Calling Upon Indicators Efficiently in NinjaScript

RWT-004
2009-08-18

In all of the coding tutorials and examples I've seen, Ninjatrader indicators are used in-line. So, if you want to know when the `SMA` is going up, you'd check:

```
SMA(Close, 5)[0] > SMA(Close, 5)[1]
```

I find this practice repulsive for two reasons: redundancy and inefficiency. This paper chronicles the path I took to finding a better way. If you just want to know what the better way is, skip to the 'Solution' section at the end.

Redundancy

The `SMA` example above doesn't look terribly egregious, but I've seen examples out there where the indicator took five or ten arguments! As a result, those arguments are repeated on every use of the indicator. I wouldn't write it that way just out of laziness! More importantly, what if this `SMA` is used twelve times in my code (typical for certain kinds of indicators), and I later decide I want an `EMA`, or a different length? Now, I have to make sure I change *every single instance* in my code in *exactly the same way*. Even with support from the text editor, it is an error-prone position to be in.

For this reason, even the first code I wrote for Ninjatrader stored off the indicators it used, if they appeared more than once in the algorithm. This way, there was no redundant specification of the arguments. So, I'd write:

```
SMA mysma = SMA(Close, 5);
bool goingUp = mysma[0] > mysma[1];
```

I carried on happily in this fashion for much of our port of EOTPro. In the back of my mind, I wondered how every indicator magically got a method I could call to create the associated object. I wasn't sure I was actually using C#, thinking that Ninjascript might just be a closely-related DSL. Toward the end of the porting effort, I was forced to look into it.

Inefficiency

A big problem hit with our `EOTBillsArrows` indicator. When I applied the initial version to a chart, my computer ground to a halt, soaked up all the memory, and

became unresponsive. Eventually, the platform would crash. Others in my team also experienced the problem on much bigger machines. What was going on?

Not having a proper debugger, I did what any programmer would do... I started commenting stuff out to see if performance got any better. I quickly located the troubled line:

```
// This will bring Ninja to its knees!
EOTPattyB patty = EOTPattyB(Input, 3, 13, 6, 1, 1, 18,
                            2.5, 3, 3, 2, 2, 8, 11, 5,
                            12, 26, 9, 0);
```

Since the `EOTPattyB` indicator didn't stress the computer when applied to the chart, I knew immediately that I'd have to figure out what magic was being used to create the indicator object in the offending line.

In every indicator you create in Ninjatrader, a region is added to your code automatically. It bears the ominous label: "NinjaScript generated code. Neither change nor remove." Luckily, reading it is not prohibited! Inside, I found a lot of the details about how the platform creates the illusion of a DSL.

The generated code injects a method into the Indicator base class whose name matches the indicator name, and whose arguments match your indicator's inputs. This method will create a new instance of the indicator and tie it into the calculation engine. To keep from creating a new instance of the indicator every time you call for it, the method is memoized.

The problem is in the caching scheme that Ninjatrader uses for momoization. First off, it's a simple linear search! Let's say you use `SMA(Close, 5)` in your code, and further let's say that you have 50 simple moving averages in play across all of your active indicators and charts. This means that every time you use an `SMA`, it checks up to 50 instances, one at a time, in this fashion:

```
if(cacheSMA != null)
  for(int idx = 0; idx < cacheSMA.Length; idx++)
     if(cacheSMA[idx].Period == period &&
        cacheSMA[idx].EqualsInput(input))
          return cacheSMA[idx];
```

For indicators with *n* active instances and *k* inputs, this is an `O(nk)` algorithm. Granted, for many indicators, *n* will be just a handful. Nevertheless, 3 indicators at 20 inputs is an expected 3/2 * 20 = 30 comparisons per use of the indicator, just to give you the illusion that you can call up indicators on the fly.

As sad as that is, I knew that couldn't be the whole issue. In the `EOTPattyB` problem at hand, there was only one use of the indicator, and even a slow lookup wouldn't explain the memory leak. Further testing showed that Ninjatrader's generated lookup code could never match our inputs against its cache, and thus created a new instance of the indicator on every bar! A chart with 1000 bars would have 1000 `EOTPattyB`'s on it, and good luck if you set *CalculateOnBarClose* to false!

I never bothered to try to figure out how their caching code choked on this particular indicator. It didn't matter, since I can't change their generated code, and I had a better solution anyway.

Solution

Because of the issues above, I always call upon indicators in Ninjascript in the following manner: I define a variable in my class to hold the indicator instance. I then use the `OnStartUp()` method[1] to save off an instance of the needed indicators for use in `OnBarUpdate()`. Here's an outline of the approach:

```
class MyIndicator : Indicator {
   private SMA sma1;
   private SMA sma2;
   ...

   protected void override OnStartup() {
      sma1 = SMA(Close,5);
      sma2 = SMA(Close, 13);
      ...
   }

   void OnBarUpdate() {
      if(sma1[0] > sma2[0]) { ... }
      ...
   }
}
```

First, note that we still use Ninjatrader's magic generated code, so that we can be relatively sure that our indicator will still work through version changes. However, we only use it once, so that our code works even if their caching code is broken. This workaround also factors out the redundancy by putting the indicator inputs in one spot.

Richard Todd
http://www.movethemarkets.com

End Notes

1. `OnStartUp()` is a Ninjatrader 7 feature. In Ninjatrader 6.5, you can do the `OnStartUp()` work at the top of `OnBarUpdate()`, checking the value of a boolean to make sure that you only do it once. t's uglier, but it works the same.

Comments