Chapter 7

Using the PDSA .NET Providers

As with any provider library you will need to add some configuration settings to your App.Config or Web.Config file in order to use the providers. The PDSA Providers are no exception. Depending on the provider you use, you will need to add a set of configuration settings to your config file for your application. The sections below will outline the basic providers and what settings you need to add to your configuration file.

Overview of PDSA Providers

Each provider system in the PDSA Framework has a similar structure (Figure 1). There is always a “Manager” class whose job it is to read the list of providers from the .Config file and set a **Provider** property with the default provider listed in the .Config file. It will also create a **Providers** collection property with a list of all of the providers available for use.

You can access the individual attributes from each provider through the **ConfigurationProvider** property. You can get to all of the configuration attributes through the **ConfigurationProviders** property.

There are three methods that you use on each provider manager class to help you get an individual provider or a collection of providers. These methods are **GetProvider**, **GetProviderNames** and **GetProvidersCollection**. There is a **Reset** method which will clear all properties on the manager class back to null and a **ResetCache** which will eliminate any loaded providers from the cache.

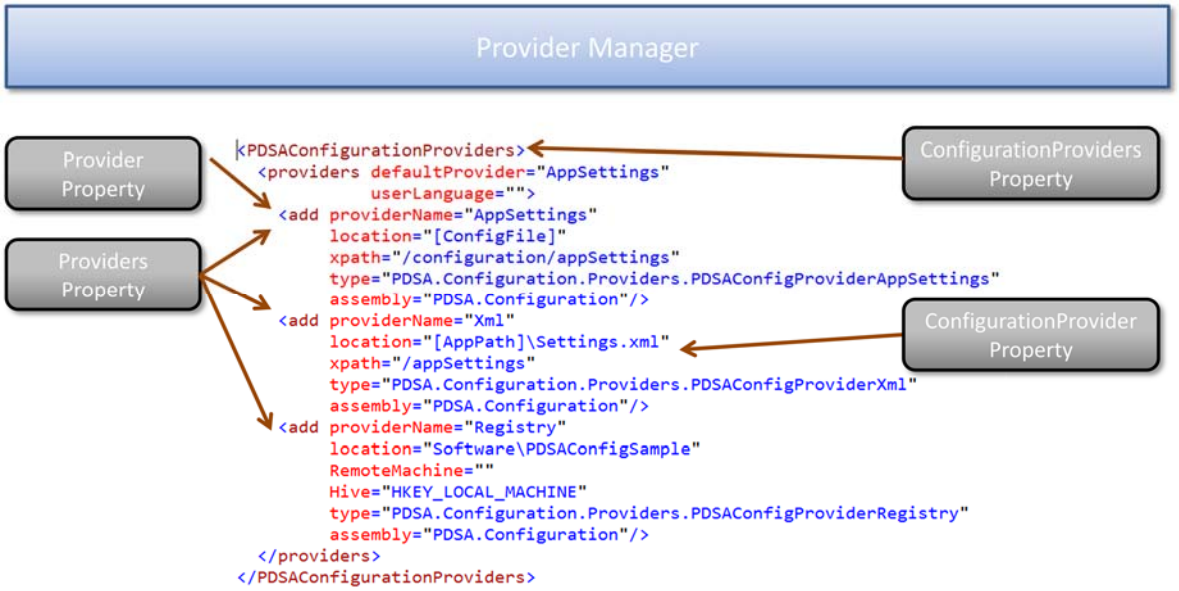

Figure 1: Overview of the PDSA Providers

Caching Provider

These provider classes help you retrieve and save any data in a cache for quick retrieval in your desktop or web application. There are two providers for caching; one for desktop applications (WPF, Windows Services and Windows Forms) and one for ASP.NET (MVC/Web Forms) applications. Any serializable data can be safely cached within the caching provider. You a few different choices when using the caching:

- Cache the data for the whole application and all users
- Cache data for just a single user
- The data can last until the application stops running
- You can set an absolute expiration date/time on the data after which time the data is automatically removed from the cache
- You can set a sliding expiration date/time on the data. If you access the data before the expiration date/time, then the date/time is set forward to the new date/time, otherwise the data is automatically removed from the cache.

