

HOL P301 - Implementing a Custom Padarn Log Provider

Table of Contents

Table of Contents	2
HOL Requirements	3
Summary.....	4
Lab Objective	4
Pre-Requisites.....	4
Introduction to Logging	4
Logging within Padarn	5
The Motivation	5
Exercise 1: Adding the AdminLogView ASPX.....	6
Exercise 2: Stubbing out the Code-Behind Classes.....	9
Exercise 3: Providing a custom ILogProvider.....	13
Exercise 4: The code-behind logic for AdminLogView page.....	18
Exercise 5: Enabling logging support in Padarn.....	20
Exercise 6: Walking through the expanded CookieWork page	22
Hands-on Lab Summary	26

HOL Requirements

The following items are required to run this HOL:

- A Desktop PC running Microsoft Windows® XP or Windows Vista
- Microsoft Visual Studio 2008 Professional (or higher)
- A Web Browser
- A Padarn reference system or developer kit

While Padarn will run on almost any hardware that supports the Microsoft .NET Compact Framework 2.0 or higher, this lab assumes that you have one of the Padarn reference hardware platforms with an OpenNETCF-validated Windows CE image running on it. If you are using an alternate hardware or software configuration, the steps outlined in this Hands-on Lab may not be accurate for your environment.

Summary

In this lab, you will learn to take advantage of the logging capabilities provided by Padarn to collect useful information to display to administrator-type users or for general archival purposes. Logging serves many purposes – quite a few are beyond the scope of this lab – but we will show you how to get started and present this information in a straightforward fashion.

Lab Objective

Upon completion of this lab, you will be familiar with the process of extending the [ILogProvider](#) interface to facilitate custom logging behavior.

In this HOL, you will perform the following exercises:

- ✓ Review the hosting of a Padarn web server within a CF.NET manager application
- ✓ Review the creation of a Padarn webpage that leverages cookies for authentication
- ✓ Create a DLL implementing a custom log provider
- ✓ Learn how to configure Padarn to enable logging against the custom log provider
- ✓ Verify that the solution is working by accessing the a Padarn page that pulls data from the log provider

Pre-Requisites

This lab builds on the code created through the HOL 100 and 200 Series. Although the necessary components from those labs are included in this lab's example solution, it's advisable to go through the explanations in the HOL 100 and 200 Series to better understand how Padarn webpages interact with the Padarn web server and to understand how the [HttpRequest.Page](#) object behaves. Furthermore, it is assumed that cookie creation/retrieval is understood by the user. Cookies are explained in depth in HOL 203.

Introduction to Logging

It might be obvious why logging on a production server is important, but let's review some of the reasons why you might wish to log through Padarn. Since Padarn follows the basic ASP.NET 2.0 model, there is heavy emphasis on request and response objects. Should these become invalidated due to missing cookie data or authentication tokens, it's sometimes advisable to make note of these events. While Padarn isn't designed to defend itself against distributed denial-of-service (DoS) attacks, by keeping a log of incoming IP addresses, it becomes possible to keep track of the origin of all users.

Other applications for logging include data analysis report generation that provides insight into client characteristics: web browser make and version, browser capabilities, peak access times, session

duration, etc. Although it is beyond the scope of this article to create detailed charts or dynamic reports, we will be creating a useful summary page of all errors and valid page requests coming into a Padarn instance. Our exercise begins where we left off with HOL P203: Setting Client Browser Cookies from a Padarn Page.

Logging within Padarn

By default, Padarn performs no external logging whatsoever – this is because there needs to be specification for maximum log length and other filtering rules so as to not inundate the limited resource Padarn hardware with large text files. If logging is enabled through the Padarn web server base configuration file (ocfhttpd.exe.config – see bottom of HOL for details), a default [ILogProvider](#) implementation is enabled, which stores a variety of logging statements in one file per day of operation. If this functionality isn't what you're looking for, then you're in luck! This HOL will aide you in creating your own logging routine.

The Motivation

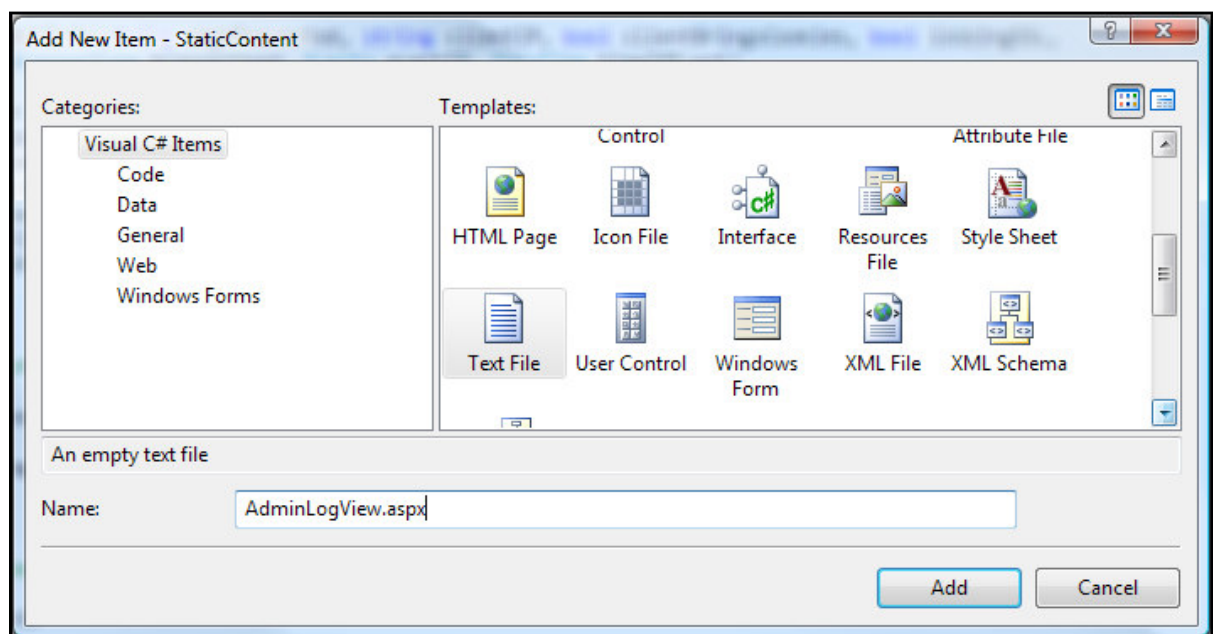
As touched upon earlier, logging provides a mechanism for archival of pertinent access details and other Padarn state particulars. Whether these details are stored on the Padarn server, a UNC network share, or even a structured database, v1.1.92 of Padarn exposes an [ILogProvider](#) interface so that you can do as you please with Padarn event logging. In this HOL, we will create a non-serialized data store that maintains the following particulars for a given page access/error event: page accessed, client IP address, whether the client is using cookies valid for the Padarn domain, whether SSL is being used between server and client, what kind of browser is being used (make and version), an event ID if an error occurred, and finally the time of the event. Since this data will not be saved to disk (so as to conserve storage space), we will create a Padarn webpage that displays the current pool of event data.

