.NET Logging Wrapper 3.0 Requirements Specification

1. Scope

1.1 Overview

The existing .NET Logging Wrapper 2.0 component provides a standard logging API with support for pluggable back-end logging solutions.

This document outlines several core enhancements to the component as well as a couple of new ancillary features to aid integration. These are:

- The storing of log message text and other attributes in configuration files
- The ability to pass log event properties through to log4net and other providers
- Removing the need for a separate back-end provider configuration in simple usage
- The provision of a trace event listener for integration with Enterprise Library 3.1
- Support for TopCoder's Enterprise Logging Service as a back-end
- Filtering of specific logging levels within the Logging Wrapper component
- Configuration API support.

The requirements have been kept terse in favor of a broad set of examples, but the examples are recommendations only; the designer is free to implement the required features however they choose.

1.2 Logic Requirements

1.2.1 Configuration API support

Logging Wrapper 3.0 will support reading configuration using the Configuration API component. It will not force the use of any specific persistence provider, and by default it will be able to read unaltered Logging Wrapper v2.0 configuration.

1.2.2 Named Log Messages

Logging Wrapper 3.0 will provide the ability to name specific log messages, define the text of the message, and define the logging level for the message all within the component configuration.

Logging Wrapper 3.0 will then allow callers the option of indirectly referencing a log message by passing a log message name rather than log message text.

When a log message name is passed to it, Logging Wrapper 3.0 will use the name to look up the log message text and the logging level from the component configuration.

It is intended that this be the norm in future Logging Wrapper usage, so ease-of-use is very important.

See example 1.4.1.

1.2.3 Named Parameters

Logging Wrapper 3.0 will take the array of parameters that have been passed to it in conjunction with a Named Log Message (see requirement 1.2.2) and will attempt to associate each parameter with a parameter name taken from the component configuration (see example 1.4.1).

This association will be made individually for each Named Log Message (1.2.2) and by parameter index (a zero based integer value).

The name and value will be made available to the back-end solution if arbitrary message attributes are supported by it. If unsupported, no action is taken.

Log4net does support arbitrary attributes; a pattern layout will be able to reference named parameters using the property variable: `%property{namedParameterName}`

See http://logging.apache.org/log4net/release/manual/contexts.html for more detail.

A practical example of this functionality can be found in example 1.4.2.

1.2.4 Logging Resilience

Logging Wrapper 3.0 will catch and wrap all exceptions thrown from the back-end solution and will attempt to catch all exceptions thrown internally within this component. A true/false configuration option will be supported. If set, the component will attempt to log these exceptions under the logging level WARN, and will not propagate them to the caller. Even if the warning cannot be logged, the exception will not be propagated to the caller. See example 1.4.4.

1.2.5 Zero-configuration for back-end solutions

The component will define several common internal default configurations that support the TopCoder process.

These will be at least:

- TEST - For use when the component is under automated test
  - Example: all messages will be logged verbosely to the file '../../test_files/log.txt'
- COMPONENT - For use when the component is standalone
  - Example: only Level.INFO and above messages will be logged, and these will be logged to the file 'log.txt' in the current directory
- CERTIFICATION - For use when the component is integrated with an application
  - Example: all messages are logged to a dated log file in a logs/ subfolder. Logs are rolled over regularly
- CLIENT_DEBUG - For use when the component is deployed at a client site
  - Example: As CERTIFICATION, but only a limited set of log files are kept
- CLIENT_STRESS - For use when the component is must be at its fastest at a client site
  - Example: As CLIENT_DEBUG, but only Level.ERROR and Level.FATAL are logged.
- RELEASE - For use when the component is in production at a client site
  - Example: As CLIENT_STRESS, but only Level.WARN and above messages are logged. Approximately 1 month's worth of dated log files will be kept.

The component will support selecting a default configuration as a single, optional, configuration property. When this property has been set Logging Wrapper 3.0 will check at start-up for the existence of an external configuration file for the back-end solution, and if it does **not** exist, it will generate one based on the selected default configuration.

The details of each configuration will be internal to the Logging Wrapper and specific to the back-end solution in use.

See example 1.4.3.

1.2.6 Logging to the Enterprise Logging Service (ELS) generic component

The ELS provides a WCF service that provides access to an instance of the Logging Wrapper.

