# Getting Started

## The Basics

The simplest possible example using the ubiquitous “Hello World”:

**Hello.cs**

#define TRACE

using System.Diagnostics;

class Program {

static TraceSource \_trace = new TraceSource("Hello");

public static void Main() {

\_trace.TraceInformation("Hello World!");

}

}

**Hello.exe.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.diagnostics>

<sources>

<source name="Hello" switchValue="All">

<listeners>

<add name="console"

type="System.Diagnostics.ConsoleTraceListener" />

</listeners>

</source>

</sources>

</system.diagnostics>

</configuration>

Compile with “csc Hello.cs”, and run, you should get the following output:

PS C:\Microsoft.Diagnostics\Examples> .\Hello.exe

Hello Information: 0 : Hello World!

Note that instead of “#define TRACE” you would usually compile as “csc Hello.cs /d:TRACE”, which is turned on by default in Visual Studio.

# Logging Primer

A simple “Hello World” isn’t however very useful for showing the different capabilities of logging, so we will use a slightly more complicated example. First of all, here is the program we will use both without and then with simple logging.

* Hello World (no logging)
* Hello Logging

Once the program has had logging statements added, any of the trace listeners and filters can be configured to send the trace output to a wide variety of destinations.

* Service Trace Viewer
* Windows Event Log

## Extending System.Diagnostics

The best thing is – all of the logging examples above can be done right now – they are already part of the .NET Framework that you are using – and they are designed to be extended.

So, as well as teaching you about what you already have, this project provides some extensions to the features already available in .NET – for example a trace listener that logs to a database or writes to the console in color. These extensions are clearly marked with the extension symbol where they appear in the documentation.

For example, here are examples of some of the extended listeners provided.

* Hello Color [EX]
* Hello Database [EX]

# Hello World (no logging)

This version of “Hello World” involves a bunch of Worker classes that Poke() each other to say “Hello World”. Sometimes they get sick of being poked.

**HelloWorld.cs – Header**

using System;

using System.Collections.ObjectModel;

using System.Threading;

**HelloWorld.cs – Program class**

class Program {

public static Random Random = new Random();

public static Collection<Worker> Workers = new Collection<Worker>();

public static void Main(string[] args) {

int numberOfWorkers = Program.Random.Next(2, 4);

for(int i = 1; i <= numberOfWorkers; i++ ) {

Worker worker = new Worker() { Id = string.Format("Worker {0}", i) };

Workers.Add(worker);

}

StartWorkers();

foreach(Worker worker in Workers) {

worker.FinishedEvent.WaitOne();

}

}

static void StartWorkers() {

foreach(Worker worker in Workers) {

ThreadPool.QueueUserWorkItem(worker.Work);

}

}

}

**HelloWorld.cs – Worker class**

class Worker {

int \_count;

public AutoResetEvent FinishedEvent = new AutoResetEvent(false);

public string Id;

public void Poke() {

Thread.Sleep(Program.Random.Next(500));

\_count++;

if( \_count < 4 )

Console.WriteLine("Hello World {1}", Id, \_count);

else if( \_count < 6 ) {

Console.WriteLine("Hi", Id);

}

}

public void Work(object state) {

int numberOfPokes = Program.Random.Next(3, 7);

for(int i = 1; i < numberOfPokes; i++) {

Thread.Sleep(Program.Random.Next(500));

int index = Program.Random.Next(Program.Workers.Count);

Program.Workers[index].Poke();

}

FinishedEvent.Set();

}

}

Compiling and running this program may produce the following:

PS C:\Microsoft.Diagnostics\Examples> .\HelloWorld.exe

Hello World 1

Hello World 2

Hello World 1

Hello World 3

Hi

Hello World 2

Hello World 1

Hi

Hello World 3

Hello World 2

Lots of “Hello World”, but a bit difficult to tell which bit of code did what.

# Hello Logging

Let’s introduce some logging into our application, and we can see what it can do.

Note that this code is intended to demonstrate logging and does not necessarily follow best practices in order to keep things simple (e.g. normally you would use properties instead of public fields). Also there is a lot of logging code because even though the example is short it tries to include many of the logging features.

