|  |  |
| --- | --- |
| February 3, 2011 |  |
|  | Security Framework |
| Version 1.0 |  |
| Logger user’s guide | |



Table of Contents

1 Introduction 1

2 Logging concept 1

2.1 Logging levels 1

2.2 What and Where to log 2

3 Configuration file structure 2

3.1 Errors logger configuration reference 2

3.1.1 Attributes 3

3.1.2 Remarks 3

3.2 Errors logger configuration file structure 4

3.2.1 LoggerClassConfiguration 4

3.2.2 Configuration 5

3.2.3 Listeners 6

3.2.4 Formatters 8

3.2.5 ExceptionPolicyConfiguration 9

3.2.6 CategorySources 12

3.2.7 Special (predefined) Category Sources 13

# Introduction

Logger allows saving debug information according to the developers needs and information about errors, attacks, harmful and potentially harmful activities and situations happening in the Security Framework for debugging, testing and further support purposes. There is a set of logging frameworks and tools available for .NET applications. Most popular free loggers are:

1. [Log4Net](http://logging.apache.org/log4net/).
2. [MS Enterprise Library Logging and Exception handling blocks](http://msdn.microsoft.com/en-us/library/ff632023.aspx).
3. [NLog](http://nlog.codeplex.com/).

In the current Security Framework implementation MS Enterprise Library Logging and Exception handling application blocks are used.

# Logging concept

Loggers help to monitor the application`s state on different stages of its lifecycle. There are different target groups for log messages and different kinds of logging situations. Log writers are often divided into different levels. Such approach easies future log processing and allows to turn on/off specified levels whether it is needed (some loggers allow to do this operation even “in-flight” without of the application restart).

There should be clear for what purposes would be the log message used: who, when and how would process it. Log messages should specify what and when has been happened. For all log information there should be possibility to process it (automated or manual). There must not be any non-processible information in the log. All codes, IDs, etc. should be specified so the users could understand there meaning.

For security purposes some log messages (for example user-relevant information) could be stored in the encrypted form.

## Logging levels

There are 5 main logging types: Fatal, Error, Warning, Info, Debug.

| Level | Content | When the logger is ON | Term of use |
| --- | --- | --- | --- |
| Fatal | Critical (could lead to the application crash) attacks, errors, activities and situations. | Always | Long (years) |
| Error | Attacks, errors, harmful activities and situations. | Always | Long (years) |
| Warning | Possible attacks, minor exceptions, potentially harmful activities and situations. | Almost | Middle (months) |
| Info | Application status messages. | Almost | Short (hours or days) |
| Debug | Developer-relevant info that could be useful for debugging. | Sometimes | Short (hours or days) |

## What and Where to log

**Fatal** and **Error** log writers are almost placed where attacks, errors, harmful activities and situations are processed (usually at the “catch” block). The log message should contain at least situation description: what error or kind of attack appeared, what exception has been raised. Also there should be information about the client which actions have led to such situation. All the potentially dangerous code must be wrapped by Error (almost) and Fatal log writers.

**Warning** logs are not so widely used as Error logs and often instead of them. Sometimes there are error-like situations which should not be written to Error log: possible attack attempts (but not enough information to say that it was an attack), some discovered and well-wrapped non-fixable exceptions (for example from 3rd-party libs) which are raised at time to time.

**Info** logs should contain only coarse-grained information about the progress of the application in the production environment. They shouldn’t be used for too small activities because of performance issues and future log processing.

Info log writers are often placed at the start and of the end of main activities. In this case log messages contain initial information (for example method parameters) for start point and/or result information (result of some calculations and so on); also messages should contain activity signature. There are two main approaches for such logging: direct log calls from the method and aspect-oriented approach (where logger is realized as aspect and inserted to the start and/or the end of the method using “pointcuts”). Also Info logs could contain information about resource usage (resource capture and release, file creation/deleting/copying).

For Security Framework Info logs should be used at least for the operations with the configuration files and for initialization and shutting down of the SecurityKernel.

**Debug** logs could contain any information that could help developers to debug the application. It could contain useless information for non-developers; it is possible to use it in the code whether and how often it is needed. It is important to remember that debug logs are often turned off on the production so information for further system support should not be placed there.

# Configuration file structure

Configuration is a set of XML documents. Each Security Framework subsystem has a separate configuration file. There is a root configuration file - MtSfConfigurationLoader.xml that references all subsystem’s configuration files.