You may also roll-your-own provider to cache data in any manner or data store you wish.

Sample Folder

[InstallFolder]\Samples\WebForms\PDSACacheProviderSample

Configuration

To use the PDSA Cache system you will first need to add the following <section> to the <configSections> element at the top of your .config file.

```
<configSections>
  <section name="PDSACacheProviders"
           type="PDSA.Cache.Configuration.PDSACacheSectionHandler,
                 PDSA.Cache, Culture=Neutral"/>
</configSections>
```

You then specify which provider you wish to use. The provider you select will depend on if you are using a desktop application or a web application.

```
<PDSACacheProviders>
  <providers defaultProvider="Client">
    <add providerName="Client"
         type="PDSA.Cache.Providers.PDSACacheClient"
         assembly="PDSA.Cache,Culture=neutral" />
  <providers defaultProvider="Web">
    <add providerName="Web"
         type="PDSA.Cache.Providers.PDSACacheASPNET"
         assembly="PDSA.Cache.Web,Culture=neutral" />
  </providers>
</PDSACacheProviders>
```

Caching Data in Desktop Application

When using a desktop application such as WPF, a Windows Service or Windows Forms you will need to create a “Global” variable that will hold an instance of the PDSACacheManager object. Once you have this variable created you can then **Add** and **Get** data from the cache as shown in the following code.

```
PDSACacheManager _Cache = new PDSACacheManager();

private void btnAddKey_Click(object sender, RoutedEventArgs e)
{
  _Cache.Provider.Add("CustomerId", 1);
}

private void btnGetKey_Click(object sender, RoutedEventArgs e)
{
  Debug.WriteLine(_Cache.Provider.Get("CustomerId"));
}
```

User Specific Cache Data

On desktop applications it is not as important to have user-specific data as there is only one application running for that one user. However, if you maybe need to cache data on a server for your desktop application, then this might make sense. The **Add** method has an overload that you can use to pass in a unique identifier for this user. You could use their User Identity or maybe a unique GUID.

```
private Guid _UserKey = System.Guid.NewGuid();

private void btnAddUserKey_Click(object sender, RoutedEventArgs e)
{
  _Cache.Provider.Add("CustomerId", 1, @"xyz\bjones");
  _Cache.Provider.Add("CustomerId", 1, _UserKey.ToString());
}

private void btnGetUserKey_Click(object sender, RoutedEventArgs e)
{
  Debug.WriteLine(_Cache.Provider.Get("CustomerId",
                                      @"xyz\bjones"));
  Debug.WriteLine(_Cache.Provider.Get("CustomerId",
                                      _UserKey.ToString()));
}
```

Absolute Expiration on Cache Data

To have some data that expires in exactly 10 seconds, you can add that data using the following code:

```
private void btnAddAbsolute_Click(object sender, RoutedEventArgs e)
{
  _Cache.Provider.Add("CustomerId", 1,
                          DateTime.Now.AddSeconds(10));
}
```

Sliding Expiration on Cache Data

To have some data that expires either at a specified time interval, or if the data is accessed then that data is renewed for the same specified time interval, you can use the **Add** method that accepts a TimeSpan object:

```
private void btnAddSlidingExpiration_Click(object sender,
 RoutedEventArgs e)
{
  _Cache.Provider.Add("CustomerId", 1, new TimeSpan(0, 0, 10));
}
```

Removing Data from Cache

You can remove individual items from the cache using the **Remove** method. You can also use the **Clear** method to remove all items from the cache.

```
private void btnRemove_Click(object sender, RoutedEventArgs e)
{
  _Cache.Provider.Remove("CustomerId");
  _Cache.Provider.Remove("CustomerId", @"xyz\bjones");
  _Cache.Provider.Remove("CustomerId", _UserKey.ToString());

  _Cache.Provider.Clear();
}
```

Caching Data in a Web Application

The PDSA Cache for ASP.NET Applications works exactly the same as the above client-side caching provider. The only difference is there is an additional set of methods called **AddForUser**. These methods automatically set the UserKey to HttpContext.Current.Session.SessionId and call the appropriate Add method passing in this value.

