Software Design Description

Wayne.Lib.Log

2008-02-13
Rev 1.4

Table of Contents

0. Document information

0.1 Revision history

Revision	*Date/Sign*	*Change description*
1.0	Roger Månsson	Created
1.1	Roger Månsson	Added the File path parameters.
1.2	Mattias Larsson	Added debug log 'category', recursive ancestry filters etc.
1.3	Roger Månsson	Added event logging. Renamed in XSD: LogConfig->LogConfigFile and LogFile->LogConfig.
1.4	2008-02-13 Mattias Larsson	Refreshed doc.

0.2 Purpose and scope

The purpose of this document is to describe the usage and design of the log library.

0.3 References

1. Introduction

This document describes the functionality of the assembly Wayne.Lib.Log. It does yet only describe **debug logging** so event and error logging are yet to be implemented. It is included in the API reference though.

The logger is configured through an XML file. It specifies what should be logged and in what way. What should be logged is evaluated by filters that can both include and exclude. The output for a set of filters can be configured. At the moment, only text file logging is available but more logging facilities can be added in the future.

The filters use the interface IIdentifiableEntity defined in the Wayne.Lib.Common assembly. The interface exposes the properties that are used for the filtering. Any part of an application that is going to log something must have an IIdentifiableEntity reference. It can be that the class itself implements it but it can also be that several objects share one identifiable instance that is used for the logging.

When the logging takes place the application will also specify a detail level. The log file configuration can include what detail level that should be used on the items matching the filter.

Further on a Category can be specified when performing the logging.

2. Debug logging

2.1 Usage

Begin by pointing out your application's log configuration xml-file in the start-up of the program:

```
Logger.SetConfigFile("MyLogConfig.xml");
Logger.RefreshConfiguration();
```

You can also hook to the OnThreadException-event to get any exception occurring in the logging thread:

```
Logger.OnThreadException += new
   EventHandler<Wayne.Lib.EventArgs<Wayne.Lib.Exception<ExceptionType>>>
   (Logger_OnThreadException);
```

If you are intending to use a debug logger frequently in your application, it's a good design to keep one "persistently" by creating one as a private field and using that one throughout the program – rather than creating a new debug logger everywhere you want to log something.
In this case, you specify the persistent-flag (as 'true') when you ask for a debug logger:

```
debugLog = Logger.CreateDebugLog(this, true, DebugLogLevel.Normal);
```

Now you can call the Add-method of your debugLog whenever you want to log something.
The Add-method comes in different flavors:

The simplest one just takes an object to log (using the detail level given when created the debug log.):

```
      debugLog.Add("Blah blah");
```

The debug level can be specified:

```
      debugLog.Add("Blah blah", DebugLogLevel.Detailed);
```

To categorize the debug output an optional Category-object can be specified, with or without a specified debug level:

```
      debugLog.Add("Blah blah", "Category");
      debugLog.Add("Blah blah", "Category", DebugLogLevel.Detailed);
```

If you want to create a volatile debug logger, the using-statement is the best practice to use:

```
    using (IDebugLog dLog = Logger.CreateDebugLog(this))
    {
        if (dLog.Active)
            dLog.Add("Blah blah");
    }
```

Another **important** thing to know is that before doing any logging, the best practice is to check whether the log is actually active or not. If not, quite a lot of code is executed – just to realize that nothing is logged. This is especially true when composed strings are being logged; since the whole string must first be built up – just to be thrown away.

```
if (dLog.Active)
    dLog.Add("Coords: (" + x.ToString() + "," + y.ToString() + ")");
```

2.2 Configuration

The configuration of the logging is specified for the whole AppDomain (usually for the whole exe file). The configuration is read when the Logger.Refresh(string logConfigFile) is called with a path to the log configuration xml file. This should be done by some initial code in the application.

2.2.1. XML format

Wayne LogConfig	Log Configuration		
	Field		**Description**
m	**LogConfigFile**		Root node
m	**LogConfig**	**(+)**	LogConfig is the root node for each log hive, it consists of a set of filters and a set of output handlers.
m	@Name	string bool	LogName is a name that identifies the log hive, but has no actual use at the moment.
o	@Enabled		Use Enabled to turn on/off the whole log hive without having to remove anything from the xml file. If not specified, default is **true**
o	Description	1	A text description of the Log config.
m	Filters	1	A collection of filters that filters out what should be logged in this file.
o	Filter	(*)	One entity filter is a pattern that should be included or excluded from the logging in this file. The pattern matching is done through regular expressions on all but the Detail level. It is filtered so only loggings with detail level less than or equal to the set detail level.
o	@EntityType	string	Matching the IIdentifiableEntity's entity type.
o	@EntitySubType	string	Matching the IIdentifiableEntiry's entity subtype.
o	@Id	string	Matching the Id. Note that the Id is a regular expression string that should textually match the integer Id in the target entity.
o	@Enabled	bool	Enables/disables this filter. Default (when omitted) is **true.**
o	@DetailLevel	->	Max detail level that should be logged. (Normal|Detailed|Maximized)
m	@FilterType	->	Specifies if this is an inclusive or exclusive filter. (Exlude|Include)
	@LogAncestry-Name	string	The LogAncestryName attribute tells whether the full hierarchy of the IIdentifiableEntity should be put in the prefix of each debug line.
o	Category	(*)	A collection of specific Categories that should override the default filtering.
m	@Name	string	Matching the name of the category.
o	@Enabled	bool	Enables/disables this category filter. Default is **true.**
o	@DetailLevel	->	Max detail level that should be logged (see above).
m	@FilterType	->	Specifies if this is an inclusive or exclusive filter (see above).
o	Filter		Child entity. *See the Filter-node above!* (Recursive)
M	Filters	1	A collection of filters that filters out what should be logged in this file.
m	Outputs	1	A collection of the output handlers that this log hive should output to.
o	Output	(*)	
m	@Type	String	Type of logger output to use. Only implemented so far is **'TextFileLogWriter'** that outputs to an ordingary log file.
o	@Enabled	bool	Enables/disables this logger output. Default (when omitted) is **true**
m	Parameters	1	Parameters for the output hander. Dependent on which type that is chosen.
	---- EITHER ----		
m	TextFileParams	1	

