Building a Universal Windows Platform (UWP) Application (Part 4) – Logging w/ MetroLog

by **admin** | Nov 17, 2016

I just rolled off a client recently that needed to build a Universal Windows Platform (UWP) application as part of their hardware and software solution. For those who aren't yet familiar with UWP, you can check out this article by Tyler Whitney.

As many of you developers out there are aware, sometimes you have to build or bring with you a number of application infrastructure items before you can even get started with the core application logic. For example, you might need some helpers, services and base classes that make your job easier or allow you to start with your base patterns, such as, MVC, MVVM, etc.

So, I have decided to create a series of posts that build up a bunch of common, important parts of an application that you might want to have in place before you even start developing your core functionality. I have discussed a number of topics in my previous posts:

Building a Universal Windows Platform (UWP) Application (Part 1) – Using Template10

Building a Universal Windows Platform (UWP) Application (Part 2) – T4 and Strings

Building a Universal Windows Platform (UWP) Application (Part 3) – Multilingual Support

If you haven't already read the previous posts, I recommend you do since they all build on each other.

Every developer knows the pain of distributing their application to someone who isn't running it in debug mode in Visual Studio and is having a problem with the software. "Oh, I haven't done anything.", "It used to work!" or "This software just isn't working." are common statements that we hear, right?

Today, we are going to discuss how we can add logging support to our application so that we can get better information from a user to help find and fix any issues that they are experiencing.

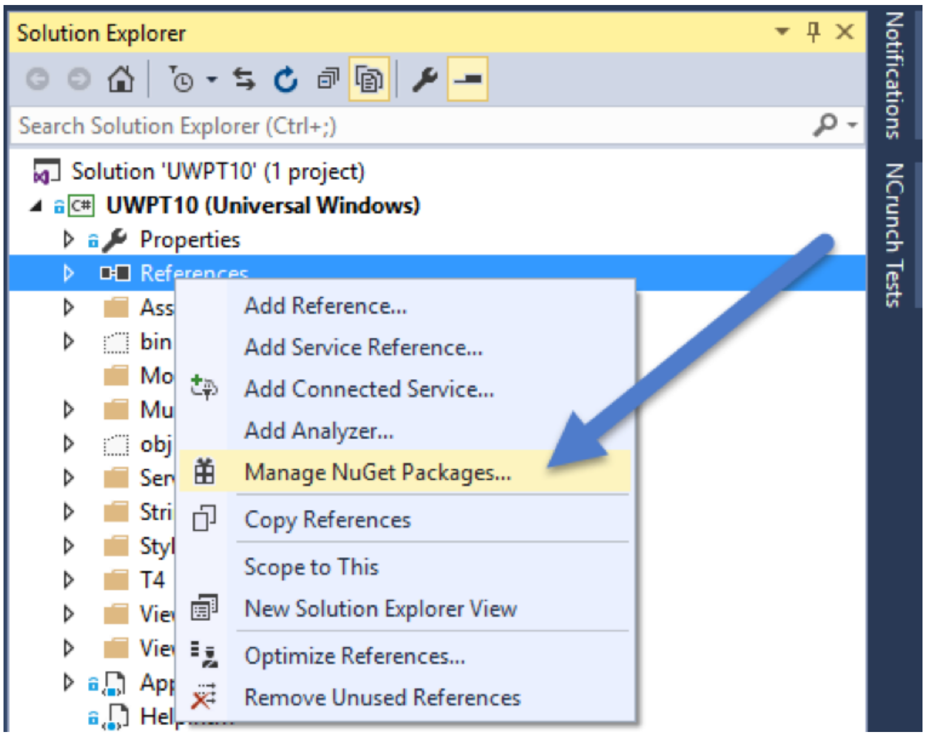
How To Add MetroLog

We could roll our own logging system, but why when you can use another third party system? Okay, I know why you might want to roll your own so that you have complete control over the code. You can add features, etc. Correct? Usually.

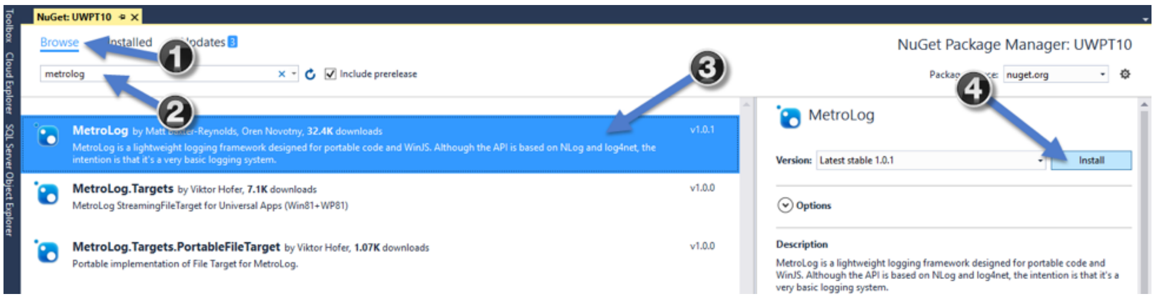
However, for this post, I am going to use a NuGet package called MetroLog to implement some very basic logging. If you want to learn more about this package, you can simply go to their project in GitHub.

