# Overview

JLogger is a singleton component as a .NET Standard 2.0 library component that can be used with Core 2.0+, Standard, Framework 4.6.1+, Mono, and [Xamarin](https://docs.microsoft.com/en-us/standard/net-standard).

JLogger has these characteristics:

* Write to a file, or write to a database table (SQL Server scripts to create the table and stored procedures are included).
* Multithreaded use – As a singleton, it is accessible from any thread, and uses locking techniques to ensure there are no collisions.
* High throughput – If the log is being used by many threads concurrently, the log writes do not stop the calling thread. JLogger uses a first-in, first-out (FIFO) queue where log writes are put in a queue and written to a file in a separate thread, concurrently in the background. The WriteDebugLog command takes the parameters, creates the log data, puts it in a queue. None of those steps are blocking.
* Send an Email – A debug log write can optionally send an email (SMTP configuration data required)
* Multiple Log Entry Types – there are several log entry types to choose from. What they each mean is up to the user writing the code. Some log types are reserved for the component, and would be ignored in processing the log entry. These are detailed below.
* New Log File each Day – after midnight, a new log file is created so log files are named to show the date and time span the log was active.
* Log Retention – logs (and log entries in a database table) are automatically removed after the specified number of days, unless zero is specified, in which case no log files (or log entries in a database table) are deleted.
* Tab-delimited Log File – the log is written as a tab-delimited file. This enables opening up the file in programs like Excel for analysis.
* This file and the DB folder with the SQL Server scripts are located where the NuGet package is installed.

# LOG\_TYPE Enum

The enum contains values for the types of logs, and for how the logs are created and managed.

## LOG\_TYPE Values for Logging

* Unspecified - Used as a default value until an assignment is made.
* Flow – Used to denote a log entry that can be used to trace program flow in the log.
* Error – Used to denote serious exceptions that generally require follow-up and fixing.
* Informational – Denotes the log entry is for information only.
* Warning – Means the log entry is warning about a potentially serious condition.
* System – Log entry relates to system data.
* Performance – Log entry that usually shows elapsed time (as placed in the log message by the coder) and/or start time.
* Test – Used to indicate the log entry was intended for test results.
* SendEmail – Used in the WriteDebugLog LOG\_TYPE variable to specify that this log entry should also send an email. The email is only sent if the same flag is used in Log Management. Use of this flag here only applies to the specific log entry, not the entire log.
* Database - Log entry related to database operations
* Service - Log entry related to service operations
* Cloud - Log entry related to cloud operations
* Management - Log entry related to management concerns or operations
* Fatal - Log entry related to some fatal operation or state
* Network - Log entry related to network issue or operation
* Threat - Log entry related to a threat condition
* StartupShutdown – Used internally. Not for use by external code.

## LOG\_TYPE Values for Log Management

* ShowModuleMethodAndLineNumber – Tells JLogger to include any available values for what module name, method name, and line number the exception or log entry was made. This is very useful for finding and correcting bugs in development, quality assurance, and production code.
* ShowTimeOnly - Shows time only, not date, in the debug log. Useful since debug logs are closed and a new one created on the first write the next day after the log file was opened. Do not use this flag if you want each log entry to show date and time.
* HideThreadID - Hides the thread ID from being printed in the debug log.
* IncludeStackTrace – Writes the stack trace to the debug log. Otherwise, leaves that column blank.
* SendEmail – This is used if sending an email from a log entry that also uses SendMail. The flag, when used in management (DebugLogOptions) enables sending email if the log entry also calls for it by use of this flag. If this flag is not set, the use of SendMail in a specific log entry is ignored, and no email is sent. This allows globally turning email sends on and off by simply changing the DebugLogOptions of the Logger instance.
* IncludeExceptionData – This tells the JLogger instance to examine the Data collection on all Exceptions, and log any name-value pairs it finds there. The Exception.Data collection is often used in catch blocks to add real time values to the exception before executing a “throw”. Use of the Data collection for this saves much time in troubleshooting.

# Example Code

These lines of code are used to illustrate the use of JLogger. There are more variations than documentation can show, but this shows a fully functioning use of JLogger.