m	FilePath		Path that the file should be written to. Mixed element type so the tags below can be inserted into the paths. Example C:\Wayne\Log\FPos_<Id/>_<EntityType/>_<Date/>
O	Id EntityType EntitySubType Date	*	Id,EntityType,EntitySubType will be inserted from each logging identifiable entity. The date will be inserted with the format that is specified. The default format is yyyyMMdd. This enables month,date, hour logging.
	---- OR----		
	EventLogSubscriptionParams @SubscriberId @StorageType	1	Parmeters for event logging. The id string for the expected subscriber to the events. How the events should be stored between the event is issued and it is handled by the subscriber (NoStorage, InMemory, RestartSafe).
o	Leftovers See LogConfigFile/LogConfig/Outputs/Output	1	All loggings that did not match the filters can be dumped to a log file if this tag exists and is enabled. The format is exactly as the output node in the LogConfig/Outputs node.
o	LeftoverEntities	1	Same as Leftovers, but only the full entity name is logged once – no log texts are added. This is to get a list of the unconfigured entities.

The recursive filter mechanism is used to filter out particular entities of the same type that have different parents. For instance, assume a complex application with several Wayne-sockets in different modules. The internal socket logging could be impossible to be filtered out for only one of the sockets (since they have the same IIIdentifiableEntity-type and subtype and could have the same id – but actually are different entities).

In this case, you could specify the whole “entity path” (or as much as needed) to uniquely identify the entity.

Assuming an application with two sockets in different modules, both having the Id=1.
The application form is an IIIdentifiableEntity called “Application” with Id=0, and it contains two sub entities, called “ModuleA” and “ModuleB”; both with Id=0. The two modules contains the two sockets.

Just having the filter:

```
<Filter EntityType="Socket" FilterType="Include"/>
```

would give the following output in a log file.

```
11:39:53:557 Socket1: xxxxxxxxxxx...
11:39:54:258 Socket1: xxxxxxxxxxx...
11:40:01:114 Socket1: xxxxxxxxxxx...
11:40:02:532 Socket1: xxxxxxxxxxx...
```

which doesn’t tell us who’s socket it is that produced the log lines; is it the module A’s socket or module B?

If we want all socket communications in the same log file, we could specify the LogAncestryName attribute:

```
<Filter EntityType="Socket" FilterType="Include" LogAncestryName="true"/>
```

This would reveal the whole “entity path” to the sockets, showing that the lines originated from different sockets.

```
11:39:53:557 Socket1.ModuleA0.Application0: xxxxxxxxxxx...
11:39:54:258 Socket1.ModuleB0.Application0: xxxxxxxxxxx...
11:40:01:114 Socket1.ModuleA0.Application0: xxxxxxxxxxx...
11:40:02:532 Socket1.ModuleB0.Application0: xxxxxxxxxxx...
```

But assume we only want to see the socket from ModuleA. Then we could put the following filter in the configuration file:

```
<Filter EntityType="ModuleA" FilterType="Include">
```

```
  <Filter EntityType="Socket" FilterType="Include"/>
</Filter>
```

Or even

```
<Filter EntityType="Application" FilterType="Include">
  <Filter EntityType="ModuleA" FilterType="Include">
    <Filter EntityType="Socket" FilterType="Include"/>
  </Filter>
</Filter>
```

To explicitly exclude all socket logging (not putting it in the leftovers) but the one from Module A, the following filters will do:

```
<Filter EntityType="ModuleA" FilterType="Include">
  <Filter EntityType="Socket" FilterType="Include"/>
</Filter>
<Filter EntityType="Socket" FilterType="Exclude"/>
```

This means, that when a socket entity wants to log, the log configuration mechanism is trying to find a filter match with as long parent-hierarchy as possible.
In this case, the module A's socket will match both the *included* and *excluded* filter above, but the *included* has the most uniquely defined "parent path" (so the excluded will be ignored). The module B's socket will match only the *excluded* filter.

Note: If the same level of parent-hierarchy is found (bad configuration I guess), then an *included* filter is stronger than an *excluded*. Example:

```
<Filter EntityType="Socket" FilterType="Included"/>
<Filter EntityType="Socket" FilterType="Exclude"/>
```

Here the socket logging will be *included*.