Assemblies

PDSA.Framework

Configuration Provider

These provider classes help you retrieve and save configuration information for your application. You can retrieve and save configuration settings from five locations with the providers that come with the PDSA Framework.

- The <appSettings> section in your .Config file
- An XML file
- The Registry on the local machine
- The Registry on a remote machine
- The **pdsaApplicationSetting** table

You may also roll-your-own provider to store settings in any other data store you wish.

Sample Folder

[InstallFolder]\Samples\WebForms\PDSAConfigurationSample

Configuration

To use the PDSA Configuration system you will first need to add the following <section> to the <configSections> element at the top of your .config file.

```
<configSections>
  <section name="PDSAConfigurationProviders"
           type="PDSA.Configuration.Configuration.
                 PDSAConfigSectionHandler,
                 PDSA.Configuration, Culture=Neutral"/>
</configSections>
```

Next you add a <PDSAConfigurationProviders> section anywhere within your .config file.

```
<PDSAConfigurationProviders>
  <providers defaultProvider="AppSettings"
             userLanguage="">
    <add providerName="AppSettings"
         location="[ConfigFile]"
         xpath="/configuration/appSettings"
         type="PDSA.Configuration.Providers.
                 PDSAConfigProviderAppSettings"
         assembly="PDSA.Configuration"/>
    <add providerName="Xml"
         location="[AppPath]\Settings.xml"
         xpath="/appSettings"
         type="PDSA.Configuration.Providers.PDSAConfigProviderXml"
         assembly="PDSA.Configuration"/>
    <add providerName="Registry"
         location="Software\PDSAConfigSample"
         RemoteMachine=""
         Hive="HKEY_LOCAL_MACHINE"
         type="PDSA.Configuration.Providers.
               PDSAConfigProviderRegistry"
         assembly="PDSA.Configuration"/>
  </providers>
</PDSAConfigurationProviders>
```

Read Configuration Settings

To read data from your default provider, which in the code above would be from the <appSettings> section in your configuration file you would use code like the following:

```
private void ReadUsingDefaultProvider()
{
  PDSAConfigManager mgr = null;

  // Create a new Configuration Manager object
  mgr = new PDSAConfigManager();

  // The first time you access the "Provider" property it will
  // get the default provider from the .Config file to use
  // to return the values requested
  Debug.WriteLine(mgr.Provider.GetSetting("AppName", "N/A"));
  Debug.WriteLine(mgr.Provider.GetSetting("UserTracking", false));
  Debug.WriteLine(mgr.Provider.GetSetting(
         "FormTimeOutInSeconds", -1));
}
```

You use the GetSetting method on the configuration provider to retrieve a value. This method has several overloads. The first parameter is always the key name you wish to retrieve. The second parameter is the value to return if the key is not found in the configuration storage location. This second parameter can accept a string, integer, boolean and many other data types.

Save Configuration Settings

You can save data back to any of the providers using the SaveSetting method of the provider. You pass in the name of the key and the value to save.

```
private void SaveConfigSettings()
{
  PDSAConfigManager mgr = null;

  mgr = new PDSAConfigManager();

  // Get the XML provider
  mgr.Provider = mgr.GetProvider("Xml");

  // Save the settings back
  mgr.Provider.SaveSetting("AppName", "Changed App Name");
  mgr.Provider.SaveSetting("UserTracking", "false");
  mgr.Provider.SaveSetting("FormTimeOutInSeconds", "55");
}
```

Assemblies

PDSA.Framework

Cryptography Provider

The Cryptography provider classes help you hash and encrypt/decrypt information for your application. You can perform the following with the Cryptography providers:

- Encrypt and Decrypt values using TripleDES, DES, RC2, Rijndael, and AES
- Hash values using SHA1, SHA256, SHA385, SHA512 and MD5

You may also roll-your-own provider to hash or encrypt/decrypt values using any algorithm you wish.

