Using the CQG Charting API

Document Version 2015-01

April 6, 2015

Table of Contents

About this Document

Historically, CQG has offered CQG IC users the ability to create their own market data analysis tools using custom indicators. For example, CQG IC's Formula Builder can be used to define study and condition formulas for use with CQG IC applications, such as charts, quote boards, and trading tools.

With the introduction of CQG's easy-to-use charting API, you now have the option of building custom analytic tools as their own libraries (DLL).

This document provides the instructions and necessary references to create those analytics.

Programming experience is needed, especially experience developing Windows applications.

Publication History

Version	**Date**	**Comments**
2015-01	April 6, 2015	Initial publication.

Customer Support

Contact apisupt@cqg.com for assistance with the CQG Charting API.

For questions about a specific issue, attaching supporting documentation, such as a project, helps expedite resolution.

CQG Charting API

CQG's Charting API is used to build custom analytic tools, that incorporate your proprietary logic, as their own libraries (DLL).

These tools:

- can be created using several technologies
- are able to implement complex logic
- use external data with real-time updates
- are fully integrated with CQG IC for use wherever possible
- are displayed on a graphic interface enabling advanced visual representations on charts.

CQG uses the logic to call methods of the library to calculate market data and then displays the results using native interfaces. You specify inputs.

The API was designed for maximum usability, so the user library is implemented using just two methods: Initialize and Calculate.

Initialize is used to provide the operator with parameter values (if there are any). It is called when the library is (re)initialized. Possible scenarios are:

- applying a user study to chart
- changing parameters
- switching CQG pages

Values of parameters are passed to the library in this method. Even if you use no parameters, the method must be defined, and it must return a positive integer, as this return value informs CQG what depth of historic values is required.

Calculate is the method used to perform actual calculations. This simple structure supports algorithms of any complexity, provided the input data can include numerous inputs and custom depth of historical values.

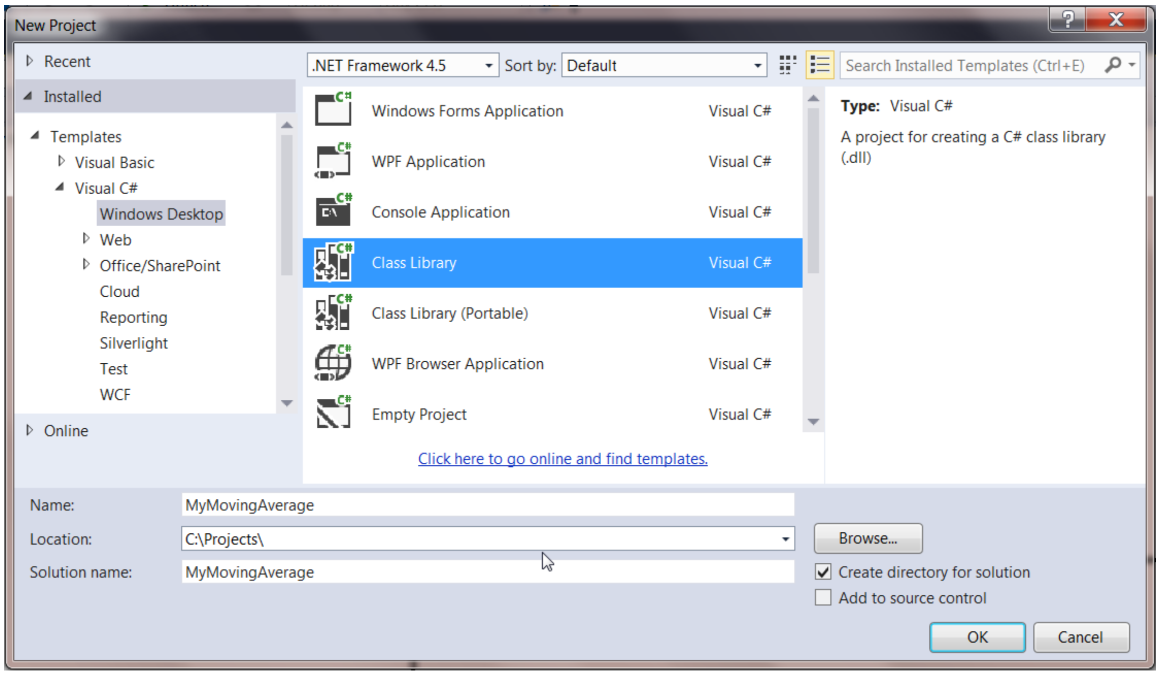
In addition to the simple framework, the library can instruct CQG IC to display custom graphic elements (primitives). This option is available if you want to go beyond the standard palette of CQG IC data representation options.

You can extend the Calculate method to define, and provide to CQG IC, a list of graphic primitives – symbols, text, lines, shapes – and CQG IC displays them at positions specified.

One significant benefit is that the library is also capable of initiating data updates in CQG IC. In this way, you can display third-party data updates in CQG IC. Initialize method provides and interface to the IBarsRebuilder interface, which can be used later by the DLL to request updates of a whole or partial data set. The integration of the library is straightforward: the compiled DLL, saved to a directory, is accessed by name in CQG expressions.