2.3 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<LogConfigFile xmlns="http://www.wayne.com/2006-05-15/LogConfig.xsd">
  <LogConfig LogName="FPos" Enabled="true">
    <Filters>
      <Filter EntityType="FPos" FilterType="Include"/>
    </Filters>
    <Outputs>
      <Output Type="TextFileLogWriter">
        <Parameters>
          <TextFileParams>
            <FilePath>C:\Wayne\Log\FPos.txt</FilePath>
          </TextFileParams>
        </Parameters>
      </Output>
    </Outputs>
  </LogConfig>
</LogConfigFile>
```

In this example, we want all loggings from the entity type 'FPOS' to be written in the log file FPOS.txt.

If we want to exclude the FPOS 2 from the logging we add an excluding filter:

```
<?xml version="1.0" encoding="utf-8" ?>
<LogConfigFile xmlns="http://www.wayne.com/2006-05-15/LogConfig.xsd">
  <LogConfig LogName="FPos" Enabled="true">
    <Filters>
      <Filter EntityType="FPos" FilterType="Include"/>
      <Filter EntityType="FPos" Id="2" FilterType="Exclude"/>
```

```
    </Filters>
    <Outputs>
      <Output Type="TextFileLogWriter">
        <Parameters>
          <TextFileParams>
            <FilePath>C:\Wayne\Log\FPos.txt</FilePath>
          </TextFileParams>
        </Parameters>
      </Output>
    </Outputs>
  </LogConfig>
</LogConfigFile>
```

Here is a more complex example using two log files and regular expressions for the terminal filters. All identifiable entities that begins with 'Ter' and ends with 'al' will be logged in the terminal log. The log category "TaxCalc" of the pinpad log will be maximized, but the category "SocketComm" will be totally excluded.

```
<?xml version="1.0" encoding="utf-8" ?>
<LogConfigFile xmlns="http://www.wayne.com/2006-05-15/LogConfig.xsd">
  <LogConfig LogName="TerminalLog" Enabled="true">
    <Filters>
      <Filter Id="1|2|3" EntityType="Ter.*al" FilterType="Include"
              Enabled="true" DetailLevel="Detailed"/>
      <Filter Id="1" FilterType="Exclude" Enabled="false"/>
    </Filters>
    <Outputs>
      <Output Type="TextFileLogWriter" Enabled="true">
        <Parameters>
          <TextFileParams>
            <FilePath>C:\Wayne\Log\Terminal.txt</FilePath>
          </TextFileParams>
        </Parameters>
      </Output>
    </Outputs>
  </LogConfig>
  <LogConfig LogName="Pinpadlog">
    <Filters>
      <Filter EntitySubType="" EntityType="Pinpad" FilterType="Include"
              DetailLevel="Detailed">
        <Category Name="TaxCalc" DetailLevel="Maximized"
                  FilterType="Include"/>
        <Category Name="SocketComm" FilterType="Exclude"/>
      </Filter>
    </Filters>
    <Outputs>
      <Output Type="TextFileLogWriter">
        <Parameters>
          <TextFileParams>
            <FilePath>C:\Wayne\Log\Pinpad.txt</FilePath>
          </TextFileParams>
        </Parameters>
      </Output>
    </Outputs>
  </LogConfig>
</LogConfigFile>
```

2.4 Special tricks

2.4.1. DotNetLog

There is a constant EntityType that is ‘DotNetLog’ that can be used to direct the built-in dotnet debug log to a log file. By adding this filter to a log file, that log file will get all that is logged in the program through the System.Diagnostics.Debug class.

```
<Filter EntityType="DotNetLog" FilterType="Include"/>
```

3. Event logging

Event logging is performed through the same channel as the debug logging. An EventLogEntry is created with the identifiable entity and a category and it is sent to the Logger. The logger has a setup that identifies how the events should be handled. It uses the same schema as the Debug log configuration, but it is a separate XML file. The event log entries are processed by Event Subscribers. A class implements the interface IEventSubscriber and registers to the logger. After that it will receive the events it is configured to handle.

3.1 Publishing events

The definition of the events is done in the same application where they are generated. An event is defined by deriving from or using the EventLogEntry class. Each event has a category and a sender, which identifies the event. It is good practice to keep the supported event categories in an enumeration the possible event types together.

Example:
We want to notify that a link has gone down.

First we define the enumeration member that should represent this event.

```
enum EventLogType
{
     . . .
     LinkDown
     . . .

}
```

Then in the Link down method, we create an EventLogEntry with information about the sending class, a descriptive text and the category.

```
void LinkDown(object sender, EventArgs e)
{
    EventLogEntry entry = new EventLogEntry(this, "Link is down",
                                                EventLogType.LinkDown);
    Logger.AddEntry(entry);
}
```

Or even better would be to derive a special EventLogEntry class that takes the EventLogType as an argument to the constructor. By deriving subclasses from the EventLogEntry additional information can be supplied in a structured way.

3.2 Event Subscribers

An event subscriber is a software module that can register into the Logger and receive event notifications. What events that should be handled by each subscriber is defined in the event log configuration file. Each subscriber is identified through the SubscriberId string that should be unique.

Logger will publish these methods that are aimed at the event publishers:

Logger.RegisterEventSubscriber(IEventSubscriber subscriber)
Registers the event subscriber in the Logger.

Logger.UnregisterEventSubscriber(IEventSubscriber subscriber)

Unregisters the event subscriber from the logger.

Logger.EventLogHandled(LogEntry, IEventSubscriber subscriber)
This method should be called by the subscriber for each event it receives when it is handled. It is then removed from the persistent or in-memory storage.

Example:

This class has a socket, and sends the category every time it receives an event.

