How To Implement A Custom SharePoint 2010 Logging Service For ULS And Windows Event Log

Jürgen Bäurle

January 2011

Parago Media GmbH & Co. KG

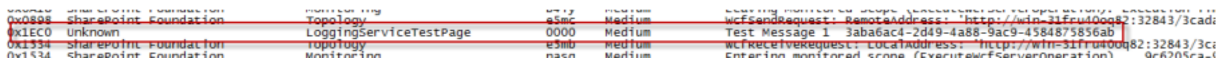
Introduction

Prior to Microsoft SharePoint 2010 there was no official documented way to programmatically use the built-in ULS service (Unified Logging Service) to log own custom messages. There are still solutions available on the internet that can be used, but SharePoint 2010 now supports full-blown logging service support.

To log a message to the SharePoint log files just call the WriteTrace method of the SPDiagnosticsService class:

```
SPDiagnosticsService.Local.WriteTrace(0,
  new SPDiagnosticsCategory("MyCategory", TraceSeverity.Medium, EventSeverity.Information),
  TraceSeverity.Medium, "My log message");
```

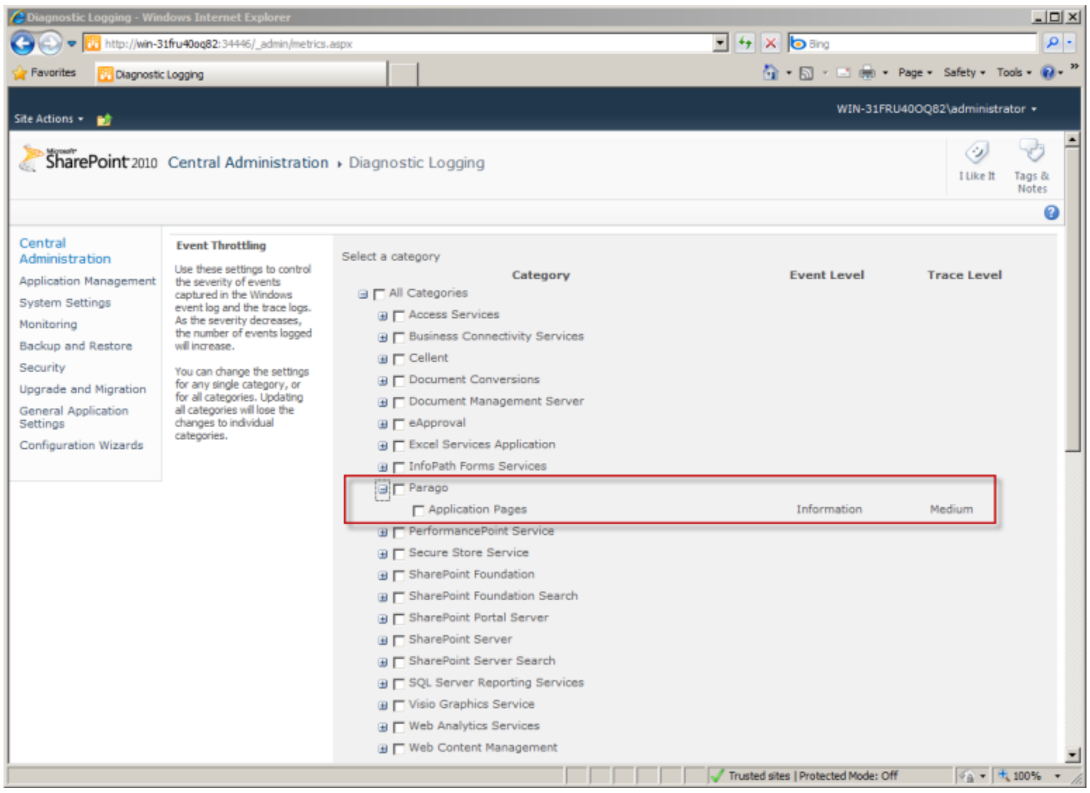
The problem with this technique is that the log entry does not contain any category information, instead SharePoint uses the value "Unknown":



```
0x0898  SharePoint Foundation   Topology                 e5mc  Medium   WcfSendRequest: RemoteAddress: 'http://win-31fru40oq82:32843/3cada
0x1EC0  Unknown                 LoggingServiceTestPage   0000  Medium   Test Message 1  3aba6ac4-2d49-4a88-9ac9-4584875856ab
0x1534  SharePoint Foundation   Topology                 e5mb  Medium   WcfReceiveRequest: LocalAddress: 'http://win-31fru40oq82:32843/3ca
0x1534  SharePoint Foundation   Monitoring               nasq  Medium   Entering monitored scope (ExecuteWcfServerOperation)   9c6205ca-
```