Creating a Simple Study using VS2013 and C#: Moving Average

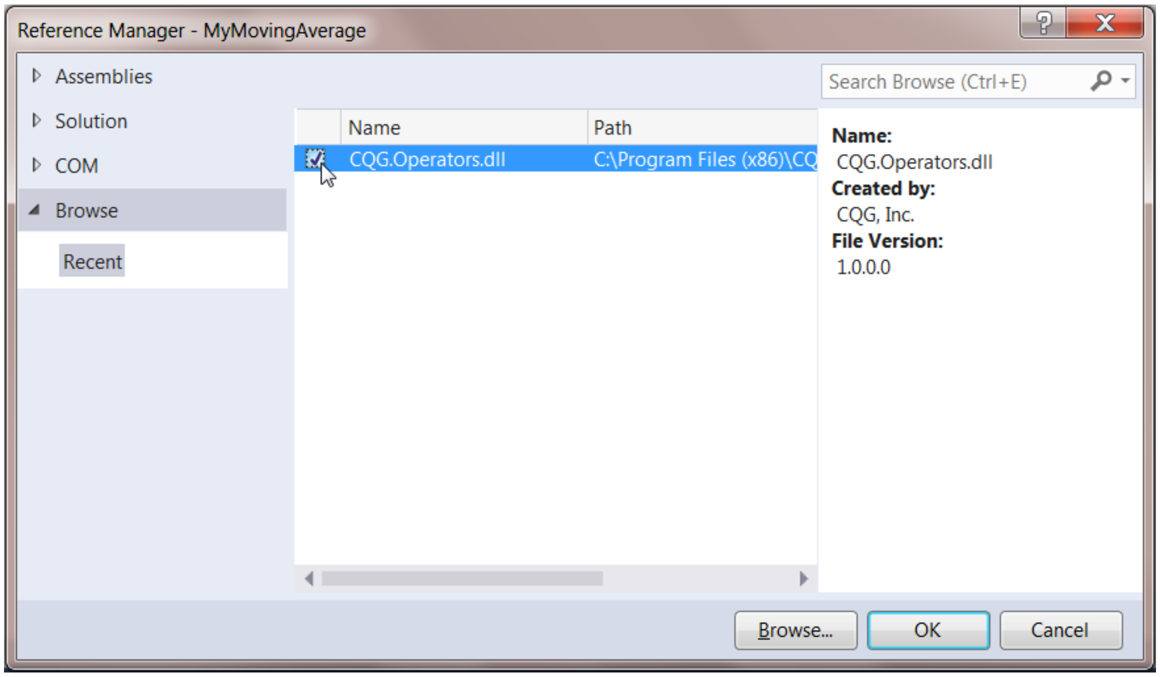
1. **Create a new project.**
   1.1. Open Visual Studio, and create a new project.
   1.2. Select Visual C# Class Library as the project type.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Desktop
Web
Office/SharePoint
Cloud
Reporting
Silverlight
Test
WCF
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Class Library (Portable) Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#
A project for creating a C# class library (.dll)

Name: MyMovingAverage
Location: C:\Projects\
Solution name: MyMovingAverage
☑ Create directory for solution
☐ Add to source control

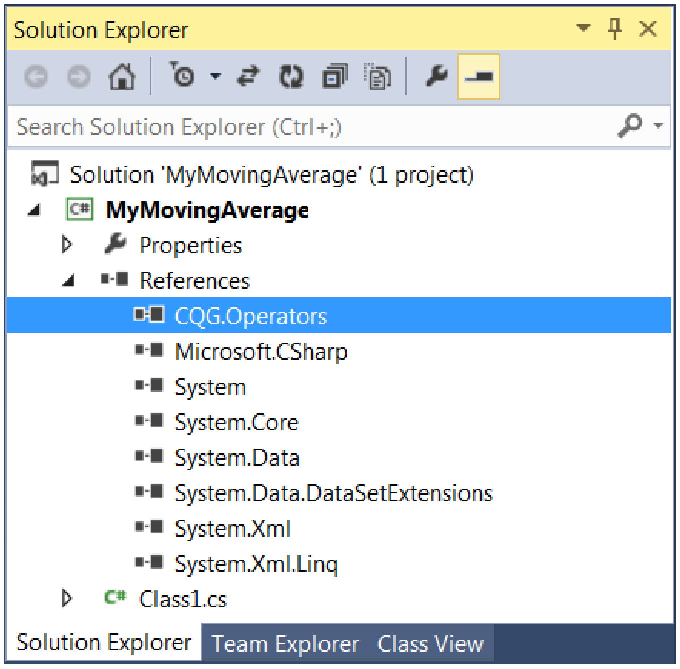
2. **Add a reference to the CQG Components library**.
   2.1. On the **Project** menu, select **Add reference** to open the Reference Manager window.


Reference Manager - MyMovingAverage

	Name	Path
✓	CQG.Operators.dll	C:\Program Files (x86)\CQ

Name:
CQG.Operators.dll
Created by:
CQG, Inc.
File Version:
1.0.0.0

   2.2. Click the **Browse** button to locate CQG.Operators.dll. By default, it is located here: C:\Program Files (x86)\CQG\CQGNet\Bin\Components.
   2.3. Select it as the reference.
   2.4. After closing Reference Manager, go to Solution Explorer and confirm that CQG.Operators appears in the References section.



Solution 'MyMovingAverage' (1 project)
MyMovingAverage
Properties
References
CQG.Operators
Microsoft.CSharp
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Xml
System.Xml.Linq
Class1.cs

3. **Write code**.