```
partial class Class2 : IEventSubscriber
{
    public string SubscriberId
    {
        get { return "Class2"; }
    }

    public void HandleEvent(EventLogEntry eventLogEntry)
    {
        //Send the category only. This is only an example !
        socket.Send(eventLogEntry.EntityCategory);

        //Notify the logger that the event is handled and can be removed from
        //The storage.
        Logger.EventLogHandled(this, eventLogEntry);
    }
}
```

To begin receiving events, the object must be registered in the Logger

```
class OwnerClass
{
    Class2 class2;

    public void CreateSubscriber()
    {
        class2 = new Class2();
        //Register the subscriber
        Logger.RegisterEventSubcriber(class2);
    }
    public void DestroySubscriber()
    {
        //Unregister the subscriber
        Logger.UnregisterEventSubscriber(class2);
        class2.Dispose();
    }
}
```

3.3 Event log configuration

The event log is configured in the same way as the debug logging with an XML file defined by the LogConfig schema. If event logging should be enabled in the application, a separate XML configuration should be specified for the event logging.

```
Logger.SetConfigFile("MyDebugConfiguration.xml", "MyEventLogConfiguration.xml");
```

The log configuration consists of a set of filters that works the same as in the debug logging. The only difference is that an event log entry can not have any other debug level than normal, and therefore the filter types ‘Detailed’ and ‘Maximized’ Does not have any effect on the event logging.

In the outputs section the output type EventLogSubscription is used instead of the TextFileLogWriter. Thus the parameters to the output should be of the type EventLogSubscriptionParams. In these parameters, you specify which subscribers that are subscribing on the events that match the filter. For each subscriber the storage type can be set. That is about how the event should be handled in case the subscriber is not registered, or it does not properly handle the events.

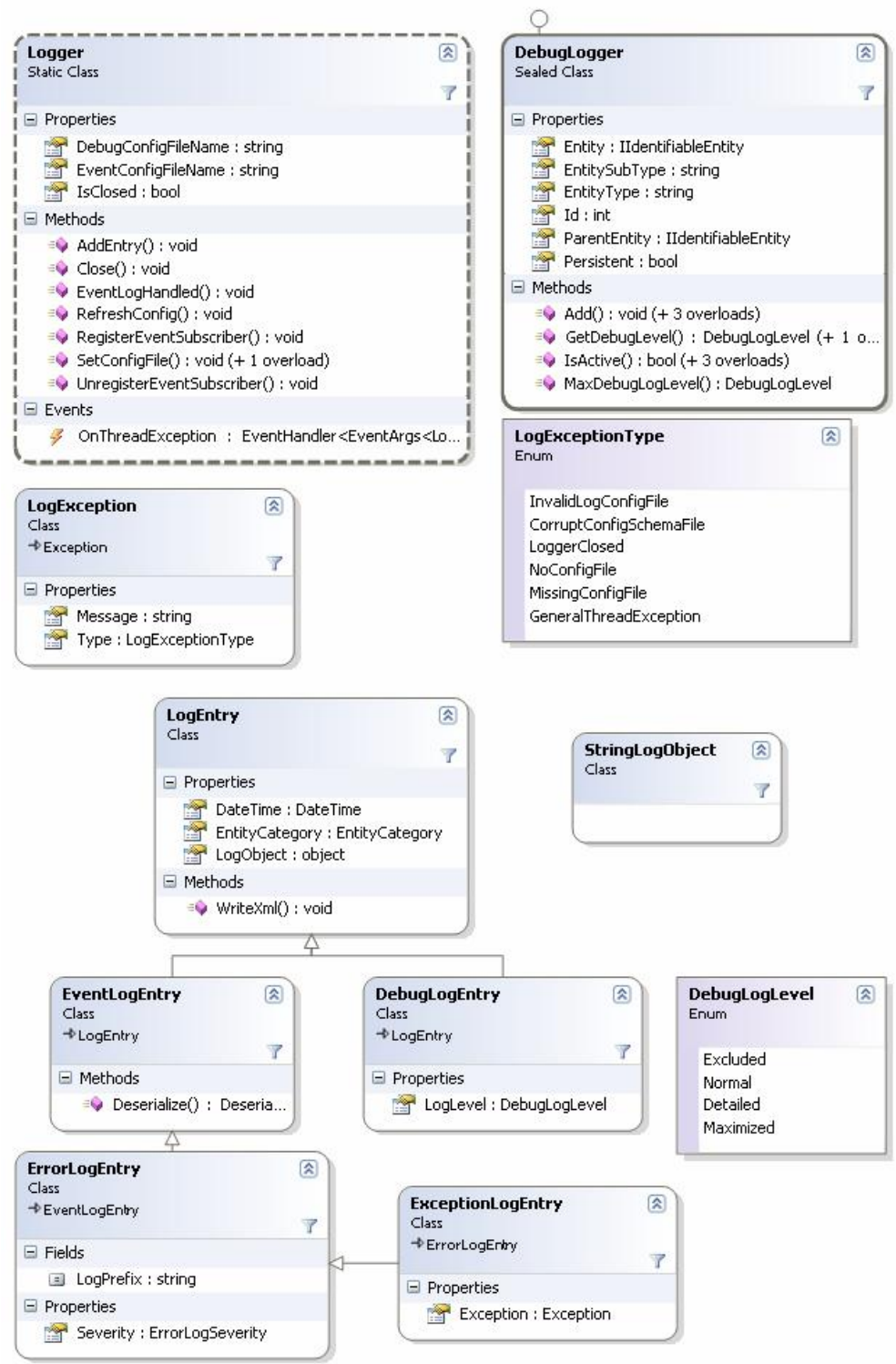
NoStorage – The events will only be notified to the registered event subscribers, and it is not stored any information in case of shutdown or power failure.

Use for non-critical events like measurements or other not so important events.

InMemory – If the subscriber is not registered or the subscriber does not set the events as handled, the events will be stored in the memory until the program closes. After that they are lost. To be safe not to fill the memory with events there is a max limit set to how much may be stored in memory before it starts to delete the events like a circular buffer.

RestartSafe – Each unhandled event is stored in restart-safe memory (disk or database) until it is handled by the subscriber. Use for critical events like alarms.

4. Diagrams


Logger
Static Class
Properties
DebugConfigFileName : string
EventConfigFileName : string
IsClosed : bool
Methods
AddEntry() : void
Close() : void
EventLogHandled() : void
RefreshConfig() : void
RegisterEventSubscriber() : void
SetConfigFile() : void (+ 1 overload)
UnregisterEventSubscriber() : void
Events