// Usings

using Jeff.Jones.JLogger;

using Jeff.Jones.JHelpers;

// Setting a class-wide variable. What you set may

// be different for development, QA, production, and troubleshooting production.

// This global value for the program is usually stored in some

// configuration data location. What is shown is an example.

// This can also come from a property, setting, or config file.

// Enum.TryParse() can be used to convert a string to the enum value.

private LOG\_TYPE m\_DebugLogOptions = LOG\_TYPE.Error |

LOG\_TYPE.Informational |

LOG\_TYPE.ShowTimeOnly |

LOG\_TYPE.Warning |

LOG\_TYPE.HideThreadID |

LOG\_TYPE.ShowModuleMethodAndLineNumber |

LOG\_TYPE.System |

LOG\_TYPE.SendEmail;

// Setting variables used to configure the Logger

// Typically in the programs startup code, as early as possible.

String filePath = <Some File Path like C:\Logs>

String fileNamePrefix = <Some file name prefix, like “WebSiteLog\_”>

Int32 daysToRetainLogs = <Number of days to retain previous log files or log entries>

if <Using a DB instead of a file for logging>

{

String dbServer = <SQL Server instance name>

String dbDatabase = <Database where the log table resides>

Boolean useWindowsAuth = <True or False as to using Windows authentication>

String dbUserName = <SQL Server user name, if not Windows Authentication>

String dbPassword = <SQL Server password, if not Windows Authentication>

String dbLogName = <Typically “DBLog” (see scripts), but could be changed by user>

response = Logger.Instance.SetDBConfiguration(dbServer,

dbUserName,

dbPassword,

useWindowsAuth,

true, // True enables DB logging

dbDatabase,

daysToRetainLogs,

m\_DebugLogOptions);

}

else

{

response = Logger.Instance.SetLogData(filePath,

fileNamePrefix,

daysToRetainLogs,

m\_DebugLogOptions,

""); // Emergency log prefix name;

// defaults to JLOGGER\_EMERGENCY\_LOG\_

}

if <Using the feature for sending emails>

{

Int32 smtpPort = <what port your SMTP server listens on>

Boolean useSSL = <True if SSL is required>

List<String> sendToAddresses = <One or more email addresses to send to>

String smtpServerName = <name of your SMTP server>

String smtpLogonEmail = <SMTP server logon email address>

String smtpPassword = <SMTP server password>

String smtpFromAddress = <Email address to list in your email as the “From” address>

String smtpReplyToAddrsss = <Email address to list in your email as the “Reply To” address>

response = Logger.Instance.SetEmailData(smtpServerName,

smtpLogonEmail,

smtpPassword,

smtpPort,

sendToAddresses,

smtpFromAddress,

smtpReplyToAddrsss,

useSSL);

}

// Now that the logger is configured, this method starts the logging.

Logger.Instance.StartLog();

// This ends the configuration example

// This is a code example of how to end the logging when the app shuts down.

Logger.Instance.StopLog();

Logger.Instance.Dispose();

// This ends the shutdown example

// Example of use in a method

// With most of this code related to logging being “boilerplate”, it does not add much to coding time

// Also note that because of the bitmask, if the desired bit is not on, the Log method is never called,

// which also reduces overhead.

void SomeMethod()

{

DateTime methodStart = DateTime.Now;

if ((m\_DebugLogOptions & LOG\_TYPE.Flow) == LOG\_TYPE.Flow)

{

Logger.Instance.WriteDebugLog(LOG\_TYPE.Flow, "1st line in method");

}

try

{

// Do some work here, then you could write a debug log entry only for testing

if ((m\_DebugLogOptions & LOG\_TYPE.Test) == LOG\_TYPE.Test)

{

Logger.Instance.WriteDebugLog(LOG\_TYPE.Test,

"Test Message",

"More info here");

}

// Do some more work

} // END try

// Example of using and logging multiple catches

catch (DivideByZeroException exDiv)

{

exDiv.Data.Add("x", 100);

exDiv.Data.Add("y", 0);

if ((m\_DebugLogOptions & LOG\_TYPE.Error) == LOG\_TYPE.Error)

