4.4 Authorization API (IEPassAuthorization) ........................................ 12
4.4.1 GetRoles (API method) ................................................................. 12
4.4.2 Setting Role-Based Permissions .................................................. 12
4.5 Profile API (IEPassProfile) ............................................................... 13
4.5.1 GetProfile (API Method) .............................................................. 13
4.5.2 SaveProfile (API Method) ............................................................ 13
4.6 Events API (IEPassEvents) ............................................................... 14
4.6.1 The PreSignon Event ................................................................. 14
4.6.2 The PreAuthenticate Event ......................................................... 14
4.6.3 The PostAuthenticate Event ...................................................... 15
4.6.4 The PreLogout Event ................................................................. 16
4.6.5 The PostLogout Event ............................................................... 16
4.6.6 The UserTimeout Event .............................................................. 16
4.7 Other API Members ........................................................................ 17
4.7.1 The EPassPrincipal and EPassIdentity objects ............................. 17
4.7.2 The IProfileDebug interface ....................................................... 17
5 LOGGING AND ERROR HANDLING ................................................. 18
5.1 Configuring Logging ....................................................................... 18
5.2 Handling Errors ............................................................................. 19
5.2.1 Error Redirects ........................................................................... 19
6 HOW DO I ............................................................................................ 20
6.1 Certificates ..................................................................................... 20
6.1.1 ...Generate a Signing Cert? .......................................................... 20
6.1.2 ...Install a Certificate into the Cert Store? ................................... 21
6.1.3 ...Set Certificate Permissions? ................................................ 22
6.2 Configuration ................................................................................. 23
6.2.1 ...Get Intellisense for a Custom Configuration Section? ............. 23
6.3 Authorization .................................................................................. 24
6.3.1 ...Restrict Access To My Page In ASP.NET WebForms? ............... 24
6.3.2 ...Restrict Access To My Page In ASP.NET MVC? ......................... 24
6.3.3 ...Restrict Access to a Feature or Programatically Control Access? .... 25
7 TROUBLESHOOTING ........................................................................ 26
7.1 Configuration Errors ....................................................................... 26
7.2 Certificate Errors ........................................................................... 27
7.3 API Errors ....................................................................................... 28
7.4 ePass Montana Errors ..................................................................... 29
7.4.1 Analyzing Log Information ........................................................ 30
7.5 Submitting a Request to the Help Desk .......................................... 30
8 USE CASES ......................................................................................... 31
8.1 Basic Implementations ................................................................... 31
8.1.1 License Application Site ............................................................. 31
8.1.2 Government Employee Classifieds ............................................. 31
8.2 API Implementations.................................................................................................................. 32
8.2.1 Water Bill ePayment Site ........................................................................................................ 32
8.2.2 Job Listings Site ....................................................................................................................... 32
8.2.3 Online Store ............................................................................................................................ 33
8.2.4 ePass Montana Retrofit .......................................................................................................... 33
1 INTRODUCTION

1.1 Overview

ePass Montana is the State of Montana’s single sign on authentication solution for Montana.gov eGovernment services. ePass Montana uses federated identity management technology to allow all agency eGovernment services the ability to be integrated. The primary technology behind ePass Montana is SAML 2.0 (Security Assertion Markup Language).

The ePass Montana Service Provider Module offers a SAML 2.0 service provider module for the .NET platform tailored specifically to ePass Montana.

1.2 Purpose

The purpose of the ePass Montana Service Provider Module is to simplify ePass Montana integration for our state partners that work in the .NET platform. It offers native SAML support to your .NET application. There is no need for any middleware with this solution.

Our goal in providing this module was to make the integration fast and intuitive. Everything required to connect your application to ePass Montana is covered in the Quick Start guide. We do however understand that some of our partners may have unique requirements that may not conform to the base integration. For that the module offers an advanced configurations and extensibility points.

1.3 Running the Demos

The ePass Developer Toolkit includes two demo projects located in the Demos directory of the Toolkit, a MVC application (EPassMtSPMvcDemo) and a Web Forms application (EPassMtSPFormsDemo). Both target the .NET 4.5 framework but the module is backward-compatible to the 3.5 framework.

To run the demo applications you must first install the following certificates to the Local Machine \ Personal (My) store and assign read permission to the private key for the Application Pool account or Everyone.

<table>
<thead>
<tr>
<th>MVC Demo Certificate</th>
<th>Cert Location: [Toolkit Location]\Runtime\ePassMtSP_MVCDemo_Test_Cert.pfx</th>
<th>Cert Password: MVCDem0C3rtT3st</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebForms Demo Certificate</td>
<td>Cert Location: [Toolkit Location]\Runtime\ePassMtSP_FormsDemo_Test_Cert.pfx</td>
<td>Cert Password: FormsDem0C3rtT3st</td>
</tr>
</tbody>
</table>

For instructions on installing the certificate see Chapter 6.1.2. For instructions on assigning certificate permissions see Chapter 6.1.3.

You can now open and run the Visual Studio solution. The only other requirement is the demo applications are run under localhost:

http://localhost/EPassMtSPMvcDemo
http://localhost/EPassMtSPFormsDemo
1.4 The ePass Montana Service Provider Integrator Support Lifecycle
Once you have evaluated the demo application and have decided to move forward with integration, you must follow the steps contained in the lifecycle shown below.

Process Steps

2. Agency completes ePass Montana Work Order and is reviewed and signed by DOA and MI.
3. Integrator or Agency submits service provider request (SPR) form on the ePass Montana developer toolkit site.
4. SPR form is received by MI Helpdesk and a helpdesk ticket is created.
5. MI Helpdesk reviews the request within 1-3 business days to insure accuracy.
6. If the SPR form is not accurate or incomplete, MI Helpdesk contacts the requestor asking them to re-submit a complete form. The 3-5 business day window is reset.
7. MI Helpdesk adds the agency contact to the ePass Montana contact list.
8. MI Helpdesk configures the service in ePass Montana Administration as a Service Provider (SP).
9. MI Helpdesk notifies customer and asks the customer to verify the service is operational.
10. Requester tests the confirmation and corresponds that the service is operational with ePass Montana where a user of the service can log in and out of the service via ePass Montana.
11. MI Helpdesk closes the service provider request ticket.
2  ADDING EPASS MONTANA AUTHENTICATION TO YOUR APPLICATION

This chapter covers the minimum requirements for integrating ePass Montana authentication into your custom application. Subsequent chapters cover additional settings and integration points that can be used to tailor the integration to suit your needs. The steps outlined in this chapter must be performed by all integrations and once completed your application will have the ability to sign-in and logout using ePass Montana.

2.1  Quick Start Guide

Submit a Work Order

Complete the Work Order Request Form online. This will generate a work order that will be sent to you for signature and submission after it has been approved by SITSD.

Work Order Request Form:  https://app.mt.gov/toolkit/ServiceProvider/WorkOrderRequest

Generate and Install Your Certificate

ePass requires two SSL signing certificates, one for test and one for production. We recommend using the ePass Developer Toolkit – Generate Certificate tool located here:  https://app.mt.gov/toolkit/Integrate/GenerateCertificate

Subject (CN):  CN=[EntityID]/ePass  
Key Size:  2048

* By default the module looks for this subject. If you use a different name see Chapter 3.2.1.