Sample Folders

[InstallFolder]\Samples\WPF\PDSACryptographyLibrarySample

and

[InstallFolder]\Samples\WPF\PDSACryptographyProviderSample

Configuration

To use the PDSA Cryptography system you will first need to add the following <section>'s to the <configSections> element at the top of your .config file.

```
<configSections>
  <section name="PDSASymmetricProviders"
           type="PDSA.Cryptography.Configuration.
                 PDSASymmetricSectionHandler, PDSA.Cryptography,
                 Culture=Neutral"/>
  <section name="PDSAHashProviders"
           type="PDSA.Cryptography.Configuration.
           PDSAHashSectionHandler, PDSA.Cryptography,
           Culture=Neutral"/>
</configSections>
```

Next you add the appropriate section that you need. For example, if you are going to be using the PDSA Symmetric providers for encryption and decryption you will need to add the <PDSASymmetricProviders> section in your .Config file.

```
<PDSASymmetricProviders>
  <providers defaultProvider="TripleDES">
    <add providerName="TripleDES"
         type="PDSA.Cryptography.Providers.PDSASymmetricTripleDES"
         assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="DES"
         type="PDSA.Cryptography.Providers.PDSASymmetricDES"
         assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="RC2"
         type="PDSA.Cryptography.Providers.PDSASymmetricRC2"
         assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="Rijndael"
         type="PDSA.Cryptography.Providers.PDSASymmetricRijndael"
         assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="Aes"
         type="PDSA.Cryptography.Providers.PDSASymmetricAes"
         assembly="PDSA.Cryptography,Culture=neutral"/>
  </providers>
</PDSASymmetricProviders>
```

If you are going to be using the PDSA Hashing providers then you need to add the <PDSAHashProviders> section to your .Config file.

```
<PDSAHashProviders>
  <providers defaultProvider="SHA1">
    <add providerName="SHA1"
          type="PDSA.Cryptography.Providers.PDSAHashSHA1"
          assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="SHA256"
          type="PDSA.Cryptography.Providers.PDSAHashSHA256"
          assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="SHA384"
          type="PDSA.Cryptography.Providers.PDSAHashSHA384"
          assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="SHA512"
          type="PDSA.Cryptography.Providers.PDSAHashSHA512"
          assembly="PDSA.Cryptography,Culture=neutral"/>
    <add providerName="MD5"
          type="PDSA.Cryptography.Providers.PDSAHashMD5"
          assembly="PDSA.Cryptography,Culture=neutral"/>
  </providers>
</PDSAHashProviders>
```

Encrypting and Decrypting

To encrypt and decrypt a string using TripleDES for example, you can write code like the following:

```
private void EncryptUsingHardCodedKeys()
{
  PDSASymmetricManager mgr = new PDSASymmetricManager();
  string encryptedString;

  mgr.Provider = mgr.GetProvider("TripleDES");
  mgr.Provider.KeyString = "0Ixly1qRWl2BYBrB+dYl2rPbzo2j2aes";
  mgr.Provider.IVString = "uimngi1dPlw=";

  encryptedString = mgr.Provider.Encrypt(
       @"Server=Localhost;Database=Sandbox;uid=sa;pwd=P@ssw0rd");
  MessageBox.Show(encryptedString);

  MessageBox.Show(mgr.Provider.Decrypt(encryptedString));
}
```

Notice that after filling in the KeyString and IVString properties, you simply call the Encrypt() method on the Provider and a base-64 encoded string will be returned back to your method. You can then pass that encrypted string to the Decrypt() method to return it back into a human-readable string.

You can also have the Key and IV values generated automatically using the GenerateKey and GenerateIV methods.

```
private void EncryptConnectString()
{
  PDSASymmetricManager mgr = new PDSASymmetricManager();
  string encryptedString;

  mgr.Provider.GenerateKey();
  mgr.Provider.GenerateIV();

  encryptedString = mgr.Provider.Encrypt(
       @"Server=Localhost;Database=Sandbox;uid=sa;pwd=P@ssw0rd");

  MessageBox.Show(encryptedString);

  MessageBox.Show(mgr.Provider.Decrypt(encryptedString));
}
```

Hashing

Hashing a value is very easy as there is no key or initialization vector required. You simply choose the hashing algorithm to use and call the CreateHash method.

```
private void GenerateHashUsingDefault()
{
  PDSAHashManager mgr = new PDSAHashManager();

  MessageBox.Show(mgr.Provider.CreateHash("MyPassword"));
}
```

