



dsTS Library Reference Guide

© 2006 Divergence Software, Inc.

dsTS Library Reference Guide

© 2006 Divergence Software, Inc.

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: November 2006

Table of Contents

Foreword	0
Part I dsTS Trading System Library	6
1 Introduction	6
2 Disclaimer	6
Part II Function Reference	8
1 Using the Library.....	8
2 Trade Management.....	8
ClassVar.Contracts	8
ClassVar.DoLong	8
ClassVar.DoShort	9
ClassVar.OnCloseOnly	9
ClassVar.AllowReversals	9
ClassVar.StartTime	10
ClassVar.EndTime	10
ClassVar.Slippage	10
ClassVar.MaxTradesPerDay	11
ClassVar.MaxProfitPerDay	11
ClassVar.MaxLossPerDay	11
ClassVar.Process()	12
ClassVar.AddLongRule()	12
ClassVar.AddShortRule()	12
ClassVar.GoLong()	13
ClassVar.GoShort()	13
ClassVar.ScaleInLong()	13
ClassVar.ScaleInShort()	14
ClassVar.GoSell()	14
ClassVar.GoCover()	15
ClassVar.ScaleOutLong()	15
ClassVar.ScaleOutShort()	16
ClassVar.GoSellEOD()	16
ClassVar.GoCoverEOD()	16
ClassVar.GetTradePL()	17
ClassVar.GetStatus()	17
ClassVar.GetEntryPrice()	17
ClassVar.GetContracts()	17
ClassVar.GetBarsInTrade()	17
3 Profit Target Management.....	18
ClassVar.ProfitTarget	18
ClassVar.ProfitTargetPercent	18
4 Stop Management.....	18
ClassVar.StopUseClose	19
ClassVar.UseFixedPointStop	19
ClassVar.FixedPointStop	19
ClassVar.UseFixedPercentStop	20
ClassVar.FixedPercentStop	20

ClassVar.UseTrailingStop	20
ClassVar.TrailingStop	21
ClassVar.UseATRStop	21
ClassVar.ATRStop	22
ClassVar.ATRsToUse	22
ClassVar.ATRPeriod	23
ClassVar.UseTimeStop	24
ClassVar.TimeStop	24
ClassVar.UseCustomStop	24
ClassVar.CustomStop	25
ClassVar.GetStopValue()	26
5 Alert Management.....	26
ClassVar.SoundBuy	26
ClassVar.SoundSell	26
ClassVar.SoundStop	27
ClassVar.SoundTarget	27
ClassVar.SetAlertSounds()	27
ClassVar.SetAlertTypes()	27
ClassVar.GenerateAlert()	28
6 Display Management.....	28
ClassVar.FontSize	28
ClassVar.Font	28
ClassVar.ColorLong	29
ClassVar.ColorShort	29
ClassVar.ColorStop	29
ClassVar.ColorTarget	29
ClassVar.DrawText	30
ClassVar.DisplaySignals	30
ClassVar.DrawLegend()	30
ClassVar.DrawStats()	30
Part III Examples	33
1 Simple Example.....	33
2 Complex Example.....	35
Part IV Troubleshooting	41
1 Troubleshooting Tips.....	41
Index	42

Part



1 dsTS Trading System Library

1.1 Introduction

Rev: 10/30/2006

The **Divergence Software, Inc.** Trading System Library (**dsTS**) encapsulates all of the logic necessary to implement a back testable trading system in eSignal into one, easy-to-use library. While there are many options and features that can be adjusted by the end-user, practically any existing script can be converted into a full-fledged trading system with as few as 4 lines of code and most any trading system, no matter how complex, can be implemented using the **dsTS Library** in a fraction of the time it would require without the **dsTS Library**. The **dsTS Library** allows you to focus on your trading system logic by taking care of all of the "back room" details such as integration with the eSignal Strategy Analyzer, trade, stop and profit target management as well as buy/sell signal display and alert management.

Chris Kryza
Divergence Software, Inc.
ckryza@sr-analyst.com
www.sr-analyst.com



1.2 Disclaimer

The **dsTS Library** is provided as-is with no warranties and/or guarantees. Results obtained from the use of the **dsTS Library** or any information provided in this Developer's Reference, while believe to be reliable, are not guaranteed to be accurate or profitable. It should be understood that investing or speculating in the financial markets involves risk and may result in either a partial or total loss of one's investment capital.

Divergence Software, Inc. and/or its principals and employees will accept no liability whatsoever for any loss arising from any use of the **dsTS Library** or the information contained in this Developer's Reference. **Divergence Software, Inc.** does not offer trading advice of any kind and we are solely involved in the business of financial software development and education.

Reproduction, duplication, or redistribution of any material in this Developer's Reference, in any form, without prior written permission from **Divergence Software, Inc.** is strictly prohibited. Other brand and product names used in this Developer's Reference may be registered trademarks of their respective owners. Their use is for identification purposes only.



Part



2 Function Reference

2.1 Using the Library

Calling the dsTS Library

Any script that will make use of the dsTS functionality must first import the library. This is done by adding the following call at the top of the script:

```
Var myLib = addLibrary( "dsTS.efsLib" );
```

You then need to create an instance of the TradingSystem class, as follows:

```
ClassVar = new myLib.TradingSystem();
```

Note that ClassVar must be a variable that is external to the main() and preMain() functions (see the Example Script in this section).

Because of the class-style implementation, it is possible to define more than one trading system within a single script. So you could theoretically define two separate and distinct trading systems, each with their own rules, stops, alerts, colors, etc. and both could be run from within the same script.

Note: Every script that uses the dsTS library **MUST** include the dsTS Process() function in main() so that it is called on each iteration of the script.

[Simple Example](#)

[Complex Example](#)

2.2 Trade Management

2.2.1 ClassVar.Contracts

ClassVar.Contracts

Use this property to set the default number of contracts to trade in the Strategy Analyzer. This property can be set to any integer greater than 0. The default value is 100. Note that this value should only be set once in the initialization section of your script (see earlier example code). If you need to determine the total number of currently open contracts, use the GetContracts() function which is described near the end of this section.

Example:

```
TS.Contracts = 250;
```

2.2.2 ClassVar.DoLong

ClassVar.DoLong

Use this property to enable/disable Long trades. This property can be set to either **true** or **false**. The default value is **true**.

Example:

```
TS.DoLong = true;
```

The above setting would enable Long trades.

2.2.3 ClassVar.DoShort

ClassVar.DoShort

Use this property to enable/disable Short trades. This property can be set to either **true** or **false**. The default value is **true**.

Example:

```
TS.DoShort = false;
```

The above setting would disable Short trades.

