Creating Optimizers in Wealth-Lab Pro®

Introduction

Wealth-Lab Pro® includes the ability to "optimize" a trading Strategy. The process of optimization consists of trying a range of different parameter values in succession and analyzing the results of all of the runs. Optimization can help determine if a trading Strategy is robust, or is the result of a statistical fluke.

In Wealth-Lab Pro, the optimization process is extendible via the **Optimizer** base class. Classes that derive from this base implement an optimization algorithm that can be used in the product. WL comes with two concrete implementations out of the box, the **Exhaustive** and **MonteCarlo** methods, but programmers can create their own Optimizers that integrate seamlessly with the software.

Creating a new Optimizer

To create a new Optimizer, start with a .NET class library assembly in your development tool of choice. Add a reference to the WealthLab.DLL assembly that you'll find in the WL Pro executable folder. Create a new class and assign **Optimizer** as the base class. Be sure to add "WealthLab" to your source code's **using** statement so your library can locate the Optimizer base class.

The **Optimization** base class consists of a number of properties and methods, some informational, and some you must override. The primary methods involved in performing the optimization are **FirstRun** and **NextRun**. WL calls **FirstRun** the first time an optimization is started. Here you will normally set the **WealthScript** parameter values to their starting values. WL will then execute the Strategy using the parameter values that you assigned.

Example of Exhaustive FirstRun method:

```
//Set the parameters to their starting values
public override void FirstRun()
{
    foreach (StrategyParameter sp in WealthScript.Parameters)
        sp.Value = sp.Start;
}
```

WL then calls the **NextRun** method in your **Optimizer** class, passing as parameters the results of the previous optimization run. Here you should change the **WealthScript** parameters to the next set of values that should be tested. Return true from **NextRun** if another optimization run should be executed, or false if the optimization is completed at this point.

Finally, WL calls **RunCompleted**, allowing your **Optimizer** a chance to update its custom user interface tabs, if it has created any (see below).

Setting the Parameters

The job of the **FirstRun** and **NextRun** methods is to assign values to the parameters of the **Strategy** that is being optimized. You can do this by accessing the **WealthScript** property of the **Optimization** base class. The **WealthScript** property returns the instance of the **WealthScript**-derived class that represents the **Strategy** being executed. It contains a property called **Parameters** which is of type **List<StrategyParameter>**. Each **StrategyParameter** instance in the list has properties that return the **Start**, **Stop**, and **Step** values, and has a property called **Value** that you should assign within **FirstRun**/**NextRun**.

Examining the Results of an Optimization Run

When WL calls the **NextRun** method, it passes your **Optimizer** two parameters that contain information about the results of the previous optimization run. These two parameters are:

SystemPerformance sp - Contains an instance of a **SystemPerformance** class that contains standard performance results and equity curves of the previous optimization run.

OptimizationResult or - Contains an instance of an **OptimizationResults** object that contains the parameter values that were tested, and the results that are specific to the **Scorecard** that was selected by the user during the optimization (see below).

When WLP has completed an optimization, it calls the **RunCompleted** method of your **Optimizer**, and passes a parameter of type **OptimizationResultsList**, which contains a list of **OptimizationResult** objects in the **Results** property that consists of the results for all runs.

Adding Custom Tabs

Your **Optimizer** can install and populate custom tabs in the optimization user interface. To achieve this functionality, override the **Optimizer's** virtual **Initialize** method, which gets called when they user selects your optimization method from the drop down list. Within your method body, access the **Optimizer.Host** property, which implements the **IOptimizationHost** interface. This interface contains a method called **CreateTab**, in which you pass the text that should appear in the new tab, as well as a **UserControl** derived object that contains the body of the tab.

When WL completed an optimization, it calls **RunCompleted**. Here you can update the user interface in your custom tab(s). Also, when the user moves the slider values, WL will call **Optimizer's RefreshViews** method. If your user interface needs to respond to the slider changes, do so here. You access the current values of the parameters via **WealthScript.StrategyParameters[x].Value**.

Optimizer Base Class

public abstract string Description

Return a few lines of description that explains the underlying method your **Optimizer** uses.

public abstract void FirstRun(); WL calls this method at the beginning of an optimization run. You should assign the **StrategyParameter Values** of the **WealthScript** object to the values that they should assume for the first optimization run.
public abstract string FriendlyName Return a brief name for your **Optimizer**. This name appears in the optimization methods drop down list in the optimization user interface.
public IOptimizationHost Host Returns an instance of the **IOptimizationHost** interface (see below).
public abstract void Initialize WLP calls this method when the optimization method is selected in the drop down list. Here you can perform variable initialization, as well as the creation of custom user interface tabs.
public abstract bool NextRun(SystemPerformance sp, OptimizationResult or) Override this method to assign the **StrategyParameter** values for the next optimization run. The **SystemPerformance** object contains the complete performance results for the previous run, including equity curves. The **OptimizationResult** object contains the user-selected **Scorecard** results from the previous run, and the strategy parameter values that were used for that run.
public abstract double NumberOfRuns Return the number of runs that would result if the user begins an optimization.
public IPrintHost PrintHost Returns an instance of the **IPrintHost** interface, which allows your **Optimizer** to participate in printing.
public virtual void RefreshViews WLP calls this method whenever the user changes a parameter slider. If your **Optimizer** creates custom user interface tabs that need to respond to these changes, update their interfaces here.
public virtual void RunCompleted(OptimizationResultList results) WLP calls this method when an optimization run is completed. Here you can populate the results of custom user interface tabs, if applicable.
public Strategy Strategy Returns the instance of the **Strategy** that is being optimized.
public WealthScript WealthScript Returns the instance of the **WealthScript**-derived class that represents the **Strategy** being optimized. Access the parameters of the **Strategy** via the **StrategyParameters** property, which is a **List<StrategyParameter>**.

IOptimizationHost Interface

An instance of this interface is available via the **Optimizer's Host** property.

void CreateTab(**string** text, UserControl uc); Allows you to create custom tabs for your Optimizer that can depict the optimization results using any user interface you desire. The first parameter is the text of the resulting tab, and the second parameter is a UserControl derived object that contains the user interface that should appear in the tab. If you want to create custom tabs in your Optimizer, you should call this method in the **Initialize** method of your **Optimizer** derived class.
IList<**string**> MetricNames Returns a list of strings that contain the names of the performance metrics in the **Scorecard** that the user has currently selected for optimization.

OptimizationResult Class

This class represents the results for a single optimization run. Your **Optimizer** is passed an instance during the **NextRun** method. This instance contains the results of the previous optimization run. Also, the **RunCompleted** method contains a parameter that consists of a list of **OptimizationResult** objects that represent all of the optimization results.

public List<**double**> ParameterValues Contains a list of double values that contains the strategy parameter values that were used for this optimization run.
public List<**double**> Results Contains a list of double values that contains the performance metric results for the metrics in the **Scorecard** that was selected by the user for this optimization run. You can obtain the corresponding metric names by accessing the **Host.MetricNames** property of your **Optimizer**.
public string Symbol Returns the symbol that this optimization run was based on.

OptimizationResultList Class

This class contains a list of **OptimizationResult** objects accessed via the **Results** property. It is passed as a parameter in the **Optimizer.RunCompleted** method. The class also contain a number of other public properties and methods that are used by the Wealth-Lab pro client and not intended for use by custom **Optimizers**.