**HelloLogging.cs – Header**

using System;

using System.Collections.ObjectModel;

using System.Diagnostics;

using System.Threading;

**HelloLogging.cs – Program class**

class Program {

static TraceSource \_trace = new TraceSource("HelloProgram");

public static Random Random = new Random();

public static Collection<Worker> Workers = new Collection<Worker>();

public static void Main(string[] args) {

// Trace start

Trace.CorrelationManager.ActivityId = Guid.NewGuid();

Trace.CorrelationManager.StartLogicalOperation("Main");

\_trace.TraceEvent(TraceEventType.Start, 1000, "Program start.");

// Run program

int numberOfWorkers = Program.Random.Next(2, 4);

\_trace.TraceEvent(TraceEventType.Information, 2000, "Creating {0} workers", numberOfWorkers);

for(int i = 1; i <= numberOfWorkers; i++ ) {

Worker worker = new Worker() { Id = string.Format("Worker {0}", i) };

Workers.Add(worker);

}

StartWorkers();

foreach(Worker worker in Workers) {

worker.FinishedEvent.WaitOne();

}

// Trace stop

\_trace.TraceEvent(TraceEventType.Stop, 8000, "Program stop.");

Trace.CorrelationManager.StopLogicalOperation();

\_trace.Flush();

}

static void StartWorkers() {

// Trace transfer in

Guid newActivity = Guid.NewGuid();

\_trace.TraceTransfer(6011, "Transferred to Start", newActivity);

Guid oldActivity = Trace.CorrelationManager.ActivityId;

Trace.CorrelationManager.ActivityId = newActivity;

\_trace.TraceEvent(TraceEventType.Start, 1010, "Starting workers.");

// Do work

foreach(Worker worker in Workers) {

ThreadPool.QueueUserWorkItem(worker.Work);

}

// Trace transfer back

\_trace.TraceTransfer(6012, "Transferred back", oldActivity);

\_trace.TraceEvent(TraceEventType.Stop, 8010, "Finished starting.");

Trace.CorrelationManager.ActivityId = oldActivity;

}

}

**HelloLogging.cs – Worker class**

class Worker {

int \_count;

static TraceSource \_trace = new TraceSource("HelloWorker");

public AutoResetEvent FinishedEvent = new AutoResetEvent(false);

public string Id;

public void Poke() {

// Trace - mark with logical operation

Trace.CorrelationManager.StartLogicalOperation(string.Format("Poked:{0}", Id));

\_trace.TraceEvent(TraceEventType.Verbose, 0, "Worker {0} was poked", Id);

// Work

Thread.Sleep(Program.Random.Next(500));

\_count++;

if( \_count < 4 )

Console.WriteLine("Hello World {1}", Id, \_count);

else if( \_count < 6 ) {

Console.WriteLine("Hi", Id);

\_trace.TraceEvent(TraceEventType.Warning, 4500, "Worker {0} getting annoyed", Id);

} else {

\_trace.TraceEvent(TraceEventType.Error, 5500, "Worker {0} - too many pokes", Id);

}

// Trace - end logical operation

Trace.CorrelationManager.StopLogicalOperation();

}

public void Work(object state) {

// Trace transfer to thread

Guid newActivity = Guid.NewGuid();

\_trace.TraceTransfer(6501, "Transfered to worker", newActivity);

Trace.CorrelationManager.ActivityId = newActivity;

Trace.CorrelationManager.StartLogicalOperation(string.Format("Worker:{0}", Id));

\_trace.TraceEvent(TraceEventType.Start, 1500, "Worker {0} start.", Id);

// Do work

int numberOfPokes = Program.Random.Next(3, 7);

\_trace.TraceEvent(TraceEventType.Information, 2500, "Worker {0} will poke {1} times", Id, numberOfPokes);

for(int i = 1; i < numberOfPokes; i++) {

Thread.Sleep(Program.Random.Next(500));

int index = Program.Random.Next(Program.Workers.Count);

\_trace.TraceEvent(TraceEventType.Verbose, 0, "Worker {0} poking {1}", Id, index);

Program.Workers[index].Poke();

}

FinishedEvent.Set();

// Trace stop (no transfer)

\_trace.TraceEvent(TraceEventType.Stop, 8500, "Worker stop.");

Trace.CorrelationManager.StopLogicalOperation();

}

}

For logging you also need a configuration file:

**HelloLogging.exe.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.diagnostics>

<sources>

<source name="HelloProgram" switchValue="Information,ActivityTracing">

<listeners>

<add name="console" />

</listeners>

</source>

<source name="HelloWorker" switchValue="All">

<listeners>

<add name="console" />

</listeners>

</source>

</sources>

<sharedListeners>

<add name="console"

type="System.Diagnostics.ConsoleTraceListener" />

</sharedListeners>

</system.diagnostics>

</configuration>

Now, compile with “csc HelloLogging.cs /d:TRACE” – don’t forget the TRACE flag, or you won’t get any logging – and run.