Once you have generated a signing cert it should be installed into the Microsoft Local Machine / Personal (My) Certificate store and read permission must be given to the Application Pool user.

For information on configuring the module to look in a different store see Chapters 3.2.2-3.
For instructions on installing the certificate see Chapter 6.1.2.
For instructions on assigning certificate permissions see Chapter 6.1.3.

Reference the Module

Add a reference to the ePass Montana SP Module assembly (ePassMtSP.dll) in your project. The dll can be found here:  <ToolkitLocation>\Runtime\ ePassMtSP.dll
Configure the Module

Add these settings to the web.config

```xml
<configuration>
  <configSections>
  </configSections>
  ...
    RootURL="[https://www.yourserver.com/yourapp]">
  ...
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true">
      <add name="ePassMtSPModule" type="MT.Gov.EPass.SPModule.SsoModule" />
    </modules>
  </system.webServer>
</configuration>
```

For information on getting Intellisense for the EPassConfig element see Chapter 6.2.1. For details about configuring the EPassConfig element see Chapter 3.3.

If you are using ASP.NET MVC you will need to configure a route exception to allow the built-in controller to handle ePass requests. Add the following as the first route in your App_Start\RouteConfig.cs class:

```csharp
routes.Ignore("{*allEpass}", new { allEpass = @".*\.(epass|/\.*|.\*)?" });
```

Complete the Service Provider Request Form

To register your service provider with ePass Montana, the integrator must submit the Service Provider Request Form to MI.

**Online Form:** [https://app.mt.gov/toolkit/Integrate/ServiceProviderRequest](https://app.mt.gov/toolkit/Integrate/ServiceProviderRequest)

**NOTE:** You can generate the metadata by navigating to ~/Metadata.epass in your application.

Once MI has processed your form you will be successfully integrated! You will be notified via email that your service provider has been configured and is ready for use. To perform login or logout actions simply link to these module-provided endpoints:

- **Login URL:** ~/Login.epass
- **Logout URL:** ~/Logout.epass

2.2 UI Requirements

The consuming application must use the approved login image as the link to ~/Login.epass. The login image is available for download here: [https://app.mt.gov/toolkit/Home/InterfaceRequirements](https://app.mt.gov/toolkit/Home/InterfaceRequirements)

The consuming application must contain a "Logout" link (~/Logout.epass) and an "ePass Montana Home" link ([https://epass.mt.gov](https://epass.mt.gov)) placed in the upper-right corner of the layout or master page (all pages in the application). These links should only be displayed when the user is logged in.
3 ADVANCED CONFIGURATION

The ePass Montana Service Provider Module only requires the configuration entries shown in the Quick Start. This chapter defines all configuration options offered by the module that can be used to customize the module to suite your particular needs.

3.1 Registering the Configuration Section

As mentioned in the Quick-Start chapter the module’s configuration section must be registered in the web.config before the module can be configured or even used. To register the custom configuration section add the following configuration element to your web.config.

```xml
<configuration>
  <configSections>
  </configSections>
</configuration>
```

3.2 Adding Intellisense for the Configuration Section

The EPassMtSP Module provides an xml schema that allows Visual Studio to provide Intellisense in the web.config editor. To enable this feature follow the instruction in Chapter 6.2.1 How Do I… Get Intellisense for a Custom Configuration Section? of this document.
3.3 The EPassConfig Element

The EPassConfig element is the root element of the module’s configuration section. It must be a child of the <configuration> element and can occur anywhere after the <configSections> element. All configuration settings that are required by the module are contained in this element and all attributes of this element are required.

```
<configuration>
  <configSections />
               RootURL="https://agncy.mt.gov/app" />
</configuration>
```

3.3.1 EntityID Attribute

The EntityID attribute identifies your service to ePass Montana. EntityID’s are generally take the form of a URN (a URL that is not necessarily resolvable in a web browser, just follows the format). We recommend following the naming pattern https://env.agncy.mt.gov/app where:

- `env` = local, dev, test, or prod (demo, uat, any environment descriptor desired)
- `agency` = your agency’s abbreviation
- `app` = your applications name or abbreviation

**NOTE:** The naming convention listed above is just a recommendation, use what works for your application (add additional path information, omit the prod designator for the production EntityID...).

**IMPORTANT:** The EntityID must match the Entity ID you entered in the Service Provider Request form exactly (case sensitive) and must be unique to the ePass Montana system.

3.3.2 Environment Attribute

Specifies which ePass Montana environment to use for authentication. Values are limited to “Test” or “Prod”.

3.3.3 RootURL Attribute

The URL path to your application. This should include “https://” but not a page name. The RootURL is used by the module to determine the SSO endpoints and is the default for all other endpoint settings that are not specified in the <Endpoints> element. We recommend you set a default page in IIS.

3.4 The Endpoints Element

The Endpoints element is where you can specify all of the target URLs used by the module. This chapter will define those targets and what the default is if the attribute is not set. The Endpoints element is a child of the EPassConfig element. All attributes of the Endpoints element are optional. All URLs specified in this element can be absolute URLs or application-relative URLs (starting with “~/”).
3.4.1 DefaultPage Attribute
The Default Page is the default URL for all of the other endpoints used by the module. It defaults to EPassConfig.RootURL.

3.4.2 LoginPage Attribute
The LoginPage endpoint specifies the page in your application that contains the Login Control. This does not need to be a dedicated login page, in many cases it is the same as the DefaultPage (home page). The LoginPage endpoint defaults to the value of the DefaultPage endpoint.

3.4.3 LoginErrorPage Attribute
The LoginErrorPage endpoint specifies the page in your application that the user will be sent to in the case of an error in the login process. This does not need to be a dedicated page as the URL will be enhanced with a Querystring of ?ErrorCode=LoginError&ErrorMessage=Message. The LoginErrorPage endpoint defaults to the value of the LoginPage endpoint.

3.4.4 PostLoginRedirect Attribute
The PostLoginRedirect endpoint specifies the page in your application that the user will be sent to after a successful login. The PostLoginRedirect endpoint defaults to the value of the DefaultPage endpoint.

3.4.5 PostLogoutRedirect Attribute
The PostLogoutRedirect endpoint specifies the page in your application that the user will be sent to after a successful logout. The PostLogoutRedirect endpoint defaults to the value of the DefaultPage endpoint.

3.4.6 UnauthorizedRedirect Attribute
The UnauthorizedRedirect endpoint specifies the page in your application that the user will be sent to if they try to access a resource they are not authorized to view. This does not need to be a dedicated page as the URL will be enhanced with a Querystring of ?ErrorCode=LogoutError&ErrorMessage=Message. The UnauthorizedRedirect endpoint defaults to the value of the LoginPage endpoint.

3.4.7 RegistrationPage Attribute
The RegistrationPage endpoint is only used if EPassConfig.Options.RequireFederation = "true". Users will be sent to this page if the API call to IEPassFederation.Federate yields a principal where principal.IsFederated = false. This URL should point to the first (or only) page of the consuming application’s registration process.