But for now, we are just going to install the NuGet package and use the library.

First, **right click** on your projects **References** and **select Manage NuGet Packages…**.



Now, within the NuGet screen, **Browse** for **metrolog**, **select** the **MetroLog** package from the list and then **select Install**.



metrolog

[✓] Include prerelease

Package source: nuget.org

MetroLog by Matt Baxter-Reynolds, Oren Novotny, **32.4K** downloads v1.0.1
MetroLog is a lightweight logging framework designed for portable code and WinJS. Although the API is based on NLog and log4net, the intention is that it's a very basic logging system.

MetroLog.Targets by Viktor Hofer, **7.1K** downloads v1.0.0
MetroLog StreamingFileTarget for Universal Apps (Win81+WP81)

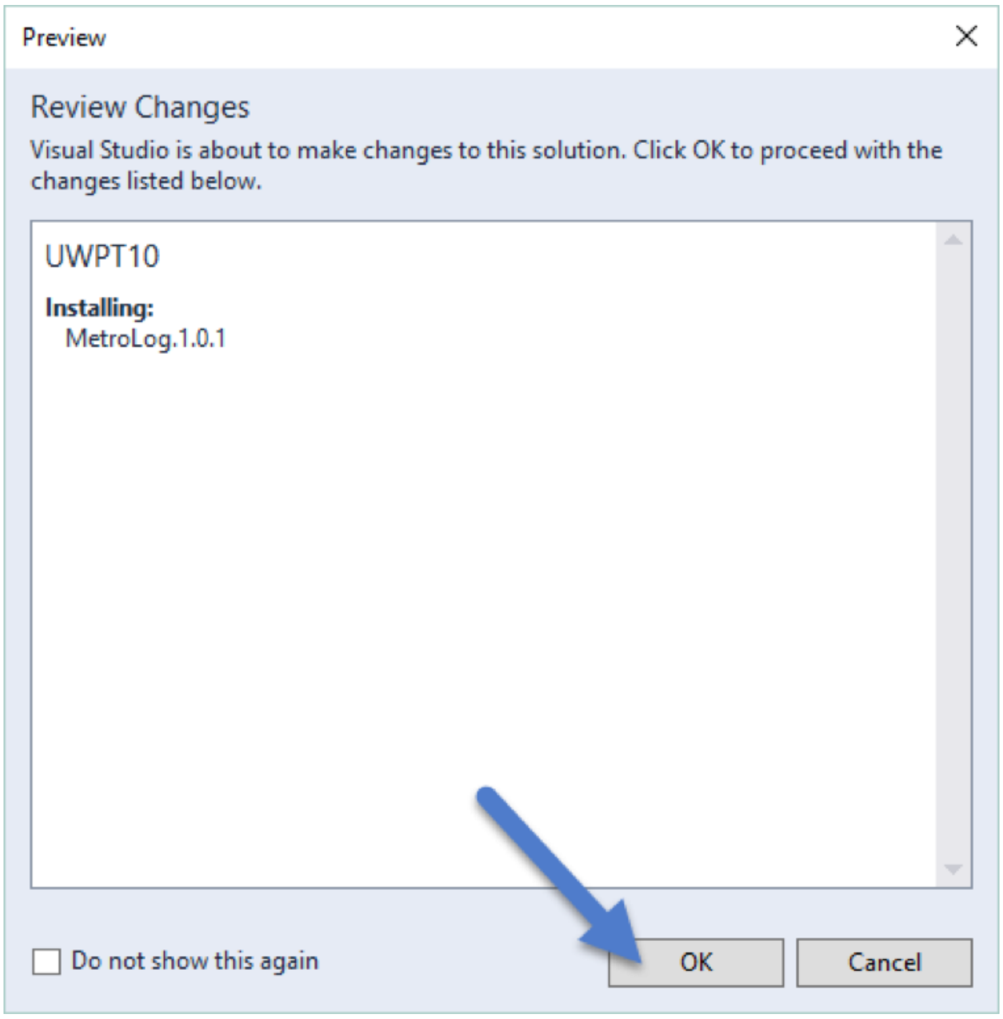
MetroLog.Targets.PortableFileTarget by Viktor Hofer, **1.07K** downloads v1.0.0
Portable implementation of File Target for MetroLog.

MetroLog

Version: Latest stable 1.0.1

Description
MetroLog is a lightweight logging framework designed for portable code and WinJS. Although the API is based on NLog and log4net, the intention is that it's a very basic logging system.

A dialog will be presented asking for you to review the changes that are going to be made to your project. **Select OK**.



