Logging

# Purpose

Write logging events to an output destination for purposes of support AFTER the product has been released to customers.

# Features

## Logging

* Thin wrapper around built-in .NET System.Diagnostics Trace and TraceSource logging. This allows transparent use of ALL pre-existing Trace and TraceSource API *including* what is already built-in to WPF,WCF, System.Net, etc.
* No changes or extensions to the App/Web config configuration.
* Simple logging API that avoid any message processing/formatting unless message is actually going to be written.
* Efficient TraceSource allocation. Unlike the TraceSource constructor which always creates a new trace source, this logging constructor reuses pre-existing open trace sources.
* Redirect all Trace/Debug messages that do not contain a trace source name (some do!) to the built-in ‘TRACE’ trace source.
* Optionally clone all Console.WriteXXX() messages to the built-in ‘CONSOLE’ trace source. Useful when there is no console window.
* Optionally capture all first-chance exceptions (cannot be hidden by try/catch/swallow blocks) and write to the built-in ‘FIRSTCHANCE’ trace source. Exceptions cannot hide.
* Detects changes to the <system.diagnostics> section of the application or web config. Changes to other parts of the config are ignored. Useful for changing the logging severity without rebooting the application.
* Optionally capture and copy all logging events and post asynchronously to subscribed C# event handlers. Maybe show logging in a live UI status window?
* Programmatically set/reset trace source severity as needed.
* Upon exit, queued messages are flushed before exit. None are lost.

## Logging Listeners/Appenders

Basic logging features available to all the following listeners. These are not available to the built-in System.Diagnostics listeners.

* Optionally lazily write messages to their output destinations in a low-priority worker thread.
* Optionally handle custom message formatting with many more parameters than that provided by System.Diagnostics.
* Handle redirection of Trace/Debug messages that do not contain a trace source (some do!) to the ‘TRACE’ trace source.
* Safely cleanup upon close/dispose. Asynchronous messages are completely flushed before application exit.
* Efficiently re-use pre-existing trace sources. System.Diagnostics does not!
* Thread-safe and cross-process safe.
* Initialized only upon first log message/event.

### EventLogTraceListener

All of all the basic logging listener features, plus…

* Optionally creates the event log and event source if they do not exist.

### FileTraceListener

All of all the basic logging listener features, plus…

* Output file name may include any pre-existing environment variables.
* File will create a new file (aka ‘roll over‘) after a max file size has been set, default=100MB
* Rollover file count exceeding a maximum limit will be deleted, oldest first. Default=-1 (e.g. never deleted).
* Files may optionally have a header and/or footer. Important for XML-like files. Nice for CSV.
* Same file may be safely written to by multiple processes.
* CSV-like format strings (e.g. {0:s},{1},”{2}”, …) will be reformatted and results written out as formal CSV readable by Excel, including multi-line fields.

### DebugTraceListener

Just the basic logging listener features.

### EmailTraceListener

All of all the basic logging listener features, plus…

* Optionally override the <system.net><mailsettings> properties.

### DatabaseTraceListener

Just the basic logging listener features.

## Performance Test

This test writes all messages to a single file. 3 threads each writing 100,000 messages \* 3 processes (900,000 messages total) asynchronously took 6 seconds to complete. It took an additional 4 seconds to for all three processes to flush their message queues upon exit. This result is ideal, as more resource-intensive properties (like retrieving the callstack) and/or more resource-intensive logging will cause the duration to increase.

# Example App.Config

The following example shows, with comments, all the possible permutations of the system.diagnostics section of the app/web.config.

<?xml version="1.0"?>

<configuration>

  <connectionStrings>

    <add name="LoggerDB" connectionString="Data Source=(local);Initial Catalog=VIA\_Security;Integrated Security=SSPI" providerName="System.Data.SqlClient"/>

  </connectionStrings>

  <system.diagnostics>

    <!--default list of listeners for new sources not listed below-->

    <trace autoflush="false" indentsize="4"><listeners><clear/><add name="DebugListener"/></listeners></trace>

    <switches>

      <!-- Switch value choices: Off Critical Error Warning Information Verbose All -->

