



*Genesis Software Corp. - GS Log Telemetry Tool Kit*

# GSLOG TELEMETRY WRITER

Developer Guide

# GSLog Developer Guide

Tap or click the desired content to jump to the page

## Contents

Introduction .....	2
Installation .....	2
Overview .....	2
GSLog.LogFile “Text File Writer” .....	3
DBLog and SQLLog “Database Writer” .....	4
GSLog.DBLog Data Structure .....	4
GSLog.MemoryLog .....	5
GSLog.MemoryDBComboLog .....	6
GSLog.CustomDBLog “Database Writer” .....	7
Reference – Class Details .....	11
IGSLog Interface .....	11
GSLog.LogFile .....	11
Properties .....	11
Methods .....	12
GSLog.DBLog .....	13
Properties .....	13
Methods .....	15
GSLog.MemoryLog .....	16
Properties .....	16
Methods .....	17
GSLog.MemoryDBComboLog .....	18
Properties .....	18
Methods .....	18
GSLog.CustomDBLog .....	20
Properties .....	20
Methods .....	20
Supporting Classes .....	20
GSLog.LogStatus .....	20
GSLog.LogCategory .....	20
Methods .....	20
Conclusion .....	21

## Introduction

For most software developers, making a tool like this is so easy, we do it for ourselves often. Except for when we work for someone else. Then we use what we are given or directed to use. Today, 2021, the architects I work for have us use log4net which is an industry standard log writing tool I am very proficient with as well. The GSLog telemetry writer works better for me and the methods I use for telemetry design and analysis with the sibling EZ Log Viewer tool. Having the writing portion of the toolkit in a separate .NET class assembly enables me to easily plug in to any .NET application I am working on and spend little time standing up the telemetry subsystem. I find GSLog to be far simpler to use and much more effective.

The goals of the GSLog telemetry writer when I wrote it were

1. Installation and configuration be ultra-simple.
2. Flexibility to write to a text file, a database table, or memory region and switch between them with little code changes.
3. Control how much telemetry is being written without having to rebuild and deploy, and be able to react to changes in verbosity in real time.
4. Contain features useful to an application program for managing log files and reactive exception handling.

## Installation

By default, the assembly file (GSLog.dll) is installed to the C:\Program Files\Genesis\EZ Log\Writer. However, you can re-direct the installation to a folder of your choice. The installation does not install the assembly to the GAC.

After installation of the assembly, GSLog.dll, you can move or copy it to where ever you want to. Simply reference the assembly file into your Dot Net project and you're good to go. *(In addition, you could use Visual Studio's nuget utility to install the assembly to your visual studio project. **Look this assembly up in nuget using "EZ-Log.GSLog" as the search term.**)*

## Overview

Whatever you implement. Text file, database, and/or memory region logging, the primary method you will use to write telemetry in this assembly is the WriteLog method. This method is implemented in each of the GSLog types. Consider the method signatures below.

WriteLog(*(Optional)* DetailLevel, *(Optional)* Category, Message)

The "DetailLevel" parameter is what you use to set the minimal "LogLevel" at which the log entry will be written during execution of the application. This parameter is one mechanism you can use to control how much telemetry is being written by the application. *(Similar to the info, debug, error, paradigm used by log4net only expandable)*

How it works: your source code makes a log entry with a DetailLevel value of 3. If the supporting GSLog writer object's LogLevel property is set to 2 or less at the time the WriteLog statement is being processed, the log entry will NOT be written because the **LogLevel of the telemetry writer is less than the DetailLevel of the telemetry or WriteLog statement.** You can omit the DetailLevel as a parameter if you prefer.

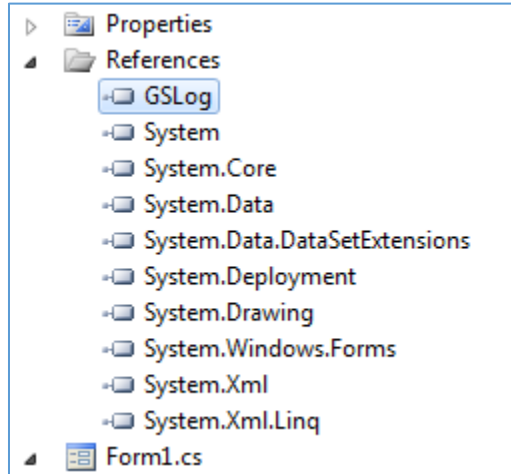
The "Category" parameter is used to catalog the log entry. The parameter value is optional and you can omit the parameter if you prefer.

## GSLog Developer Guide

---

The “Message” parameter is the information you want written to the log. While it is a required parameter, an empty string is accepted. An empty string will add a blank line to the log file.

To use the assembly in your .NET projects, just make sure you have a reference to the GSLog.dll assembly.



Continue reading to see how to use each of GSLog’s Text File, Data Base, and Memory Region telemetry writers.

### GSLog.LogFile “Text File Writer”

Setup is pretty simple. (See VB Sample Code below)

In addition to writing to a text file, the text file writer has features that help with managing the files that are written.