Review Changes

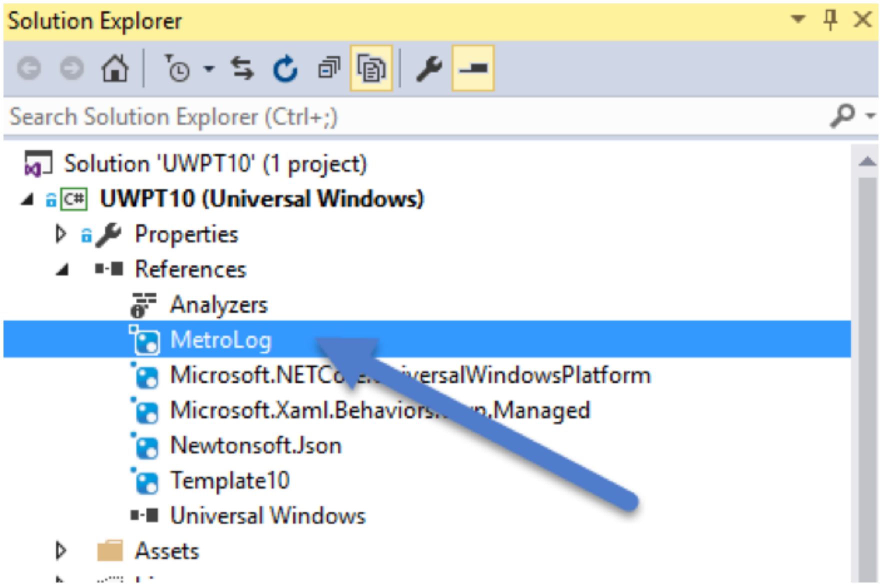
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

UWPT10

Installing:
MetroLog.1.0.1

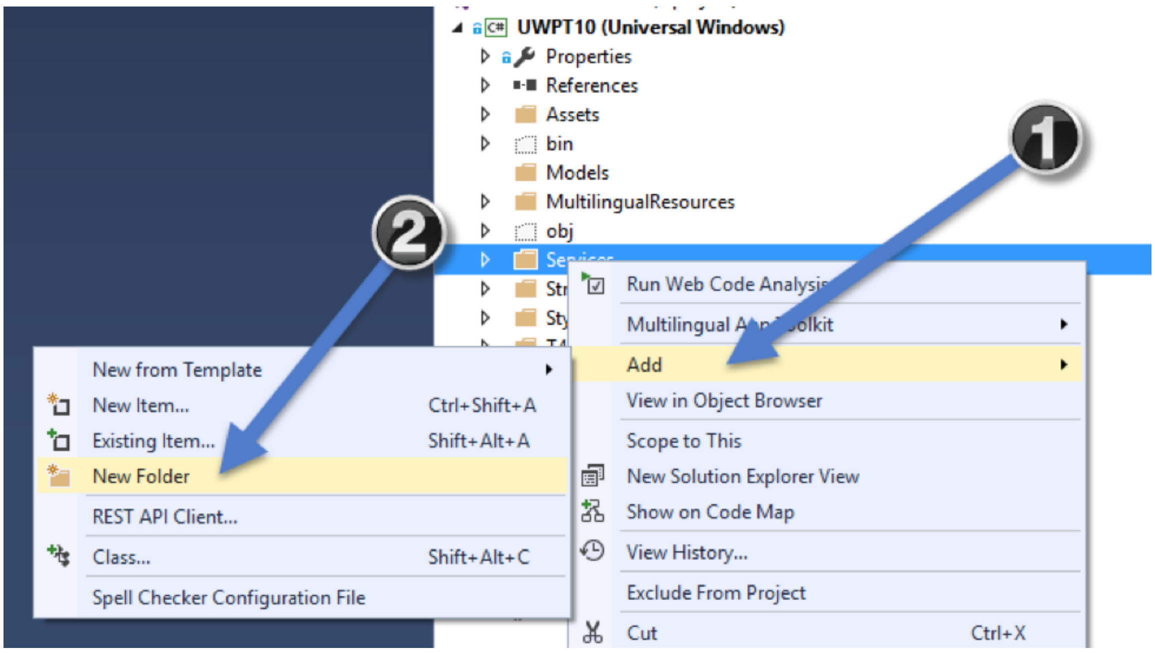
☐ Do not show this again

Once it is installed, you will see a reference to it.