OnThreadException : EventHandler<EventArgs<Lo...
DebugLogger
Sealed Class
Properties
Entity : IIdentifiableEntity
EntitySubType : string
EntityType : string
Id : int
ParentEntity : IIdentifiableEntity
Persistent : bool
Methods
Add() : void (+ 3 overloads)
GetDebugLevel() : DebugLogLevel (+ 1 o...
IsActive() : bool (+ 3 overloads)
MaxDebugLogLevel() : DebugLogLevel
LogExceptionType
Enum
InvalidLogConfigFile
CorruptConfigSchemaFile
LoggerClosed
NoConfigFile
MissingConfigFile
GeneralThreadException
LogException
Class
Exception
Properties
Message : string
Type : LogExceptionType
LogEntry
Class
Properties
DateTime : DateTime
EntityCategory : EntityCategory
LogObject : object
Methods
WriteXml() : void
StringLogObject
Class
EventLogEntry
Class
LogEntry
Methods
Deserialize() : Deseria...
DebugLogEntry
Class
LogEntry
Properties
LogLevel : DebugLogLevel
DebugLogLevel
Enum
Excluded
Normal
Detailed
Maximized
ErrorLogEntry
Class
EventLogEntry
Fields
LogPrefix : string
Properties
Severity : ErrorLogSeverity
ExceptionLogEntry
Class
ErrorLogEntry
Properties
Exception : Exception

5. Namespace Wayne.Lib.Log

Interfaces

IEventSubscriber	Interface that event subscribers should implement.
IExternalLogWriter	Interface to an external log writer

Classes

DebugLogEntry	A Debug LogEntry.
DebugLogger	Class used to make debug logs.
DeserializedLogEntry	An EventLog entry that has been deserialized from a serialized form. The additional data that is supplied with the data is now only accessible as an Xml element in the LogDataElement property.
EntityCategory	This class wraps an IIdentifiableEntity and a Category to be used as a key in e.g. Dictionaries and Lists.
ErrorLogEntry	Base LogEntry for Errors.
EventLogEntry	LogEntry for Events.
ExceptionLogEntry	Log entry for exception errors.
LogEntry	An generic entry to be logged containing details regarding what to log, the datetime and who was performing the logging etc. This class is inherited by DebugLogEntry, EventLogEntry and ErrorLogEntry which adds on more specific properties.
LogException	General log exception.
Logger	Logger is a static class used to create log objects.
LogTextWritingParameters	
StringLogObject	The StringLogObject-class serves as a helpclass to convert one or more objects into one or more strings to log. Also provides format abilities.

Enumerations

DebugLogLevel	Describes the level of the debug information. Can be used to reduce the size of the logs.
DefaultErrorCategory	This is the default set up of error categories to be used when logging ErrorLogEntries.
EntityLogKind	In detail of the name of an identifiable entity.
ErrorLogSeverity	Describes the severity of an error.
LogExceptionType	The different types of log exceptions.

5.1 Interfaces

5.1.1. Interface IEventSubscriber

```
public interface IEventSubscriber
```

Summary
Interface that event subscribers should implement.
Properties

SubscriberId `string`	R	Identifies the subscriber. This name is used in the configuration to identify the receiver of the events.

Methods

HandleEvent `public void HandleEvent(Lib.Log.EventLogEntry eventLogEntry);` Called when an event should be handled by this subscriber.	
eventLogEntry	

5.1.2. Interface IExternalLogWriter

