**Logging Project Documentation**

Purpose

The purpose of the logging library is to generate automatic log files that developers can use as a debugging tool. In cases of program crashes, this can be used to help pinpoint the location of an error.

Methods

The Logging library can be referenced via the Logging.dll file and includes the following public, static methods:

* Save ( )
  + Saves log file under same name to the currently set directory (directory that executed program by default).
  + Used to save log file under the same name but into a new folder after SetLogFolder( ) is called.
* SaveAs ( string <name\_of\_new\_file> )
  + Saves the current log file under a new name.
  + Does not create a new file.
* ToggleTimeStamp ( bool <time\_stamp\_value> )
  + Toggles whether or not the timestamp is included during Write and WriteLine calls.
  + True will include the timestamp and false will exclude the timestamp.
* SetLogFolder ( string <path\_to\_new\_folder\_to\_save\_into> )
* Write (char <value>)
  + Writes specified Unicode character value to log file.
  + Inclusion of a timestamp can be toggled using the ToggleTimeStamp function. If not toggled, timestamp is default set to true.
  + Applies to all Write() functions.
* Write (char[] <buffer>)
  + Writes specified array of Unicode characters to log file.
* Write (decimal <value>)
  + Writes text representation of the specified System.Decimal value to log file.
* Write (int <value>)
  + Writes the text representation of the specified 32-bit signed integer value to the log file.
* Write (long <value>)
  + Writes the text representation of the specified 64-bit signed integer to the log file.
* Write (object <value>)
  + Writes the text representation of the specified object to the log file.
* Write (ulong <value>)
  + Writes the text representation of the specified 64-bit unsigned integer value to the log file.
* Write (uint <value>)
  + Writes the text representation of the specified 32-bit unsigned integer value to the log file.
* Write (float <value>)
  + Writes the text representation of the specified single-precision floating-point value to the log file.
* Write (double <value>)
  + Writes the text representation of the specified double-precision floating-point value to the log file.
* Write (bool <value>)
  + Writes the text representation of the specified Boolean value to the log file.
* Write (string <value>)
  + Writes the string to the log file.
* Write (string <format>, params object[] <args>)
  + Writes the text representation of the specified array of objects to the log file using the specified format information.
* Write (string <format>, object <arg0>)
  + Writes the text representation of the specified object to the log file using the specified format information.
* Write (char[] <buffer>, int <index>, int <count>)
  + Writes the specified subarray of Unicode characters to the log file.
* Write (string <format>, object <arg0>, object <arg1>)
  + Writes the text representation of the specified objects to the log file using the specified format information.
* Write (string <format>, object <arg0>, object <arg1>, object <arg2>)
  + Writes the text representation of the specified objects to the log file using the specified format information.
* Write (string <format>, object <arg0>, object <arg1>, object <arg2>, object <arg3>)
  + Writes the text representation of the specified objects to the log file using the specified format information.
* WriteLine (char <value>)
  + Writes specified Unicode character value, followed by the current line terminator, to log file.
  + Inclusion of a timestamp can be toggled using the ToggleTimeStamp function. If not toggled, timestamp is default set to true.
  + Applies to all Write() functions.
* WriteLine (char[] <buffer>)
  + Writes specified array of Unicode characters, followed by the current line terminator, to log file.
* WriteLine (decimal <value>)
  + Writes text representation of the specified System.Decimal value, followed by the current line terminator, to log file.
* WriteLine (int <value>)
  + Writes the text representation of the specified 32-bit signed integer value, followed by the current line terminator, to the log file.
* WriteLine (long <value>)
  + Writes the text representation of the specified 64-bit signed integer, followed by the current line terminator, to the log file.
* WriteLine (object <value>)
  + Writes the text representation of the specified object, followed by the current line terminator, to the log file.
* WriteLine (ulong <value>)
  + Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the log file.
* WriteLine (uint <value>)
  + Writes the text representation of the specified 32-bit unsigned integer value, followed by the current line terminator, to the log file.
* WriteLine (float <value>)
  + Writes the text representation of the specified single-precision floating-point value, followed by the current line terminator, to the log file.
* WriteLine (double <value>)
  + Writes the text representation of the specified double-precision floating-point value, followed by the current line terminator, to the log file.
* WriteLine (bool <value>)
  + Writes the text representation of the specified Boolean value, followed by the current line terminator, to the log file.
* WriteLine (string <value>)
  + Writes the string, followed by the current line terminator, to the log file.
* WriteLine (string <format>, params object[] <args>)
  + Writes the text representation of the specified array of objects, followed by the current line terminator, to the log file using the specified format information.
* WriteLine (string <format>, object <arg0>)
  + Writes the text representation of the specified object, followed by the current line terminator, to the log file using the specified format information.
* WriteLine (char[] <buffer>, int <index>, int <count>)
  + Writes the specified subarray of Unicode characters, followed by the current line terminator, to the log file.
* WriteLine (string <format>, object <arg0>, object <arg1>)
  + Writes the text representation of the specified objects, followed by the current line terminator, to the log file using the specified format information.
* WriteLine (string <format>, object <arg0>, object <arg1>, object <arg2>)
  + Writes the text representation of the specified objects, followed by the current line terminator, to the log file using the specified format information.
* WriteLine (string <format>, object <arg0>, object <arg1>, object <arg2>, object <arg3>)
  + Writes the text representation of the specified objects, followed by the current line terminator, to the log file using the specified format information.