Of course to make your values harder to guess, you can always add on a “salt” value. This is just any random characters you wish.

```
private void GenerateHashUsingDefaultWithSalt()
{
  PDSAHashManager mgr = new PDSAHashManager();

  mgr.Provider.UseSalt = true;
  mgr.Provider.SaltValue = "somesaltvalue";
  MessageBox.Show(mgr.Provider.CreateHash("MyPassword"));
}
```

Each Hash provider does provide a CreateSalt() method to randomly generate a salt value for you.

```
private void GenerateHashWithSalt()
{
  PDSAHashManager mgr = new PDSAHashManager();

  mgr.Provider = mgr.GetProvider("SHA1");
  mgr.Provider.SaltValue = mgr.Provider.CreateSalt();
  mgr.Provider.UseSalt = true;

  MessageBox.Show("Salt: " + mgr.Provider.SaltValue);
  MessageBox.Show(mgr.Provider.CreateHash("MyPassword"));
}
```

Assemblies

PDSA.Framework

Data Layer Provider

The Data Layer provider classes help you create a consistent interface to and simplify ADO.NET. With our Data Layer you can target any database server and keep your user interface consisten. The Data Layer is the underlying classes that are called from all of the Haystack generated data classes. You can perform the following operations with the Data Providers:

- Create connections, commands, parameters and all other ADO.NET objects generically using the ADO.NET Interfaces. This allows for future adaptability and to easily create new providers
- Added a Transaction collection object to support transactions across your data objects

You may also roll-your-own provider to talk to any data provider you wish.

Sample Folder

[InstallFolder]\Samples\WPF\PDSADataLayerProviderSample

Configuration

To use the PDSA Data Providers you will first need to add the following <section> to the <configSections> element at the top of your .config file.

```
<configSections>
  <section name="PDSADataProviders"
          type="PDSA.DataLayer.Configuration.
                PDSADataSectionHandler, PDSA.DataLayer"/>

</configSections>
```

Next you add the <PDSADataProviders> section to your .Config file.

```
<PDSADataProviders>
  <providers defaultProvider="SqlClient">
    <add providerName="SqlClient"
          useStoredProcedures="false"
          useDBAuditTracking="false"
          MinDate="1753-1-1"
          MaxDate="2030-12-31"
          DBLanguage="en-US"
          DBDateFormat="yyyy-MM-dd HH:mm:ss"
          connectStringName="SqlClient"
          type="PDSA.DataLayer.Providers.PDSADataSqlClient"
          assembly="PDSA.DataLayer"/>
  </providers>
</PDSADataProviders>
```

There are some unique attributes attached to each Data Provider. These attributes control the provider or control the data classes that are generated from Haystack. These attributes will be discussed in the next section. One of the most important attributes is connectStringName. This attribute refers to the “name” attribute in the <connectStrings> section as shown below:

```
<connectionStrings>
  <add name="SqlClient"
        connectionString="Server=Localhost;
        Database=PDSASamples;Integrated Security=Yes;
        Persist Security Info=False;App=DataLayerSample"/>
</connectionStrings>
```

Attributes in Data Provider

The following is a list of the different attributes in each data provider. Each of these attributes can be overridden by the programmer at runtime.

Attribute	Description
DBLanguage	This is the language the database server is running.
DBDateFormat	This is the format that the database server expects its dates to be submitted in. The generated code will take care of putting the dates into the appropriate format regardless of the format they are entered on the UI.
useStoredProcedures	This flag can be used by the data classes to use the stored procedures generated.

useDBAuditTracking	Whether or not the generated data classes should generate the XML audit tracking string for each INSERT, UPDATE and DELETE.
MinDate	The minimum date the database server will accept.
MaxDate	The maximum date the database server will accept.
connectStringName	This string must match the "name" attribute in one of the <add> elements within the <connectionStrings> element.

There are way too many methods that you can use in these data providers. So it will be best to open the samples to check out the various methods. Like all of the PDSA providers, you should start with the PDSADataManager class as shown below.

```
private void GetDataSetHardCoded()
{
  DataSet ds;
  PDSADataManager mgr = new PDSADataManager();

  ds = mgr.Provider.GetDataSet("SELECT * FROM Product",
     "Server=Localhost;Database=Sandbox;Integrated Security=Yes");
}
```

Assemblies

PDSA.DataLayer

Database Schema Provider

The Data Schema provider classes help you read schema data from SQL Server. Haystack uses these providers to perform its job. You will not likely use these too much. You may also roll-your-own provider to talk to any database schema you wish.

