to the location where it was originally anchored. So, a shape object originally created at a bar index of -10 will be at a bar index of -20 after 10 new bars come in.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where the shape object will be anchored (x-axis)    |
| yValue   | the series value where the shape object will be anchored (y-axis) |
| shape    | see Shape Objects below   |
| url      | a url to be launched if shape is clicked, or null if not used     |
| Color    | the <a href="#">color</a> for the shape object                    |
| Flags    | see Shape Flags below   |
| tagID    | optional. a unique identifier for this shape object or null       |

### Shape Objects

|                 |                    |
|-----------------|--------------------|
| Shape.CIRCLE    | draws a circle     |
| Shape.SQUARE    | draws a square     |
| Shape.TRIANGLE  | draws a triangle   |
| Shape.DIAMOND   | draws a diamond    |
| Shape.LEFTARROW | draws a left arrow |
|                 |                    |

|                     |                                    |
|---------------------|------------------------------------|
| Shape.RIGHTARROW    | draws a right arrow                |
| Shape.UPARROW       | draws a up arrow                   |
| Shape.DOWNARROW     | draws a down arrow                 |
| Shape.LEFTTRIANGLE  | draws a left-pointing triangle     |
| Shape.RIGHTTRIANGLE | draws a right-pointing triangle    |
| Shape.UPTRIANGLE    | draws an upward-pointing triangle  |
| Shape.DOWNTRIANGLE  | draws a downward-pointing triangle |

## Shape Flags

|  |                                    |
|--|------------------------------------|
| Shape.LEFT   | align the shape to the left        |
| Shape.RIGHT  | align the shape to the right       |
| Shape.TOP  | align the shape to the top         |
| Shape.BOTTOM   | align the shape to the bottom      |
| Shape.ONTOP  | draw the shape on top of the study |
| <b>Note:</b> Shape flags can be ORed together to combine various attributes (e.g.,<br>Shape.ONTOP   Shape.LEFT ) |                                    |

## Usage

```
//draw a circle over the high of the bar 5 bars back and color it red
drawShapeRelative( -5, high(-5), Shape.CIRCLE, null, Color.red,
Shape.TOP | Shape.ONTOP );
```

## **drawShapeAbsolute( barIndex, yValue, shape, url, color, Flags [, tagID] )**

The drawShapeAbsolute function allows you to draw shape objects at any location on your chart. Any shape object created with drawShapeAbsolute will remain at its designated relative offset from the most current bar. So, a shape object drawn with drawShapeAbsolute at a bar index of -10 will always remain 10 bars back from the most current bar.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where the shape object will be anchored (x-axis)    |
| yValue   | the series value where the shape object will be anchored (y-axis) |
| shape    | see Shape Objects below   |
| url      | a url to be launched if shape is clicked, or null if not used     |
| Color    | the <a href="#">color</a> for the shape object                    |
| Flags    | see Shape Flags below   |
| tagID    | optional. a unique identifier for this shape object or null       |

### Shape Objects

|                |                  |
|----------------|------------------|
| Shape.CIRCLE   | draws a circle   |
| Shape.SQUARE   | draws a square   |
| Shape.TRIANGLE | draws a triangle |
| Shape DIAMOND  | draws a diamond  |
|                |                  |

|                     |                                    |
|---------------------|------------------------------------|
| Shape.LEFTARROW     | draws a left arrow                 |
| Shape.RIGHTARROW    | draws a right arrow                |
| Shape.UPARROW       | draws a up arrow                   |
| Shape.DOWNARROW     | draws a down arrow                 |
| Shape.LEFTTRIANGLE  | draws a left-pointing triangle     |
| Shape.RIGHTTRIANGLE | draws a right-pointing triangle    |
| Shape.UPTRIANGLE    | draws an upward-pointing triangle  |
| Shape.DOWNTRIANGLE  | draws a downward-pointing triangle |

## Shape Flags

|  |                                    |
|--|------------------------------------|
| Shape.LEFT   | align the shape to the left        |
| Shape.RIGHT  | align the shape to the right       |
| Shape.TOP  | align the shape to the top         |
| Shape.BOTTOM   | align the shape to the bottom      |
| Shape.ONTOP  | draw the shape on top of the study |
| Note: Shape flags can be ORed together to combine various attributes (e.g., Shape.ONTOP   Shape.LEFT ) |                                    |

## Usage

//draw a circle over the high of the bar 5 bars back and color it red. Since we are using

//drawShapeAbsolute, the circle will always remain at an offset of 5 bars back

```
drawShapeAbsolute( -5, high(-5), Shape.CIRCLE, null, Color.red,
Shape.TOP | Shape.ONTOP );
```

### clearShape()

Remove all shape objects currently drawn on the chart by drawShapeRelative() or drawShapeAbsolute().

#### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

#### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
        //clear all shape objects that may have been drawn during the  
        //creation of the last bar  
        clearShape();  
  
        //now we can start drawing new objects based on the current bar  
    }  
}
```

## **removeShape( tagID )**

Remove a specific shape object originally drawn by drawShapeRelative() or drawShapeAbsolute(). The object is identified by its tagID.

### Parameters

|       |   |
|-------|---|
| tagID | the unique ID of the shape object to remove |
|-------|---|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific shape object from the chart. We we originally  
        //drew this shape object, we gave it a tagID of 201 so that we  
        //could identify it later.  
        removeShape( 201 );  
  
    }  
}
```

## **drawImageRelative( barIndex, yValue, image, url, Flags [, tagID] )**

The drawImageRelative function allows you to draw image objects at any location on your chart. Any image object created with drawImageRelative will 'stick' to the location where it was originally anchored. So, a image object originally created at a bar index of -10 will be at a bar index of -20 after 10 new bars come in.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where the image object will be anchored (x-axis)    |
| yValue   | the series value where the image object will be anchored (y-axis) |
| image    | see Image Objects below   |
| url      | a url to be launched if image is clicked, or null if not used     |
| Flags    | see Image Flags below   |
| tagID    | optional. a unique identifier for this image object or null       |

### Image Objects

|                       |                     |                      |
|-----------------------|---------------------|----------------------|
| SystemHappyFace       | SystemNotHappy      | SystemSadFace        |
| SystemReallyHappyFace | SystemCheckMarkRed  | SystemCheckMarkGreen |
| SystemCheckMarkYellow | SystemCheckMarkBlue | SystemCheckMarkBlack |
| SystemCheckMarkWhite  | SystemLightOn       | SystemLightOff       |
| SystemCircleRed       | SystemCircleGreen   | SystemCircleYellow   |
| SystemCircleBlue      | SystemCircleBlack   | SystemCircleWhite    |
|                       |                     |                      |

|                        |                         |                          |
|------------------------|-------------------------|--------------------------|
| SystemCircleButtonRed  | SystemCircleButtonGreen | SystemCircleButtonYellow |
| SystemCircleButtonBlue | SystemCircleButtonBlack | SystemCircleButtonWhite  |
| SystemQuestionRed      | SystemQuestionGreen     | SystemQuestionYellow     |
| SystemQuestionBlue     | SystemQuestionBlack     | SystemQuestionWhite      |
| SystemExclamationRed   | SystemExclamationGreen  | SystemExclamationYellow  |
| SystemExclamationBlue  | SystemExclamationBlack  | SystemExclamationWhite   |
| SystemInfoRed          | SystemInfoGreen         | SystemInfoYellow         |
| SystemInfoBlue         | SystemInfoBlack         | SystemInfoWhite          |
| SystemStopRed          | SystemStopGreen         | SystemStopYellow         |
| SystemStopBlue         | SystemStopBlack         | SystemStopWhite          |
| SystemLightning        | SystemSnowflake         | SystemSun                |

## Image Flags

|   |                                    |
|---|------------------------------------|
| Image.LEFT  | align the image to the left        |
| Image.RIGHT   | align the image to the right       |
| Image.TOP   | align the image to the top         |
| Image.BOTTOM  | align the image to the bottom      |
| Image.ONTOP   | draw the image on top of the study |
| <b>Note:</b> Image flags can be ORed together to combine various attributes (e.g., Image.ONTOP   Image.LEFT ) |                                    |

## Usage

```
//draw a happy face over the high of the bar 5 bars back and color it red
drawImageRelative( -5, high(-5), SystemHappyFace, null, Image.TOP |
Image.ONTOP );
```



## **drawImageAbsolute( barIndex, yValue, image, url, Flags [, tagID] )**

The drawImageAbsolute function allows you to draw image objects at any location on your chart. Any image object created with drawImageAbsolute will remain at its designated relative offset from the most current bar. So, a image object drawn with drawShapeAbsolute at a bar index of -10 will always remain 10 bars back from the most current bar.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where the image object will be anchored (x-axis)    |
| yValue   | the series value where the image object will be anchored (y-axis) |
| image    | see Image Objects below   |
| url      | a url to be launched if image is clicked, or null if not used     |
| Flags    | see Image Flags below   |
| tagID    | optional. a unique identifier for this image object or null       |

### Image Objects

|                       |                     |                      |
|-----------------------|---------------------|----------------------|
| SystemHappyFace       | SystemNotHappy      | SystemSadFace        |
| SystemReallyHappyFace | SystemCheckMarkRed  | SystemCheckMarkGreen |
| SystemCheckMarkYellow | SystemCheckMarkBlue | SystemCheckMarkBlack |
| SystemCheckMarkWhite  | SystemLightOn       | SystemLightOff       |
| SystemCircleRed       | SystemCircleGreen   | SystemCircleYellow   |
|                       |                     |                      |

|                        |                         |                          |
|------------------------|-------------------------|--------------------------|
| SystemCircleBlue       | SystemCircleBlack       | SystemCircleWhite        |
| SystemCircleButtonRed  | SystemCircleButtonGreen | SystemCircleButtonYellow |
| SystemCircleButtonBlue | SystemCircleButtonBlack | SystemCircleButtonWhite  |
| SystemQuestionRed      | SystemQuestionGreen     | SystemQuestionYellow     |
| SystemQuestionBlue     | SystemQuestionBlack     | SystemQuestionWhite      |
| SystemExclamationRed   | SystemExclamationGreen  | SystemExclamationYellow  |
| SystemExclamationBlue  | SystemExclamationBlack  | SystemExclamationWhite   |
| SystemInfoRed          | SystemInfoGreen         | SystemInfoYellow         |
| SystemInfoBlue         | SystemInfoBlack         | SystemInfoWhite          |
| SystemStopRed          | SystemStopGreen         | SystemStopYellow         |
| SystemStopBlue         | SystemStopBlack         | SystemStopWhite          |
| SystemLightning        | SystemSnowflake         | SystemSun                |

## Image Flags

|  |                                    |
|--|------------------------------------|
| Image.LEFT   | align the image to the left        |
| Image.RIGHT  | align the image to the right       |
| Image.TOP  | align the image to the top         |
| Image.BOTTOM   | align the image to the bottom      |
| Image.ONTOP  | draw the image on top of the study |
| Note: Image flags can be ORed together to combine various attributes (e.g., Image.ONTOP   Image.LEFT ) |                                    |

## Usage

//draw a happy face over the high of the bar 5 bars back and color it red.  
Since we are using

//drawShapeAbsolute, the happy face will always remain at an offset of 5  
bars back

drawImageAbsolute( -5, high(-5), SystemHappyFace, null, Color.red,  
Image.TOP | Image.ONTOP );



## **clearImage()**

Remove all image objects currently drawn on the chart by drawImageRelative() or drawImageAbsolute().

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
        //clear all image objects that may have been drawn during the  
        //creation of the last bar  
        clearImage();  
  
        //now we can start drawing new objects based on the current bar  
    }  
}
```

## **removeImage( tagID )**

Remove a specific image object originally drawn by drawImageRelative() or drawImageAbsolute(). The object is identified by its tagID.

### Parameters

|       |   |
|-------|---|
| tagID | the unique ID of the image object to remove |
|-------|---|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific image object from the chart. We we originally  
        //drew this image object, we gave it a tagID of 201 so that we  
        //could identify it later.  
        removeImage( 201 );  
  
    }  
}
```

## **drawLineRelative( x1, y1, x2, y2, style, thickness, color [, tagID] )**

The drawLineRelative function allows you to draw line objects at any location on your chart. Any line object created with drawlineRelative will 'stick' to the location where it was originally anchored. So, a line object originally created at a bar index of -10 will be at a bar index of -20 after 10 new bars come in.

