Basic Role Logger

1. Contents

2. Introduction

I think every software engineer knows what logging is. Anyways, here is a definition from techopedia.com:Data logging is the process of collecting and storing data over a period of time in order to analyze specific trends or record the data-based events/actions of a system, network or IT environment. It enables the tracking of all interactions through which data, files or applications are stored, accessed or modified on a storage device or application.

In few words, a software developer uses data logging systems to collect different pieces of information during the execution of a given piece of software, in order to: show the user what the application does in real time, inspect the logs while tracing bugs, and much more. In other words this is the analog of a computer program keeping a diary.

Each piece of data is called a log entry, and usually has a timestamp. Also, different stuff is logged for different reasons. For example, one might log fatal errors in order to analyze them, while logging execution information to show to the user in the UI, and logging detailed debug information that can be "walked through" in case of a mysterious fatal error. This is why log entries often have different types or roles. This can be viewed also from the perspective that an amateur user will need only basic information like "Started" and "Done" for example, more advanced user will like to see a bit more, a developer - yet more, and in case of a nontrivial issue the developer might need a very detailed log. This differentiation of log entries is based on message segnificance, or also called a level of Verbosity, and is most often used in the command-line tools. This is why a log message often has a role/verbosity level, which in this paper is referred to as a log-entry type.

This is a basic in-memory logging system - fairly standard, lightweight and easy to use. Logs are collected as LogEntry objects that can be manipulated in code or cast to different kind of string representations. Also, entire logs can be retrieved as strings, based on log type or not.

Note that this snippet does not support logging to a file, trace or a standard output. This will be covered in different papers/snippets

3. Background

The code is mine, based on the functionality I've observed in different logger implementations on different projects. It is intentionally minimalistic and not a library - the software format that I'm actively trying to avoid, in favor of dynamically generated code.

By the way, If you like this snippet, you might want to check out some of my other articles and source codes on my website: https://codeprompt.github.io/basic-role-logger/basic-role-logger.htm

4. Using the code

A new instance of the logger is created and Log methods are used "LogDebug(string)", "LogInfo(string)", "LogWarning(string)", "LogError(string)" and "Log(eLogType, string)" to log stuff. Finally a text representation is retrieved as a string that can be displayed to the user or saved to a file.

Alternatively, a list of entry objects is retrieved and iterated, and the message of each object is outputted in a different color and font depending on its role.

Note that all methods and properties are thread-safe and use deep-cloning.

5. Overview

The enumeration "eLogType" enumerates the different roles a log message might have. Each log message has exactly one role.

The actual logging is done by the methods of the "Logger" class: "LogDebug(string)", "LogInfo(string)", "LogWarning(string)", "LogError(string)" and "Log(eLogType, string)". Later, "ToFullString()", "ToShortString()" or "ToString()" should be called to retrieve the logs text.

Log entries might be retrieved, when a formatted string is undesirable - like in the case when the logs will be used in some programming logic.

Also, logging might be restricted to certain types by the "LogVerbosity" and "OutputVerbosity". When set, entries with type precedence lower than the specified will be ignored. The type precedence is : "Trace" : 0, "Debug" : 1, "Info" : 2, "Warning" : 3, "Error" : 4. "LogVerbosity" restricts what is being logged, while "OutputVerbosity" restricts only what is being outputted, so if set, all entries will still be logged but not outputted, until the restriction is removed or "AllEntries" is used to retrieve all the entries.

Finally, all logs can be discarded by using "Clear()".

The "LogEntry" class represents a single logged message. It has a "Type" which represents an importance level or role of the message, a message "TimeStamp" represented by .Net "DateTime" object, and the actual message string - "Message".

6. Point Map

eLogType		
.Trace		
.Debug		
.Info		
.Warning		
.Error		
LogEntry		
.eLogType		Type
.string		Message
.DateTime		TimeStamp
_eLogType		_type
_string		_message
_DateTime		_timestamp
_(eLogType, string)		type, message
_(DateTime,eLogType, string)		timestamp, type, message
.ToFullString()	-> string	
.ToFullString(string)	-> string	dateFormat
.ToShortString()	-> string	
.ToShortString(string)	-> string	dateFormat
.ToString()	-> string	
Logger		
_List<LogEntry>		_entries
.List<LogEntry>		Entries
.List<LogEntry>		AllEntries
.eLogType		LogVerbosity
.eLogType		OutputVerbosity
.ToFullString()	-> string	
.ToFullString(string)	-> string	dateFormat
.ToShortString()	-> string	
.ToShortString(string)	-> string	dateFormat
.ToString()	-> string	
.()		RecordComplete
.(eLogType)		verbosity
.(eLogType, eLogType)		verbosityLog, verbosityOutput
.Clear()		
.LogTrace(string)		message
.LogDebug(string)		message
.LogInfo(string)		message
.LogWarning(string)		message
.LogError(string)		message
.Log(eLogType, string)		type, message

8. Member Map - LogEntry

Type
Gets the type of the log entry.

Message
Gets the actual message of the log entry.

TimeStamp
Gets the local time at the moment of logging of the entry.

Private backing fields("_type", "_message", "_timestamp") - these are the backing fields behind the properties above.
When a public (get only)property is called the corresponding backing field value is returned.
This is one of the most commonly-used (small) patterns in C#

Internal constructors are used by the "Logger" to create the entry objects.
The "LogEntry" objects are not intended to be generated outside or modified the logger,
so there are no public constructors.