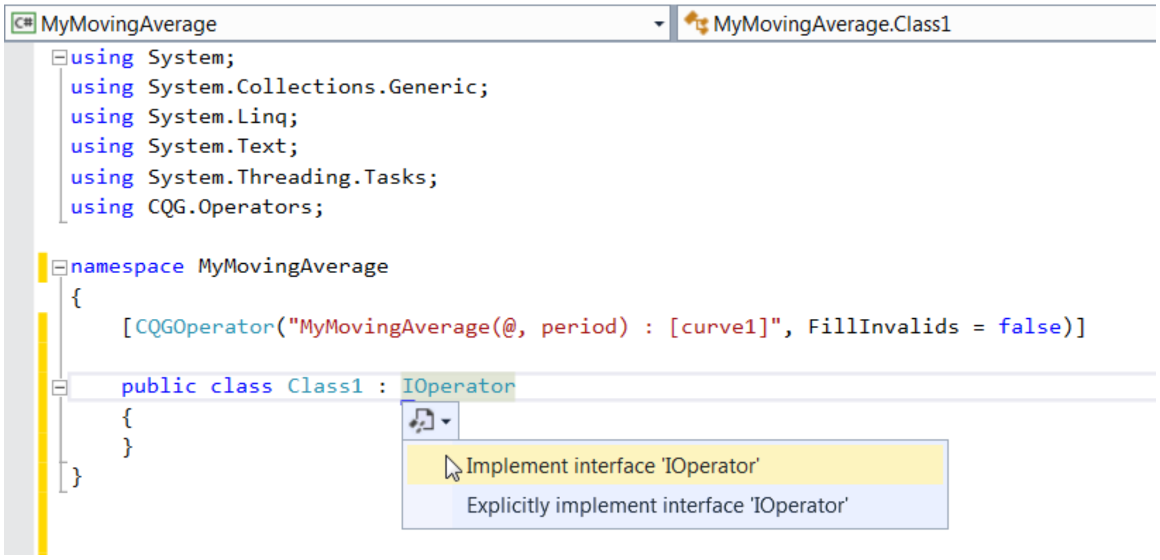
   3.1. Reference CQG.Operators namespace (at the beginning of the file, add "using CQG.Operators;").

   3.2. Add your operator expression signature, name, and parameters.

   We'll call this method MyMovingAverage. It will have one input, the value to get average for, and one parameter, the period of averaging. The system will not fill gaps in data.

   3.3. Add `[CQGOperator("MyMovingAverage(@, period) : [curve1]", FillInvalids = false)]` (see [CQGOperatorAttribute Class]).

   3.4. Derive your class from IOperators and add implementation of its methods (see [IOperator interface]).


MyMovingAverage
MyMovingAverage.Class1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CQG.Operators;

namespace MyMovingAverage
{
    [CQGOperator("MyMovingAverage(@, period) : [curve1]", FillInvalids = false)]

    public class Class1 : IOperator
    {
    }
}
```

Implement interface 'IOperator'
Explicitly implement interface 'IOperator'

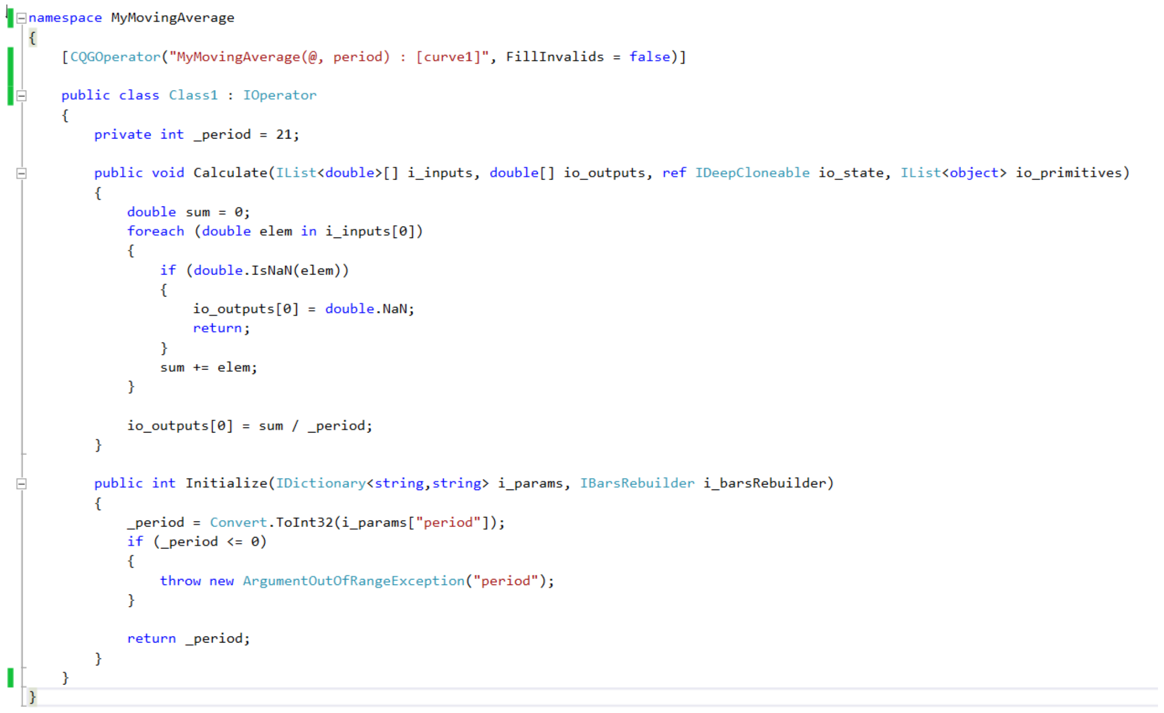
After the definition of Initialize and Calculate methods are added, we implement them.

The Initialize method is used to receive the parameter value, the period of averaging.

This value is also returned by the method, so that CQG provides sufficient depth of historic data in the Calculate method.

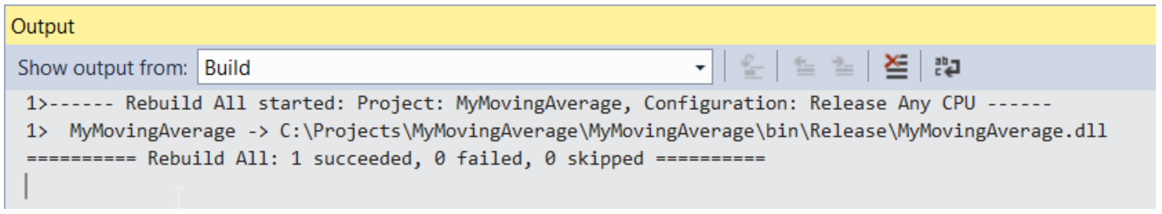
The Calculate method calculates the average of the inputs, which defines the required size (as per the value returned in Initialize).

The entire implementation is brief.