Of course, that does not matter if someone develops small solutions. Developing custom solutions you may want to implement an own logging service with custom categories and UI integration within the Central Administration (CA) of SharePoint 2010.

This article shows how to develop a custom logging service that integrates with the Diagnostic Logging UI of the Central Administration:



Diagnostic Logging - Windows Internet Explorer

http://win-31fru40oq82:34446/_admin/metrics.aspx

Site Actions · WIN-31FRU40OQ82\administrator

SharePoint 2010 Central Administration › Diagnostic Logging

Central Administration
- Application Management
- System Settings
- Monitoring
- Backup and Restore
- Security
- Upgrade and Migration
- General Application Settings
- Configuration Wizards

Event Throttling

Use these settings to control the severity of events captured in the Windows event log and the trace logs. As the severity decreases, the number of events logged will increase.

You can change the settings for any single category, or for all categories. Updating all categories will lose the changes to individual categories.

Select a category

Category	Event Level	Trace Level
All Categories		
Access Services		
Business Connectivity Services		
Cellent		
Document Conversions		
Document Management Server		
eApproval		
Excel Services Application		
InfoPath Forms Services		
Parago		
Application Pages	Information	Medium
PerformancePoint Service		
Secure Store Service		
SharePoint Foundation		
SharePoint Foundation Search		
SharePoint Portal Server		
SharePoint Server		
SharePoint Server Search		
SQL Server Reporting Services		
Visio Graphics Service		
Web Analytics Services		
Web Content Management		

Custom Logging Service

A custom logging service must inherit from the SPDiagnosticsServiceBase class. This is the base class for diagnostic services in SharePoint and it offers the option to log messages to log files via ULS and to the Windows Event Log. By overriding the ProvideAreas method the service provides information about diagnostic areas, categories and logging levels. A diagnostic area is a logical container of one or more categories.

The sample service defines one diagnostic area and one category (used by application pages) for this area:

```
[Guid("D64DEDE4-3D1D-42CC-AF40-DB09F0DFA309")]
public class LoggingService : SPDiagnosticsServiceBase
{
  public static class Categories
  {
    public static string ApplicationPages = "Application Pages";
  }

  public static LoggingService Local
  {
    get { return SPFarm.Local.Services.GetValue<LoggingService>(DefaultName); }
  }

  public static string DefaultName
  {
    get { return "Parago Logging Service"; }
  }

  public static string AreaName
  {
    get { return "Parago"; }
  }

  protected override IEnumerable<SPDiagnosticsArea> ProvideAreas()
  {
    List<SPDiagnosticsArea> areas = new List<SPDiagnosticsArea>
    {
      new SPDiagnosticsArea(AreaName, 0, 0, false, new List<SPDiagnosticsCategory>
      {
        new SPDiagnosticsCategory(Categories.ApplicationPages, null, TraceSeverity.Medium,
              EventSeverity.Information, 0, 0, false, true)
      })
    };

    return areas;
  }

  ...

}
```

The area name as well as the category names will be also shown in the Diagnostic Logging UI of the CA. It is also possible to define a resource DLL to localize the names.

The service will offer the two static methods WriteTrace and WriteEvent. WriteTrace writes the log message to the SharePoint log files, usually saved in the SharePoint folder *C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\LOGS*.

