Atomia Identity

16 April 2020, Version 20.3.7411

Table of Contents

1. Structure

- Overview
  - A schema of all servers modules, agents and services
- Installation
  - Prerequisites (like Atomia DNS Master server installation#Software used by UCP DNS Master server)
- Updates
- Configuration
  - Description of `web.config`
- API Reference - User API
- Identity SDK
- Usage
  - Code Examples
- Technical information
- FAQ
- Release Log
- Roadmap (if applicable)
- License

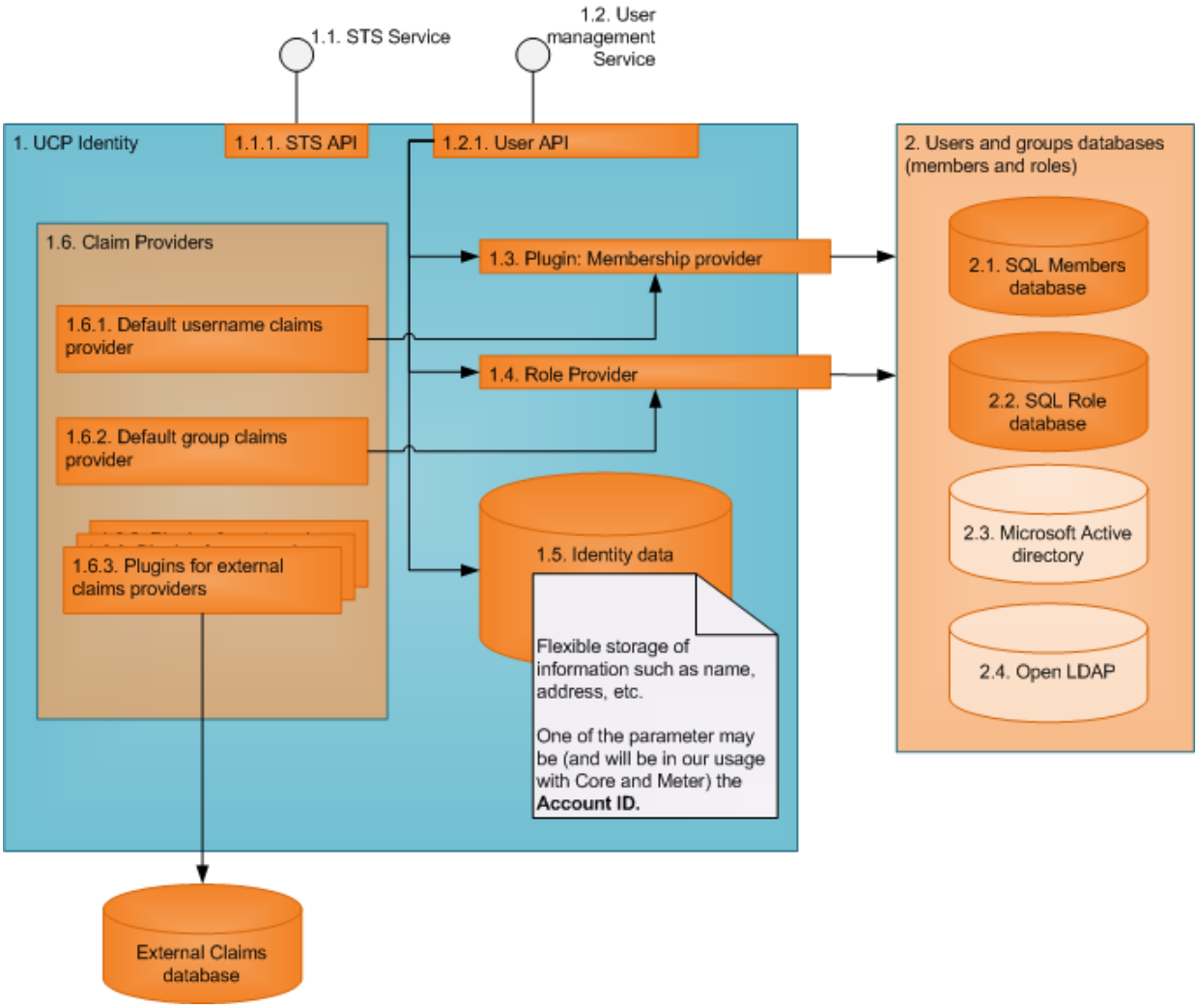
2. Atomia Identity - Overview

2.1. Description

Atomia Identity is the service for Identity management in the Atomia Provisioning system.

It provides the following functionalities:

- Claim-based, SAML, authentication and authorization.
- Pluggable sources of claims.
- Identity data management (name, address, email, etc.).
- User and groups management.

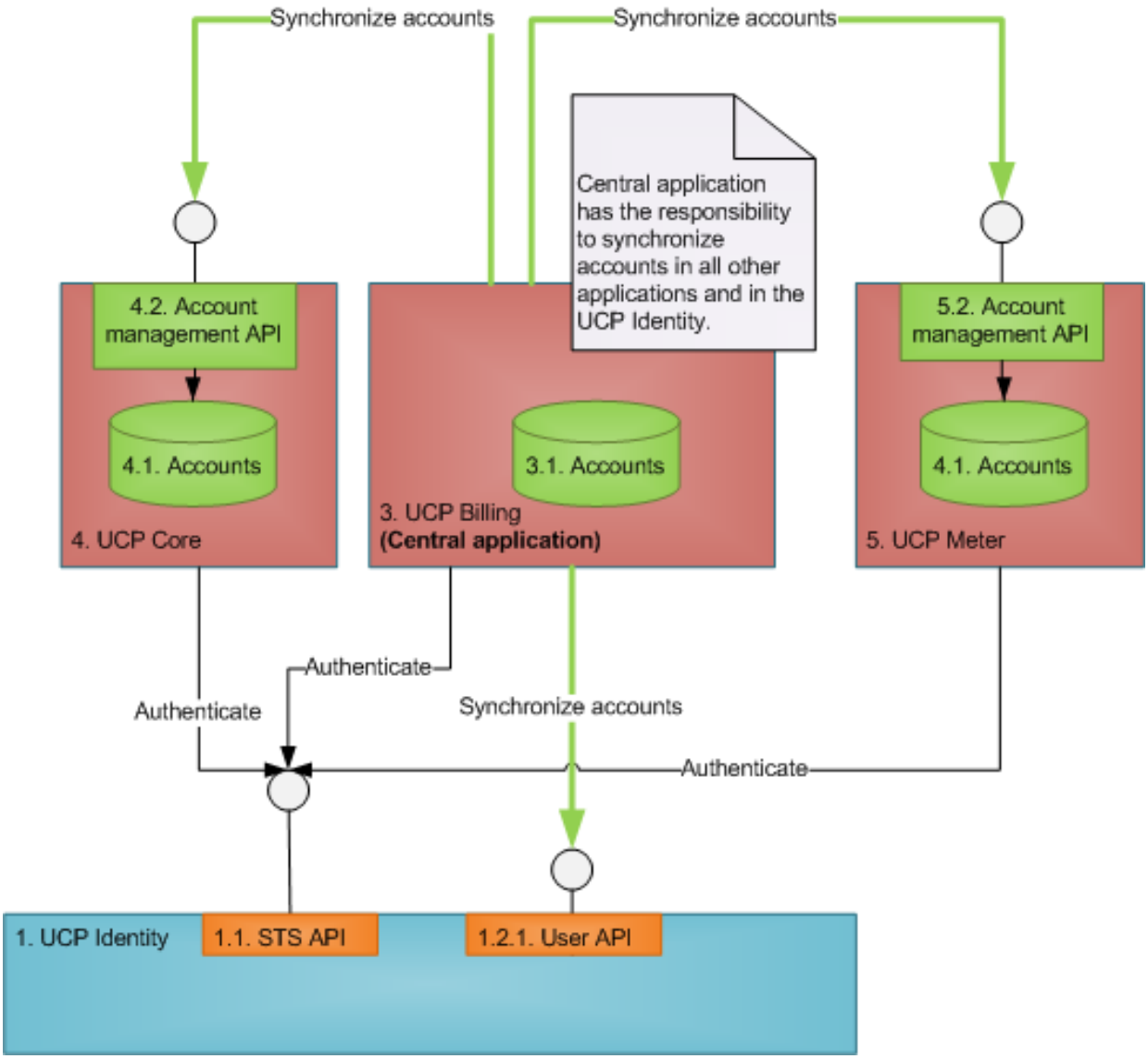


2.2. Environment

Typically, Atomia Identity is used by several applications. These applications use the two main features of Atomia Identity:

1. **STS (Security Token Service) for SAML based authentication and authorization** - this feature is used by all applications to authenticate and authorize users.

2. **User Management** - this service is used by the central application to maintain the list of users and their properties.



3. Atomia Identity - Installation

3.1. System Requirements

To install Atomia Identity the system must meet following requirements:

- Microsoft Windows Vista or Microsoft Windows Server 2008 or newer.
- Microsoft .NET Framework 3.5 SP1.
- Microsoft SQL Server 2008 (Express)
- Internet Information Services 7 (IIS 7)
  - Installed Windows, Basic and Anonymous authentication features.
  - ASP, ASP.NET, .NET Extensibility, ISAPI extensions, ISAPI filters features installed.

3.2. Firewall requirements

Identity communicates with Provisioning service, Hosting control panel, Billing control panel and AtomiaAccountApi services. Firewall should allow communications in both ways with servers where those applications/services are installed. Communication is done over http protocol on port 80.

3.3. System and domain accounts requirements

For provisioning services there must be domain account with following rights:

- full domain rights
- elevated system privileges

This account will be used for accessing MSSQL server database and IIS7 administration.

3.4. Installation

To install Atomia Identity follow this instructions.

- First step is to download and start `AtomiaIdentitySetup.exe` application. Window shown on Figure 1 will be shown.

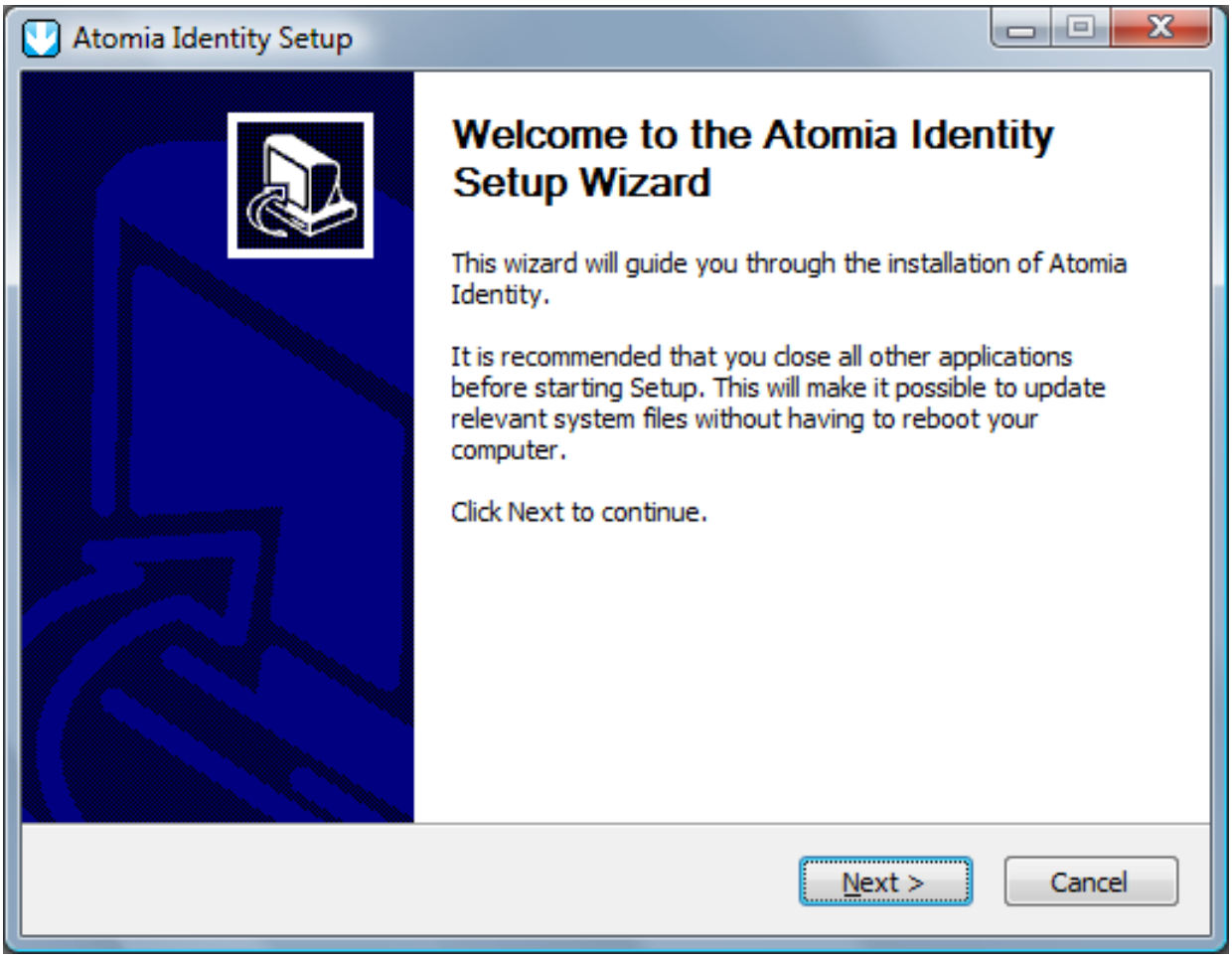


Figure 1: Atomia Identity Setup welcome screen

- To proceed with installation click the **Next** button.
- Window with the license text will be shown. To continue with setup click on the **Next** button.