**IMPORTANT:** Once the registration page has finished processing the user should be redirected to ~/CompleteRegistration.epass so the remaining integration points will be executed (GetRoles, GetProfile, PostLoginEvent...)
3.5 The Options Element
The Options element contains many of the module's advanced settings. The Options element is a child of the EPassConfig element. All attributes are optional and are defined in this chapter.

```xml
<configuration>
  <configSections />
  <EPassConfig>
    <Options NameIdFormat="persistent|emailAddress|unspecified" PrimaryAttributeSource="ePass|StateAD|CDB"
      Timeout="30" APIAssembly="ApplicationAssemblyName" APIClass="Fully.Qualified.Class.Name"
      RequireFederation="true|false" />
  </EPassConfig>
</configuration>
```

3.5.1 NameIdFormat Attribute
The NameIdFormat attribute specifies the format the user's identity to be sent from ePass. This setting must match the value entered in the Service Provider Request form. Valid settings are:

- urn:oasis:names:tc:SAML:2.0:nameid-format:persistent [default and recommended]
  Users will be given a generated identifier that is unique to the user and application.
- urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
  The email address the user registered for ePass Montana with will be used as the identifier.
- urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
  The identifier will be composed of the desired user attributes as specified in the ePass Montana setup.

3.5.2 PrimaryAttributeSource Attribute
The PrimaryAttributeSource attribute specifies which source to use for user attributes if a given attribute is provided by multiple Identity Providers. For example if you choose to have ePass Montana + CDB (Registered User) configured as an Identity provider in ePass Montana, both systems will provide a UserName attribute. This setting will specify which UserName is loaded to


NOTE: For more information about Registered Users see [https://app.mt.gov/registered](https://app.mt.gov/registered)

3.5.3 Timeout Attribute
The Timeout attribute allows the module consumer to set the user’s session timeout. The timeout is specified in minutes and is defaulted to 30.

3.5.4 APIClass and APIAssembly Attributes
If you wish to use the EPassMtSP Module's API you will need to configure these attributes. The class specified by these attributes must be thread-safe and implement one or many of the API interfaces. The APIAssembly will be the name of the assembly that contains the API Implementation class (likely your application’s main assembly) and APIClass will be the fully-qualified name of the implementing class.

See [Chapter 4.1 Using the API](#) for more information.

3.5.5 Set Thread Principal
The SetThreadPrincipal flag (true|false) tells the module to set the thread principal to the current user principal [System.Threading.Thread.CurrentPrincipal].
3.5.6 RequireFederation Attribute
The RequireFederation attribute is a boolean ("true" or "false") flag that tells the module if your application requires users to be federated (Linked to information on your systems). If true, and the IEPassFederation.Federate (Chapter 4.2.1 Federate (API method)) method is unable to federate, the user is redirected to Endpoints.RegistrationURL if configured or Endpoints.UnauthorizedRedirect if not.

3.6 The Signing Certificate Element
The SigningCertificate element should be used if you do not use the standard subject and store for your SAML signing certificate. By default the signing certificate subject should be CN=[EntityID].Cert and should be located in the [Local Computer].Personal(My) certificate store. All nonstandard values must be set using the attributes of this element.

```xml
<configuration>
  <configSections />
  <EPassConfig>
    <SigningCert Location="LocalMachine|CurrentUser " Store="AddressBook|AuthRoot|CertificateAuthority|Disallowed|My|Root|TrustedPeople|TrustedPublisher" Subject="CN=[EntityID].Cert" />
  </EPassConfig>
</configuration>
```

3.6.1 Certificate Subject
The CertificateSubject attribute allows the module consumer to specify a nonstandard subject for the SAML signing certificate. The default subject is CN=[EntityID]/ePass where EntityID is the value from EPassConfig.EntityID.

**NOTE:** "/ePass" is appended to the entity ID to provide clarity of the certificate’s use and in case the EntityID matches the application URL. The application URL cert subject should be reserved for the SSL certificate.

3.6.2 Certificate Location
The CertificateLocation attribute allows the module consumer to specify a nonstandard certificate store location for the SAML signing certificate. Valid locations are CurrentUser or LocalMachine and LocalMachine is the default value.

3.6.3 Certificate Store
The CertificateStore attribute allows the module consumer to specify a nonstandard certificate store for the SAML signing certificate. The default is “My”, the Personal certificate store. Valid stores are listed below:

<table>
<thead>
<tr>
<th>AddressBook</th>
<th>My [Default]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthRoot</td>
<td>Root</td>
</tr>
<tr>
<td>CertificateAuthority</td>
<td>TrustedPeople</td>
</tr>
<tr>
<td>Disallowed</td>
<td>TrustedPublisher</td>
</tr>
</tbody>
</table>

Web.config

```xml
<configuration>
  <configSections />
  <EPassConfig>
    <SigningCert Location="LocalMachine|CurrentUser " Store="AddressBook|AuthRoot|CertificateAuthority|Disallowed|My|Root|TrustedPeople|TrustedPublisher" Subject="CN=[EntityID].Cert" />
  </EPassConfig>
</configuration>
```
4 EXTENSIBILITY (API)

The ePass Montana Service Provider Module offers several extensibility points in the form of an API to support more complex integrations. These hooks can be used to integrate authorization and profile providers, federate to existing user/role stores, or to just perform custom operations during the login process.

4.1 Using the API

To use the API you must create a thread-safe class that implements one, many, or all of the interfaces defined in this chapter (API Interfaces are contained in the ePassMtSP.dll and are under the MT.Gov.EPass.SPModule.API namespace). The easiest way to make your class thread-safe in this situation is to not have any members (class-level variables/properties), it is best to have a dedicated class that only implements the interface method(s). Once you have created an API Implementation class you must configure the module to use it. See Chapter 3.5.4 for information on configuring the API.

4.2 Order of Events in the API

Below is a listing of the order of events for each of the main EPassMtSP module processes that are exposed through the API. API calls are highlighted in bold, other key steps in the process are listed to offer a better picture of the overall process.

<table>
<thead>
<tr>
<th>Login Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User is directed to /Login.epass</td>
</tr>
<tr>
<td>2. PreSignon Event</td>
</tr>
<tr>
<td>3. ePass Montana Single Sign-On</td>
</tr>
<tr>
<td>4. PreAuthenticate Event</td>
</tr>
<tr>
<td>5. SAML assertion is processed</td>
</tr>
<tr>
<td>6. Federate API Call</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>7. GetRoles API Call</td>
</tr>
<tr>
<td>8. GetProfile API Call</td>
</tr>
<tr>
<td>9. PostAuthenticate Event</td>
</tr>
<tr>
<td>10. User Principal is assigned to HttpContext.User</td>
</tr>
<tr>
<td>11. User is redirected to the PostLoginRedirect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logout Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User is directed to / Logout.epass</td>
</tr>
<tr>
<td>2. PreLogout Event</td>
</tr>
<tr>
<td>3. ePass Montana Single Logout</td>
</tr>
<tr>
<td>4. SaveProfile API Call</td>
</tr>
<tr>
<td>5. User Principal is destroyed</td>
</tr>
<tr>
<td>6. PostLogout Event</td>
</tr>
<tr>
<td>7. User is redirected to PostLogoutRedirect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User is inactive for the timeout duration</td>
</tr>
<tr>
<td>2. SaveProfile API Call</td>
</tr>
<tr>
<td>3. User Principal is destroyed</td>
</tr>
<tr>
<td>4. UserTimeout Event</td>
</tr>
<tr>
<td>5. Next time the user refreshes their browser they will be sent to the login page with a RedirectURL of the page they are trying to access.</td>
</tr>
</tbody>
</table>
4.3 Federation API (IEPassFederation)

The Federation API should be used if you have the need to associate an ePass Montana user to a legacy user account or to some identity (data key) in your system. Implementing the IEPassFederation interface allows the module consumer to federate (associate) the ePass Montana user to a local user or identity at the time of login. The federation record (presumably a database table or some other data store) that maps the ePassID to the local identity is maintained by the module consumer and accessed through this API.