The sample below shows the setup for the application to write a text file for each day, and also limits the number of log files in the folder where logs are written to 20 files.

```
Dim log As New GSLog.LogFile
log.DataFile = $"C:\MyApp\Logs\FileName {Now.ToString("MM-dd-yyyy")}.log"
log.LogLevel = 3
log.CheckMaxFiles = True
log.MaxFilesInLogFolder = 20
log.WriteLog(1, "App Startup", "Version 1.0")
```

There are properties that you can use to set the maximum file size if you prefer to have a single log file as seen below.

```
Dim log As New GSLog.LogFile
log.DataFile = $"C:\MyApp\Logs\FileName.log"
log.LogLevel = 3
log.MaxFileSize = 5000000
log.WriteLog(1, "App Startup", "Version 1.0")
```

Learn them in the Reference Section starting on page 11 in this document.

## GSLog Developer Guide

### DBLog and SLog “Database Writer”

Notice how easy it is to setup and use the DBLog of SLog class in the following VB Code snippet...

```
Dim log As New GSLog.DBLog
log.conn_str = "Your connection string here"
log.Open("Process Name 1")
log.WriteLog(1, "App Startup", "Version 1.0")

log.Open("Process Name 2")
log.WriteLog(1, "App Startup", "Version 1.0")
log.Close()
```

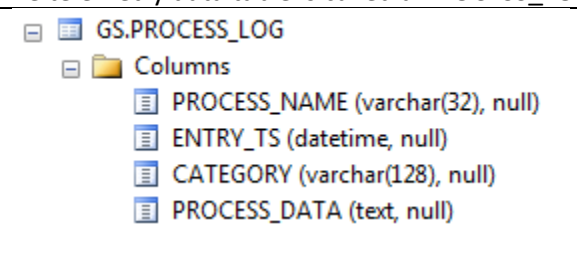
The sample above opens the log and writes entries for 2 separate processes.

What about table structure?

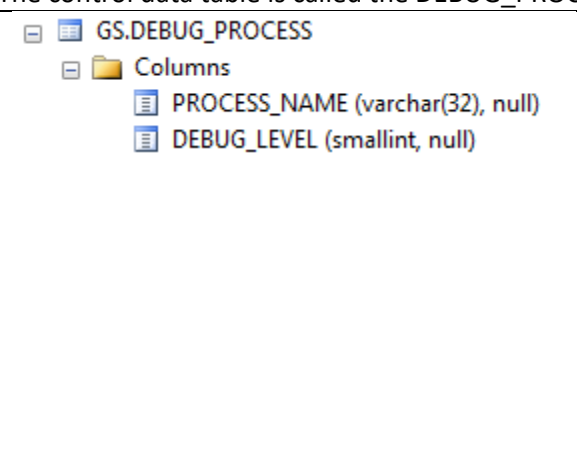
### GSLog.DBLog Data Structure

The GSLog DBLog native table structure for consists of 2 tables. 1 to hold the telemetry data, 1 to hold control meta data. It may sound complicated but it is really easy. Consider the details below.

The telemetry data table is called a PROCESS\_LOG which is a table for one or more processes

	<p>The PROCESS_NAME column is used to support multiple applications or individual processes within a single application, (like a web application), sharing a single DB Table.</p> <p><i>The Log Viewer tool in the GSLog Telemetry toolkit takes advantage of this data structure in a way that makes monitoring and debugging multiple applications easy and convenient. (<a href="http://www.gen-e-sys.com">www.gen-e-sys.com</a> for more info)</i></p>
---	--

The control data table is called the DEBUG\_PROCESS table.

	<p>The PROCESS_NAME column in this table points to the associated column in the telemetry table. The DEBUG_LEVEL is used by the application to know how much telemetry to send. A numeric value is far easier to adapt to changes in telemetry strategy after the app is online.</p> <p>You can change the DEBUG_LEVEL in real time and your application will react to it.</p> <p><i>The Log Viewer tool in the GSLog Telemetry toolkit takes advantage of this data structure by making it convenient to change the DEBUG_LEVEL for an application or process.</i></p>
--	---

How were the tables created? I could have created the tables manually. Often, I do. Possibly to change the length of a column. However, the assembly can also create the supporting tables. Consider the code snippet below.

## GSLog Developer Guide

---

```
Dim log As New GSLog.DBLog
log.conn_str = "Your connection string here"
log.MakeTables()
```

You can use your own table naming if you like. See the Class Reference Section starting page 11 for the GSLog.DBLog reference later in this document for more information. Self-generating tables work well for my applications that install and create the tables on initial startup.

### GSLog.MemoryLog

The memory log is used so you can write the telemetry for a process to a temporary location and output the telemetry to permanent storage later.