```
namespace MyMovingAverage
{
    [CQGOperator("MyMovingAverage(@, period) : [curve1]", FillInvalids = false)]

    public class Class1 : IOperator
    {
        private int _period = 21;

        public void Calculate(IList<double>[] i_inputs, double[] io_outputs, ref IDeepCloneable io_state, IList<object> io_primitives)
        {
            double sum = 0;
            foreach (double elem in i_inputs[0])
            {
                if (double.IsNaN(elem))
                {
                    io_outputs[0] = double.NaN;
                    return;
                }
                sum += elem;
            }

            io_outputs[0] = sum / _period;
        }

        public int Initialize(IDictionary<string,string> i_params, IBarsRebuilder i_barsRebuilder)
        {
            _period = Convert.ToInt32(i_params["period"]);
            if (_period <= 0)
            {
                throw new ArgumentOutOfRangeException("period");
            }

            return _period;
        }
    }
}
```

4. **Build the solution**.



```
1>------ Rebuild All started: Project: MyMovingAverage, Configuration: Release Any CPU ------
1>  MyMovingAverage -> C:\Projects\MyMovingAverage\MyMovingAverage\bin\Release\MyMovingAverage.dll
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

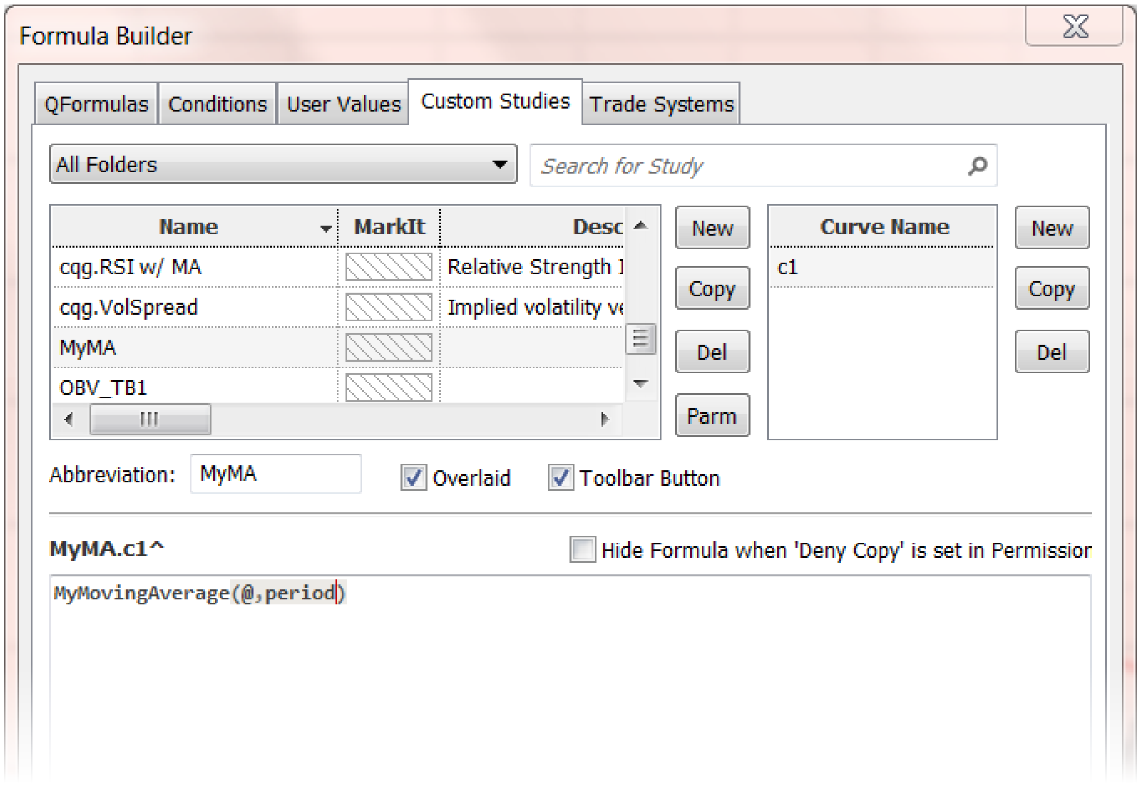
Note the path where the resulting dll was placed, and copy it to the User Operators folder.