For first-time users the federation record must be established. Since ePass Montana does not establish who a user actually is (only an online identity), a registration process is generally required to establish the user’s identity as it relates to the consuming system. Adding a registration page to the login flow is supported by the ePassMtSP Module and is covered in this chapter.

Note: See Chapter 8 Use Cases for more information about federation and what it is used for.

4.3.1 Federate (API method)

The IEPassFederation interface defines a single method: Federate(EPassPrincipal principal). The objective of implementing this method is to find the user in the consumer-managed federation table and enhance the principal object with the local system identifier. This will generally involve a query to the local data store using the user ID (principal.UserId) as a key. If federation is unsuccessful (the user is not found) and the RequireFederation (chapter 3.5.5) flag is set to true, the user will be sent to the RegistrationPage (chapter 3.4.7) URL. Below are the minimum code requirements for proper implementation:

```c#
void Federate(EPassPrincipal principal)
{
    // Look up user in local data store
    // Successful Federation:
    principal.FedId = "LocalUserIdentifier";
    principal.IsFederated = true;
    // User Not Found:
    principal.IsFederated = false;
}
```

4.3.2 The Registration Page

The registration page is used for federating unknown but authenticated users (first-time users). It should gather enough information about the user to identify them on your system.

Users are redirected to the registration page if the RequireFederation (chapter 3.5.5) flag is set to true and the user is not federated (User.IsFederated=false). Users will continue to be redirected to this page as long as they remain unfederated. This means the page can only post to itself and it must be a single page form (or the first page will need to set IsFederated=true before continuing).

When the registration process is complete the federation should be saved to the local data store (mapping the ePass Montana userID to the local identifier), the user principal object should be enhanced with a FedID (local identifier) and the IsFederated flag should be set to true. The user should be redirected to ~/CompleteRegistration.epass so the remaining integration points will be executed (GetRoles, GetProfile, PostLoginEvent...).
4.4 Authorization API (IEPassAuthorization)
The Authorization API is responsible for assigning roles to users which allows the consuming application to offer different levels of access to different users.

The consuming application will need to have some kind of role storage and management system. This could be anything from a user permissions database backed by a custom administrative web application to Active Directory server or ADAM (Active Directory Application Mode) instance.

4.4.1 GetRoles (API method)
The IEPassAuthorization interface defines a single method GetRoles. The objective of implementing this method is to find the user in the consumer-managed user-roles table and return a list of roles that are assigned to the user. This will generally involve a query to the local data store using the user ID (principal.UserID) or federated ID (principal.FedId) as a key.

```csharp
public List<string> GetRoles(EPassPrincipal principal)
{
    string localID = principal.FedId;
    List<string> roles;

    // Look up user in local data store and get the associated roles
    return roles;
}
```

4.4.2 Setting Role-Based Permissions
Implementing the Authorization API alone will not protect your web application pages. You must tell ASP what user restrictions to apply to your pages. ASP.NET, both MVC and WebForms, offer in-built mechanisms for authorizing users based on roles. See Chapter 6.3 How Do I... Authorization to learn more about configuring these permissions.
4.5 Profile API (IExchangeProfile)

The Profile API offers extensibility points for loading and saving user profile data. Profile Information would generally consist of user settings/preferences and additional identifying information, it can be used to store any information you want your users to always have.

Profile information is loaded as an Object type so the module consumer can choose any type they wish, a custom class object or generic collection perhaps. Once profile data has been loaded it can be accessed through the `IEPassPrincipal.User`.Profile property.

To use the Profile API implement the IExchangeProfile interface in the API implementation class configured at the beginning of this chapter.

4.5.1 GetProfile (API Method)

The GetProfile method is responsible for loading the profile information from the data store and returning it to be loaded to the User principal object. The current principal object is passed in as a parameter so the implementer can identify the user.

```csharp
public object GetProfile(IExchangePrincipal principal) {
    object profile;
    string localID = principal.FedId;
    // Use the localId to load the profile data from the data store
    return profile;
}
```

4.5.2 SaveProfile (API Method)

The SaveProfile method is responsible for saving the profile information to the data store. It is called before the user is logged out of the system. The current principal object is passed in as a parameter for the implementer to identify the user and extract the profile data.

```csharp
public void SaveProfile(IExchangePrincipal principal) {
    string localID = principal.FedId;
    object profile = principal.Profile;
    // Save the profile object to the datastore
    return;
}
```
4.6 Events API (IEPassEvents)

The Events API offers extensibility points for handling SSO related events thrown by the module. These events can be used to execute custom code and in some cases to effect the processing of the event.

To use the Events API implement the IEPassEvents interface in the API implementation class configured at the beginning of this chapter. For events you are not interested just put a single return statement in the method body.

4.6.1 The PreSignon Event

Implementing this method allows the consuming application to execute custom code before the user is redirected to ePass Montana for login. The consumer of this event is allowed to cancel the signon action and redirect the user if desired.

The method has a single parameter of type PreSignonEventArgs, below is a list of the members of that object and the uses for each.

- **Context** - current HTTP context, information only
- **Cancel** - If set to true, Sign-on will be aborted and the user will be sent to RedirectURL
- **RedirectURL** - Specifies the URL to redirect the user to in the event of the sign-on being canceled. The default URL is EPassConfig.Endpoints.LoginPage.
- **Message** - Setting this property will attach the specified message to the URL as a query string parameter: ?ErrorCode=Message&ErrorMessage=[this property value]

Example: Canceling the sign-on action

```csharp
public void PreSignon(PreSignonEventArgs eventArgs)
{
    bool abortCondition = true;
    if (abortCondition)
    {
        eventArgs.Cancel = true;
        eventArgs.Message = "You are not allowed to login on this site";
        // This will redirect the user to the LoginPage or DefaultPage
        // For a specific page set the eventArgs.RedirectURL property
    }
    return;
}
```

4.6.2 The PreAuthenticate Event

Implementing this method allows the consuming application to execute custom code before the assertion is processed. It also allows the consumer to override the UserId, RedirectURL, Attributes or cancel the login action and redirect the user. Modifying the values provided by this event is not recommend and could cause unexpected results. The ability to do so is offered primarily to cover any unforeseen need of the module consumer, use at your own risk.