{

Logger.Instance.WriteDebugLog(LOG\_TYPE.Error,

exDiv,

"Division by zero was intentional");

}

throw;

} // END first catch

catch (Exception exUnhandled)

{

exUnhandled.Data.Add("Some runtime variable", someRuntimeVariable);

// Example of having a log entry send an email

if ((m\_DebugLogOptions & LOG\_TYPE.Error) == LOG\_TYPE.Error)

{

Logger.Instance.WriteDebugLog(LOG\_TYPE.Error & LOG\_TYPE.SendEmail,

exUnhandled,

"An optional detail message");

}

}// END second catch

finally

{

// BEGIN dispose of method-level resources here =====================================

// if (dac != null)

// {

// dac.Dispose();

// dac = null;

// }

// END dispose of method-level resources here =======================================

if ((m\_DebugLogOptions & LOG\_TYPE.Performance) == LOG\_TYPE.Performance)

{

TimeSpan elapsedTime = DateTime.Now - methodStart;

Logger.Instance.WriteDebugLog(LOG\_TYPE.Performance,

$"END; elapsed time = [{elapsedTime,0:mm} mins, {elapsedTime,0:ss} secs, {elapsedTime:fff} msecs].",

elapsedTime.TotalMilliseconds.ToString());

}

} // END finally

} // END of method

# Logger Methods and Properties

## Static

* DEFAULT\_DEBUG\_LOG\_OPTIONS – constant with manufacturer recommend initial settings. You do NOT have to use this and can build your own that resides in your program.
* DEFAULT\_LOG\_RETENTION – constant that is the default for how many days log files are retained. You can use your own value and do NOT have to use this one.
* LOG\_CACHE\_FREQUENCY – constant for how long between attempts to write the log queue to the log file, in milliseconds.

## Instance

DaysToRetainLogs – (Get/Set) - How many days that the Logger instance retains previous log files.

DebugLogOptions – (Get/Set) - The debug flags that are active during the lifetime of the Logger instance

Dispose() – Implement the IDisposable.Dispose() method. Developers are supposed to call this method when done with this Object. There is no guarantee when or if the GC will call it, so the developer is responsible to. GC does NOT clean up unmanaged resources, such as COM objects, so we have to clean those up, too. There are no COM objects used in JLogger.

EmailEnabled – (Get ONLY) - True if sending email is enabled globally, false if off globally. Email sending is set by the LOG\_TYPE.SendMail bit being on or off in DebugLogOptions.

EmailLogonName - (Get/Set) - The logon name expected by the SMTP email server.

EmailPassword - (Get/Set) - The logon password expected by the SMTP email server.

EmailServer - (Get/Set) - The IP address or DNS name of the outgoing mail server

FromAddress - (Get/Set) - The email address to use with sending emails to indicate who the email is from.

IsDisposing – (Get ONLY) - Tells the caller if this instance is already being disposed. Returns true if the JLogger instance is being disposed, false if not.

LogFileName – (Get ONLY) - Fully qualified file name for the log file.

LogPath - (Get/Set) - Fully qualified path for the log file.

ReplyToAddress - (Get/Set) - The email address used to tell the recipient what address to reply to.

SendToAddresses – (Get ONLY List<String>, but List<String> still supports Add and other functionality. List<String> cannot be “set” as a List<String> object – internal creation only.) - These are the email addresses for log emails to be sent to.

Boolean SetEmailData(String emailServer,

String emailLogonName,

String emailPassword,

Int32 smtpPort,

List<String> sendToAddresses,

String fromAddress,

String replyToAddress,

Boolean useSSL) – Configure the email send functionality.

Boolean SetLogData(String logPath,

String logFileNamePrefix,

Int32 daysToRetainLogs,

LOG\_TYPE debugLogOptions,

String emergencyLogPrefixName = DEFAULT\_EMERG\_LOG\_PREFIX) – Configures the Logger for logging before starting the log.

SMTPPort – (Get/Set) - The port that the SMTP email server listens on.

Boolean StartLog() – Once the Logger instance is configured, this is used to start logging.

Boolean StopLog() – When the Logger instance is running, this is used to stop logging.

Boolean UseSSL – (Get/Set) - True if the email server requires using SSL, false if not.