      <add name="Group1" value="All"/>

    </switches>

    <sources>

      <!-- switchValue Choices: Off Critical Error Warning Information Verbose All -->

      <!-- TRACE, CONSOLE, and FIRSTCHANCE are either on or off: To turn on: TRACE=Verbose, CONSOLE=Information, FirstChance=Error -->

      <source name="TRACE" switchName="Group1"><listeners><clear/><add name="CsvListener"/></listeners></source>

      <source name="CONSOLE" switchName="Off"><listeners><clear/><add name="CsvListener"/><add name="DebugListener"/></listeners></source>

      <source name="FIRSTCHANCE" switchName="Group1"><listeners><clear/><add name="CsvListener"/><add name="DebugListener"/></listeners></source>

      <source name="General" switchValue="All" switchType=""><listeners><clear/><add name="FileListener"/></listeners></source>

      <source name="MySource" switchValue="Warning"><listeners><clear/><add name="MySourceFileListener" type="MyLogger.FileTraceListener, Logger.TestHarness" initializeData="Filename=MySource.log;SqueezeMsg=true"/></listeners></source>

      <source name="SystemStatus" switchValue="Off" switchType=""><listeners><clear/><add name="EmailListener"/></listeners></source>

    </sources>

    <sharedListeners>

      <!-- Listener common properties -

      /// Our base trace listener class that handles 99% of the work.

      /// Listeners derived from this class:

      /// (1) Optionally handle custom message formatting.

      /// (2) Handle redirection of Trace/Debug messages that do not contain a traceSource (some do!) to the "TRACE" traceSource.

      /// (3) Safely cleanup upon close/dispose.

      /// (4) Efficiently re-use pre-existing trace sources. System.Diagnostics by default does not!

      /// (5) Thread-safe.

      /// (6) Initialized only upon first log message/event.

      ///

      /// The app.config listener attribute 'initializeData', contains a stringized, case-insensitive, dictionary of name/value (delimited by '=') pairs delimited by ';'

      /// These dictionary keys and default values are listed below:

      ///    SqueezeMsg=false - squeeze multi-line FullMessage and duplicate whitespace into a single line.

      ///    Async=true - lazily write messages to output destination. False may incur performance penalties as messages are written immediately.

      ///    IndentSize=Trace.IndentSize (typically 4) - How many spaces to indent succeeding lines in a multi-line FullMessage.

      ///    Format=(default determined by derived class) - same as string.Format format specifier with arguments. See string.Format().

      ///    Possible 'Format' argument values are (case-insensitive):

      ///       string UserMessage - message provided by user to logging api. If null or empty, ExceptionMessage is returned.

      ///       string Exception - full exception provided by user to logging api. If exception not available, CallStack is returned.

      ///       string ExceptionMessage - The message part of exception or "" if exception not provided.

      ///       TraceEventType Severity - the severity assigned to this logging event.

      ///       ushort SourceId - the source index as defined by the order of sources in app.config

      ///       string SourceName - the name of the source for this logging event.

      ///       string DomainName - friendly name of the AppDomain that these logging api are running under or "" if AppDomain.FriendlyName not set.

      ///       string EntryAssemblyName - namepart of the assembly that started this AppDomain

      ///       Guid ActivityId - Unique id in order to group events across AppDomains/Processes.

      ///       string CallStack - call stack for this logging api (excluding the internal logging calls). May incur a logging performance penality.

      ///       string LogicalOperationStack - comma-delimited list of logical operations.

      ///       DateTime LocalDateTime - local time when this logging API was called.

      ///       DateTime DateTime - UTC time when this logging API was called.

      ///       int ProcessId - process ID for this instance of the application.

      ///       string ProcessName - process name for this application.

      ///       int ThreadId - managed thread id for the current thread.

      ///       string ThreadName - thread name for the current thread.

      ///       long Timestamp - high-resolution time that this logging api was called.

      ///       string UserData - user-defined object. Must have had overridden ToString() to get more than the class name.

      ///   App.Config 'Format' example:

      ///   "FORMAT=&quot;{0:yyyy-MM-ddTHH:mm:ss.fff} Severity: {1}, Source: {2}\r\nMessage: {3}\r\nException: {4}&quot;,DateTime,Severity,SourceName,UserMessage,ExceptionMessage"

      ///   or with implicit newlines....