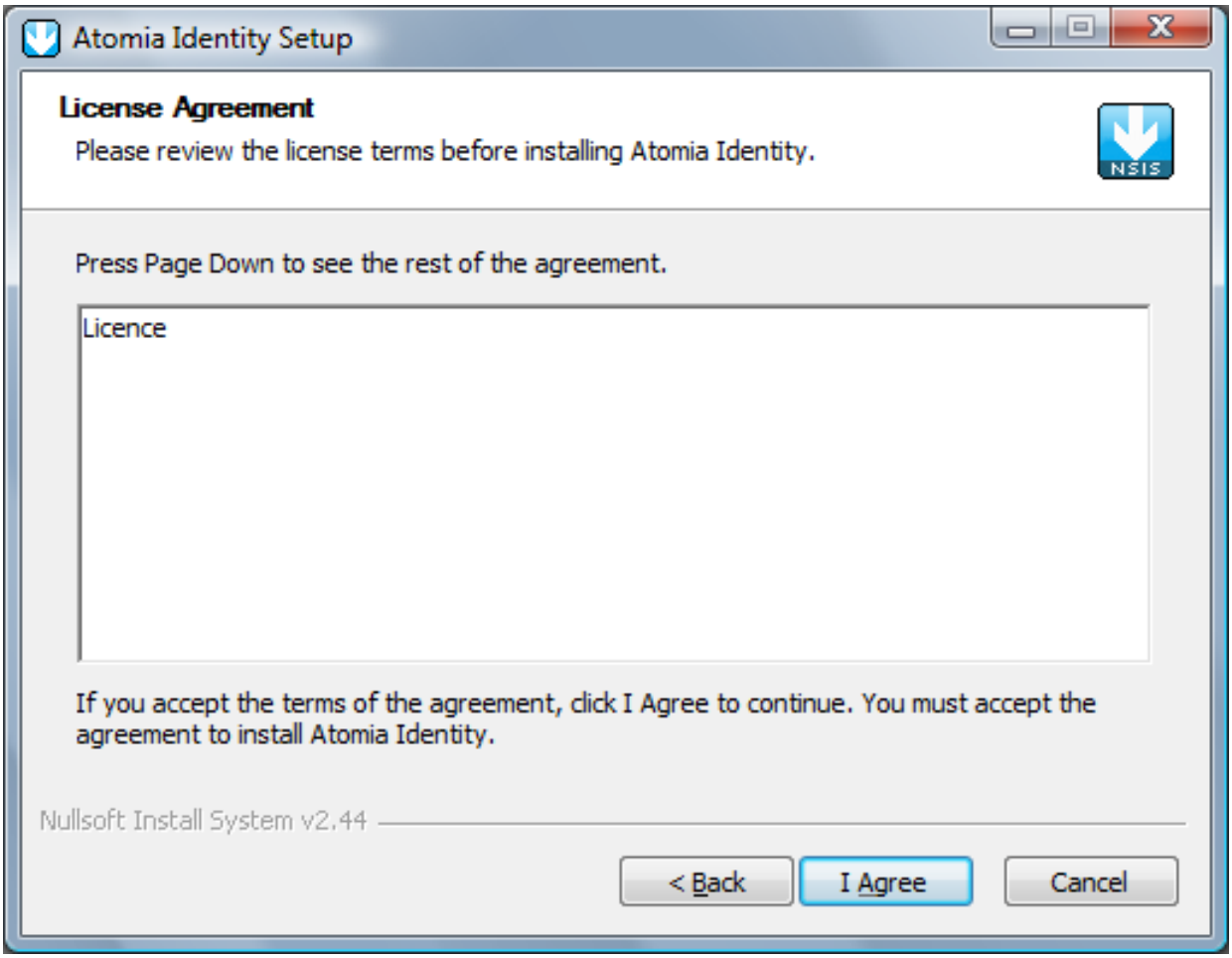


Figure 2: Atomia Identity License Agreement

- Click **Next** to proceed.
- The next step is installation of Microsoft SQL Server Express and databases. Enter the name of the server and choose the type of authentication. Click **Next** .

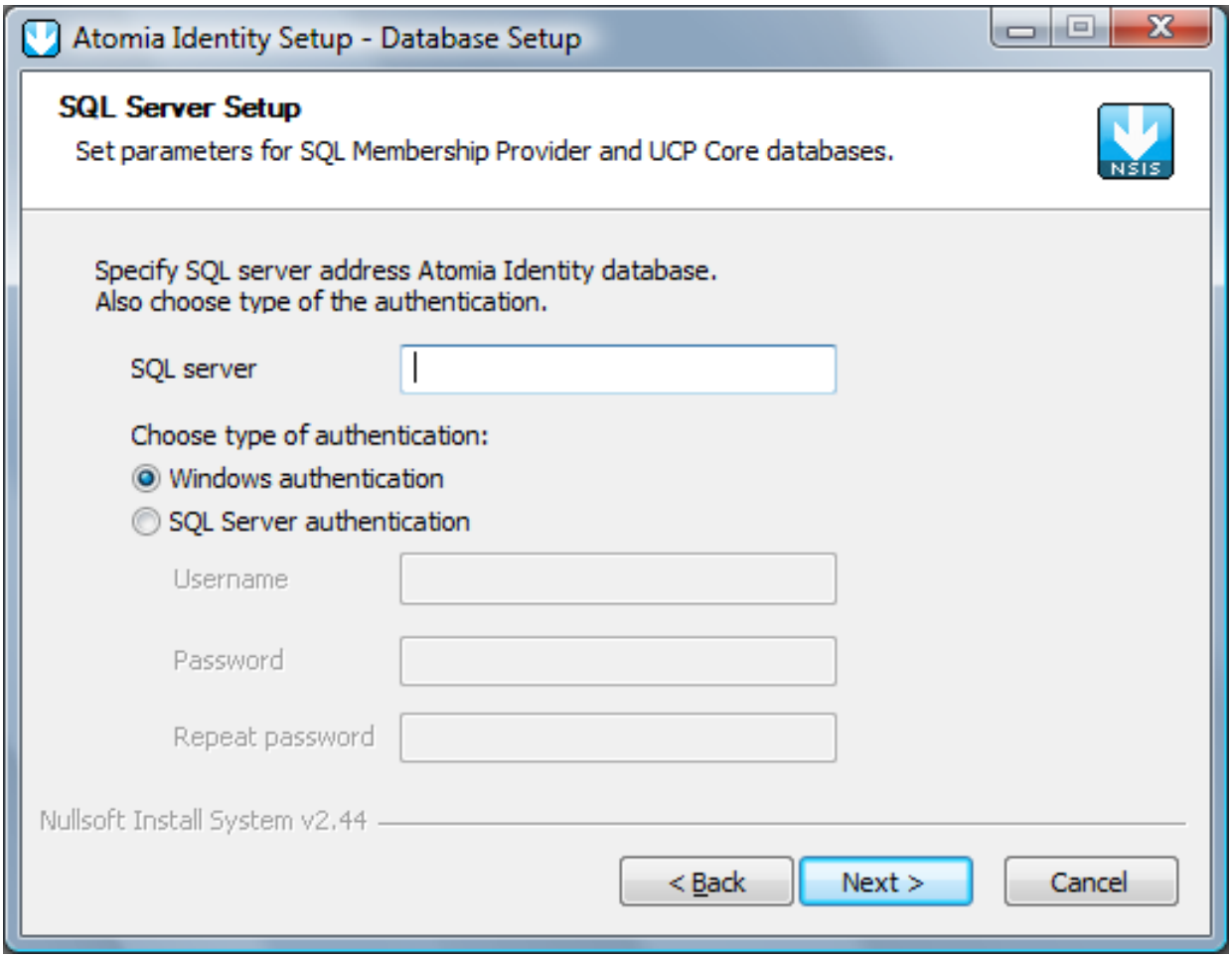


Figure 3: Microsoft SQL Server Express installation

- In the next step provide an account that will be used to run the Atomia Identity service and click **Next** to proceed.

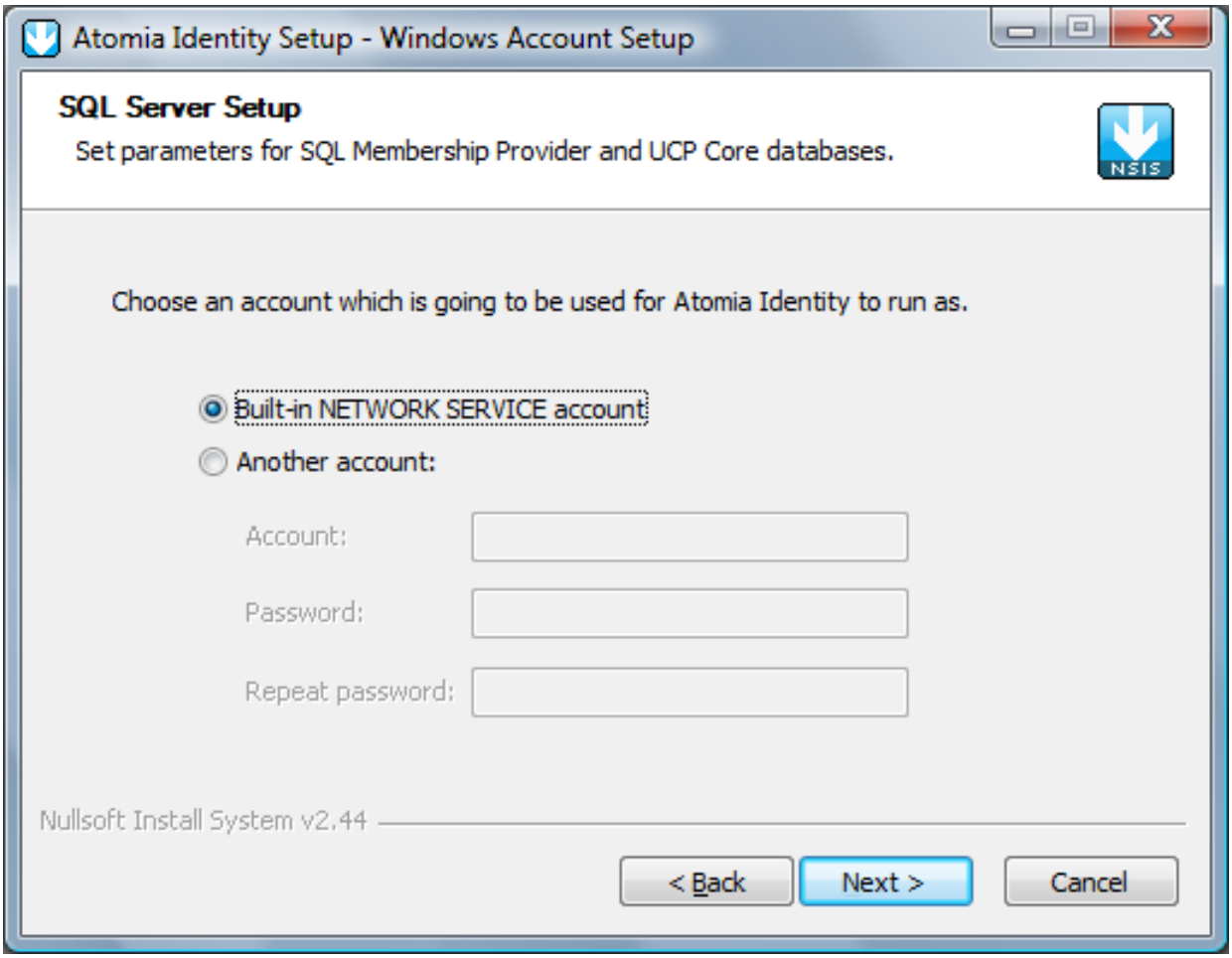


Figure 4: Atomia Identity service account setup

- Select whether to install the Atomia Identity for current user only or for all users of the computer.

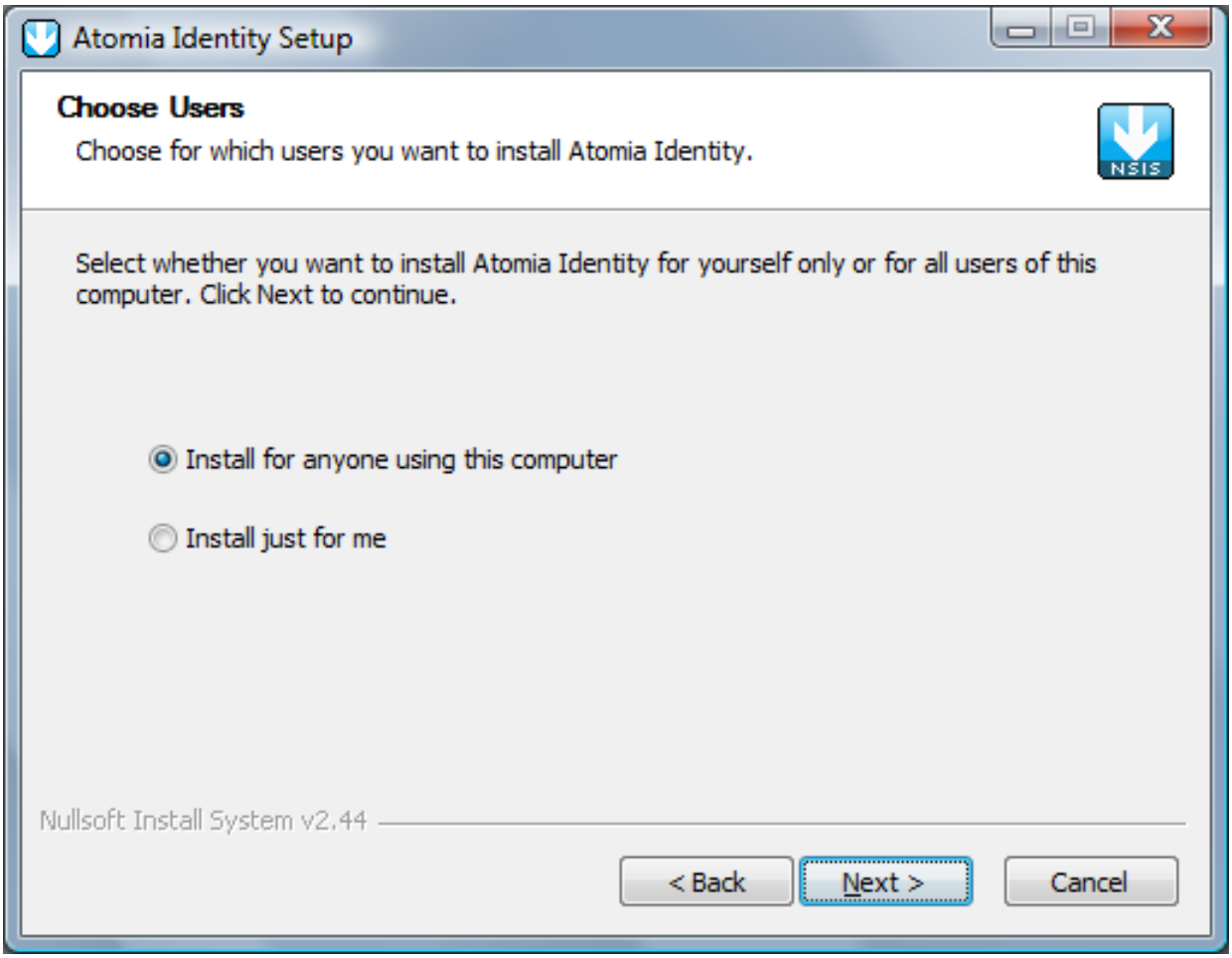


Figure 5: Atomia Identity target user

- Select the destination folder where the Atomia Identity is going to be installed. Click **Next** to continue with installation.

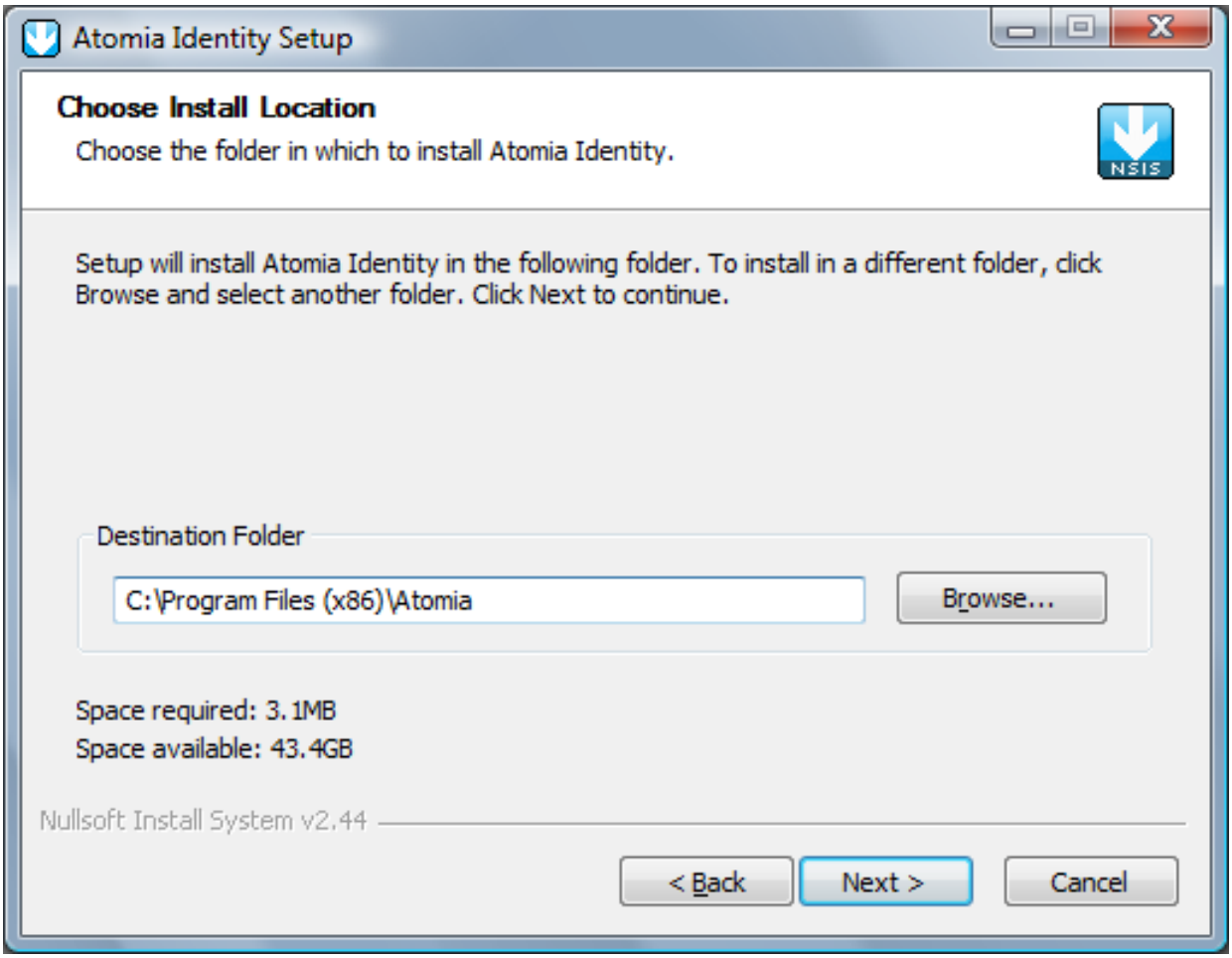


Figure 6: Atomia Identity installation folder

- Select the Start Menu folder where to install the shortcuts for the Atomia Identity..

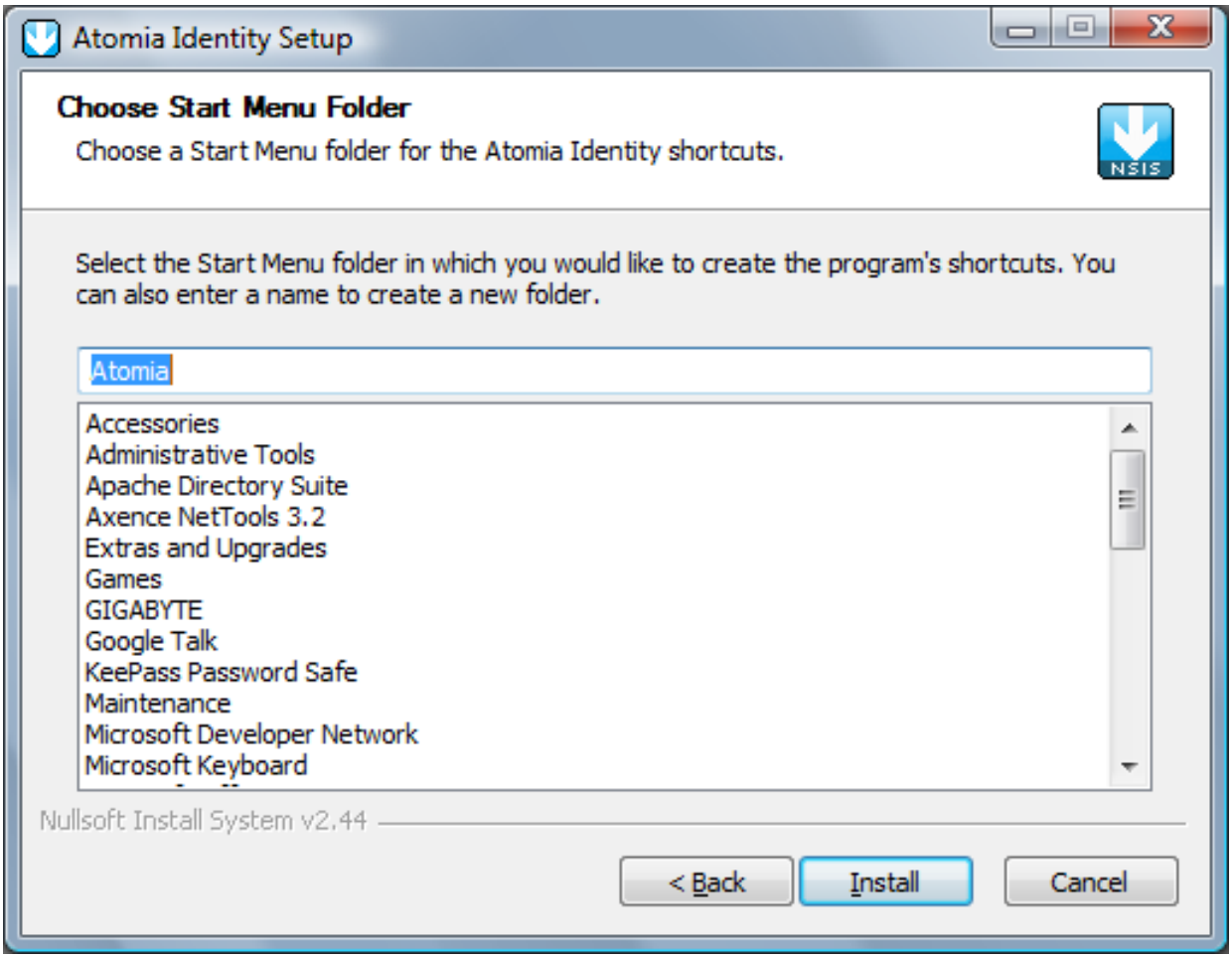


Figure 7: Atomia Identity Start Menu folder

- Click **Next** and the installation process begins. The following window will be shown.

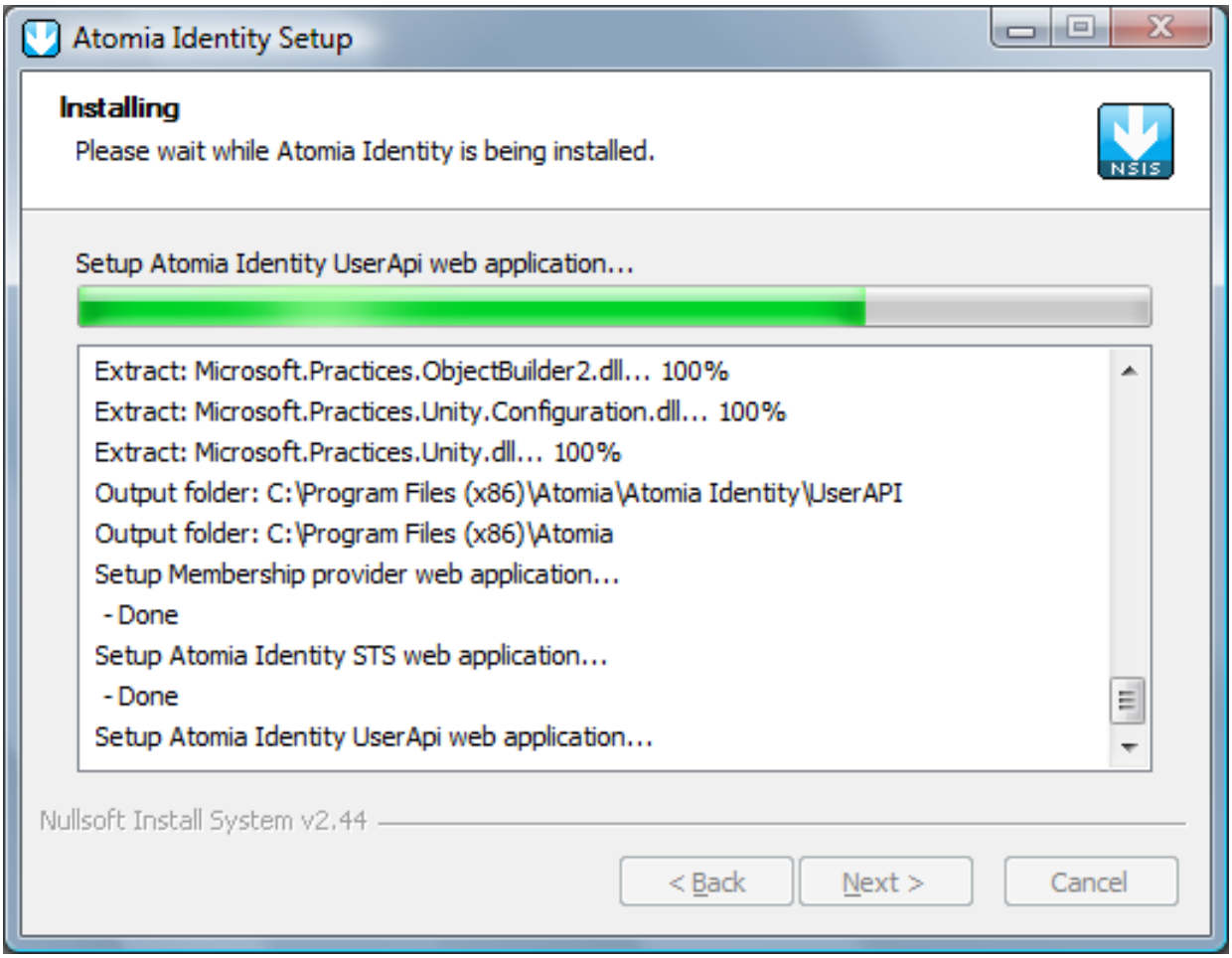


Figure 8: Atomia Identity installation progress

- Wait until installation is finished. You should be able to see the window as the one shown on Figure 10. Click **Next** .

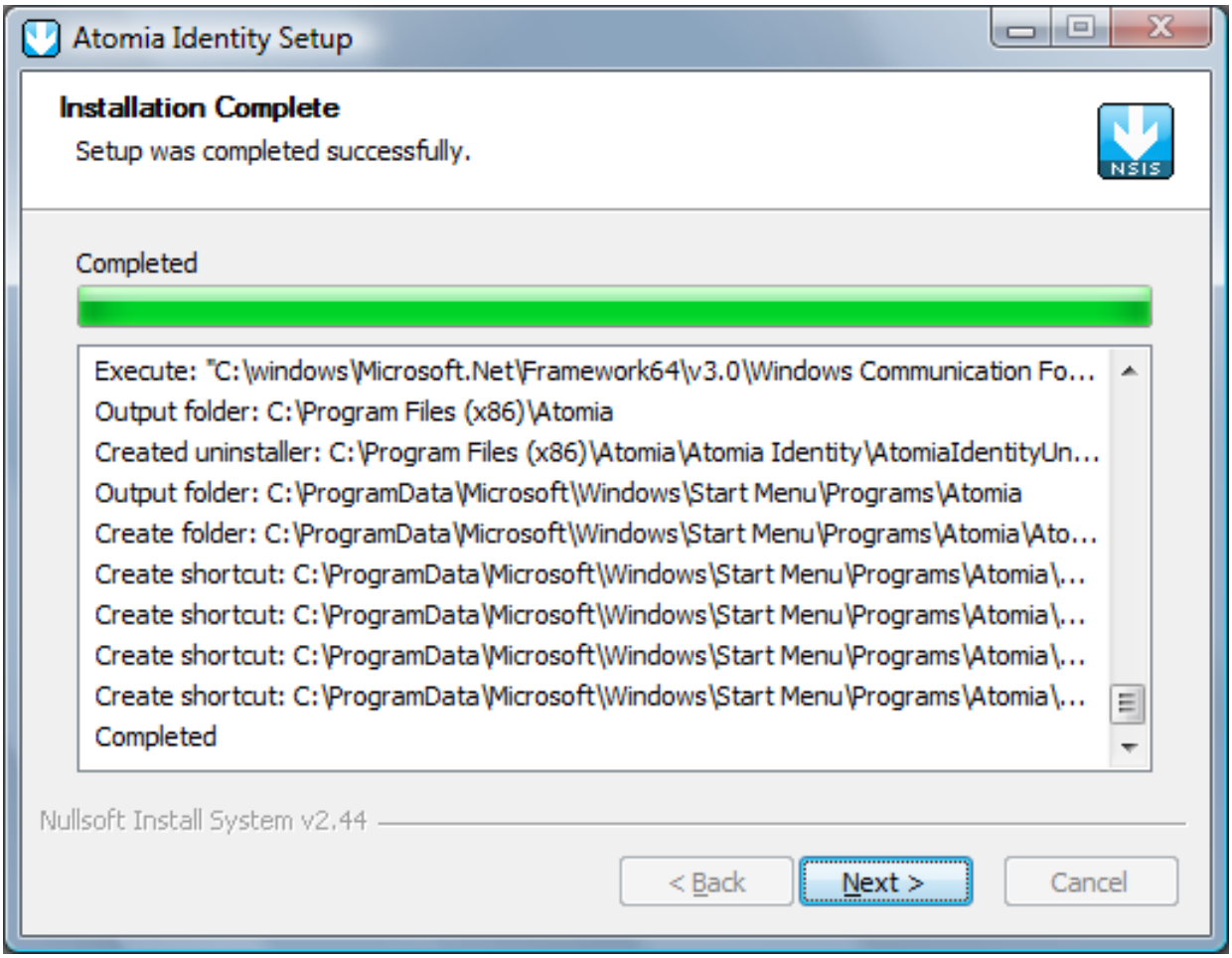


Figure 9: Atomia Identity installation is completed

- The installation procedure ends with the window shown in Figure 10. Click **Finish** .

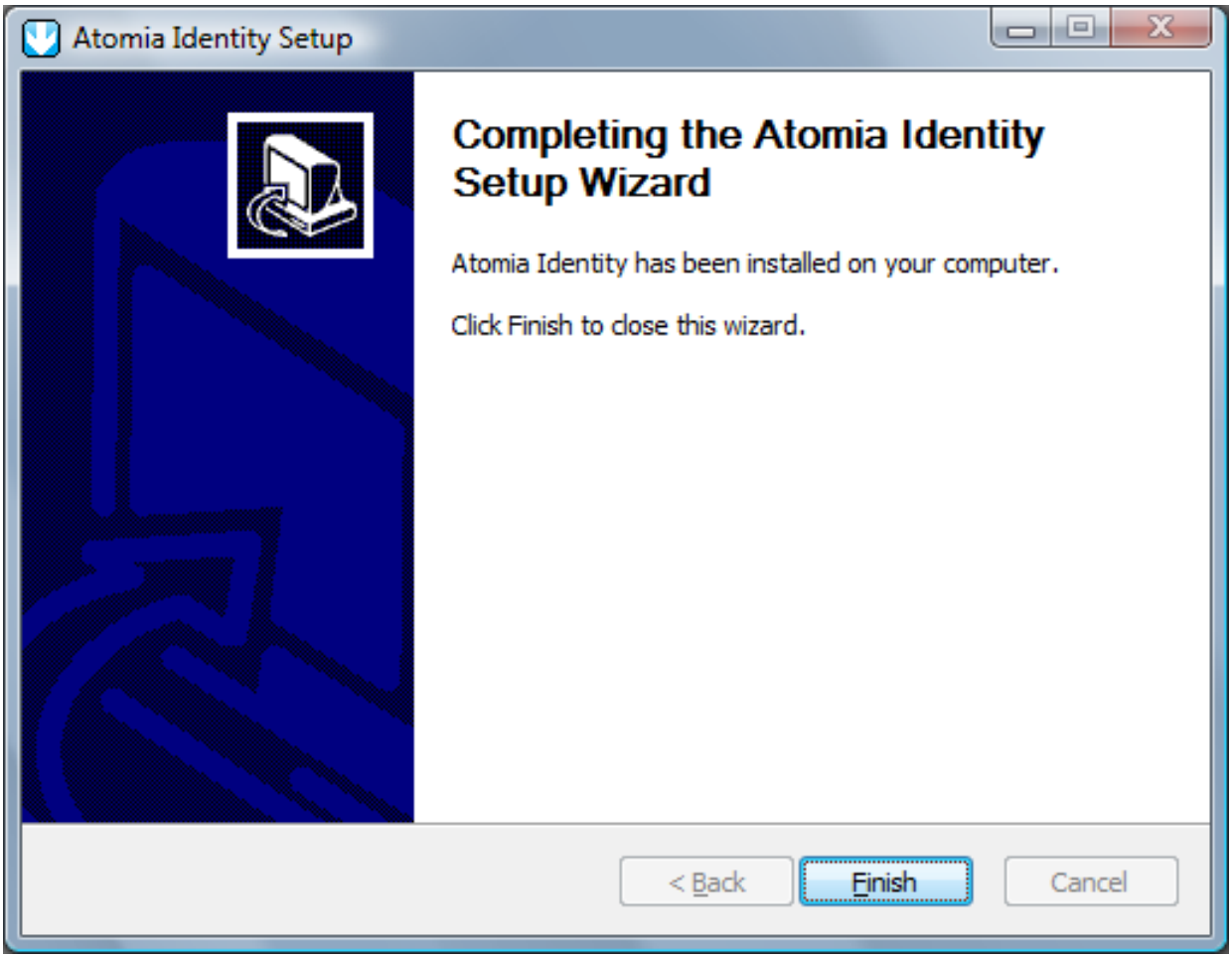


Figure 10: Atomia Identity installation is finished

3.5. Installation troubleshooting

3.5.1. Atomia Provisioning separately installed:

If Atomia Provisioning Service is installed separately, then some manual changes must be made to the web.configuration files of Atomia Provisioning and Atomia Identity. Also, some of the certificates must be added from one machine to another.

3.5.1.1. Certificates:

- From the machine where Atomia Provisioning is installed, take AtomiaProvisioningCer.cer from the installation folder and on the Atomia Identity machine import this certificate (under the Local computer) into the following stores: Personal, Trusted root certificate authorities and Trusted people.

3.5.1.2. Changes to the Atomia Provisioning Web.config file:

- Change the connection string for the Atomia Indentity databases (AtomiaUserManagement and AtomiaIdentity):

```
<connectionStrings>
<add name="IdentityProviderConnectionString" connectionString="Data Source=localhost
\SQLEXPRESS;Initial Catalog=AtomiaIdentity;Integrated Security=SSPI;" providerName="" />
<add name="UserManagementConectionString" connectionString="Data Source=localhost
\SQLEXPRESS;Initial Catalog=AtomiaUserManagement;Integrated Security=True;"
 providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Change certificate thumbprint for the Atomia Indentity certificate.

```
...
```

```
<microsoft.identityModel>
<service>
<issuerNameRegistry
 type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<add name="CN=Atomia Identity" thumbprint="0B1801359CD5F0787E38AF9820544E76B6F9772A" />
</trustedIssuers>
</issuerNameRegistry>
...
```

3.5.1.3. Changes to the Atomia Test Client app.config file

- All addresses which point to the location of the AtomiaIdentitySts service, ie "http://localhost/AtomiaIdentitySts/..." should be changed to point to the real location of the Atomia Indeity Sts service.
- All addresses which point to the Atomia Provisioning service should point to the localhost Atomia provisioning service "http://localhost/AtomiaProvisioning/..."

4. Atomia Identity - Configuration

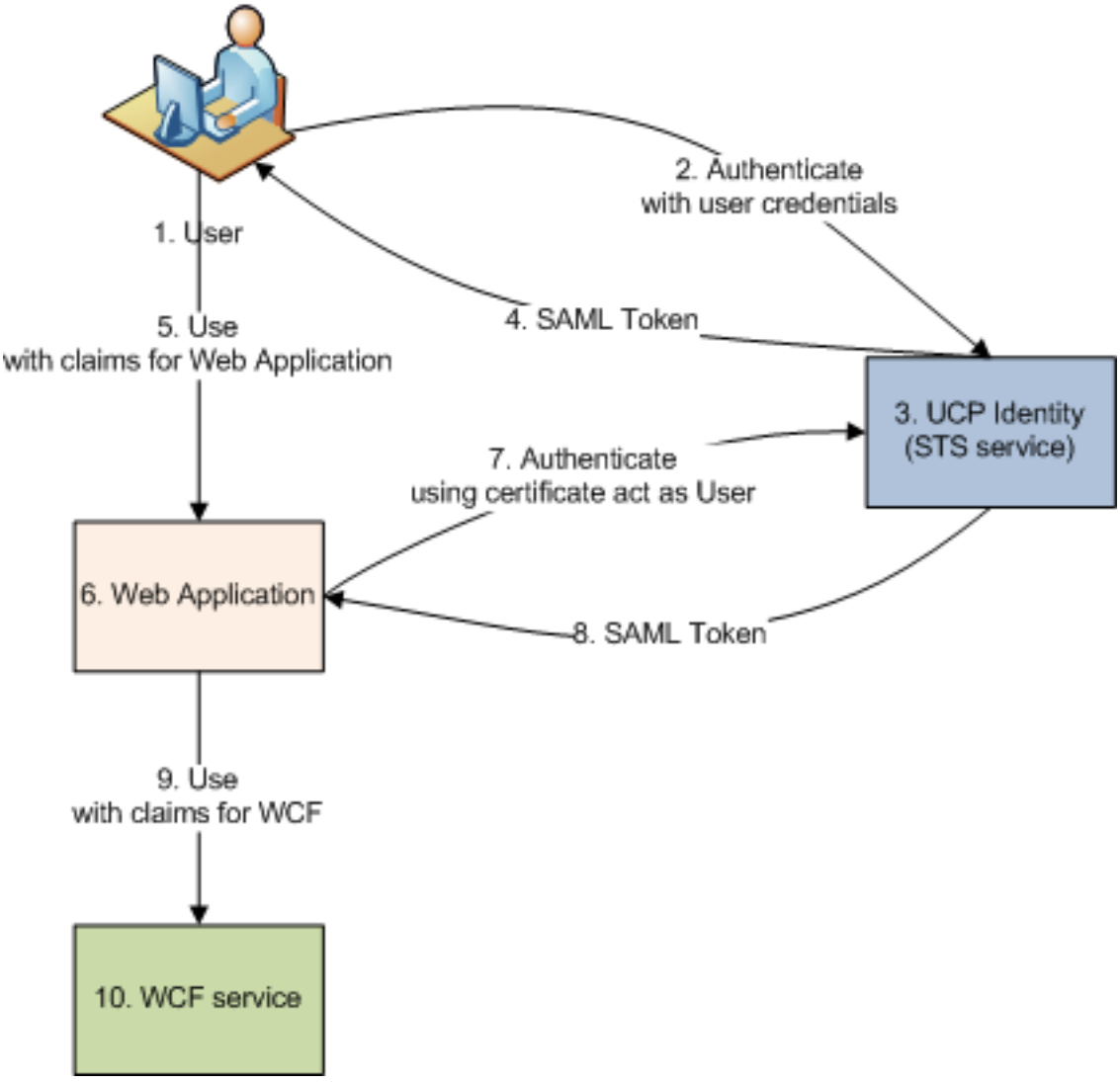
Work in progress

This document is not finished yet. When it gets finished this box will be removed.

4.1. About

This document should describe how should developers configure and use the **Atomia Identity Security Token Service (STS)** from their applications.

4.2. Atomia Identity (STS) Authentication architecture



By looking at the image above we could distinguish two different Authentication paths:

1. **Authenticating user to use web application** (steps 1, 2, 3, 4) - To be able to use the web application user must authenticate to the STS, providing the username and password. Then the user gets the SAML token key containing the set of claims those identify the user to the Web Application.

2. **Authenticating web application to use WCF service (with identity delegation)** (steps 6, 7, 3, 8) - In order to use the WCF service through the web application, web application needs to delegate user's credentials to the STS by providing the user's token (given to the user in the previous iteration) and its certificate. Then the WCF service has the identity information about the logged user and its delegate (web application).

4.3. Configuring WCF to use Identity STS authentication

Video Source

4.3.1. WCF application config file

We are starting from conficuration like this:

```
<configuration>
<system.serviceModel>
<services>
<service behaviorConfiguration="CoreServiceBehavior"
name="Service.CoreService">
<endpoint address="" binding="wsHttpBinding" contract="Service.ICoreService">
```

```
<identity>
<dns value="localhost" />
</identity>
</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
<host>
<baseAddresses>
<add baseAddress="http://localhost:8700/CoreService/" />
</baseAddresses>
</host>
</service>
</services>
<behaviors>
<serviceBehaviors>
<behavior name="CoreServiceBehavior">
<serviceMetadata httpGetEnabled="true" />
<serviceDebug includeExceptionDetailInFaults="false" />
</behavior>
</serviceBehaviors>
</behaviors>
</system.serviceModel>
</configuration>
```

As we can see we are starting with one WCF service `ICoreService` that does not need any authentication.

To enable authentication using `Atomia Identity` in this file we will add config section for `microsoft.identity`

```
<!-- NEW -->
<configSections>
<section name="microsoft.identityModel"
 type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
</configSections>
<microsoft.identityModel>
<service>
<issuerNameRegistry
 type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<add name="CN=Atomia Identity" thumbprint="72071a7a2b933bd5b73bbb4b026c575ccb2d2ca4"/>
</trustedIssuers>
</issuerNameRegistry>
<securityTokenHandlers>
<remove type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
<add type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<samlSecurityTokenRequirement>
<nameClaimType value="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"/>
<roleClaimType value="http://schemas.troxo.com/ucp/2009/04/ucpcore/claims/groups"/>
</samlSecurityTokenRequirement>
</add>
</securityTokenHandlers>
</service>
</microsoft.identityModel>
```

In this config file few things are important to note:

- line 09 - We define with which issuer our WCF application has trusted relation.
- line 17 - Defines value of which token type will be interpreted as username
- line 18 - Defines value of which token type will be interpreted as users role

Now, we should configure our service to use federated authentication

```
<system.serviceModel>
<services>
<service behaviorConfiguration="CoreServiceBehavior"
name="Service.CoreService">
```

```
<endpoint address="" binding="wsFederationHttpBinding" contract="Service.ICoreService"
 bindingConfiguration="wsFed" >
</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
<host>
<baseAddresses>
<add baseAddress="http://localhost:8700/CoreService/" />
</baseAddresses>
</host>
</service>
</services>
<bindings>
<wsFederationHttpBinding>
<binding name="wsFed"  >
<security mode="Message">
<message issuedKeyType="SymmetricKey" issuedTokenType="">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
 isOptional="false"/>
<add claimType="http://schemas.troxo.com/ucp/2009/04/ucpcore/claims/groups"
 isOptional="true"/>
</claimTypeRequirements>
<issuer address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/username/" />
<issuerMetadata address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
</bindings>
<behaviors>
<serviceBehaviors>
<behavior name="CoreServiceBehavior">
<serviceMetadata httpGetEnabled="true"/>
<serviceDebug includeExceptionDetailInFaults="true"/>
<serviceAuthorization principalPermissionMode="None"/>
<serviceCredentials>
<issuedTokenAuthentication allowUntrustedRsaIssuers="false"
 certificateValidationMode="PeerOrChainTrust" audienceUriMode="Never"
 revocationMode="Online" trustedStoreLocation="LocalMachine">
<knownCertificates>
<add findValue="UCP Authorization Service" storeLocation="LocalMachine"
 storeName="TrustedPeople" x509FindType="FindBySubjectName"/>
</knownCertificates>
</issuedTokenAuthentication>
<serviceCertificate storeLocation="LocalMachine" storeName="My"
 x509FindType="FindBySubjectName" findValue="UCP Core">
</serviceCertificate>
</serviceCredentials>
</behavior>
</serviceBehaviors>
</behaviors>
</system.serviceModel>
```

- Line 06 - Defines that we use now wsFederationHttpBinding
- Lines 19 to 31 - Describes federated http bindings setting
  - Lines 23 to 26 - What claim types we are requesting from `Atomia Identity` for our WCF
  - Lines 27 and 28 - Defines location of identity provider for

On the STS side we should add RP (Rely party) certificate and inform STS that for given URI should use that certificate. This is done in two steps:

1. Set which RpCertProvider to use for given RP URI

```
<type type="IRpCertProvider" mapTo="AtomiaRpCertificateProvider" name="http://
localhost:8700/CoreService/"></type>
```

2. For AtomiaRpCertProvider set how to find certificate

```
<add storeLocation="LocalMachine" storeName="TrustedPeople" x509FindType="FindBySubjectName"
 findValue="CN=MyWebService" rpAddress="http://localhost:8700/CoreService/" />
```

Of course, you will have to install that certificate on given location.

4.3.2. Client configuration

If we now update service reference on client side Visual Studio will generate config file like:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.serviceModel>
<bindings>
<wsFederationHttpBinding>
<binding name="WSFederationHttpBinding_ICoreService" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
<message algorithmSuite="Default" issuedKeyType="SymmetricKey"
negotiateServiceCredential="true">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
isOptional="false" />
<add claimType="http://schemas.troxo.com/ucp/2009/04/ucpcore/claims/groups"
isOptional="true" />
</claimTypeRequirements>
<issuer address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/username/" />
<issuerMetadata address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
</bindings>
<client>
<endpoint address="http://localhost:8700/CoreService/" binding="wsFederationHttpBinding"
bindingConfiguration="WSFederationHttpBinding_ICoreService"
contract="CoreServiceNamespace.ICoreService" name="WSFederationHttpBinding_ICoreService">
<identity>
<dns value="localhost" />
</identity>
</endpoint>
</client>
</system.serviceModel>
</configuration>
```

If you look line 24 you will see that is set what is address of STS server but its not set how to authenticate to this service. We will need to update that line like:

```
<issuer address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/username/"
 binding="wsHttpBinding"
bindingConfiguration="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/username/">
<identity>
<certificate encodedValue="AwAAAAEAAAAUAAAAu0r+Dvx..." />
</identity>
</issuer>
```

Now we should add new bindingConfiguration:

```
<wsHttpBinding>
<binding name="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/username/"
closeTimeout="00:01:00" openTimeout="00:10:00" receiveTimeout="00:10:00"
sendTimeout="00:01:00" bypassProxyOnLocal="false" transactionFlow="false"
hostNameComparisonMode="StrongWildcard" maxBufferPoolSize="524288"
maxReceivedMessageSize="65536" messageEncoding="Text" textEncoding="utf-8"
useDefaultWebProxy="true" allowCookies="false">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
```

```
<message clientCredentialType="UserName" negotiateServiceCredential="true"
algorithmSuite="Default" establishSecurityContext="true" />
</security>
</binding>
</wsHttpBinding>
```

With this binding configuration we have set how the client will be authenticated to the STS.

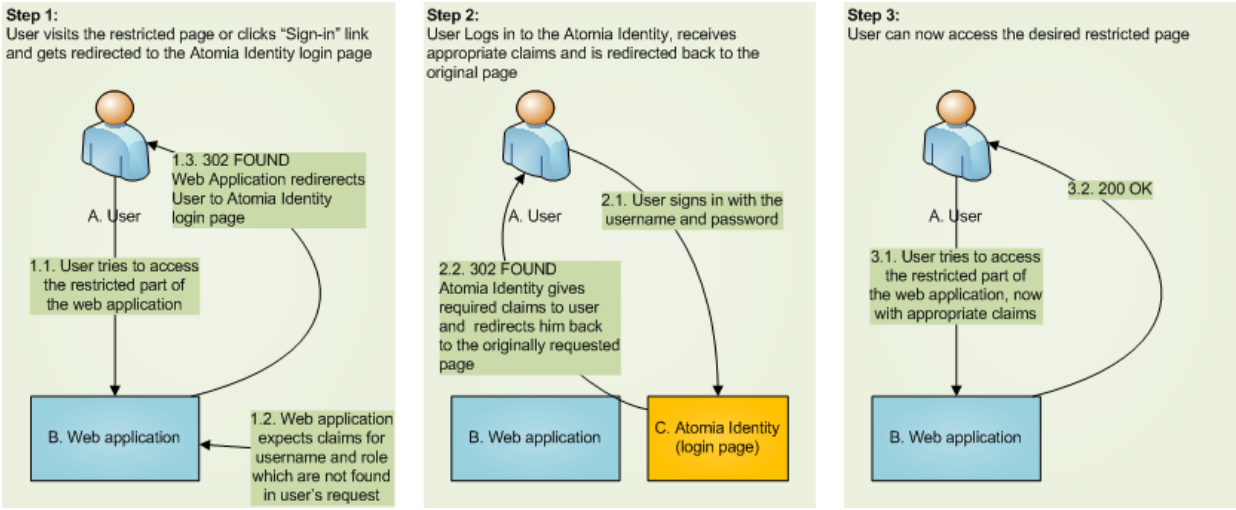
4.3.3. Certificates locations

If we assume that certificate for Atomia Identity is `AtomiaIdentity.cer` and certificates for RP are `WCFService.cer` and `WCFService.pfx` locations for them will be:

1. On STS Side:
   a. `WCFService.cer => LocalComputer|TrustedPeople`
2. On RP Side:
   a. `AtomiaIdentity.cer => LocalComputer|Trusted root certificate authorities`
   b. `AtomiaIdentity.cer => LocalComputer|Trusted people`
   c. `WCFService.pfx => LocalComputer|Personal`
3. On client side
   a. `WCFService.cer => LocalComputer|Trusted root certificate authorities`

4.4. Authenticating user to use web application to the web application using sign-in form with username and password

Sample source code



4.4.1. Atomia Identity STS Configuration

In order that STS could identify the user for the web application it should be hosted on the web server and provide some **login page** where your application will redirect for logging in. When you log in SAML token with claims has been created by the STS and you will be redirected back to your application. To make the service to be able to create a token it must contain the page that needs to check if the user need to be redirected to the login page and to translate SAML token(in our example that page will be titled as **PassiveStsEndPoint.aspx**).