The method has a single parameter of type PreAuthenticateEventArgs, below is a list of the members of that object and the uses for each.

- **UserId** – User ID provided by the ePass Montana assertion
SamlAttributes – Dictionary (Key/Value pair) object containing the SAML attributes provided by the ePass Montana assertion

Cancel - If set to true, authentication will be aborted and the user will be sent to RedirectURL

RedirectURL - Specifies the URL to redirect the user to in the event of the authentication being canceled. The default URL is EPassConfig.Endpoints.LoginPage. If the authentication is NOT canceled, this attribute can be used to override the URL the user will be sent to after a successful login (the user would normally be redirected to EPassConfig.Endpoints.PostLoginRedirect or the SAML RelayState, the page the user was trying to access before login)

Message - Setting this property will attach the specified message to the URL as a query string parameter: ?ErrorCode=Message&ErrorMessage=[this property value]

4.6.3 The PostAuthenticate Event

Implementing this method allows the consuming application to execute custom code after the assertion is processed. It also allows the consumer modify the principal (User) object directly after it has been created.

Warning: This method may be called multiple times during a user session. It is called whenever the credential is not found in cache (user changes servers, service is restarted...) and must be regenerated from the encrypted cookie (and, of course, after initial login)

The method has a single parameter, the EPassPrincipal (User) object of the newly authenticated user. The principal object can be read to facilitate custom code or can be modified if desired. Use of the Federation API, Authentication API, and/or the Profile API should be used to update the principal object when possible. Edit functionality is provided to fulfill requirements that cannot be handled by the other APIs.
4.6.4 The PreLogout Event
Implementing this method allows the consuming application to execute custom code before the user is logged out. The method has a single parameter of the EPassPrincipal (User) object. It is provided only for reference, it should not be changed inside this event.

```csharp
public void PreLogout(EPassPrincipal principal)
{
    HttpContext context = HttpContext.Current;
    DAO.SavePartialApplication(context.Session["CurrentApp"], principal.UserId);
    return;
}
```

4.6.5 The PostLogout Event
Implementing this method allows the consuming application to execute custom code after the user is logged out.

The method has a single parameter of type PostLogoutEventArgs, below is a list of the members of that object and the uses for each.

- **Context** – the current HttpContext
- **RedirectURL** - Specifies the URL to redirect the user to (the user would normally be redirected to EPassConfig.Endpoints.PostLogoutRedirect)

```csharp
public void PostLogout(PostLogoutEventArgs eventArgs)
{
    bool redirectCondition = true;
    if (redirectCondition)
    {
        eventArgs.RedirectURL = "~/newTargetPage";
    }
    return;
}
```

4.6.6 The UserTimeout Event
Implementing this method allows the consuming application to execute custom code when a user times out and their session expires. There is no active request so no user interaction is possible and there is no current HttpContext or session. This method would likely have the same logic as the PreLogout event, the challenge is the lack of a user request to work with.

```csharp
public void UserTimeout(EPassPrincipal principal)
{
    Dictionary<string, object> profile = (Dictionary<string, object>)principal.Profile;
    DAO.SaveReturnToPage(principal.UserId, profile["ApplicationId"], profile["LastPageNumber"]) ;
    return;
}
```
4.7 Other API Members

4.7.1 The EPassPrincipal and EPassIdentity objects
The EPassPrincipal object implements IPrincipal, the base user unit in .NET. All interface methods and properties of that interface are supported. The user principal object can be accessed anywhere in MVC or WebForms context using the User keyword. To get the full benefits of the information provided by the EPassPrincipal object the User object should be cast to this type. The EPassPrincipal object uses the EPassIdentity for its Identity property, there is no reason to cast this object, it is just a base implementation of IIdentity.

Class Members

- **Attributes** – Combined attributes from all IdP sources (The first attributes imported are from EPassConfig.Options.PrimaryAttributeSource, these values are not replaced unless they are null)
- **CDBAttributes** – Attributes obtained only from the CDB (Registered User) IdP
- **EPassAttributes** – Attributes obtained only from the ePass Montana IdP
- **FedId** – Federation ID or Local ID, this property is consumer-controlled and should be added as part of the Federate() API call.
- **Identity** – The currently logged in user’s Identity of type EPassIdentity
- **IsFederated** – Indicates the federation status, set in the Federate() API call or by the registration page
- **Profile** – Object container to be used by the consumer to store a user data object of their choice.
- **Roles** – List of role name the user belongs to. Set by the GetRoles() API call.
- **SessionID** – Identifies the user’s session, used by the module directly to perform logout operations
- **StateAttributes** – Attributes obtained only from the State of Montana IdP
- **UserId** – Helper property that returns User.Identity.Name. This is generally set to the ePass Montana persistent ID.

4.7.2 The IProfileDebug interface
Implementing the IProfileDebug interface on your profile object (the object returned from the GetProfile() API call) allows the profile information to be shown on the ~/User.epass page. There is no advantage to this except for debugging purposes.
5 LOGGING AND ERROR HANDLING

5.1 Configuring Logging

The EPassMtSp module implements the common logging interface. Common.Logging is available as a nuget package here: http://www.nuget.org/packages/Common.Logging. You will also need to add the logging factory of your choice to the project, there are many available including Log4Net and NLog. To enable logging follow the directions outlined in the Common.Logging documentation found here: http://netcommon.sourceforge.net/docs/2.0.0/reference/html/ch01.html

Example: Configuring Common.Logging.Log4Net [web.config]

```xml
<configuration>
  <configSections>
    <sectionGroup name="common">
      <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging"/>
    </sectionGroup>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"/>
  </configSections>
  ...
  <common>
    <logging>
        <arg key="configType" value="INLINE"/>
      </factoryAdapter>
    </logging>
  </common>
  <log4net>
    <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %logger [%property{NDC}] - %message%nnewline"/>
      </layout>
    </appender>
    <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">
      <file value="logs/application.log"/>
      <appendToFile value="true"/>
      <rollingStyle value="Date"/>
      <maxSizeRollBackups value="7"/>
      <staticLogFileName value="true"/>
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %logger [%property{NDC}] - %message%nnewline"/>
      </layout>
    </appender>
    <appender name="ePassMtSpAppender" type="log4net.Appender.RollingFileAppender">
      <file value="logs/ePassMtSP.log"/>
      <appendToFile value="true"/>
      <rollingStyle value="Date"/>
      <maxSizeRollBackups value="7"/>
      <staticLogFileName value="true"/>
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %logger [%property{NDC}] - %message%nnewline"/>
      </layout>
    </appender>
  </log4net>
  ...
</configuration>
```
5.2 Handling Errors

In most cases, error in the EPassMtSP Module are trapped, logged, and the user redirected to the appropriate page with a query string value indicating the error.

5.2.1 Error Redirects

When an error occurs the user will be redirected to the appropriate page for notification and correction. The target pages for these errors are determined by the error type. The following grid shows the error type and target page.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Target URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntegrationError</td>
<td>Requested URL</td>
</tr>
<tr>
<td>LoginError</td>
<td>EPassConfig.Endpoints.LoginErrorPage</td>
</tr>
<tr>
<td>Message</td>
<td>Requested URL</td>
</tr>
</tbody>
</table>

Requested URL = The user will not be redirected, the original URL will be enhanced with the error query string for display by that page.