_(eLogType, string)
The values set the corresponding fields.
The timestamp is taken at the moment of calling the constructor(local time)
and assigned to its corresponding field - "_timestamp".
->DateTime.Now

_(DateTime, eLogType, string)
The values set the corresponding fields.
This is used to clone the object, so the timestamp parameter.

.ToFullString() **->string**
Cast the _timeStamp to string of format "HH:mm:ss",
cast the _type to string, and concatenate with the _message field,
so the result is something like that: "17:46:29[Trace] : My logged message"
->DateTime.ToString("HH:mm:ss")

.ToFullString(string dateFormat) **->string**
Cast the _timeStamp to string of format provided in the "dateFormat" parameter,
cast the _type to string, and concatenate with the _message field,
so the result is something like that: "17:46:29[Trace] : My logged message".
The dateFormat must be acceptable to the "DateTime.ToString(string)" method.
You can look into that further here : https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings
or by searching online for something like "Custom Date and Time Format Strings .Net C#" e.g.
->DateTime.ToString(dateFormat)

.ToShortString() **->string**
Cast the _timeStamp to string of format "HH:mm:ss" nd concatenate with the _message field,
so the result is something like that: "17:46:29 : My logged message"
->DateTime.ToString("HH:mm:ss")

.ToShortString(string dateFormat) **->string**
Cast the _timeStamp to string of format provided in the "dateFormat" parameter and concatenate with the _message field,
so the result is something like that: "17:46:29 : My logged message".
The dateFormat must be acceptable to the "DateTime.ToString(string)" method.
You can look into that further here : https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings
or by searching online for something like "Custom Date and Time Format Strings .Net C#" e.g.
->DateTime.ToString(dateFormat)

.ToString() **->string**
Return the _message.

9. Member Map - Logger

The minimal Log verbosity.
Entries with lower importance will not be logged.
LogVerbosity

The minimal Output verbosity.
Entries with lower importance will be logged but will not be outputted.
OutputVerbosity

AllEntries
Gets all the logged entries from the backing storage field "_entries", regardless of verbosity: Create new empty List of LogEntry, Lock _entries and iterate through each entry creating a new entry with the same type, message and timestamp, and adding it to the list. Then return the list. ->new LogEntry(entry.TimeStamp, entry.Type, entry.Message)

Entries
Gets all the logged entries from the backing storage field "_entries":
Create new empty List of LogEntry, Lock _entries and iterate through each entry:
If the type of the current entry is with lower precedence than the current set OutputVerbosity then ignore it
(the precedence goes like that: "Trace", "Debug", "Info", "Warning" and most significant "Error"),
else create a new entry with the same type, message and timestamp, and add it to the list.
Then return the list.
->new LogEntry(entry.TimeStamp, entry.Type, entry.Message)

.ToFullString() **->string**
Get all the outputable entries by calling the "Entries" property.
Get string representation of each by calling ToFullString() and join those
with "new line" escape sequence string as a separator.
For more info check the corresponding method in Member Map - LogEntry
->entry.ToFullString()

.ToFullString(string dateFormat) **->string**
Get all the outputable entries by calling the "Entries" property. Get string representation of each by calling ToFullString(dateFormat) and join those with "new line" escape sequence string as a separator. For more info check the corresponding method in Member Map - LogEntry ->entry.ToFullString(dateFormat)

.ToShortString() **->string**
Get all the outputable entries by calling the "Entries" property.
Get string representation of each by calling ToShortString() and join those
with "new line" escape sequence string as a separator.
For more info check the corresponding method in Member Map - LogEntry
->entry.ToShortString()

.ToShortString(string dateFormat) **->string**
Get all the outputable entries by calling the "Entries" property.
Get string representation of each by calling ToShortString(dateFormat) and join those
with "new line" escape sequence string as a separator.
For more info check the corresponding method in Member Map - LogEntry
->entry.ToShortString(dateFormat)

.ToString() **->string**
Get all the outputable entries by calling the "Entries" property.
Get string representation of each by calling ToString() and join those
with "new line" escape sequence string as a separator.
For more info check the corresponding method in Member Map - LogEntry
->entry.ToString()

.()
.(eLogType verbosity)
.(eLogType verbosityLog, eLogType verbosityOutput)
Ctors. Pretty much self explanatory.

.LogTrace(string)
Log the message with type "Trace".
->Log(eLogType.Trace, message)

.LogDebug(string)
Log the message with type "Debug".
->Log(eLogType.Debug, message)

.LogInfo(string)
Log the message with type "Info".
->Log(eLogType.Info, message)

.LogWarning(string)
Log the message with type "Warning".
->Log(eLogType.Warning, message)

.LogError(string)
Log the message with type "Error".
->Log(eLogType.Error, message)

.Log(eLogType, string)
If the "LogVerbosity" is greater then "type" return. else create new "LogEntry", lock "_entries" and add it to "_entries". ->new LogEntry(type, message)

.Clear()
Remove all the stored entries.

Viktor Chernev (vchernev91@abv.bg)
https://codeprompt.github.io/basic-role-logger/basic-role-logger.htm

22 Aug 2018