Exercise 1: Adding the AdminLogView ASPX

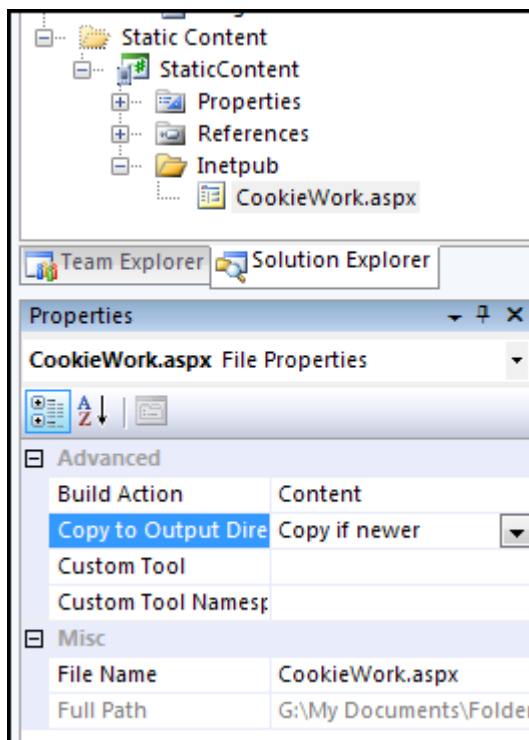
Whenever a user accesses a Padarn webpage, he will be accessing an HTML, ASPX, or other standard front-end ASP.NET-compatible document type. However, Padarn will immediately call into an appropriate code-behind assembly (DLL) that contains page load logic for each page. We've been using this model for each of our previous HOLs, and we'll continue in the same direction here.

To create the ASPX file:

1. In Visual Studio's **Solution Explorer** Pane, right-click on the **Inetpub** folder in the **StaticContent** project and select **Add -> New Item** to display the **Add New Item** dialog.
2. From the list of **Visual Studio Templates**, select **Text File**
3. In the **Name** textbox enter "AdminLogView.aspx"



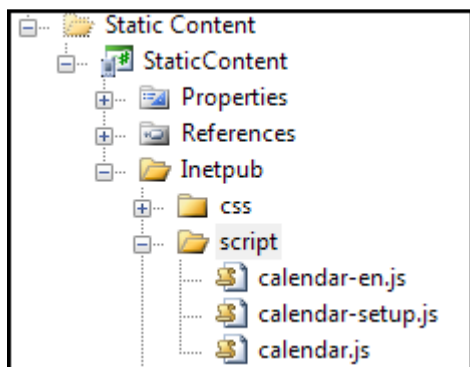
4. Click **Add** to add the new file to the project
5. In Visual Studio's **Solution Explorer** Pane, click on the newly-created **RPCEExample.aspx** file
6. In the **Properties** pane, set the **Build Action** for **CookieWork.aspx** to "Content"
7. In the **Properties** pane, set the **Copy to Output Directory** property for **AdminLogView.aspx** to "Copy if Newer"



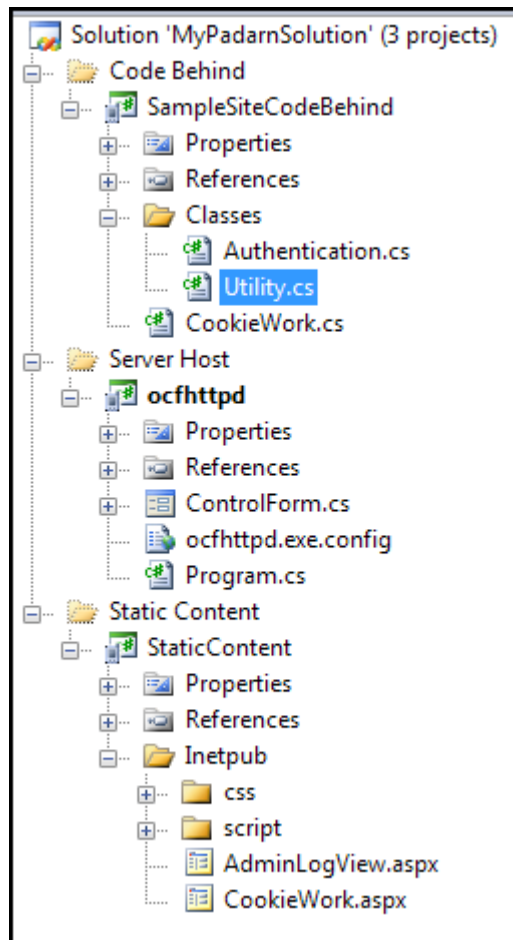
8. Paste the following code into the newly-created AdminLogView.aspx file (this will tell Padarn to look for the code-behind logic in a DLL called CookieWorkSample.dll, using reflection to peer into class SampleSite.AdminLogView):

```
<%@ Page CodeBehind=" CookieWorkSample.dll" Inherits="SampleSite.AdminLogView" %>
```

9. We also want to use some freely-available JavaScript logic to provide us with a useful calendar control for the AdminLogView page. This control will allow the user to select a date to filter log entries. Those files are added under the Inetpub\script folder and are included with this exercise's code files. Verify that the build action for these three scripts is "Copy if Newer".