The WriteEvent writes the log message to the Windows Event Log. The event source is called like the AreaName and later on created within the FeatureReceiver:

```
[Guid("D64DEDE4-3D1D-42CC-AF40-DB09F0DFA309")]
public class LoggingService : SPDiagnosticsServiceBase
{

  ...

  public static void WriteTrace(string categoryName, TraceSeverity traceSeverity,
    string message)
  {
    if(string.IsNullOrEmpty(message))
      return;

    try
    {
      LoggingService service = Local;

      if(service != null)
      {
        SPDiagnosticsCategory category = service.Areas[AreaName].Categories[categoryName];
        service.WriteTrace(1, category, traceSeverity, message);
      }
    }
    catch { }
  }

  public static void WriteEvent(string categoryName, EventSeverity eventSeverity,
    string message)
  {
    if(string.IsNullOrEmpty(message))
      return;

    try
    {
      LoggingService service = Local;

      if(service != null)
      {
        SPDiagnosticsCategory category = service.Areas[AreaName].Categories[categoryName];
        service.WriteEvent(1, category, eventSeverity, message);
      }
    }
    catch { }
  }
}
```

The usage of the new custom logging service is quite simple:

```
// ULS Logging
LoggingService.WriteTrace(LoggingService.Categories.ApplicationPages,
  TraceSeverity.Medium, "...");

// Windows Event Log
LoggingService.WriteEvent(LoggingService.Categories.ApplicationPages,
  EventSeverity.Information, "...");
```

Next, we need to register it with SharePoint.

Service Registration

The custom logging service must be registered with SharePoint 2010 to show up in the Diagnostic Logging UI of the CA. The event sources also must be created on each server of the SharePoint farm. These two registration steps can be bundle within the FeatureActivated override method of the FeatureReceiver.

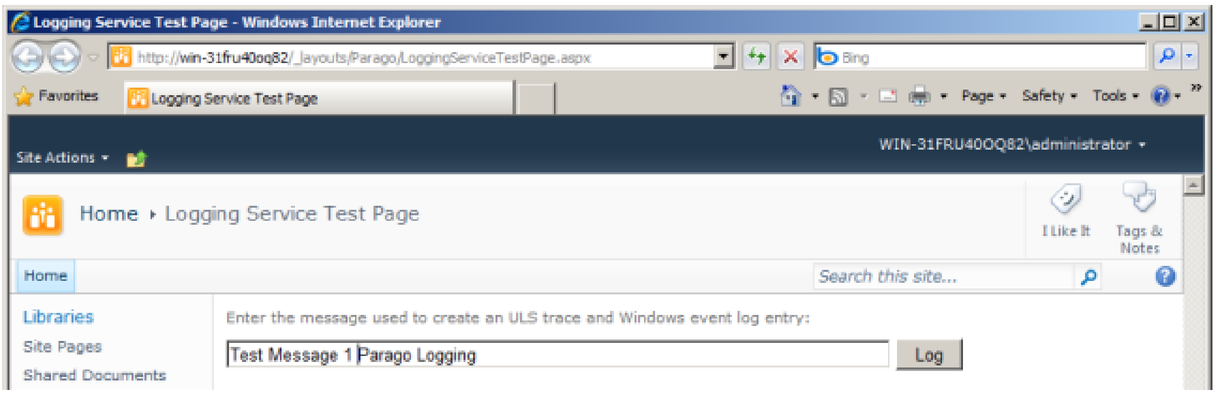
```
[Guid("50CA5F69-381F-4C2A-BE6C-F28219AFF20C")]
public class FeatureEventReceiver : SPFeatureReceiver
{
  const string EventLogApplicationRegistryKeyPath =
    @"SYSTEM\CurrentControlSet\services\eventlog\Application";

  public override void FeatureActivated(SPFeatureReceiverProperties properties)
  {
    RegisterLoggingService(properties);
  }

  public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
  {
    UnRegisterLoggingService(properties);
  }

  static void RegisterLoggingService(SPFeatureReceiverProperties properties)
  {
    SPFarm farm = properties.Definition.Farm;

    if(farm != null)
    {
      LoggingService service = LoggingService.Local;

      if(service == null)
      {
        service = new LoggingService();
        service.Update();

        if(service.Status != SPObjectStatus.Online)
          service.Provision();
      }

      foreach(SPServer server in farm.Servers)
      {
        RegistryKey baseKey = RegistryKey.OpenRemoteBaseKey(RegistryHive.LocalMachine,
                                server.Address);

        if(baseKey != null)
        {
          RegistryKey eventLogKey = baseKey.OpenSubKey(EventLogApplicationRegistryKeyPath,
                                      true);

          if(eventLogKey != null)
          {
            RegistryKey loggingServiceKey = eventLogKey.OpenSubKey(LoggingService.AreaName);

            if(loggingServiceKey == null)
            {
              loggingServiceKey = eventLogKey.CreateSubKey(LoggingService.AreaName,
                                    RegistryKeyPermissionCheck.ReadWriteSubTree);
              loggingServiceKey.SetValue("EventMessageFile",
                @"C:\Windows\Microsoft.NET\Framework\v2.0.50727\EventLogMessages.dll",
                RegistryValueKind.String);
            }
          }
        }
      }
    }
  }

  static void UnRegisterLoggingService(SPFeatureReceiverProperties properties)
  {
    SPFarm farm = properties.Definition.Farm;

    if(farm != null)
    {
      LoggingService service = LoggingService.Local;

      if(service != null)
        service.Delete();

      foreach(SPServer server in farm.Servers)
      {
        RegistryKey baseKey = RegistryKey.OpenRemoteBaseKey(RegistryHive.LocalMachine,
                                server.Address);

        if(baseKey != null)
        {
          RegistryKey eventLogKey = baseKey.OpenSubKey(EventLogApplicationRegistryKeyPath,
                                      true);

          if(eventLogKey != null)
          {
            RegistryKey loggingServiceKey = eventLogKey.OpenSubKey(LoggingService.AreaName);

            if(loggingServiceKey != null)
              eventLogKey.DeleteSubKey(LoggingService.AreaName);
          }
        }
      }
    }
  }
}
```