By default, it's here: C:\Users\Public\Documents\CQGNet\Private\).

You may need to create the folder.

In the end, MyMovingAverage.dll should be found here:
C:\Users\Public\Documents\CQGNet\Private\User Operators

5. **Create a custom study in CQG IC**.
   5.1. Start or restart CQG.
   5.2. Open Formula Builder, and create a custom study, calling your library by its signature:



Name	MarkIt	Desc
cqg.RSI w/ MA	▨	Relative Strength I
cqg.VolSpread	▨	Implied volatility ve
MyMA	▨	
OBV_TB1	▨	

Curve Name
c1

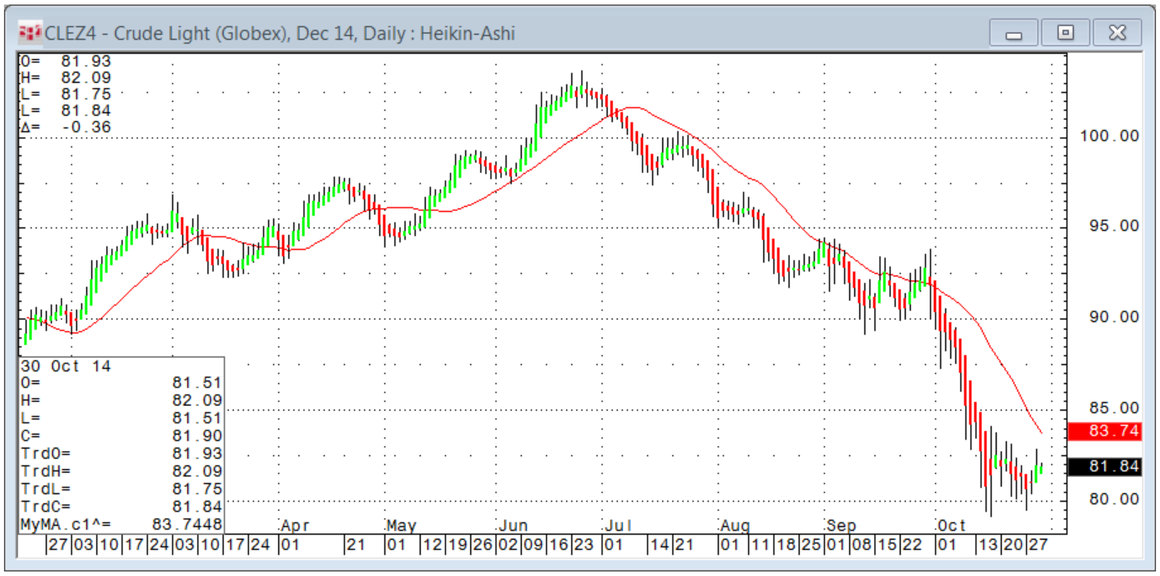
Abbreviation: MyMA [✓] Overlaid [✓] Toolbar Button

MyMA.c1^ [ ] Hide Formula when 'Deny Copy' is set in Permissior

```
MyMovingAverage(@,period)
```

In this example, we defined period as a parameter of the MyMA custom study.

The study can now be applied wherever studies can be used.



```
O=   81.93
H=   82.09
L=   81.75
L=   81.84
Δ=   -0.36
```

```
30 Oct 14
O=              81.51
H=              82.09
L=              81.51
C=              81.90
TrdO=           81.93
TrdH=           82.09
TrdL=           81.75
TrdC=           81.84
MyMA.c1^=     83.7448
```

Creating a Study with State

Sometimes it may be beneficial to use a different model of the operator, a model with state. In this case, the operator uses data it collects on previous bar(s) to compute the value for the current bar.

State could be used, for example, to collect arrays of historic data (the operator is guaranteed to be called sequentially for bars from earliest to the most recent) or to keep results of temporary calculations.

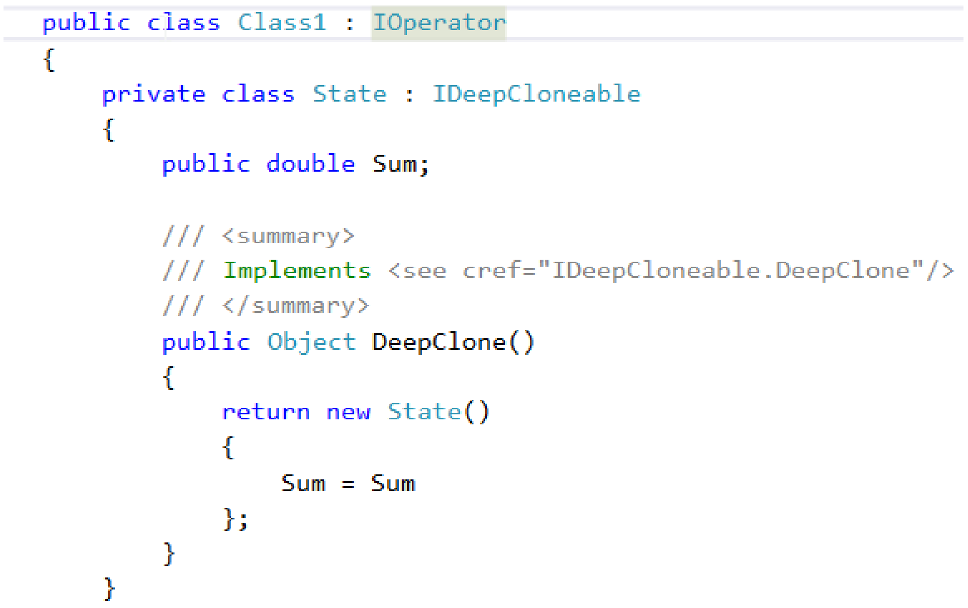
Generally, state may be a more efficient solution when the operator calculates large arrays of data.