Note that .NET 3.0 WCF libraries may be used to communicate with ELS as long as such usage does not require WCF libraries to be deployed when ELS is not being used.

1.2.6.1 ELS Logging Wrapper Implementation

Logging Wrapper 3.0 will provide a back-end solution that forwards log messages to an ELS instance, and this will have the class name `TopCoder.LoggingWrapper.ELS.ELSImpl`. See example 1.4.5.

1.2.6.2 ELS log4net appender

Logging Wrapper 3.0 will provide a log4net appender that forwards log messages to an ELS instance, and will have the class name `TopCoder.LoggingWrapper.ELS.ELSAppender`. See example 1.4.6.

1.2.7 Providing a Trace Listener Implementation for Enterprise Library 3.1

Logging Wrapper 3.0 will provide an Enterprise Library 3.1 Trace Listener (see 2.1.2) that handles log events using Logging Wrapper 3.0 functionality. The class will be named `TopCoder.LoggingWrapper.EntLib.LoggingWrapperTraceListener.` The class will also reside in a separate assembly or the designer must otherwise eliminate the dependency on the Enterprise Library when this feature is not in use.

Information on the Enterprise Library Logging Application Block can be found here:
http://msdn2.microsoft.com/en-us/library/aa480464.aspx

A sample configuration fragment of how the Logging Application Block might be configured to use this Trace Listener can be found in example 1.4.7.

1.2.8 Filtering by Logging Level

Logging Wrapper 3.0 will provide the ability to filter log messages by Level. Filtered log messages will not be written to the logs. Which Levels are filtered will be identified within the component configuration. In the absence of specific configuration, no Levels will be filtered. Example 1.4.8 demonstrates a sample configuration for this feature.

1.2.9 Inherited Requirements:

The following requirements are from Logging Wrapper v2.0, with 1.2.9.3 (originally 1.2.3) slightly reworded to encompass the new features.

1.2.9.1 Multiple Configurations

It will still be possible, as in Logging Wrapper 2.0, to have several different Logs which may be configured differently. To facilitate this, the configuration namespace must still be an optional parameter when creating a Log (if no namespace is given, then a default namespace should be used.)

1.2.9.2 Thread-Safety

The component should be thread-safe to ensure multiple users can work with the same or different logs without problems.

1.2.9.3 Signing Issues

TopCoder needs to be able to sign this component without signing the Log4Net piece of the application or the Enterprise Library piece of the application. Namespaces have been separated out to allow multiple assemblies to be used where necessary.

1.2.9.4 Backwards Compatibility

The old API must remain for backwards compatibility purposes, but any or all of it may be marked as [Obsolete] and implemented as a wrapper around the new functionality.

1.2.9.5 ASP.NET Compatibility

When using this component as part of an ASP.NET web application, there are additional security restrictions on what files and registry logs the app is allowed to write to. The designer and developer should ensure that there are no unnecessary restrictions to the functionality of this component under such security constraints. Any necessary restrictions should be clearly documented.

1.2.9.6 Documentation Quality

Since this component is used in so many other components, it is critical that the “Installation and Configuration” and “Usage Notes” sections of the component specification be clear, concise, and correct. Reviewers must ensure that this documentation is sufficient to easily get the Logging Wrapper setup and usable for other components.

1.3 Required Algorithms

None required.

1.4 Example of the Software Usage

The initial use of the .NET Logging Wrapper component has been within other TopCoder Software components. This will allow TopCoder Software components to be plugged into an existing environment without requiring the additional configuration and implementation of a specific logging solution.