In all cases query string parameters will be added to the URL in the form of:

?ErrorCode=[ErrorType]&ErrorMessage=[DefaultMessage]

The module will pass a default error message in the ErrorMessage parameter. It is however recommended that the consuming application display their own message based on the ErrorCode to better suit their needs.

To see the actual error and stack trace the consuming application will need to implement Logging (covered previously in this chapter).

5.2.1.1 Error Types

The list of possible error types are defined by the ErrorTypeEnum that is located in the MT.Gov.EPass.SPModule.API namespace. Here is a list of the enumerated values with a short description of each.

- **ConfigError** - Error with the configuration of the module
- **IntegrationError** - Error referencing or calling the API implementation class
- **LoginError** - Error processing user login
- **LogoutError** - Error processing user logout
- **FederationError** - Error federating ePass Montana user to local user
- **Unauthorized** - The user is trying to access a resource they do not have permission to
- **Message** - General alert from the module
HOW DO I...

This chapter offers additional instructions for common tasks such as generating and installing a certificate file and working with the web.config.

6.1 Certificates

6.1.1 Generate a Signing Cert?

The ePass Montana Toolkit site offers a certificate generation tool found here: https://app.mt.gov/toolkit/Integrate/GenerateCertificate

Use this form to generate and download a signing certificate for use with the ePass Montana Service Provider Module. Make sure you do not lose your password or certificate as they are not maintained by the site and cannot be recovered. Don’t forget to install and set permissions on your new certificate as outlined in the remainder of this chapter.
6.1.2 Install a Certificate into the Cert Store?

Start the Microsoft Management Console:

- Start ➔ type “mmc” in the search box ➔ click mmc.exe
- OR Start ➔ run ➔ mmc
- OR c:\Windows\System32\mmc.exe

Add the Certificates Snap-in:

- File ➔ Add/Remove Snap-in...
  - Choose Certificates on the left ➔ [Add >]
  - Select the “Computer account” radio ➔ [Next]
  - Select the “Local computer” radio ➔ [Finish]
  - Press [Ok] to close the Snap-in Manager

Navigate to the Personal Certificate Store:

Certificates (Local Computer) ➔ Personal ➔ Certificates

Import the Certificate:

- Action ➔ All Tasks ➔ Import...
  - [Next]
  - [Browse...]
  - Change the file type filter to include .pfx or .*
  - Select the certificate ➔ [Open]
  - [Next]
  - Enter the password ➔ check Include all extended properties ➔ [Next]
  - Select the “Place all certificates in the following store” radio
    - The Certificate store should be Personal (if not click [Browse...] and pick it)
  - [Next]
  - [Finish]

Your certificate should now appear in the folder Certificates (Local Computer)\Personal\Certificates.
6.1.3  ...Set Certificate Permissions?
Open Microsoft Management Console and add the Certificate Snap-in (see Chapter 6.1.2).

Navigate to and select the desired certificate.

Open the permissions manager:
   Action OR Right Click ➔ All Tasks ➔ Manage Private Keys...

Add your Application Pool Identity (User) to the list and assign Read permissions.

If you don’t know your App Pool Identity you can find it in IIS Manager under Advanced Settings for the Application Pool. If your app pool lists ApplicationPoolIdentity as the Identity it is a virtual account not found in the user list, you need to select your local machine from the locations popup in the add dialog and enter the user name “IIS AppPool\<AppPoolName>” where <AppPoolName> is the name of the application pool assigned to your service.

Note: If you are running an operating system prior to Vista or Server 2008 this functionality is not provided by the snap-in. You will need to use a command line tool that can be found here: http://msdn.microsoft.com/en-us/library/windows/desktop/aa384088(v=vs.85).aspx
6.2 Configuration

6.2.1 ...Get Intellisense for a Custom Configuration Section?

To enable Intellisense for the EPassConfig element:

1. Open the web.config
2. Right-Click in the white-space and select Properties
3. In the Properties Dialog
   - Click in the text to the right of Schemas
   - Click the [...] button
4. In the XML Schemas Dialog
   - Click the [Add...] button
   - Browse to `<ToolkitLocation>\Runtime\EPassMtSPConfigSchema.xsd`
   - Click Open, the schema should now appear in the list
   - Make sure the Use column is checked for this schema
   - Click [OK]
6.3 Authorization

6.3.1 ...Restrict Access To My Page In ASP.NET WebForms?

Add the sections listed below to your web.config. If you add an authorization section in your main web.config under the system.web element it will be the default for all pages in your application. To control access to a specific region of your site you can put a separate web.config in a child folder to control the files in that folder or use the location element as shown below.

The example at the right denies access to all unauthenticated users by default. Then it allows all users to access the login.aspx page. Finally it restricts access to all pages in the Admin directory to user who are assigned the Admin or Owner role.

For full details see the Microsoft documentation here:

Authorization: [Link]
Location: [Link]
Subdirectory Config: [Link]

6.3.2 ...Restrict Access To My Page In ASP.NET MVC?

For MVC applications authorization is generally implemented using the Authorize attribute. With no parameters the attribute denies access to anonymous users. The attribute accepts a Roles and Users parameter to specify which of either have access. The attribute can be set on the controller class or action method as desired. To give anonymous access to an action whose controller denies it the AllowAnonymous attribute can be used.

For full details see the Microsoft documentation here:

The Authorize Attribute: [Link]
6.3.3  ...Restrict Access to a Feature or Programmatically Control Access?
To apply access logic to a control or feature inside a page in your view, controller, or code behind you can simply use the `User.IsInRole("RoleName")` method. The User object is globally available in all the contexts listed above. It is of type IPrincipal and when using the module is of type EPassPrincipal, either scope will expose this method. The method returns a simple Boolean value indicating if the User belongs to the role.

For full details see the Microsoft documentation here:

7 TROUBLESHOOTING

This chapter common errors that could occur while implementing an ePass Montana integration and how to resolve them. In order to do any effective troubleshooting you must configure logging on your application as outlined in Chapter 5.1 of this document. Without log information it is unlikely you will be able to determine the issue.

7.1 Configuration Errors

Many of the configuration errors cannot be handled by the module because the module itself does not function properly without a proper configuration. Another issue is that the logging for the module depends of proper configuration so there will not be log entries for some of these errors.

The requested page cannot be accessed because the related configuration data for the page is invalid.

In this case your application will not even load. You will receive the server error above directly from IIS.

**LIKELY CAUSE:** Missing the EPassConfig configSection  
OR Name discrepancy between the configSection and the actual EPassConfig element

**RESOLUTION:** See Chapters 3.1-3.3 for proper configuration registration.

PassConfig Configuration Section not found. The consuming application must include a custom configuration section named EPassConfig of type MT.Gov.EPass.SPModule.Config.EPassMtSPConfig

This error is not handled by the module, it is thrown to the consuming application because the configuration is required by the module to handle the error redirects.