2.2.4 ClassVar.OnCloseOnly

ClassVar.OnCloseOnly

Use this property to tell the Trading System class when trades should be evaluated and this property can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.OnCloseOnly = true;
```

If OnCloseOnly is set to **true**, all Buy and Sell rules will be evaluated as of the close of each bar and trade entry will take place as of the open of the following bar for historical bars as well as for realtime bars.

If OnCloseOnly is set to **false**, all Buy and Sell rules will be evaluated on each tick for realtime bars but evaluated on bar close for all historical bars in your chart.

Note that stops and limit orders will always take affect on the bar on which the condition was met.

2.2.5 ClassVar.AllowReversals

ClassVar.AllowReversals

This property determines whether or not a reversal is allowed once a trade has been entered and can be set to either **true** or **false**. The default value is **true**.

Example:

```
TS.AllowReversals = false;
```

If the AllowReversals property is set to false, a Long trade can only be closed when either the stop or profit target is hit. If a Short signal evaluates to true at any time during the Long trade, it will be ignored, and vice-versa for a Short trade.

2.2.6 ClassVar.StartTime

ClassVar.StartTime

This property allows you to define a trading session that will be used when an Intraday bar interval is loaded in your chart. This property is ignored when an Intraday interval is not being used. This property accepts a string value that contains a time (HH:MM) in military format (e.g., a 24-hour clock). The default value is **null** which means that the property will not be used.

Example:

```
TS.StartTime = "09:30";  
TS.EndTime = "16:15";
```

The above example would set the start of trading to 9:30am and the end of trading to 4:15pm. Note that, on any Intraday chart, Buy and Sell rules would not be evaluated outside of the defined session. If you use StartTime then you must use EndTime and vice versa. If you use StartTime and EndTime and you wish to set up a 24-hour trading session, the correct method is to set StartTime to "00:00" and EndTime to "24:00".

Also, StartTime and EndTime can cross day boundaries if StartTime is greater than EndTime. For example, a StartTime of "09:30" and an EndTime of "16:15" would allow trades to be executed between 9:30am and 4:15pm on the same day. However a StartTime of "11:30" and an EndTime of "10:30" would allow trades to be executed from 11:30am up until 10:30am the next morning.

2.2.7 ClassVar.EndTime

ClassVar.EndTime

This property allows you to define a trading session that will be used when an Intraday bar interval is loaded in your chart. This property is ignored when an Intraday interval is not being used. This property accepts a string value that contains a time (HH:MM) in military format (e.g., a 24-hour clock). The default value is **null** which means that the property will not be used.

Example:

```
TS.StartTime = "09:30";  
TS.EndTime = "16:15";
```

The above example would set the start of trading to 9:30am and the end of trading to 4:15pm. Note that, on any Intraday chart, Buy and Sell rules would not be evaluated outside of the defined session. If you use StartTime then you must use EndTime and vice versa. If you use StartTime and EndTime and you wish to set up a 24-hour trading session, the correct method is to set StartTime to "00:00" and EndTime to "24:00".

Also, StartTime and EndTime can cross day boundaries if StartTime is greater than EndTime. For example, a StartTime of "09:30" and an EndTime of "16:15" would allow trades to be executed between 9:30am and 4:15pm on the same day. However a StartTime of "11:30" and an EndTime of "10:30" would allow trades to be executed from 11:30am up until 10:30am the next morning.

2.2.8 ClassVar.Slippage

ClassVar.Slippage

This property allows you to define a slippage amount per trade to be used by the trading system. Any value you enter here will only be used internally by the dsTS library when calculating and displaying the

basis system statistics (see [ClassVar.DrawStats\(\)](#))

Example:

```
TS.Slippage = 0.25;
```

The above example would set the slippage amount per trade to 0.25. This means that as each trade is completed, the amount of 0.25 would be deducted from the net trade profit/loss for purposes of the calculations in the DrawStats() function.

2.2.9 ClassVar.MaxTradesPerDay

ClassVar.MaxTradesPerDay

This property allows you to limit the total number of trades taken per day and is only applicable to Intraday charts.

Example:

```
TS.MaxTradesPerDay = 2;
```

The above example would limit the system to only taking a maximum of 2 trades per trading day.

2.2.10 ClassVar.MaxProfitPerDay

ClassVar.MaxProfitPerDay

This property allows you to limit the total number of trades taken per day and is only applicable to Intraday charts. If MaxProfitPerDay is set to a value greater than 0, then trades will be taken each day up until the total profit for the day is greater than or equal to the value you have specified. The value you specify should be in points (not dollars).

Example:

```
TS.MaxProfitPerDay = 3.5;
```

In the above example, trading will cease on any trading day if the total profit for the day is greater than or equal to 3.5 points.

2.2.11 ClassVar.MaxLossPerDay

ClassVar.MaxLossPerDay

This property allows you to limit the total number of trades taken per day and is only applicable to Intraday charts. If MaxLossPerDay is set to a value greater than 0, then trades will be taken each day up until the total loss for the day is greater than or equal to the value you have specified. The value you specify should be in points (not dollars). Note that, even though we are talking about measuring "loss", you should enter the MaxLossPerDay value as a positive number.

Example:

```
TS.MaxLossPerDay = 6.0;
```

In the above example, trading will cease on any trading day if the total loss for the day is greater than or equal to 6 points.

2.2.12 ClassVar.Process()

ClassVar.Process()

This is the most important function in the dsTS library and it must be used in every script that makes use of the dsTS library. The Process() function manages all aspects of the trading system, such as evaluating buy/sell rules and monitoring stops and profit targets. Process() will return the stop value if/when a trade is entered.

Example:

```
StopValue = TS.Process();
```

2.2.13 ClassVar.AddLongRule()

ClassVar.AddLongRule(Expression)

This member function is used to add buy rules into the system. Note that you can add as many rules as you need (e.g, multiple buy and/or sell rules) by simply calling the functions multiple times with different expressions.

Expression can be any valid mathematical equation and can make use of any variables that have external scope. In our example script we used the following:

```
//add our long/short rules for the CCI  
TS.AddLongRule( "nCCI>-150 && nCCI_1<=-150" );  
TS.AddShortRule( "nCCI<150 && nCCI_1>=150" );
```

These two rules simply state that we want to enter a Long trade when CCI crosses up through -150 and we want to enter a Short trade when CCI crosses down through +150.

The "nCCI" and "nCCI_1" variables were declared outside of the main() and preMain() functions and, therefore, are available for use within any *Expression* that we would like to create.

Any rules that we set up using the AddLongRule() function will be evaluated each time the ClassVar.Process() function is called.

2.2.14 ClassVar.AddShortRule()

ClassVar.AddShortRule(Expression)

This member function is used to add sell rules into the system. Note that you can add as many rules as you need (e.g, multiple buy and/or sell rules) by simply calling the functions multiple times with different expressions.

Expression can be any valid mathematical equation and can make use of any variables that have external scope. In our example script we used the following:

```
//add our long/short rules for the CCI  
TS.AddLongRule( "nCCI>-150 && nCCI_1<=-150" );  
TS.AddShortRule( "nCCI<150 && nCCI_1>=150" );
```

These two rules simply state that we want to enter a Long trade when CCI crosses up through -150 and we want to enter a Short trade when CCI crosses down through +150.

The "nCCI" and "nCCI_1" variables were declared outside of the main() and preMain() functions and, therefore, are available for use within any *Expression* that we would like to create.

Any rules that we set up using the AddShortRule() function will be evaluated each time the ClassVar.Process() function is called.

2.2.15 ClassVar.GoLong()

ClassVar.GoLong([Price])

This function will open a Long trade. If no Price parameter is passed, the order will be treated as a Market Order and the current ClassVar.OnCloseOnly setting will be honored.

If a Price parameter is passed, then the order will be treated as a Limit Order and a trade will be opened the moment the Price value is reached.

Example:

```
TS.GoLong( 1382.50 );
```

The above example would attempt to go Long at a price of 1382.50 on the current bar.

Note that this function would typically be used in lieu of the AddLongRule() function.

2.2.16 ClassVar.GoShort()

ClassVar.GoShort([Price])

This function will open a Short trade. If no Price parameter is passed, the order will be treated as a Market Order and the current ClassVar.OnCloseOnly setting will be honored.

If a Price parameter is passed, then the order will be treated as a Limit Order and a trade will be opened the moment the Price value is reached.

Example:

```
TS.GoShort( 1382.50 );
```

The above example would attempt to go Short at a price of 1382.50 on the current bar.

Note that this function would typically be used in lieu of the AddShortRule() function.

2.2.17 ClassVar.ScaleInLong()

ClassVar.ScaleInLong(Price, Contracts)

This function allows you to add to an existing Long position and would typically be used in conjunction with the GoLong() function. The Price and Contracts parameters are both required and each ScaleInLong() call will be treated as a Limit Order.

Example:

```
nOpenContracts = TS.GetContracts();
nEntryPrice = TS.GetEntryPrice();

if ( nOpenContracts == 100 ) {
    TS.ScaleInLong( nEntryPrice + 1.00, 200 );
}
if( nOpenContracts == 300 ) {
    TS.ScaleInLong( nEntryPrice + 2.00, 200 );
}
```

The above example assumes that we entered our Long trade initially by purchasing 100 shares/contracts. It uses the GetContracts() and GetEntryPrice() functions to monitor the status of our open Long trade. If price moves up and exceeds the initial trade entry price by 1 point, we will add to our position by purchasing an additional 200 contracts. Total open contracts will now be 300 contracts. If price then moves up 2 points past our initial entry price, we will purchase yet another 200 contracts at that price. Since our total number of open contracts is now 500, the logic in the two "if" statements in the example will prevent any further ScaleInLong() calls.

2.2.18 ClassVar.ScaleInShort()

ClassVar.ScaleInShort(Price, Contracts)

This function allows you to add to an existing Short position and would typically be used in conjunction with the GoShort() function. The Price and Contracts parameters are both required and each ScaleInShort() call will be treated as a Limit Order.

Example:

```
nOpenContracts = TS.GetContracts();
nEntryPrice = TS.GetEntryPrice();

if ( nOpenContracts == 100 ) {
    TS.ScaleInLong( nEntryPrice + 1.00, 200 );
}
if( nOpenContracts == 300 ) {
    TS.ScaleInLong( nEntryPrice + 2.00, 200 );
}
```

The above example assumes that we entered our Long trade initially by purchasing 100 shares/contracts. It uses the GetContracts() and GetEntryPrice() functions to monitor the status of our open Long trade. If price moves up and exceeds the initial trade entry price by 1 point, we will add to our position by purchasing an additional 200 contracts. Total open contracts will now be 300 contracts. If price then moves up 2 points past our initial entry price, we will purchase yet another 200 contracts at that price. Since our total number of open contracts is now 500, the logic in the two "if" statements in the example will prevent any further ScaleInLong() calls.

2.2.19 ClassVar.GoSell()

ClassVar.GoSell([Price] [,Text])

This function will close a Long trade. If no Price parameter is passed, the order will be treated as a Market Order and the current ClassVar.OnCloseOnly setting will be honored.

If a Price parameter is passed, then the order will be treated as a Limit Order and a trade will be closed the moment the Price value is reached.

The Text parameter is an optional string that will be displayed on the chart. If no Text parameter is passed, then "Sell" will be displayed on the chart.

Example:

```
TS.GoSell( 150.22 );
```

The above example would attempt to Sell at a price of 150.22. If we were not already Long, the call itself would be ignored.

Note that this function would typically be used in lieu of the AddLongRule() function.

2.2.20 ClassVar.GoCover()

ClassVar.GoCover([Price] [,Text])

This function will close a Short trade. If no Price parameter is passed, the order will be treated as a Market Order and the current ClassVar.OnCloseOnly setting will be honored.

If a Price parameter is passed, then the order will be treated as a Limit Order and a trade will be closed the moment the Price value is reached.

The Text parameter is an optional string that will be displayed on the chart. If no Text parameter is passed, then "Covr" will be displayed on the chart.

Example:

```
TS.GoCover( 150.22 );
```

The above example would attempt to Cover at a price of 150.22. If we were not already Short, the call itself would be ignored.

Note that this function would typically be used in lieu of the AddShortRule() function.

2.2.21 ClassVar.ScaleOutLong()

ClassVar.ScaleOutLong(Price, Contracts)

This function allows you to scale out of an existing Long position and would typically be used in conjunction with the GoLong() function. The Price and Contracts parameters are both required and each ScaleOutLong() call will be treated as a Limit Order.

Example:

```
nOpenContracts = TS.GetContracts();  
nEntryPrice = TS.GetEntryPrice();  
  
if ( nOpenContracts == 100 ) {  
    TS.ScaleOutLong( nEntryPrice + 1.00, 50 );  
    TS.ScaleOutShort( nEntryPrice - 1.00, 50 );  
}
```

```
}
```

The above example assumes that we entered our Long or Short trade initially by purchasing 100 shares/contracts. It uses the `GetEntryPrice()` and `GetContracts()` functions to monitor the status of our open trade. If the Long or Short trade achieves a profit of 1 point, we will take a profit by scaling out of $\frac{1}{2}$ of our position (e.g., 50 contracts out of the original 100 contracts) at the indicated price level.

2.2.22 **ClassVar.ScaleOutShort()**

ClassVar.ScaleOutShort(Price, Contracts)

This function allows you to scale out of an existing Short position and would typically be used in conjunction with the `GoShort()` function. The `Price` and `Contracts` parameters are both required and each `ScaleOutShort()` call will be treated as a Limit Order.

Example:

```
nOpenContracts = TS.GetContracts();
nEntryPrice = TS.GetEntryPrice();

if ( nOpenContracts == 100 ) {
    TS.ScaleOutLong( nEntryPrice + 1.00, 50 );
    TS.ScaleOutShort( nEntryPrice - 1.00, 50 );
}
```

The above example assumes that we entered our Long or Short trade initially by purchasing 100 shares/contracts. It uses the `GetEntryPrice()` and `GetContracts()` functions to monitor the status of our open trade. If the Long or Short trade achieves a profit of 1 point, we will take a profit by scaling out of $\frac{1}{2}$ of our position (e.g., 50 contracts out of the original 100 contracts) at the indicated price level.

2.2.23 **ClassVar.GoSellEOD()**

ClassVar.GoSellEOD()

This function will submit a Market Order to close a Long trade. The EOD (e.g., End of Day) designation simply refers to the fact that this function would typically be used to close an open trade at the close of a specified trading session.

Example:

```
TS.GoSellEOD();
```

The above example would close any open Long trade at the end of the trading session.

2.2.24 **ClassVar.GoCoverEOD()**

ClassVar.GoCoverEOD()

This function will submit a Market Order to close a Short trade. The EOD (e.g., End of Day) designation simply refers to the fact that this function would typically be used to close an open trade at the close of a specified trading session.

Example:

```
TS.GoCoverEOD();
```

The above example would close any open Short trade at the end of the trading session.

2.2.25 **ClassVar.GetTradePL()**

ClassVar.GetTradePL()

This function will return the profit/loss of the currently active trade, if any.

Example:

```
CurrentPL = TS.GetTradePL();
```

If we were in a Long trade, the above example would return the current profit or loss we have in that trade. If no trade is currently active then this call would return **null**.

2.2.26 **ClassVar.GetStatus()**

ClassVar.GetStatus()

This function returns the current trade status and will return either 1 (for Long), 0 (for flat) or -1 (for Short).

Example:

```
MyTradeStatus = TS.GetStatus();
```

2.2.27 **ClassVar.GetEntryPrice()**

ClassVar.GetEntryPrice()

This function returns the initial entry price of the currently open trade or **null** if no trade is currently open.

Example:

```
nEntryPrice = TS.GetEntryPrice();
```

2.2.28 **ClassVar.GetContracts()**

ClassVar.GetContracts()

This function returns the total number of open shares/contracts or 0 (zero) if no trade is currently open.

Example:

```
nContracts = TS.GetContracts();
```

2.2.29 **ClassVar.GetBarsInTrade()**

ClassVar.GetBarsInTrade()

This function will return the number of bars that we have been in the current trade.

Example:

```
MyBarsInTrade = TS.GetBarsInTrade();
```

2.3 Profit Target Management

2.3.1 ClassVar.ProfitTarget

ClassVar.ProfitTarget

This property allows you to set a profit target, in points. Once a trade is entered it will be monitored automatically and if/when the profit target is reached, the trade will be closed.

Example:

```
TS.ProfitTarget = 8.0;
```

This example sets the profit target to 8 points. If/when any trade reaches 8 points in profit, the trade will automatically be closed.

2.3.2 ClassVar.ProfitTargetPercent

ClassVar.ProfitTargetPercent

This property allows you to set a profit target as a percentage of the trade entry price. Once a trade is entered it will be monitored automatically and if/when the profit target is reached, the trade will be closed.

Example:

```
TS.ProfitTargetPercent = 1.0;
```

This example sets the profit target to 1% of the trade entry price. If/when any trade reaches the equivalent of 1% in profit, the trade will automatically be closed.

2.4 Stop Management

Stop Management Properties and Functions

There are 6 different kinds of stops that you can use in conjunction with the dsTS library:

- Fixed Point Stop
- Fixed Percentage Stop
- Trailing Stop
- ATR Ratchet Stop
- Time Stop
- Custom Stop

Once configured, any and all stops that are defined will be evaluated each time the ClassVar.Process() function is called and any open trades will be closed, as necessary. You can also combine different stop types (e.g., an Fixed Point initial stop combined with an ATR Ratchet stop would be a good combination). All stop types being used will be evaluated and the one that is closest to the current

price will be used.

2.4.1 ClassVar.StopUseClose

ClassVar.StopUseClose

This property allows you to define the behavior of any stops you have created and can be set to either **true** or **false**. The default value is **false**.

If this property is set to true, then a bar must close through the stop price for the stop to trigger. If this property is set to false then a stop is triggered whenever price trades through the stop price.

Example:

```
TS.StopUseClose = true;
```

2.4.2 ClassVar.UseFixedPointStop

ClassVar.UseFixedPointStop

ClassVar.FixedPointStop

The first property allows you to toggle the use of a Fixed Point Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseFixedPointStop = true;
```

The second property allows you to set the actual stop, in points.

Example:

```
TS.FixedPointStop = 5.25;
```

2.4.3 ClassVar.FixedPointStop

ClassVar.UseFixedPointStop

ClassVar.FixedPointStop

The first property allows you to toggle the use of a Fixed Point Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseFixedPointStop = true;
```

The second property allows you to set the actual stop, in points.

Example:

```
TS.FixedPointStop = 5.25;
```

2.4.4 ClassVar.UseFixedPercentStop

[ClassVar.UseFixedPercentStop](#)
[ClassVar.FixedPercentStop](#)

The first property allows you to toggle the use of a Fixed Percentage Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseFixedPercentStop = true;
```

The second property allows you to set the actual stop, in percent.

Example:

```
TS.FixedPercentStop = 1.5;
```

This example would set the stop to 1.5% of the trade entry price.

2.4.5 ClassVar.FixedPercentStop

[ClassVar.UseFixedPercentStop](#)
[ClassVar.FixedPercentStop](#)

The first property allows you to toggle the use of a Fixed Percentage Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseFixedPercentStop = true;
```

The second property allows you to set the actual stop, in percent.

Example:

```
TS.FixedPercentStop = 1.5;
```

This example would set the stop to 1.5% of the trade entry price.

2.4.6 ClassVar.UseTrailingStop

[ClassVar.UseTrailingStop](#)
[ClassVar.TrailingStop](#)

The first property allows you to toggle the use of a Trailing Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseTrailingStop = true;
```

The second property allows you to set the number of bars that will be used to create the trailing stop. For Long trades, the trailing stop will be set to the Lowest-Low within the number of bars that you specify. For Short trades, the trailing stop will be set to the Highest-High within the number of bars that you specify.

Example:

```
TS.TrailingStop = 24;
```

The example above would result in the trailing stop being set to the lowest-low of the last 24 bars (for a Long trade) and the highest-high of the last 24 bars (for a Short trade).

2.4.7 ClassVar.TrailingStop

[ClassVar.UseTrailingStop](#)
[ClassVar.TrailingStop](#)

The first property allows you to toggle the use of a Trailing Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseTrailingStop = true;
```

The second property allows you to set the number of bars that will be used to create the trailing stop. For Long trades, the trailing stop will be set to the Lowest-Low within the number of bars that you specify. For Short trades, the trailing stop will be set to the Highest-High within the number of bars that you specify.

Example:

```
TS.TrailingStop = 24;
```

The example above would result in the trailing stop being set to the lowest-low of the last 24 bars (for a Long trade) and the highest-high of the last 24 bars (for a Short trade).

2.4.8 ClassVar.UseATRStop

[ClassVar.UseATRStop](#)
[ClassVar.ATRStop](#)
[ClassVar.ATRstoUse](#)
[ClassVar.ATRPeriod](#)

The first property allows you to toggle the use of a ATR Ratchet Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseATRStop = true;
```

The second property the ATR Ratchet percentage that will be used by the stop to dynamically increment (e.g., ratchet) value. The default value is 0.005 and this should be fine for most applications.