Example configuration file for Logging Wrapper 3.0 using Configuration Manager:

```
<?xml version="1.0" encoding="utf-8"?>
<ConfigManager>
 <namespace name="TopCoder.LoggingWrapper">
    <property name="logger_name">
      <value>logger</value>
    </property>
    <property name="default_level">
      <value>DEBUG</value>
    </property>
    <property name="logger_class">
      <value>TopCoder.LoggingWrapper.Log4NETImpl</value>
    </property>
    <property name="config_file">
      <value>log4net.config</value>
    </property>
 </namespace>

 <namespace name="TopCoder.LoggingWrapper.OnLogDataError">
    <property name="text">
      <value>Invalid data has been received: {0}</value>
    </property>
       <property name="level">
         <value>WARN</value>
       </property>
       <property name="name0">
         <value>dataContent</value>
       </property>
 </namespace>
</ConfigManager>
```

1.4.1 Use of named log messages:

A common problem in applications built with logging is the proliferation of logging messages throughout the code, and intended for multiple users: developers at our clients, or for our own TopCoder development. Spelling mistakes require a rebuild of the application and cannot be located easily. Additionally, the need for specific groups of logging messages changes as different areas of the application are inspected, so logging levels are also subject to change.

With the configuration above, a message has been created with the key ‘OnLogDataError’. This message has a defined format string, a defined debugging level, and has also named the first parameter ‘dataContent’ so that it will be available to the back-end solution with that key.

It might be referenced in code like this:

```
const ON_LOG_DATA_ERROR = "{OnLogDataError}";
Logger logger = LogManager.CreateLogger();

...

logger.Log(ON_LOG_DATA_ERROR,badData);
```

However, the final decision on the API is up to the designer.

1.4.2 Use with database appenders:

This log4net configuration demonstrates how this named parameter support allows the use of database appenders in a way that is not possible in Logging Wrapper v2:

```
<log4net debug="false">
  <appender name="AdoNetAppender_Oracle" type="log4net.Appender.AdoNetAppender">
    <connectionType value="System.Data.OracleClient.OracleConnection,
System.Data.OracleClient, Version=1.0.3300.0" />
    <connectionString value="data source=[mydb];User ID=[user];Password=[password]" />
    <commandText value="INSERT INTO DataErrors (Data) VALUES (:data)" />
    <bufferSize value="128" />
    <parameter>
        <parameterName value=":data" />
        <dbType value="String" />
        <size value="4000" />
        <layout type="log4net.Layout.PatternLayout">
            <conversionPattern value="%property{dataContent}" />
        </layout>
    </parameter>
  </appender>
</log4net>
```

Note that the layout for the database parameter **:data** is defined to use the named property dataContent. For more information on database appenders:

http://logging.apache.org/log4net/release/config-examples.html

1.4.3 Zero-configuration for back-end solutions:

Currently a user of the Logging Wrapper component must also understand the back-end solution and configure it manually. With the new logging wrapper, a default log4net configuration file will be emitted that will serve the common cases. This emitted file will be configured differently depending on which default configuration has been elected in the Configuration API or Configuration Manager files.

```
<ConfigManager>
```

```
 <namespace name="TopCoder.LoggingWrapper">
    <property name="logger_name">
      <value>logger</value>
    </property>
        <property name="default_config">
          <value>Release</value>
        </property>
    <property name="logger_class">
      <value>TopCoder.LoggingWrapper.Log4NETImpl</value>
    </property>
    <property name="config_file">
      <value>log4net.config</value>
    </property>
 </namespace>
</ConfigManager>
```

The above example configuration indicates that a default configuration of 'Release' should be used. If Logging Wrapper 3.0 cannot find an existing log4net.config file, it must create one. Per the designers choice, 'Release' mode might mean that only log messages at or above the WARN level are logged, that a rolling file appender is used, and that only 2 months worth of logs are kept.

1.4.4 Logging Resilience:

After deploying to the client, it is not acceptable for TopCoder software to break. While we try to avoid this, it sometimes happens. In debug code (such as logging), this is not acceptable. So, Logging Wrapper 3.0 provides an optional parameter that causes Logging Wrapper to log a warning and otherwise ignore the error:

```
<ConfigManager>
 <namespace name="TopCoder.LoggingWrapper">
    <property name="propagate_exceptions">
      <value>false</value>
    </property>
</namespace>
</ConfigManager>
```

1.4.5 Use with Enterprise Logging Service (ELS) as an implementation:

Another TopCoder component, ELS presents a WCF wrapper façade around the Logging Wrapper. With this revision of the Logging Wrapper, the user may choose to use the ELS client implementation for the back-end solution:

```
<ConfigManager>
 <namespace name="TopCoder.LoggingWrapper">
    <property name="logger_class">
      <value>TopCoder.LoggingWrapper.ELS.ELSImpl</value>
    </property>
        <property name="els_url">
          <value>http://localhost/EnterpriseLogger/LoggingService.svc</value>
        </property>
 </namespace>
</ConfigManager>
```