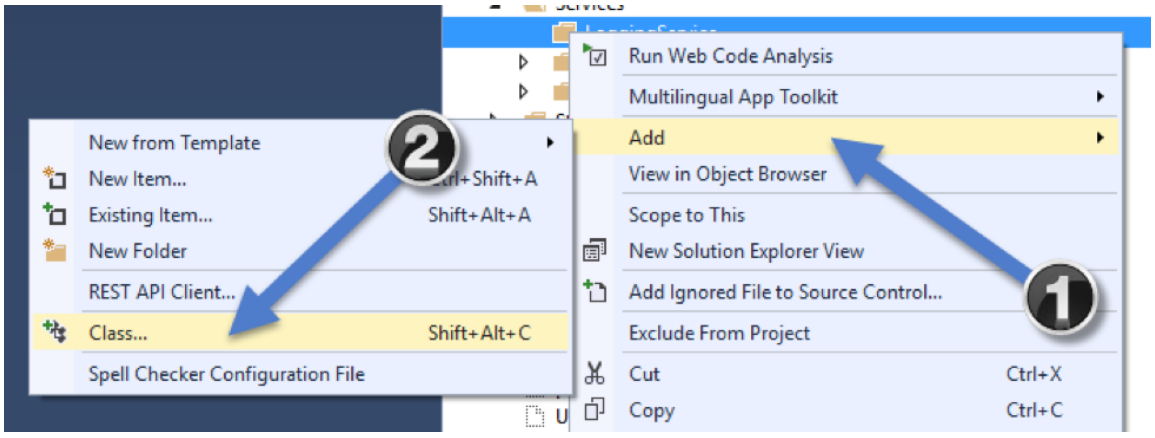

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'UWPT10' (1 project)
UWPT10 (Universal Windows)
Properties
References
Analyzers
MetroLog
Microsoft.NETCore.UniversalWindowsPlatform
Microsoft.Xaml.Behaviors.Uwp.Managed
Newtonsoft.Json
Template10
Universal Windows
Assets

Now, let's create our service that we will use in our application.

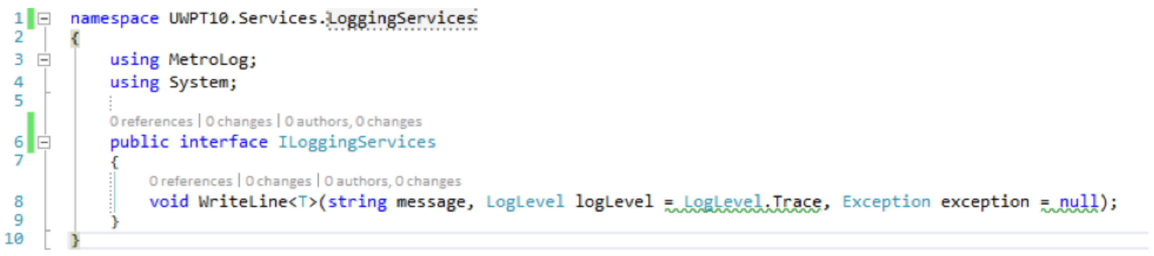
First, create a new folder by **right-clicking** on the **Services directory, select Add, select New Folder** and **name** the **folder LoggingServices**.



In our new folder, let's create a couple of class files, by **right-clicking** on the **LoggingServices** folder, **selecting Add** and then **selecting Class...** Create two classes called: **ILoggingServices.cs** and **LoggingServices.cs**



Open up **ILoggingServices.cs** and let's create an interface that has a simple method called WriteLine that accepts a string message, a log level and any exception information that we have captured. It will probably look something like this:



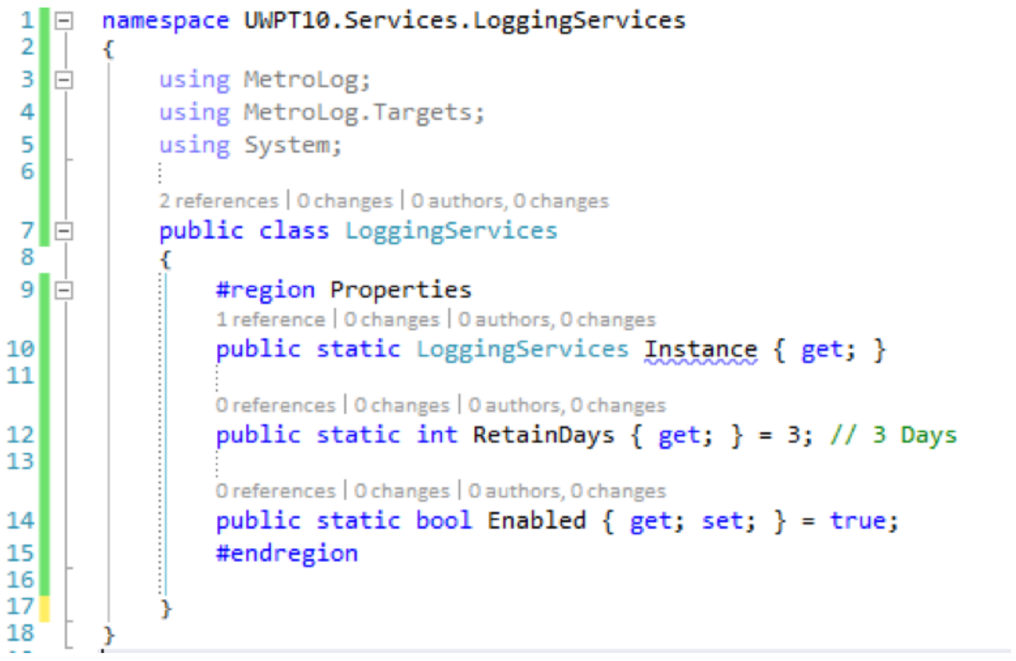
```
namespace UWPT10.Services.LoggingServices
{
    using MetroLog;
    using System;

    0 references | 0 changes | 0 authors, 0 changes
    public interface ILoggingServices
    {
        0 references | 0 changes | 0 authors, 0 changes
        void WriteLine<T>(string message, LogLevel logLevel = LogLevel.Trace, Exception exception = null);
    }
}
```