Consider the following Visual Basic sample

0 references

```
Private Function GetCircumference(ByVal Diameter As Double) As Double
    *****
    Dim iDebugLevel As Integer = 3
    Dim sDebugLocation As String = String.Format("{0}:{1}", Me.GetType.Name, System.Reflection.MethodBase.GetCurrentMethod.Name)
    Dim log As New GSLog.MemoryLog
    log.WriteLog(iDebugLevel, sDebugLocation, $"Entry Point: Diameter={Diameter.ToString()}")
    *****
    Dim ReturnValue As Double = -1
    Try
        log.WriteLog(iDebugLevel, sDebugLocation, "Multiply Diameter * Math.PI")
        ReturnValue = Diameter * Math.PI
    Catch ex As Exception
        log.WriteLog(0, sDebugLocation, $"Unexpected Error: {ex.Message}")
        SaveLog(log.LogDump())
    End Try
    Return ReturnValue
End Function
```

1 reference

```
Private Sub SaveLog(LogContents As String)
    Dim sw As New System.IO.StreamWriter("C:\Temp\Error.log")
    sw.Write(LogContents)
    sw.Close()
End Sub
```

The above example is indeed overly simple. The point to this is; if no exception occurred during the process, nothing would get written to the log. When there IS an exception however, the log entry is written and it has all of the entries leading up to the error because of the memory region. I found this telemetry style almost revelational.

## GSLog Developer Guide

---

### GSLog.MemoryDBComboLog

This class puts together the DB Writer with the Memory Writer. To put it simply, the WriteLog method stores all the entries in a memory region. The LogDump method is changed to save the memory contents to the database instead of returning a string. The memory contents are saved to the PROCESS\_DATA column (see the DBLog class reference at the end of this document)

This class saves the effort of spinning up 2 separate telemetry writers to accomplish the same thing.

```
Private Function GetCircumference(ByVal Diameter As Double) As Double
    *****
    Dim iDebugLevel As Integer = 3
    Dim sDebugLocation As String = String.Format("{0}:{1}", Me.GetType.Name, System.Reflection.MethodBase.GetCurrentMethod.Name)
    Dim log As New GSLog.MemoryDBComboLog
    log.conn_str = "Your Connection String Here"
    log.Open("Process 1")
    log.WriteLog(iDebugLevel, sDebugLocation, $"Entry Point: Diameter={Diameter.ToString()}")
    *****
    Dim ReturnValue As Double = -1
    Try
        log.WriteLog(iDebugLevel, sDebugLocation, "Multiply Diameter * Math.PI")
        ReturnValue = Diameter * Math.PI
    Catch ex As Exception
        log.WriteLog(0, sDebugLocation, $"Unexpected Error: {ex.Message}")
        log.LogDump(sDebugLocation)
    End Try
    Return ReturnValue
End Function
```

Same thing as before but instead of writing the memory log contents to a text file, (and manually at that), I place it in a single DB record. Very efficient. When used in conjunction with the Log Viewer in the GSLog Telemetry toolkit ([www.gen-e-sys.com](http://www.gen-e-sys.com)) you will be finding answers to issues quickly.

## GSLog Developer Guide

---

### GSLog.CustomDBLog “Database Writer”

I added this type to make GSLog write to a log table that had specific table structure other than the structure native to the GSLog toolkit. I had to write an application that worked alongside a “sibling” application that shared a log table. I wanted to use GSLog so I added this class.

Truly the CustomDBLog class is one where I was “forcing” GSLog.DBLog to be something it is NOT. If you have to use the CustomDBLog class, honestly, it would be faster for you to write a custom routine yourself without GSLog. But if you’re like me and just have to use GSLog...

You should be fairly expert in using the DBLog class before reading further in the section. If you are not very familiar with the DBLog class, it would be better to continue on to the next section of the document. The Reference section starting at page 11.

Now for the internals of the CustomDBLog writer.

I chose to employ the .NET [VB ParamArray /C# params] mechanism to append custom data to the WriteLog method of DBLog. I felt it would not be uncomfortable based on familiarity with the ParamArray/params mechanism when we use “String.Format()”.

Consider the code that follows.

```
Dim conn_str As String = "<your connection string here>"
'See the details for more information on the the constructor
Dim oLog As New CustomDBLog(conn_str, "rp_authlib", "EntryDate", "Details", "Subject")
oLog.Open("CUSTOM_LOG2", 3)
Dim sCustomData As String = ""
Dim CustomDate As Date = #9/22/2017#
Dim CustomInt As Integer = 5181
Dim CustomString As String = "Cust#"
Dim x As Integer = 1

oLog.WriteLog(1, "CustomLogTest2", "***** Start Logging *****", "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")

x += 1
CustomInt -= 3
CustomDate = CustomDate.AddDays(3)
oLog.WriteLog(3, "CustomLogTest2", "Written @ Detail Level 3", "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")

x += 1
CustomInt -= 3
CustomDate = CustomDate.AddDays(3)
oLog.WriteLog(4, "CustomLogTest2", "Written @ Detail Level 4", "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")

x += 1
CustomInt -= 3
CustomDate = CustomDate.AddDays(3)
oLog.WriteLog(0, "CustomLogTest2", "Written @ Detail Level 0", "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")

x += 1
CustomInt -= 3
CustomDate = CustomDate.AddDays(3)
oLog.WriteLog(1, "CustomLogTest2", "***** End Logging *****", "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")
```

Notice WriteLog now has extra parameters?

Looks kind of familiar? yes?

Each parameter array entry is a name value pair in the form of a string.



## GSLog Developer Guide

The left side of the value pair is the DB Column Name. The right side is the value as you would write it in a SQL INSERT statement.