You can define state. It must implement IDeepCloneable. Method Calculate receives state from the previous bar as an input/output parameter and must update it before exit.

We’ll use Moving Average state to illustrate the idea. The state stores the sum of [last period – 1] inputs, so to calculate the current value, we’ll add a new element and divide by the period.

Modifying Moving Average operator code (from simple study example)

1. Add class State to your operator definition.



```
public class Class1 : IOperator
{
    private class State : IDeepCloneable
    {
        public double Sum;

        /// <summary>
        /// Implements <see cref="IDeepCloneable.DeepClone"/>
        /// </summary>
        public Object DeepClone()
        {
            return new State()
            {
                Sum = Sum
            };
        }
    }
```

2. Change the Calculate method as follows:

```
public void Calculate(IList<double>[] i_inputs, double[] io_outputs, ref IDeepCloneable io_state, IList<object> io_primitives)
{
    State state = (State)io_state;
    IList<double> inputs = i_inputs[0];

    if (state == null)
    {
        state = new State();

        for (int i = 0; i < inputs.Count; ++i)
        {
            state.Sum += inputs[i];
        }

        io_state = state;
    }
    else
    {
        state.Sum += inputs[inputs.Count - 1];
    }

    io_outputs[0] = state.Sum / _period;
    state.Sum -= inputs[0];
}
```

3. Rebuild the solution.
4. Close CQG.
5. Copy the DLL to C:\Users\Public\Documents\CQGNet\Private\User operators.
6. Open CQG IC. The study is now functional.

Note that the new solution still uses a period-sized array of input data. We could use on the most recent value as the input if we kept the array of [period] recent input values within the state.

Interface Reference

CQG.Operators.IOperator interface

Main interface for an external CQG operator. Has two member functions: Initialize and Calculate.

Initialize

```
int CQG.Operators.IOperator.Initialize
      (     IDictionary< string, string >       i_params,
            IBarsRebuilder    i_barsRebuilder
      )
```

Method initializes operators parameters. It must return number of input elements required for calculation (period) or -1 if an error occurs.

i_params =	Contains pairs 'parameter name' - 'parameter value'. Parameter names are specified in operator's expression. For example, operator 'Op(@, param1, param2)' will have 2 parameters 'param1' and 'param2'. Additionally the dictionary may contain some general parameters that could be useful for operators Available general parameters: “-GlobalExtraBarCount” Integer, specifies extra bar count for study calculation. The value is set globally and could be changed through System Preferences dialog (Page Limits – Extra Bars for Study Calculation) “-RootSymbol” String, specifies the underlying contract symbol “-SerialNumber” Integer, specifies the system serial number “-PriceToDollarMultiplier” Double, specifies price-to-dollar multiplier for current contract “-MinimumValueChange” Double, specifies minimum price change for current contract Notice that general parameter names start with a hyphen to avoid naming conflicts with user-defined parameters.
i_barsRebuilder =	Interface that could be used to initiate bars rebuilds.

Calculate

```
void CQG.Operators.IOperator.Calculate
      (     IList< double >[]       i_inputs,
            double[]    io_outputs,
            ref IDeepCloneable      io_state,
            IList< object >   io_primitives
      )
```

Calculates operator output values for the specified inputs. Can use state from previous value and provide state for next value. Can return drawing primitives to be displayed with the study.

i_inputs =	Collection of input data. Array items correspond to operator inputs. For example, for operator 'Op(@, param1, param2)' we’ll have one element in the array and for operator 'Op(@, @, param)' we’ll have two elements in the array. Each array item is a list of N elements, where N is return value of IOperator.Initialize.
io_outputs =	Array of N elements, where N is a number of outputs specified in operator expression. For example, operators 'Op(@)', 'Op(@) : [1]', 'Op(@) : [curve1]' we’ll have one element in the array and operators 'Op(@) : [2]', 'Op(@) : [curve1, curve2] we’ll have two elements in the array. The array should be filled with calculated values for each output correspondingly.
io_state =	State from the previous calculations. Initially the value is null; if the operator assigns something to the parameter then it receives the value on the Calculate call for the next bar. Simple operators that do not use state should ignore this value.
io_primitives =	List can be filled with primitive objects (allowed types are BorderPrimitive, TextPrimitive, LinePrimitive and SymbolPrimitive) associated with the calculation. Operators that produce only curve outputs should ignore this parameter.

CQG.Operators.IDeepCloneable interface

In contrast to ICloneable interface, which doesn't specify whether the cloning should be deep copy, shallow copy, or something between, this interface requires deep copy functionality. After a copy is created, any changes to the original object shouldn't affect the copy. Interface has one member function: DeepClone.