LogLevel is defined in the MetroLog library, so don't worry about that for now. It basically just sets the level at which this message should be logged.

Now, let's open the *LoggingServices.cs* and add some code there. Here is a list of properties that I would like to add to the LoggingServices class:

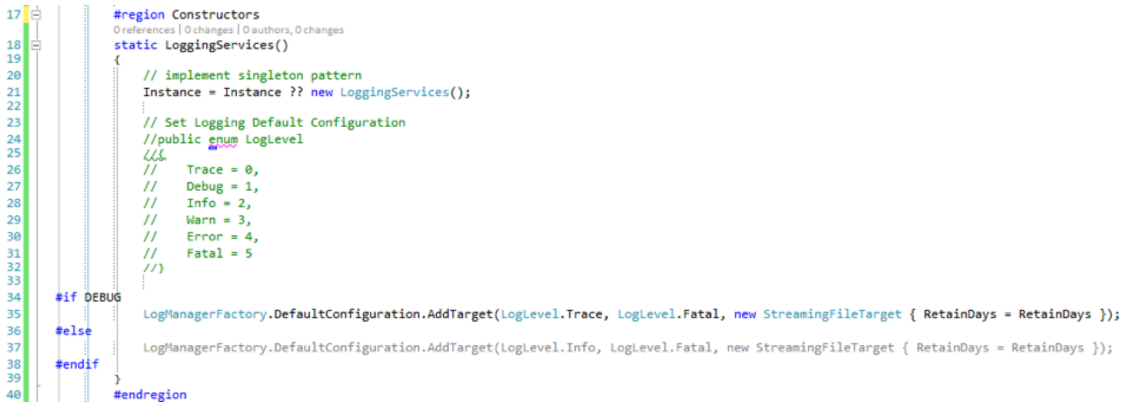
- *Instance* – for implementation of a singleton pattern.
- *RetainDays* – the number of days the logs will remain on the file system. This allows us to keep the footprint pretty minimal. Well, depending on the size of each log file, I guess.
- *Enabled* – this property can be used to turn on and off logging all together.

It might look something like this:



```
namespace UWPT10.Services.LoggingServices
{
    using MetroLog;
    using MetroLog.Targets;
    using System;

    2 references | 0 changes | 0 authors, 0 changes
    public class LoggingServices
    {
        #region Properties
        1 reference | 0 changes | 0 authors, 0 changes
        public static LoggingServices Instance { get; }

        0 references | 0 changes | 0 authors, 0 changes
        public static int RetainDays { get; } = 3; // 3 Days

        0 references | 0 changes | 0 authors, 0 changes
        public static bool Enabled { get; set; } = true;
        #endregion

    }
}
```

Next, let's create our constructor.