Example:

```
TS.ATRStop = 0.015;
```

The third property allows you to set the number of ATRs to use in construction of the stop. The default value is 2.0.

Example:

```
TS.ATRstoUse = 5;
```

The fourth property allows you to set the lookback period to be used in construction of the stop. The default value is 20 (bars).

Example:

```
TS.ATRPeriod = 35;
```

2.4.9 ClassVar.ATRStop

[ClassVar.UseATRStop](#)
[ClassVar.ATRStop](#)
[ClassVar.ATRstoUse](#)
[ClassVar.ATRPeriod](#)

The first property allows you to toggle the use of a ATR Ratchet Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseATRStop = true;
```

The second property the ATR Ratchet percentage that will be used by the stop to dynamically increment (e.g., ratchet) value. The default value is 0.005 and this should be fine for most applications.

Example:

```
TS.ATRStop = 0.015;
```

The third property allows you to set the number of ATRs to use in construction of the stop. The default value is 2.0.

Example:

```
TS.ATRstoUse = 5;
```

The fourth property allows you to set the lookback period to be used in construction of the stop. The default value is 20 (bars).

Example:

```
TS.ATRPeriod = 35;
```

2.4.10 ClassVar.ATRstoUse

[ClassVar.UseATRStop](#)
[ClassVar.ATRStop](#)
[ClassVar.ATRstoUse](#)
[ClassVar.ATRPeriod](#)

The first property allows you to toggle the use of a ATR Ratchet Stop. It can be set to either **true** or

false. The default value is **false**.

Example:

```
TS.UseATRStop = true;
```

The second property the ATR Ratchet percentage that will be used by the stop to dynamically increment (e.g., ratchet) value. The default value is 0.005 and this should be fine for most applications.

Example:

```
TS.ATRStop = 0.015;
```

The third property allows you to set the number of ATRs to use in construction of the stop. The default value is 2.0.

Example:

```
TS.ATRstoUse = 5;
```

The fourth property allows you to set the lookback period to be used in construction of the stop. The default value is 20 (bars).

Example:

```
TS.ATRPeriod = 35;
```

2.4.11 ClassVar.ATRPeriod

[ClassVar.UseATRStop](#)

[ClassVar.ATRStop](#)

[ClassVar.ATRstoUse](#)

[ClassVar.ATRPeriod](#)

The first property allows you to toggle the use of a ATR Ratchet Stop. It can be set to either **true** or **false**. The default value is **false**.

Example:

```
TS.UseATRStop = true;
```

The second property the ATR Ratchet percentage that will be used by the stop to dynamically increment (e.g., ratchet) value. The default value is 0.005 and this should be fine for most applications.

Example:

```
TS.ATRStop = 0.015;
```

The third property allows you to set the number of ATRs to use in construction of the stop. The default value is 2.0.

Example:

```
TS.ATRstoUse = 5;
```

The fourth property allows you to set the lookback period to be used in construction of the stop. The default value is 20 (bars).

Example:

```
TS.ATRPeriod = 35;
```

2.4.12 ClassVar.UseTimeStop

[ClassVar.UseTimeStop](#)
[ClassVar.TimeStop](#)

The first property allows you to toggle the use of a user-defined Time Stop. It can be set to either **true** or **false**. The default value is **false**. The Time Stop feature allows you to specify the maximum length, in bars, of a trade.

Example:

```
TS.UseTimeStop = true;
```

The second property allows you to specify the number of bars that will be used to implement the Time Stop. The example below would set a Time Stop of 15 bars which means that any open trade would be closed as of the open of the 16th bar of the trade.

Example:

```
TS.TimeStop = 15;
```

2.4.13 ClassVar.TimeStop

[ClassVar.UseTimeStop](#)
[ClassVar.TimeStop](#)

The first property allows you to toggle the use of a user-defined Time Stop. It can be set to either **true** or **false**. The default value is **false**. The Time Stop feature allows you to specify the maximum length, in bars, of a trade.

Example:

```
TS.UseTimeStop = true;
```

The second property allows you to specify the number of bars that will be used to implement the Time Stop. The example below would set a Time Stop of 15 bars which means that any open trade would be closed as of the open of the 16th bar of the trade.

Example:

```
TS.TimeStop = 15;
```

2.4.14 ClassVar.UseCustomStop

[ClassVar.UseCustomStop](#)
[ClassVar.CustomStop](#)

The first property allows you to toggle the use of a user-defined Custom Stop. It can be set to either **true** or **false**. The default value is **false**. With this Custom Stop feature, you can implement and test a wide variety of stop systems.

Example:

```
TS.UseCustomStop = true;
```

The second property allows you to pass a function to the dsTS library that will define your custom stop. This function will be evaluated each time that the ClassVar.Process() function is called.

Example:

```
TS.CustomStop = MyCustomStop;
```

Outside of main(), you would need to write the "MyCustomStop" function. Here is an example:

```
//user-defined custom stop logic
function MyCustomStop() {
    if ( TS.GetBarsInTrade()>3 ) {
        if ( TS.GetStatus()==1 ) {
            return( Math.min( low(-1), low(-2), low(-3) ) );
        }
        if ( TS.GetStatus()==-1 ) {
            return( Math.max( high(-1), high(-2), high(-3) ) );
        }
    }
    return( null );
}
```

The function above would design a stop that waits until at least 3 bars have elapsed in the trade and then it would set the stop to either the lowest-low of the last 3 bars (for a Long trade) or the highest-high of the last 3 bars (for a Short trade).

2.4.15 ClassVar.CustomStop

ClassVar.UseCustomStop ClassVar.CustomStop

The first property allows you to toggle the use of a user-defined Custom Stop. It can be set to either **true** or **false**. The default value is **false**. With this Custom Stop feature, you can implement and test a wide variety of stop systems.

Example:

```
TS.UseCustomStop = true;
```

The second property allows you to pass a function to the dsTS library that will define your custom stop. This function will be evaluated each time that the ClassVar.Process() function is called.

Example:

```
TS.CustomStop = MyCustomStop;
```

Outside of main(), you would need to write the "MyCustomStop" function. Here is an example:

```
//user-defined custom stop logic
function MyCustomStop() {

    if ( TS.GetBarsInTrade()>3 ) {

        if ( TS.GetStatus()==1 ) {
            return( Math.min( low(-1), low(-2), low(-3) ) );
        }
        if ( TS.GetStatus()==-1 ) {
            return( Math.max( high(-1), high(-2), high(-3) ) );
        }

    }

    return( null );

}
```

The function above would design a stop that waits until at least 3 bars have elapsed in the trade and then it would set the stop to either the lowest-low of the last 3 bars (for a Long trade) or the highest-high of the last 3 bars (for a Short trade).

2.4.16 ClassVar.GetStopValue()

ClassVar.GetStopValue()

This function will return the current stop value (or null if no stop is active).

Example:

```
StopValue = TS.GetStopValue();
```

2.5 Alert Management

2.5.1 ClassVar.SoundBuy

ClassVar.SoundBuy

This property allows you to set the sound that will be associated with Buys.

Example:

```
TS.SoundBuy = "WARNING.WAV";
```

[ClassVar.SetAlertSounds\(\)](#)

2.5.2 ClassVar.SoundSell

ClassVar.SoundSell

This property allows you to set the sound that will be associated with Sells.

Example:

```
TS.SoundSell = "WARNING.WAV";
```

```
ClassVar.SetAlertSounds\(\)
```

2.5.3 ClassVar.SoundStop

ClassVar.SoundStop

This property allows you to set the sound that will be associated with Stops being hit.

Example:

```
TS.SoundStop = "WARNING.WAV";
```

```
ClassVar.SetAlertSounds\(\)
```

2.5.4 ClassVar.SoundTarget

ClassVar.SoundTarget

This property allows you to set the sound that will be associated with Profit Targets being hit.

Example:

```
TS.SoundTarget = "WARNING.WAV";
```

```
ClassVar.SetAlertSounds\(\)
```

2.5.5 ClassVar.SetAlertSounds()

ClassVar.SetAlertSounds(BuySound, [SellSound], [StopSound], [TargetSound])

The SetAlertSounds() function allows you to set all of the stop sounds that you would like to use in one call. The parameters that you pass to this function should be strings that contain the file name of a WAV file that exists in your eSignal Sounds subdirectory. All sounds default to "BULLET.WAV".

Example:

```
TS.SetAlertSounds( "BULLET.WAV", "DING.WAV", "BOING.WAV", "BUZZ.WAV" );
```

```
ClassVar.SoundBuy
```

```
ClassVar.SoundSell
```

```
ClassVar.SoundStop
```

```
ClassVar.SoundTarget
```

2.5.6 ClassVar.SetAlertTypes()

ClassVar.SetAlertTypes(Audible, PopUp, Email)

This function accepts Boolean values and allows you to set the specific type of alerts that you want to receive and they are Audible Alerts, PopUp Alerts and Email Alerts.

Example:

```
TS.SetAlertTypes( true, false, false );
```

This example would enable Audible alerts but disable PopUp and Email alerts. You would then use the SetAlertSounds() function (or the individual SoundBuy, SoundSell, SoundStop and SoundTarget properties) to set the WAV files to use for sound alerts.

[ClassVar.SetAlertSounds\(\)](#)
[ClassVar.SoundBuy](#)
[ClassVar.SoundSell](#)
[ClassVar.SoundStop](#)
[ClassVar.SoundTarget](#)

2.5.7 ClassVar.GenerateAlert()

ClassVar.GenerateAlert(WAVFile, Direction, Message, [Price])

This function allows you to manually generate an alert. It requires that you pass the WAV file to be used, the Trade Direction (e.g., 1 for Long, 2 for Short), the Alert Message that you want to display (for PopUp alerts) and Price Value.

Example:

```
TS.GenerateAlert( "DING.WAV", 1, "Go Long!!", close(0) );
```

The example above will force an alert to be generated and, depending upon the alert types that have been activated, will play DING.WAV as an Audible Alert and will PopUp a message in the Alerts dialog that says "Go Long!! -> 100.50" (if the price passed to the function was 100.50).

[ClassVar.SetAlertTypes\(\)](#)
[ClassVar.SetAlertSounds\(\)](#)

2.6 Display Management

2.6.1 ClassVar.FontSize

ClassVar.FontSize
ClassVar.Font

These two properties allow you to set the fontsize and font that will be used when drawing the buy/sell signals on the chart. The default fontsize is 12 and the default font is null (meaning that it will use whatever you have set as your default font in eSignal).

Example:

```
TS.Font = "Arial";  
TS.FontSize = 15;
```

2.6.2 ClassVar.Font

ClassVar.FontSize
ClassVar.Font

These two properties allow you to set the fontsize and font that will be used when drawing the buy/sell signals on the chart. The default fontsize is 12 and the default font is null (meaning that it will use whatever you have set as your default font in eSignal).

Example:

```
TS.Font = "Arial";  
TS.FontSize = 15;
```

2.6.3 ClassVar.ColorLong

ClassVar.ColorLong

This property allows you to set the color that will be used to draw Long signals on the chart. The default Long color is green, the default Short color is red, the default Stop color is magenta and the default Profit Target color is blue.

Example:

```
TS.ColorLong = Color.teal;  
TS.ColorStop = Color.black;
```

2.6.4 ClassVar.ColorShort

ClassVar.ColorShort

This property allows you to set the color that will be used to draw Short signals on the chart. The default Long color is green, the default Short color is red, the default Stop color is magenta and the default Profit Target color is blue.

Example:

```
TS.ColorLong = Color.teal;  
TS.ColorStop = Color.black;
```

2.6.5 ClassVar.ColorStop

ClassVar.ColorStop

This property allows you to set the color that will be used to draw Stop signals on the chart. The default Long color is green, the default Short color is red, the default Stop color is magenta and the default Profit Target color is blue.

Example:

```
TS.ColorLong = Color.teal;  
TS.ColorStop = Color.black;
```

2.6.6 ClassVar.ColorTarget

ClassVar.ColorTarget

This property allows you to set the color that will be used to draw Profit Target signals on the chart. The default Long color is green, the default Short color is red, the default Stop color is magenta and the default Profit Target color is blue.

Example:

```
TS.ColorLong = Color.teal;  
TS.ColorStop = Color.black;
```

2.6.7 ClassVar.DrawText

ClassVar.DrawText

This property allows you to toggle between displaying text or symbol buy/sell signals on the chart. It can be set to either **true** or **false**. If set to **true**, text signals will be drawn. If set to **false**, symbols will be used to identify the signals.

Example:

```
TS.DrawText = true;
```

2.6.8 ClassVar.DisplaySignals

ClassVar.DisplaySignals

This property allows you to toggle the display of signals in general. It can be set to either **true** or **false**. If set to **true** then signals will be displayed on the chart. If set to **false**, then signals will not be displayed on the chart.

Example:

```
TS.DisplaySignals = false;
```

2.6.9 ClassVar.DrawLegend()

ClassVar.DrawLegend()

This function will force the display of a small symbol legend at the bottom-left of your chart if ClassVar.DrawText is set to **false**.

Example:

```
TS.DrawText = false;  
TS.DrawLegend();
```

2.6.10 ClassVar.DrawStats()

ClassVar.DrawStats(Decimals, Color, FontSize)

This function will calculate and draw some basic trading system statistics on the chart. This can save you a lot of time if you are tweaking a trading system and just want a quick idea of how the changes you make impact the bottom line. The stats information will be drawn in the lower-left of your chart in a Courier font. You should NOT call this function on each new bar or tick, rather you should use the "isLastBarOnChart()" construct as in the example code below.

The function takes three parameters:

- Decimals – the number of decimals of precision with which to round price values.
- Color – the color to be used when drawing the stats.
- FontSize – the font size to use when drawing the stats.

Example:

```
if ( isLastBarOnChart() ) {
```

```
} TS.DrawStats( 2, Color.purple, 12 );
```

Part



3 Examples

3.1 Simple Example

A Simple Example

The following code listing shows how easy it is to create a simple stop-and-reverse system using the CCI indicator:

```
//External Variables

var myLib = addLibrary( "dsTS.efsLib");

var TS                = null;
var nBarCounter       = 0;
var nCCI              = null;
var nCCI_1            = null;
var bInitialized      = false;

//== PreMain function required by eSignal to set things up
function preMain() {
var x;

    setPriceStudy(true);
    setStudyTitle("Using dsTS (Simple Stop and Reverse)");
    setShowTitleParameters( false );
}

//== Main processing function
function main() {
var x;

    //script is initializing
    if ( getBarState() == BARSTATE_ALLBARS ) {
        return null;
    }

    //Note: this block of code only gets executed once!
    //This is typically where you would set all of the dsTS
    //properties. By doing it this way, the properties are
    //set only once when the script first initializes
    if ( bInitialized == false ) {

        //create our CCI indicator
        nStudy1 = getSeries( cci( 35, "close" ) );

        //create our Trading System object
        TS = new myLib.TradingSystem();

        //Add our Long and Short logic. You can add as many rules
        //as you want
        //add our long/short rules for the CCI

        TS.AddLongRule( "nCCI>-150 && nCCI_1<=-150" );
        TS.AddShortRule( "nCCI<150 && nCCI_1>=150" );

        bInitialized = true;
    }

    //called on each new bar
    if ( getBarState() == BARSTATE_NEWBAR ) {
```

```

        //keep track of prior bar's value
        nCCI_1 = nCCI;

        nBarCounter++;
    }

    //gather our indicator values
    nCCI = nStudy1.getValue( 0 );

    //call the dsTS Process function which will take care of
    //all of the details
    TS.Process();
}

```

If you load the above script into an eSignal chart, all of the buys and sells will be displayed and you will be able to backtest the logic by right-clicking in your chart and selecting the "Tools/Backtesting" menu option.

Want to add audible and popup alerts to this trading system? Just add one more line of code:

```
TS.SetAlertTypes( true, true, false );
```

Want to add a 5-point fixed stop? Just two lines of code:

```
TS.UseFixedPointStop = true;
TS.FixedPointStop = 5.0;
```

Want to add an adaptive trailing stop as well? Just 2 lines of code will add an ATR Ratchet Stop:

```
TS.UseATRStop = true;
TS.ATRPeriod = 20;
```

Want to add a 8-point profit target? One line of code:

```
TS.ProfitTarget = 8.0;
```

Want to restrict the system so that it only takes trades between 9:30am and 4:00pm if it is run on an Intraday chart? Two lines of code:

```
TS.StartTime = "09:30";
TS.EndTime = "16:00";
```

Want to see how the trading system looks taking Long trades only? One line of code:

```
TS.DoShort = false;
```

Want to limit the system to 3 trades per day? One line of code:

```
TS.MaxTradesPerDay = 3;
```

Basically, you define all of the properties of the dsTS Library that you want to use only once (typically within an "if" block of code that only gets executed once as in the above example) and then you place the call to the Process() function in the main body of your code where it will get called every time the script itself gets called. So even a complex trading system with profit targets, multiple stop types, user-defined trading times, etc. should only take a few lines to code to implement. With the dsTS Library, practically any existing script can be quickly converted into a trading system. See the end of this document for an example of a more involved trading system implementation.

3.2 Complex Example

A More Complex Example

```
//External Variables

var myLib = addLibrary( "dsTS.efsLib");

var TS                                = null;
var grID                              = 0;
var nBarCounter                       = 0;
var nStudy1                           = null;
var nStudy2                           = null;
var nStudy3                           = null;
var nCCI                              = null;
var nCCI_1                            = null;

var bInitialized                      = false;

//== PreMain function required by eSignal to set things up
function preMain() {
var x;

    setPriceStudy(true);
    setStudyTitle("dsTradingSystem Object Class Example #2");
    setCursorLabelName("Stop", 0);
    setDefaultBarFgColor( Color.purple, 0 );
    setPlotType( PLOTTYPE_FLATLINES, 0 );
    setDefaultBarThickness( 2, 0 );
    setShowTitleParameters( false );

    grID = 0;
}

//== Main processing function
function main( ) {
var x;

    //script is initializing
    if ( getBarState() == BARSTATE_ALLBARS ) {
        return null;
    }

    if ( bInitialized == false ) {

        //create our basic indicators
        nStudy1 = getSeries( cci( 35, "close" ) );

        //create our Trading System object
        TS = new myLib.TradingSystem();

        //The various TradingSystem options can now be set. Note
        //that you really do not need
        //to set any of these options (other than at least 1
        //BuyRule and/or 1 ShortRule).

        //Add our Long and Short logic. You can add as many rules
        //as you want

        //add our long/short rules for the Moving Average
        TS.AddLongRule( "nCCI>-150 && nCCI_1<=-150" );
        TS.AddShortRule( "nCCI<150 && nCCI_1>=150" );
    }
}

```

```

//At this point, you are done with the requirements. You
//do not need to define anything else.
//The code below, however, shows you the settings that
//you CAN change only if you want to.

//Note: As an alternative to using the BuyRule and SellRule
//methods, you can also initiate trades by calling one of
//these methods directly from within your
//    main() function logic.
//TS.GoLong( [Price] );
//TS.GoShort( [Price] );
//TS.GoSell( [Price] );
//TS.GoCover( [Price] );
//TS.GoSellEOD();
//TS.GoCoverEOD();
//In all cases, Price is optional and the order becomes a
//market order. If price is specified, the order becomes a
//stop order and will only be trigger if Price is hit.

//set the # of contract we want to trade
TS.Contracts    = 250;

//Specify if we want to do Long or Short Trades or Both
TS.DoShort      = true;
TS.DoLong       = true;

//Specify start/end times if we want to - must be military
//time. These only kick in if using Intraday bar interval
//This will restrict trading to occur only between the
//hours you set
TS.StartTime    = "09:30";
TS.EndTime      = "15:50";

//Do we check for signals on close of bar only or on each
//tick?
TS.OnCloseOnly = false;

//Profit Targets and Stops

//Set our Profit Target, in points (if any)
TS.ProfitTarget = 8.0;

//There are 5 types of stop that you can use:
//1 - Fixed Point Stop
//2 - Fixed Percentage Stop
//3 - Trailing Stop - lowest-low of last X bars if long,
//   highest-high of last X bars if short
//4 - ATR Ratchet Stop
//5 - User-Defined custom stop

//You turn the on/off like this (they all default to Off)
TS.UseFixedPointStop    = true;
TS.UseFixedPercentStop  = false;
TS.UseTrailingStop      = false;
TS.UseATRStop           = true;
TS.UseCustomStop        = false;

//Set our Initial Fixed Point Stop, in points (if any)
TS.FixedPointStop       = 3.0;