### Parameters

|           |  |
|-----------|--|
| x1        | the starting bar index for the line                        |
| y1        | the starting series value for the line                     |
| x2        | the ending bar index for the line                          |
| y2        | the ending series value for the line                       |
| style     | see Pen Styles below                                       |
| thickness | integer value assigning line thickness                     |
| color     | <a href="#">color</a> to use when drawing the line         |
| tagID     | optional. a unique identifier for this line object or null |

### Pen Styles

|               |                          |
|---------------|--------------------------|
| PS_SOLID      | draw a solid line        |
| PS_DOT        | draw a dotted line       |
| PS_DASH       | draw a dashed line       |
| PS_DASHDOT    | draw a dash-dot line     |
| PS_DASHDOTDOT | draw a dash-dot-dot line |

## Usage

//draw a line from the high 20 bars ago to the high 5 bars ago. draw it as a solid

//line with a thickness of 2 and a color of teal. assign a tagID of 105 so we can

//identify it later if we need to remove it.

```
drawLineRelative( -20, high(-20), -5, high(-5), PS_SOLID, 2, Color.teal,  
105 );
```

## **drawLineAbsolute( x1, y1, x2, y2, style, thickness, color [, tagID] )**

The drawLineAbsolute function allows you to draw line objects at any location on your chart. Any line object created with drawLineAbsolute will remain at its designated relative offset from the most current bar. So, a line object drawn with drawLineAbsolute at a bar index of -10 will always remain 10 bars back from the most current bar.

### Parameters

|           |  |
|-----------|--|
| x1        | the starting bar index for the line                        |
| y1        | the starting series value for the line                     |
| x2        | the ending bar index for the line                          |
| y2        | the ending series value for the line                       |
| style     | see Pen Styles below                                       |
| thickness | integer value assigning line thickness                     |
| color     | <a href="#">color</a> to use when drawing the line         |
| tagID     | optional. a unique identifier for this line object or null |

### Pen Styles

|               |                          |
|---------------|--------------------------|
| PS_SOLID      | draw a solid line        |
| PS_DOT        | draw a dotted line       |
| PS_DASH       | draw a dashed line       |
| PS_DASHDOT    | draw a dash-dot line     |
| PS_DASHDOTDOT | draw a dash-dot-dot line |

## Usage

```
//draw a line from the high 20 bars ago to the high 5 bars ago. draw it as a
solid
//line with a thickness of 2 and a color of teal. assign a tagID of 105 so we
can
//identify it later if we need to remove it.
drawLineAbsolute( -20, high(-20), -5, high(-5), PS_SOLID, 2, Color.teal,
105 );
```

## clearLines()

Remove all line objects currently drawn on the chart by drawLineRelative() or drawLineAbsolute().

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
        //clear all line objects that may have been drawn during the  
        //creation of the last bar  
        clearLines();  
  
        //now we can start drawing new objects based on the current bar  
    }  
}
```

### removeLine( tagID )

Remove a specific line object originally drawn by drawLineRelative() or drawLineAbsolute(). The object is identified by its tagID.

#### Parameters

|       |  |
|-------|--|
| tagID | the unique ID of the line object to remove |
|-------|--|

#### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific line object from the chart. We we originally  
        //drew this line object, we gave it a tagID of 201 so that we  
        //could identify it later.  
        removeLine( 201 );  
  
    }  
}
```

## **addLineTool()**

The addLineTool function allows you to add various eSignal line tool objects into your chart. The supported line tool types are:

|            |                            |
|------------|----------------------------|
| MOB        | Advanced Get Make-or-Break |
| ELLIPSE    | Advanced Get Ellipse       |
| HORZ       | Horizontal line            |
| VERT       | Vertical line              |
| REGRESSION | regression channel         |
| ARROW      | line with arrow            |
| SEGMENT    | line segment               |
| RAY        | ray                        |
| EXTENDED   | extended line              |
| CIRCLE     | circle                     |

### Line Tool Variations

```
addLineTool(LineTool.MOB, x, y, name);  
addLineTool(LineTool.ELLIPSE, x1, x2, name);  
addLineTool(LineTool.HORZ, y, nThickness, Color, name);  
addLineTool(LineTool.VERT, x, nThickness, Color, name);  
addLineTool(LineTool.REGRESSION, x1, x2, nThickness, Color, name);  
addLineTool(LineTool.ARROW, x1, y1, x2, y2, nThickness, Color, name);  
addLineTool(LineTool.SEGMENT, x1, y1, x2, y2, nThickness, Color, name);  
addLineTool(LineTool.RAY, x1, y1, x2, y2, nThickness, Color,
```

```
name);
addLineTool(LineTool.EXTENDED, x1, y1, x2, y2, nThickness,
Color, name);
addLineTool(LineTool.CIRCLE, x1, y1, x2, y2, nThickness, Color,
name);
```

## Usage

```
function preMain() {

    setPriceStudy(true);
    setStudyTitle("linetool");

}

function main() {
var c;

    if( getCurrentBarIndex() == 0 ) {

        c = close(0);

        addLineTool( LineTool.HORZ, c, 2, Color.blue, "horz1" );
        addLineTool( LineTool.VERT, -34, 2, Color.blue, "vert1" );
        addLineTool( LineTool.ARROW, -104, c-5, -69, c-4, 2, Color.blue,
"arrow1" );
        addLineTool( LineTool.SEGMENT, -104, c-4, -69, c-3, 2, Color.blue,
"segment1" );
        addLineTool( LineTool.RAY, -104, c-3, -69, c-2, 2, Color.blue, "ray"
);
        addLineTool( LineTool.EXTENDED, -104, c-2, -69, c-1, 2,
Color.blue, "extended" );
        addLineTool( LineTool.REGRESSION, -104, -69, 1, Color.blue,
"reg1" );
        addLineTool( LineTool.CIRCLE, -50, c, -34, c, 1, Color.blue,
```

"circle1" );

}

}

[clearLineTool\(\)](#)  
[removeLineTool\(\)](#)

## **clearLineTool( type )**

Remove all line tool objects of the type passed.

### Parameters

|      |                                   |
|------|-----------------------------------|
| type | type of linetool object to remove |
|------|-----------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //clear all linetool MOB objects that may have been drawn during  
        the  
        //creation of the last bar  
        clearLineTool( LineTool.MOB );  
  
        //now we can start drawing new objects based on the current bar  
  
    }  
  
}
```

[addLineTool\(\)](#)  
[removeLineTool\(\)](#)

## **removeLineTool( type, tagID )**

Remove a specific line object originally drawn by drawLineRelative() or drawLineAbsolute(). The object is identified by its tagID.

### Parameters

|       |  |
|-------|--|
| type  | the type of linetool object to remove          |
| tagID | the unique ID of the linetool object to remove |

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        //remove a specific linetool MOB object that was previously created  
        //with a tagID of 204  
        removeLineTool( LineTool.MOB, 204 );  
  
        //now we can start drawing new objects based on the current bar  
  
    }  
}
```

[addLineTool\(\)](#)

[clearLineTool\(\)](#)

## Colors

### Color.RGB( rValue, gValue, bValue )

|        |                           |
|--------|---------------------------|
| rValue | a value between 0 and 255 |
| gValue | a value between 0 and 255 |
| bValue | a value between 0 and 255 |

### Available Colors

|                 |                        |
|-----------------|------------------------|
| Color.white     | <a href="#">Sample</a> |
| Color.black     | <a href="#">Sample</a> |
| Color.darkgrey  | <a href="#">Sample</a> |
| Color.grey      | <a href="#">Sample</a> |
| Color.lightgrey | <a href="#">Sample</a> |
| Color.navy      | <a href="#">Sample</a> |
| Color.blue      | <a href="#">Sample</a> |
| Color.aqua      | <a href="#">Sample</a> |
| Color.cyan      | <a href="#">Sample</a> |
| Color.teal      | <a href="#">Sample</a> |
| Color.darkgreen | <a href="#">Sample</a> |
| Color.green     | <a href="#">Sample</a> |
| Color.lime      | <a href="#">Sample</a> |
| Color.olive     | <a href="#">Sample</a> |
| Color.khaki     | <a href="#">Sample</a> |
| Color.brown     | <a href="#">Sample</a> |
| Color.purple    | <a href="#">Sample</a> |
| Color.red       | <a href="#">Sample</a> |

|                   |        |
|-------------------|--------|
| Color.magenta     | Sample |
| Color.maroon      | Sample |
| Color.yellow      | Sample |
| Color.lightyellow | Sample |
| Color.paleyellow  | Sample |

You can use the `drawText` family of functions to display various symbols on your chart (*i.e.*, arrows, circles, etc.) by using one of the symbol font sets that you probably have available on your computer. Every Windows user should have the **Symbol** font set and many users will also have the **Wingdings** font set, which is comprised of **Wingdings**, **Wingdings 2**, and **Wingdings 3**.

The best tool to use for finding and using special symbols in conjunction with the `drawText` functions is the Windows **Character Map** utility. You can load this utility as follows:

1. Click on your **Start** button in Windows;
2. Click on **All Programs**;
3. Click on **Accessories**;
4. Click on **System Tools**;
5. Click on **Character Map**.

**Note:** Probably a good idea to create a shortcut on your Windows Desktop for the Character Map utility.

Once the Character Map utility is loaded, you can use the Font selector box at the top to browse through all of the fonts installed on your computer. For special symbol purposes you will want to focus on the **Symbol** font and the **Wingdings** fonts. (**Note:** Not all users will have the Wingdings fonts installed on their computers. If that is the case you will be limited to the symbols available in the Windows Symbol font).

To print a special symbol using the `drawText` functions you will want to:

1. Load the Character Map utility;
2. Select the symbol font set that you want to use (*i.e.*, Symbol or one of the Wingdings fonts);
3. In the Character Map display, click on the specific symbol you want to use;
4. Look at the bottom of the Character Map display and write down the Character Code that you see there;
5. Convert that Character Code to a Unicode Escape Sequence by taking '\u00' (*i.e.*, a backslash, the letter u, and two zeros) and adding the last two characters of the Character Code to the end.

So, if the Character Code for your symbol is 0xE3, your Unicode Escape Sequence would be '\u00E3'. If your Character Code was 0x77, your Unicode Escape Sequence would be '\u0077'. Pretty straightforward.

Now just use this Unicode Escape Sequence as the text parameter in your [drawTextRelative](#), [drawTextAbsolute](#), or [drawTextPixel](#) function call and you are all set. Be sure to enclose your Unicode Escape Sequence with single quotes when you type it in (*i.e.*, use '\u0084' rather than \u0084).

Text objects (*e.g.*, any object created using the drawText series of functions) will always overlay lines drawn using the drawLine functions as well as lines plotted by eSignal via the return function.

### Example:

```
drawTextRelative( -20, 96.0, "TEXT", Color.purple, null, Text.BOLD |  
Text.VCENTER, null, 12, -9 );  
drawLineRelative( -50, 96.0, 0, 96.0, PS_SOLID, 2, Color.blue, -10 );  
  
return( 96.0 );
```

In the example above, we are printing the word "TEXT" at the 96.0 price level, drawing a line at the 96.0 price level and returning 96.0 as a value to be plotted. In this case the word "TEXT" will overlay both the drawn line and the plotted line and the drawn line will overlay the plotted line.

Line objects (*e.g.*, objects created using the drawLine series of functions) will always overlay lines plotted by eSignal via the return function. Within Line objects, the most recently drawn Line object will overlay previously drawn Line objects.

### Example:

```
drawLineRelative( -50, 96.0, 0, 96.0, PS_SOLID, 2, Color.blue, -10 );  
drawLineRelative( -50, 96.0, 0, 96.0, PS_SOLID, 2, Color.red, -11 );
```

```
return( 96.0 );
```

In the example above, two lines are drawn at the 96.0 price level and we are returning 96.0 as a value to be plotted. The second line drawn (*e.g.*, the red line) will always overlay the first line drawn (*e.g.*, the blue line) and both drawn lines will overlay the plotted line.

Within plotted lines, the z-order will be determined by the order the values are returned, with the later values having precedence over the earlier values.

### **Example:**

```
return new Array( Plot1, Plot2, Plot3 );
```

Assuming that Plot1, Plot2 and Plot3 all meet at the same price level, Plot3 will be drawn over Plot2 and Plot2 will be drawn over Plot1.