10. The solution will now look like this:

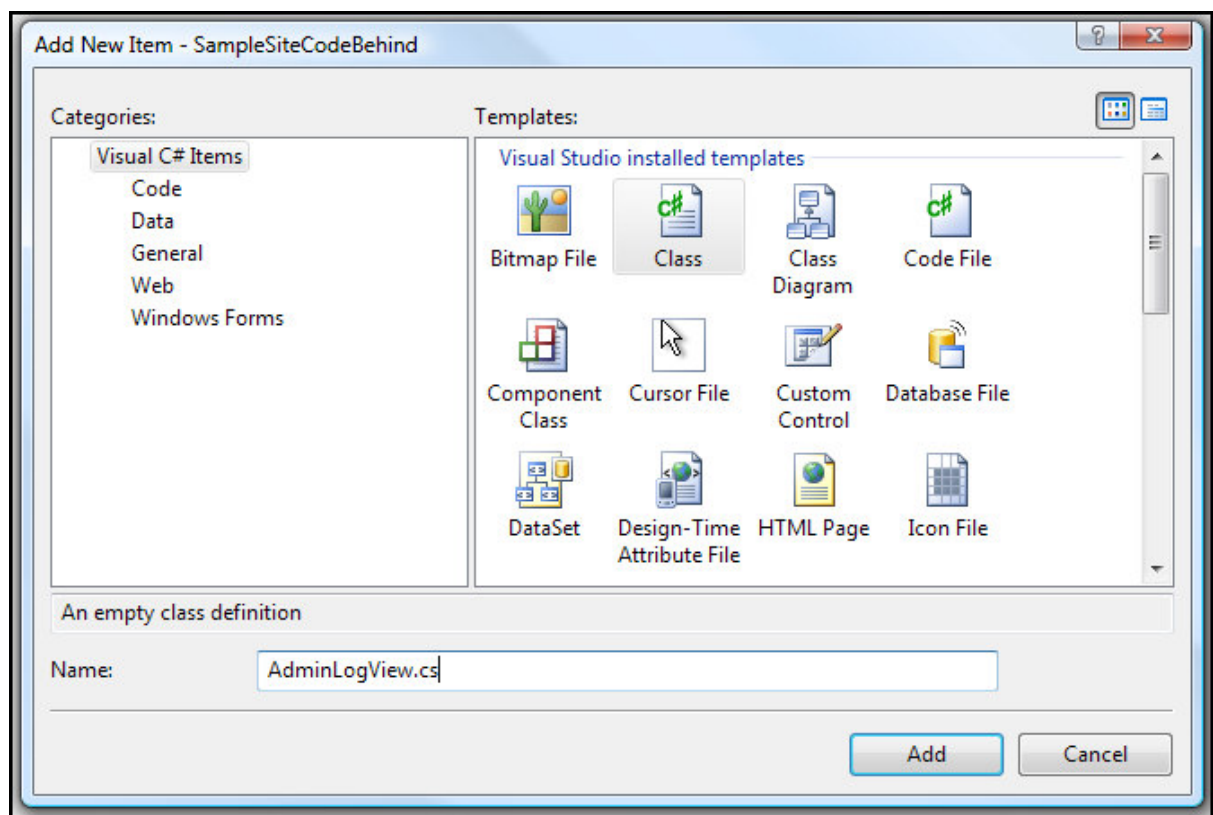


Exercise 2: Stubbing out the Code-Behind Classes

Since we are reusing code from HOL 203, we already have the authentication scheme in place as well as the code-behind to create the login webpage that the user will see. All we need to do is add a single code-behind file to create the administrative log view page, which we'll call AdminLogView.cs.

To create the main code-behind class:

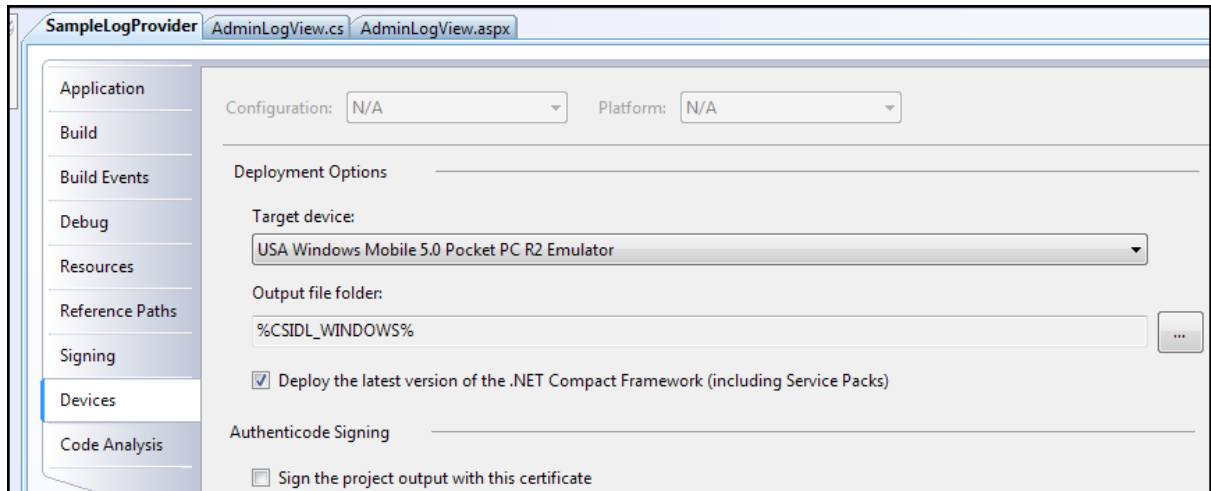
1. In Visual Studio's Solution Explorer pane, right-click on the SampleSiteCodeBehind project and select **Add -> New Item** to display the Add New Item dialog.
2. From the list of Visual Studio Installed Templates select **Class**.
3. In the Name textbox, enter "AdminLogView.cs".



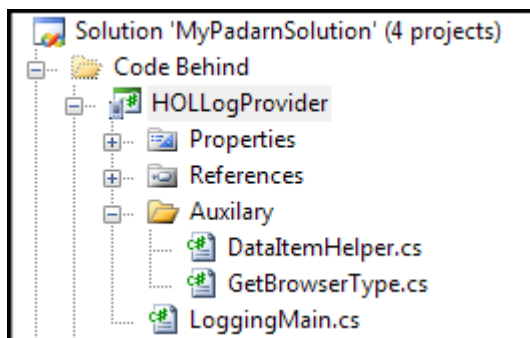
4. Click **Add** to add the new file to the project

We will now create a project that will contain all functionality for our custom logging implementation. This is a crucial step – because Padarn dynamically injects a custom logging routine (at runtime), it is important that this functionality exist in a standalone DLL.

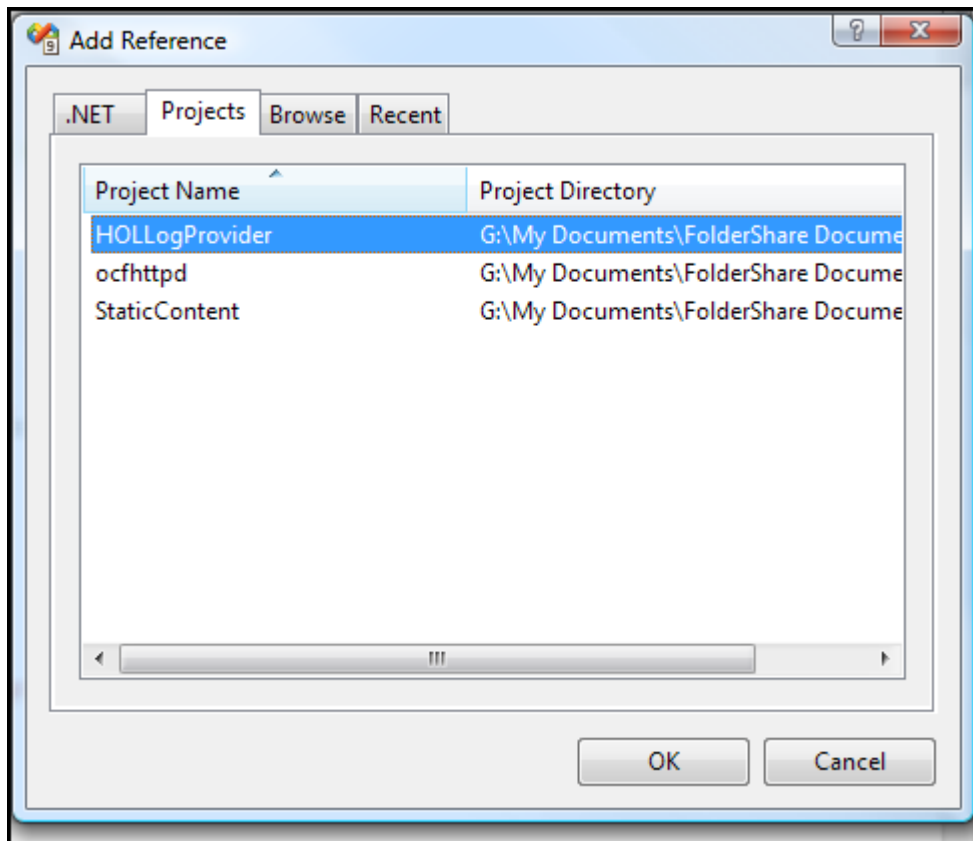
1. Create a new Visual C# - Smart Device project called "HOLLogProvider.csproj".
2. Delete any of the code files it has created for you (Class1.cs).
3. We will need to make sure this file gets deployed to the \Windows folder of the Padarn server. To do this, right click on the project and select **Properties**. View the Devices tab.