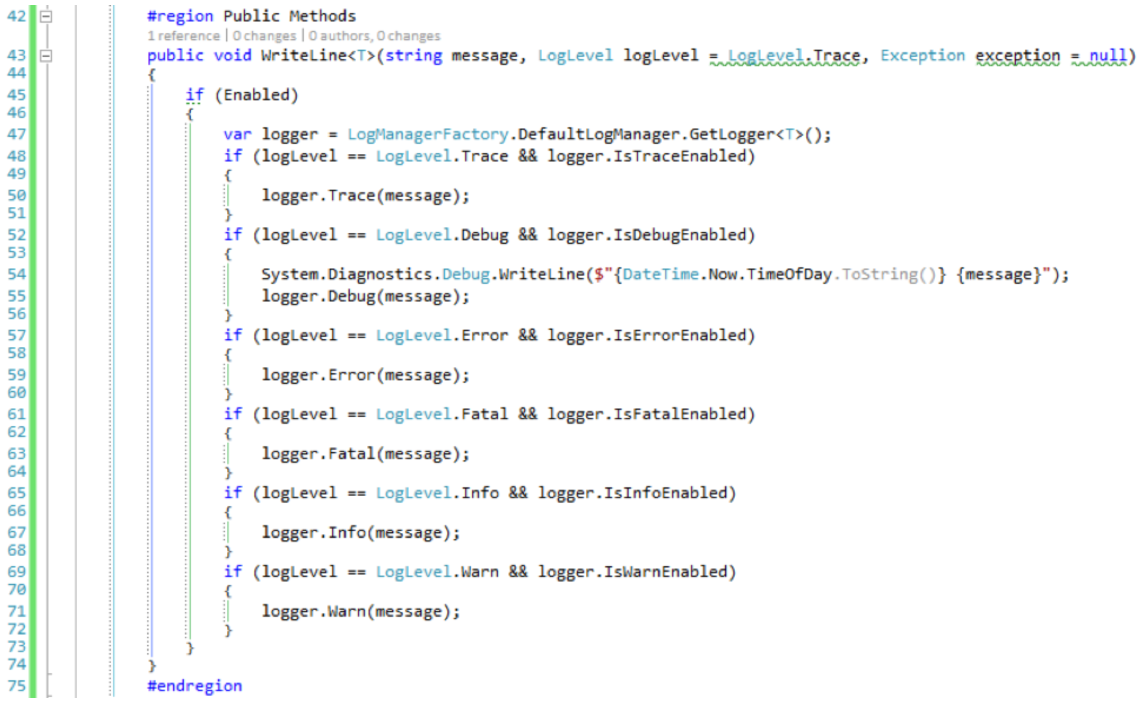


```
#region Constructors
0 references | 0 changes | 0 authors, 0 changes
static LoggingServices()
{
    // implement singleton pattern
    Instance = Instance ?? new LoggingServices();

    // Set Logging Default Configuration
    //public enum LogLevel
    //{
    //    Trace = 0,
    //    Debug = 1,
    //    Info = 2,
    //    Warn = 3,
    //    Error = 4,
    //    Fatal = 5
    //}

#if DEBUG
    LogManagerFactory.DefaultConfiguration.AddTarget(LogLevel.Trace, LogLevel.Fatal, new StreamingFileTarget { RetainDays = RetainDays });
#else
    LogManagerFactory.DefaultConfiguration.AddTarget(LogLevel.Info, LogLevel.Fatal, new StreamingFileTarget { RetainDays = RetainDays });
#endif
}
#endregion
```

The very first thing we do is implement the constructor piece of our singleton pattern by initializing our Instance. Second, I thought it might be helpful for you to see the enum that is defined for logging levels for the MetroLog library. Basically, Trace is the lowest level and Fatal is the highest level.

As you can see, we can configure the MetroLog library to log everything if we are in Debug mode and only levels Info through Fatal for a release version of the application. This will keep it simple for us.

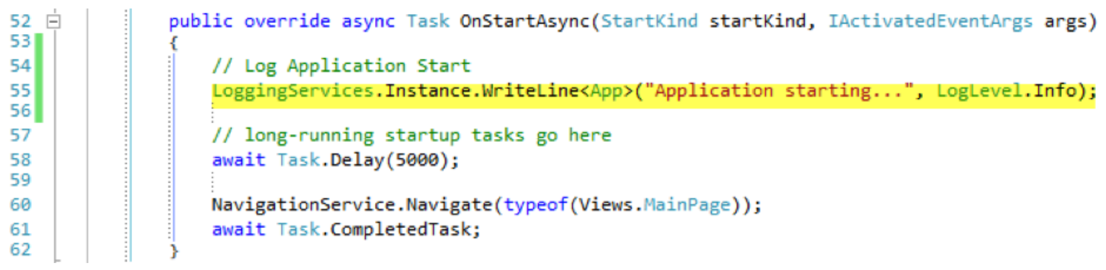
Okay, now, let's add the interface method to write the message to the log:



```
#region Public Methods
1 reference | 0 changes | 0 authors, 0 changes
public void WriteLine<T>(string message, LogLevel logLevel = LogLevel.Trace, Exception exception = null)
{
    if (Enabled)
    {
        var logger = LogManagerFactory.DefaultLogManager.GetLogger<T>();
        if (logLevel == LogLevel.Trace && logger.IsTraceEnabled)
        {
            logger.Trace(message);
        }
        if (logLevel == LogLevel.Debug && logger.IsDebugEnabled)
        {
            System.Diagnostics.Debug.WriteLine($"{DateTime.Now.TimeOfDay.ToString()} {message}");
            logger.Debug(message);
        }
        if (logLevel == LogLevel.Error && logger.IsErrorEnabled)
        {
            logger.Error(message);
        }
        if (logLevel == LogLevel.Fatal && logger.IsFatalEnabled)
        {
            logger.Fatal(message);
        }
        if (logLevel == LogLevel.Info && logger.IsInfoEnabled)
        {
            logger.Info(message);
        }
        if (logLevel == LogLevel.Warn && logger.IsWarnEnabled)
        {
            logger.Warn(message);
        }
    }
}
#endregion
```

You will notice that I have added the WriteLine<T> method that accepts a string, a LogLevel and exception. If the logging service is enabled, it will check if the logging level passed in is enabled and then attempt to log the information type. Most all of the levels are the same structure, except that for the LogLevel.Debug. I decided to add a Debug.WriteLine so that I could see some of the debug messages in my console window in Visual Studio. Pretty simple huh?