This section describes the utility functions that are available in eSignal EFS:

[addEntitlement\(\)](#)

[askForInput\(\)](#)

[call\(\)](#)

[callFunction\(\)](#)

[checkVersion\(\)](#)

[debugClear\(\)](#)

[debugPrint\(\)](#)

[debugPrintln\(\)](#)

[formatPriceNumber\(\)](#)

[getCurrentBarCount\(\)](#)

[getBuildNumber\(\)](#)

[verifyBuild\(\)](#)

[getFormulaRoot\(\)](#)

[getSoundFileRoot\(\)](#)

[getFormulaOutputRoot\(\)](#)

[getInvokerID\(\)](#)

[getRuntimeRoot\(\)](#)

[getUserName\(\)](#)

[getTextHeight\(\)](#)

[getTextWidth\(\)](#)

[isComputeOnClose\(\)](#)

[isDaily\(\)](#)

[isDWM\(\)](#)

[isFinite\(\)](#)

[isInSession\(\)](#)

[isIntraday\(\)](#)

[isMonthly\(\)](#)

[isNaN\(\)](#)

[isPlayBackMode\(\)](#)

[isRawTick\(\)](#)

[isReplayMode\(\)](#)

[isTick\(\)](#)

[isWeekly\(\)](#)

[parseFloat\(\)](#)

[parseInt\(\)](#)

[ref\(\)](#)

[setBar\(\)](#)

[reloadEFS\(\)](#)

[setBarBgColor\(\)](#)

[setDefaultBarBgColor\(\)](#)

[setBarFgColor\(\)](#)

[setDefaultBarFgColor\(\)](#)

[setChartBG\(\)](#)

[setDefaultChartBG\(\)](#)

[setColorPriceBars\(\)](#)  
[setPriceBarColor\(\)](#)  
[setDefaultPriceBarColor\(\)](#)

[setBarStyle\(\)](#)  
[setDefaultBarStyle\(\)](#)

[setBarThickness\(\)](#)  
[setDefaultBarThickness\(\)](#)

[setPlotType\(\)](#)  
[setHistogramBase\(\)](#)

[setComputeOnClose\(\)](#)  
[setCursorLabelName\(\)](#)  
[setPriceStudy\(\)](#)  
[setShowCursorLabel\(\)](#)  
[setShowTitleParameters\(\)](#)  
[setStudyTitle\(\)](#)  
[setStudyMax\(\)](#)  
[setStudyMin\(\)](#)

[setGlobalValue\(\)](#)  
[getGlobalValue\(\)](#)  
[removeGlobalValue\(\)](#)

## **addEntitlement( code/url, description [, URL] )**

Provides script developers with a mechanism to quickly activate or deactivate scripts for paying customers. Using this feature, you can encrypt and distribute a single copy of a particular script and only authorized users will be able to run the script.

### Parameters

|             |  |
|-------------|--|
| code/url    | code (provided by eSignal) or fully-qualified URL to text file containing valid user names |
| description | text to display to unentitled user   |
| URL         | optional. URL to launch if user clicks on description                                      |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );  
  
    //when this script is loaded, addEntitlement() will be called and the current  
    //user's eSignal  
    //program username will be checked against a list of authorized user  
    //names. If this user is  
    //not in that list, script operation will abort and an error message will be
```

*displayed*

```
    addEntitlement( "www.myWebSite.com/AuthorizedUsers.txt", "Sorry. Not  
authorized.", "www.MyWebSite.com" );
```

```
}
```

## **askForInput( [title] )**

Calling this function will cause an abbreviated version of the Edit Studies dialog to be display so that users can edit parameters for the script. It can be called from preMain() or from within the main() function. Typically used in conjunction with Function Parameter routines.

### Parameters

|       |                                      |
|-------|--------------------------------------|
| title | optional. title to display in dialog |
|-------|--------------------------------------|

### Usage

```
function preMain() {
```

```
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );
```

```
    ...
```

```
    ...
```

```
    //force the script parameter menu to be displayed when the script  
    //is first loaded  
    askForInput( "My EMA Script" );
```

```
}
```

## FunctionParameter Object

## **call( efsname [, arguments] )**

This function is provided for backwards compatibility with EFS1. It allows you to call another script and retrieve the return values from that script.

**Note:** call() has been replaced in EFS2 by the new [efs\(\)](#)function.

### Parameters

|           |   |
|-----------|---|
| efsname   | the path and filename of the EFS script to be called                  |
| arguments | list of input parameters, if any, required by the script being called |

### Usage

```
function main() {
var myVar;
...
...
//call a second script that calculates a custom moving average
//and retrieve the current value. In this example, the script
//we are calling has one input parameter.
myVar = call( "myCustomMA.efs", 10 );
}
```

[callFunction\(\)](#)

### callFunction( efsname, funcname [, arguments] )

This function is provided for backwards compatibility with EFS1. It allows you to call a specific function within another script and retrieve the return values from that function.

#### Parameters

|           |   |
|-----------|---|
| efsname   | the path and filename of the EFS script to be called                  |
| funcname  | the name of the function within the EFS script to be called           |
| arguments | list of input parameters, if any, required by the script being called |

#### Usage

```
function main() {  
    var myVar;  
  
    ...  
  
    ...  
  
    //call a function within another script and retrieve the  
    //value(s) returned by the function.  
    myVar = call( "myCustomMA.efs", "calcStdDev", 150 );  
  
}  
  
call\(\)
```

## **checkVersion( versionNo, URL, fName )**

Provides script developers with a mechanism to get updated versions of scripts and/or libraries into the user's hands. The EFS update settings govern how often a user's system will check for updated scripts/libraries. To access the *EFS Update* dialog, select **Tools-->EFS-->AutoUpdates** from the eSignal menu.

### Parameters

|           |  |
|-----------|--|
| versionNo | the current version number in this script        |
| URL       | the url to check for a more recent version       |
| fName     | file name (required when used with efsLib files) |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );  
  
    //Assume that the name of this script is ExampleScript.efs  
    //Each time this script is loaded, it will check the URL specified  
    //in the checkVersion function. If the versionNo in the file found  
    //at the URL is greater than the value of versionNo in the running
```

*//EFS, the user will be prompted to allow eSignal to download the newer script. The download as well as the replacement of the old EFS with the new is automatic.*

**checkVersion( 1.0, "http://myWebSite.com/ExampleScript.efs" );**

*//So, if I was the author of ExampleScript.efs and if I had created a new version, I would simply change the versionNo in my new version from 1.0 to 1.1 and then I would upload the EFS file to the location specified on my web site. Any users of my script would then be prompted to update the very next time they run the script.*

*//Here is the correct syntax to use when implementing checkVersion with a library file*

**checkVersion( 1.0, "http://myWebSite.com/ExampleLibrary.efsLib",  
"ExampleLibrary.efsLib" );**

**}**

## debugClear()

Clear the contents of the eSignal Formula Output Window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "The current close value is: " + close(0) + "\n" );  
    }  
  
}
```

## debugPrint( string )

Print a string to the eSignal Formula Output Window.

### Parameters

|        |                               |
|--------|-------------------------------|
| string | the text string to be printed |
|--------|-------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "The current close value is: " + close(0) + "\n" );  
    }  
  
}
```

## **debugPrintln( string )**

Print a string to the eSignal Formula Output Window and include a carriage return.

### Parameters

|        |                               |
|--------|-------------------------------|
| string | the text string to be printed |
|--------|-------------------------------|

### Usage

```
function main() {  
  
    ...  
  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrintln( "The current close value is: " + close(0) );  
    }  
  
}
```

## **formatPriceNumber( value )**

Returns a string with the passed value correctly formatted for the symbol in the chart that is currently displayed. Useful for converting a raw price value to a suitable display value when working with bonds, notes, currencies or any other instrument that has special formatting requirements.

### Parameters

|       |                                 |
|-------|---------------------------------|
| value | the price value to be formatted |
|-------|---------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "The current close value is: " + formatPriceNumber(  
close(0) ) + "\n" );  
    }  
}
```

## **getCurrentBarCount()**

New in EFS2. Returns the current bar count calculated as ( 1+ [getCurrentBarIndex\(\)](#) - [getOldestBarIndex\(\)](#) ).

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    myVar = getCurrentBarCount();  
  
}
```

## getBuildNumber()

Returns the eSignal Build Number of the system in which the script is being run. It can be used to alter script behavior depending upon the version of eSignal that the user is running.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "Your eSignal Build Number is: " + getBuildNumber() +  
        "\n" );  
    }  
}
```

## getFormulaRoot()

Returns the full path to the eSignal Formula folder.

**Note:** you can view/change the location of the Formula folder from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "Your Formula folder is located here: " +  
getFormulaRoot() + "\n" );  
    }  
}
```

## getSoundFileRoot()

Returns the full path to the eSignal Sound Files folder.

**Note:** you can view/change the location of the Sound Files folder from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "Your Sound Files folder is located here: " +  
getSoundFileRoot() + "\n" );  
    }  
}
```

## getFormulaOutputRoot()

Returns the full path to the eSignal Formula Output folder. The Formula Output folder is the only folder into which an eSignal script can create files.

**Note:** you can view/change the location of the Formula Output folder from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {
```

```
    ...
```

```
    ...
```

```
        if ( isLastBarOnChart() == true ) {
            //clear the Formula Output Window
            debugClear();
            //print a line of text to that window
            debugPrint( "Your Formula Output folder is located here: " +
getFormulaOutputRoot() + "\n" );
        }
```

}

## File Object

## getInvokerID()

[Top](#) [Previous](#) [Next](#)

## getInvokerID()

Returns a unique identifier for the chart window that the current script is running under.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
  
    ...  
  
    myVar = getInvokerID();  
  
}
```

## getRuntimeRoot()

Returns the full path to the eSignal application folder.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "Your copy of eSignal is located here: " +  
            getRuntimeRoot() + "\n" );  
    }  
  
}
```

## **getTextHeight( text [, fontName] [, fontSize] )**

Returns the height of *text* in pixels.

### Parameters

|          |                                       |
|----------|---------------------------------------|
| text     | the string to evaluate                |
| fontName | optional. the name of the font to use |
| fontSize | optional. the font size to use        |

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    //return the height, in pixels of the letter Y in  
    //the Arial font with a size of 24  
    myVar = getTextHeight( "Y", "Arial", 24 );  
}
```

[getTextWidth\(\)](#)

## getTextWidth( *text* [, *fontName*] [, *fontSize*] )

Returns the width of *text* in pixels.

### Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>text</i>     | the string to evaluate                |
| <i>fontName</i> | optional. the name of the font to use |
| <i>fontSize</i> | optional. the font size to use        |

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    //return the width, in pixels, of the string  
    //"Test Results" in a 24-point Arial font.  
    myVar = getTextWidth( "Test Results", "Arial", 24 );  
  
}
```

[getTextHeight\(\)](#)

## **getUserName()**

Returns the eSignal program username of the user running the script.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "Your eSignal Program Username is: " getUserName() +  
        "\n" );  
    }  
  
}
```

## isComputeOnClose()

New in EFS2. This function returns **true** if *setComputeOnClose()* is being used in the *preMain()* function or if the "Make all formulas compute on close" option is checked in the *Formula Engine Settings* dialog.

**Note:** you can view/change the global compute-on-close setting from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    //there may be times where you want to handle certain logic  
    //differently if setComputeOnClose() is being used  
    if ( isComputeOnClose()==true ) {  
        //do this  
    }  
    else {  
        //do that
```

}

}

[setComputeOnClose\(\)](#)

## **isDaily( [interval] )**

Returns **true** if the interval passed is a daily interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isDaily()==true ) {  
            debugPrint( "Sorry. This script does not run in daily charts.\n"  
);  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

## **isDWM( [interval] )**

Returns **true** if the interval passed is a daily, weekly or monthly interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isDWM()==true ) {  
            debugPrint( "Sorry. This script does not run in daily, weekly or  
monthly charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

## **isFinite( value )**

Returns **true** if the value passed is a finite number.

### Parameters

|       |                           |
|-------|---------------------------|
| value | the value to be evaluated |
|-------|---------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    if ( ! isFinite( myVar ) ) {  
        //the variable myVar does not contain a finite number  
        //so take some action  
    }  
  
}
```

## isInSession()

This function compares the current clock time to the start and end times of the eSignal Time Template that you have applied to your chart. It returns **true** if the current clock time is within the start and end times of your Time Template, otherwise it returns **false**. If you use a 24-hour Time Template, this function will always return **true**.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isInSession() == false ) {  
        //clear the Formula Output Window  
        debugClear();  
        //print a line of text to that window  
        debugPrint( "We are currently outside of the session as specified by  
your current Time Template.\n" );  
    }  
}
```

## **isIntraday( [interval] )**

Returns **true** if the interval passed is an intraday bar interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isIntraday()==true ) {  
            debugPrint( "Sorry. This script does not run in intraday  
charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

### isMonthly( [interval] )

Returns **true** if the interval passed is a monthly interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

#### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

#### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isMonthly()==true ) {  
            debugPrint( "Sorry. This script does not run in monthly  
charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
  
}
```

}

...

...

}

## **isNaN( value )**

Returns **true** if the value passed is NaN (Not a Number).

### Parameters

|       |                           |
|-------|---------------------------|
| value | the value to be evaluated |
|-------|---------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    if ( isNaN( myVar ) == true ) {  
        //the variable myVar does not contain a valid number  
        //so take some action  
    }  
  
}
```

## **isPlayBackMode( \$PLAYBACK symbol )**

Returns **true** if the script is currently running in Playback mode.