Based on the source code above. The resulting table is shown below:

```
SELECT * FROM rp_AuthLib
```

EntryID	EntryDate	Machine	Location	Subject	UserID	Details
2BB65C4D-BDD9-4625-8FB2-1A5BDC112996	2017-09-26 16:22:05.997	ROB-PC2	Cust#	CustomLogTest2	RobP	***** Start Logging *****
64680269-BAB2-4CFA-AA7A-7758DCAF364C	2017-09-26 16:22:07.090	ROB-PC2	Cust#	CustomLogTest2	RobP	Written @ Detail Level 3
F4A9A966-3E93-4083-AD2A-1B6620A8AB81	2017-09-26 16:22:08.543	ROB-PC2	Cust#	CustomLogTest2	RobP	Written @ Detail Level 0
405270DB-277D-4C44-8C33-DD516EF035C9	2017-09-26 16:22:09.197	ROB-PC2	Cust#	CustomLogTest2	RobP	***** End Logging *****

Unlike GSLog's DBLog, the CustomDBLog type takes a bit of setup.

First, we have to communicate to the writer what the data structure of the table it is writing to is. Rather than do this from a config file, (which can still be used...), I put the means to communicate some of the structure in the Constructor where we instantiate the class.

With this in mind let's look at the constructor for the CustomDBLog.

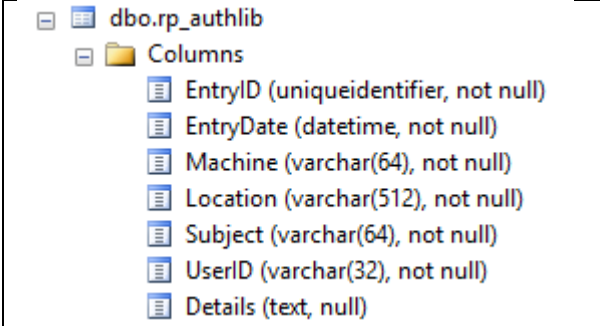
```
C#
public CustomDBLog(string connectionString, string TableName, string EntryDate_ColumnName, string
LogMessage_ColumnName, string Category_ColumnName = "", string ProcessName_ColumnName = "")
```

```
VB
Public Sub New(connectionString As String, TableName As String, EntryDate_ColumnName As String,
LogMessage_ColumnName As String, Optional Category_ColumnName As String="", Optional
ProcessName_ColumnName As String="")
```

So..., the connectionString and TableName parameters should be obvious.

The "Date" column is mapped using the (EntryDate\_ColumnName) parameter, and the "Message" column is mapped using the (LogMessage\_ColumnName) parameter. Because the WriteLog method signature asks for a Category parameter, we optionally map the Category Column using Category\_ColumnName parameter of the constructor. The ProcessName\_ColumnName is similar but has a twist that I cover in a few more paragraphs.

Now let's look at the DB Structure of a custom log table I am using for an example called "rp\_authLib"

	<p>It is easy to see that there are more fields than what we are specifying in the constructor.</p> <p>The following explanation should help to see the mechanics of setting up the Custom DB Log writer.</p>
--	---

Using the example at the beginning of this document:

## GSLog Developer Guide

---

```
Dim oLog As New CustomDBLog(conn_str, "rp_authlib", "EntryDate", "Details", "Subject")
```

```
Or  
GSLog.CustomDBLog oLog = new GSLog.CustomDBLog(conn_str, "rp_authlib", "EntryDate", "Details", "Subject");
```

To paraphrase; the constructor asks for the first 4 parameters that will meet our “must have” requirements for custom logging. But what about those columns in the table we did NOT map? See the WriteLog Method below to see how we will populate those columns.

### *WriteLog Method*

Here is the signature for the CustomDBLog WriteLog method.

```
VB  
Sub WriteLog(DetailLevel As Integer, Category As String, Message As String, ParamArray CustomData As String())  
C#  
void WriteLog(int DetailLevel, string Category, string Message, params string[] CustomData)
```

For the CustomDBLog writer, the WriteLog signature is the same as all the other logging classes in this assembly with an additional parameter at the end. That parameter gives us the means to populate the columns not mapped in the constructor.

Let’s look at an implementation from the example provided earlier.

```
oLog.WriteLog((Standard parameters)1, "CustomLogTest2", "***** Start Logging *****",  
(Parameter Array) "EntryID=NewID()", $"Machine={Environment.MachineName}", $"Location={CustomString}", "UserID=RobP")
```

DetailLevel is nothing new in the EZ-Log writer paradigm. The second argument is the Category that we mapped to the “Subject” column in the constructor. The “Message” parameter was mapped to the “Details” column in the constructor. Everything past the “standard” parameters are name/value pairs having the Name matching the DB Column Name and the value being written.

This minor expansion of the WriteLog signature, provides a means to write to custom data tables easily and in a manner consistent with the interface.

You may ask yourself what happens if I choose not to map the “category” to a column in the table. You still will have to put text in the WriteLog method. Whatever text you put in the parameter is ignored if you do not map the Category Column in the constructor. Simple. I leave it to you. You’re going to put text in the parameter anyway, might as well map it. Still, you could have a very valid reason not to do that as well.

We need to move on to talk about mapping the “PROCESS\_NAME” to “ProcessName\_ColumnName”. The Open method explained next is the key.

