Log4Delphi Design Overview

Table of contents

1. Brief Overview

Inserting log statements into source code is a low-tech method for debugging it. It may also be the only way since debuggers are not always available or applicable, often the case for distributed applications.

On the other hand, some people argue that log statements pollute source code and decrease legibility. Generally log statements increase the size of the code and reduce its speed, even when logging is turned off. Given that a reasonably sized application may contain thousands of log statements, speed is of particular importance.

With Log4Delphi it is possible to enable logging at runtime without modifying the application binary. The Log4Delphi suite is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and confidence in the application. Logging and testing should not be confused. They are complementary. When logging is wisely used, it can prove to be an essential tool.

The target of the log output can be a console window, file or a TStream object, making it possible to send log statements to any imaginable destination, even to a remote log server!

2. Design

Based on the above overview, Log4Delphi is a Native 32 bit Borland Delphi package that acts as a logging suite for Delphi applications. It is an object-oriented package that uses Delphi standard practices to ensure that the suite is easily integrated into new and existing applications. As such, each class is defined in its own unit, with each unit named after the class with a `Unit` suffix, for example: `TAppenderUnit`. Standard object oriented principles are used as well as a number of well chosen and obvious design patterns.

The suite is based on five core components, namely: Loggers, Levels, Appenders, Layouts and LoggingEvents. Loggers have the responsibility of actually performming the logging and they provide the interface that application developers will use to use the logging suite. Levels represent different levels of logging that may be performed. Appenders are responsible for actually appending a logging message to the target, for example a file. Layouts are used to format logging messages. Logging events represent the actual event that should be logged.

A typical sequence that occurs when an application sends a message to the logging suite is as follows:

1. A logger is selected to perform logging. It checks its assigned logging level to see if it may log the message.
2. If the logger may log the message, it creates a logging event and sends the event to all its appenders.
3. Each appender checks its assigned logging level to see if it may log the event.
4. If the appender may log the event it passes the event to its assigned layout to format the event as a message.
5. The appender then appends the formatted message to its target.

This may be represented by the following diagram. [Click to view full size]

[Click to view full size]

3. UML

The UML Class Diagram for the project is given below. It only includes the core components of the suite and is designed to give an overview of the design of the suite. [Click to view full size]

[Click to view full size]