## Errors logger configuration reference

The root configuration file references the Errors logger configuration and Common Security Framework configuration (which is important for Errors logger) as it’s shown below:

<ConfigurationLoader>

<Items>

|  |
| --- |
| <item Id="162212C5-8ABC-40F2-9E36-BEEFC6739EE4" IsServiceSubsystem="true" IsEnabled="true" Name="LoggerClassConfiguration" Path="MtSfLogging.xml" Type="MetraTech.SecurityFramework.Core.Common.Logging.Configuration.LoggerClassConfiguration"/> |

### Attributes

| Attribute | Description |
| --- | --- |
| Id | Required attribute.  Defines a unique item ID. Uses internally.  Value: must be GUID. |
| IsServiceSubsystem | Optional attribute.  Indicates that a subsystem is important for other subsystems. Uses internally.  Values: true | false.  It should be set to “true” for Errors logger. |
| IsEnabled | Required attribute.  Values: true | false.  Indicates whether the subsystem is enabled. Uses for turning the subsystem off when it is not needed. |
| Name | Required attribute.  Defines the subsystem name. Uses internally. |
| Path | Required attribute.  Defines a path to the subsystem configuration file. This path is relative to the root configuration file. I.e., when the subsystem configuration file is placed to the same folder the root configuration file is, simply provide a file name only. |
| SubsystemType | Required attribute for the Security Monitor configuration reference.  Defines a type that provides an access to the subsystem functionality. This value has not to be changed. |
| Type | Required attribute.  Defines a type that takes the subsystem properties when it is loaded from the configuration file. This value has not to be changed. |

### Remarks

The really important attributes are the Path and IsEnabled. First of them indicates a file with the configuration and second allows turning the subsystem off. You may want to turn the Security Monitor off if it causes performance problems, for example.

## Errors logger configuration file structure

The Errors logger configuration file structure at high level is shown below:

|  |
| --- |
| <LoggerClassConfiguration typeName="MetraTech.SecurityFramework.Core.Common.Logging.EntLibErrorLogger, MetraTech.SecurityFramework">  <Configuration>  <Listeners>  <Item >  </Listeners>  <Formatters>  <Item>  </Formatters>  <ExceptionPolicyConfiguration>  <ExceptionTypes>  <Item>  <ExceptionHandlers>  <Item>  </ExceptionHandlers>  </Item>  <Item>  </ExceptionTypes>  </ExceptionPolicyConfiguration>  <CategorySources>  <Item>  </CategorySources>  <AllEvents>  <listeners>  <Item>  </listeners>  </AllEvents>  <NotProcessed>  <listeners>  <Item>  </listeners>  </NotProcessed>  <Errors>  <listeners>  <Item>  </listeners>  </Errors>  </Configuration> |

The following sections describe attributes, elements, and child elements.

### LoggerClassConfiguration

Root element. It’s not required but recommended do not change the element’s name. It matches with name of the class that takes the configuration properties when it is loading.

The root element has only one attribute:

| Attribute | Description |
| --- | --- |
| typeName | Required attribute.  Defines a logger type name. |

Actually there is only one errors logger. Its type name is “MetraTech.SecurityFramework.Core.Common.Logging.EntLibErrorLogger, MetraTech.SecurityFramework”.

Sections below describe this logger configuration.

### Configuration

Defines the entire logger configuration.

<LoggerClassConfiguration>

|  |
| --- |
| <Configuration tracingEnabled="True|False" defaultCategory="<Configured\_category\_name>"> |

The element has the following attributes and elements.

Attributes:

| Attribute | Description |
| --- | --- |
| tracingEnabled | Optional attribute.  Defines whether the same listeners will be used for .NET tracing functionality (via Trace class).  Values: true | false. |
| defaultCategory | Required attribute.  Defines a default category source for logged entries.  Value: must be any of configured category source names (see section CategorySources for description). |

Elements:

| Element | Description |
| --- | --- |
| Listeners | Required element.  Defines a collection of trace listeners to be used for processing/recording logged entries. |
| Formatters | Required element.  Defines a collection of formatters to format a logged entry to some human readable representation. |
| ExceptionPolicyConfiguration | Required element.  Defines a collection of exception processing rules. You can define different rules for each exception’s type separately. |
| CategorySources | Required element.  Defines a collection of custom category sources. At least one category source has to be defined here. |
| AllEvents | Required element.  Contains setting for predefined category source – All Events. It’s not recommended to change the default configuration for this element. Uses internally. |
| NotProcessed | Required element.  Contains setting for predefined category source – Not Processed. It’s not recommended to change the default configuration for this element. Uses internally. |
| Errors | Required element.  Contains setting for predefined category source – Errors. It’s not recommended to change the default configuration for this element. Uses internally. |

### ­Listeners

Defines a collection of trace listeners to be used for processing/recording logged entries.