### *Open Method*

Here is the call from the example.

```
oLog.Open("CUSTOM_LOG2", 3)
```

This is exactly the same as DBLog and it acts the very same way as the DBLog.Open method. What is different is in DBLog, the ProcessName parameter is written to our “standard” Log Table because the column is there. However, for the CustomDBLog it is just a bit different. A column to hold that data may not be available.

If the constructor’s “ProcessName\_ColumnName” parameter was populated, thus mapping a column in the log table, then the process name used in the Open method will be written to the mapped column of the Log Entry. Otherwise it

## GSLog Developer Guide

---

will be used to look up the PROCESS\_NAME in the table referred to in the TableNameProcessName property and get the current LogLevel for the PROCESS\_NAME

This means custom logging can take advantage of the dynamic LogLevel changes set up in the EZ-Log Writer controller. However, unlike the DB Log and MemoryDBCombo Log, if you don't care to take advantage of Dynamic Logging via the [TableNameProcessName](#) property you can choose not to use the "Open" method at all.

Altering our example:

```
Dim conn_str As String = "<your connection string here>"  
'See the details for more information on the the constructor  
Dim oLog As New CustomDBLog(conn_str, "rp_authlib", "EntryDate", "Details", "Subject")  
oLog.Open("CUSTOM_LOG2", 3)  
oLog.LogLevel = 3 'No Dynamic Log Level
```

A Note you should consider: If do not use the open method and forget to set the LogLevel manually, the default value for LogLevel is 0 (zero) effectively turning off any logging.

# GSLog Developer Guide

## Reference – Class Details

In this section I try to share insight on specifically what a method does or how a property value effects the process.

### IGSLog Interface

There is one interface to this telemetry writer. It provides the mechanism to easily switch between writing to a Database or Text File or Memory Region or any of the Classes supplied by this assembly. The interface also allows you to extend the workings to extend this logging system to integrate with an existing logging system.

### GSLog.LogFile

#### Properties

Name	Data Type	Description
DataFile	String	The full path to the text file the writer should write to
LogLevel	Integer	Controls the verbosity of how much information is written to the log. The higher the LogLevel the more detail is written to the log.
CategoryPrefix	String	The WriteLog method described below has a Category parameter that the programmer uses to group Log Entries. The category prefix is used to prepend information to the category when a log entry is written to help isolate a single user's log entries for multiuser applications sharing a Log File or Database table. Useful? Rarely, but nice when you need it.
AutoEscalate	Boolean	Default=True See below for details. When the value for this property is False. Auto Escalate feature is off
AutoEscalateLevel	Integer	Default=3 When the detail level of the WriteLog method is zero (0) the "error" indicator, the Log Writer's "LogLevel" property will be escalated to the value set by this property.  <i>It has been my experience that most users, when they experience an error, and the error is not a program crash, they will usually attempt the function or process a second time. If during program execution the "LogLevel" is not set to capture information that could lead to solving or discovering what the problem was, the AutoEscalateLevel property will offer the chance discovering valuable data the second time the user attempts the process.</i>
FileAccessRetryCount	Integer	Default Value=100. For multi user applications that have a central text file for a log, contention for write access to the file could exist if two or more instances of an application were writing to the log simultaneously. The Log Writer instance will re-try to write the entry as many times as the value of this property dictates before throwing a file contention exception. Note: Experience in testing shows that a value of 100 works well. However for apps with many simultaneous users a larger value may be needed.
LogType	GSLog.eLogType	Read Only property. Returns: GSLog.eLogType.TextFile.
MaxFileSize	Integer	DefaultValue=5,000,000 (5MB) If the current size for the DataFile exceeds the MaxFileSize value, the DataFile is backed up as <Log File Name>_Last and a new log is started. If a backup file exists already for the log, the backup file is deleted.
CheckMaxFiles	Boolean	Default=True. See the next property for details.
MaxFilesInLogFolder		Default=10. Minimum setting for this property is 2 Often log files are written on a daily basis to keep log files small and to easily lookup a specific day.  Often, after a software application is stable, I forget to "clean up" the log folder. This property in conjunction with the CheckMaxFiles property helps to limit the number of daily log files kept in a folder.  When the number of log files in a folder exceeds the limit set by this property, the oldest logs are deleted. And the newer log files are kept.

## GSLog Developer Guide

---

### Methods

Name	Return Data Type	Description
WriteLog	None	<p>Writes a single line to the log.</p> <p>Parameters</p> <p>DetailLevel: Works in conjunction with the instance LogLevel property. If the DetailLevel parameter value in the method call is equal to or less than the class instance's value of the LogLevel at the time the method is called, an entry is added to the log.</p> <p>Category: The category the log entry should be grouped under. This parameter allows the programmer to set multiple groupings for the log analysis.</p> <p>Message: The information the programmer wants to appear in the log.</p>
Status	Log Status Structure	I added this method so I could have an app ask the log for its status
DateFormat	String	I use the .NET string defs. Default="yyyyMMdd HHmmss.ffff"

# GSLog Developer Guide

## GSLog.DBLog

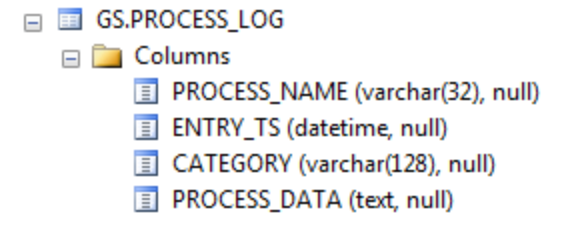
This telemetry writer basically issues a SQL Insert statement inserting a row into a table. I use Microsoft's OleDB Data component because OleDB has a data provider available for many popular database engines and the SQL being generated by this class is rudimentary.

Some people are partial to using native database providers. However, even if I used a data driver native to the target database, this assembly really has no need to take advantage of whatever benefit there is to get from a native data driver/provider. Be that as it may, I respect that there may be times where a native driver is a "requirement" from our customer.

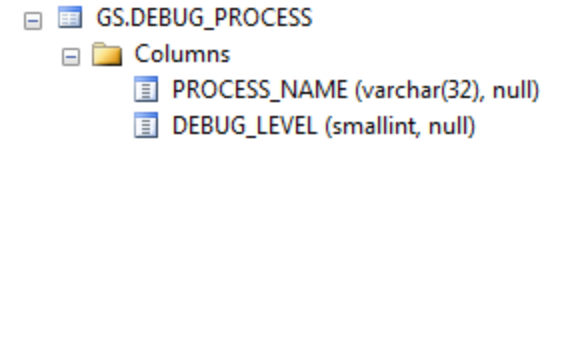
For those requiring use of SQL Server native .NET data providers, an additional DB Writer is implemented using the native SQL Client data provider. The Type name that uses the SQLClient provider is GSLog.SQLLog. The details and methods of SQLLog are the same as the DBLog class.

The GSLog native table structure consists of 2 tables. 1 to hold the telemetry data, 1 to hold control meta data. Consider the details below.

The telemetry data table is the PROCESS\_LOG

	<p><i>The Log Viewer tool in the GSLog Telemetry toolkit takes advantage of this data structure in a way that makes monitoring and debugging multiple applications easy and convenient. (<a href="http://www.gen-e-sys.com">www.gen-e-sys.com</a> for more info)</i></p>
---	--

The control data table is the DEBUG\_PROCESS

	<p>As you can see, the PROCESS_NAME column is related to the PROCESS_LOG table. The DEBUG_LEVEL is used by the application to know how much telemetry to send.</p> <p>You change the DEBUG_LEVEL in this table and your application will react to it in real time if needed.</p> <p><i>The Log Viewer tool in the GSLog Telemetry toolkit found at <a href="http://www.gen-e-sys.com">www.gen-e-sys.com</a> takes advantage of this data structure by making it convenient to change the DEBUG_LEVEL for an application or process.</i></p>
--	---

Consider the properties and methods of this class below

### Properties

Name	Data Type	Description
conn_str	String	The .NET connection string.
Connection	OleDB Connection or SQLClient Connection	Please see above.

## GSLog Developer Guide

CategoryPrefix	String	The WriteLog method described below has a Category parameter that the programmer uses to group Log Entries. The category prefix is used to prepend information to the category when a log entry is written to help isolate a single user's log entries for multiuser applications sharing a Log File or Database table. Useful? Rarely, but nice when you need it
AutoEscalate	Boolean	Default=True See below for details. When the value for this property is False. The auto escalate feature is turned off.
AutoEscalateLevel	Integer	Default=3 When the detail level of the WriteLog method is zero (0) the Log Writer's "LogLevel" property will be escalated to the value set by this property.  <i>It has been my experience that most users, when they experience an error, and the error is not a program crash, they will usually attempt the function or process a second time. If during program execution the "LogLevel" is not set to capture information that could lead to solving or discovering what the problem was, the AutoEscalateLevel property will offer the chance discovering valuable data the second time the user attempts the process.</i>
DBType	eDBType Enum	Default=SQLServer This is the type of DB being used to store the telemetry. Certain popular DB Engines have a specific syntax when inserting and formatting date values. Setting this properly adjusts the syntax of the SQL written by the WriteLog method to suit the DBType. <b>public enum</b> eDBType { MSAccess, SQLServer, <i>For Oracle DB's use the SQLServer entry</i> DB2 }
TableNameProcessLog	String	By default this property's value is "PROCESS_LOG" See the Table Structure Section above for details  When the WriteLog method generates the SQL INSERT statement, it references this property for the Table Name portion of the SQL Statement
TableNameProcessName	String	By default the Table Name is "DEBUG_PROCESS". This is the name of the "control" table used to read the LogLevel property
LogLevel	Integer	Default=0 This property value is automatically set by the "Open" method but can still be changed by the application.
LogLevelSensitivity	eLogLevelSensitivity	Default=Low. The LogLevel property value is examined by the WriteLog method to see if the entry should not be written to the log. When value of this property is set to "High", the WriteLog method will fetch the LogLevel value from the DB before checking the LogLevel value to see if the Entry should NOT be written.  For the SQL Version we take advantage of extended syntax and query the table with the nolock SQL clause.
LogType	eLogType	Read Only property. Returns: GSLog.eLogType.Database.