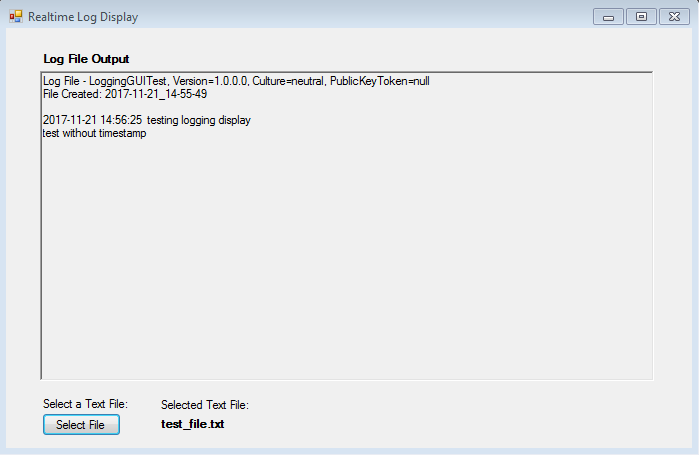
Before the user calls any of these functions, a log file will be default generated with the name “Log\_<time\_of\_creation>” in the directory that the program using the dll was executed from. This is done via a static constructor that is always run prior to any of the static methods.

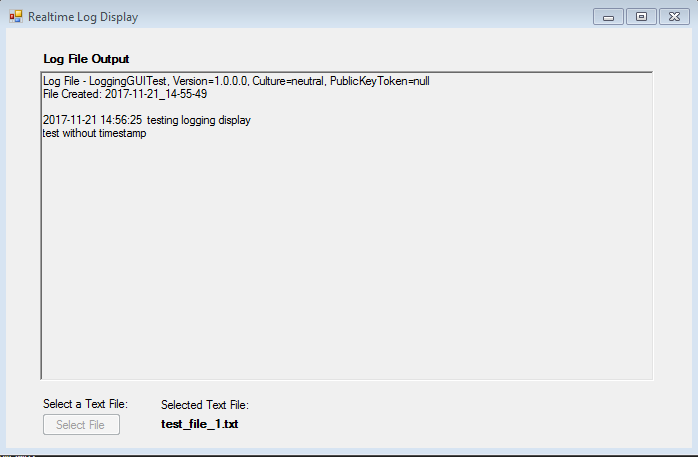
**Logging Display**

A visual real-time logging display is also available as an executable called LoggingDisplay.exe. This allows the user to select a log file to monitor and observe changes in real time. If the application that is writing to the log file crashes, the log monitor will remain active and the user can observe the last log lines before the program crashed.

To monitor changes of the log, a FileSystemWatcher is used to trigger an event every time the log file is written to or saved under a new name. In order to allow two processes to access the same file, I made the FileShare property of the FileStream I used both ReadWrite. When an event is triggered, the text file is read and the contents are copied onto the GUI display. The only issue with this is the fact that FileSystemWatcher does not support tracking file path when saving to another directory. This is because moving files to other directories using File.Move() is done via one create and one delete operation, and therefore FileSystemWatcher is unable to monitor the change. In order to solve this issue, I decided to use NamedPipes.

LoggingDisplay uses a NamedPipeServer and waits for a client (program using Logging.dll) to connect. Once connected, file path changes are sent via the pipe and thus allows for inter process communication. This connection is only created when a user decides to save a file after a LoggingDisplay is already monitoring it. If the user never decides to re-save a file, then a Pipe connection is never made. Once a connection is made, the user can no longer begin monitoring other log files. Prior to the connection, the user can switch between log files to monitor.





When the file is saved after a LoggingDisplay has been connected to it, a connection between the PipeClient of the application and PipeServer of the LoggingDisplay is made. When this happens, you can no longer select new files to monitor.