Boolean WriteDebugLog(LOG\_TYPE pExceptionType,

Exception pExceptionToUse,

String pOptionalLogMessage) - Method used to write exception information to the log. This method writes a DebugLogItem instance to a queue, which is then emptied FIFO on a separate thread so calling this method does not block main thread activity.

Boolean WriteDebugLog(LOG\_TYPE pExceptionType,

String message,

String secondaryMessage = "") - Method used to write message information to the log. This method writes a DebugLogItem instance to a queue, which is then emptied FIFO on a separate thread so calling this method does not block main thread activity.

**Database Logging – Use the included table script to create the DBLog table in a SQL Server database of your choice. Then execute the two included stored procedure scripts to enable log inserts and log retention management (deleting log entries older than the specified “days to retain”).**

**The default name of the logging table in the chosen SQL Server database is DBLog. If this needs to be changed due to a conflict, carefully and thoroughly edit the table and two stored procedure scripts accordingly.**

Boolean DBEnabled – (Get/Set) - True if logging to DB is enabled, false if not.

String DBServer – (Get/Set) - Name of the SQL Server instance.

Boolean DBUseAuthentication – (Get/Set) - True to use Windows Authentication to access SQL Server, false if not.

String DBDatabase – (Get/Set) - Name of the SQL Server database where the log table is located.

String DBUserName – (Get/Set) - SQL Server logon name with permissions for the log table (DBLog), if not using Windows Authentication. Otherwise, just use “”.

String DBPassword – (Get/Set) - SQL Server password matching the logon name, if not using Windows Authentication. Otherwise, just use “”.

Boolean SetDBConfiguration(String dbServer,

String dbUserName,

String dbPassword,

Boolean useWindowsAuthentication,

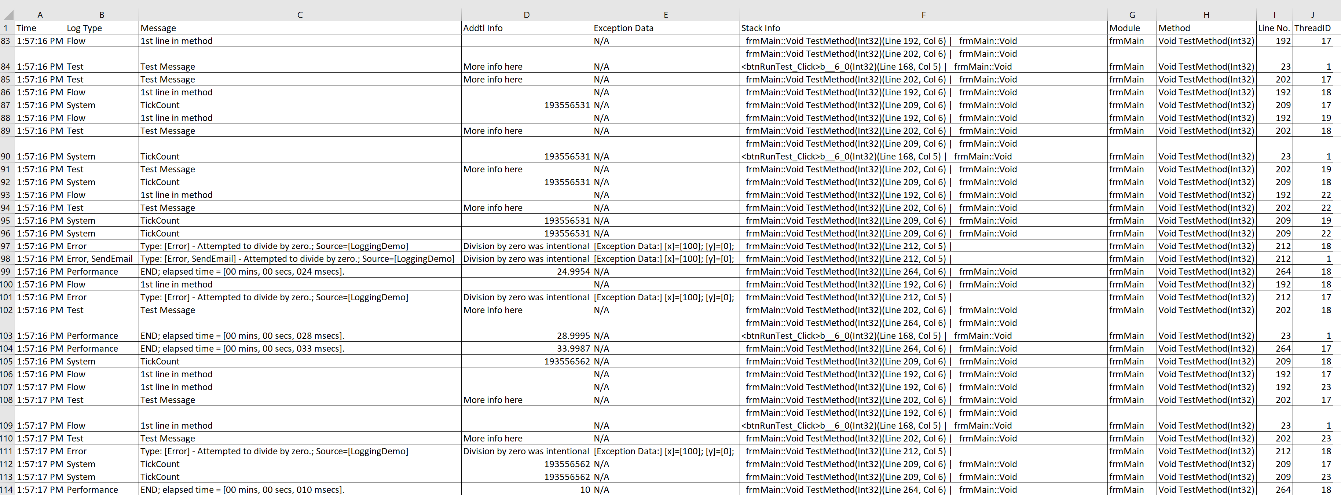
Boolean dbLogEnabled,

String dbName,

Int32 daysToRetainLogs,

LOG\_TYPE debugLogOptions) – used to configure database logging.

Sample file-based log opened with Excel:



Sample Database Log using SSMS:

