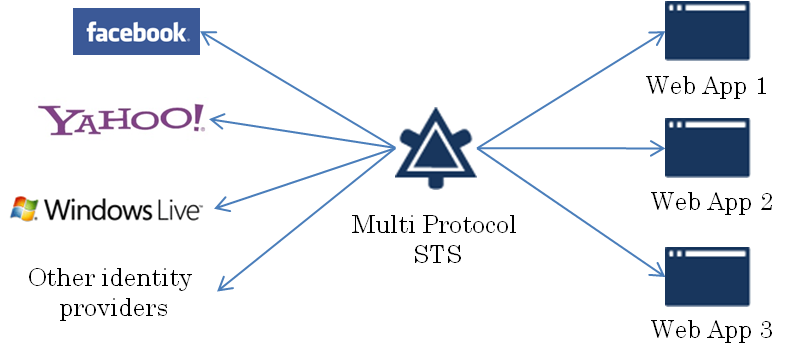
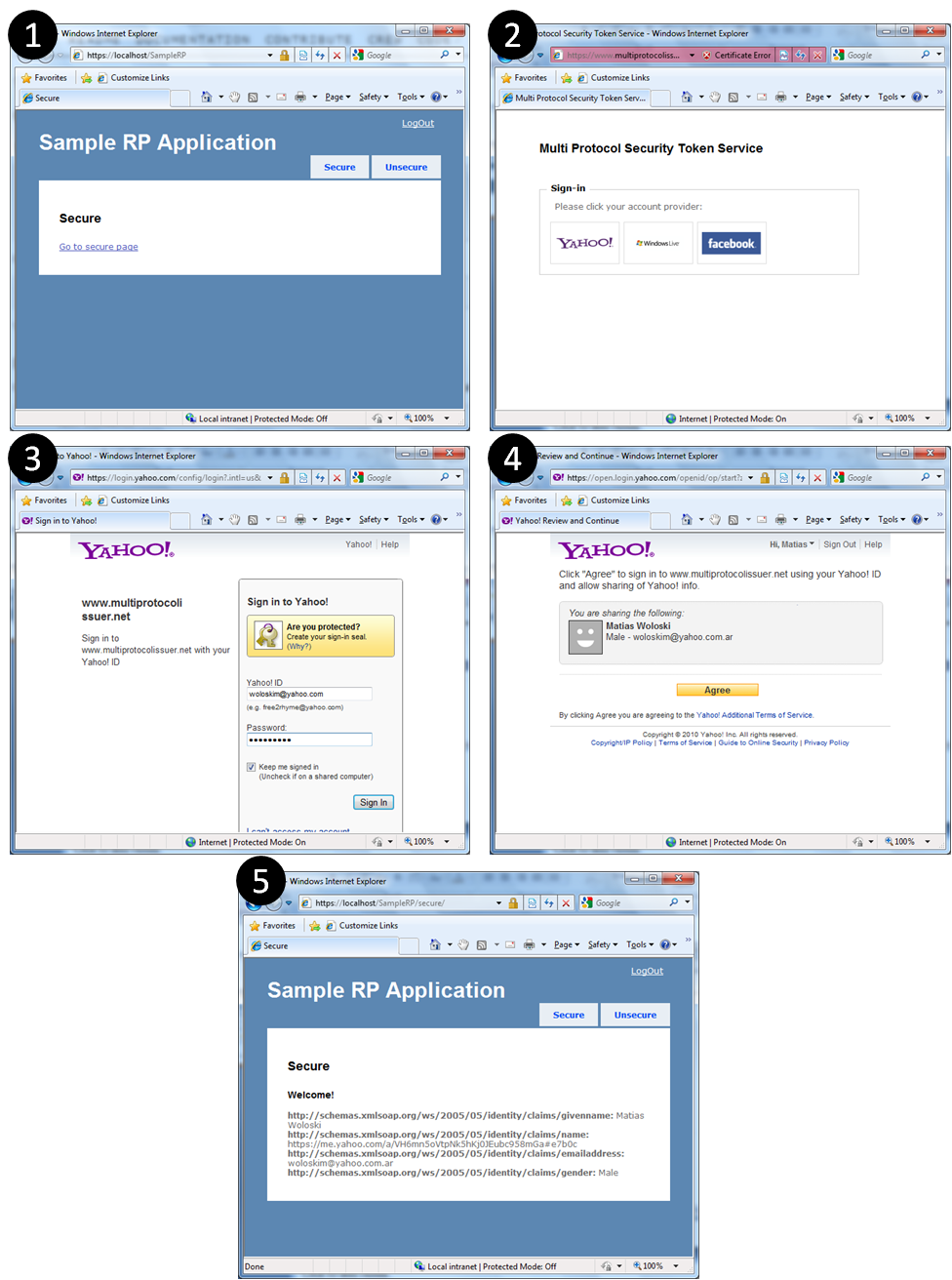
Protocol Bridge Claims Provider

# Getting Started

The Multi Protocol Security Token Service is a WS-Federation aware token issuer that allows bridging with other authentication protocols like OpenID, OAuth or other custom protocols like LiveID.

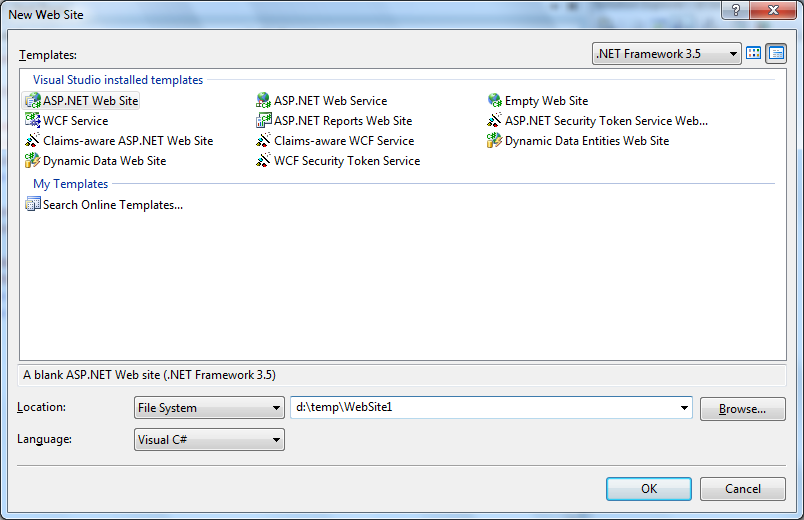
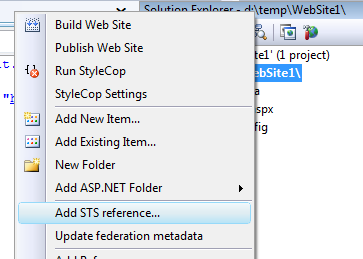
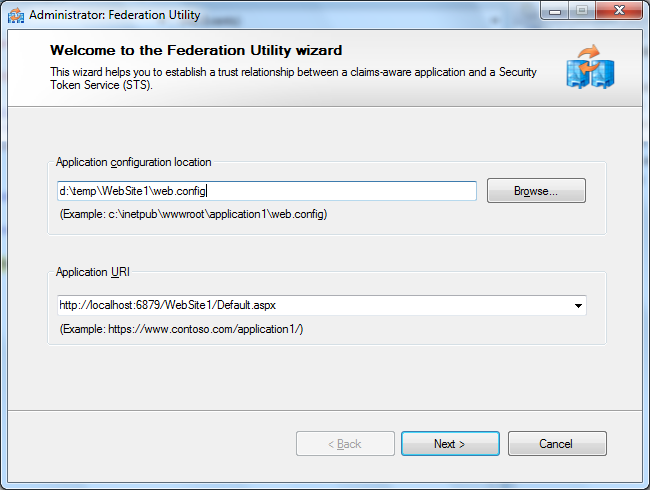
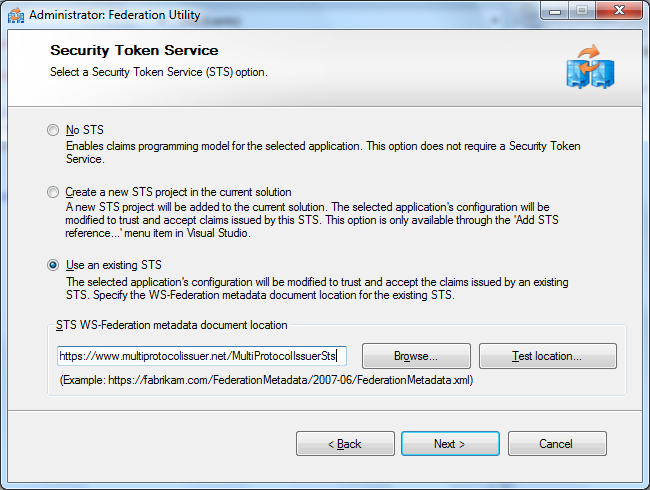
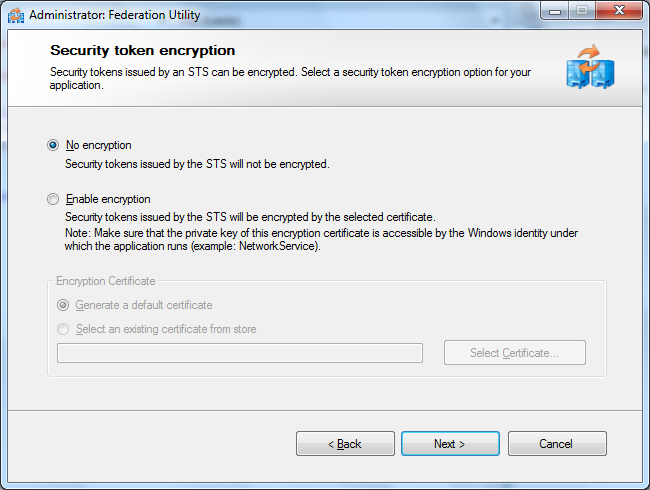
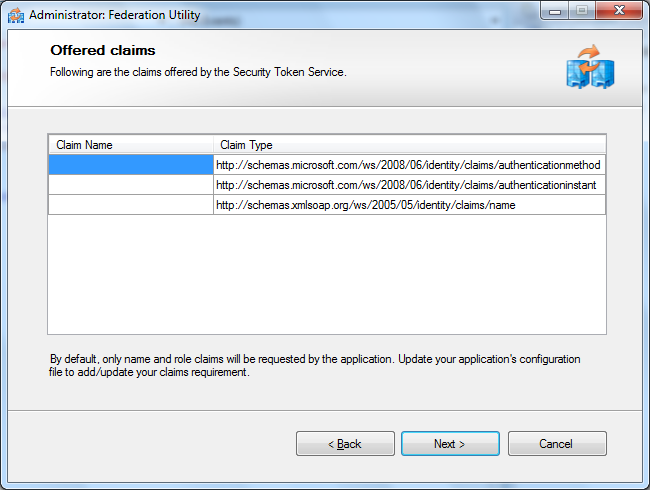


Once installed, open a browser and go to one of the sample web applications <https://localhost/SampleRP>. This is the flow between the web application (ASP.NET MVC with WIF), the STS (ASP.NET MVC with WIF) and Yahoo identity provider (OpenID).



## Configuring an ASP.NET application to use the Multi Protocol STS

The following step by step instructions show how to configure an ASP.NET application to trusts the Protocol Bridge Claims Provider.

* 1. Open Visual Studio 2008 (or 2010)
  2. Create a new **ASP.NET Web Site**
     1. 
  3. Right click on the project and press **Add STS Reference…**
     1. 
  4. Leave the default values but add **Default.aspx** in the **Application URI textbox**.
     1. 
  5. Select Use an existing STS and enter the url of the multi protocol STS (the installation will create an entry in the HOSTS file and virtual directory in IIS default website). The default url is <https://www.multiprotocolissuer.net/MultiProtocolIssuerSts>. Press Yes when the certificate warning shows up.
     1. 
  6. Select **No encryption** and press Next. By default the Multi Protocol STS does not encrypt tokens.
     1. 
  7. These are the claims offered by the STS. Press Next
     1. 
  8. Press Finish.
  9. Replace the Default.aspx markup with this (this markup will extract the claims from the IClaimsIdentity of the principal and show them)
     1. ASPX
     2. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" ValidateRequest="false" %>
     3. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
     4. <html xmlns="http://www.w3.org/1999/xhtml">
     5. <head runat="server">
     6. <title></title>
     7. </head>
     8. <body>
     9. <form id="form1" runat="server">
     10. <div>
     11. <% foreach (var claim in ((Microsoft.IdentityModel.Claims.IClaimsIdentity)User.Identity).Claims)
     12. { %>
     13. <strong><%=claim.ClaimType %></strong>:<%=claim.Value %><br />
     14. <% } %>
     15. </div>
     16. </form>
     17. </body>
     18. </html>
  10. Open the Multi Protocol STS Web.config and add the following scope to the list of scopes.
      1. Web.config
      2. <scope
      3. uri="http://localhost:6879/WebSite1/Default.aspx"
      4. identifier="http://localhost:6879/WebSite1/Default.aspx"
      5. useClaimsPolicyEngine="false" />

# The story behind

This work is the result of a proof of concept we did with the Microsoft Federated Identity Interop group (lead by [Mike Jones](http://self-issued.info)), [Medtronic](http://www.medtronic.com) and [PayPal](https://www.paypal-ids.com/). The business scenario brought by Medtronic is around an insulin pump trial registration. In order to register to this trial, users would login with PayPal, which represents a trusted authority for authentication and attributes like shipping address and age for Medtronic. However, authentication alone is not enough for the purpose of the trial. Medtronic wants to enrich the identity of the user with other attributes that belong to their realm. In essence, they are doing an identity mash up and using this “mashed” token to access other applications/services. An example of this kind of applications is a site we created called *Contoso Opinions* that allowed users with a Medtronic token (containing PayPal and Medtronic attributes) to comment about their experience with the device.

While there are many ways to solve a scenario like this, we chose to create an intermediary Security Token Service that understands the OpenID protocol (used by PayPal), WS-Federation protocol and SAML tokens (used by Medtronic apps). This intermediary STS would also allow SSO between the web applications, avoiding re-authentication with the original identity provider (PayPal).

We decided together with the Microsoft Federated Identity Interop team to make the implementation of this intermediary multi protocol STS available under open source.

# Configuration

## General Properties

The Multi Protocol STS has some general properties

* 1. Web.config
  2. <multiProtocolIssuer identifier="https://www.multiprotocolissuer.net/" responseEndpoint="https://www.multiprotocolissuer.net/MultiProtocolIssuerSts/response">
  3. <signingCertificate findValue="CN=localhost" storeLocation="LocalMachine" storeName="My" x509FindType="FindBySubjectName"/>

|  |  |
| --- | --- |
| Configuration Attribute/Element | Description |
| identifier | The identifier of the STS (an arbitrary URI) |
| responseEndpoint | An endpoint that would accept the HTTP response (if any) from the different protocols |
| siginingCertificate | the X509 certificate used to sign the outgoing tokens |

## Claims Providers

In the Multi Protocol STS, a claims provider is an authoritative source of claims. These are configured in the Web.config as shown below. A claims provider talks a specific protocol (OpenID with Attribute Exchange, OpenID with SReg, OAuth 2.0, a custom protocol, etc.), so a claims provider is linked to a protocol handler, a class that implements the protocol. The Multi Protocol STS was tested against three well known claims provider: Yahoo, Facebook and LiveId.

* 1. XML
  2. <claimProviders>
  3. <add identifier="urn:Yahoo:AX" url="http://yahoo.com/" protocolHandler="OpenIdAxHandler" profile="AX" />
  4. <add identifier="urn:LiveId" url="https://login.live.com/wlogin.srf" protocolHandler="LiveIdHandler" allowCompleteProfileForm="true">
  5. <params>
  6. <add key="wll\_appid" value="000000004403607C"/>
  7. <add key="wll\_secret" value="IbAu38iuwwRbst5vl1CuJRfHx3e7vcWl"/>
  8. <add key="wll\_securityalgorithm" value="wsignin1.0"/>
  9. </params>
  10. </add>
  11. <add identifier="urn:Facebook" url="https://graph.facebook.com/oauth/authorize" protocolHandler="FacebookHandler" profile="OAuth2.0">
  12. <params>
  13. <add key="api\_url" value="https://graph.facebook.com" />
  14. <add key="application\_id" value="122744481090516" />
  15. <add key="api\_key" value="b61ec28402117d024abc235177c82e92" />
  16. <add key="secret" value="51fb4330162e14914832131f39f8a530" />
  17. </params>
  18. </add>
  19. </claimProviders>

|  |  |
| --- | --- |
| Configuration Attribute/Element | Description |
| identifier | The identifier of the claim provider (an arbitrary URI). This identifier has to match with the *whr* parameter used in the website that hosts the STS. Clicking on one of the logos, will redirect to /authenticate?whr=*identifier-of-the-claims-provider* |
| url | The endpoint of the claim provider where the user will be redirected |
| protocolHandler | The friendly name of the protocol handler class that will process the sign in request and response. The list of protocol handlers can be found in the Web.config in the <unity> section. |
| profile | The profile implemented within the particular protocol. This attribute is just informative and does not change any runtime behavior |
| params | A set of key value pairs that will be sent to the protocol handler. Typically, this key value pairs will hold API keys, app identifiers, etc. needed to talk against the claim provider  **Note**: if you will use this in a real application make sure to change the default parameters. |

## Scopes

In the Multi Protocol STS, a scope is the application that is requesting a token. These are configured in the Web.config as shown below and represent a white-list of applications that can use the STS.

* 1. Web.config
  2. <scope uri="https://localhost/SampleRP" identifier="https://localhost/SampleRP" useClaimsPolicyEngine="true">
  3. <claimRequirements>
  4. <add type="http://schema.openid.net/namePerson" name="name" demandLevel="Request" />
  5. <add type="http://schema.openid.net/contact/email" name="email" demandLevel="Request" />
  6. <add type="http://schema.openid.net/person/gender" name="gender" demandLevel="Request" />
  7. <add type="http://axschema.org/namePerson" name="name" demandLevel="Request" />
  8. <add type="http://axschema.org/contact/email" name="email" demandLevel="Request" />
  9. <add type="http://axschema.org/person/gender" name="gender" demandLevel="Request" />
  10. <add type="http://schema.facebook.com/me/email" name="email" demandLevel="Request" />
  11. </claimRequirements>
  13. <allowedClaimProviders>
  14. <add name="urn:Yahoo:AX" />
  15. <add name="urn:MyOpenID:Sreg" />
  16. <add name="urn:AOL:AX" />
  17. <add name="urn:LiveId" />
  18. <add name="urn:Facebook" />
  19. </allowedClaimProviders>
  21. </scope>

|  |  |
| --- | --- |
| Configuration Attribute/Element | Description |
| identifier | ***Required***. The identifier of the scope (an arbitrary URI). This identifier has to match with the *wtrealm* parameter sent by the web application. |
| url | ***Required***. The endpoint of the scope where the token will be posted back if no *wreply* parameter was sent. |
| useClaimsPolicyEngine | ***Optional*** (default false). If false, all the claims coming from the claim provider will be copied to the output token. If true, a rules engine will be used to transform the incoming claims to outgoing claims. This engine uses XML to author the rules. See the Claims Policy Engine for more details [TBD]. |
| claimRequirements | ***Optional***. Some of the protocols (like OpenID) require to specify the claim types to demand from the claim providers. This section allows configuring what claims should be requested for a specific application. Depending on the implementation of the claims provider it will ask the user to give consent about the disclosure of those claims. Each claim has a demand level (NoRequest, Request, Require). |
| allowedClaimProviders | ***Optional***. A set of claim providers allowed for a specific application. The intended usage of this is to show/hide the logos on the identity selector web page. It does not affect the runtime behavior. |

# More info

## Known Issues

* + Single sign on is not working properly with web applications hosted in localhost.
  + The allowedClaimProviders configuration does not alter the behavior of the STS
  + Exception handling can be enhanced with more detailed errors
  + No performance/stress test has been done

## Main contributors

* Juan Pablo Garcia (blog: <http://blogs.southworks.net/jpgarcia> twitter: @jpgd mail: juanpablo dot garcia at southworks dot net)
* Matias Woloski (blog: <http://blogs.southwokrs.net/mwoloski> twitter:@woloski mail: matias at southworks dot net)

1. [](http://github.com/southworks)