Actually there is only one available listener type in Security Framework:

<LoggerClassConfiguration>

<Configuration>

|  |
| --- |
| <Listeners>  <Item RealType="MetraTech.SecurityFramework.Core.Common.Logging.Configuration.FlatFileTraceListenerConfiguration" name="<Listener\_name>" listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" fileName="<Absolute\_file\_path>" formatter="<Formatter\_name>" traceOutputOptions="None|LogicalOperationStack|DateTime|Timestamp|ProcessId|ThreadId|Callstack" filter="Critical|Error|Warning|Information|Verbose" />  </Listeners> |

The listener’s configuration has the following attributes.

| Attribute | Description |
| --- | --- |
| RealType | Required attribute.  Defines a type holding the listener’s configuration after reading from the file.  Value: must be “MetraTech.SecurityFramework.Core.Common.Logging.Configuration.FlatFileTraceListenerConfiguration” |
| Name | Required attribute.  Defines a name of the listener. Used by category sources to reference a listener.  Value: must be unique within the collection. |
| ListenerDataType | Required attribute.  Defines a type to create and configure a trace listener.  Value: must be “Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35”. |
| FileName | Required attribute.  Defines a full path to the errors log file.  **The path should be absolute. The directory must exist and the account the application is ran under must have Modify permissions for this directory.** |
| Formatter | Required attribute.  Defines a formatter to be used by the trace listener.  Value: must be any of formatter names configured in the Formatters section (see below). |
| TraceOutputOptions | Optional attribute.  Determines the optional content of trace output. The attribute can contain several values separated by comma. This attribute’s value affect  Values:   * LogicalOperationStack - Write the logical operation stack, which is represented by the return value of the CorrelationManager.LogicalOperationStack property * DateTime - Write the date and time. * Timestamp - Write the timestamp, which is represented by the return value of the GetTimestamp method. * ProcessId - Write the process identity, which is represented by the return value of the Process.Id property. * ThreadId - Write the thread identity, which is represented by the return value of the Thread.ManagedThreadId property for the current thread. * Callstack - Write the call stack, which is represented by the return value of the Environment.StackTrace property. |
| Filter | Required attribute.  Defines a source level to be used as a filter for logged entries. Only entries their level is equal or higher will be logged.  Values:   * Critical - Fatal error or application crash. * Error - Recoverable error. * Warning - Informational message. * Information - Non-critical problem. * Verbose - Debugging trace. |

### Formatters

Defines a collection of text formatters to be used by trace listeners.

Actually there is only one formatter type available in Security Framework:

<LoggerClassConfiguration>

<Configuration>

|  |
| --- |
| <Formatters>  <Item typeName="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"  template="Timestamp: {timestamp}{newline}&#xA;Message: {message}{newline}&#xA;Category: {category}{newline}&#xA;Priority: {priority}{newline}&#xA;EventId: {eventid}{newline}&#xA;Severity: {severity}{newline}&#xA;Title:{title}{newline}&#xA;Machine: {localMachine}{newline}&#xA;App Domain: {localAppDomain}{newline}&#xA;ProcessId: {localProcessId}{newline}&#xA;Process Name: {localProcessName}{newline}&#xA;Thread Name: {threadName}{newline}&#xA;Win32 ThreadId:{win32ThreadId}{newline}&#xA;Extended Properties: {dictionary({key} - {value}{newline})}"  name="<Formatter\_name>" />  </Formatters> |

The formatter’s configuration has the following attributes:

| Attribute | Description |
| --- | --- |
| TypeName | Required attribute.  Defines a formatter type name.  Value: must be “Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35” |
| Template | Required attribute.  Defines a template to format text by. This value does not affect Security Framework errors log and uses for .NET tracing capabilities. |
| Name | Required attribute.  Defines a name of the formatter. Used by listeners to reference a formatter.  Value: must be unique within the collection. |

### ExceptionPolicyConfiguration

Defines a policy of exception processing. Allows to send logging entries to the proper destination and skip some of exception types.

<LoggerClassConfiguration>

<Configuration>

|  |
| --- |
| <ExceptionPolicyConfiguration PolicyName="<Policy\_name>">  <ExceptionTypes>  <Item name="<Item\_name>" typeName="System.Exception, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"  PostHandlingAction="None|NotifyRethrow|ThrowNewException">  <ExceptionHandlers>  <Item name="<Exception\_handler\_name>" typeName="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"  logCategory="<Category\_source\_name>" eventId="<Event\_Id>" severity="Critical|Error|Warning|Information|Verbose" title="<Handler\_title>"  formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.TextExceptionFormatter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling"  priority="0" />  </ExceptionHandlers>  </Item>  <Item name="BadInputDataException" typeName="MetraTech.SecurityFramework.BadInputDataException, MetraTech.SecurityFramework, Version=1.0.0.0, Culture=neutral, PublicKeyToken=a0f5c26dbec45a75"  PostHandlingAction="None|NotifyRethrow|ThrowNewException"></Item>  <Item name="ThreadAbortException" typeName="System.Threading.ThreadAbortException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"  postHandlingAction="NotifyRethrow" />  </ExceptionTypes>  </ExceptionPolicyConfiguration> |

The element has only one attribute:

| Attribute | Description |
| --- | --- |
| PolicyName | Required attribute.  Defines a name of the exception handling policy.  Value: text. |

The element contains a collection ExceptionTypes that defines a processing behavior for each exception type separately.