### Parameters

|        |                     |
|--------|---------------------|
| symbol | the symbol to check |
|--------|---------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isPlayBackMode( $PLAYBACK getSymbol() ) == true ) {  
            debugPrint( "Sorry. This script does not run in playback  
mode.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

...

...

}

## **isRawTick( [interval] )**

Returns **true** if the interval passed is a tick interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isRawTick()==true ) {  
            debugPrint( "Sorry. This script does not run in tick charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

## isReplayMode()

Returns **true** if the script is currently running in eSignal Replay mode.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isReplayMode() == true ) {  
            debugPrint( "Sorry. This script does not run in replay mode.\n"  
        );  
        bBail=true;  
        return;  
    }  
  
    bInit = true;
```

}

...

...

}

## **isTick( [interval] )**

Returns **true** if the interval passed is a tick interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isTick()==true ) {  
            debugPrint( "Sorry. This script does not run in tick charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

## **isWeekly( [interval] )**

Returns **true** if the interval passed is a weekly interval. If no interval is passed, the function uses the interval of the chart in which the script is running.

### Parameters

|          |                       |
|----------|-----------------------|
| interval | the interval to check |
|----------|-----------------------|

### Usage

```
var bBail = false;  
var bInit = false;  
  
function main() {  
  
    if ( bBail==true ) {  
        return;  
    }  
  
    if ( bInit==false ) {  
  
        if ( isWeekly()==true ) {  
            debugPrint( "Sorry. This script does not run in weekly  
charts.\n" );  
            bBail=true;  
            return;  
        }  
  
        bInit = true;  
    }  
}
```

}

...

...

}

## parseFloat( value )

Converts the value passed to a floating point number.

### Parameters

|       |                                  |
|-------|----------------------------------|
| value | the string value to be converted |
|-------|----------------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    //convert the string representation into a floating point number  
    myVar = parseFloat( "1.45602" );  
  
}
```

## parseInt( value )

Converts the value passed to an integer value.

### Parameters

|       |                                  |
|-------|----------------------------------|
| value | the string value to be converted |
|-------|----------------------------------|

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    //convert the string representation into an integer  
    myVar = parseInt( "5" );  
  
}
```

## **ref( barIndex )**

The ref() function allows you to retrieve prior values for a built-in study (or studies). If multiple return values are being used then the ref() function will return its results in an array.

### Parameters

|          |  |
|----------|--|
| barIndex | offset indicating which historical indicator data points to retrieve |
|----------|--|

### Usage

```
var study1      = null;
var study2      = null;
var bInit        = false;
var BarCntr     = 0;
```

```
function preMain() {
    setPriceStudy(true);
}
```

```
function main() {
var nMA1;
var nMA2;
var myRef;
var myValue1, myValue2;

if ( bInit==false ) {
```

```

study1 = sma( 10 );
study2 = sma( 20 );

bInit=true;
}

nMA1 = study1.getValue(0);
nMA2 = study2.getValue(0);

if (getBarState() == BARSTATE_NEWBAR) {
    BarCntr += 1;
}

/** BarCntr logic **/

// We're using 20 because our longest sma() requires a minimum of 20 bars.
if (BarCntr > 20) {

    //myRef will now contain the two MA values from the prior bar.
    //they will be in an array so we need to treat them as such
    myRef = ref(-1);
    if ( myRef != null ) {
        myValue1 = myRef[0];
        myValue2 = myRef[1];
    }

}

// Open the Formula Output Window from the tools menu to view the values of myRef.
if (BarCntr > 20) {
    debugPrintln("Bar Index: " + getCurrentBarIndex() + " myValue1= "
+ myValue1 + " myValue2= " + myValue2);
}

```

}

**return new** Array( nMA1, nMA2 );

}

## reloadEFS()

This function forces the current EFS script to be reloaded by the chart. Note that the values set in any variables external to the main() function will be preserved during the reload process. Use this function with caution as it can cause undesirable behavior.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
var bCompleteRefreshNeeded = false;  
  
function main() {  
  
    ...  
    ...  
  
    if ( getBarState() == BARSTATE_NEWBAR ) {  
  
        if ( bCompleteRefreshNeeded == true ) {  
            bCompleteRefreshNeeded = false;  
            reloadEFS();  
        }  
  
    }  
  
}
```



**setBar( type, barIndex, seriesIndex, value )**  
**setBar( type, barIndex, value | arrayOfValues )**

This function allows you to adjust the value and/or the properties of a prior bar of a study being displayed. For example, if you are calculating and displaying 2 moving average studies in your script, you could use the setBar() function to change the actual value, color, line style or thickness of a previously-displayed value from either one of the studies or even both of the studies.

### Parameters

|             |   |
|-------------|---|
| type        | see Valid Types below                     |
| barIndex    | the bar index of the bar to modify        |
| seriesIndex | the index into the array of return values |
| value       | the new value (or array of values)        |

### Valid Types

|               |  |
|---------------|--|
| Bar.Value     | change the value of a prior bar                            |
| Bar.Style     | change the line style of a prior bar                       |
| Bar.FgColor   | change the foreground <a href="#">color</a> of a prior bar |
| Bar.BgColor   | change the background <a href="#">color</a> of a prior bar |
| Bar.Thickness | change the line thickness of a prior bar                   |

### Usage

//set the value of the prior bar of the first study you are returning

```
//from your script to 22.  
setBar( Bar.Value, -1, 22 );  
  
//set the value of the prior bar of the 3rd study you are returning  
//from your script to 25.  
setBar( Bar.Value, -1, 3, 25 );  
  
//set the line style of the first study returned from your script to PS_SOLID (3  
bars ago)  
setBar( Bar.Style, -3, 2, PS_SOLID );  
  
//Assume your script is returning 3 values to be plotted. The following code will  
change  
//the background color of each study (at a point 10 bars ago) to blue, green and  
yellow respectively.  
setBar( Bar.BgColor, -10, new Array( Color.blue, Color.green, Color.yellow ) );
```

## **setBarBgColor( color [, series] [, yMin] [, yMax] )**

This function sets the background color for a series value that is being returned to the chart for plotting.

### Parameters

|        |  |
|--------|--|
| color  | the <a href="#">color</a> to use                   |
| series | optional. an integer identifying the series offset |
| yMin   | optional. paint background down to this yvalue     |
| yMax   | optional. paint background up to this yvalue       |

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    if ( myVar > 0 ) {  
        setBarBgColor( Color.green, 0 );  
    }  
    else {  
        setBarBgColor( Color.red, 0 );  
    }  
  
    return( myVar );
```

}

[setDefaultBarBgColor\(\)](#)

## **setBarFgColor( color [, series] )**

This function sets the foreground color for a series value that is being returned to the chart for plotting.

### Parameters

|        |  |
|--------|--|
| color  | the <a href="#">color</a> to use                   |
| series | optional. an integer identifying the series offset |

### Usage

```
function main() {
    var myVar;

    ...
    ...

    if ( myVar > 0 ) {
        setBarFgColor( Color.green, 0 );
    }
    else {
        setBarFgColor( Color.red, 0 );
    }

    return( myVar );
}
```

[setDefaultBarFgColor\(\)](#)

## **setBarStyle( style [, series] )**

This function sets the line style for a series value that is being returned to the chart for plotting.

### Parameters

|        |  |
|--------|--|
| style  | see Pen Styles below                               |
| series | optional. an integer identifying the series offset |

### Pen Styles

|               |                          |
|---------------|--------------------------|
| PS_SOLID      | draw a solid line        |
| PS_DOT        | draw a dotted line       |
| PS_DASH       | draw a dashed line       |
| PS_DASHDOT    | draw a dash-dot line     |
| PS_DASHDOTDOT | draw a dash-dot-dot line |

### Usage

```
function main() {
    var myVar;

    ...
    ...

    if ( myVar > 0 ) {
        setBarStyle( PS_SOLID, 0 );
    }
    else {
```

```
    setBarStyle( PS_DOT, 0 );  
}  
  
return( myVar );  
}
```

## [setDefaultBarStyle\(\)](#)

## **setBarThickness( thickness [, series] )**

This function sets the line thickness for a series value that is being returned to the chart for plotting.

### Parameters

|           |  |
|-----------|--|
| thickness | integer value for line thickness                   |
| series    | optional. an integer identifying the series offset |

### Usage

```
function main() {
var myVar;

...
...

if ( myVar > 0 ) {
    setBarStyle( PS_SOLID, 0 );
    setBarThickness( 4, 0 );
}
else {
    setBarStyle( PS_DOT, 0 );
    setBarThickness( 2, 0 );
}

return( myVar );
}
```

[setDefaultBarThickness\(\)](#)

## **setChartBG( color )**

This function sets the background color for the entire chart.

### Parameters

|       |                  |
|-------|------------------|
| color | the color to use |
|-------|------------------|

### Usage

```
function main() {
    var myVar;

    ...
    ...

    if ( myVar > 0 ) {
        setChartBG( Color.green );
    }
    else {
        setChartBG( Color.red );
    }

    return( myVar );
}
```

[setDefaultChartBG\(\)](#)

### **setColorPriceBars( boolean )**

This function can only be called from within the preMain() function.  
Used in conjunction with setPriceBarColor() and  
setDefaultPriceBarColor() to activate the price bar coloring feature.

#### Parameters

|         |               |
|---------|---------------|
| boolean | true or false |
|---------|---------------|

#### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );  
  
    //activate the price bar coloring feature  
    setColorPriceBars(true);  
    setDefaultPriceBarColor( Color.lime );  
  
    //force this script to only update on each new bar  
    setComputeOnClose();  
  
}
```

[setPriceBarColor\(\)](#)

[setDefaultPriceBarColor\(\)](#)

## **setDefaultBarBgColor( color [, series] [, yMin] [, yMax] )**

This function sets the default background color for a series value that is being returned to the chart for plotting. Generally called from within the preMain() function.

### Parameters

|        |  |
|--------|--|
| color  | the <a href="#">color</a> to use                   |
| series | optional. an integer identifying the series offset |
| yMin   | optional. paint background down to this yvalue     |
| yMax   | optional. paint background up to this yvalue       |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarBgColor( Color.yellow, 0 );  
    setShowTitleParameters( false );  
  
}
```

[setBarBgColor\(\)](#)

## setDefaultBarFgColor( color [, series] )

This function sets the default foreground color for a series value that is being returned to the chart for plotting. Generally called from within the preMain() function.

### Parameters

|        |  |
|--------|--|
| color  | the <a href="#">color</a> to use                   |
| series | optional. an integer identifying the series offset |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarBgColor( Color.yellow, 0 );  
    setShowTitleParameters( false );  
  
}
```

[setBarFgColor\(\)](#)

## **setDefaultBarStyle( style [, series] )**

This function sets the default line style for a series value that is being returned to the chart for plotting. Generally called from within preMain()

### Parameters

|        |  |
|--------|--|
| style  | see Pen Styles below                               |
| series | optional. an integer identifying the series offset |

### Pen Styles

|               |                          |
|---------------|--------------------------|
| PS_SOLID      | draw a solid line        |
| PS_DOT        | draw a dotted line       |
| PS_DASH       | draw a dashed line       |
| PS_DASHDOT    | draw a dash-dot line     |
| PS_DASHDOTDOT | draw a dash-dot-dot line |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarBgColor( Color.yellow, 0 );  
    setDefaultBarStyle( PS_SOLID, 0 );  
    setShowTitleParameters( false );  
}
```

}

[setBarStyle\(\)](#)

## **setDefaultBarThickness( thickness [, series] )**

This function sets the default line thickness for a series value that is being returned to the chart for plotting. Generally called from within the preMain() function.

### Parameters

|           |  |
|-----------|--|
| thickness | integer value for line thickness                   |
| series    | optional. an integer identifying the series offset |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarBgColor( Color.yellow, 0 );  
    setDefaultBarStyle( PS_SOLID, 0 );  
    setDefaultBarThickness( 2, 0 );  
    setShowTitleParameters( false );  
  
}
```

[setBarThickness\(\)](#)

## **setDefaultChartBG( color )**

This function sets the default background color for the entire chart.  
Generally called from within the preMain() function.