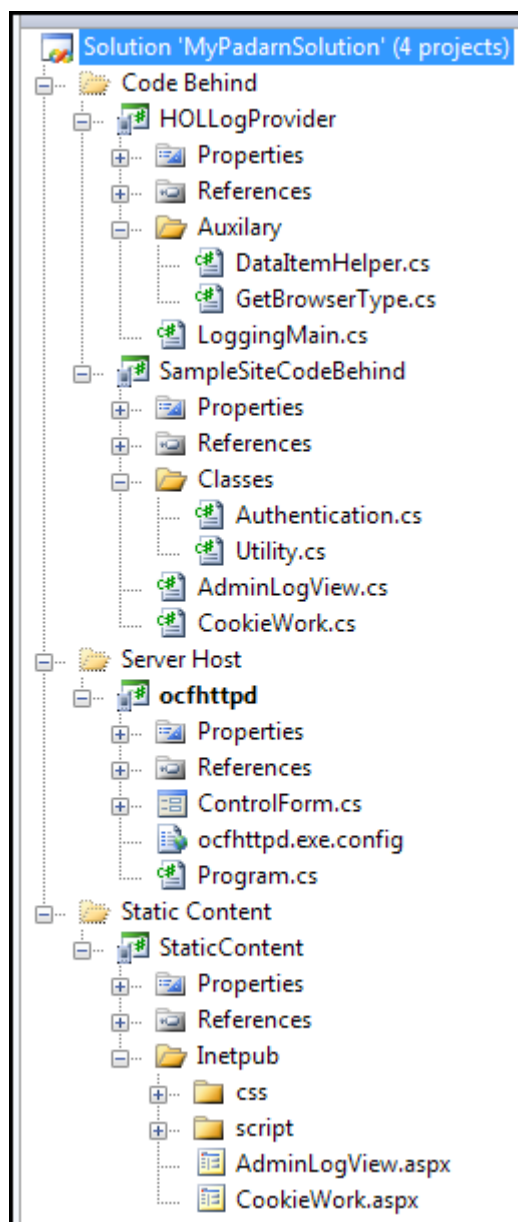
4. Click the ellipse (...) and choose "Windows Folder" in the Location of output on device.
5. Create a sub-folder within the HOLLogProvider project called "Auxiliary".
6. Create two class files under the Auxiliary folder called DataItemHelper.cs and GetBrowserType.cs. We'll discuss these files later on.
7. Create a single class file under the main HOLLogProvider node called LoggingMain.cs.
8. After performing these steps, this is what your project should look like:



9. Next, we'll need to add a project reference from our SampleCodeBehind project to the HOLLogProvider project. We do this so that when page requests come into the CookieWork code-behind class, we'll be able to interact with our logging component. Right click on SampleSiteCodeBehind's References folder and choose "Add Reference".



10. At the completion of this exercise, the below view is how your solution should appear:



Exercise 3: Providing a custom ILogProvider

In the 1.1.X series of Padarn, the logging support is purposefully kept to the basics. As Padarn is a product very much driven by the demands of its customers, there will likely be improvements to the feature set over time. For the time being, creating your own `ILogProvider` main class is quite simple. You'll need to implement three functions and one property: `LogPadarnError`, `LogPageAccess`, `LogRuntimeInfo` (experimental), and `ServerConfiguration`. You do not need to explicitly instantiate your implementation; instead, Padarn will dynamically inject an instance into the Padarn web server instance at runtime (so long as the configuration file points to the `HOLLogProvider` DLL – see below for details). This makes your code that much tidier!

`LogPadarnError` will be called whenever any exceptional situation takes place, including page not found (user error) or .NET exceptions (internal error). `LogPageAccess` will be called whenever a new page is opened or if a redirect to a page takes place. Both of these functions contain a `LogDataItem`, which provides a tremendous amount of information.

```
5 namespace OpenNETCF.Web.Logging
6 {
7     public sealed class LogDataItem
8     {
9         public NameValueCollection Headers { get; set; }
10        public string PageName { get; set; }
11        public string RemoteClientIP { get; set; }
12        public ServerConfig ServerConfiguration { get; set; }
13    }
14 }
```

In addition to all of the header data being sent by the client, it also contains details about the Padarn web server under which the page is being served. Note: in most cases, errors will not return a valid `LogDataItem`, so it is important to always check for nullness! This is a design decision that is due to the level in the stack where most errors are caught – it's just not possible to send back client specifics or anything about the page. Certain security-related errors will, in fact, generate a `LogDataItem`.

`LogRuntimeInfo` is an experimental function that should be treated as an unsupported API (although it is a required interface member at present time). It should be used only for debugging – it will receive a wide variety of activity messages from the Padarn internals.

`ServerConfiguration` exists to allow non-XML overrides of some of the basic Padarn configuration data. More importantly, though, it allows read access to Padarn configuration data during an error event.

```
public class LoggingSupport : ILogProvider
{
    #region Implemented Interfaces

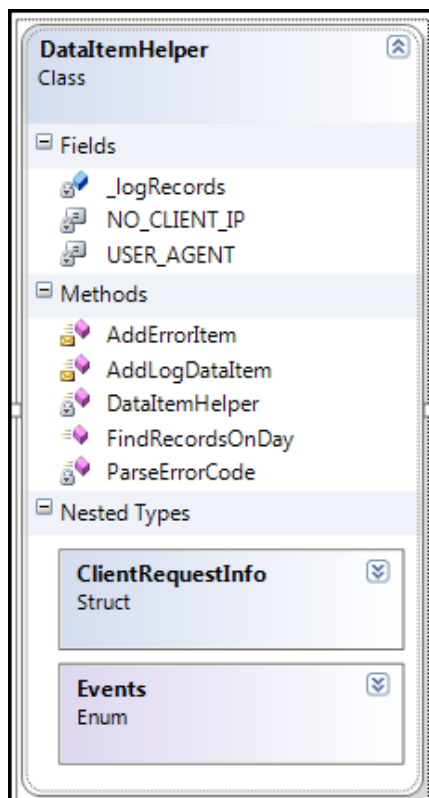
    /// <summary>
    /// Gets or sets configuration data pertaining to running web server
    /// </summary>
    public ServerConfig ServerConfiguration { get; set; }

    public void LogPadarnError(string errorInfo, LogDataItem dataItem)
    {
        //Store away this error. LogDataItem may or may not be null
        DataItemHelper.AddErrorItem(errorInfo, dataItem, ServerConfiguration);
    }

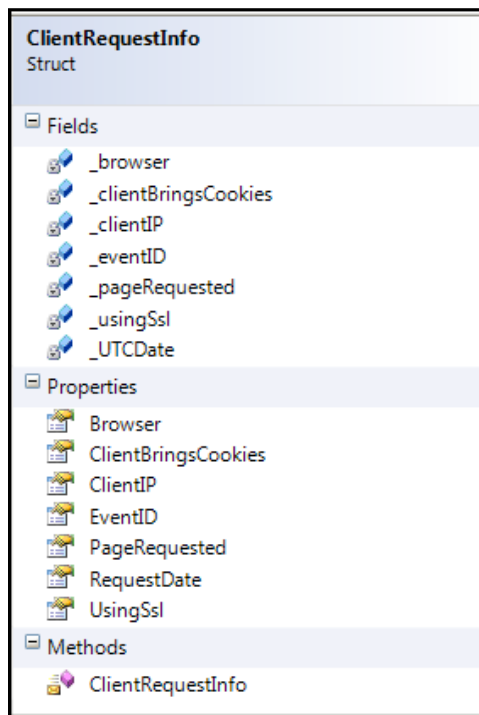
    public void LogPageAccess(LogDataItem dataItem)
    {
        //Store away this entry item
        DataItemHelper.AddLogDataItem(dataItem, ServerConfiguration);
    }

    public void LogRuntimeInfo(ZoneFlags zoneMask, string info)
    {
        Debug.WriteLine(string.Format(" zone '{0}' : {1}", zoneMask.ToString(),
            info));
    }
}
```