The FeatureActivated override is calling the RegisterLoggingService helper method to register with the SharePoint system if the service is not already available. Since one of the base classes of LoggingService is the SPService class which provides an Update and Provision method, we can register the new service farm-wide.

The second step is to create a new Windows Event Log source. Therefore we have to go through the collection of SharePoint farm servers and remotely add the new source by generating registry entries on each server.

To unregister we redo the registration steps by overriding FeatureDeactivating method of the FeatureReceiver class. The UnRegisterLoggingService method then deletes the service and removes all registry keys on all servers in the SharePoint farm.

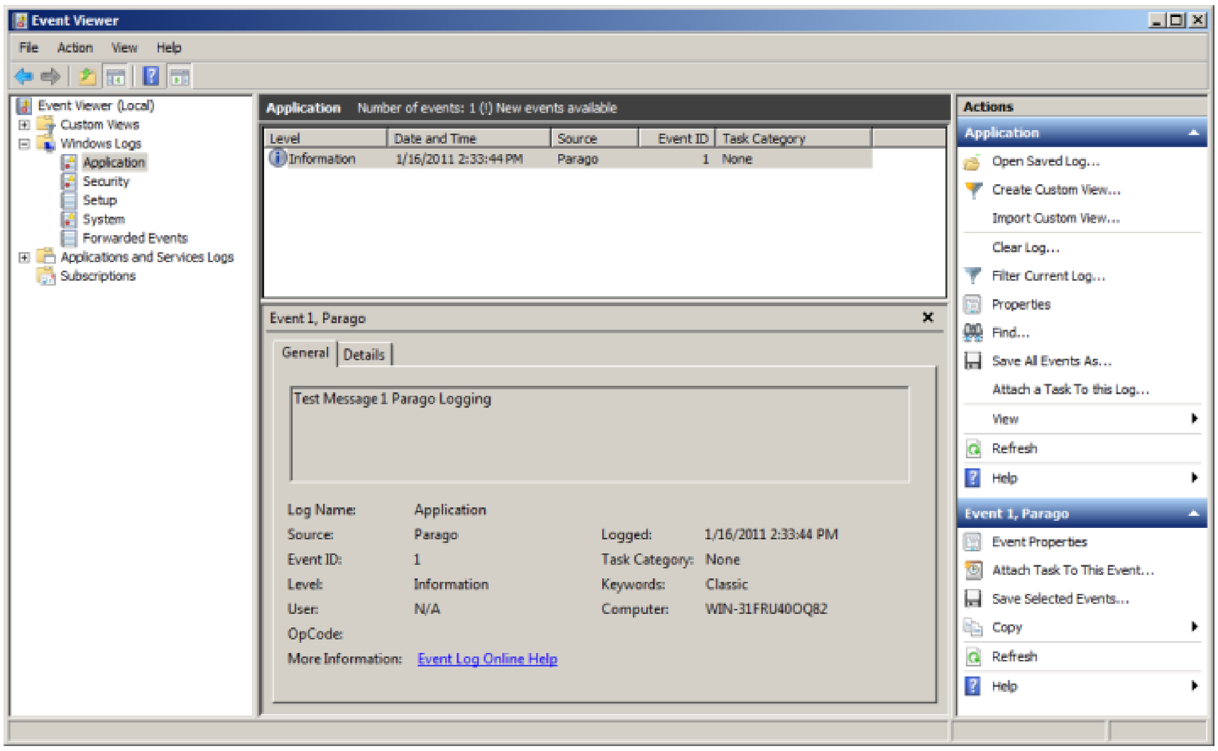
The sample solution also created an application page to test the logging service. The source code is available as download.


