Creating a PosSizer in Wealth-Lab Pro®

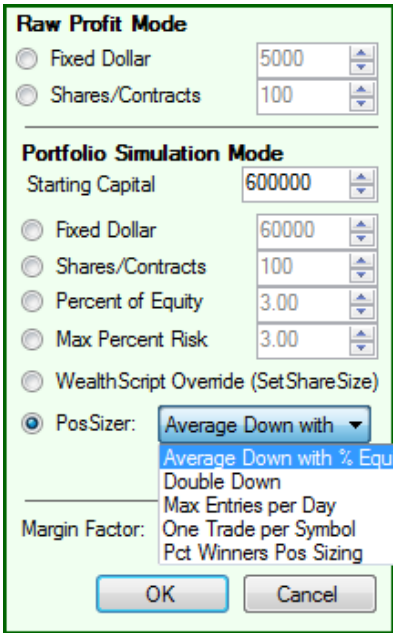
Introduction

When backtesting a Strategy in Wealth-Lab Pro® there are several position sizing methods available. For example, you can allocate a fixed number of shares for each position, or a fixed dollar amount. When allocating a fixed dollar amount, the number of shares is determined by dividing the specified dollar amount by the Position's "basis price". The "basis price" of a Position is usually the closing price of the bar on which the Position was triggered. The actual entry price is not known until the beginning of the next bar, so the entry price is usually different than the basis price.

In order to use more complex position sizing methods, **PosSizers** can be used. A **PosSizer** is a a .NET class the derives from the **PosSizer** base class. It contains a **SizePosition** method that you override to specify the exact logic to use to establish the size of each Position in a backtest. **PosSizers** can use a variety of information to make this decision, including prior trades, the equity curve, and how much cash is currently available to the simulated account.

How PosSizers are integrated in Wealth-Lab Pro

PosSizers are one of the position sizing choices in the Portfolio Simulation section of the position sizing drop down. Each **PosSizer** appears in the drop down, and optionally allows the user to configure its parameters.



How to Create a PosSizer

PosSizers can be built in an external .NET development tool such as Microsoft Visual Studio or SharpDevelop. Follow these steps to create a new **PosSizer**:

1. Create a .NET class library project.
2. Add a reference to **WealthLab.dll**.
3. Create a new class derived from **PosSizer**.
4. Implement the properties and methods, including the **SizePosition** method.
5. Compile your class library to produce a .NET assembly DLL.
6. Copy the DLL into the Wealth-Lab folder.
7. The next time you start Wealth-Lab, your PosSizer(s) will be available in the drop down.

Returning the Position Size

For each Position in a backtest, your PosSizer's **SizePosition** method will be called. This method returns a double value which specifies the number of shares/contracts that should be assigned to the Position currently being processed. Your PosSizer can access the current Position object being sized via the "currentPos" parameter.

In your position sizing logic, be very careful that the PosSizer does not access any future information. For example, the PosSizer has access to currently active Positions via the **ActivePositions** property. But the individual Position objects in this list may have an **ExitBar** and **ExitPrice** assigned, even though they are currently active at the time the current Position is being sized.

Your PosSizer can access the current **CashCurve** and **EquityCurve** of the backtest, and create indicators off of these values. However, because these DataSeries are being dynamically built during the position sizing phase of backtest, a PosSizer should use the "**Value**" methods to access indicator values instead of the "**Series**" methods. The "**Value**" methods of an indicator calculate and return the indicator value each time they are called, while the "**Series**" methods create new DataSeries that represent the complete indicator, cache these DataSeries, and return the cached values on subsequent calls. For this reason, calling "**Series**" indicator methods will return incorrect indicator results if "Series" is called in multiple calls to the PosSizer's **SizePosition** method.