Also in the web.config of the Atomia Identity (STS) we need to define relaying party application address:

```
<relyingPartyConfiguration>
<relyingParty>
...
```

```
<add storeLocation="LocalMachine" storeName="TrustedPeople" x509FindType="FindBySubjectName"
 findValue="CN=UCP Core" rpAddress="http://localhost/MvcIdentity"/>
</relyingParty>
</relyingPartyConfiguration>
```

For the Atomia Identity (STS) we must set which Certificate Provider to use for our web application URI:

```
<type type="IRpCertProvider" mapTo="AtomiaRpCertificateProvider" name="http://localhost/
MvcIdentity">
</type>
```

We should also set what claims types will need to provide Atomia Identity for our web application

In element with xpath `configuration\atomiaSTSConfig\passiveRpClaimRequests` should be added element that looks like:

```
<realm address="http://localhost:63340/">
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"/>
<add claimType="http://schemas.atomia.com/atomia/2009/04/identity/claims/groups"/>
</realm>
```

4.4.2. Web Application Configuration

- In order to reference and use the Atomia Identity (STS), one should have **Microsoft Geneva Framework** installed and referenced as a dll (Microsoft.IdentityModel.dll). You can download and install from this url - download Geneva Framework .
- In order to use Atomia Identity (STS) there are also some config sections should be implemented in web.config file.

```
<configSections>
...
<section name="microsoft.identityModel"
 type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
..
</configSections>
...
<microsoft.identityModel>
<service>
<issuerNameRegistry
 type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<add name="CN=UCP Authorization Service"
 thumbprint="bb4afe0efc6ab35a1006355c231e5b3a5d829625"/>
</trustedIssuers>
</issuerNameRegistry>
<audienceUris>
<add value="http://localhost/MvcIdentity"/>
</audienceUris>
<securityTokenHandlers>
<add type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
</add>
</securityTokenHandlers>
<federatedAuthentication>
<wsFederation requireHttps="false" passiveRedirectEnabled="true" issuer="http://
localhost:50680/AtomiaIdentityStS/PassiveStsEndpoint.aspx" realm="http://localhost/
MvcIdentity" ></wsFederation>
<cookieHandler requireSsl="false"/>
</federatedAuthentication>
<serviceCertificate>
<certificateReference x509FindType="FindBySubjectName" findValue="UCP Core"
 storeLocation="LocalMachine" storeName="My"/>
</serviceCertificate>
</service>
</microsoft.identityModel>
...
```

```
<system.web>
...
<authentication mode="None"/>
...
<httpModules>
...
<add name="SessionAuthenticationModule"
 type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel,
 Version=0.6.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
<add name="WSFederationAuthenticationModule"
 type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
...
</httpModules>
</system.web>
...
<system.webServer>
<modules>
<add name="SessionAuthenticationModule"
 type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.IdentityModel,
 Version=0.6.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
<add name="WSFederationAuthenticationModule"
 type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
 preCondition="managedHandler"/>
...
```

- First section is the **microsoft.identityModel**
  - in the configSections copy the section tag that adds the microsoft.identityModel configuration.
  - take the whole section **microsoft.identityModel** and copy to the web.config.
  - in **trustedIssuers** section you add the issuer you trust (name of the STS service and the thumbprint of its certificate).
  - in **audienceUris** set your url identification.
  - **federatedAuthentication/wsFederation** attribute **issuer** set the url of the STS' page that accepts authentication request and translates the SAML token into set of claims (in above - STS configuration section - we used to call that page **PassiveStsEndPoint.aspx**).
- **system.web** section
  - For the **authentication** tag set the attribute **mode** to *None* since we are using external STS authentication service.
  - The next step inserts 2 new HttpModules in the pipeline.
- In the **system.webServer/modules** need to add same two modules as in the previous httpModules section.
- Set the attribute **enabled** to **false** in the **<roleManager>** tag.

When all above is set the web application is ready to use the STS as an authentication provider. No other membership providers or authentication section need to be defined in the configuration file.

4.4.3. Using authorization data in the web application

The usage of the application after this configuration is quite easy.

1. Define pages those need the authorization (using web.config or using ClassAttributes [UCP:Authorize] before methods definitions in the MVC controllers classes).
   a. In ASP.NET web forms for the current directory in web.config you need to set

```
<authorization>
<deny users="?" />
</authorization>
```

b. In ASP.NET MVC, above the method definition in the controller class that do some action you need to put the attribute class **[UCP:Authorize]**

```
[Authorize]
public ActionResult Management()
{
return View();
}
```

2. In both cases above when the user tries to access pages those require to be authorized he will be redirected to the STS Login page.
3. When the web application is being authenticated we are able to read claims provided by the STS.
4. IClaimsIdentity contains property **Claims** represents the ClaimsCollection.
5. Every claim in collection is of type Claim which contains basic claim property as ClaimType and Value.

Therefore Relaying Party (RP) service by the given set of claims could authorize the user for some actions.

```
IClaimsIdentity claimsIdentity = Thread.CurrentPrincipal.Identity as IClaimsIdentity;
foreach (Claim claim in claimsIdentity.Claims)
{
Console.WriteLine(claim.ClaimType);
Console.WriteLine(claim.Value);
}
```

or if it is asp.net web form page

```
IClaimsIdentity ci = User.Identity as IClaimsIdentity;
foreach (var claim in ci.Claims)
{
Response.Write(string.Format("<div>Clam type: {0}; Claim value: {1}; Claim issuer: {2}</
div>",
claim.ClaimType, claim.Value, claim.Issuer));
}
```

4.5. Configuring STS to allow clients certificate based authentication

The STS should be aware of the web application's certificate via which the web application will authenticate to the STS.

Add the information about your web application's certificate to the `web.config` of the STS within the `microsoft.identityModel` section:

```
<issuerNameRegistry
 type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<add name="CN=Your Web Application Certificate Name"
 thumbprint="14f8d3701f0121e20a934baf140d712b4d83550d"/>
</trustedIssuers>
</issuerNameRegistry>
```

The same element should be added in element with xpath `configuration\microsoft.identityModel \service\securityTokenHandlers\securityTokenHandlerConfiguration\issuerNameRegistry \trustedIssuers`

Change the following properties in the above example:

XML Section	Property	Description	Example
issuerNameRegistry -> trustedIssuer	`name`	CN value of your web application's certificate	CN=Your Web Application Certificate Name

XML Section	Property	Description	Example
issuerNameRegistry -> trustedIssuer	`thumbprint`	Thumbprint of your web application's certificate	14f8d3701f0121e20a934baf140d712b4d8

To view details of your web application's certificate use Microsoft Management Console application (**mmc**).

There's one additional setting that needs to be added to the `web.config` within the `relyingParty` section :

```
<add storeLocation="LocalMachine" storeName="TrustedPeople" x509FindType="FindBySubjectName"
 findValue="CN=Your Web Application Certificate Name" rpAddress="Your Web Application's
 address"/>
```

For the Atomia Identity (STS) we must set which Certificate Provider to use for our web application URI:

```
<type type="IRpCertProvider" mapTo="AtomiaRpCertificateProvider" name="http://localhost/
MvcIdentity">
</type>
```

Change the following properties in the above example:

XML Section	Property	Description	Example
relyingParty -> add	`findValue`	CN value of your web application's certificate	CN=Your Web Application Certificate Name
relyingParty -> add	`rpAddress`	The address of your web application	http://localhost:52144/

4.6. Configuring WCF to allow clients certificate based authentication

4.6.1. Step 1

Add the reference to Microsoft.Identity.dll in the WCF application.

4.6.2. Step 2

Add the **microsoft.Identity** config section in the WCF application's web.config file:

```
<configSections>
....
<section name="microsoft.identityModel"
 type="Microsoft.IdentityModel.Configuration.MicrosoftIdentityModelSection,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
....
</configSections>
```

4.6.3. Step 3

Add a new section in the WCF application's web.config file within the `configuration` section:

```
<microsoft.identityModel>
<service>
<issuerNameRegistry
 type="Microsoft.IdentityModel.Tokens.ConfigurationBasedIssuerNameRegistry,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<trustedIssuers>
<add name="CN=UCP Authorization Service"
 thumbprint="72071a7a2b933bd5b73bbb4b026c575ccb2d2ca4"/>
</trustedIssuers>
```

```
</issuerNameRegistry>
<securityTokenHandlers>
<remove type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35"/>
<add type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
 Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral,
 PublicKeyToken=31bf3856ad364e35">
<samlSecurityTokenRequirement audienceUriMode="Never">
<nameClaimType value="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"/>
<roleClaimType value="http://schemas.troxo.com/ucp/2009/04/ucpcore/claims/groups"/>
</samlSecurityTokenRequirement>
</add>
</securityTokenHandlers>
</service>
</microsoft.identityModel>
```

Change the following properties in the above example:

XML Section	Property	Description	Example
issuerNameRegistry -> trustedIssuer	`name`	CN value of the trusted application that issued the token	CN=UCP Authorization Service
issuerNameRegistry -> trustedIssuer	`thumbprint`	Thumbprint of the certificate from the trusted application that issued the token	72071a7a2b933bd5b73bbb4b026c575cc
securityTokenHandlers	`type`	The class(with assembly info) that is handling the STS token	*Microsoft.IdentityModel.Tokens.Saml11.S Microsoft.IdentityModel, Version=0.6.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35*
securityTokenHandlers -> samlSecurityTokenRequirement	`nameClaimType`, `roleClaimTypes`	List of claims that are contained in the token	`<nameClaimType value="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"/> <roleClaimTypes> <add value="http://schemas.troxo.com/ucp/2009/04/ucpcore/claims/groups"/> </roleClaimTypes>`

4.6.4. Step 4

Add a section in the WCF application's web.config file within the `<system.serviceModel>` section:

```
<services>
<service behaviorConfiguration="UCPAuthPrototype.TestService.CoreServiceBehavior"
name="UCPAuthPrototype.TestService.CoreService">
<endpoint address="" binding="wsFederationHttpBinding"
 contract="UCPAuthPrototype.TestService.ICoreService"
 bindingConfiguration="STSBindingConfiguration">
</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
<host>
<baseAddresses>
<add baseAddress="http://localhost:8700/CoreService/" />
</baseAddresses>
</host>
</service>
```

```
</services>
<bindings>
<wsFederationHttpBinding>
<binding name="STSBindingConfiguration"  >
<security mode="Message">
<message issuedKeyType="SymmetricKey" issuedTokenType="">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
 isOptional="false"/>
<add claimType="http://schemas.microsoft.com/ws/2006/04/identity/claims/role"
 isOptional="true"/>
</claimTypeRequirements>
<issuer address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/cert/" />
<issuerMetadata address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
</bindings>
<behaviors>
<serviceBehaviors>
<behavior name="UCPAuthPrototype.TestService.CoreServiceBehavior">
<serviceMetadata httpGetEnabled="true" />
<serviceDebug includeExceptionDetailInFaults="true" />
<serviceAuthorization principalPermissionMode="None"/>
<serviceCredentials>
<issuedTokenAuthentication allowUntrustedRsaIssuers="false"
 certificateValidationMode="PeerTrust"  audienceUriMode="Never" revocationMode="Online"
 trustedStoreLocation="LocalMachine">
<knownCertificates>
<add findValue="UCP Authorization Service" storeLocation="LocalMachine"
 storeName="TrustedPeople" x509FindType="FindBySubjectName"/>
</knownCertificates>
</issuedTokenAuthentication>
<serviceCertificate storeLocation="LocalMachine" storeName="My"
 x509FindType="FindBySubjectName" findValue="UCP Core"/>
</serviceCredentials>
</behavior>
</serviceBehaviors>
</behaviors>
```

Let's explain the `service` , `bindings` and `behaviors` sections separately - let's start from the `bindings` section:

```
<bindings>
<wsFederationHttpBinding>
<binding name="STSBindingConfiguration"  >
<security mode="Message">
<message issuedKeyType="SymmetricKey" issuedTokenType="">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
 isOptional="false"/>
<add claimType="http://schemas.microsoft.com/ws/2006/04/identity/claims/role"
 isOptional="true"/>
</claimTypeRequirements>
<issuer address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/cert/" />
<issuerMetadata address="http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
</bindings>
```

The binding defines the way of communication between the WCF service and the Web application that wants to use the WCF service. This element holds a collection of standard and custom bindings which are is identified by their name. So, the communication will be done through wsFederationHttpBinding - a binding that supports WS-Federation.

These are the settings that can be changed in the above section:

XML Section	Property	Description	Example
wsFederationHttpBinding -> binding	`name`	Binding identifier	STSBindingConfiguration

XML Section	Property	Description	Example
message	{{claimTypeRequirements}}	The list of claims the web application needs to provide in order to use the WCF service	`<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name" isOptional="false"/><add claimType="http://schemas.microsoft.com/ws/2006/04/identity/claims/role" isOptional="true"/>`
message -> issuer	`address`	The address of the STS where the web application should obtain the claims from	http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/cert/
message -> issuerMetadata	`address`	The address of the STS's metadata	http://localhost:50680/AtomiaIdentityStS/AtomiaSts.svc/mex

The `behavior` section defines behavior elements consumed by services. Each behavior element is identified by its unique name attribute.

```
<behaviors>
<serviceBehaviors>
<behavior name="UCPAuthPrototype.TestService.CoreServiceBehavior">
<serviceMetadata httpGetEnabled="true" />
<serviceDebug includeExceptionDetailInFaults="true" />
<serviceAuthorization principalPermissionMode="None"/>
<serviceCredentials>
<issuedTokenAuthentication allowUntrustedRsaIssuers="false"
 certificateValidationMode="PeerTrust"  audienceUriMode="Never" revocationMode="Online"
 trustedStoreLocation="LocalMachine">
<knownCertificates>
<add findValue="UCP Authorization Service" storeLocation="LocalMachine"
 storeName="TrustedPeople" x509FindType="FindBySubjectName"/>
</knownCertificates>
</issuedTokenAuthentication>
<serviceCertificate storeLocation="LocalMachine" storeName="My"
 x509FindType="FindBySubjectName" findValue="UCP Core"/>
</serviceCredentials>
</behavior>
</serviceBehaviors>
</behaviors>
```

We can customize these settings:

XML Section	Property	Description	Example
serviceBehaviors -> behavior	`name`	Behavior identifier	UCPAuthPrototype.TestService.CoreServ
issuedTokenAuthentication -> knownCertificates	`findValue`	A string in the X.509 certificate store that contains the certificate used by the STS for signing and encrypting the tokens issued to web application (so	UCP Authorization Service

XML Section	Property	Description	Example
		the WCF service can authenticate the web application)	
serviceCertificate	`findValue`	A string in the X.509 certificate store that contains the certificate used for signing and encrypting messages from a web application to the WCF service	UCP Core

Finally, the `service` section contains the settings for a Windows Communication Foundation (WCF) service. It also contains endpoints that expose the service.

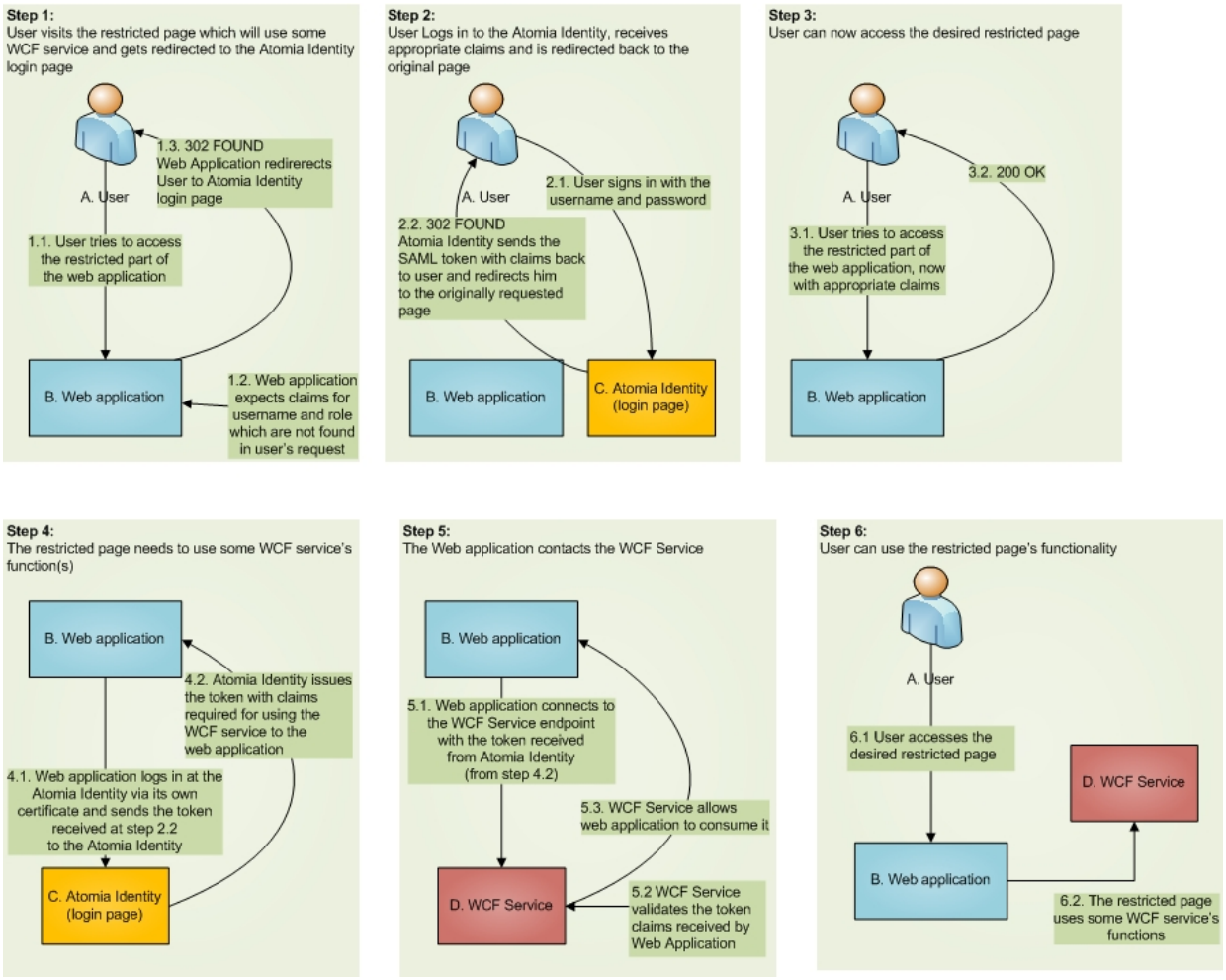
```
<service behaviorConfiguration="UCPAuthPrototype.TestService.CoreServiceBehavior"
name="UCPAuthPrototype.TestService.CoreService">
<endpoint address="" binding="wsFederationHttpBinding"
 contract="UCPAuthPrototype.TestService.ICoreService"
 bindingConfiguration="STSBindingConfiguration">
</endpoint>
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
<host>
<baseAddresses>
<add baseAddress="http://localhost:8700/CoreService/" />
</baseAddresses>
</host>
</service>
```

These settings need to be customized so the service will use the binding ad behavior defined above:

XML Section	Property	Description	Example
service	`name`	Specifies the type of the service to be instantiated. The format should be Namespace.Class.	UCPAuthPrototype.TestService.CoreServ
service	`behaviorConfiguration`	A string that contains the behavior name of the behavior to be used to instantiate the service	UCPAuthPrototype.TestService.CoreServ
host	`baseAddress`	A string that specifies a base address used by the service host.	http://localhost:8700/CoreService/
endpoint	`binding`	Specifies the type of binding to use.	wsFederationHttpBinding
endpoint	`bindingConfiguration`	A string that specifies the binding name of the binding to use when the endpoint is instantiated	STSBindingConfiguration

4.7. Identity delegation

Source code



This part is for developers of web applications who want to use WCF services with Atomia Identity authorization. It will show you how a web application can issue calls to WCF service with the privileges of the user who authenticated to that web application. This mechanism is called *Identity Delegation* . In the Identity Delegation mechanism, web application is able to use WCF by providing the appropriate SAML token to the WCF service.

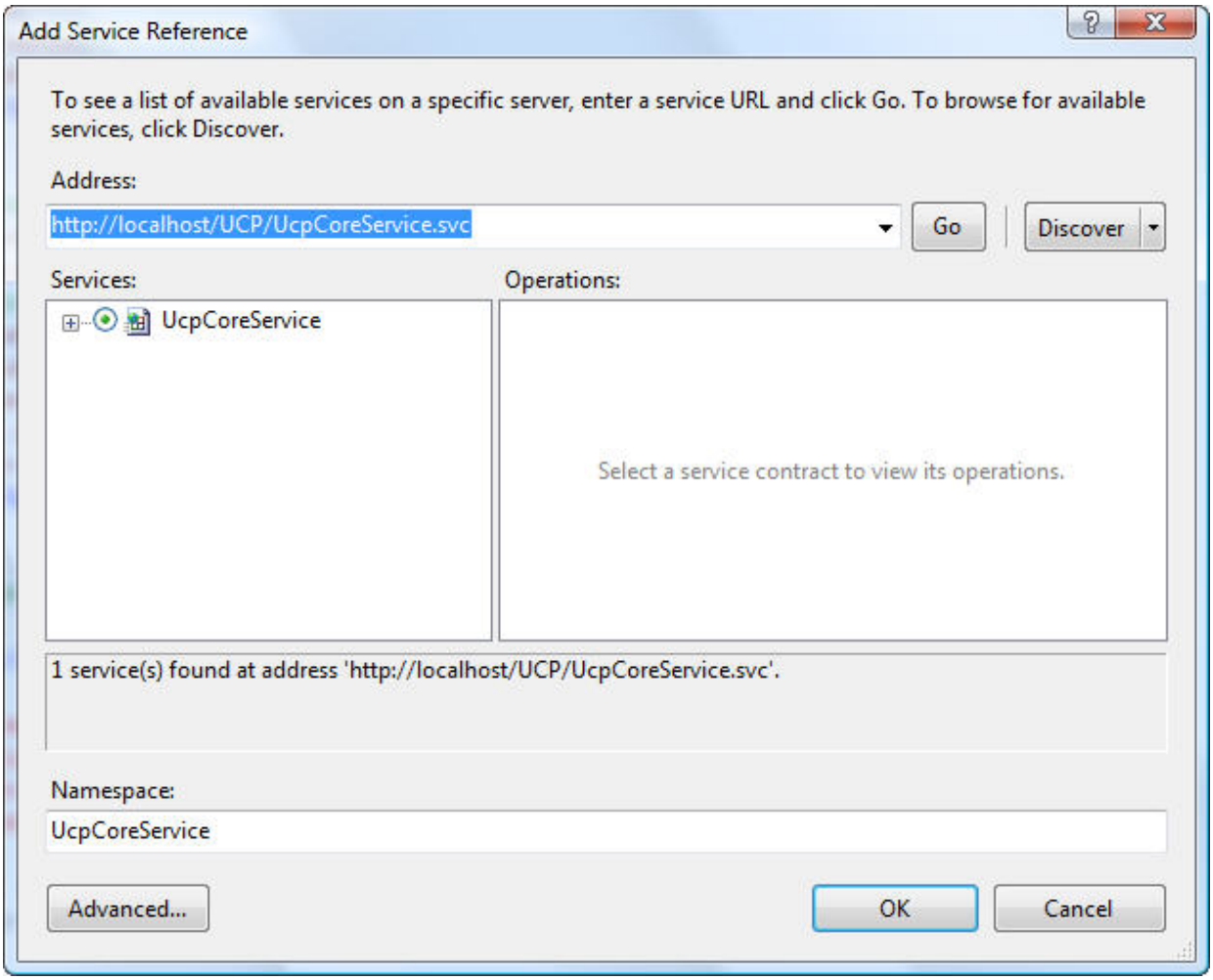
This token should contain:

- identity claims and
- delegation information

4.7.1. Web application configuration for identity delegation

4.7.1.1. Step 1

Add a service reference to the Web application



After adding a service reference to the web application, the `web.config` file is automatically updated - we have a new section within `configuration` :

```
<system.serviceModel>
<bindings>
<wsFederationHttpBinding>
<binding name="WSFederationHttpBinding_IUcpCoreService" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
textEncoding="utf-8" useDefaultWebProxy="true">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
<message algorithmSuite="Default" issuedKeyType="SymmetricKey"
negotiateServiceCredential="true">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
isOptional="false" />
<add claimType="http://schemas.microsoft.com/ws/2006/04/identity/claims/role"
isOptional="true" />
</claimTypeRequirements>
<issuer address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/username/" />
<issuerMetadata address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
</bindings>
<client>
<endpoint address="http://t098/UCP/UcpCoreService.svc" binding="wsFederationHttpBinding"
bindingConfiguration="WSFederationHttpBinding_IUcpCoreService"
contract="UcpCoreService.IUcpCoreService" name="WSFederationHttpBinding_IUcpCoreService">
<identity>
```

```
<certificate encodedValue="AwAAAAEAAAAUAAAAkI...." />
</identity>
</endpoint>
</client>
</system.serviceModel>
```

4.7.1.2. Step 2

We need to adjust Web Application's `web.config` file in order to use Token delegation to the WCF service. First, we need to change the `issuer` section within the *security->message* tag in our `binding` section - our web application will be authenticated on the STS via certificate instead of username/password combination. So, instead of:

```
<issuer address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/username/" />
```

we should have this kind of section:

```
<issuer address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/cert/"
 binding="wsHttpBinding" bindingConfiguration="http://localhost/AtomiaIdentityStS/
AtomiaSts.svc/cert/">
<identity>
<certificate encodedValue="MIIByzCCATSgAwIBAgIQ2PtXByKML5dI68y5...." />
</identity>
</issuer>
```

XML Section	Property	Description	Example
identity-> certificate	`encodedValue`	The certificate (public key) web application is using to sign and encrypt messages to STS (can be found at STS's wsdl)	MIIByzCCATSgAwIBAgIQ2PtXByKML5d

Since we're specifying the way of communication between the web application and the STS via the `binding` and `bindingConfiguration` attributes, we need define this binding within the `bindings` section :

```
<wsHttpBinding>
<binding name="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/cert/"
 closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
allowCookies="false">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
<message clientCredentialType="Certificate" />
</security>
</binding>
</wsHttpBinding>
```

where the part

```
<security mode="Message">
<message clientCredentialType="Certificate" />
</security>
```

defines that the web application will be authenticated at the STS via its own certificate.