### Parameters

|       |                                  |
|-------|----------------------------------|
| color | the <a href="#">color</a> to use |
|-------|----------------------------------|

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarThickness( 2, 0 );  
  
    setDefaultChartBG( Color.white );  
  
    setShowTitleParameters( false );  
  
}
```

[setChartBG\(\)](#)

## **setDefaultPriceBarColor( color )**

This function sets the default price bar color. Used in conjunction with `setColorPriceBars()` and `setPriceBarColor()`. Can only be called from within the `preMain()` function.

### Parameters

|       |                                  |
|-------|----------------------------------|
| color | the <a href="#">color</a> to use |
|-------|----------------------------------|

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarBgColor( Color.yellow, 0 );  
  
    setColorPriceBars(true);  
    setDefaultPriceBarColor( Color.black );  
  
    setShowTitleParameters( false );  
  
}
```

[setColorPriceBars\(\)](#)

[setDefaultPriceBarColor\(\)](#)

## **setHistogramBase( value )**

Sets the base of a histogram plot to this value. Used in conjunction with setPlotType( PLOTTYPE\_HISTOGRAM ).

### Parameters

|       |  |
|-------|--|
| value | the value to use as the histogram base |
|-------|--|

### Usage

```
function preMain() {

    setPriceStudy(false);
    setStudyTitle("Example Script");
    setCursorLabelName("RSI", 0);
    setDefaultBarFgColor( Color.blue, 0 );
    setDefaultBarBgColor( Color.yellow, 0 );
    setDefaultBarStyle( PS_SOLID, 0 );
    setDefaultBarThickness( 2, 0 );

    setPlotType( PLOTTYPE_HISTOGRAM, 0 );
    setHistogramBase( 50 );

    setShowTitleParameters( false );
}

setPlotType\(\)
```

## **setPlotType( plotType [, series] )**

This function sets the plot type for a series value that is being returned to the chart for plotting.

### Parameters

|          |  |
|----------|--|
| plotType | see Plot Types below                               |
| series   | optional. an integer identifying the series offset |

### Plot Types

|                           |  |
|---------------------------|--|
| PLOTTYPE_LINE             | plot a line  |
| PLOTTYPE_DOT              | plot a dotted line   |
| PLOTTYPE_SQUAREWAVE       | plot a stairstep line  |
| PLOTTYPE_HISTOGRAM        | plot a histogram   |
| PLOTTYPE_FLATLINES        | plot a line that does not connect values                     |
| PLOTTYPE_INSTANTCOLORLINE | plot a line that responds to color changes on the signal bar |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );  
}
```

```
//set the plot type to a stair step
setPlotType( PLOTTYPE_SQUAREWAVE, 0 );

//force this script to only update on each new bar
setComputeOnClose();

}
```

### setHistogramBase()

## **setPriceBarColor( color [, wickColor] [, bThin] [, bHollow] )**

This function sets price bar color for the current bar. Used in conjunction with setColorPriceBars() and setDefaultPriceBarColor()

### Parameters

|           |   |
|-----------|---|
| color     | the <a href="#">color</a> to use                        |
| wickColor | optional. the <a href="#">color</a> for the candle wick |
| bThin     | optional. set thin wick (true or false)                 |
| bHollow   | optional. set hollow candle (true or false)             |

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    if ( myVar > 0 ) {  
        setPriceBarColor( Color.green );  
    }  
    else {  
        setPriceBarColor( Color.red );  
    }  
  
    return( myVar );
```

}

[setColorPriceBars\(\)](#)

[setDefaultPriceBarColor\(\)](#)

## **setComputeOnClose()**

This function can only be called from within the preMain() function. When called, it will force the current EFS script to update only when each new bar arrives (as opposed to updating on each new tick).

**Note:** you can view/change the global compute-on-close setting from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
    setCursorLabelName("EMA", 0);  
    setDefaultBarFgColor( Color.blue, 0 );  
    setShowTitleParameters( false );  
  
    //force this script to only update on each new bar  
    setComputeOnClose();  
  
}
```



## **setCursorLabelName( label [, offset] )**

This function sets the name of a value (to be displayed in the eSignal cursor window) that is returned by an eSignal script. A script can return 1 value or return multiple values. You would use setCursorLabelName() to set a name for each value that your script returns.

### Parameters

|        |  |
|--------|--|
| label  | text string to be displayed in cursor window           |
| offset | optional. the offset into the array of returned values |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Example Script");  
  
    //this script returns 3 values to be plotted on the chart.  
    //use setCursorLabelName() to set a name for each return  
    //value  
  
    setCursorLabelName("EMA", 0);  
    setCursorLabelName("SMA", 1 );  
    setCursorLabelName("WMA", 2 );  
  
    setDefaultBarFgColor( Color.blue, 0 );
```

```
    setDefaultBarFgColor( Color.red, 1 );
    setDefaultBarFgColor( Color.green, 2 );

    setShowTitleParameters( false );

}
```

## **setPriceStudy( boolean )**

This function controls where the output of a script will be displayed. If setPriceStudy(true) is used, then the output of the script will be plotted in the price pane of the chart. If setPriceStudy(false) is used, then the output of the script will be displayed in a separate indicator pane.

### Parameters

|         |               |
|---------|---------------|
| boolean | true or false |
|---------|---------------|

### Usage

```
function preMain() {  
  
    //we want this script to plot its results in the  
    //price pane  
    setPriceStudy(true);  
  
    setStudyTitle("Example Script");  
  
    //this script returns 3 values to be plotted on the chart.  
    //use setCursorLabelName() to set a name for each return  
    //value  
  
    setCursorLabelName("EMA", 0);  
    setCursorLabelName("SMA", 1 );  
    setCursorLabelName("WMA", 2 );  
  
    setDefaultBarFgColor( Color.blue, 0 );
```

```
    setDefaultBarFgColor( Color.red, 1 );
    setDefaultBarFgColor( Color.green, 2 );

    setShowTitleParameters( false );

}
```

## **setShowCursorLabel( boolean )**

This function controls whether or not the script return values are displayed in the eSignal cursor window. When setShowCursorLabel(false) is used, it will suppress the display of the return values in the cursor window. Note that the values returned by the script will still be plotted on the chart, they just will not be displayed in the cursor window.

### Parameters

|         |               |
|---------|---------------|
| boolean | true or false |
|---------|---------------|

### Usage

```
function preMain() {  
  
    //we want this script to plot its results in the  
    //price pane  
    setPriceStudy(true);  
  
    setStudyTitle("Example Script");  
  
    //this script returns 3 values to be plotted on the chart.  
    //use setCursorLabelName() to set a name for each return  
    //value  
  
    setCursorLabelName("EMA", 0);  
    setCursorLabelName("SMA", 1 );  
    setCursorLabelName("WMA", 2 );
```

```
    setDefaultBarFgColor( Color.blue, 0 );
    setDefaultBarFgColor( Color.red, 1 );
    setDefaultBarFgColor( Color.green, 2 );

    //if we wanted to suppress the display of the 3 return values
     //defined above, we could use the following call
    setShowCursorLabel(false);

    setShowTitleParameters( false );

}
```

## **setShowTitleParameters( boolean )**

This function controls whether or not the current script parameter values will be displayed to the right of the study title on a chart.

### Parameters

|         |               |
|---------|---------------|
| boolean | true or false |
|---------|---------------|

### Usage

```
function preMain() {  
  
    //we want this script to plot its results in the  
    //price pane  
    setPriceStudy(true);  
  
    setStudyTitle("Example Script");  
  
    //this script returns 3 values to be plotted on the chart.  
    //use setCursorLabelName() to set a name for each return  
    //value  
  
    setCursorLabelName("EMA", 0);  
    setCursorLabelName("SMA", 1 );  
    setCursorLabelName("WMA", 2 );  
  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarFgColor( Color.red, 1 );  
    setDefaultBarFgColor( Color.green, 2 );
```

```
//suppress the display of parameter values to the right
//of the script title
setShowTitleParameters( false );
}

}
```

### setStudyTitle( title )

This function is used to set the title of the study.

#### Parameters

|       |   |
|-------|---|
| title | a string containing the name or title of the script |
|-------|---|

#### Usage

```
function preMain() {  
  
    //we want this script to plot its results in the  
    //price pane  
    setPriceStudy(true);  
  
    //set the study title to "Example Script". this title  
    //will be displayed at the upper-left of the chart  
    setStudyTitle("Example Script");  
  
    //this script returns 3 values to be plotted on the chart.  
    //use setCursorLabelName() to set a name for each return  
    //value  
  
    setCursorLabelName("EMA", 0);  
    setCursorLabelName("SMA", 1 );  
    setCursorLabelName("WMA", 2 );  
  
    setDefaultBarFgColor( Color.blue, 0 );  
    setDefaultBarFgColor( Color.red, 1 );
```

```
        setDefaultBarFgColor( Color.green, 2 );  
  
        setShowTitleParameters( false );  
  
    }  
  
}
```

## **setStudyMax( value )**

This function is used to control the scaling of an indicator pane (*e.g.*, when `setPriceStudy(false)` is being used).

### Parameters

|       |                                      |
|-------|--------------------------------------|
| value | the maximum return value for scaling |
|-------|--------------------------------------|

### Usage

```
function preMain() {  
  
    //we want this script to plot its results in a  
    //separate indicator pane  
    setPriceStudy(false);  
  
    setStudyTitle("Example Script");  
  
    setCursorLabelName("RSI", 0);  
  
    setDefaultBarFgColor( Color.blue, 0 );  
  
    setShowTitleParameters( false );  
  
    //here we will set the maximum and minimum values  
    //for scaling purposes.  
    setStudyMax( 100 );  
    setStudyMin( 0 );
```



## **setStudyMin( value )**

This function is used to control the scaling of an indicator pane (*e.g.*, when `setPriceStudy(false)` is being used).

### Parameters

|       |                                      |
|-------|--------------------------------------|
| value | the minimum return value for scaling |
|-------|--------------------------------------|

### Usage

```
function preMain() {  
  
    //we want this script to plot its results in a  
    //separate indicator pane  
    setPriceStudy(false);  
  
    setStudyTitle("Example Script");  
  
    setCursorLabelName("RSI", 0);  
  
    setDefaultBarFgColor( Color.blue, 0 );  
  
    setShowTitleParameters( false );  
  
    //here we will set the maximum and minimum values  
    //for scaling purposes.  
    setStudyMax( 100 );  
    setStudyMin( 0 );
```



## repaintChart()

This function will force the chart to repaint. All drawn objects will be updated.

### Parameters

|      |                              |
|------|------------------------------|
| none | function takes no parameters |
|------|------------------------------|

### Usage

```
function main() {  
  
    ...  
    ...  
  
    if ( isLastBarOnChart() == true ) {  
  
        //force the chart to repaint  
        repaintChart();  
  
    }  
  
}
```

## setGlobalValue( varName, value )

This function creates a true global variable and sets it to the specified value. Global variables are accessible to other scripts running in other charts.

### Parameters

|         |  |
|---------|--|
| varName | the name to assign the global variable (text string) |
| value   | the value to assign the global variable              |

### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    if ( myVar > 0 ) {  
        setGlobalValue( "myStatus", 100 );  
    }  
    else {  
        setGlobalValue( "myStatus", -100 );  
    }  
  
    return( myVar );  
}
```

[getGlobalValue\(\)](#)  
[removeGlobalValue\(\)](#)

### **getGlobalValue( varName )**

This function retrieves the value of a global variable. Global variables are accessible to other scripts running in other charts.

#### Parameters

|         |   |
|---------|---|
| varName | the name that was assigned to the global variable |
|---------|---|

#### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    myVar = getGlobalValue( "myStatus" );  
    if ( myVar == 100 ) {  
        setBarFgColor( Color.green, 0 );  
    }  
    else if ( myVar == -100 ) {  
        setBarFgColor( Color.red, 0 );  
    }  
  
}  
  
setGlobalValue\(\)  
removeGlobalValue\(\)
```

### **removeGlobalValue( varName )**

This function removes a global variable from memory. Global variables are accessible to other scripts running in other charts.

#### Parameters

|         |   |
|---------|---|
| varName | the name that was assigned to the global variable |
|---------|---|

#### Usage

```
function main() {  
    var myVar;  
  
    ...  
    ...  
  
    myVar = getGlobalValue( "myStatus" );  
    if ( myVar != null ) {  
        removeGlobalValue( "myStatus" );  
    }  
}
```

[getGlobalValue\(\)](#)  
[setGlobalValue\(\)](#)

## **verifyBuild( buildNum )**

New in EFS2. Compares the user's eSignal build number to the buildNum value passed to the function. If the user's build number is older than the value passed, a message will be displayed on the user's chart along with a hyperlink to the eSignal download page.