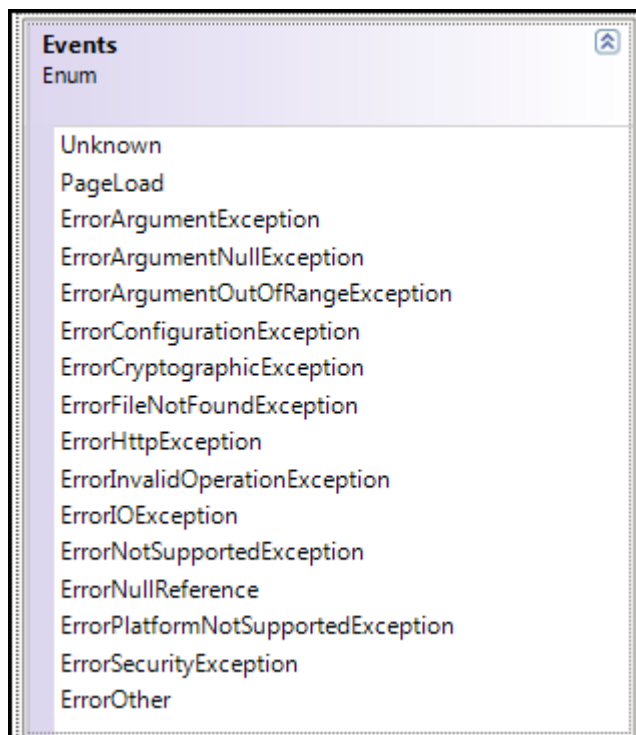
As you can see, all of the functions call into the `DataItemHelper` class. We do nothing with server configuration data. Let's take a closer look at this class.



Our class consists of a generic list of `ClientRequestInfo` structures, a few static methods to add to our list, an `Events` enumeration, and the `ClientRequestInfo` definition itself:



The `Events` enumeration attempts to capture as many of the common Padarn errors and exceptions as possible.



A function called `ParseErrorCode` will take in the error string from `LogPadarnError` and attempt to match it to one of several known types and return that enumeration value.

The two static functions are quite straightforward in how they add to the list:

```
internal static void AddLogDataItem(LogDataItem logData, ServerConfig serverConfig)
{
    if (logData != null && serverConfig != null)
    {
        string userAgentString = logData.Headers[USER_AGENT];
        GetBrowserType.BrowserType browser =
            GetBrowserType.DetermineBrowserType(userAgentString);

        //Add record to data structure
        _logRecords.Add(new ClientRequestInfo(logData.PageName, logData.RemoteClientIP,
            serverConfig.Cookies.Domain != null, serverConfig.UseSsl, browser,
            Events.PageLoad, DateTime.Now));
    }
}

internal static void AddErrorItem(string errorInfo, LogDataItem logData,
    ServerConfig serverConfig)
{
    string pageName = logData != null ? logData.PageName : errorInfo;
    string remoteIP = logData != null ? logData.RemoteClientIP : String.Empty;
    GetBrowserType.BrowserType browser = GetBrowserType.BrowserType.Unknown;

    if (logData != null)
    {
        string userAgentString = logData.Headers[USER_AGENT];
        browser = GetBrowserType.DetermineBrowserType(userAgentString);
    }

    _logRecords.Add(new ClientRequestInfo(pageName, remoteIP,
        serverConfig.Cookies.Domain != null,
        serverConfig.UseSsl, browser, ParseErrorCode(errorInfo), DateTime.Now));
}
```

We will create a function called `FindRecordsOnDay` that will be used by the `AdminLogView` code-behind to fetch a subset of log records captured on a particular day.

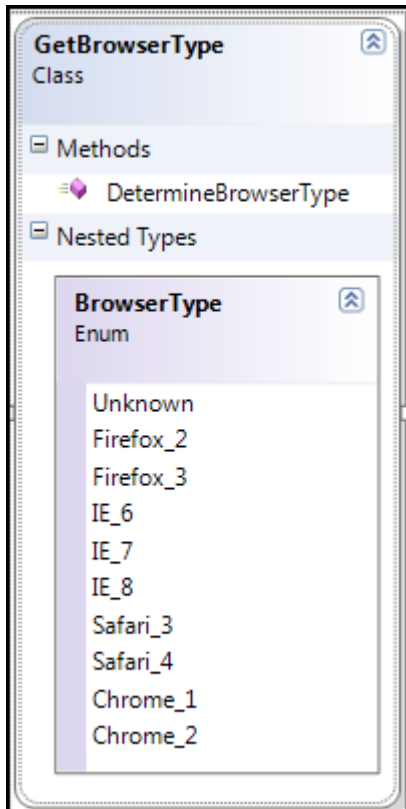
```
public static List<ClientRequestInfo> FindRecordsOnDay(DateTime specifiedDay)
{
    List<ClientRequestInfo> records = new List<ClientRequestInfo>();

    for (int i = 0; i < _logRecords.Count; i++)
    {
        //Go through each item and determine if it occurred during target day
        if (_logRecords[i].RequestDate.Date == specifiedDay.Date)
        {
            records.Add(_logRecords[i]);
        }
    }

    return records;
}
```


It is important to note at this point that the `DataItemHelper` class lives in memory and does not serialize its data to file. It is beyond the scope of this article to demonstrate storing this information to the Padarn's file store using a `TextWriter` class, for example.

In addition to the basic information returned through the `LogDataItem` object, we've provided some reusable code to help keep track of the various browsers being used to access a given Padarn page. Since each web browser formats the user agent string in a different manner, we've come up with a set of rules for the current crop of web browsers, as of early 2009.