## GSLog Developer Guide

### Methods

Name	Return Data Type	Description
WriteLog		<p>Writes a single line to the log.</p> <p>Parameters</p> <p style="padding-left: 20px;">DetailLevel: Works in conjunction with the LogLevel property. If the DetailLevel is equal to or less than the value of the LogLevel at the time the method is called an entry is added to the log.</p> <p style="padding-left: 20px;">Category: The category the log entry should be grouped under. This parameter allows the programmer to set multiple groupings for the log analysis.</p> <p style="padding-left: 20px;">Message: The information the programmer wants to appear in the log.</p> <p style="padding-left: 20px;">There are overrides making the detail level and category optional.</p>
Open		<p>Opens a Process Log.</p> <p>Parameters:</p> <p style="padding-left: 20px;">ProcessName: String with the Name of the Processes Log to open</p> <p style="padding-left: 20px;">DefaultLogLevel: Optional integer default value=9.</p> <p>As a part of the initialization or instantiation of this class the Programmer must "Open" a Process Log. During the Open process the "TableNameProcessName" property table is opened looking for a row with a PROCESS_NAME matching the given parameter. If a matching row is found the DEBUG_LEVEL field of the matching row is read and stored in the LogLevel Property to be used for subsequent WriteLog calls.</p> <p>If a Matching PROCESS_NAME is NOT found, a new row is inserted to the table with the new Row's DEBUG_LEVEL set to the "DefaultLogLevel" parameter value.</p>
Close		<p>By default, Connections are closed when the instance "Self Connects" and the instance is Disposed of by the GC or manually. This process is automatically called during the handling of a Dispose event or call.</p> <p>You usually don't HAVE to use this method.</p> <p>If it makes you feel better and you want to manually close and dispose the "Connection" property, this is the method to use.</p>
MakeTables		<p>This Method will create the 2 supporting tables of the logging system. Uses the "TableName----" properties for the table names.</p> <p>Intended for applications that self-generate supporting table structure.</p>
Status	Log Status Structure	I added this method so I could have an app ask the log for its status.



## GSLog Developer Guide

### GSLog.MemoryLog

Holds all log entries in an internal List< string> object that is output via a class method with other decorations.

The details of the class are.

#### Properties

Name	Data Type	Description
CategoryPrefix	String	The WriteLog method described below has a Category parameter that the programmer uses to group Log Entries. The category prefix is used to prepend information to the category when a log entry is written to help isolate a single user's log entries for multiuser applications sharing a Log File or Database table. Useful? Rarely, but nice when you need it
AutoEscalate	Boolean	Default=True See below for details. When the value for this property is False. The LogLevel Property is not automatically escalated when WriteLog is called with a zero Log Level parameter.
AutoEscalateLevel	Integer	Default=3 When the detail level of the WriteLog method is zero (0) the Log Writer's "LogLevel" property will be escalated to the value set by this property.  <i>It has been my experience that most users, when they experience an error, and the error is not a program crash, they will usually attempt the function or process a second time. If during program execution the "LogLevel" is not set to capture information that could lead to solving or discovering what the problem was, the AutoEscalateLevel property will offer the chance discovering valuable data the second time the user attempts the process.</i>
InjectTimeStamp	Boolean	Default=True. When true the WriteLog method writes a time stamp to each entry in the internal list.
TimeStampFormatString	String	Default=":yyyyMMdd HHmm:ss.ffffff" This should say it clearly <pre>         ...         If Me.InjectTimeStamp Then             _LogEntries.Add(String.Format("{0} &amp; Me.TimeStampFormatString &amp; "}: {1}::{2}", Now, If(         Else     </pre> Sometimes I am only interested in the seconds of the timestamp. This property allows me to control exactly what is injected by the writer.  The date time string formatting follows the Microsoft guidelines provided under the "Format String Component" documentation of the String.Format method.  <a href="https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting">https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting</a>
LogLevel	Integer	Controls the verbosity of how much information is written to the log. The higher the LogLevel the more detail is written to the log.
LogType	GSLog.eLogType	Read Only property. Returns: GSLog.eLogType.MemoryLog

## GSLog Developer Guide

### Methods

Name	Return Data Type	Description
WriteLog	None	<p>Writes a single line to the log.</p> <p>Parameters</p> <p>DetailLevel: Works in conjunction with the instance LogLevel property. If the DetailLevel parameter value in the method call is equal to or less than the class instance's value of the LogLevel at the time the method is called, an entry is added to the log.</p> <p>Category: The category the log entry should be grouped under. This parameter allows the programmer to set multiple groupings for the log analysis.</p> <p>Message: The information the programmer wants to appear in the log.</p>
LogDump	String	<p>Returns all the WriteLog entries made prior to calling this method as a serial string (including whitespace).</p> <p>Parameters:</p> <p>LogTitle (optional) string: Prepends a "decorated" string to the beginning or end of the log text. Used to demarcate the logdump when embedded with other text. (Like an exception message).</p> <p>EmptyTheLogWhenDone (optional) Boolean Default=False When set to true after the return string is prepared for return the internal string builder is emptied.</p>
Clear		Empties the internal string list
Status	Log Status Structure	I added this method so I could have an app ask the log for its status

## GSLog Developer Guide

### GSLog.MemoryDBComboLog

After I added the MemoryLog log type, I added this hybrid type that pretty much puts together the Memory and DB Log types. Prior to writing this style of logging I would use the memory log to store the play by play of a process and when an exception occurred, I would write the memory log into the message text (with the exception information of course) and save that as a single record in the database table using the DB Log Style. This class simply rolls all of that process into a single method call.