      ///   "FORMAT=&quot;

      ///   {0:yyyy-MM-ddTHH:mm:ss.fff} Severity: {1}, Source: {2}Message: {3}

      ///   Exception: {4}&quot;, DateTime, Severity, SourceName, UserMessage, ExceptionMessage"

      ///

      /// Note: trace source 'traceOutputOptions' attribute is ignored as the initializeData 'Format' property handles this much better.

      ///

      /// Additional dictionary items are handled by the derived class.

      -->

      <!-- EventLogTraceListener

      /// Write log messages to the Windows Event Log.

      /// Creates Event log and/or source if it does not already exist.

      ///

      /// 'initializeData' is a dictionary of name/value pairs.

      /// These are:

      ///   Machine="." - computer whos event log to write to. Requires write access.

      ///   Log=(no default). EventLog log to write to. If undefined EventLog logging disabled.

      ///   Source=(no default). EventLog source to write to. If undefined EventLog logging disabled.

      ///

      /// If 'Format' is undefined, the default is:

      ///   string.Format("Category: {0}\r\n{1}{2}", SourceName, UserMessage, Exception);

      -->

      <add name="EventLogListener" type="MyLogger.EventLogTraceListener, Logger.TestHarness" initializeData="Log=OmnicellAnalytics;Source=Analytics">

        <!--Optional MultiSourceFilter initializeData attribute contains a comma-delimited list of sources (case-insensitive) to ignore -->

        <filter type="MyLogger.MultiSourceFilter, Logger.TestHarness" initializeData="Boogers,xxoo,zzz"/>

      </add>

      <!-- DebugTraceListener

      /// Write log messages to the debugger output.

      /// Output is available to an external debug viewer such as Microsoft's Dbgview.exe or

      /// the VisualStudio debugger output window, but not both.

      /// See: https://technet.microsoft.com/en-us/sysinternals/bb896647

      ///

      /// Note: There are no 'initializeData' properties unique to this derived class.

      -->

      <add name="DebugListener" type="MyLogger.DebugTraceListener, Logger.TestHarness" initializeData="SqueezeMsg=false"/>

      <!-- FileTraceListener

      /// Write log messages to the specified file.

      ///

      /// 'initializeData' is a dictionary of name/value pairs

      /// These are

      ///   Filename=Same as appname with a ".log" extension - Relative or full filepath which may

/// contain environment variables including pseudo-environment variables: ProcessName,

/// ProcessId(as 4 hex digits), AppDomainName, and BaseDir. DateTime in filename is not supported.

      ///   MaxSize=104857600 (100MB) - max file size before starting over with a new file.

      ///   MaxFiles=-1 (infinite) - Maximum number of log files before deleting the oldest.

      ///   FileHeader=(no default) - String literal to insert as the first line(s) in a new file.

      ///   FileFooter=(no default) - String literal to append as the last line(s) in a file being closed.

      ///

      /// If 'Format' is undefined, the default is (CSV):

      ///   string.Format("{0:yyyy-MM-dd HH:mm:ss.fff},{1},{2},\"{3}\"", LocalDateTime, Severity, SourceName, UserMessage);

      -->

      <add name="FileListener" type="MyLogger.FileTraceListener, Logger.TestHarness" initializeData="Filename=MyLog.log;Format=&quot;{0:yyyy-MM-ddTHH:mm:ss.fff} Severity: {1}, Source: {2}\r\nMessage: {3}\r\nException: {4}&quot;,DateTime,Severity,SourceName,UserMessage,ExceptionMessage"/>

      <add name="CsvListener"  type="MyLogger.FileTraceListener, Logger.TestHarness" initializeData="

           Filename=MyLog.csv;

           FileHeader=DateTime,TimeStamp,Severity,SourceName,Message;

           Format=&quot;{0:yyyy-MM-ddTHH:mm:ss.fff}, {1}, {2}, {3}, {4}&quot;,DateTime,TimeStamp,Severity,SourceName,UserMessage"/>

      <!-- EmailTraceListener

      /// Write log messages as email messages to the mail server.

      ///

      /// 'initializeData' is a dictionary of name/value pairs.

      /// These are:

      ///   Subject="Log: "+SourceName - email subject line.