Now, how do we use this service? Well, for this post, let's just add a simple log entry each time we start the application. To do this, let's **open App.xaml.cs**. Now, **go** down **to** the **OnStartAsync** method. At the top of the function, let's add our LoggingServices call.

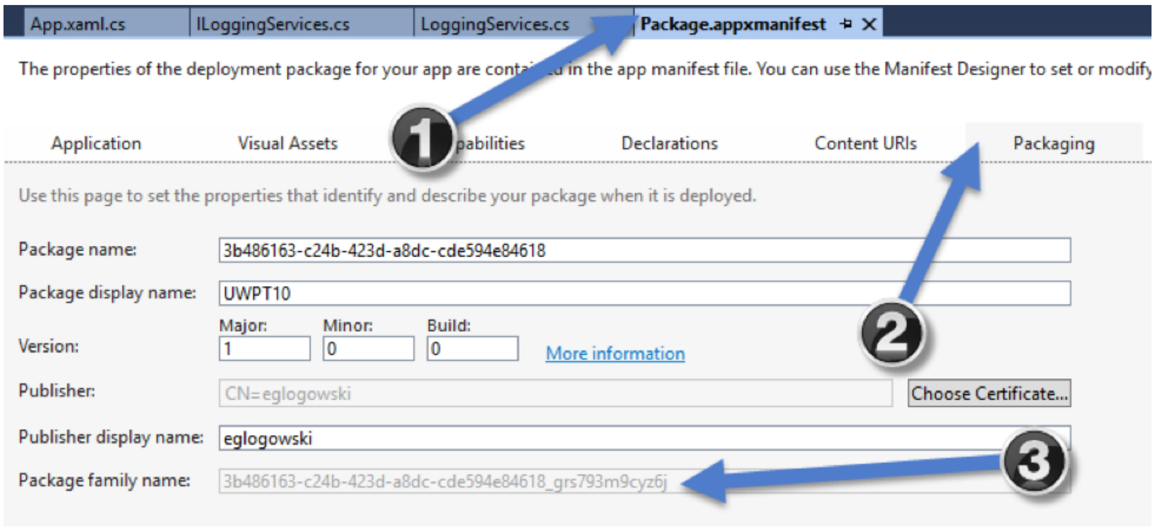


```
public override async Task OnStartAsync(StartKind startKind, IActivatedEventArgs args)
{
    // Log Application Start
    LoggingServices.Instance.WriteLine<App>("Application starting...", LogLevel.Info);

    // long-running startup tasks go here
    await Task.Delay(5000);

    NavigationService.Navigate(typeof(Views.MainPage));
    await Task.CompletedTask;
}
```

Run the application. The log should have been written to with our "Application starting…" string. Let's take a look at it. By default, MetroLog saves the log to the following **location**:

C:\Users\[login]\AppData\Local\Packages\[package family name]\LocalState\MetroLogs

Where *[login]* is the name of the logged in user account and *[package family name]* is the ID generated for the application. How do you find the *[package family name]*? If you **open** the application manifest, **Package.appxmanifest**, you will see the **Package family name** at the **bottom** of the **Packaging tab**.

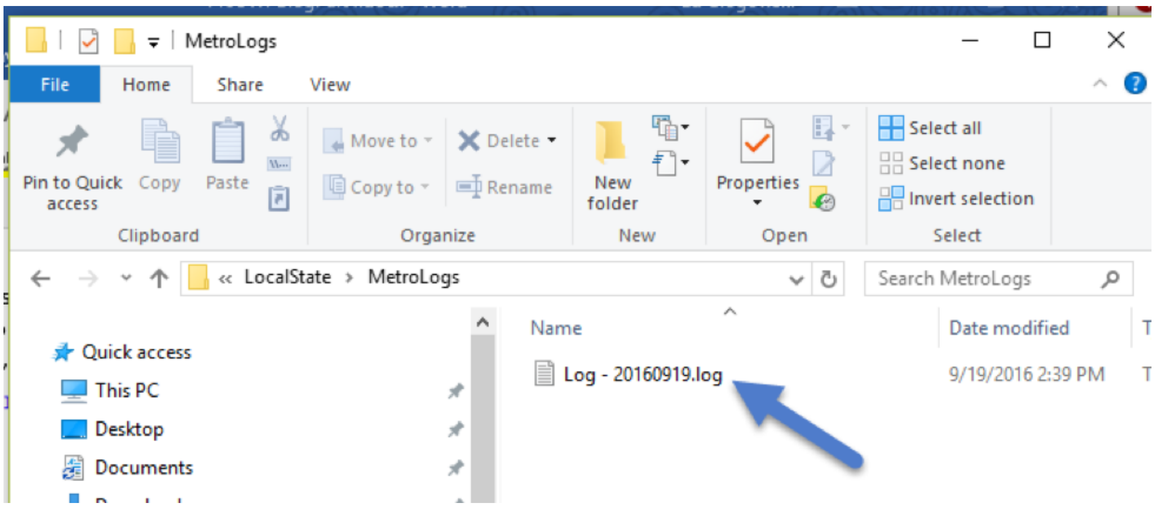


The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify

Use this page to set the properties that identify and describe your package when it is deployed.