```
public interface IExternalLogWriter
```

Summary
Interface to an external log writer
Properties

Active `bool`	R	Tells whether the log writer is currently active.
ExternalLogName `string`	R	Identifies the external log writer used in the configuration.
ExternalLogType `string`	R	Identifies the type of external log writer used in the configuration.

Methods

Log `public void Log(Lib.Log.LogEntry logEntry, string formattedText);` Called when a log entry should be handled by this external log writer.	
logEntry	The LogEntry to log.
formattedText	The LogEntry as a formatted string.

5.2 Classes

5.2.1. Class DebugLogEntry

```
public class DebugLogEntry : LogEntry
```

Summary
A Debug LogEntry.

Properties

LogLevel `Lib.Log.DebugLogLevel`	R	The log level.

Constructors

`public DebugLogEntry(Lib.IIdentifiableEntity entity, object logObject);` Constructor.	
entity	The entity that performed the logging.
logObject	The object to log.

`public DebugLogEntry(Lib.IIdentifiableEntity entity, object logObject, Lib.Log.DebugLogLevel logLevel);` Constructor.	
entity	The entity that performed the logging.
logObject	The object to log.
logLevel	The log level.

`public DebugLogEntry(Lib.IIdentifiableEntity entity, object logObject, object category);` Constructor.	
entity	The entity that performed the logging.
logObject	The object to log.
category	The category of the log object.

`public DebugLogEntry(Lib.IIdentifiableEntity entity, object logObject, object category, Lib.Log.DebugLogLevel logLevel);` Constructor.	
entity	The entity that performed the logging.
logObject	The object to log.
category	The category of the log object.
logLevel	The log level.

5.2.2. Class DebugLogger

```
public class DebugLogger : Object
```

Summary
Class used to make debug logs.
Example
This is an example of how to write a debug log entry.

```
using (DebugLogger dLog = new DebugLogger(this))
{
    if (dLog.IsActive(DebugLogLevel.Detailed))
```

```
    {
        dLog.Add("This is line 1.", DebugLogLevel.Detailed);
        dLog.Add("This is line 2.", "MyCategory",
DebugLogLevel.Detailed);
    }
}
```

Properties

Entity `Lib.IIdentifiableEntity`	R	The identifiable entity that has created this debug log.
EntitySubType `string`	R	The EntitySubType of the Entity.
EntityType `string`	R	The EntityType of the Entity.
Id `int`	R	The Id of the Entity.
ParentEntity `Lib.IIdentifiableEntity`	R	The ParentEntity of the Entity.
Persistent `bool`	R	Tells whether the debug log is persistent or not.

Constructors

`public DebugLogger(Lib.IIdentifiableEntity entity);` Construction of non-persistent DebugLogger.	
entity	

`public DebugLogger(Lib.IIdentifiableEntity entity, bool persistent);` Construction	
entity	
persistent	

Methods

Add `public void Add(object obj);` Adds a new object to the debug log entry.	
obj	The log object that are added.

Add `public void Add(object obj, Lib.Log.DebugLogLevel level);` Adds a new object to the debug log entry.	
obj	The log object that are added.
level	TDB

Add `public void Add(object obj, object category);`

Adds a new object to the debug log entry.	
obj	The log object that are added.
category	A specific category that this log is about.

Add
```
public void Add(object obj, object category, Lib.Log.DebugLogLevel level);
```
Adds a new object to the debug log entry.

obj	The log object that are added.
category	A specific category that this log is about.
level	TDB

Dispose
```
public void Dispose();
```
Dispose.

GetDebugLevel
```
public Lib.Log.DebugLogLevel GetDebugLevel();
```
Get the current debug level for the default category.

GetDebugLevel
```
public Lib.Log.DebugLogLevel GetDebugLevel(object category);
```
Get the current debug level for the given category.

category	

IsActive
```
public bool IsActive();
```
Tells whether the default category is active in the Normal level.

IsActive
```
public bool IsActive(object category);
```
Tells whether the given category is active in the Normal level.

category	

IsActive
```
public bool IsActive(Lib.Log.DebugLogLevel debugLogLevel);
```
Tells whether the default category is active in the given level.

debugLogLevel	

IsActive
```
public bool IsActive(object category, Lib.Log.DebugLogLevel debugLogLevel);
```
Tells whether the given category is active in the given level.

category	
debugLogLevel	

MaxDebugLogLevel
`public Lib.Log.DebugLogLevel MaxDebugLogLevel(Lib.Log.DebugLogLevel level1, Lib.Log.DebugLogLevel level2);`
Static method to get the highest of two DebugLogLevel's.

level1	
level2	

5.2.3. Class DeserializedLogEntry

```
public class DeserializedLogEntry : EventLogEntry
```

Summary
An EventLog entry that has been deserialized from a serialized form. The additional data that is supplied with the data is now only accessible as an Xml element in the LogDataElement property.
Properties

LogDataElement `Xml.XmlElement`	R	Xml element that contains the additional data that was supplied with the event originally.

5.2.4. Class EntityCategory