Sample Folder

[InstallFolder]\Samples\WPF\PDSADBSchemaProviderSample

Configuration

To use the PDSA Database Schema Providers you will first need to add the following <section>'s to the <configSections> element at the top of your .config file.

```
<configSections>

  <section name="PDSADataProviders"
           type="PDSA.DataLayer.Configuration.
                 PDSADataSectionHandler, PDSA.DataLayer "/>

  <section name="PDSADBSchemaProviders"
           type="PDSA.DataLayer.Schema.Configuration.
                 PDSADBSchemaSectionHandler, PDSA.DBSchema "/>

</configSections>
```

Next you add the <PDSADBSchemaProviders> section to your .Config file.

```
<PDSADBSchemaProviders>
  <providers defaultProvider="SqlClient">
    <add providerName="SqlClient"
         filterOutSchemas="INFORMATION_SCHEMA,sys,"
         userNameIsSchemaName="False"
         upperCaseSchemaNames="False"
         dataProviderName="SqlClient"
         type="PDSA.DataLayer.Schema.Providers.
               PDSADBSchemaSqlClient"
               assembly="PDSA.DBSchema, Culture=neutral"/>
  </providers>
</PDSADBSchemaProviders>
```

There are some unique attributes attached to each DBSchema Provider. These attributes help the provider gather information from the underlying schema. These attributes will be discussed in the next section. One of the most important attributes is dataProviderName. This attribute refers to the “providerName” attribute in the <PDSADataProviders> section as shown below:

Attributes in DBSchema Provider

The following is a list of the different attributes in each data provider. Each of these attributes can be overridden by the programmer at runtime.

Attribute	Description
filterOutSchemas	Set this attribute to a comma-delimited list of schemas that you do not wish to see when retrieving objects from your catalog/database.
userNameIsSchemaName	If the user name in your connection string is the schema name in your catalog (as it is in Oracle), then set this attribute to “true”.
upperCaseSchemaNames	If all schema names are listed in upper case (as they are in Oracle), set this attribute to “true”.

There are way too many methods that you can use in these schema providers. So it will be best to open the samples to check out the various methods. Like all of the PDSA providers, you should start with the PDSADBSchemaManager class as shown below.

```
private void AllCatalogsLoadHardCoded()
{
  PDSADBSchemaManager mgr = new PDSADBSchemaManager();

  lstData.ItemsSource = mgr.Provider.GetCatalogs();
}
```

Assemblies

PDSA.DataLayer

Logging Provider

This namespace contains provider classes to help you log information from your application. Logs can be informational, debug, exceptions, warning, audit tracking, user tracking, or any custom data you wish to log. There are a few different providers to help you log data to a table, a file, the event log or send an email with the log data.

Sample Project

[InstallFolder]\Samples\WPF\PDSALoggingProviderSample

Things you can Log

There are 8 different types of logging you can record in the PDSA Logging System. They are the following:

- Informational
- Exception
- Audit
- Debug
- Warning
- UserLogin
- UserLogout
- UserVisit

Configuration

For each provider listed in the App.Config or Web.Config file you can specify a semi-colon delimited list of which ones you want to record in which provider. For example, in the EventLog provider you can see that the 'logTypes' attribute contains only 3 of the 8.

```
<add providerName="eventLog"
     mode="On"
     logSystemInfo="false"
     logTypes="Informational;Warning;Exception"
     logValueOnly="false"
     type="PDSA.Logging.Providers.PDSALogEventLog"
     assembly="PDSA.Logging,Culture=neutral"
     logName="Application"/>
```

In the SqlServer provider there are all 8 listed.

```
<add providerName="SqlServer"
     mode="On"
     logSystemInfo="true"
     useStoredProcedures="true"
     logTypes="Informational;Exception;Debug;
               Audit;Warning;UserLogin;UserLogout;UserVisit"
     logValueOnly="false"
     type="PDSA.Logging.Providers.PDSALogSqlServer"
     assembly="PDSA.Logging.SqlServer,Culture=neutral"
     dataProviderName="SqlClient"/>
```

If you log an Exception it will go to both the EventLog and the SQL Server because the 'mode' attribute is 'on' for both, and both have 'Exception' listed in their 'logTypes'. However, if you log a 'UserLogin' it will only go to the SQL Server provider because EventLog does not list 'UserLogin' in the 'logTypes'.