**NOTE:** This error is sometimes a child (Inner Exception) of the following error:  

**LIKELY CAUSE:** EPassConfig element is missing from the web.config  
OR EPassConfig element not named “EPassConfig”  
OR EPassConfig element is malformed

**RESOLUTION:** See Chapters 3.1-3.3 for proper configuration registration.  
See Chapter 6.2.1 for setting up Intellisense to ensure proper XML structure
7.2 Certificate Errors
Another possible source of module errors is missing or misconfigured certificates. This chapter outlines common errors associated with signing certificates. Certificate errors are handled by the module, logged, and the user is redirected to:

```
```

If you have set up your default view to display errors, you will see the error in your web browser.

The certificate with a subject of [CN=<Your Cert Subject>] was not found in the LocalMachine\My(Personal) store.

**Likely Cause:**
- The subject name is incorrect
- The signing cert has not been installed
- The cert has been installed to a different store (location)

**Resolution:**
See Chapter 3.6 for configuring the cert name and/or location
See Chapter 6.1.2 for installing the cert

The signing certificate's private key is inaccessible, make sure the application pool account has read permissions to the certificate. Certificate Subject: <Your Cert Subject>, Certificate Store: LocalMachine\My(Personal)

**Likely Cause:**
- The application pool account does not have read permissions to the certificates private key.

**Resolution:**
See Chapter 6.1.3 for assigning certificate permissions

More that one certificate with a subject of [<Your Cert Subject>] was found in the LocalMachine\My(Personal) store. The certificate subject must be unique. Remove the duplicate certificates from the cert store.

**Likely Cause:**
- The Certificate Subject is ambiguous in the designated certificate store.

**Resolution:**
Follow the first part of the directions in Chapter 6.1.2 to set up the certificate MMC add-in, select and delete the duplicate certificate.
7.3 API Errors

Aside from API configuration errors all API errors are caught and logged then normal processing resumes in the module. If this is not the desired behavior the module consumer can catch and handle the errors locally in the implementing method.

The only time normal processing is effected is if EPassConfig.Options.RequireFederation = true and the API call does not result in a federated user. In this case the user is redirected to EPassConfig.Endpoints.RegistrationPage.

API Exceptions can only be found in the logs. It is important to configure logging (see Chapter 5.1) when integrating via the API. If you would like to see the exceptions in the Visual Studio Output (Debug) window and you are using Log4Net you can configure a debug appender (log4net.Appender.DebugAppender). Other logging options may implement this feature as well but are not documented here.

API Integration Error. The API Assembly name cannot be determined, please specify the <EPassConfig><Options APIAssembly="YourAssemblyName"> setting in the web config.

LIKELY CAUSE: The APIAssembly name is missing or incorrect (do not include .dll in the name)
RESOLUTION: See Chapter 3.5.4 for proper configuration.

API Integration Error. Could not create an instance of class [YourAPIClassName] from assembly [YourAssemblyName]. The class does not exist or is inaccessible. It must be scoped as public and implement a default constructor.

LIKELY CAUSE: The APIClass name is incorrect or not fully qualified
OR The APIAssembly name is incorrect (does not contain APIClass)
OR The APIClass does not implement any of the API interfaces
RESOLUTION: See Chapter 4.1 for more information about using the API and Chapters 3.5.4-3.5.5 for proper configuration.


LIKELY CAUSE: The APIClass does not implement any API interfaces.
RESOLUTION: See Chapter 4.1 for information about integrating with the API.
7.4 ePass Montana Errors

ePass Montana does not display actual error messages to the user. The only error message you will receive from ePass Montana is shown here. All this means is that some problem occurred when processing the requested action (login or logout).

As this module is written specifically for ePass Montana and most of the intricacies of the SSO process are built-in, errors are almost always related to module configuration. The first step in troubleshooting ePass Montana errors is to validate that your configuration exactly matches your Service Provider Request Form. Validate the following:

**Entity ID**
The Entity ID must exactly match (case sensitive) between your SP Request Form - Entity ID field and your web.config – EpassConfig.EntityID configuration entry.

**URL**
The URL of your service must match the URL provided in your SP Request Form - Service Provider UAT URL field for test or the Service Provider Prod URL field for Production. ePass Montana will only allow requests from this URL. You can request additional URLs for your local and/or development environments but they must have unique EntityIDs as well.

**NameID Format**
The NameID format must also match between the SP Request Form and the web.config

<table>
<thead>
<tr>
<th>SP Request Form NameID Format</th>
<th>Web.config EPassConfig.Options. NameIdFormat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent Identifier</td>
<td>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</td>
</tr>
<tr>
<td>Email Address</td>
<td>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</td>
</tr>
<tr>
<td>Unspecified</td>
<td>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</td>
</tr>
</tbody>
</table>

**Metadata**
As part of the integration process you should have generated your SAML Metadata by navigating to ~/Metadata.epass (Step 5 in the Quick Start Guide) which means the metadata was generated specifically for your configuration and is correct. If there have been any changes to the following since you generated your metadata it must be regenerated and updated in ePass Montana:

- EntityID
- RootURL
- SigningCertificate
- NameIDFormat
7.4.1 Analyzing Log Information
Once you have validated your configuration, the next step is to look over the logs. If you are not familiar with SAML assertions the contents of the logs may be a bit confusing but have a look and see if there are any errors listed around the time of the attempt. In any case this log information is required to submit a request to the help desk.

Error on Login
For login errors look for “AuthnRequest sent:” in the log, the text following is the actual request sent to ePass Montana. If the request were successful you would see the following entries in the log shortly after the authentication request:

- ...SignOn handler called
- ...SamlResponse decoding
- ...Successfully decoded SamlResponse:

Directly after “Successfully decoded SamlResponse:” is the actual response from ePass Montana. Chances are that if you are reading this section you will not see anything past AuthnRequest in the logs because ePass Montana has not responded. Capture the AuthnRequest and follow the steps in the next section (Submitting a Request to the Help Desk).

Error on Logout
For logout errors look for “Logout handler called”, then “Logout request sent... using "REDIRECT" binding:”. The text following the “:” is the actual logout request sent to ePass Montana. If the request were successful you would see the following entries in the log shortly after the logout request:

- ...Logout handler called (this is ePass Montana calling the handler back as opposed to the initial call above from the app)
- ...Logout response received
- ...Parsing Logout response
- ...Successfully parsed Logout response:
- ...Executing Logout Actions

Directly after “Successfully parsed Logout response:” is the actual response from ePass Montana. Again, some of these entries will probably be absent as you are experiencing ePass Montana logout errors. Capture the Logout request and follow the steps in the next section (Submitting a Request to the Help Desk).

7.5 Submitting a Request to the Help Desk
Once you have validated your configuration and have confirmed the error still exists you may contact the MI Help Desk at helpdesk@egovmt.com. Please mark the email with a subject of ePass Montana Integration Error in .NET and include the following details:

- Agency Name, Reporter Name and contact information
- Your EntityID
- URL of the application having the issue
- ePass Montana Environment you are trying to connect to (Test, Production)
- Description of the error, including what functionality you are having issue with (Login, Logout, API...)
- Any specific error information obtained (message, stack trace)
- Time the error occurred (helps us find the appropriate server logs)

To better assist you we also request that you attach your web.config and log file for analysis by our technical staff.