Deep Clone

```
object CQG.Operators.IDeepCloneable.DeepClone ()
```

Creates and returns a new object that is a deep copy of the current instance

CQG.Operators.IBarsRebuilder Interface

Interface initiates rebuild values for the defined range of inputs (bars). Interface has two member functions: RebuildAll and RebuildRange.

RebuildAll

```
void CQG.Operators.IBarsRebuilder.RebuildAll ()
Initiates update of all bars.
```

RebuildRange

```
void CQG.Operators.IBarsRebuilder.RebuildRange
      (     DateTime    startTime,
            DateTime    endTime
      )
```

Initiates update of bars in the range specified.

i_start =	Start time to rebuild.
i_end =	End time to rebuild (inclusive).

CQG.Operators.CQGOperatorAttribute class

Defines the metadata attribute to use on CQG Operator classes (the signature of the operator when called from within CQG IC). The class with this attribute must implement IOperator interface.

CQG.Operators.CQGOperatorAttribute.FillInvalids

bool CQG.Operators.CQGOperatorAttribute.FillInvalids (getset)

Indicates whether input in IOperator.Calculate method always has valid values.

If it's set to true then all inputs contain only valid values, guaranteed. The method wouldn't be called until we accumulate enough valid values, and later all invalid input values are automatically replaced with previous valid input value. However, if set to false, any input value could be Double.NaN and the operator should correctly process such inputs.

By default, this property is true.

CQG.Operators.CQGOperatorAttribute.Expression property

string CQG.Operators.CQGOperatorAttribute.Expression (get)

The format of the expression is:
cqg_operator := name name(params) -or- name(params) : [outputs_count] -or- name(params) : [outputs]

where:

name = operator name that could only contain alpha-numeric digits and underscore symbol.

params = comma separated list of inputs and parameter names. Inputs are specified by '@' symbol.

outputs_count = count of operator outputs.

outputs = comma separated list of output names. For output name it's allow to use 'curveN' where N is a number in range [1,20] and some special names like 'open', 'high', 'low', 'close'

Examples:

SampleOp_1(@)

Operator with name 'SampleOp_1' having one input, one output and no parameters

SampleOp_2(@, param1) : [2]

Operator with name 'SampleOp_2' having one input, two outputs and one parameter named 'param1'

SampleOp_3(@, param1, @, param2) : [curve1]

Operator with name 'SampleOp_3' having two inputs, one output and two parameters named 'param1' and 'param2'

SampleOp_4(@, @, @) : [curve1, curve2, curve3]

Operator with name 'SampleOp_4' having three inputs, three outputs and no parameters

Primitives Package

Package contains collection of structures that instruct CQG IC to display custom graphic information on charts.

BorderPrimitive class

Used to specify custom rectangular or elliptic graphic primitive to draw on chart at a specified position. Primitive can have border, filling, and tooltip.

Member	attr/func	Description
BorderPrimitiveType BorderType	public attribute	Specifies shape of the primitive (see BorderPrimitiveType).
ChartPosition Point1	public attribute	Specifies top-left point of the primitive.
ChartPosition Point2	public attribute	Specifies bottom-right point of the primitive.
PrimitivePen Pen	public attribute	Specifies pen to draw the border of the primitive. If null, border is not visible.
Color BackgroundColor = Color.Empty	public attribute	Specifies color to fill the primitive. Default = Color.Empty (no fill)
string Tooltip	public attribute	Tooltip text, shown on mouse hovering the primitive. If null or empty string, tooltip is not displayed.

ChartPosition class

Used to specify position on the chart using time and value as coordinates.

Member	**attr/func**	**Description**
ChartPosition (ChartTime time=null, double value=double.NaN)	public function	Initializes a new instance of ChartPosition object. Parameters: time = position of primitive on time scale (x-coordinate of the chart) value = position of primitive on value scale (y-coordinate of the chart).
ChartTime Time	public attribute	Specifies position of primitive on time scale (x-coordinate of the chart)
double Value	public attribute	Specifies position of primitive on value scale (y-coordinate of the chart)

ChartTime class

Used to specify position of primitive on time scale (x-coordinate of the chart).

Member	attr/func	Description
ChartTime ()	public function	Initializes a new instance of ChartTime object corresponding to primitive position on the bar currently being calculated.
ChartTime (int barOffset)	public function	Initializes a new instance of ChartTime object corresponding to primitive position on bar with some offset related to the currently calculated bar. Parameter: barOffset = offset to the left (negative) or to the right (positive) from the calculated bar, in bar count.
ChartTime (DateTime barTime, int barOffset=0, int barIndex=0)	public function	Initializes a new instance of ChartTime object corresponding to bar with specified date/time and offset. Parameters: barTime = time of the bar to position primitive to barOffset = if not zero, primitive position is shifted on the specified bar count barIndex = if there are two or more bars with the same time, this zero-based index selects a particular bar
DateTime BarTime	public attribute	Date/Time of the bar using minutes or milliseconds resolution.
int BarOffset	public attribute	Offset to the left (negative) or to the right (positive) in bar count.
int BarIndex	public attribute	Bar index (zero-based, it may be greater than zero if there are several bars with the same time).

LinePrimitive class

Used to specify custom line graphic primitive to draw on chart at a specified position. Primitive can have tooltip.