Using the Logging Provider

After you have the log provider configured, you create an instance of the PDSALoggingManager class and call the Log() method passing in some data.

```
PDSALoggingManager mgr = new PDSALoggingManager();

mgr.Log("Data to Log");
```

You may also pass in any extra values to the Log() method in the form of a NameValueCollection object. The key/value pairs you add to this collection will be written to the log file with the other information to log.

```
PDSALoggingManager mgr = new PDSALoggingManager();

// Publish some extra values along with the exception information
NameValueCollection nvc = new NameValueCollection();

nvc.Add("Value1", "This is Value 1");
nvc.Add("Value2", "This is Value 2");

// Just use the 'Log' method for informational logs
mgr.Log("Data to Log", nvc);
```

Logging Exceptions

The PDSA Logging System is a great way to record exceptions that happen in your application. Within your Catch block you will want to call the LogException() method passing in your Exception object as shown below:

```
try
{
  // Do something to create an exception
  throw new ArgumentNullException("BadArgument");
}
catch (Exception ex)
{
  PDSALoggingManager mgr = new PDSALoggingManager();
  // Log the exception
  mgr.LogException(ex);

  MessageBox.Show("Exception has been published.");
}
```

Just like with any of the Log() methods, you can also pass in additional information. This is especially helpful with exception information as you can pass in any variables that will help you determine where the error occurred and possibly why if you pass in a variable whos' value is incorrect.

```
try
{
  // Do something to create an exception
  throw new ArgumentNullException("BadArgument");
}
catch (Exception ex)
{
  // Publish some extra values along with the exception information
  NameValueCollection nvc = new NameValueCollection();

  nvc.Add("Value1", "This is Value 1");
  nvc.Add("Value2", "This is Value 2");

  _LogManager.LogException(ex, nvc,
         PDSAApplicationSettings.GetSettings().
             Entities.DefaultEntityId);

  MessageBox.Show("Exception has been published.");
}
```

Use the AppLog Classes

To simplify the usage of the PDSA Logging system each template project in the PDSA .NET Productivity Framework has a set of classes that start with “AppLog”. Now, in your applications you can simply call static/Shared methods like the following:

```
AppLogInfoManger.Log("Data to Log");
AppLogWarningManager.Log("Data to Log");
AppLogDebugManager.Log("Data to Log");

AppLogExceptionManager.Log(ex);
```

There are AppLog classes for each type of log you wish to write. You should look at the Samples in the PDSA .NET Productivity Framework to see examples of each one.

Resource / Localization Provider

In order to localize an application you will need to store all messages, labels, grid headers, and even the data in your tables in that user’s language. Microsoft has made storing labels and messages easy to do using resource files that are built-in to Visual Studio and .NET. But, to get a consistent system that is easy to maintain and use across your whole application this system falls way short. This is why we created our PDSA Resource system. This method provides a consistent approach to all your data localization needs. In addition, we provide a web interface to the

messages to allow remote translators to do their job. We also give you tools to help identify which messages and labels have not yet been localized.

Sample Folder

[InstallFolder]\Samples\WPF\PDSAResourceProviderSample

General Usage

To use this system with the pdsa.pdsaResource table, make sure the .Config file is set to point to the database where this table is located.

Now, when you want to retrieve a value to replace into a Label on your screen, for example, you make the following call:

```
lblFirstName.Text = AppResources.GetResource("FirstName",
                                             "Not Found");
```

The default language is set in the .config file, however you can change this at runtime by making the following call:

```
AppResources.SetLanguage("en-US");
```

Storing Resources in SQL Server

If you want to store resources in SQL Server, you will need to ensure that you have installed the scripts for the PDSA Framework located in the [InstallFolder]\SqlScripts. There are two tables that you will work with in order to use multi-lingual resources.

pdsaLanguage Table

In this table you will need to add a record for each language you wish to support in your application. In Figure 2 you can see two languages have been setup.