Note: The MI Help Desk will respond to your request within 3-5 business days.
8 USE CASES
This chapter illustrates several common use cases of an ePass Montana integration. These use cases are not all-inclusive and it is unlikely that all of a module consumers requirements would be met by a single use case. The purpose of these examples is to show how the configuration and extensibility of the module could be applied to solve some common business needs.

8.1 Basic Implementations
Three things are true of the use cases in this section. First the true Identity (who they actually are) of the user is not important, only that the user maintains some identity (who they say they are) between visits. Second, the application does not require role-based security, all users have the same level of access. And third, the application does not require any additional profile information (Address, Date of Birth, Language...).

8.1.1 License Application Site
The consuming application is a simple online automation of a paper process for applying for a license. The application is required to let users save their application and resume it later and to offer updates to the user after submission as to the status of their application.

As the user will be returning to the site the ePassMtSP Module will be used to authenticate the user and identify them on subsequent visits. The consuming application will store the persistent ID provided by ePass Montana ((EPassPrincipal)User).UserId or User.Identity.Name with the application data itself. When the user returns the application data can be retrieved from the data store using the same persistent ID. When submitted the application would tie to a status field in the data store that could be displayed to the user if they return after submission. Finally the application could use the name and the email address provided by the module ((EPassPrincipal)User).Attributes.FirstName, .LastName, .EMail to send the user a notification when their application had been processed.

To accomplish this integration the application developer would need to complete all steps in the Quick Start Guide in Chapter 2 and set the EPassConfig.Endpoints.PostLoginRedirect setting in the web.config. The page specified by the PostLoginRedirect URL would start a new application, resume an application, or display the status of the submitted application base on the data tied to the persistent ePass ID.

8.1.2 Government Employee Classifieds
The consuming application is a State of Montana employee classifieds application. There is no public landing page for the service therefore all pages are secure. The only requirement to access the system is a valid c number.

When the entire application is secure there is no need for a login page. Simply secure your application as outlined in Chapter 6.3 and when a user navigates to any page they will immediately be redirected to the URL specified by EPassConfig.Endpoints LoginPage. Setting this value to “~/Login.epass” will cause the user to be immediately redirected to the ePass Montana Login Page. Since application only allows state employees to access the system, the State of Montana Identity Provider (IdP) is the only IdP selected when completing the Service Provider Request form. That ensures the user has logged in with a valid c number. The application could again use the name and the email address provided by the module ((EPassPrincipal)User).Attributes.FirstName, .LastName, .Email as required by the application.
8.2 API Implementations

The use cases outlined in this section all require API integration of some kind. They are provided as an example of what can be done by leveraging the API integration points. These use cases require all steps in the Quick Start Guide be completed in addition to and as a predecessor of the steps outlined in each use case.

8.2.1 Water Bill ePayment Site

The consuming application allows users to view and pay their water bill online. The user’s identity is important in that they must identify themselves as an account holder with the water department.

The key components to this implementation are federation and registration. To ensure the user has registered on the system the EPassConfig.Options.RequireFederation flag must be set to “true” and the EPassConfig.Endpoints.RegistrationPage must be set in the web.config. Additionally the consuming application must implement the Federation API (see Chapter 4.3). When a user first logs in the Federate method will be called on the API implementation class. If that method does not set principal.IsFederated = true (because the user was not found), the user will be redirected to the RegistrationPage. The consuming application will prompt the user to enter identifying information on that page (Address, Account #, last payment amount and date…) and create a federation record linking the ePass Montana ID to the internal account number so the Federate method can retrieve this information on subsequent visits. The consuming application then sets 

((EPassPrincipal)User).IsFederated = true and .FedId = accountNumber before redirecting the user to ~/CompleteRegistration.epass to resume the login process (see Chapter 3.4.7 for more details). The consuming application can now use ((EPassPrincipal)User).FedId anywhere in the application to retrieve the account holder’s information.

8.2.2 Job Listings Site

The consuming application is a job search tool for State of Montana jobs. The application retains the user’s search preferences between visits as well as a list of recently viewed jobs.

The federation and registration features follow the pattern of the water bill use case except for the following:

The Federate method only validates that the user has an existing profile. If so, the Federate method sets principal.IsFederated=true (no FedId is set as there is no local id for the user, the user profile is tied to the ePass Montana user ID).

The Registration page asks some basic demographic information and presents the terms and conditions (the user’s true identity does not need to be established). The registration data is saved to the user profile data store before setting principal.IsFederated=true and redirecting the user to ~/CompleteRegistration.epass to resume the login process.

This implementation must also implement the ProfileAPI as outlined in Chapter 4.5. The GetProfile method loads the profile established in the registration process to a business object and returns it. As the user searches their profile is read ((EPassPrincipal)User).Profile and is updated with search history and preferences. When the user logs out or times-out the SaveProfile method is called to update the data store.
8.2.3 Online Store

The consuming application is a simple online ecommerce storefront. For end users (shoppers) the security model looks a lot like the Job Listing example. The unique requirement here is that Administrators must be allowed to log into the site and fulfill orders and manage inventory.

This requirement necessitates the use of role base security. To add role based security using the ePass Montana Service Provider Module you must implement the Authorization API as defined in Chapter 4.4. To assign roles to administrators or other privileged users the consuming application must have a role data store and a mechanism for identifying and federating the targeted users. There are several different ways to achieve this that include but are not limited to:

1. Set up a temporary user token and link it to the desired roles in the local data store. Send the administrator a link to a protected page that includes this token in the URL. When the Federate() API method is called the token can be extracted from the http context and used to federate the user.

2. Offer an admin registration form that is also protected. The form could submit an email to the role administrator, and since the user would have had to login to access the page the ePassMontana persistent ID could be included for manual confirmation and affiliation in the role data store.

3. Add the ePass UserName to the federation data store and have the administrator register as a normal user and send an email with their ePass ID to the role administrator requesting additional roles. The role administrator could then look up the ePassMontana persistent ID in the federation table and assign the appropriate roles.

4. Allow the ePass Montana IdP as well as the State of Montana IdP when completing the ePass Montana Service Provider Request Form. Administrators would be required to login using their c-number and the consuming application could look up the c-number in the State Active Directory to obtain the groups the user is assigned. Base on group membership the appropriate roles could be assigned.

Whichever method is used the goal is to federate the user and associate them to a list of role name using the Federate (Chapter 4.3) and GetRoles (Chapter 4.4) API methods. Additionally the administrative pages must be protected, limiting access to the desired roles as outlined in Chapter 6.3.

8.2.4 ePass Montana Retrofit

If the consuming application already has a security provider the conversion to ePass Montana Authentication is pretty straight forward. The implementation will need to implement the Federation (Chapter 4.3) and Authorization (Chapter 4.4) APIs. The registration page would be the old login page. When a user logs in and is not federated, they are prompted for their old credentials. The consuming application validates the credentials and federates the users. From then on the Federate() API method would set the FedId to the old user name and the GetRoles() API method would use the FedId to retrieve the original list of roles. The existing Role Management UI of the consuming application could even still be used.