The single function in this class, `DetermineBrowserType`, will be used by our custom log provider to get the model and version of the web browser being used by the client.

Exercise 4: The code-behind logic for AdminLogView page

Now that we've created the majority of the code that will be used by our custom logging provider, we're ready to implement the logic that will present the user with a searchable view of this information. This will all live within AdminLogView.cs. Like with all code-behind classes, the functionality will be initiated from the Page_Load function, which is the entry point upon user's access of the page in question.

Our Page_Load function has two modes of operation: initial and redirected. In the initial view, we create a page that prompts the user to select a date via a calendar control. In the redirected page, we take the selected date and query the log store for records created during the day in question.

Padarn makes it easy to add scripts to an HTML document's header:

```
Document doc = new Document();
doc.Head = new DocumentHead("OpenNETCF Padarn Web Server",
    new StyleInfo("css/SampleSite.css"));

//Add the calendar scripts
doc.Head.Scripts.Add(new Script("script/calendar.js"));
doc.Head.Scripts.Add(new Script("script/calendar-en.js"));
doc.Head.Scripts.Add(new Script("script/calendar-setup.js"));

Utility.AddPadarnHeaderToDocument(doc, true, "AdminLogView");
```

Next, we'll add the calendar's associated Search button and input text:

```
Form calendarForm = new Form("AdminLogView.aspx", FormMethod.Post);
calendarForm.Add(new RawText("<input type=\"text\" id=\"data\" name=\"data\" />"));
calendarForm.Add(new RawText("<button id=\"trigger\">...</button>"));
calendarForm.Add(new Button(new ButtonInfo(ButtonType.Submit, "Search")));
timeDiv.Add(calendarForm);
```

Notice the self redirect as the HTTP POST action when the Search button is pressed. Finally, a little in-line JavaScript work:

```
timeDiv.Add(new RawText("<script type=\"text/javascript\">\r\n" +
    "Calendar.setup(\r\n" +
    "    {\r\n" +
    "        inputField : \"data\", \r\n" +
    "        ifFormat : \"%m-%d-%y %H:%M\", \r\n" +
    "        button : \"trigger\", \r\n" +
    "        showsTime : \"true\" \r\n" +
    "    } \r\n" +
    "); \r\n" +
    "</script>\r\n"
));
```

Next, we'll concern ourselves with the redirect code. We will want to first check to see if any records exist for a particular day. If none exist, we show simple boilerplate. Otherwise, we'll create an HTML table and fill in rows containing an ID, page location, IP address of requesting device, whether cookies are enabled for the active domain, whether SSL is enabled, the browser type (using our [GetBrowserType](#) class), the event ID, and finally the time of the event. We perform a small amount of in-line formatting so that the user isn't presented with whitespace if the particular log entry is missing data:

```
foreach (DataItemHelper.ClientRequestInfo data in listEntries)
{
    row = new Row();
    row.Cells.Add(new FormattedText(++recordCount.ToString(), TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.PageRequested, TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.ClientIP, TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.ClientBringsCookies ? "YES" : "NO", TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.UsingSsl ? "ON" : "OFF", TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.Browser.ToString(), TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.EventID.ToString(), TextFormat.None, "2"));
    row.Cells.Add(new FormattedText(data.RequestDate.ToLongTimeString(), TextFormat.None, "2"));
    table.Rows.Add(row);
}
```


Exercise 5: Enabling logging support in Padarn


The oft-mentioned Padarn basic configuration file, `ocfhttpd.exe.config`, has a number of useful properties for customizing if and how events are logged. The applicable elements exist within the `WebServer` tag.



```
<WebServer
  LocalIP="0.0.0.0"
  DefaultPort="80"
  MaxConnections="20"
  DocumentRoot="\Windows\Inetpub\"
  Logging="true"
  LogFolder="\Temp\Logs"
  LogExtensions="aspx;html;htm;zip"
  LogProvider="HOLLogProvider.dll"
```

We'll first want to ensure `Logging` is set to `true` and `LogFolder` is set to something valid (`\Temp\PadarnLogs` will be used by default and whatever folder is used will be created if it doesn't exist). `LogExtensions` specifies the document types whose page access will be recorded. For example, if you're serving several EXE files, you'll want to add "EXE" to this section, semi-colon delimited, so that each EXE download is sent to the `LogPageAccess` unction you've specified.


Finally, `LogProvider` is where you specify the LogProvider DLL you've established, which will exist in the `\Windows` folder. If this value isn't specified and `Logging` is set to "true", all logging will be handled by a default implementation of `ILogProvider` (`DefaultLogProvider`). This will create a file for each day's events in the format of `LOG_<YEAR>-<MONTH>-<DAY>.txt`. These files will reside in the folder specified above. Error logs will all be placed in a file called `PadarnErrors.txt`, and this will be placed in the same `LogFolder` path. Below is a listing of the `\Temp\Logs` folder followed by the contents of the log file for 3/21/2009.


File Explorer
1:13
X


Logs

	Name
	PadarnErrors 3/21/09 230B
	LOG_2009-3-21 3/21/09 226B

Up
Menu





Word Mobile
1:14
ok




```

[2009-03-21 01:10:11]
\CookieWork.aspx <192.168.11.7>
[2009-03-21 01:10:18]
\CookieWork.aspx <192.168.11.7>
[2009-03-21 01:10:32]
\CookieWork.aspx <192.168.11.7>
[2009-03-21 01:12:28]
\AdminLogView.aspx <192.168.11.7>

```

B
I
U

View
Menu

Exercise 6: Walking through the expanded CookieWork page

Now that we have logging configured and linking to our custom provider, we're ready to see our routine in action. In order to validate that our code-behind class work properly, we need to ensure a valid instance of Padarn is up and running; furthermore, you should validate that AdminLogView.aspx and CookieWork.aspx (along with the various CSS and JavaScript files) have been deployed as static content to the Padarn host machine. Instructions for how to accomplish this are in HOL 100.