TS.FixedPercentStop     = 1; //e.g., 1%

//set the number of bars to use for our trailing stop, if
//any
//For a Long trade, if used the trailing stop will be the
//lowest-low of the last [TrailingStop] bars
//For a Short trade, if used the trailing stop will be the
//highest-high of the last [TrailingStop] bars

TS.TrailingStop         = 12;

```

```

//ATR Ratchet Stop
TS.ATRStop           = 0.005;
TS.ATRstoUse         = 1.5;
TS.ATRPeriod        = 20;

//you can define our own custom stop logic if you want to.
//See the example user-defined custom stop
//function right after the end of the main() function
TS.CustomStop       = MyCustomStop;

//Must price actually close through a stop?
TS.StopUseClose     = false;

//Here are the Alert Methods and Properties

//4 WAV files can be specified for Audible Alerts: Buy,
//Sell, Stop and ProfitTarget. You simply pass the
//file names as they appear in your eSignal Sounds
//subdirectory. Default value for all is "BULLET.WAV"

TS.SetAlertSounds( "BULLET.WAV", "DING.WAV", "BOING.WAV", "BUZZ.WAV" );

//You can also set the sounds individually, if you prefer:
TS.SoundBuy         = "BULLET.WAV";
TS.SoundSell        = "DING.WAV";
TS.SoundStop        = "BOING.WAV";
TS.SoundTarget      = "BUZZ.WAV";

//3 Alert Types are available: Audible, Popup and Email.
//You can use any or all (or any combination) by
//setting the appropriate parameter to either true or
//false.

TS.SetAlertTypes( false, true, false );

//Note that if you ever want to manually generate an alert
//from within your code, you can use the following
//method

//The parameters are WAVfile, 1=Buy 2=Sell, Alert Text, Price
//TS.GenerateAlert( "DING.WAV", 1, "My Alert", close(0) );

//Here are some cosmetic settings we can change

//you can adjust the font size and the font to be used
TS.FontSize         = 11;
TS.Font             = "Arial";

//you can adjust the colors used when drawing the text and
//symbols
TS.ColorLong        = Color.green;
TS.ColorShort       = Color.red;
TS.ColorStop        = Color.magenta;
TS.ColorTarget      = Color.blue;

//you can select between drawing symbols for buy/sells or
//drawing the actual "Buy" "Sell" text.
TS.DrawText         = true;

//Do we want to display the buy/sell signals on the chart?
TS.DisplaySignals   = true;

//Cosmetic - draw a symbol legend at bottom-left of chart
TS.DrawLegend();

bInitialized = true;

```

```

}

//called on each new bar
if ( getBarState() == BARSTATE_NEWBAR ) {

    //code below is remmed out but demonstrates an alternative
    //way of using this library to manually create your
    //own signals
    /*
    if ( nCCI>-150 && nCCI_1<=-150 ) {
        TS.GoLong();
    }
    if ( nCCI<150 && nCCI_1>=150 ) {
        TS.GoShort();
    }
    */

    //keep track of prior bar's value
    nCCI_1 = nCCI;

    nBarCounter++;

}

//gather our indicator values
nCCI = nStudy1.getValue( 0 );

//get the current stop amount, if any, for display purposes
nStop = TS.GetStopValue();

//If a new trade is entered on this bar, there will be a new stop
//amt so get it
if ( nStop==null ) {
    //Process causes the trading strategy to be analyzed for
    //this bar
    nStop = TS.Process();
}
else {
    //Process causes the trading strategy to be analyzed for
    //this bar
    TS.Process();
}

//various trade information functions can be called if we want to
x = TS.GetStatus();
x = TS.GetBarsInTrade();
x = TS.GetTradePL();

//draw our basic trading system stats at lower-left of chart
if ( isLastBarOnChart() ) {
    TS.DrawStats( 2, Color.blue, 11 );
}

return( nStop );

}

//user-defined custom stop logic
function MyCustomStop() {

    if ( TS.GetBarsInTrade()>3 ) {

        if ( TS.GetStatus()==1 ) {
            return( Math.min( low(-1), low(-2), low(-3) ) );
        }
        if ( TS.GetStatus()==-1 ) {
            return( Math.max( high(-1), high(-2), high(-3) ) );
        }
    }

}

```

```
    return( null );  
}
```

Part



4 TroubleShooting

4.1 TroubleShooting Tips

Troubleshooting

The most common problem you are likely to encounter is calling a non-existent dsTS library function. This can occur if you misspell a function name. The problem is that no error will be generated by the EFS engine and the offending line of code will simply be ignored. So it is extremely important that you check your work on a regular basis.

Example:

```
MyBarsInTrade = TS.GetBarsInTrade();
```

The above example is the correct way to call the function.

```
MyBarsInTrade = TS.GetBarsinTrade();
```

In the example above, we forgot to capitalize the "in" in GetBarsInTrade. No error will be generated but no value will be returned either because there is no function in the dsTS library called "GetBarsinTrade".

The second most common problem is forgetting to add the TS.Process() function to your script. The Process function **MUST** be included in all scripts that make use of the dsTS Library. It manages all aspects of the trading system, such as evaluating buy/sell rules and monitoring stops and profit targets. Generally, the best place to put this function call is just before any final return statement in your script so that you will be sure that it is called each time the script is called.

Index

- A -

AddLongRule() 12
 AddShortRule() 12
 AllowReversals 9
 ATRPeriod 23
 ATRStop 22
 ATRsToUse 22

- C -

ColorLong 29
 ColorShort 29
 ColorStop 29
 ColorTarget 29
 Contracts 8
 CustomStop 25

- D -

DisplaySignals 30
 DoLong 8
 DoShort 9
 DrawLegend() 30
 DrawStats() 30
 DrawText 30

- E -

EndTime 10

- F -

FixedPercentStop 20
 FixedPointStop 19
 Font 28
 FontSize 28
 Functions 12, 13, 14, 15, 16, 17, 26, 27, 28, 30

- G -

GenerateAlert() 28

GetBarsInTrade() 17
 GetContracts() 17
 GetEntryPrice() 17
 GetStatus() 17
 GetStopValue() 26
 GetTradePL() 17
 GoCover() 15
 GoCoverEOD() 16
 GoLong() 13
 GoSell() 14
 GoSellEOD() 16
 GoShort() 13

- M -

MaxLossPerDay 11
 MaxProfitPerDay 11
 MaxTradesPerDay 11

- O -

OnCloseOnly 9

- P -

Process() 12
 Profit Target Percent 18
 ProfitTarget 18
 Properties 8, 9, 10, 11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

- S -

ScaleInLong() 13
 ScaleInShort() 14
 ScaleOutLong() 15
 ScaleOutShort() 16
 SetAlertSounds() 27
 SetAlertTypes() 27
 Slippage 10
 SoundBuy 26
 SoundSell 26
 SoundStop 27
 SoundTarget 27
 StartTime 10
 StopUseClose 19

- T -

TimeStop 24

TrailingStop 21

- U -

UseATRStop 21

UseCustomStop 24

UseFixedPercentStop 20

UseFixedPointStop 19

UseTimeStop 24

UseTrailingStop 20

Back Cover