Migrating .NET Apps to Apprenda

A Prescriptive Guide

Cloud Enablement

Apprenda's Platform subsystems automatically assist applications in being "Cloud-ready" simply by virtue of their deployment by the Platform. Things like workload scalability, communications bus, and more are native parts of the Platform that applications sit "atop" and automatically leverage. That said, there are design patterns and architecture decisions that Developers can make during development of an app that take full advantage of the Platform's capabilities. These approaches should also be considered when migrating an app portfolio to Apprenda. This guide offers prescriptive instructions on the most common considerations that should be made while migrating a .NET web/SOA application to Apprenda. We recommend you read this document in its entirety and build a migration plan employing these "recipes" for each application you are migrating.

The document starts with common Cloud-readiness considerations, and progressively moves through leveraging Apprenda for enhanced Cloud-readiness and eventually pure SaaS.

While reviewing each task or recommendation in this guide, keep in mind these icons:


Required for Cloud readiness


Required for Apprenda readiness


Optional for Apprenda readiness

Application Structure

Componentization of Application Tiers

Often referred to as Service Oriented Architecture (SOA), modern applications should separate data, business logic and user interfaces. This architectural solution is the foundation of successful scalability of the distinct concerns consistent with the needs of the Users and availability of underlying infrastructure resources. Apprenda's Developer topics provide the basics of each tier. The benefit of organizing apps into multiple tiers is that it allows the each tier to be scaled independently of the others depending upon usage patterns and need.

> **Further Reading**
> Separation of applications into tiers for Cloud readiness:
> http://docs.apprenda.com/current/developer-topics

Best practices published by Microsoft in the early 2000's prescribed the ASP.NET WebForms model, which was a direct "port" of the WinForms programming model to the web. When Microsoft introduced Windows Communication Foundation in .NET 3.0 (2006), they made it possible to host WCF web services in IIS, providing a pseudo-path to SOA by bolting on services support to the UI tier. Therefore, many current enterprise apps are either not service-oriented at all or have employed some form of web services via the WCF-in-IIS model (.svc endpoints). While the latter approach does programmatically separate UX from business logic, it doesn't necessarily structurally decouple the two. By hosting both UX and business logic in the IIS container, the application consumes the same physical resources for both. In a properly architected service-oriented application this is not the case. Apprenda provides a separate stand-alone hosting container for WCF services (the Apprenda Service Container). By building stand-alone WCF services and allowing Apprenda to run them, a Developer can turn over several application requirements to the Platform – things such as dynamic scalability for each service are native capabilities of Apprenda.

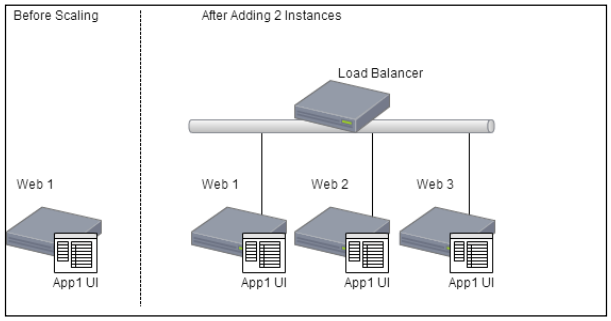


Figure 1: An application scaling at the UI tier (with automated load balancing by Apprenda software load balancer)