### Parameters

|          |   |
|----------|---|
| buildNum | the build number to be used as the cutoff |
|----------|---|

### Usage

```
function main() {  
  
    //ensure that the user is at least running eSignal Build 719  
    verifyBuild( 719 );  
  
}  
  
getBuildNumber\(\)
```

This section describes the alert functions that are available in eSignal EFS:

[Alert.playSound\(\)](#)

[Alert.addToList\(\)](#)

[Alert.email\(\)](#)

### Alert.playSound( wavfile )

This function generates an audio alert using the sound file passed as a parameter.

**Note:** you can view/change the location of the Sound Files folder from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

#### Parameters

|         |  |
|---------|--|
| wavfile | full path to the WAV soundfile to play |
|---------|--|

#### Usage

```
var bCondition = false;  
  
function main() {  
  
    ...  
    ...  
    if ( bCondition==true ) {  
        Alert.playSound( "c:\program files\esignal\Sounds\Warning.wav" );  
        ...  
    }  
}
```

## **Alert.addToList( symbol, description, foreColor, backColor )**

This function generates a pop-up alert using the text and colors passed as parameters.

### Parameters

|             |  |
|-------------|--|
| symbol      | symbol for which to trigger the alert      |
| description | text description of alert                  |
| foreColor   | foreground <a href="#">color</a> for alert |
| backColor   | background <a href="#">color</a> for alert |

### Usage

```
var bCondition = false;  
  
function main() {  
  
    ...  
    ...  
    if ( bCondition==true ) {  
        Alert.addToList( "IBM", "Breakout!", Color.green, Color.black );  
        ...  
    }  
}
```

### Alert.email( body [, subject] [,prefix] )

This function generates an email alert using the text passed as parameters. Note that you must first set up your email configuration in the Alerts dialog in eSignal.

**Note:** you can customize the email settings from within the eSignal program. Click on **View-->Alerts...** to pull up the *Alert Edit/Status* window.

#### Parameters

|         |   |
|---------|---|
| body    | text for body of email  |
| subject | optional. text for subject line   |
| prefix  | optional. defaults to <b>true</b> . if set to <b>false</b> , the symbol will not be included before the Subject and Text of the email |

#### Usage

```
var bCondition = false;  
  
function main() {  
  
    ...  
    ...  
    if ( bCondition==true ) {  
        Alert.email( "Breakout alert for IBM", "Alert!!!" );  
        ...  
    }  
}
```



This section describes the mouse event-handler functions that are available in eSignal EFS:

[onLButtonDown\(\)](#)

[onLButtonUp\(\)](#)

[onLButtonDblClk\(\)](#)

[onRButtonDown\(\)](#)

[onRButtonUp\(\)](#)

[onRButtonDblClk\(\)](#)

## **onLButtonDown( barIndex, yValue [,x] [,y] )**

This function is an event handler for left-button mouse events. Be aware that onLButtonDown and onLButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

## **onLButtonUp( barIndex, yValue [,x] [,y] )**

This function is an event handler for left-button mouse events. Be aware that onLButtonDown and onLButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

## **onLButtonDblClk( barIndex, yValue [,x] [,y] )**

This function is an event handler for left-button double-click mouse events. Be aware that onLButtonDown and onLButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

## **onRButtonDown( barIndex, yValue [,x] [,y] )**

This function is an event handler for right-button mouse events. Be aware that onRButtonDown and onRButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

## **onRButtonUp( barIndex, yValue [,x] [,y] )**

This function is an event handler for right-button mouse events. Be aware that onRButtonDown and onRButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

## **onRButtonDblClk( barIndex, yValue [,x] [,y] )**

This function is an event handler for right-button double-click mouse events. Be aware that onRButtonDown and onRButtonUp also get called on double-click events.

### Parameters

|          |   |
|----------|---|
| barIndex | the bar index where mouse was clicked   |
| yValue   | the price value where mouse was clicked |
| x        | optional. x coordinate in study pane    |
| y        | optional. y coordinate in study pane    |

### Usage

```
function preMain() {  
    setPriceStudy(true);  
    setStudyTitle("Mouse Button Clicks");  
    setShowCursorLabel(false);  
  
}  
  
function main() {  
  
}  
  
function onLButtonDown( barIndex, yValue) {
```

```
    debugPrintln("LeftDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonUp( barIndex, yValue) {

    debugPrintln("LeftUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonDown( barIndex, yValue) {

    debugPrintln("RightDown: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onRButtonUp( barIndex, yValue) {

    debugPrintln("RightUp: " + barIndex + ", " + yValue);
    updateClickInfo(barIndex, yValue);

}

function onLButtonDblClk( barIndex, yValue) {

    debugPrintln("LeftDblClk: " + barIndex + ", " + yValue);
    updateDblClickInfo(barIndex, yValue);

}

function onRButtonDblClk( barIndex, yValue) {

    debugPrintln("RightDblClk: " + barIndex + ", " + yValue);
```

```
updateDblClickInfo(barIndex, yValue);

}

function updateClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 15, "Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"ClickInfo");

}

function updateDblClickInfo(barIndex, yValue) {

    drawTextAbsolute(5, 35, "Double Clicked bar " + barIndex + " at " +
yValue.toFixed(2),
    Color.white, Color.navy,
Text.RELATIVETOLEFT|Text.RELATIVETOBOTTOM|Text.BOLD, null, 14,
"DblClickInfo");
}
```

This section describes the Generic Broker functions that are available in eSignal EFS. These functions can be used to submit trades to any of the Integrated Brokers and they also work with the eSignal Paper Broker.

[buyMarket\(\)](#)

[buyLimit\(\)](#)

[buyStop\(\)](#)

[buyStopLimit\(\)](#)

[sellMarket\(\)](#)

[sellLimit\(\)](#)

[sellStop\(\)](#)

[sellStopLimit\(\)](#)

[sellShortMarket\(\)](#)

[sellShortLimit\(\)](#)

[sellShortStop\(\)](#)

[sellShortStopLimit\(\)](#)

[buy\(\)](#)

[sell\(\)](#)

## **buyMarket( symbol, quantity [, route] [, expiry] )**

Submit a market order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
buyMarket( "IDC", 100 );  
buyMarket( getSymbol(), 200 );
```

## **buyLimit( symbol, quantity, limit [, route] [, expiry] )**

Submit a limit order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| limit    | the limit price   |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
buyLimit( "IDC", 100, 15.00 );
buyLimit( getSymbol(), 200, getMostRecentAsk(), null, SB_GTC );
```

## **buyStop( symbol, quantity, stop [, route] [, expiry] )**

Submit a stop order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
buyStop( "IDC", 100, 15.00 );
buyStop( getSymbol(), 200, 98.75, null, SB_GTC );
```

**buyStopLimit( symbol, quantity, stop, limit [, route] [, expiry] )**

Submit a stop/limit order to broker.

Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| limit    | the limit price, must be $\geq$ stop price                            |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

Usage

```
buyStopLimit( "IDC", 100, 15.00, 15.25 );
buyStopLimit( getSymbol(), 200, 99.00, 100.00, null, SB_GTC );
```

## **sellMarket( symbol, quantity [, route] [, expiry] )**

Submit a market order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellMarket( "IDC", 100 );  
sellMarket( getSymbol(), 200 );
```

## **sellLimit( symbol, quantity, limit [, route] [, expiry] )**

Submit a limit order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| limit    | the limit price   |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellLimit( "IDC", 100, 15.00 );
sellLimit( getSymbol(), 200, 100.00, null, SB_GTC );
```

## **sellStop( symbol, quantity, stop [, route] [, expiry] )**

Submit a stop order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellStop( "IDC", 100, 15.00 );
sellStop( getSymbol(), 200, 99.00, null, SB_GTC );
```

## **sellStopLimit( symbol, quantity, stop, limit [, route] [, expiry] )**

Submit a stop/limit order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| limit    | the limit price, must be <= stop price                                |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellStopLimit( "IDC", 100, 15.00, 14.50 );
sellStopLimit( getSymbol(), 200, 98.75, 98.00, null, SB_GTC );
```

**sellShortMarket( symbol, quantity [, route] [, expiry] )**

Submit a market order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellShortMarket( "IDC", 100 );  
sellShortMarket( getSymbol(), 200 );
```

## **sellShortLimit( symbol, quantity, limit [, route] [, expiry] )**

Submit a limit order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| limit    | the limit price   |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellShortLimit( "IDC", 100, 19.00 );
sellShortLimit( getSymbol(), 200, 100.00, null, SB_GTC );
```

## **sellShortStop( symbol, quantity, stop [, route] [, expiry] )**

Submit a stop order to broker.

### Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

### Usage

```
sellShortStop( "IDC", 100, 15.00 );
sellShortStop( getSymbol(), 200, 99.00, null, SB_GTC );
```

**sellShortStopLimit( symbol, quantity, stop, limit [, route] [, expiry] )**

Submit a stop/limit order to broker.

Parameters

|          |   |
|----------|---|
| symbol   | the symbol for which to initiate the trade                            |
| quantity | the number of shares/contracts  |
| stop     | the stop price  |
| limit    | the limit price, must be <= stop price                                |
| route    | optional. the route or exchange for the trade (broker specific)       |
| expiry   | optional. expiration for the order:<br><br>SB_DAY<br>SB_GTC<br>SB_IOC |

Usage

```
sellShortStopLimit( "IDC", 100, 15.00, 14.50 );
sellShortStopLimit( getSymbol(), 200, 98.75, 98.00, null, SB_GTC );
```

## **buy( symbol )**

Submit a market order to broker using the pre-set default number of shares/contracts.

### Parameters

|        |  |
|--------|--|
| symbol | the symbol for which to initiate the trade |
|--------|--|

### Usage

```
buy( "IDC" );
buy( getSymbol() );
```

## **sell( symbol )**

Submit a market order to broker using the pre-set default number of shares/contracts.

### Parameters

|        |  |
|--------|--|
| symbol | the symbol for which to initiate the trade |
|--------|--|

### Usage

```
sell( "IDC" );
sell( getSymbol() );
```

This section describes the built-in EFS Objects for script menu creation, backtesting, DDE output, HTTP data retrieval and DLL access:

|   |   |
|---|---|
| <a href="#"><u>Strategy Object</u></a>          | methods for backtesting a trading system      |
| <a href="#"><u>FunctionParameter Object</u></a> | methods for creating a script parameter menu  |
| <a href="#"><u>File Object</u></a>              | methods for reading from and writing to files |
| <a href="#"><u>DDEOutput Object</u></a>         | methods for exporting data to a DDE client    |
| <a href="#"><u>HTTP Object</u></a>              | methods for downloading web data              |
| <a href="#"><u>DLL Object</u></a>               | methods for interfacing with DLLs             |

## Strategy Object

The Strategy Object allows EFS scripts to communicate with the eSignal Strategy Analyzer and perform backtests of trading systems.

### Strategy Methods

|                     |  |
|---------------------|--|
| doLong()            | enter a long position                          |
| doSell()            | exit a long position                           |
| doCover()           | exit a short position                          |
| doShort()           | enter a short position                         |
| isInTrade()         | returns true if system is currently in a trade |
| isLong()            | returns true if system is currently long       |
| isShort()           | returns true if system is currently short      |
| setStop()           | set a stop value                               |
| clearStop()         | clear a stop value                             |
| getPositionSize()   | returns the position size                      |
| getDefaultLotSize() | returns the default lot size                   |

Strategy.**doLong**(Description, Fill Type, Fill Bar, LotSize, StopOrLimit)

Strategy.**doSell**(Description, Fill Type, Fill Bar, LotSize, StopOrLimit)

Strategy.**doCover**(Description, Fill Type, Fill Bar, LotSize, StopOrLimit)

Strategy.**doShort**(Description, Fill Type, Fill Bar, LotSize, StopOrLimit)

Strategy.**isInTrade**()

Strategy.**isLong**()

Strategy.**isShort**()

**Strategy.setStop(dStop)**  
**Strategy.clearStop()**  
**Strategy.getPositionSize()**  
**Strategy.getDefaultLotSize()**

## Fill Type Constants

|                 |  |
|-----------------|--|
| Strategy.CLOSE  | uses the closing price as the fill price     |
| Strategy.LIMIT  | uses the StopOrLimit price as the fill price |
| Strategy.MARKET | uses the open price as the fill price        |
| Strategy.STOP   | uses the StopOrLimit price as the fill price |

## Fill Bar Constants

|                  |   |
|------------------|---|
| Strategy.THISBAR | uses the OHLC values of the current bar to determine the fill price in conjunction with the Fill Type |
| Strategy.NEXTBAR | uses the OHLC values of the next bar to determine the fill price in conjunction with the Fill Type    |

## LotSize Constants

|                  |  |
|------------------|--|
| Strategy.DEFAULT | uses the Default LotSize as the number of shares   |
| Strategy.ALL     | typically used with Strategy.doSell or Strategy.doCover; specifies that all shares being held to be sold/covered |

## Notes

- If calling `Strategy.doLong`, `Strategy.doShort`, `Strategy.doSell`, or `Strategy.doCover` and a Fill Type of `Strategy.LIMIT` or `Strategy.STOP` is provided, the `StopOrLimit` price must be specified.
- You must be long to use `Strategy.doSell`.
- You must be short to use `Strategy.doCover`.
- If a long position is currently held, calling `Strategy.doShort` will close the long position and enter a short position.
- If a short position is currently held, calling `Strategy.doLong` will close the short position and enter a long position.

## Usage