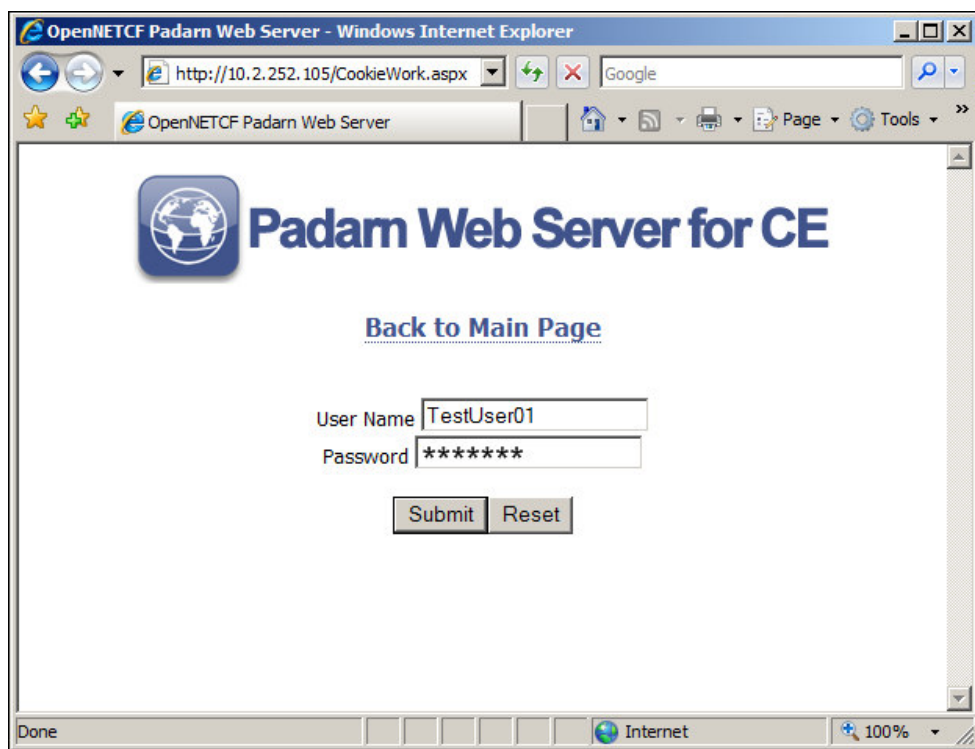
1. Initiate Padarn through the MyPadarnSolution Visual Studio 2008 solution. Right-click on the ocfttpd project and select **Debug -> Start New Instance** (or press F5). For this demonstration, we will be using the Windows Mobile 5.0 Pocket PC R2 Emulator.



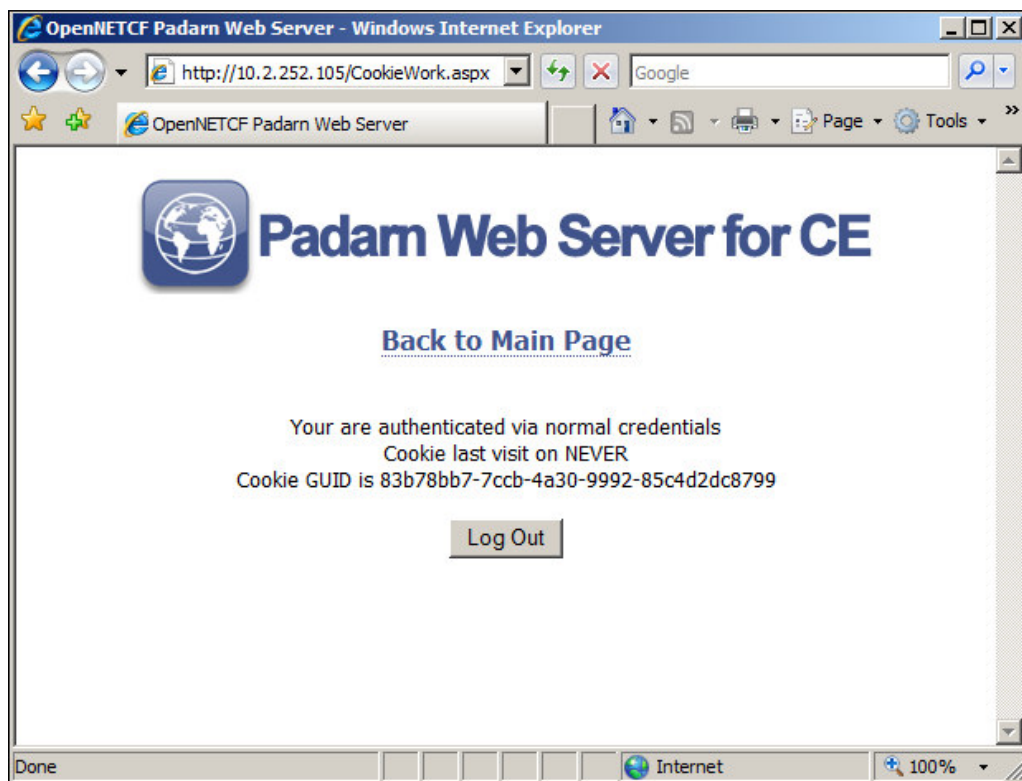
2. Press the **Start** button to initiate the Padarn instance.



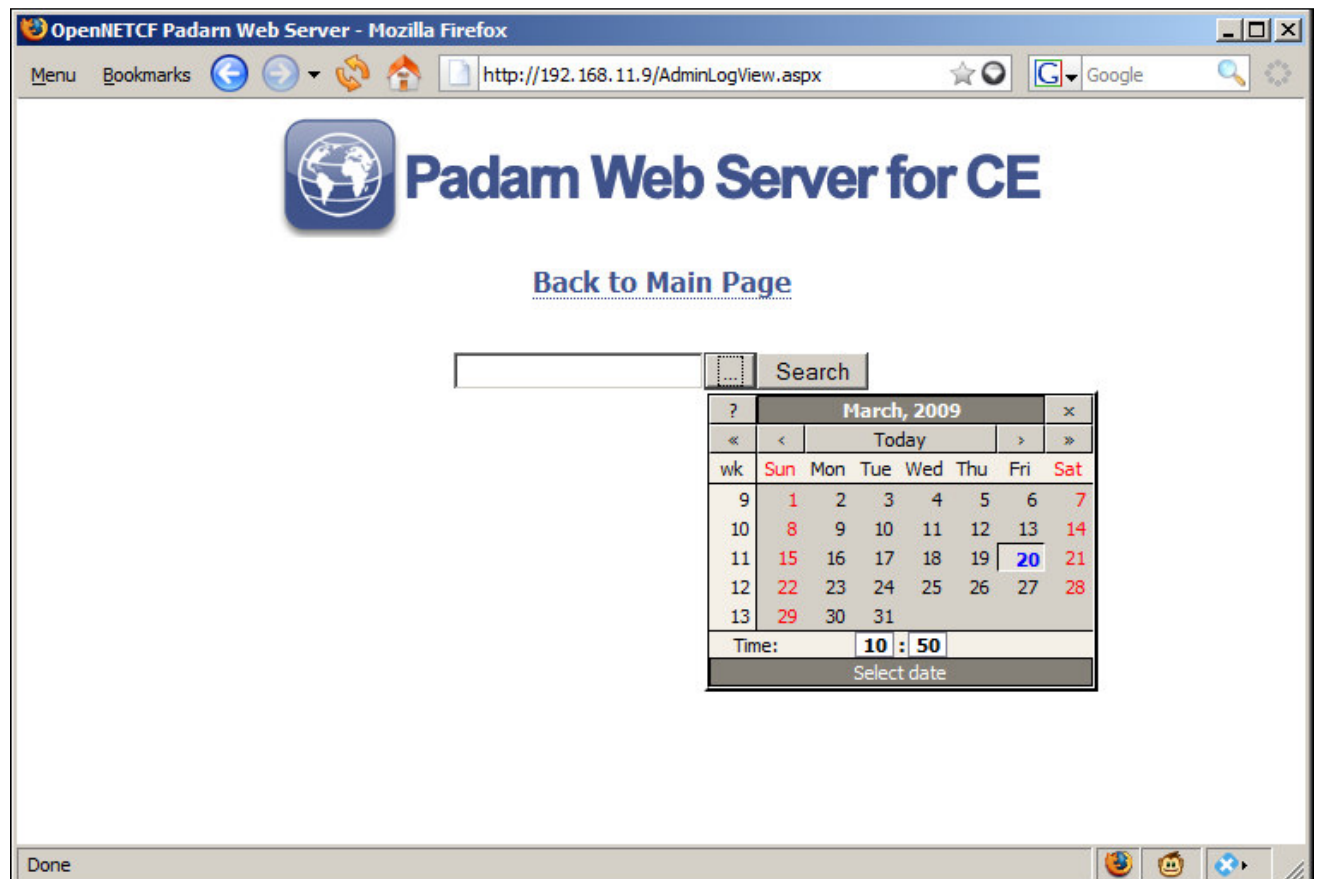
3. From a web browser that can communicate with the Padarn instance (for example, a Windows XP laptop that is host to the Windows Mobile 5.0 emulator). The IP address of the Padarn instance is key for bring up any Padarn-served content. Various utilities exist for Windows CE that provide this information. For example, you could use the [OpenNETCF.Net.NetworkInformation](#) namespace.
4. Visit <Padarn Server IP Address>/CookieWorkAAA.aspx in your favorite web browser. This will create an error record in the [DataItemHelper](#)._logRecords data structure.
5. Next, visit <Padarn Server IP Address>/CookieWork.aspx. This will bring up the normal login page.