```
public class EntityCategory : Object
```

Summary
This class wraps an IIdentifiableEntity and a Category to be used as a key in e.g. Dictionaries and Lists.
Properties

CategoryString `string`	R	The category.
Entity `Lib.IIdentifiableEntity`	R	The entity.
LastTouched `DateTime`	R	A date time that specifies when the object was last touched.

Methods

Equals
`public bool Equals(object obj);`
Equals

obj	

Equals
`public bool Equals(Lib.IIdentifiableEntity entity, object category);`
Equals

entity	
category	

GetHashCode
```
public int GetHashCode();
```
GetHashCode

GetName
```
public string GetName(Lib.Log.EntityLogKind entityLogKind, bool
suppressCategory);
```
Get the log-name.

entityLogKind	In which detail the id-entity should be presented.
suppressCategory	Should the category be suppressed or not.

Touch
```
public void Touch();
```
Touch the entity category.

5.2.5. Class ErrorLogEntry

```
public class ErrorLogEntry : EventLogEntry
```

Summary
Base LogEntry for Errors.

Fields

LogPrefix `string`	The keyword "***ERROR" put in the log file.

Properties

Severity `Lib.Log.ErrorLogSeverity`	R	The severity of the error.

Constructors

```
public ErrorLogEntry(Lib.IIdentifiableEntity entity, Lib.Log.ErrorLogSeverity
severity, object logObject);
```
Constructor.

entity	
severity	
logObject	The object to log.

```
public ErrorLogEntry(Lib.IIdentifiableEntity entity, Lib.Log.ErrorLogSeverity
severity, object logObject, object category);
```
Constructor.

entity	

severity	
logObject	The object to log.
category	

5.2.6. Class EventLogEntry

```
public class EventLogEntry : LogEntry
```

Summary
LogEntry for Events.
Constructors

`public EventLogEntry(Lib.IIdentifiableEntity entity, object logObject);` Constructor.	
entity	
logObject	The object to log.

`public EventLogEntry(Lib.IIdentifiableEntity entity, object logObject, object category);` Constructor.	
entity	
logObject	The object to log.
category	

`public EventLogEntry(Xml.XmlElement logEntryNode);` Constructor.	
logEntryNode	XML node.

Methods

Deserialize `public Lib.Log.DeserializedLogEntry Deserialize(Xml.XmlElement xmlElement);` Deserializing from an XML-element.	
xmlElement	

5.2.7. Class ExceptionLogEntry

```
public class ExceptionLogEntry : ErrorLogEntry
```

Summary
Log entry for exception errors.
Properties

Exception	R	Exception information.

`Exception`		

Constructors

`public ExceptionLogEntry(Lib.IIdentifiableEntity entity, Lib.Log.ErrorLogSeverity severity, object logObject, Exception exception);` Constructor.
entity
severity
logObject — The object to log.
exception

`public ExceptionLogEntry(Lib.IIdentifiableEntity entity, Lib.Log.ErrorLogSeverity severity, object logObject, object category, Exception exception);` Constructor.
entity
severity
logObject — The object to log.
category
exception

5.2.8. Class LogEntry

```
public class LogEntry : Object
```

Summary
An generic entry to be logged containing details regarding what to log, the datetime and who was performing the logging etc. This class is inherited by DebugLogEntry, EventLogEntry and ErrorLogEntry which adds on more specific properties.

Properties

DateTime `DateTime`	R	The date time of the logging.
EntityCategory `Lib.Log.EntityCategory`	R	The EntityCategory that performed the logging.
LogObject `object`	R	The object to log.

Constructors

`familyorassembly LogEntry(Xml.XmlElement logEntryNode);` Deserialization constructor.
logEntryNode

Methods

WriteLogObjectData `protected void WriteLogObjectData(Xml.XmlWriter xmlWriter);`	
xmlWriter	

WriteXml `public void WriteXml(Xml.XmlWriter xmlWriter, string prefix);` Serializes this object into the specified xmlWriter.	
xmlWriter	
prefix	

5.2.9. Class LogException

```
public class LogException : Exception
```

Summary
General log exception.
Properties

LogExceptionType `Lib.Log.LogExceptionType`	R	The type of exception.
Message `string`	R	The Message

Constructors

`public LogException(Lib.Log.LogExceptionType type);` Construction.	
type	

`public LogException(Lib.Log.LogExceptionType type, string message);` Construction.	
type	
message	

`public LogException(Lib.Log.LogExceptionType type, string message, Exception inner);` Construction.	
type	
message	
inner	

Methods

5.2.10. Class Logger

```
abstract public class Logger : Object
```

Summary
Logger is a static class used to create log objects.

Properties

DebugConfigFileName `string`	R	The current debug log configuration file.
EventConfigFileName `string`	R	The current event log configuration file.
IsClosed `bool`	R	Tells whether someone has called the Close() method.

Methods

AddEntry
```
public void AddEntry(Lib.Log.LogEntry logEntry);
```
Logs the given LogEntry.

logEntry	The LogEntry to log.

Close
```
public void Close();
```
Closes the logger. This should be done as the last things before the application terminates.

EventLogHandled
```
public void EventLogHandled(Lib.Log.IEventSubscriber eventSubscriber,
Lib.Log.EventLogEntry eventLogEntry);
```
Remove an event log from the storage. This method should be called from a registered IEventSubscriber when it has handled an event.

eventSubscriber	
eventLogEntry	

GetExternalLoggerParameters
```
public bool GetExternalLoggerParameters(Lib.Log.IExternalLogWriter
externalLogWriter, Lib.Log.LogTextWritingParameters@ writingParameters);
```

externalLogWriter	
writingParameters	

RefreshConfig
```
public void RefreshConfig();
```
Re-loads the configuration for the logging.

RegisterEventSubscriber
```
public void RegisterEventSubscriber(Lib.Log.IEventSubscriber eventSubscriber);
```

Register an IEventSubscriber, so it can start receiving events. The event subscriber will be sent the pending events that has been stored since the subscriber was registered the last time.	
eventSubscriber	

RegisterExternalLogger
```
public void RegisterExternalLogger(Lib.Log.IExternalLogWriter externalLogWriter);
```
Register an ExternalLogWriter.

externalLogWriter	The external log writer to register.

SetConfigFile
```
public void SetConfigFile(string debugConfigFileName, string
eventConfigFileName);
```
Reloads the configuration from the specified configuration file.

debugConfigFileName	Log configuration for the debug logging.
eventConfigFileName	Log configuration for the event logging.

SetConfigFile
```
public void SetConfigFile(string debugConfigFileName);
```
Reloads the configuration from the specified configuration file for the debug logging. To activate the event logging SetConfigFile(string,string) should be called.

debugConfigFileName	Log configuration for the debug logging.

UnregisterEntity
```
public void UnregisterEntity(Lib.IIdentifiableEntity entity);
```
Removes the specified entity from the internal filter buffers.

entity	

UnregisterEventSubscriber
```
public void UnregisterEventSubscriber(Lib.Log.IEventSubscriber eventSubscriber);
```
Unregister a registered IEventSubscriber.

eventSubscriber	

UnregisterExternalLogger
```
public void UnregisterExternalLogger(Lib.Log.IExternalLogWriter
externalLogWriter);
```
Unregister an ExternalLogWriter.

externalLogWriter	The external log writer to unregister.

Events

OnThreadException
```
public EventHandler{Wayne.Lib.EventArgs{Wayne.Lib.Log.LogException}}
OnThreadException;
```
An event that is fired when the logging thread is catching an exception.

5.2.11. Class LogTextWritingParameters

```
public class LogTextWritingParameters : Object
```

Summary

Properties

DateTimeFormat `string`	R	
EntityLogKind `Lib.Log.EntityLogKind`	R	
SuppressCategory `bool`	R	

Constructors

`public LogTextWritingParameters(string dateTimeFormat, Lib.Log.EntityLogKind entityLogKind, bool suppressCategory);`	
dateTimeFormat	
entityLogKind	
suppressCategory	

5.2.12. Class StringLogObject

```
public class StringLogObject : Object
```

Summary

The StringLogObject-class serves as a helpclass to convert one or more objects into one or more strings to log. Also provides format abilities.

Constructors

`public StringLogObject(Object[] logObjects);` Constructor.	
logObjects	A number of objects to log.

`public StringLogObject(string format, IFormatProvider provider, Array array);` Constructor.	
format	A format-string to format the items of an array.
provider	An IFormatProvider to format the items of an array.
array	An array of objects to log.

`public StringLogObject(string format, IFormatProvider provider, Object[] logObjects);` Constructor.	
format	A format-string to format the items of an array.

provider	An IFormatProvider to format the items of an array.
logObjects	A number of objects to log.

5.3 Enumerations

5.3.1. Enumeration DebugLogLevel

Summary
Describes the level of the debug information. Can be used to reduce the size of the logs.
Fields

Excluded	Not logged.
Normal	Normal debug information.
Detailed	Detailed debug information.
Maximized	Maximized debug information.

5.3.2. Enumeration DefaultErrorCategory

Summary
This is the default set up of error categories to be used when logging ErrorLogEntries.
Fields

Bug	Whoops. Our mistake. A bug.
Configurational	This error is due to a badly configured system.
Communication	This error occurred as a result of some communication problems.
Peripheral	Some kind of Peripheral equipment failed in some way.
UnexpectedResult	The program got an unexpected result from some kind of operation. This could for instance be a computed value that is out of the allowed range.
XmlValidation	Invalid XML data.

5.3.3. Enumeration EntityLogKind

Summary
In detail of the name of an identifiable entity.
Fields

None	No name.
Entity	Only the identifiable entity itself (no parents).
Ancestors	The names of identifiable entitis all the way from the root entity to the current entity.

5.3.4. Enumeration ErrorLogSeverity

Summary
Describes the severity of an error.
Fields

Cosmetic	The kindest type of error. The application can proceed its execution without

	any problem.
Recoverable	Quite a bad error has occurred. The application can however continue without any loss of data or similar.
RecoverableDataLoss	This is a really bad error. Somehow some kind of data is lost -- but the application can continue its execution.
Irrecoverable	The worst imaginable errors. When this error has occurred the application cannot continue. This could for instance be a bad configuration, e.g. two servers listening to the same port.

5.3.5. Enumeration LogExceptionType

Summary
The different types of log exceptions.
Fields

InvalidLogConfigFile	The log config file has a bad format.
CorruptConfigSchemaFile	The internal config schema file is corrupt.
LoggerClosed	This operation is not allowed since the Logger is closed.
NoConfigFile	There is no configuration file specified.
MissingConfigFile	The specified configuration file is missing.
GeneralThreadException	When an exception has occurred within the thread's execution method, the Logger will fire an OnThreadException holding this exception.