WriteAsync .NET

Testing, coding, in that order.

Using PLA.dll to collect perf counters

Who doesn't love diagnostics?! I am a big proponent of efficient and well thought out tracing and performance counters. They are invaluable for debugging, performance testing, health monitoring, and many other tasks worthy of future blog posts.

Seasoned Windows professionals are generally familiar with the Windows Resource and Performance Monitor tool. Slightly more advanced users are typically aware of the logman.exe tool which allows access to data collectors and performance logs via the command line. But in my experience, only a select few know about PLA.dll, the programmatic interface to performance logs and alerts on Windows.

As it turns out, due to strong COM interop support in .NET and the simplicity of adding COM reference assemblies in Visual Studio, it is a breeze to take advantage of PLA in your C# apps. Just go to your project, open the "Add Reference…" dialog, select "Browse…", locate "pla.dll" (ships with Windows, typically in the `%systemroot%\system32` folder), add it, and you're ready to go.

Since PLA.dll is a COM library and geared mostly towards C++ developers, it is tricky to get the hang of using it in managed app. This is why I like to wrap it in a simpler, .NET-friendly façade when I'm exposing it to larger applications. (Pro tip: using the protocol documentation for MS-PLA can fill in some details left out by the MSDN documentation.)

Here is a sample wrapper that shows one way to expose performance counter collection via data collector sets. All of this code is available on the PlaSample project in GitHub.

We start with a type representing a counter name:

```
public class CounterName
{
    public CounterName()
    {
    }
    public string Machine { get; set; }

    public string Category { get; set; }

    public string Counter { get; set; }

    public string Instance { get; set; }

    public override string ToString()
    {
        // . . .
    }
}
```

Now we abstract the data collector set into a `CounterCollectorInfo`:

```
public class CounterCollectorInfo
{
    public CounterCollectorInfo(string name)
    {
        this.Name = name;
        this.CounterNames = new List<CounterName>();
    }

    public string Name { get; private set; }

    public string OutputPath { get; set; }

    public TimeSpan? SampleInterval { get; set; }

    public LogFileFormat? LogFileFormat { get; set; }

    public IList<CounterName> CounterNames { get; private
}
```

The `LogFileFormat` enum is mirror of the underlying PLA enum describing, predictably, the format of a perf counter log file:

```
public enum LogFileFormat
{
    CommaSeparated = 0,
    TabSeparated = 1,
    Sql = 2,
    Binary = 3,
}
```

Now for the nitty-gritty of interacting with PLA — code to create the data collector set for logging perf counters:

```
public ICollectorSet Create()
{
    // Data collector set is the core abstraction for coll
    DataCollectorSet dcs = new DataCollectorSet();

    // Set base folder to place output files.
    dcs.RootPath = this.OutputPath;

    // Create a data collector for perf counters.
    IPerformanceCounterDataCollector dc = (IPerformanceCou
PerformanceCounter);
    dc.name = this.Name + "_DC";
    dcs.DataCollectors.Add(dc);

    // Set output file name to use a pattern, as described
    // http://msdn.microsoft.com/en-us/library/windows/des
    dc.FileName = this.Name;
    dc.FileNameFormat = AutoPathFormat.plaPattern;
    dc.FileNameFormatPattern = @"\-yyyyMMdd\-HHmmss";

    // Set sample interval, if present.
    if (this.SampleInterval.HasValue)
    {
        dc.SampleInterval = (uint)this.SampleInterval.Valu
    }

    // Set log file format, if present.
    if (this.LogFileFormat.HasValue)
    {
        dc.LogFileFormat = (FileFormat)this.LogFileFormat.
    }

    // Build up the list of performance counters.
    string[] counterNames = new string[this.CounterNames.C
    for (int i = 0; i < this.CounterNames.Count; ++i)
    {
        counterNames[i] = this.CounterNames[i].ToString();
    }

    dc.PerformanceCounters = counterNames;

    // Now actually create (or modify existing) the set.
    dcs.Commit(this.Name, null, CommitMode.plaCreateOrModi

    // Return an opaque wrapper with which the user can co
    return new CollectorSetWrapper(dcs);
}
```

The interface `ICollectorSet` provides a simple non-PLA-dependent API for interacting with the data collector set:

```
public interface ICollectorSet : ISessionController
{
    void Delete();
}

public interface ISessionController
{
    void Start();

    void Stop();
}
```

Finally, pulling it all together, a sample application to create a basic counter set and output a CSV file. Note that manipulating data collector sets requires special privileges. The simplest way to avoid “access denied” errors is to just run any PLA app elevated.

```
CounterCollectorInfo info = new CounterCollectorInfo("MyCo

info.SampleInterval = TimeSpan.FromSeconds(1.0d);
info.LogFileFormat = LogFileFormat.CommaSeparated;
info.OutputPath = Environment.CurrentDirectory;

info.CounterNames.Add(new CounterName() { Category = "Proc
info.CounterNames.Add(new CounterName() { Category = "Syst
info.CounterNames.Add(new CounterName() { Category = "Proc

ICollectorSet collector = info.Create();
collector.Start();

Thread.Sleep(5000);

collector.Stop();

collector.Delete();
```

Run the program and you'll get an output CSV file named "MyCounters-" followed by a timestamp. The contents will look something like this:

```
"(PDH-CSV 4.0) (Pacific Standard Time)(480)","\\Your-PC-Name\Process(explorer)\Thread Count","\\Your-PC-Name\System\System Calls/sec","\\Your-PC-Name\Processor(_Total)\Interrupts/sec"
"12/27/2013 12:24:33.231","39","46015.248698065603","3062.4673653801251"
. . .
```

By Brian Rogers | 27 December, 2013 | diagnostics | 1 Comment |

← Alternatives to DisposeAsync   Threads don't scale →

One thought on "Using PLA.dll to collect perf counters"

ranyao
12 February, 2014 at 8:14 am

Is it possible to schedule a task for PerfCounterAlert using Pla library? Something like to create a dump file when #thread > threshold?

Reply ↓

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Post Comment