Logging Service Test Page - Windows Internet Explorer
http://win-31fru40oq82/_layouts/Parago/LoggingServiceTestPage.aspx
Favorites
Logging Service Test Page
Site Actions
WIN-31FRU40OQ82\administrator
Home ▸ Logging Service Test Page
I Like It
Tags & Notes
Home
Search this site...
Libraries
Site Pages
Shared Documents
Log

Enter the message used to create an ULS trace and Windows event log entry:

Test Message 1 Parago Logging

Enter a test message and press the Log button. The message will be logged to the Windows Event Log:



Application Number of events: 1 (!) New events available

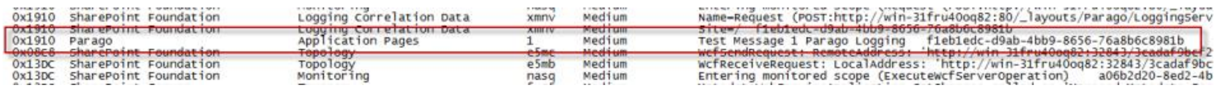
Level	Date and Time	Source	Event ID	Task Category
Information	1/16/2011 2:33:44 PM	Parago	1	None

Event 1, Parago

Test Message 1 Parago Logging

Log Name:	Application		
Source:	Parago	Logged:	1/16/2011 2:33:44 PM
Event ID:	1	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	WIN-31FRU40OQ82
OpCode:			
More Information:	Event Log Online Help		

And to the SharePoint log files:


```
0x1910  SharePoint Foundation      Logging Correlation Data    xmnv   Medium    Name=Request (POST:http://win-31fru40oq82:80/_layouts/Parago/LoggingServ
0x1910  SharePoint Foundation      Logging Correlation Data    xmnv   Medium    Site=/  f1eb1edc-d9ab-4bb9-8656-76a8b6c8981b
0x1910  Parago                     Application Pages           1      Medium    Test Message 1 Parago Logging   f1eb1edc-d9ab-4bb9-8656-76a8b6c8981b
0x08C8  SharePoint Foundation      Topology                    e5mc   Medium    WcfSendRequest: RemoteAddress: 'http://win-31fru40oq82:32843/3cadaf9bcf2
0x13DC  SharePoint Foundation      Topology                    e5mb   Medium    WcfReceiveRequest: LocalAddress: 'http://win-31fru40oq82:32843/3cadaf9bc
0x13DC  SharePoint Foundation      Monitoring                  nasg   Medium    Entering monitored scope (ExecuteWcfServerOperation)    a06b2d20-8ed2-4b
```

That’s it.

Summary

Implementing a custom logging service for SharePoint 2010 is straightforward, easy and very powerful. It is not necessary to create a custom logging solution for small SharePoint projects, but if you are developing large custom solutions you may consider using such a logging system.

Contact Information

If you have any feedback or suggestions, please feel free to contact me:

Jürgen Bäurle
jbaurle@parago.de
http://www.parago.de/jbaurle

Parago Media GmbH & Co. KG
Im Wengert 3 | 71336 Waiblingen, Germany | Phone +49.7146.861803 | Internet http://www.parago.de