Right:

```
double sma = SMA.Value(EquityCurve, bar, 20);
```

Wrong:

```
DataSeries smaSeries = SMA.Series(EquityCurve, 20);
double sma = smaSeries[bar];
```

PosSizer Base Class

public List<Position> ActivePositions Returns a list of **Position** objects that are currently "active" in the backtest. Although the Position objects in the list are active at the point in time the current Position is being sized, their ExitBar and ExitPrice properties will be assigned values if they were ultimately closed.
protected double CalcPositionSize(PosSizeMode mode, **double** posSizeValue, Bars bars,

int bar, PositionType pt, **double** costBasis, **double** riskStopLevel, **double** equity)

This is a helper method that you can call when you want a Position to use one of the basic three position sizing methods: fixed share, fixed dollar, or maximum risk. Usually you would call this method within the implementation of your PosSizer's **SizePosition** method.

public List<Position> Candidates

Returns a list of **Position** objects that represent the candidate Positions that are being sized on the current bar, at the point in time that the **SizePosition** method is being called.

public DataSeries CashCurve

Returns a DataSeries object that represents the bar by bar historical cash level of the backtest.

public List<Position> ClosedPositions

Returns a list of Position objects that have already been closed at this point in time in the backtest.

public DataSeries EquityCurve

Returns a DataSeries object that represents the bar by bar historical equity curve of the backtest.

public abstract string FriendlyName

You should override this property to return a short, descriptive name for the PosSizer. This name appears in the drop down box of the position sizing selection interface.

public virtual void Initialize()

This method is called each time the PosSizer is about to be used to size Positions in a backtest. You can override this method to perform any initialization required at this point.

public List<Position> Positions

Returns a list of Position objects that contain all of the Positions (open and closed) currently processed by the backtest.

public abstract double SizePosition(Position currentPos, Bars bars, **int** bar, **double** basisPrice, PositionType pt, **double** riskStopLevel, **double** equity, **double** cash);

Override this method and return the number of shares that the Position specified in the "currentPos" parameter should be assigned. Additional parameters include:

bars - The **Bars** object that the Position is being established on.
bar - The bar number on which the alert to enter the Position was made. PosSizers assume that Positions in a Strategy are executed at "bar + 1", and the value here is actually the Position's entry bar minus 1.
basisPrice - The Position's basis price. For market orders, this is the closing price of the bar on which the alert was generated. For limit and stop orders it is the limit or stop price.
pt - The position type (long or short).
riskStopLevel - The risk stop level that was assigned by the Strategy (if any). This applies for position sizing methods that wish to enforce a maximum loss per Position.

equity - The current equity level of the simulated account.
cash - The current cash level of the simulated account. Note that this value will decrease as multiple Positions are sized in the same bar.

Custom PosSizer Settings

Usually you want to provide some level of customization to a PosSizer, so user's can adjust certain parameters to control how Positions are sized. PosSizer's provide this capability by implementing the ICustomSettings interface. If your PosSizer class implements ICustomSettings, when it is selected from the drop down, a "Customize" button will appear in the position sizing user interface. When users click the button, a user interface that you define will appear, allowing them to set whatever parameters you decide are important for the operation of your PosSizer.

When you provide key names for custom settings in ICustomSettings, you should preface the key name with the class name of the PosSizer. This will ensure that the key value will remain unique in the Wealth-Lab settings file that is shared by all users of the ICustomSettings interface.

Persisting Custom Settings

PosSizers require a little extra support so that their settings can be persisted from instance to instance. The ICustomSettings implementation above causes parameters to be rememberd between sessions of WLP, but in order for each individual PosSizer to remember its parameters two methods need to be overriden in the base **PosSizer** class.

public virtual string GetConfigString()

You should return a single string that represents the parameter values that the **PosSizer** instance is using. If you're deriving from another **PosSizer**, such as **BasicPosSizer** (see below) be sure to call the inherited method and append its return value to your own string. **BasicPosSizer** uses a pipe character to delimit parameter values.

Example:

```
 public override string GetConfigString()
{
  return base.GetConfigString() + _maxEntries + "|";
}
```

public override void ApplyConfigString(**string** config)

Override this method to parse a config string in the format that was returned by GetConfigString above.

Example:

```
 public override void ApplyConfigString(string config)
{
  base.ApplyConfigString(config);
  string[] tokens = config.Split('|');
```

```
  _maxEntries = Int32.Parse(tokens[4]);
}
```

The BasicPosSizer Helper Class

When writing custom PosSizers, you may find that you often want to fall back on the basic position sizing methods of Fixed Dollar, Percent of Equity, or Maximum Risk, but want to allow the values of these methods the be dynamically assigned by your PosSizer. You then find yourself writing similar ICustomSettings user interface UserControls that expose radio buttons to select the three methods, and data entry fields so users can supply values for them.