6. Use credentials login = "TestUser01" and password = "TestPW01" without quotes. This will send you to the non-cookie authenticated page





7. Next, go to <Padarn Server IP Address>/AdminLogView.aspx. This is our log viewer starting point.




8. Pressing the ellipse (...) brings up the calendar control (AdminLogView scenario #1). Select a *valid* date and then press the **Search** button. Doing so engages the redirect to AdminLogView scenario #2.

OpenNETCF LogProvider Sample Output - Mozilla Firefox

Menu Bookmarks  http://192.168.11.9/AdminLogView.aspx 

Listing of Event Log for 3/20/09 :



Num #	Page	Client IP	Cookies	SSL	Browser	Event ID	Event Time
1	Exception handling HTTP Request: The file '/AdminLAAogView.aspx' cannot be found.	No client IP	YES	OFF	Unknown	ErrorOther	12:52:59 PM
2	\\Windows\\Inetpub\\CookieWork.aspx	192.168.11.7:17619	YES	OFF	Firefox_3	PageLoad	12:53:11 PM
3	\\Windows\\Inetpub\\css\\SampleSite.css	192.168.11.7:17620	YES	OFF	Firefox_3	PageLoad	12:53:14 PM
4	\\Windows\\Inetpub\\images\\header.png	192.168.11.7:17621	YES	OFF	Firefox_3	PageLoad	12:53:16 PM
5	\\Windows\\Inetpub\\CookieWork.aspx	192.168.11.7:17622	YES	OFF	Firefox_3	PageLoad	12:53:22 PM
6	\\Windows\\Inetpub\\CookieWork.aspx	192.168.11.7:17630	YES	OFF	Firefox_3	PageLoad	12:53:36 PM
7	\\Windows\\Inetpub\\css\\SampleSite.css	192.168.11.7:17631	YES	OFF	Firefox_3	PageLoad	12:53:37 PM
8	\\Windows\\Inetpub\\images\\header.png	192.168.11.7:17632	YES	OFF	Firefox_3	PageLoad	12:53:37 PM
9	\\Windows\\Inetpub\\AdminLogView.aspx	192.168.11.7:17637	YES	OFF	Firefox_3	PageLoad	12:54:45 PM
10	\\Windows\\Inetpub\\css\\SampleSite.css	192.168.11.7:17638	YES	OFF	Firefox_3	PageLoad	12:54:45 PM
11	\\Windows\\Inetpub\\script\\calendar.js	192.168.11.7:17639	YES	OFF	Firefox_3	PageLoad	12:54:45 PM
12	\\Windows\\Inetpub\\script\\calendar-en.js	192.168.11.7:17640	YES	OFF	Firefox_3	PageLoad	12:54:46 PM
13	\\Windows\\Inetpub\\script\\calendar-setup.js	192.168.11.7:17641	YES	OFF	Firefox_3	PageLoad	12:54:46 PM
14	\\Windows\\Inetpub\\images\\header.png	192.168.11.7:17642	YES	OFF	Firefox_3	PageLoad	12:54:46 PM
15	Exception handling HTTP Request: The file '/css/menuarrow.gif' cannot be found.	No client IP	YES	OFF	Unknown	ErrorOther	12:54:47 PM
16	\\Windows\\Inetpub\\AdminLogView.aspx	192.168.11.7:17644	YES	OFF	Firefox_3	PageLoad	12:54:49 PM
17	\\Windows\\Inetpub\\css\\SampleSite.css	192.168.11.7:17645	YES	OFF	Firefox_3	PageLoad	12:54:52 PM
18	\\Windows\\Inetpub\\CookieWork.aspx	192.168.11.7:17667	YES	OFF	IE_7	PageLoad	12:55:15 PM
19	\\Windows\\Inetpub\\images\\header.png	192.168.11.7:17669	YES	OFF	IE_7	PageLoad	12:55:15 PM
20	\\Windows\\Inetpub\\css\\SampleSite.css	192.168.11.7:17668	YES	OFF	IE_7	PageLoad	12:55:18 PM
21	\\Windows\\Inetpub\\AdminLogView.aspx	192.168.11.7:17671	YES	OFF	Firefox_3	PageLoad	12:55:21 PM

Done 

Notice that the errors do not contain the same amount of information as the page access records.


- Next, go to <Padarn Server IP Address>/AdminLogView.aspx. This is our log viewer starting point. Here, enter a date in the past (during which log entries have not been stored).

OpenNETCF LogProvider Sample Output - Mozilla Firefox

Menu Bookmarks  http://192.168.11.9/AdminLogView.aspx 

Listing of Event Log for 3/18/09 :

- No records found for 3/18/09 -

Done 

As you can see, our boilerplate displayed properly.

Hands-on Lab Summary

The purpose of this article was to review the logging design philosophy as implemented in the v1.1 series of OpenNETCF Padarn Web Server and to illustrate a concrete illustration of how that data can be used to present an administrator with a searchable view of web server traffic. Although currently, the usage of logging is somewhat limited when hunting down errors, it is quite useful for understanding the nature of clients using the served pages or documents.

Future releases of Padarn might have expanded functionality, though the capabilities of the version 1 product should be sufficient for most users.

In this lab you:

- ✓ Re-used an existing your Padarn website solution
- ✓ Reviewed a simple illustration of an authentication routine backed by Padarn
- ✓ Learned about the internals of Padarn logging and how to add a custom logging support DLL
- ✓ Saw the logging support in action through a searchable web view of the logging data collected

Because a device running Padarn might be headless or otherwise “out of sight, out of mind”, it’s important to make sure administrators are empowered to diagnose issues within the web server as well as have a handle on who is accessing it. Whether you use basic logging support provided in the box or choose to provide a custom logging provider, Padarn gives you the tools to closely monitor its events over time.