      ///   SendTo=(no default) - comma-delimited list of email addresses to send to. Whitespace is ignored. Addresses may be in the form of "username@domain.com" or "UserName &lt;username@domain.com&gt;". If undefined, email logging is disabled.

      ///   The following are explicitly defined here or defaulted from app.config configuration/system.net/mailSettings/smtp;

      ///   SentFrom=system.net/mailSettings/smtp/@from - the 'from' email address. Whitespace is ignored. Addresses may be in the form of "username@domain.com" or "UserName &lt;username@domain.com&gt;".

      ///   ClientDomain=LocalHost - aka "www.gmail.com"

      ///   DefaultCredentials=true - true to use windows authentication, false to use UserName and Password.

      ///   UserName=(no default)

      ///   Password=(no default)

      ///   EnableSsl=false -

      ///   MailServer=(no default) - aka "smtp.gmail.com"

      ///   Port=25 - mail server listener port to send messages to.

      ///

      /// If 'Format' is undefined, the default is:

      ///   string.Format("DateTime : {0:yyyy/MM/dd HH:mm:ss.fff}\r\nSeverity : {1}\r\nSource   : {2}\r\nMessage  : {3}", LocalDateTime, Severity, SourceName, UserMessage);

      -->

      <add name="EmailListener" type="MyLogger.EmailTraceListener, Logger.TestHarness" initializeData="

           SentFrom=System Admin &lt;charlesh@omnicell.com&gt;;

           SendTo=Chuck Hill &lt;chuckhill2@gmail.com&gt;;

           Format=&quot;

DateTime : {0:yyyy/MM/dd HH:mm:ss.fff}

Severity : {1}

Source   : {2}

Message  : {3}&quot;, DateTime, Severity, SourceName, FullMessage)"

        />

      <!-- DatabaseListener

      /// Write log messages to database table.

      ///

      /// 'initializeData' is a dictionary of name/value pairs.

      /// These are:

      ///   ConnectionString=(no default) - a string key representing AppConfig ConfigurationManager.ConnectionStrings[] dictionary entry OR literal full SQL connection string.

      ///   SqlStatement=(no default) - SQL statement to insert logging values into the database table.

      ///   Examples:

      ///      "spStoredProcedure @LocalDateTime, @Severity, @SourceName, @UserMessage"

      ///      "spStoredProcedure @Date=@LocalDateTime, @Severity=@Severity, @Source=@SourceName, @Message=@UserMessage"

      ///      "spStoredProcedure @Date={0}, @Severity={1}, @Source={2}, @Message={3}", LocalDateTime, Severity, SourceName,@UserMessage

      ///      "INSERT INTO MyTable ([Date],Severity,Source,Message) VALUES (@LocalDateTime, @Severity, @SourceName, @UserMessage)"

      ///      "INSERT INTO MyTable ([Date],Severity,Source,Message) VALUES ({0}, {1}, {2}, {3})", LocalDateTime, Severity, SourceName, UserMessage

      /// The 'Format' and 'IndentSize' properties are not used.

      -->

      <add name="DbListener" type="MyLogger.DatabaseListener, Logger.TestHarness" initializeData="

           ConnectionString=Data Source=(local)\;Initial Catalog=VIA\_Security\;Integrated Security=SSPI;

           SqlStatement=INSERT INTO MyTable ([Date],Severity,Source,Message) VALUES (@LocalDateTime, @SeverityString, @SourceName, @UserMessage)"/>

    </sharedListeners>

  </system.diagnostics>

  <system.net>

    <mailSettings>

      <!--

      <smtp deliveryMethod="Network" from="Logger &lt;myName@omnicell.com&gt;">

        <network

          clientDomain="www.omnicell.com"

          defaultCredentials="true"

          userName=""

          password=""

          enableSsl="false"

          host="smtp.omnicell.com"

          port="25"

        />

      </smtp>

       -->

      <smtp deliveryMethod="Network" from="Logger &lt;myName@omnicell.com&gt;">

        <network

          defaultCredentials="false"

          userName="myName@gmail.com"

          password="myPassword"

          enableSsl="true"

          host="smtp.gmail.com"

          port="587"

        />

      </smtp>

    </mailSettings>

  </system.net>

  <startup>

        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>

    </startup>

</configuration>