LanguageId	Resource...	Code	Name	NameResourceKey	Location	LocationResou...	IsActive	IsDisplay	IsDefault	DisplayOr...
1	en-us	USA	English	892a84b3-e6f0-477...	United States	dfa20faa-e657-...	1	1	True	1
5	es-MX	MEX	Spanish	8d868e02-fee8-4f9...	Mexico	8d868e02-fee8...	1	1	False	2
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Figure 2: Create a record in the pdsaLanguage table for each language.

In the following table is a description of each column and what it is used for.

Column	Description
ResourceLanguage	This is the standard language codes you will find in Internet Explorer under languages.

Code	The standard 3 letter abbreviation for the country.
Name	The name of this language.
NameResourceKey	
Location	A full description of the country.
LocationResourceKey	
IsActive	Set to True or 1 if you want to use this language.
IsDisplay	Set to True or 1 if you wish to display this language when editing on a maintenance screen. NOT CURRENTLY USED.
IsDefault	Set to True or 1 if this language is the default.
DisplayOrder	Set to a number that specifies in what order to display this language on a maintenance screen. NOT CURRENTLY USED.

Table 1: Columns in the pdsaLanguage table.

pdsaResource Table

The pdsaResource table is where all messages, labels, etc. are stored. Figure 3 shows an example of records in the pdsaResource table.

R...	ResourceName	ResourceNum...	ResourceClas...	Resource...	Resourc...	ResourceText	ObjectName	ObjectPrimaryK.
524	ZeroRecordsCreatedRole	*NULL*	*NULL*	*NULL*	en-US	No Role recor...	DBErrorMessages	*NULL*
525	ZeroRecordsCreatedRole...	*NULL*	*NULL*	*NULL*	en-US	No RoleEntity...	DBErrorMessages	*NULL*
526	ZeroRecordsDeleted	*NULL*	*NULL*	*NULL*	en-US	No Records w...	DBErrorMessages	*NULL*
527	ZeroRecordsReturned	*NULL*	*NULL*	*NULL*	en-US	No records w...	DBErrorMessages	*NULL*
528	ZeroRecordsUpdated	*NULL*	*NULL*	*NULL*	en-US	No Records w...	DBErrorMessages	*NULL*
926	ActiveInactive-Active	*NULL*	*NULL*	*NULL*	en-US	Active	DBErrorMessages	*NULL*
927	ActiveInactive-InActive	*NULL*	*NULL*	*NULL*	en-US	Not Active	DBErrorMessages	*NULL*
928	UserRoleAlreadyExists	*NULL*	*NULL*	*NULL*	en-US	User/Role alre...	DBErrorMessages	*NULL*
929	ReferenceTableNameDu...	*NULL*	*NULL*	*NULL*	en-US	The Referenc...	DBErrorMessages	*NULL*
100	PDSA.pdsaLanguage.Na...	*NULL*	*NULL*	892a84b3...	en-us	English	PDSA.pdsaLanguage	1
301	PDSA.pdsaLanguage.Lo...	*NULL*	*NULL*	dfa20faa-e...	en-us	United States	PDSA.pdsaLanguage	1
2...	ProductName	*NULL*	*NULL*	*NULL*	en-US	Product Name	*NULL*	*NULL*
2...	ProductName	*NULL*	*NULL*	*NULL*	es-MX	Namo Producto	*NULL*	*NULL*
N...	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Figure 3: The pdsaResource table holds all messages for your application.

In the following table is a description of each column in the pdsaResource table and what it is used for.

Column	Description
ResourceName	A unique identifier that you will use to retrieve this message.
ResourceNumber	An optional numeric identifier to further qualify this message.
ResourceClassName	An optional Class Name identifier to further qualify this message.
ResourceLanguageKey	
ResourceLanguage	
ResourceText	
ObjectName	

ObjectPrimaryKey	

Table 2: Columns in the pdsaResource table.

Assemblies

PDSA.Framework

Summary

This chapter explained some of the different provider classes available in the PDSA .NET Productivity Framework.