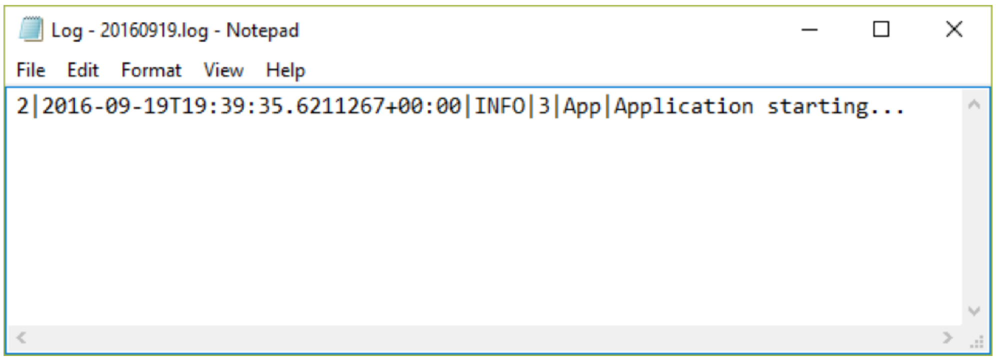
Package name:	3b486163-c24b-423d-a8dc-cde594e84618
Package display name:	UWPT10
Version:	Major: 1 Minor: 0 Build: 0
Publisher:	CN=eglogowski
Publisher display name:	eglogowski
Package family name:	3b486163-c24b-423d-a8dc-cde594e84618_grs793m9cyz6j

If you **go to the folder**, you should see the log files available.



Log - 20160919.log 9/19/2016 2:39 PM

And, if you **open the log**, it should look like this…



```
2|2016-09-19T19:39:35.6211267+00:00|INFO|3|App|Application starting...
```

If you want to tweak how the log file looks, I think there are ways to define the format. You can check that out by hitting up the site.

NOTE: The Template10 library has a very basic logging service built in too called LoggingService. We could probably rename ours to make it easier to differentiate, but that is up to you. For now, I will keep ours as LoggingServices.

Now, you can add logging statements to all your code. Remember, the log entries that get written out to the log depends on the DefaultConfiguration defined on startup.

What We Accomplished

So, to summarize what we accomplished in this blog post:

- Added the MetroLog NuGet package to project
- Added a Logging Service to our services
- Demonstrated how to add a call to the logging service
- How we can view the logs created from the logging service

In part 5 of this series, I will add some analytics service to our bag of services. This will allow us to keep track of how users are using the system. This can be very useful when trying to analyze issues with the software and how it is being used. This data can then help us to focus and prioritize our efforts on building and maintaining our software based on how it is used.

I hope you enjoyed this post. If so, please share it with others.

1 Comment **Intertech Blog** 1 **Login**

Recommend 1 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

dbnex B • 4 months ago

Great article about MetroLog. I have a question though. How about cross platform support for it in Xamarin.Forms projects. Is it supported for Android and iOS as well?
I read it is PCL friendly but I don't see any explanation and examples how to do it. Thanks for great articles, I read them all 4. Great work, thanks for sharing!

Reply • Share ›

Get Our Secret to Good Code Documentation Guide

Subscribe to our blog and gain access to our guide developed by our consulting teams.

Some ad blockers can block the form below.

Email

Subscribe

Recent Posts

Thousands of Passengers Stranded, Windows on Git, and Much More…

Building Full Stack Development Skills on Your Team

DDoS-ing GitHub, Google's ML Training, and More…

Top Trends in Javascript for 2018 and Beyond

SpaceX's Software, Email Overwhelm, and More…

Categories

Categories

Select Category ▼

Practical insights, tips, tutorials and examples from team of software development consultants and trainers.

Recent Posts

- Thousands of Passengers Stranded, Windows on Git, and Much More…
- Building Full Stack Development Skills on Your Team
- DDoS-ing GitHub, Google's ML Training, and More…
- Top Trends in Javascript for 2018 and Beyond
- SpaceX's Software, Email Overwhelm, and More…

Contact Details

1575 Thomas Center Drive
Eagan, MN 55122
General: 651.288.7000

Training: 651-288-7109
Consulting: 651-288-7001

Toll Free: 800-866-9884

About Intertech

Home
Consulting
IntertechU Course Selector