```
//assume a default lot size of 100

//go long 100 shares at the closing value of the current bar.
Strategy.doLong( "Go Long", Strategy.CLOSE, Strategy.THISBAR );

//go long 100 shares at the closing value of the current bar.
Strategy.doLong( "Go Long", Strategy.CLOSE, Strategy.THISBAR,
Strategy.DEFAULT );

//Limit order to go long 100 shares @ $20 if the low of the current bar is <= 20.
Strategy.doLong( "Go Long", Strategy.LIMIT, Strategy.THISBAR, null, 20 );

//If long, issue a stop sell order if high of the current bar >= 30.
if( Strategy.isLong() ) {
    Strategy.doSell( "Close Long", Strategy.STOP, Strategy.THISBAR,
Strategy.ALL, 30 );
}

//Set a stop order to sell all shares if high of any bar at or after the current bar
>= 30.
```

```
Strategy.setStop( 30 );
```

## FunctionParameter Object

The FunctionParameter Object class is a set of objects and functions that allow you to manage your input parameters in a EFS script.

### FunctionParameter Constructor

### **FunctionParameter( paramString, paramType )**

|             |   |
|-------------|---|
| paramString | the name of the parameter as defined in main()          |
| paramType   | the type of parameter object. see Parameter Types below |

### Parameter Types

|                           |  |
|---------------------------|--|
| FunctionParameter.STRING  | a string menu object. either a single string or a list of string options |
| FunctionParameter.NUMBER  | a numeric menu object  |
| FunctionParameter.COLOR   | a <a href="#">color</a> menu object                                      |
| FunctionParameter.BOOLEAN | a boolean menu object  |

### FunctionParameter Methods

|                 |  |
|-----------------|--|
| setName()       | set the display name of this menu item. This allows you to create a more descriptive name to display to the user |
| addOption()     | add a string option  |
| setLowerLimit() | set the lowest acceptable input value for a numeric menu option  |
| setUpperLimit() | set the highest acceptable input value for a numeric menu option   |

|                           |  |
|---------------------------|--|
| <code>setDefault()</code> | set the default value for this menu option |
|---------------------------|--|

## Usage

```

var aFPArray = new Array();

function preMain()
var x;

//initialize formula parameters
x=0;
//define a numerical menu option
aFPArray[x] = new FunctionParameter( "Param1",
FunctionParameter.NUMBER);
with( aFPArray[x] ) {
    setName( "Numeric Menu Option" );
    setLowerLimit( 5 );
    setUpperLimit( 125 );
    setDefault( 40 );
}
x++;
//define a string list menu option
aFPArray[x] = new FunctionParameter( "Param2",
FunctionParameter.STRING);
with( aFPArray[x] ) {
    setName( "String Menu Option" );
    addOption( "String1" );
    addOption( "String2" );
    setDefault( "String1" );
}
x++;
//define a color menu option
aFPArray[x] = new FunctionParameter( "Param3",
FunctionParameter.COLOR);

```

```
with( aFPArry[x] ) {
    setName( "Color Menu Option" );
    setDefault( Color.blue );
}
x++;
//define a boolean menu option
aFPArry[x] = new FunctionParameter( "Param4",
FunctionParameter.BOOLEAN);
with( aFPArry[x] ) {
    setName( "Bool Menu Option" );
    setDefault( true );
}

}

//make sure that our 4 menu parameters are included in main's declaration
function main( Param1, Param2, Param3, Param4 ) {

}

askForInput()
```

## File Object

The File Object allows EFS scripts to read/write files. All files created, read or written using the File object will be read from the eSignal Formula Output folder.

**Note:** you can view/change the location of the Formula Output folder from within the eSignal program. Click on **Tools-->EFS-->Settings** to pull up the *Formula Engine Settings* window.

### File Methods

|               |  |
|---------------|--|
| close()       | closes an open file  |
| eof()         | boolean. checks for end-of-file condition  |
| exists()      | boolean. checks for the existence of a file  |
| flush()       | writes the contents of the internal buffer to file   |
| getLength()   | returns the length of a file   |
| getPosition() | returns the current position of the pointer in an open file. the first byte of the file is 0   |
| isOpen()      | boolean. returns true if the file is open  |
| mkdir(sDir)   | boolean. creates sDir and returns true if successful   |
| open(sMode)   | boolean. opens the file in sMode and returns true if successful  |
| read(nBytes)  | reads nBytes of data from the file into a string   |
| readln()      | reads the current line from an open file and returns a string. line separator characters ("\n", "\r") are not included in the string |
|               |  |

|                                  |  |
|----------------------------------|--|
| <code>setPosition(nBytes)</code> | positions a pointer to nBytes in an open file                        |
| <code>write(sString)</code>      | boolean. writes sString to an open file                              |
| <code>writeln(sString)</code>    | boolean. writes sString along with a carriage return to an open file |

## Mode Values

|            |  |
|------------|--|
| <b>rt</b>  | opens a file for reading. If the file exists, the method succeeds and returns true; otherwise, the method fails and returns false.   |
| <b>wt</b>  | opens a file for writing. If the file does not already exist, it is created; otherwise, it is overwritten. This method always succeeds and returns true.   |
| <b>at</b>  | opens a file for appending (writing at the end of the file). If the file does not already exist, it is created. This method always succeeds and returns true.  |
| <b>rt+</b> | opens a file for reading and writing. If the file exists, the method succeeds and returns true; otherwise, the method fails and returns false. Reading and writing commence at the beginning of the file. When writing, characters at the beginning of the file are overwritten. |
| <b>wt+</b> | opens a file for reading and writing. If the file does not already exist, it is created; otherwise, it is overwritten. This method always succeeds and returns true.   |
| <b>at+</b> | opens a file for reading and appending. If the file does not already exist, it is created. This method always succeeds and returns true. Reading and appending commence at the end of the file.  |

## Usage

```
//read a file
var f = new File( "TestFolder/testReadWrite.txt" );
if( f.exists() ) {
    f.open( "rt" );
    var line;
```

```
while( !f.eof() ) {  
    line = f.readln();  
    debugPrintln( "Line: [" + line + "]" );  
}  
f.close();  
}
```

```
//create a directory  
var f = new File( "TestFolder" );  
f.mkdir();
```

[getFormulaOutputRoot\(\)](#)

## DDEOutput Object

The DDEOutput Object allows EFS to output formula data to Excel via a DDE interface.

### DDEOutput Methods

|            |                              |
|------------|------------------------------|
| set(value) | pass value to the DDE client |
|------------|------------------------------|

### Usage

```
var study      = null;
var dde        = new DDEOutput( "movavg" + getSymbol() );
var bInit      = false;

function preMain() {

    setPriceStudy(true);

}

function main() {
    var retVal;

    if ( bInit==false ) {
        //create an EMA study
        study = ema( 10 );
        bInit=true;
    }

    //gather the current value from the study
}
```

```
    retVal = study.getValue(0);

    //if it is not null, assign it to our DDE object
    if ( retVal != null) dde.set( retVal );

    return( retVal );
}
```

### Setup instructions for Excel

- 1. EFS formula with DDEOutput object must be running.
  2. In Excel, enter a DDE formula with the following format:  
**eSignal|EFS!movavgIBM**

The Excel configuration above would require that the example formula above be running on a chart of IBM. When the DDEOutput object is initialized in the EFS script (*i.e.*, **var dde = new DDEOutput(sItem)**), 'sItem' becomes the string that needs to be referenced in the Excel formula. In our example above sItem becomes "movavgIBM".

You can create multiple DDE objects within a single script. For example, if you implement a script that calculates 5 different indicators, you could create a DDEOutput Object for each indicator (remember to assign each DDEObject a unique string identifier) and then retrieve that same information from within an Excel spreadsheet.

# HTTP Object

The HTTP object allows EFS to open a file from a URL to import data for use within the EFS script. The HTTP functionality can be used to read text files (*e.g.*, .txt, .htm, .html, .xml, etc.)

**Note:** It is highly recommended that you do not use the HTTP object within the main() function where it has the potential of being called on every new trade. Instead, use the HTTP object either in the preMain() function or in custom user functions that are only called when needed.

### HTTP Methods

|          |                                 |
|----------|---------------------------------|
| open()   | URL of page to retrieve         |
| readln() | read a line from page           |
| eof()    | check for end-of-file condition |

### Usage

```
var v = new HTTP("http://www.somepage.com/whatever.txt");
OR
var v =new HTTP("http://www.somepage.com/something.asp?symbol=" +
getSymbol());
//After constructing the object, do something like this:
if(v.open()) {
    while(!v.eof()) {
        debugPrintln(v.readln());
```

```
    }  
}
```

*//Here's an example of a custom function for  
//loading a text file from a URL, which can  
//be called from preMain().  
//The MyFile.txt file would need to contain 6  
//numbers, each on a separate line. This example  
//populates a global array with the 6 numbers  
//from MyFile.txt.*

```
var aArray = new Array(6);
```

```
function preMain() {  
    loadData();  
}
```

```
function loadData() {  
    var f = new HTTP("http://www.somepage.com/MyFile.txt");  
    var i = 0;
```

```
    f.open("rt");  
    if (f.open()) {  
        for (i = 0; i < 6; ++i) {  
            aArray[i] = f.readLn();  
        }  
    }  
    f = null;  
    //debugPrintln(aArray);
```

```
    return;
```

```
}
```



## DLL Object

The eSignal EFS DLL Object allows you to interact with DLL files from within your EFS scripts.

### DLL Methods

|               |                           |
|---------------|---------------------------|
| addFunction() | add an exported function  |
| call()        | call an exported function |

**DLL.addFunction( refName, returnType, functionDefType, functionName [, parameters... ] )**

|                 |   |
|-----------------|---|
| refName         | text string to identify the function in EFS   |
| returnType      | the type of value that the DLL function returns. must be one of the following:<br><br>DLL.DOUBLE<br>DLL.INT<br>DLL.SHORT<br>DLL.FLOAT<br>DLL.STRING |
| functionDefType | the calling convention of the exported DLL function. must be one of the following:<br><br>DLL.CDECL<br>DLL.STDCALL                                  |
| functionName    | the name of the DLL function being called   |
| parameters      | the input parameters required by the  |

exported DLL function, if any. The following parameter types are supported:

DLL.BYTE  
DLL.STRING  
DLL.SHORT  
DLL.INT  
DLL.FLOAT  
DLL.DOUBLE  
DLL.VOID  
DLL.FLOATARRAY  
DLL.DOUBLEARRAY

**Note:** **functionName** is the name of the C/C++ function being called. Your compiler may mangle the function name. You will need to reference the MAP file generated by the linker to determine the actual name. In some cases the compiler prepends the function name with an underscore "\_". In other cases the compiler mangles the function name.

## Usage

```
var d = new DLL( "c:/testdll/myTest.DLL");
```

```
function preMain() {
```

```
    d.addFunction("MyCallName", DLL.DOUBLE, DLL.CDECL,  
    "Testing123", DLL.INT, DLL.STRING, DLL.FLOAT, DLL.DOUBLE,  
    DLL.BYTE);  
}
```

## DLL.call( refname [, parameters] )

|            |   |
|------------|---|
| refName    | text string used to identify the function |
| parameters | actual values to pass to the DLL          |

### Usage

```
var d = new DLL( "c:/testdll/myTest.DLL" );

function preMain() {

    d.addFunction( "MyCallName", DLL.DOUBLE, DLL.CDECL,
    "Testing123", DLL.INT, DLL.STRING, DLL.FLOAT, DLL.DOUBLE,
    DLL.BYTE );