Member	attr/func	Description
ChartPosition Point1	public attribute	Specifies position of the first point of the line.
ChartPosition Point2	public attribute	Specifies position of the second point of the line.
bool EndsInPoint1	public attribute	Indicates whether the line ends at first specified point.
bool EndsInPoint2	public attribute	Indicates whether the line ends at second specified point
PrimitivePen Pen	public attribute	Specifies pen to be used for drawing the line (see PrimitivePen).
string Tooltip	public attribute	Tooltip text, shown on mouse hovering the primitive. If null or empty string, tooltip is not displayed.

PrimitivePen class

Used to specify pen to use for painting graphic primitives.

Member	attr/func	Description
PrimitivePen (Color color, int style=0, double width=1.0)	public function	Initializes a new instance of PrimitivePen object. Parameters: color = pen color style = pen style as defined in GDI API, default = 0 (PS_SOLID) width = relative pen width
Color Color	public attribute	Defines pen color.
int Style	public attribute	Defines pen style as defined in GDI API Default = 0 (PS_SOLID)
double Width	public attribute	Defines relative pen width. Visual representation is relative to chart bar zoom, so the value corresponds to study lines width.

SymbolPrimitive class

Used to draw one of the predefined set of CQG IC symbols on chart.

Member	attr/func	Description
SymbolPrimitive (SymbolType symbolType, ChartPosition position=null)	public function	Initializes a new instance of SymbolPrimitive class. Parameters: symbolType = symbol to draw (see SymbolType) position = position of the symbol on the chart
SymbolType SymbolType	public attribute	Specifies the symbol to draw (see SymbolType).
int SizeFactor = 1	public attribute	Specifies relative size of the symbol. Range = -1000 to 1000
ChartPosition Position	public attribute	Specifies position of the symbol on the chart.
Color Color = Color.Black	public attribute	Defines symbol color.
string Tooltip	public attribute	Tooltip text, shown on mouse hovering the primitive. If null or empty string, tooltip is not displayed.

TextPrimitive class

Used to draw text label of CQG IC symbol on chart.

Member	attr/func	Description
TextPrimitive (string text, ChartPosition point=null)	public function	Initializes a new instance of TextPrimitive class. Parameters: text = text to display point = position of text on chart
string Text	public attribute	Specifies the text.
Color TextColor	public attribute	Specifies the text color.
Color BorderColor = Color.Empty	public attribute	Specifies text border color (if any).
int BorderWidth = 1	public attribute	Specifies border line width in pixels.
double Padding = 0.5	public attribute	Specifies padding between text and border in symbol size units.
Color BackgroundColor = Color.Empty	public attribute	Specifies background color.
ChartPosition Point	public attribute	Point at which text should be placed. Properties HorizontalAlignment and VerticalAlignment specify how the text should be aligned relative to this point.
HorizontalAlignment HorizontalAlignment = HorizontalAlignment.Center	public attribute	Specifies text rectangle horizontal alignment relative to Point.
VerticalAlignment VerticalAlignment = VerticalAlignment.Center	public attribute	Specifies text rectangle vertical alignment relative to Point.
double SizeFactorMultiplier = 1.0	public attribute	Specifies a multiplier for standard chart font size.

SymbolType enum

Specifies types of symbol predefined in CQG IC.

Symbol	Description
Dash	Dash symbol (several neighbor symbols on same price look like a dash line)
Cross	Diagonal cross symbol
Plus	Plus symbol
Dots	Two dots symbol
UpTriangle	Triangle symbol with one vertex on top and two on bottom
DownTriangle	Triangle symbol with two vertices on top and one on bottom
UpBigArrow	Big arrow pointing up symbol
DownBigArrow	Big arrow pointing down symbol
UpSquare	Square symbol aligned vertically by bottom side
DownSquare	Square symbol aligned vertically by top side
UpDiamond	Diamond symbol aligned by bottom vertex
DownDiamond	Diamond symbol aligned by top vertex
LeftArrow	Arrow pointing left, horizontally aligned right
RightArrow	Arrow pointing right, horizontally aligned left
UpRightArrow	Arrow pointing up-right, horizontally aligned right, vertically aligned top
DownRightArrow	Arrow pointing down-right, horizontally aligned left, vertically aligned bottom
UpArrow	Arrow pointing up, vertically aligned top
DownArrow	Arrow pointing down, vertically aligned bottom
BigSquare	Big square symbol
SmallSquare	Small square symbol
BigDiamond	Big diamond symbol
SmallDiamond	Small diamond symbol
BigCircle	Big circle symbol

Symbol	**Description**
SmallCircle	Small circle symbol
LeftArrowCentered	Arrow pointing left, horizontally and vertically aligned center
RightArrowCentered	Arrow pointing right, horizontally and vertically aligned center
UpRightArrowCentered	Arrow pointing up-right, horizontally and vertically aligned center
DownRightArrowCentered	Arrow pointing down-right, horizontally and vertically aligned center
UpArrowCentered	Arrow pointing up, horizontally and vertically aligned center
DownArrowCentered	Arrow pointing down, horizontally and vertically aligned center
Line	Line symbol (several neighbor symbols on the same price look like a line)
LetterR	Letter 'R' symbol

HorizontalAlignment enum

Specifies horizontal text alignment.

Left	Left aligned
Right	Right aligned
Center	Center aligned

VerticalAlignment enum

Specifies vertical text alignment.

Top	Top aligned
Bottom	Bottom aligned
Center	Center aligned

BorderPrimitiveType enum

Types of border for BorderPrimitive.

Rectangle	Rectangular shape
Ellipse	Elliptical shape