With the basic console logger the information may not add much clarity, however the level of detail can be controlled by simply altering the configuration file without having to recompile.

# Service Trace Viewer

A good tool for viewing log files is the Service Trace Viewer from the .NET SDK (this can normally be found in the Start > Programs menu under the Microsoft Windows SDK > Tools folder).

The best format for this tool is the XmlWriterTraceListener, although it does understand the other XML formats to a limited degree.

Use the following config file, along with the Hello Logging sample program. You do not need to recompile the sample program, using the Service Trace Viewer only requires changes to the application config file.

**HelloLogging.exe.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.diagnostics>

<sources>

<source name="HelloProgram" switchValue="Information,ActivityTracing">

<listeners>

<add name="xml" />

</listeners>

</source>

<source name="HelloWorker" switchValue="All">

<listeners>

<add name="xml" />

</listeners>

</source>

</sources>

<sharedListeners>

<add name="xml"

type="System.Diagnostics.XmlWriterTraceListener"

initializeData="HelloLogging.xml" />

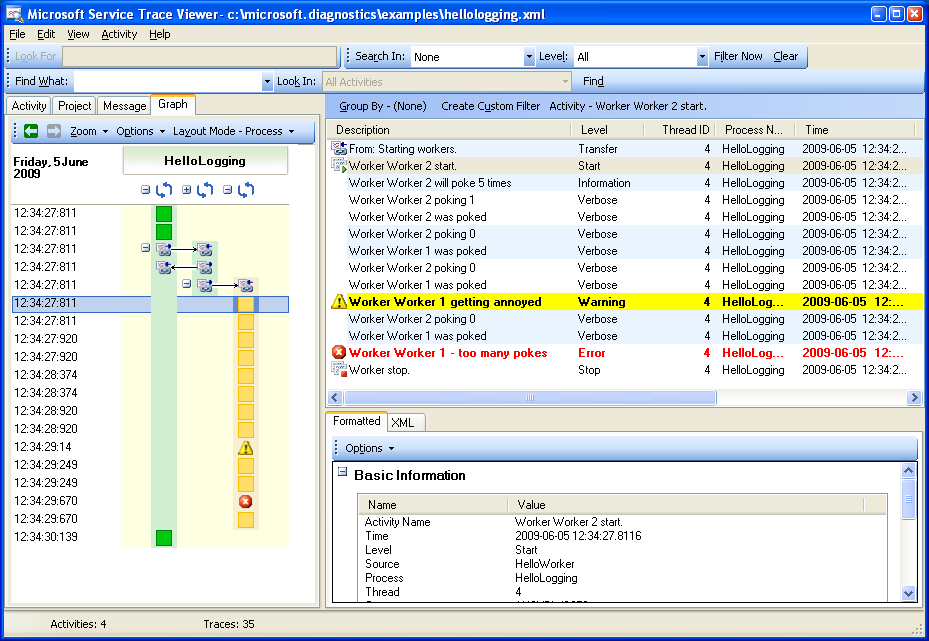
</sharedListeners>

</system.diagnostics>

</configuration>

You should get a file “HelloLogging.xml” created in the example directory – if you don’t, check that the console logging is working (i.e. make sure you compiled with the TRACE flag).

To see your log, run the Service Trace Viewer tool (SvcTraceViewer.exe from the .NET SDK), and open up the log file, you should see log details similar to the following:



The trace viewer provides a graphical overview of how the activities relate to each other and allow you to easily narrow in on any problems in the code.

# Windows Event Log

The Windows Event Log is an important tool for managing Windows based computers, particularly for Windows Services, web applications, and other server based applications that do not have a user interface.