The WealthLab.PosSizers assembly contains an abstract class called BasicPosSizer that factors all of this work into a class you can derive from. The assembly also provides an ICustomSettings UserControl called BasicPosSizerSettings that BasicPosSizer uses in its implementation of the ICustomSettings interface. You can use BasicPosSizerSettings as is, or derive a new UserControl from this as a base, providing additional data entry fields for parameters as needed.

BasicPosSizer provides additional methods and properties to help you implement a PosSizer that can fall back on the basic position sizing methods of Fixed Dollar, Percent of Equity, and Maximum Risk.

public double CalcBasicPositionSize(Bars bars, **int** bar, PositionType pt, **double** costBasis, **double** riskStop, **double** equity) You can call this method to perform the basic position sizing of Fixed Dollar, Percent of Equity, or Maximum Risk. Which method performed depends on the value of the PosSizeMode property of BasicPosSizer.
public void ChangeBasicSettings(PosSizerSettingsBase ui) Call this method in your implementation of ICustomSettings.ChangeSettings if you extended the BasicPosSizerSettings. It processes the changes of the PosSizer parameters controlled by the BasicPosSizerSettings interface.
public double FixedDollarSize Provides access to the Fixed Dollar position size that is defined in BasicPosSizerSettings.
public void InitializeSettings(PosSizerSettingsBase ui) Call this method after creating an instance of an ICustomSettings UserControl that derives from BasicPosSizerSettings. It initializes the UserControl with the parameter values that were read from the settings file.
public double MaxRiskSize Provides access to the Maximum Risk amount that is defined in BasicPosSizerSettings.
public double PctEquitySize Provides access to the Percent of Equity amount that is defined in BasicPosSizerSettings.

public PosSizeMode PosSizeMode Provides access to the position sizing mode that was selected in the BasicPosSizerSettings interface.
public void ReadBasicSettings(ISettingsHost host) Call this method in your implementation of ICustomSettings.ReadSetting if you extended the BasicPosSizerSettings. It reads the basic settings from the settings file.
public void WriteBasicSettings(ISettingsHost host) Call this method in your implementation of the ICustomSettings.WriteSettings if you extended the BasicPosSizerSettings. It writes the basic settings to the settings file.

Example

Below is the source code for the **MaxEntriesPerDay** PosSizer, which extends the BasicPosSizer, and implements ICustomSettings with a UserControl that descends from BasicPosSizerSettings. It is designed to employ a basic position sizing method specified by the user, but only allow a certain number of entries on any given bar.

```
using System;
using System.Collections.Generic;
using System.Text;
using Fidelity.Components;
using System.Windows.Forms;

namespace WealthLab.PosSizers
{
    public class MaxEntriesPerDay : BasicPosSizer, ICustomSettings
    {
        //Friendly name
        public override string FriendlyName
        {
            get
            {
                return "Max Entries per Day";
            }
        }

        //Maximum number of entries per day
        public override double SizePosition(Position currentPos, Bars bars,
int bar, double basisPrice, PositionType pt, double riskStopLevel, double
equity, double cash)
        {
            int takenToday = 0;
            for (int n = Positions.Count - 1; n >= 0; n--)
                if (Positions[n].EntryBar == bar + 1)
                {
                    takenToday++;
                    if (takenToday >= _maxEntries)
                        break;
                }
```

```
            if (takenToday >= _maxEntries)
                return 0;

            //calculate position size
            return CalcBasicPositionSize(bars, bar, pt, basisPrice,
riskStopLevel, equity);
        }

        //private members
        private int _maxEntries = 2;
        private MaxEntriesPerDaySettings _settings = null;

        //ICustomSettings
        #region ICustomSettings Members

        public UserControl GetSettingsUI()
        {
            if (_settings == null)
                _settings = new MaxEntriesPerDaySettings();
            _settings.MaxEntries = _maxEntries;
            InitializeSettings(_settings);
            return _settings;
        }

        public void ChangeSettings(UserControl ui)
        {
            _maxEntries = _settings.MaxEntries;
            ChangeBasicSettings(_settings);
        }

        public void ReadSettings(ISettingsHost host)
        {
            _maxEntries = host.Get("MaxEntriesPerDay.MaxEntries", 2);
            ReadBasicSettings(host);
        }

        public void WriteSettings(ISettingsHost host)
        {
            host.Set("MaxEntriesPerDay.MaxEntries", _maxEntries);
            WriteBasicSettings(host);
        }

        #endregion
    }
}
```