    //in the above example:
    //      "MyCallName" is the internal name we have assigned to the DLL
    //      function we are calling
    //      DLL.DOUBLE is the type of value it returns (e.g., a double-
    //      precision number)
    //      DLL.CDECL is the calling convention used by our DLL
    //      "Testing123" is the actual function name of the exported DLL
    //      function.
    //      DLL.INT signifies that the first input parameter is an integer
    //      DLL.STRING signifies that the second input parameter is a string
    //      DLL.FLOAT signifies that the third input parameter is a float
    //      DLL.DOUBLE signifies that the fourth input parameter is a double
    //      DLL.BYTE signifies that the fifth input parameter is a byte

}

function main() {
var retVal;

    retVal = d.call( "MyCallName", 123, "hello world", 456.23, 544.0014053,
```

255 );

}

This section describes some of the predefined objects in core JavaScript: Array, String, Math, Number and Date.

[Array](#) methods for creating and manipulating arrays

[Object](#)

[String](#) methods for creating and manipulating strings

[Object](#)

[Math Object](#) methods to perform various math functions

[Number](#) methods for creating and manipulating numbers

[Object](#)

[Date Object](#) methods for creating and manipulating the Javascript date object

[Operators](#)

[Reserved Words](#)

## The Array Object in Javascript

### Array Constructors

```
Array();
Array( size );
Array( element1, element2, ..., elementN);
```

### Array Examples

```
var aMyArray = new Array();
var aMyArray = new Array( 20 );
var aMyArray = new Array( 1,5,2,20,99 );
var aMyArray = new Array( "Mon", "Tue", "Wed", "Thu", "Fri" );
```

### Array Usage

```
//Adding elements to an array
for (x=0; x<MyCount; x++) {
    aMyArray[x] = close(-x)/open(-x);
}
```

```
//Retrieving elements from an array
for (x=0; x<MyCount; x++) {
    nMyValue = aMyArray[x];
    debugPrint( "Value: " + nMyValue + "\n" );
}
```

## Array Methods and Properties

|   |  |
|---|--|
| concat(array2)                          | concatenates array2 into the array object  |
| join(separator)                         | returns a string consisting of all array elements concatenated with separator            |
| length                                  | returns the number of elements in the array  |
| pop()                                   | removes the last element from the array and returns that element                         |
| push()                                  | adds one or more elements to the end of an array and returns the new length of the array |
| shift()                                 | removes the first element from an array and returns that element                         |
| unshift()                               | adds one or more elements to the beginning of an array and returns the new array length  |
| reverse()                               | reverses all of the elements in the array object   |
| slice(start,end)                        | returns a section of the array from start to end   |
| splice(start, quant, [item1,...,itemN]) | changes the contents of an array, adding new elements while removing old elements        |
| sort(sortFunc)                          | sorts the array using the user-defined sortFunc()  |

## The String Object in Javascript

### String Constructors

```
var a = "This is a string";  
var b = new String("This is another string");
```

### String Methods and Properties

|                                   |  |
|-----------------------------------|--|
| charAt(index)                     | returns the character at index   |
| charCodeAt(index)                 | returns the ASCII value of the character at index  |
| indexOf(value, [startIndex] )     | returns the index within this string of the first occurrence of value starting at the optional startIndex                    |
| length                            | returns the length of the string   |
| lastIndexOf(value, [startIndex] ) | returns the index within this string of the last occurrence of value searching backwards starting at the optional startIndex |
| split(separator, limit)           | splits this string around the matches of separator   |
| substr(startIndex, length)        | returns a substring beginning at startIndex, extracting length characters.   |
| substring(startIndex, endIndex)   | returns a substring spanning from startIndex to endIndex   |
| toLowerCase()                     | returns a lower-case representation of the string  |
| toUpperCase()                     | returns a upper-case representation of the string  |



## The Math Object in Javascript

### Math Constructors

None.

### Math Properties

|         |  |
|---------|--|
| E       | Euler's constant and the base of natural logarithms    |
| LOG2E   | base 2 logarithm of E                                  |
| LOG10E  | base 10 logarithm of E                                 |
| LN2     | natural log of 2                                       |
| LN10    | natural log of 10                                      |
| PI      | ratio of the circumference of a circle to its diameter |
| SQRT2   | square root of 2                                       |
| SQRT1_2 | square root of 1/2                                     |

### Math Methods

|                       |  |
|-----------------------|--|
| abs(value)            | returns the absolute value of value                            |
| acos(value)           | returns the arc cosine of value                                |
| asin(value)           | returns the arc sine of value                                  |
| atan(value)           | returns the arc tangent of value                               |
| atan2(valueX, valueY) | returns the angle (in radians) from the x axis to a point(x,y) |
| ceil(value)           | returns the smallest integer greater than or equal to value    |
| cos(value)            | returns the cosine of value                                    |
| exp(value)            | returns E to the power of value                                |
| floor(value)          | returns the largest integer less than or equal to value        |
|                       |  |

|                                  |  |
|----------------------------------|--|
| <code>log(value)</code>          | returns the natural log of value               |
| <code>max(value1, value2)</code> | returns the maximum value of value1 and value2 |
| <code>min(value1, value2)</code> | returns the minimum value of value1 and value2 |
| <code>pow(base, exponent)</code> | returns base to the power of exponent          |
| <code>random()</code>            | returns a random number between 0 and 1        |
| <code>round(value)</code>        | returns value rounded to the nearest integer   |
| <code>sin(value)</code>          | returns the sine of value                      |
| <code>sqrt(value)</code>         | returns the square root of value               |
| <code>tan(value)</code>          | returns the tangent of value                   |

## The Number Object in Javascript

### Number Constructors

```
var n = new Number( 9 );
```

typically you would just directly assign the numeric value to your variable:

```
var n = 9;
```

### Number Methods

|                                 |  |
|---------------------------------|--|
| <code>toFixed(num)</code>       | converts a floating point number to a string with a specified number of digits   |
| <code>toString(radix)</code>    | converts a number to a string. radix (optional) is the base used to convert  |
| <code>toExponential(num)</code> | converts a number to a string in exponential notation with num representing the number of digits after the decimal point |

# The Date Object in Javascript

## Date Constructors

```
var newDateObject = new Date();
var newDateObject = new Date(dateValue);
var newDateObject = new Date(year, month, day, [hours], [minutes],
[seconds], [milliseconds]);
var newDateObject = new Date(year, month, day);
var newDateObject = new Date(year month, day, hour, minutes, seconds);
```

If dateValue is a numeric value, it represents the number of milliseconds since January 1, 1970 00:00:00. The ranges of dates is approximately 285,616 years from either side of midnight. Negative numbers indicate dates prior to 1970.

- Year: The full year (1980).
- Month: 0-11 (January to December)
- Day: 1-31
- Hour: 0-23
- Minutes: 0-59
- Seconds: 0-59
- Milliseconds: 0-999

## Remarks

Once the date object is constructed, methods can be accessed by the following syntax:

```
var today = new Date();
var h = today.getHours();
var s = today.toString();
```

- **Local Time** is defined as: the time on the computer from where the script is executed.
- **UTC (Universal Coordinated Time)** refers to the time as set by the World Time Standard. Also known as GMT (Greenwich Mean Time).

## Date Methods

|                     |   |
|---------------------|---|
| getDate()           | returns the day of the month of the date object according to local time   |
| getDay()            | returns the day of the week of the date object where 0 = Sunday   |
| getFullYear()       | returns the year of the date object according to local time   |
| getHours()          | returns the hour of the date object according to local time   |
| getMilliseconds()   | returns the milliseconds of the date object according to local time   |
| getMinutes()        | returns the minutes of the date object according to local time  |
| getMonth()          | returns the month of the date object according to local time  |
| getSeconds()        | returns the seconds of the date object according to local time  |
| getTime()           | returns the number of milliseconds since 1/1/1970 according to local time. negative numbers indicate a date prior to 1/1/1970 |
| getTimezoneOffset() | returns the timezone offset between local   |

|  |   |
|--|---|
|  | time and GMT  |
| getUTCDate()   | returns the day of the month of the date object according to UTC            |
| getUTCDay()  | returns the day of the week of the date object according to UTC. 0 = Sunday |
| getUTCFullYear()   | returns the year of the date object according to UTC                        |
| getUTCHours()  | returns the hours of the date object according to UTC                       |
| getUTCMilliseconds()   | returns the milliseconds of the date object according to UTC                |
| getUTCMilliseconds()   | returns the minutes of the date object according to UTC                     |
| getUTCMonth()  | returns the month of the date object according to UTC                       |
| getUTCSeconds()  | returns the seconds of the date object according to UTC                     |
| getYear()  | returns a two digit representation of the year in the date object           |
| parse(dateString)  | returns the number of milliseconds in dateString since 1/1/1970 local time  |
| setDate(domValue)  | set the day of the month of this date object according to local time        |
| setFullYear(yearValue, [monthValue][, [domValue] ])                            | set the year of this date object according to local time                    |
| setHours(hourValue, [minuteValue] [, [secondsValue] [, [millisecondsValue]]] ) | set the hour of this date object according to local time                    |
| setMilliseconds(millisecondValue)  | set the milliseconds of this date object according to local time            |
| setMinutes(minuteValue)  | set the minutes of this date object according to local time                 |
| setMonth(monthValue)   | set the month of this date object according to local time                   |
| setSeconds(secondValue,  | set the seconds of this date object according                               |

|  |   |
|--|---|
| [millisecondsValue])                                   | to local time   |
| setTime(millisecondsValue)                             | set the number of milliseconds since 1/1/1970 according to local time |
| setUTCDate(domValue)                                   | set the day of the month of this date object according to UTC         |
| setUTCFullYear(yearValue, [monthValue] [, [domValue]]) | set the year of this date object according to UTC                     |

## Operators in Javascript

### Arithmetic

|    |             |
|----|-------------|
| +  | addition    |
| -  | subtraction |
| ++ | increment   |
| -- | decrement   |
| *  | multiply    |
| /  | divide      |
| %  | modulus     |

### String

|    |             |
|----|-------------|
| +  | concatenate |
| += | append      |

### Logical

|    |     |
|----|-----|
| && | AND |
|    | OR  |
| !  | NOT |

### Bitwise

|    |                    |
|----|--------------------|
| &  | bitwise AND        |
| ^  | bitwise XOR        |
|    | bitwise OR         |
| ~  | bitwise NOT        |
| << | bitwise LEFT SHIFT |

|    |                     |
|----|---------------------|
| >> | bitwise RIGHT SHIFT |
|----|---------------------|

## Assignment

|     |          |
|-----|----------|
| =   | n = 2    |
| +=  | n = n+x  |
| -=  | n = n-x  |
| *=  | n = n*x  |
| /=  | n = n/x  |
| %-  | n = n%x  |
| &=  | n = n&x  |
| ^=  | n = n^x  |
| =   | n = n x  |
| <<= | n = n<<x |
| >>= | n = n>>x |

## Comparison

|    |                          |
|----|--------------------------|
| == | is equal to              |
| != | is NOT equal to          |
| >  | is greater than          |
| >= | is greater than or equal |
| <  | is less than             |
| <= | is less than or equal to |

### Reserved Words in Javascript

The following words are reserved in the EFS/Javascript language. Do not use these words to create variable or function names.

|          |          |
|----------|----------|
| break    | comments |
| continue | dowhile  |
| for      | forin    |
| function | ifelse   |
| return   | switch   |
| var      | while    |
| with     | main     |
| preMain  |          |

With EFS2 comes the ability to create and use Function Libraries.

[Function Libraries new in EFS2](#)

## Function Libraries

EFS2 has the ability to import custom libraries that can extend the language by providing useful utility functions, custom Series objects or custom expressions. All Function Libraries are stored in the FunctionLibraries folder off of the eSignal root (*i.e.*, C:\Program Files\esignal\FunctionLibrary) and they must be saved with a file extension of ".efsLib".

Function Libraries are simply collections of user-defined functions. There are no special formatting requirements and no required functions in a Function Library, but two functions are automatically called if they do exist:

### **initLib()**

This is called just after the EFS that loaded the library has finished its preMain() sequence and is called in the same context as preMain(). Any items that you wish to happen in preMain() can also be called from initLib().

### **shutdownLib()**

This is called just before the EFS that loaded the library has called its postMain() function.

Function Libraries can be added to any EFS script by calling the addLibrary() function in a global scope.

```
var myLibrary = addLibrary( "Tools.efsLib" );
```

Any functions or objects contained in the library can now be called from within the EFS script.