When writing to the Windows Event Log you probably don’t want to write every message, particularly not a large number of verbose messages. On the other hand, you probably want to always report every warning, error, or higher level message.

If you aren’t writing directly to the Windows Event Log, then the EventLogTraceListener can be used to forward appropriate trace events to the Windows Event Log.

**HelloLogging.exe.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.diagnostics>

<sources>

<source name="HelloProgram" switchValue="Information,ActivityTracing">

<listeners>

<add name="console" />

<add name="eventlog" />

</listeners>

</source>

<source name="HelloWorker" switchValue="All">

<listeners>

<add name="console" />

<add name="eventlog" />

</listeners>

</source>

</sources>

<sharedListeners>

<add name="console"

type="System.Diagnostics.ConsoleTraceListener" />

<add name="eventlog"

type="System.Diagnostics.EventLogTraceListener"

initializeData="HelloLogging.xml" />

</sharedListeners>

</system.diagnostics>

</configuration>

When the program is run, messages based on the switchValue settings for individual sources will be sent to both trace listeners. The console trace listener will output all messages, however the event log trace listener will only record warning and above messages.

# Hello Color

The ColoredConsoleTraceListener is a replacement for the standard ConsoleTraceListener that colorizes the output according to the TraceEventType. The listener also has allows trace details to be output according to a user-supplied template.

By default errors are output in red, warnings in yellow and information messages in white. The default template matches the format output by the standard ConsoleTraceListener.

Use the following config file, along with the Hello Logging sample program. You do not need to recompile the sample program.

**HelloLogging.exe.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.diagnostics>

<sources>

<source name="HelloProgram" switchValue="Information,ActivityTracing">

<listeners>

<add name="color" />

</listeners>

</source>

<source name="HelloWorker" switchValue="All">

<listeners>

<add name="color" />

</listeners>

</source>

</sources>

<sharedListeners>

<add name="color"

type="Essential.Diagnostics.ColoredConsoleTraceListener, Essential.Diagnostics"

format="{DateTime} {EventType}: {Message}"

activityTracingColor="DarkGreen"

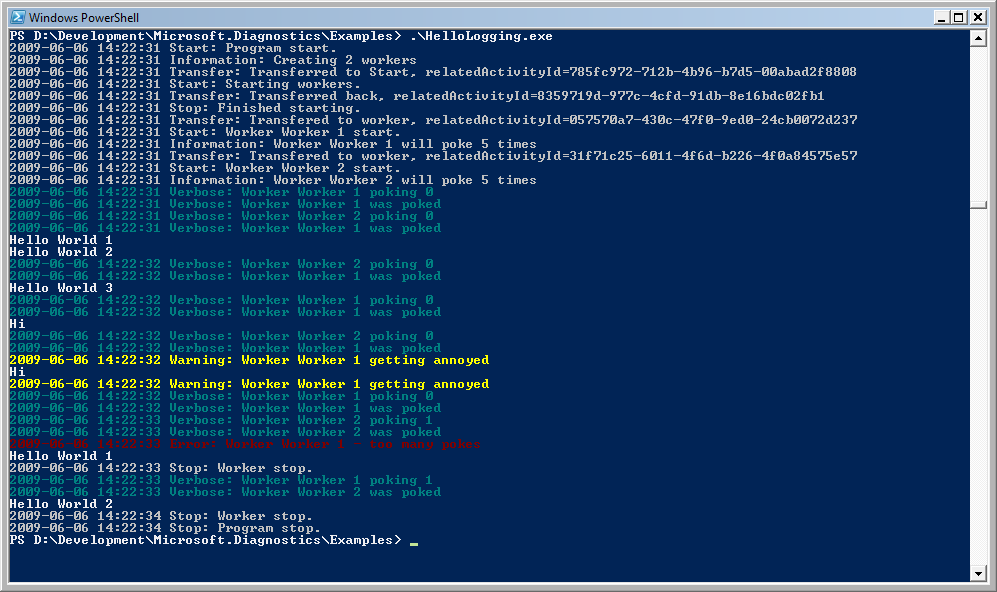
transferColor="Blue" />

</sharedListeners>

</system.diagnostics>

</configuration>

When you run the program, you should get output similar to the following.



Messages are colored and output according to the custom template.