#### Properties

Name	Retrun Data Type	Description
conn_str	String	Same as the DBLog
Connection	OleDbConnection	Same as the DBLog
CategoryPrefix	String	Same as the DBLog
AutoEscalate	Boolean	Same as the DBLog
AutoEscalateLevel	Integer	Same as the DBLog
DBType	eDBType Enum	Same as the DBLog
TableNameProcessLog	String	Same as the DBLog
TableNameProcessName	String	Same as the DBLog
InjectTimeStamp	Boolean	Same as MemoryLog
TimeStampFormatString	String	Same as MemoryLog
PersistConnection	Boolean	Same as the DBLog
LogLevel	Integer	Same as the DBLog
LogLevelSensitivity	eLogLevelSensitivity	Same as the DBLog

This methodology has a usefulness is not readily evident. Consider the following: I am one of let's say 50 (just to be modest) programmers each of us are working on a large corporation's applications. Having Web Applications and Network Server Farms in a single corporate environment ALL writing telemetry to a SINGLE location, it is very likely that; 2 or more instances of the same application, running on 2 separate servers, are writing to a single storage location at the exact same time.

This poses a challenge in that you have to filter out let's say 50 or 60 lines (usually much less) out of possibly tens of millions of lines of telemetry data rows. Then, after you have filtered things by application and environment you have to make sure the rows of data truly belong together before you even begin to locate and repair an issue with the software.

Using my strategy with the memory DB Log, when I find the entry locating the application and the time that the bug or anomaly occurred, all of the relevant information is already packaged together in a single row. Ready for analysis using the EZ-Log Viewer.

## GSLog Developer Guide

### Methods

Name	Return Data Type	Description
WriteLog	None	<p>Writes a single line to the memory log.</p> <p>Parameters</p> <p>DetailLevel: Works in conjunction with the instance LogLevel property. If the DetailLevel parameter value in the method call is equal to or less than the class instance's value of the LogLevel at the time the method is called, an entry is added to the log.</p> <p>Category: The category the log entry should be grouped under. This parameter allows the programmer to set multiple groupings for the log analysis.</p> <p>Message: The information the programmer wants to appear in the log.</p>
LogDump	String	<p>Takes all the WriteLog entries made prior to calling this method packages them into an internal string. Then Calls the SaveDBEntry method to store the write log entries in the Log Table. If an error occurs, returns the exception message. Otherwise returns an Empty String.</p> <p>Parameters:</p> <p>LogTitle (optional) string: Prepends a "decorated" string to the beginning or end of the log text. Used to demarcate the logdump when embedded with other text. (Like an exception message).</p> <p>EmptyTheLogWhenDone (optional) Boolean Default=False When set to true after the return string is prepared for return the internal list is emptied.</p>
SaveDBEntry		This does the same thing the WriteLog method does in the DBLog class.
Clear		Empties the internal list
Status	Log Status Structure	I added this method so I could have an app ask the log for its status

# GSLog Developer Guide

## GSLog.CustomDBLog

### Properties

Name	Data Type	Description
conn_str	String	Please DBLog reference for details.
Connection	OleDbConnection	Please DBLog reference for details.
CategoryPrefix	String	Please DBLog reference for details.
AutoEscalate	Boolean	Please DBLog reference for details.
AutoEscalateLevel	Integer	Please DBLog reference for details.
DBType	eDBType Enum	Please DBLog reference for details.
TableNameProcessLog	String	Please DBLog reference for details.
TableNameProcessName	String	Please DBLog reference for details.
LogLevel	Integer	Please DBLog reference for details.
LogLevelSensitivity	eLogLevelSensitivity	Please DBLog reference for details.

### Methods

Name	Return Data Type	Description
(Constructor)		(See explanation at the beginning of this document.)
WriteLog		(See explanation at the beginning of this document.)
Open		(See explanation at the beginning of this document.)
Status	Log Status Structure	I added this method so I could have an app ask the log for its status.

## Supporting Classes

### GSLog.LogStatus

A structure that is returned by the Status Method.

```
public struct LogStatus
{
    public int NumberOfEntriesWritten;
    public int CurrentLogLevel;
    public string CurrentProcessName;
    public string OutputFilePath;
    public int MaxLogFileSize;
    public int MaxLogFilesAllowed;
    public string ConnectionString;
}
```

### GSLog.LogCategory

This is basically a customized Stack. Makes it easy to set a category that includes a hierarchy.

#### Methods

Name	Data Type	Description
Current	String	Returns the current stack as a string appending each stack entry (Category Title) from top to bottom
SetCategory		Takes the "Category Title" as the input parameter and adds it to the top of the stack.
Restore		"Pops" the top entry (Category Title) off of the stack.

### Conclusion

This tool does it all for me when it comes to writing telemetry for Windows programs. It has evolved into a fine tool. I encourage you to spend the time to learn this tool's capabilities. If you do follow that advice, I hope you will discover and hold the same value for GSLog that I do. If you need further explanation or clarification, please reach out to me.

Rob Plates: Genesis Software Corp.

<http://www.gen-e-sys.com>