The collection’s items specify a policy for each of exception types. Derived exception types are processed like a base type until another policy provided for a concrete type.

Exception type definition has the following attributes and elements.

Attributes:

| Attribute | Description |
| --- | --- |
| Name | Required attribute.  Defines a name of exception type definition. Uses internally.  Value: must be unique within the collection. |
| TypeName | Required attribute.  Defines an exception type.  Value: fully or assembly qualified exception type name. |
| PostHandlingAction | Optional attribute.  Defines an action to be performed after the exception processing. *For future use.*  Value:  None – No action should be performed.  NotifyRethrow – An original exception should be re-thrown.  ThrowNewException – An original exception should be wrapped and re-thrown. |

Elements:

| Element | Description |
| --- | --- |
| ExceptionHandlers | Optional element.  Defines a collection of handlers for the specified exception type.  **Omit (or remove) this element to prevent exceptions of the specified type from falling into the errors log.** |

A handler definition has the following attributes:

| Attribute | Description |
| --- | --- |
| Name | Required attribute.  Defines a name of exception handler definition. Uses internally.  Value: must be unique within the collection. |
| TypeName | Required attribute.  Defines a type of the exception handler. Actually only the logging exception handler is available in the Security Framework.  Value: must be “Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=5.0.414.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35”. |
| LogCategory | Required attribute.  Defines a category source to use trace listeners from to handle (record) exceptions’ data.  Value: one of predefined category sources (AllEvents, Errors, NotProcessed) or any of configured custom category sources (see section CategorySources for description). |
| EventId | Required attribute.  Defines an ID for the trace entries generated from the handled exceptions.  Value: positive integer number. |
| Severity | Required attribute.  Defines a level for trace entries generated from exceptions.  Value: Critical, Error, Warning, Information, Verbose  **It should be Error by default but it’s possible to set to another value for some especial exception types.** |
| Title | Required attribute.  Defines a title for the definition. Uses internally.  Value: text |
| FormatterType | Required attribute.  Defines a formatter type. Actually only text formatter is applicable.  Value: should be “Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.TextExceptionFormatter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling” |
| Priority | Required attribute.  Defines a sequence the handlers are executed.  Value: non-negative integer number. |

By default the ExceptionTypes collection should define at least 3 exception types. It’s illustrated in the **3**.2.5 at the beginning of the section.

Base Exception type (for all exception) defines a trace listener to record all exceptions to a single log.

BadInputDataException type define a filter to exclude the Security Framework signal exceptions from the errors log. If you need to include these exceptions to the log just remove this filter.

ThreadAbortException defines a filter to exclude this type of exceptions from the errors log because it used by .NET for service purposes.

Also you can define any other filters for specific exception types.

### CategorySources

Defines a collection of custom category sources.

Category source provides a set of listeners to be used together. It is used by exception handlers to reference listeners.

<LoggerClassConfiguration>

<Configuration>

|  |
| --- |
| <CategorySources>  <Item SwitchValue="All" CategoryName="<Category\_Name>">  <Listeners>  <Item value="<Listener\_definition\_name>" />  </Listeners>  </Item>  </CategorySources> |

Category source item has the following attributes and elements.

Attributes:

| Attribute | Description |
| --- | --- |
| SwitchValue | Required attribute.  Defines a debugging and tracing switch value for the category. Used for .NET tracing capabilities.  Value: text. Usually it should be All |
| CategoryName | Required attribute.  Defines a name of the category source. Used by exception handlers to reference trace listeners.  Value: must be unique within the collection. |

Elements:

| Element | Description |
| --- | --- |
| Listeners | Optional element.  Defines a collection of trace listeners in the category. |

The listener definition has only one attribute:

| Attribute | Description |
| --- | --- |
| Name | Required attribute.  References a listener definition by its name.  Value: a name of any registered listener (see ­Listeners section for description). |

### Special (predefined) Category Sources

There are 3 special category sources: AllEvents, NotProcessed and Errors. Their definitions are the same as for custom CategorySources. But usually you need not to change those definitions.