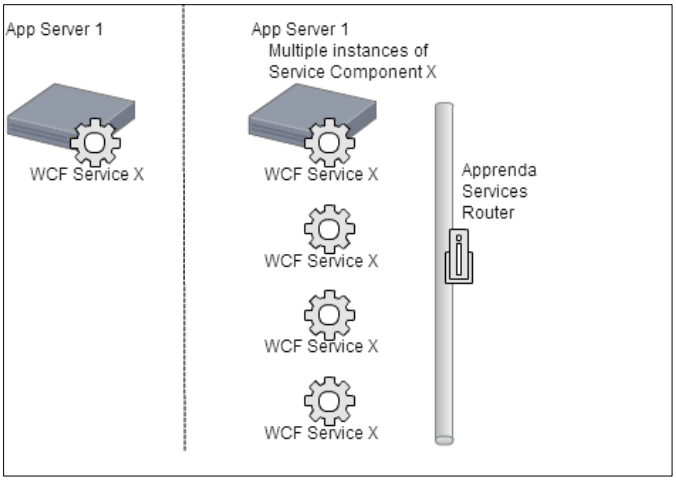
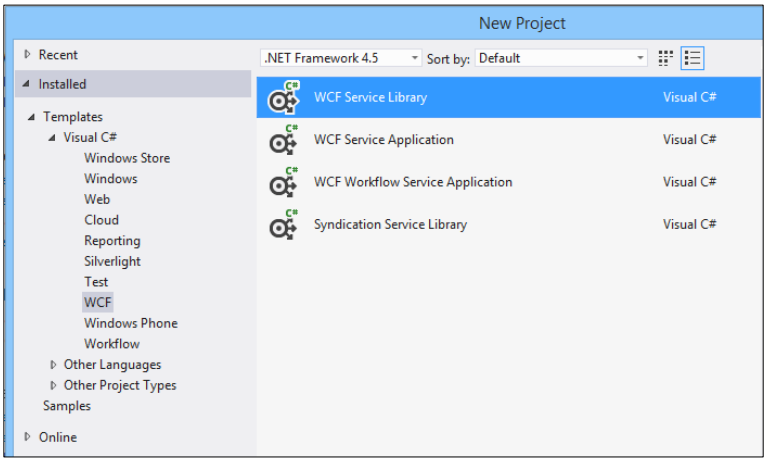


Figure 2: Stand-alone WCF services can scale horizontally (across infrastructure) or vertically (more instances) so long as there are available resources.

Building a Stand-Alone WCF Web Service

To build a standalone WCF service, start with the appropriate project in Visual Studio: **WCF Service Library**.

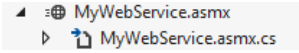


You'll use this service library to build out the interface (contract) and services (implementation) of your web service. Note: A single application may include multiple contracts and services – so thought should go into the separation of functionality. For example, if an application has a substantial amount of CRUD operations and a small administrative interface, the CRUD operations should be separated into their own web service (stand-alone WCF service library) so that service can scale (see scaling models above) independently of the administrative service functionality.

- If you are migrating from an **ASP.NET WebForms** application, consider operations that are conducted in the code-behinds (.aspx.cs) as functionality that should be moved into web services.
- If you are migrating **existing UI-based web services** (.asmx or .svc), then the service code itself is already written in the code-behinds for those services, and you can simply move the code into a **WCF Service Library**. You will need to create the interface (contract) for that service in order to publish the WCF service.

Migrating an ASMX web service from the UI tier to a stand-alone WCF service:

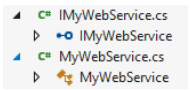
MyWebService.asmx.cs in an ASP.NET web application project:



May look like this:

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
public class MyWebService : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }
}
```

The same functionality in a standalone WCF Web Service:



MyWebService.cs:

```
public class MyWebService : IMyWebService
{
     public string HelloWorld()
     {
        return "Hello World";
     }
}
```

IMyWebService.cs

```
[ServiceContract]
public interface IMyWebService
{
    [OperationContract]
    string HelloWorld();
}
```

In order for WCF to publish and run this web service, you'll need to create the interface, or *contract* that the service exposes to its clients.

Configuration-based References Between Components

After completing the migration of your code from an ASMX web service or SVC-based web service to a stand-alone WCF service library, you'll need to configure your service so that clients can consume it. Apprenda uses a dynamic deploy-time re-configuration engine to make sure that your WCF services are properly exposed to other clients (such as your UI) once deployed to the Platform infrastructure.

> **All of your WCF service setup should be done via configuration,** not programmatically.

Here's an example of WCF configuration from our MyWebService project's App.config file contained within the system.serviceModel tag:

```
<netTcpBinding>
   <binding name="DefaultTcpBinding"
            maxReceivedMessageSize="16777216"
            listenBacklog="100"
            sendTimeout="20:00:00"
            receiveTimeout="20:00:00"
            openTimeout="20:00:00"
            closeTimeout="20:00:00">
     <security mode="None"/>
     <readerQuotas maxArrayLength="16777216"/>
   </binding>
 </netTcpBinding>