1.4.6 Use with Enterprise Logging Service (ELS) as an appender:

In some cases the user may already have log4net applications that should hook into the central ELS. In this case, a simple appender is used:

```
<log4net debug="false">
  <appender name="AdoNetAppender_Oracle" type="TopCoder.LoggingWrapper.ELS.ELSAppender">
        <url>http://localhost/EnterpriseLogger/LoggingService.svc</url>
  </appender>
</log4net>
```

1.4.7 *Use with the Microsoft Enterprise Library, Logging Application Block.*

Finally the user may have current applications that use Enterprise Library 3.1, but wish to hook those applications in to the TopCoder Logging Wrapper infrastructure.

In this case the user will modify the .NET application configuration file to add a new listener:

```
<configuration>
<loggingConfiguration name="Logging Application Block" tracingEnabled="true"
   defaultCategory="General" logWarningsWhenNoCategoriesMatch="true">
   <listeners>
     <add
listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.CustomTrace
ListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=null"
       traceOutputOptions="None"
       type="TopCoder.LoggingWrapper.EntLib.LoggingWrapperListener,
       TopCoder.LoggingWrapper.EntLib"
       name="TopCoderListener" initializeData="" />
   </listeners>
 </loggingConfiguration>
</configuration>
```

All log messages that are sent to the listener ‘TopCoderListener’ will then be processed by the LoggingWrapper.

1.4.8 *Filtering by Logging Level*

The below configuration file would cause the Logging Wrapper to filter all Debug, Success Audit and Failure Audit messages that come in. None of the messages that are sent to the Logging Wrapper at these levels will be present in the logs.

```
<ConfigManager>
 <namespace name="TopCoder.LoggingWrapper">
    <property name="filtered_levels">
        <value>DEBUG</value>
        <value>FAILUREAUDIT</value>
        <value>SUCCESSAUDIT</value>
    </property>
 </namespace>
</ConfigManager>
```

1.5 Future Component Direction

- Support for handling and passing wrapped LogEvents which are at the root of both the Enterprise Library Logging Block and the log4net framework
- Addition of the Enterprise Logging Block as a Logging Wrapper back-end implementation
- Support for categories as exist in the Enterprise Logging Block.
- Support for timing blocks of code

2. Interface Requirements

2.1.1 *Graphical User Interface Requirements*

None specified.

2.1.2 *External Interfaces*

The designer is free to add new APIs to support new features however backward compatibility must be maintained when none of the new features are being used.

The Enterprise Logging Service may be downloaded from the catalog and will be posted to the forums.

The Enterprise Library Logging Application Block is described here: http://msdn2.microsoft.com/en-us/library/aa480464.aspx

2.1.3 *Environment Requirements*

- Development language: C#

2.1.4 *Namespace*

TopCoder.LoggingWrapper
TopCoder.LoggingWrapper.ELS
TopCoder.LoggingWrapper.EntLib

3. Software Requirements

3.1 Administration Requirements

3.1.1 *What elements of the application need to be configurable?*

- Named log messages:
  - o Message key
  - o Text (text may include parameter placeholders)
  - o Parameter names (optional)
  - o Logging level
- Whether to throw exceptions if too few or too many parameters are passed for a message
- Which (if any) built-in configuration to use
- Which (if any) logging levels to filter

3.2 Technical Constraints

3.2.1 *Are there particular frameworks or standards that are required?*

.NET Framework 2.0 must be supported for the core Logging Wrapper assembly.

3.2.2 *TopCoder Software Component Dependencies:*

Configuration Manager 2.0.1
Configuration API 1.0
Enterprise Logging Wrapper 1.0 (indirectly)

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 *Third Party Component, Library, or Product Dependencies:*

Log4Net 1.2 (1.2.9 support is required)
Enterprise Library 3.1 must be supported by the Logging Wrapper Trace Listener.

3.2.4 *QA Environment:*

- Windows Server 2003
- Windows XP / Vista

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

XML documentation must provide sufficient information regarding component design and usage.