 <service name="MyWebService.MyWebService"
          behaviorConfiguration="ApprendaServiceBehavior">
  <!-- Expose an endpoint for direct communication
       by other services or by server-side website -->
  <endpoint address="net.tcp://localhost:40000/Service"
            binding="netTcpBinding"
            bindingConfiguration="DefaultTcpBinding"
            behaviorConfiguration="ApprendaEndpointBehavior"
            contract="MyWebService.IMyWebService" />
 </service>
```

In production, WCF services are always shared between clients and their location on underlying Platform resources can vary at any time. Allowing Apprenda to manage the services and their endpoints via configuration means the Developer can focus on the business logic of the service and use Apprenda to take advantage of scaling and resource management. Visit http://docs.apprenda.com/current/wcf-web-services for more information.

> **Client setup should be done via configuration**, not programmatically.

The same holds true for **WCF clients** to services on the Platform. WCF clients use service "proxies" – programmatic in-process representations or out of process (service-oriented) services. So, the final step in separating your web service functionality into a stand-alone web service is creating a proxy in your UI/client and building the WCF configuration in the Web.config (or other mechanism depending on the

client). Apprenda has published many sample applications that have client→service configuration and proxy examples: Taskr, Calculator, and more. See these example on the Dashboard of your Legacy Apprenda Developer Portal for more examples of client setup and configuration.

Database Connections and Tokenization

> Database connection strings, resource locations and service endpoints should all be specified in configuration.

Part of Apprenda's dynamic configuration engine is a tokenization system. **Database connection strings, resource locations and service endpoints should all be specified in configuration** and tokenized for dynamic deployment on Apprenda resources. http://docs.apprenda.com/current/app-config-tokens.

> Remove all local machine dependencies.

Network and other configuration details must be machine-independent in order for applications to be easily scalable and available. Apprenda manages the details of the hardware and network infrastructure so the Developer doesn't have to. For an overview of the functional nature of this feature, visit http://docs.apprenda.com/current/apprenda-deployment. Cloud-readiness means that any dependencies on a specific operating system construct, **like the Windows Registry**, should be removed.

Common Pitfalls

In-memory State: Using the Apprenda Platform gives Developers the advantage of not needing to know the Cloud infrastructure details. This feature may require a change to how an application accesses state information. As a result, avoid tying your application to a specific network location or machine instance managed by the Platform. It may work but it will likely misbehave. Use the Apprenda Platform's built-in mechanisms to handle state instead. One of the built-in Platform solutions is caching. Caching using the Apprenda API will allow state information to easily be stored and retrieved without compromising your SOA solution. In addition, it allows this information to be maintained within a specific context. See the section of this document titled **Taking Advantage of Apprenda** for further details. Common ASP.NET mechanisms for storing state that should be removed for Cloud-readiness are:

- HTTPContext.Session
- HTTPContext.Current
- HTTPApplicationState

Furthermore, because the Apprenda Platform can run multiple instances of your ASP.NET application on various servers at once, move business logic that executes during specific application runtime lifecycle stages. For example application start (Application_Start in global.asax). If this logic remains, it will execute each time an instance of your UI is started on a server.

Authentication: Applications which previously had components to authenticate Users should now rely on the Apprenda Platform for authentication. Using Platform authentication enables SSO for all applications according the needs and requirements of the enterprise. Visit the section entitled "Apprenda's Authentication Gateway" in our Platform documentation here http://docs.apprenda.com/current/web-app-tier. In a default ASP.NET application template, an approach to authentication might look like this:

```
<asp:LoginView runat="server" ViewStateMode="Disabled">
        <AnonymousTemplate>
                <ul>
                        <li><a id="registerLink" runat="server" href="~/Account/Register">Register</a></li>
                        <li><a id="loginLink" runat="server" href="~/Account/Login">Log in</a></li>
                </ul>
        </AnonymousTemplate>
        <LoggedInTemplate>
                <p>
                Hello, <a runat="server" class="username" href="~/Account/Manage" title="Manage your account">
                <asp:LoginName runat="server" CssClass="username" /></a>!
                <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl="~/" />
                </p>
        </LoggedInTemplate>
</asp:LoginView>
```

When an application defers authentication to the Apprenda Platform's mechanism, a User accessing the application is presumed to be authenticated against the Platform, so the need to have multiple logic paths (for anonymous vs. logged in) is removed. The code above can be simplified to just the part highlights in yellow.

Leveraging the Apprenda Platform

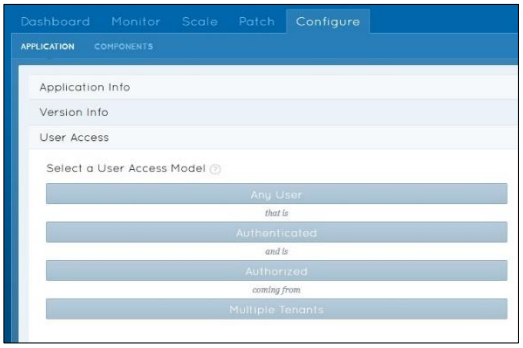
So far, this document has covered common topics that Developers should consider *about their existing code* when migrating an app to the Apprenda Platform.

> The Platform offers a wide arrange of functionality that guest applications can take advantage of simply by running on the Platform and more advanced capabilities by tapping into the different runtime and service-based APIs available.

The following sections outline some key aspects that should be considered when moving to Apprenda to enhance existing applications.

Wiring-In Platform Authentication

The Apprenda Platform, as mentioned above, offers authentication capabilities to guest applications. There are a few ways to take advantage of this capability depending on the level of customization and control desired. Enabling authentication on the Platform is as simple as turning on the functionality through the Configure tab in the Developer Portal UX's application overview or the DeploymentManifest.xml file. The following resources explain both approaches (http://docs.apprenda.com/current/app-level-redesign#useraccess, http://docs.apprenda.com/current/deployment-manifest#settings). To allow the Platform to specify authentication and identity information for you application, remove all logic in the application that accepts credentials and initiates identity information (storing username, etc. in session state). Instead, access User identity information via the Apprenda UserContext runtime API. This context will be populated for you by virtue of the fact that the Platform has authenticated the User prior to their access to the app. There are two options for enabling authentication at the Platform level for your app:

Application Settings in the Developer Portal	DeploymentManifest.XML
 http://docs.apprenda.com/current/app-level-redesign#useraccess	`<applicationServices level="Authentication" />` http://docs.apprenda.com/current/deployment-manifest#settings

Refactor In-Memory State to Apprenda's Cache

As mentioned in the previous section, when developing applications for the Cloud it is important to avoid using in-memory state to store session information.

> Apprenda offers distributed caching capabilities that are available to any guest application workload running on the Platform.

The Apprenda Cache is kept in sync across the entire grid automatically so your application will have the latest information readily available without having to worry about the replication, storage or retrieval itself. Access to the cache can be scoped and controlled at the following levels:

1. User
2. Application
3. Tenant
4. Platform

For more information on how to utilize the cache, please refer to:
http://docs.apprenda.com/current/caching

Storing information in the HTTP session:

```
//Storing in Session
Session["MyData"] = "My Data";
//Retrieving
var myData = Session["MyData"];
```

The problem with this approach in the Cloud is that requests are not guaranteed to be serviced by the same workload (instance) of an app repeatedly. Information stored in memory is not available across instances of an app workload.

Storing the same information in the Apprenda Cache (note the cache expiration of 5 minutes):

```
//Storing in the Cache
UserContext.Instance.Cache.Insert("MyData", "My Data", TimeSpan.FromMinutes(5));
//Retrieving Cache
var myData = UserContext.Instance.Cache.Find<string>("MyData");
```

Notice that the data is cached at the UserContext scope. This means that the cached data is unique to the User that put the data into the cache. The data is available to that User across the entire Platform until it expires.

Centralized Logging

One of the most important consideration when developing Cloud applications is log management. As the footprint of your application grows, it becomes more difficult to track where errors occur. Apprenda offers a logging API that facilitates centralized logging from disparate application workloads. These log messages are stored centrally along with contextual information about the application, User, Tenant (if applicable) and machine one which the event occurred. Developers can then manage their logs through the Developer Portal and Platform Operators can monitor all logs across the Platform. For more information on the logging interface, please refer to: http://docs.apprenda.com/current/logging.

Remove use of the Windows Event system (local machine dependency) and replace with Apprenda's distributed logging mechanism for a centralized view of events in disparate application workloads.

Writing to the Windows Event logs:

```
sSource = "dotNET Sample App";
sLog = "Application";
sEvent = "Sample Event";

if (!EventLog.SourceExists(sSource))
EventLog.CreateEventSource(sSource,sLog);
EventLog.WriteEntry(sSource,sEvent);
```

Writing to the Apprenda logs:

```
private static readonly ILogger log = LogManager.Instance().GetLogger(typeof(MyService));

if (log.IsErrorEnabled)

{

   log.Error("Sample Event");

}
```

Notice that we do not need to specify meta-information about the log event such as the application name or source. This information is inferred by Apprenda as the event is logged.

Leveraging Apprenda for SaaS

Apprenda allows Developers to instantaneously convert a plain .NET application into a SaaS offering with just a click of a button. By using our APIs, Developers can focus on the core functionality of their application, while letting Apprenda orchestrate such constructs as app-level multi-tenancy, data modeling, authorization, and metering. The following section will demonstrate how to leverage the Apprenda APIs to satisfy common SaaS requirements.

Apprenda Runtime Contexts

Apprenda surrounds all Platform requests with various contextual information. During your code's execution on the Platform, these contexts contain additional information such as tenant-specific connection strings, the name of the User that owns the executing thread, their permissions within the scope of your app, and more. The following contextual information exists for all requests to a SaaS application on Apprenda:

- Session – the current User's existing session
- Request – the specific request to your app that's currently executing
- Tenant – the Tenant that owns the currently-executing thread
- Provider - you
- User – the User that owns the currently-executing thread
- Subscription – the metadata that specifies the current User's authorized capabilities within your app

Detailed information about these runtime context can be found here http://docs.apprenda.com/current/contexts.

Abstracting .NET Membership Providers

Use of the ASP.NET Membership and Role Provider should be augmented to make use of the Apprenda contexts. A user accessing a SaaS application on Apprenda is presumed to already be authenticated and identified, so information accessed through the membership provider is redundant.

> The approach that minimizes code changes is to create a membership provider implementation that is backed by Apprenda's API and overrides some of the default membership provider methods, and then wire in that implementation via configuration.

An example membership provider custom override:

```
public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize, out int totalRecords)
{
    var result = TenantContext.Current.GetUsers(pageSize, pageIndex);
    totalRecords = (int)result.SearchedRowCount;

    var membershipUsers = new MembershipUserCollection();
    foreach (var user in result.Users)
    {
        membershipUsers.Add(new MembershipUser("Apprenda", string.Format("{0}{1}",     user.FirstName,user.LastName),
user.Id, user.Email, null, null, true, false, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now));
    }
    return membershipUsers;
}


public override MembershipUserCollection FindUsersByName(string usernameToMatch, int pageIndex, int pageSize, out int
totalRecords)
{
    var result = TenantContext.Current.SearchUsers( usernameToMatch, UserInfoColumns.FullName,   pageSize, pageIndex);
    totalRecords = (int) result.SearchedRowCount;

    var membershipUsers = new MembershipUserCollection();
    foreach (var user in result.Users)
    {
        membershipUsers.Add(new MembershipUser("Apprenda", string.Format("{0} {1}", user.FirstName, user.LastName),
user.Id, user.Email, null, null, true, false, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now));
    }
    return membershipUsers;
}


public override MembershipUserCollection FindUsersByEmail(string emailToMatch, int pageIndex, int pageSize, out int
totalRecords)
{
    var result = TenantContext.Current.SearchUsers(emailToMatch, UserInfoColumns.Email, pageSize, pageIndex);

    totalRecords = (int) result.SearchedRowCount;

    var membershipUsers = new MembershipUserCollection();
    foreach (var user in result.Users)
    {
        membershipUsers.Add(new MembershipUser( "Apprenda",string.Format("{0} {1}", user.FirstName, user.LastName),
user.Id, user.Email, null, null, true, false, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now, DateTime.Now));
    }
    return membershipUsers;
}
```

An example role provider custom override:

```
public override bool IsUserInRole(string username, string roleName)
{
return UserContext.Instance.IsAuthorized(roleName);
}


public override string[] GetRolesForUser(string username)
{
    var currentUser = UserContext.Instance.CurrentUser;
    if (username.Equals(string.Format("{0}{1}", currentUser.FirstName, currentUser.LastName)))
    {
       return ApplicationContext.Current.GetSecurables()
       .Select(s => s.Name)
       .Where(s => UserContext.Instance.IsAuthorized(s))
       .ToArray();
    }
    throw new InvalidOperationException("You should not be requesting roles for another user.");
}

public override string[] GetAllRoles()
{
    return ApplicationContext.Current.GetSecurables().Select(s => s.Name).ToArray();
}

public override bool RoleExists(string roleName)
{
    return ApplicationContext.Current.GetSecurables().Any(s => s.Name.Equals(roleName));
}
```

With implementations of these classes now incorporated into your application, you'll need to include them via configuration in the web.config file for your project:

```
<roleManager defaultProvider="ApprendaRoleProvider"
      enabled="true"
      cacheRolesInCookie="true"
      cookieName=".ASPROLES"
      cookieTimeout="30"
      cookiePath="/"
      cookieRequireSSL="false"
      cookieSlidingExpiration="true"
      cookieProtection="All" >
      <providers>
        <clear />
        <add
          name="ApprendaRoleProvider"
          type="Apprenda.ApprendaRoleProvider"
          applicationName="TestApp"
          writeExceptionsToEventLog="true" />
      </providers>
    </roleManager>

<membership defaultProvider="ApprendaMembershipProvider" userIsOnlineTimeWindow="15">
      <providers>
        <clear />
        <add
          name="ApprendaMembershipProvider"
          type="Apprenda.ApprendaMembershipProvider"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          requiresUniqueEmail="true"/>
      </providers>
    </membership>
```

And lastly, wire up the identity principal by pulling from the Apprenda contexts in the global.asax:

```
protected void Application_PostAcquireRequestState(object sender, EventArgs e)
{
          if (HttpContext.Current == null || !(HttpContext.Current.Handler is IRequiresSessionState))
          {
                    return;
          }

          var identity = new GenericIdentity(UserContext.Instance.CurrentUser.Email);
          var username = string.Format("{0}{1}", UserContext.Instance.CurrentUser.FirstName,
           UserContext.Instance.CurrentUser.LastName);
          var principal = new GenericPrincipal(identity, Roles.GetRolesForUser(username));
          Context.User = principal;
}
```

An application employing membership and role provider typically has expectations for roles (and logic that dictates behavior based on role). Now that the roles are backed by Apprenda Tenant roles, you will have to publish *Securables* along with your application so that the same logic can be performed based on Tenant roles. This can be achieved through the Deployment Manifest file or through the Legacy Developer Portal *Securables* section. Afterwards, you can map the Users to roles and assign them *Securables* if desired. More information on how to achieve this can be found here: http://docs.apprenda.com/current/security.

With this solution implemented in the application, standard ASP.NET membership role checks should function as before, without code changes:

```
var user = HttpContext.Current.User;
if(user.IsInRole("Sample Role")
{
          //code here
}
```

Using ActionFilters in ASP.NET MVC

> If your application is an ASP.NET MVC web app, you can achieve a similar runtime effect by employing ActionFilters on controller actions.

Create a new Class that extends the *ActionFilterAttribute* class. Override the *OnActionExecuting* method as follows:

```
public class ContextActionFilter : ActionFilterAttribute
{
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
              HttpContext.Current.User = new GenericIdentity(UserContext.Instance.CurrentUser.Email);
              base.OnActionExecuting(filterContext);
        }
}
```

This override does effectively the same thing that our global.asax example did – it replaces the prinicipal User for the current HttpContext with the Apprenda UserContext identity. In this case, however, you can simply use this attribute on any controller actions that you want this information to be available.

```
[ContextActionFilter]
public ActionResult Index()
{
   //In this action method, HttpContext.Current.User will be populated by Apprenda API data
   return View();
}
```

Metering Basics Using the Apprenda API

Developers can include programmatic API references to instruct the Apprenda Platform to track the utilization of specific functionality within an application. The Platform will match the 'Feature' marked in the implementation of the functionality with a component specified in the Subscription assigned to the user. For details, please see: http://docs.apprenda.com/current/features and http://docs.apprenda.com/current/features-and-editions.

There are four types of features that can be tagged:

- Toggle
- Boundary
- Block
- Limiter

The following example demonstrates how to declare a limiter feature called 'Projects' and apply it to functionality – in this case a method on a WCF web service.

```
[OperationContext]
[IncrementLimiter("Projects")]
Public void CreateProject(Project project)
{
    ProjectRepository.Instance.Save(project);
}
```