4.7.1.3. Step 3

The next thing we need to do is to define where web application's certificate can be found. Let's add *behaviorConfiguration="ClientCertificateBehavior"* attribute to the **<endpoint>** tag of our WCF service:

```
<endpoint address="http://t098/UCP/UcpCoreService.svc"
 behaviorConfiguration="ClientCertificateBehavior"
binding="wsFederationHttpBinding"
 bindingConfiguration="WSFederationHttpBinding_IUcpCoreService"
contract="UcpCoreService.IUcpCoreService" name="WSFederationHttpBinding_IUcpCoreService">
<identity>
<certificate encodedValue="AwAAAAEAAAAUAAAAkI0urWJG...." />
</identity>
</endpoint>
```

and a `behaviors` section above the `client` section :

```
<behaviors>
<endpointBehaviors>
<behavior name="ClientCertificateBehavior">
<clientCredentials>
<clientCertificate findValue="My web application" storeLocation="LocalMachine"
storeName="My" x509FindType="FindBySubjectName" />
</clientCredentials>
</behavior>
</endpointBehaviors>
</behaviors>
```

where the following setting needs to be adjusted

XML Section	Property	Description	Example
clientCertificate	`findValue`	A string in the X.509 certificate store that contains the certificate used for authenticating web application at the STS	My web application

4.7.1.4. Step 4

The last thing we need to do is to modify the `serviceCertificate` within `microsoft.identityModel` section in `web.config` file :

```
<serviceCertificate>
<certificateReference x509FindType="FindBySubjectName" findValue="Your Application Name"
 storeLocation="LocalMachine" storeName="My"/>
</serviceCertificate>
```

Change the following properties :

XML Section	Property	Description	Example
serviceCertificate-> certificateReference	`findValue`	A string in the X.509 certificate store that contains the certificate STS for encrypting the issued tokens for the web application	Your Application Name

4.7.1.5. Step 5

This step is just an overview what the `web.config` *system.serviceModel* section should look like:

```
<system.serviceModel>
<bindings>
<wsFederationHttpBinding>
<binding name="WSFederationHttpBinding_IUcpCoreService" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
textEncoding="utf-8" useDefaultWebProxy="true">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
```

```
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
<message algorithmSuite="Default" issuedKeyType="SymmetricKey"
negotiateServiceCredential="true">
<claimTypeRequirements>
<add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
isOptional="false" />
<add claimType="http://schemas.microsoft.com/ws/2006/04/identity/claims/role"
isOptional="true" />
</claimTypeRequirements>
<issuer address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/cert/"
 binding="wsHttpBinding" bindingConfiguration="http://localhost/AtomiaIdentityStS/
AtomiaSts.svc/cert/">
<identity>
<certificate encodedValue="MIIByzCCATSgAwIBAgIQ2PtXByKML5dI68y5ADms
+DANBgkqhkiG9w0BAQUFADAkMSIwIAYDVQQDExlVQ1AgQXV0aG9yaXphdGlvbiBTZXJ2aWNlMB4XDTA5MDYwMTExMDEyOFoXDTM5MDYw
+edt1kwnjNE0JU0Yq4n2WHz/
NK3O5a4h6V7kfG3D1iixvPEqzgbCaNw8OpM6+lLEMFoFvTRIMhSr3vM7Y9AxaoaXCY8oQ9oJgnK6XEnuh4zh234jAFokkvkiUx1t/
E/J6vAgGmTdm84SrAK
+0Lc7pONW8qnNnJzv18EglVkrcCAwEAATANBgkqhkiG9w0BAQUFAAOBgQAYlziJ46RFEWtarXbZlscFAOwyu8VQaafKrNLTMDfvv9i1r
+NCNEDGr1OXcyeuldd3ckTZRzc9Up7jIi1kQjDnfe3A51IRGYc2dX9WwWE7Q==" />
</identity>
</issuer>
<issuerMetadata address="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/mex" />
</message>
</security>
</binding>
</wsFederationHttpBinding>
<wsHttpBinding>
<binding name="http://localhost/AtomiaIdentityStS/AtomiaSts.svc/cert/"
 closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode="StrongWildcard"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
allowCookies="false">
<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
<reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
<security mode="Message">
<message clientCredentialType="Certificate" />
</security>
</binding>
</wsHttpBinding>
</bindings>
<behaviors>
<endpointBehaviors>
<behavior name="ClientCertificateBehavior">
<clientCredentials>
<clientCertificate findValue="Atomia Web Frame" storeLocation="LocalMachine" storeName="My"
 x509FindType="FindBySubjectName" />
</clientCredentials>
</behavior>
</endpointBehaviors>
</behaviors>
<client>
<endpoint address="http://t098/UCP/UcpCoreService.svc" binding="wsFederationHttpBinding"
bindingConfiguration="WSFederationHttpBinding_IUcpCoreService"
 behaviorConfiguration="ClientCertificateBehavior"
contract="UcpCoreService.IUcpCoreService" name="WSFederationHttpBinding_IUcpCoreService">
<identity>
<certificate
 encodedValue="AwAAAAEAAAAUAAAAkI0urWJGQQsCDdWMWI4QlfxNP6kgAAAAAQAAAK0BAAAwggGpMIIBEqADAgECAhApKYqxXRgmc
Iq7Av7vNVU9GSAMhZ2qQqVAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAMJFJWpNirhnHhzws7O80V3dvV0XHWuhJIeGg8Ds35MUNO4y4K2
l9QtzH8De3+TECarAOabaD3iz8K+aBqgem1VfkIB+bWgckRYkdwimCpGuxKqSDO9xEOcSaQitUphmj63as/
ddi7ccUTlHAGwtLMRJcD4iHiA144w3sD+Ms=" />
</identity>
</endpoint>
</client>
</system.serviceModel>
```

4.7.2. Web application code changes for identity delegation

There are few things need to be done in the code in order to use the Identity delegation.

- Open the Global.asax.cs of your web application

```
public static readonly string CachedChannelFactory = "WFE_CachedChannelFactory";
void SessionAuthenticationModule_ConfigurationLoaded(object sender, EventArgs e)
{
ChannelFactory<UCPAuthPrototype.WebMvcApplication.CoreService.ICoreService> service2CF =
 new
 ChannelFactory<UCPAuthPrototype.WebMvcApplication.CoreService.ICoreService>("WSFederationHttpBinding_
FederatedClientCredentials.ConfigureChannelFactory<UCPAuthPrototype.WebMvcApplication.CoreService.ICor
Application[CachedChannelFactory] = service2CF;
}
```

  - copy the code above
    - here we initialize the ChannelFactory instance for the service we will use *UCPAuthPrototype.WebMvcApplication.CoreService.ICoreService* - here you provide the proxy class of your service.
    - put that instance in the global Application dictionary.

- Before the you call of the method of some service you need to initialize it first:

```
// Get the caller's token from custom state
SessionSecurityToken sessionToken = null;
if
 (System.Web.HttpContext.Current.Items.Contains(typeof(SessionSecurityToken).AssemblyQualifiedName))
{
sessionToken =
 System.Web.HttpContext.Current.Items[typeof(SessionSecurityToken).AssemblyQualifiedName]
 as SessionSecurityToken;
}
SecurityToken callerToken = null;
// We expect only one token to be specified during Bootstrap.
if ((sessionToken != null) && (sessionToken.BootstrapTokens.Count == 1))
{
callerToken = sessionToken.BootstrapTokens[0];
}
if (callerToken == null)
{
// We lost the session state but the user still has the federated ticket
// Let's sign the user off and start again
FederatedAuthentication.SignOut();
return LogOff();
}
// Get the channel factory to the backend service from the application state
ChannelFactory<ICoreService> factory =
 (ChannelFactory<ICoreService>)System.Web.HttpContext.Current.Application[MvcApplication.CachedChannel
// Create and setup channel to talk to the backend service
ICoreService channel;
lock (factory)
{
// Setup the ActAs to point to the caller's token so that we perform a delegated call to
 the backend service
// on behalf of the original caller.
channel = factory.CreateChannelActingAs<ICoreService>(callerToken);
//factory.Credentials.IssuedToken = callerToken;
//channel = factory.CreateChannel();
}
```

  - first need to get **callerToken** (it represents the SAML token of the currently authenticated user)
  - then in the line: take the factory instance - initialized in the previous code section in the **Global.asax.cs**
  - use the factory instance to create the Channel for your service

```
channel = factory.CreateChannelActingAs<ICoreService>(callerToken);
```

    - Change *ICoreService* to the interface of your service

  - you can call your service method

```
var UcpAccounts = channel.ListAccounts();
```

4.7.3. Atomia Identity update to support authentication of web application using certificate

Web.config file of Atomia Identity update with:

1. On element with xpath `configuration\microsoft.identityModel\service\issuerNameRegistry\trustedIssuers` add line like:

```
<add name="CN=MyWebApplication" thumbprint="1C18704DF16069DCDD90CE6C6D2FFDE3002E2929" />
```

2. On element with xpath `configuration\microsoft.identityModel\service\securityTokenHandlers\securityTokenHandlerConfiguration\issuerNameRegistry\trustedIssuers` add line like:

```
<add name="CN=MyWebApplication" thumbprint="1C18704DF16069DCDD90CE6C6D2FFDE3002E2929" />
```

5. Atomia Identity - Updates

This page should contain instructions on how to update the service.

6. Atomia Identity - API Reference

6.1. AddRole

Add specified role.

```
public void AddRole(
string roleName
)
```

Parameters:

- `roleName` - Name of the role to add.

6.2. AddUser

Adds a new Atomia user to the data source.

```
public void AddUser(
AtomiaUser user,
string password);
```

Parameters:

- `user` - Atomia user.
- `password` - User's password.

6.3. AddUsersToRoles

Adds the users to roles.

```
public void AddUsersToRoles(
string[] usernames,
string[] roles
)
```

Parameters:

- `usernames` - The usernames.

- `roles` - The roles.

6.4. DeleteRole

Delete specified role.

```
public void DeleteRole(
string roleName
)
```

Parameters:

- `roleName` - Name of the role to delete.

6.5. FindByProperty

Finds Atomia user by certain property.

```
public AtomiaUser[] FindByProperty(
Dictionary<string, string> atomiaProperty,
bool fillUserProperties
)
```

Parameters:

- `atomiaProperty` - property to search for.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: Array of Atomia users which has property as the given one.

6.6. FindUsersByEmail

Returns a list of membership users where the email matches the supplied `emailPattern` .

```
public AtomiaUser[] FindUsersByEmail(
string emailPattern,
int pageIndex,
int pageSize,
out int totalRecords,
bool fillUserProperties
)
```

Parameters:

- `emailPattern` - The user email pattern to search for. If your data source supports additional search capabilities, such as wildcard characters, you can provide more extensive search capabilities for e-mail addresses.
- `pageIndex` - The zero-based index of the page of results to return.
- `pageSize` - The size of the page of results to return.
- `totalRecords` - The total number of matched users.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: Collection that contains a page of `pageSize` AtomiaUser objects beginning at the page specified by `pageIndex` .

6.7. FindUsersByName

Returns a list of membership users where the user name matches the supplied usernamePattern.

```
public AtomiaUser[] FindUsersByName(
string usernamePattern,
int pageIndex,
int pageSize,
out int totalRecords,
bool fillUserProperties);
```

Parameters:

- `usernamePattern` - The user name to search for. If your data source supports additional search capabilities, such as wildcard characters, you can provide more extensive search capabilities for user names.
- `pageIndex` - The zero-based index of the page of results to return.
- `pageSize` - The size of the page of results to return.
- `totalRecords` - The total number of matched users.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: Collection that contains a page of `pageSize` AtomiaUsers objects beginning at the page specified by `pageIndex` .

6.8. FindUsersInRole

Returns a list of users in a role where the user name contains a match of the supplied username pattern.

```
public AtomiaUser[] FindUsersInRole(
string roleName,
string usernamePattern,
bool fillUserProperties
)
```

Parameters:

- `roleName` - The name of the role.
- `usernamePattern` - The user name to search for. Wildcard support is included based on the data source.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: List of users in a role where the user name matches `usernamePattern` .

6.9. GetAllRoles

Gets all roles.

```
public string[] GetAllRoles()
```

Return value: A string array containing the names of all the roles stored.

6.10. GetAllUsers

Returns an array of membership users.

```
public AtomiaUser[] GetAllUsers(
int pageIndex,
int pageSize,
out int totalRecords,
bool fillUserProperties
)
```

Parameters:

- `pageIndex` - The index of the page of results to return. `pageIndex` is zero-based.
- `pageSize` - The size of the page of results to return.
- `totalRecords` - The total number of matched users.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: Collection that contains a page of `pageSize` AtomiaUser objects beginning at the page specified by `pageIndex` .

6.11. GetRolesForUser

Gets a list of the roles that a specified user is.

```
public string[] GetRolesForUser(
string username
)
```

Parameters:

- `username` - The username of the user that belongs to specific roles.

Return value: List with the roles.

6.12. GetUser

Gets information from the data source for a membership user.

```
public AtomiaUser GetUser(
string username,
bool fillUserProperties);
```

Parameters:

- `username` - The name of the user to return.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: Atomia user with user properties.

6.13. GetUserNameByEmail

Returns a list of membership users where the user name matches the supplied username pattern.

```
public string GetUserNameByEmail(
string email);
```

Parameters:

- `email` - The email address to search for.

Return value: The name of the user.

6.14. GetUsersInRole

Returns all user that belong to specific role.

```
public AtomiaUser[] GetUsersInRole(
string roleName,
bool fillUserProperties
)
```

Parameters:

- `roleName` - The role of the user.
- `fillUserProperties` - Indicator to point whether return array of Atomia users will be filled with Atomia user properties.

Return value: List with the names of all the users who are members of the specified role.

6.15. ModifyAtomiaUser

Modifies Atomia user properties.

```
public void ModifyAtomiaUser(
AtomiaUser user
)
```

Parameters:

- `user` - Atomia user.

6.16. ModifyPassword

Updates password for the user in the data source.

```
public void ModifyPassword(
string username,
string oldPassword,
string newPassword);
```

Parameters:

- `username` - Username of the user.
- `oldPassword` - Current password for the specified user.
- `newPassword` - New password.

6.17. RemoveUser

Deletes user.

```
public void RemoveUser(
string username
)
```

Parameters:

- `username` - The username of the user to delete.

6.18. RemoveUserFromRole

Removes user from specific role.

```
public void RemoveUserFromRole(
string[] usernames,
string[] roles
)
```

Parameters:

- `usernames` - A string array of user names to be removed from the specified roles.
- `roles` - A string array of role names to remove the specified user names from.

6.19. ValidateUser

Verifies that the specified user name and password exist in the data source.

```
public bool ValidateUser(
string username,
string password);
```

Parameters:

- `username` - Name of the user to validate.
- `password` - Password for the specified user.

Return value: True if username and password are valid, otherwise returns false.

7. Atomia Identity - Identity SDK

This page should contain information about Identity SDK.

8. Atomia Identity - Usage

8.1. Usage

This section should contain instructions on how to use Identity service.

8.2. Code examples

8.2.1. Code example 1

This code example does...

```
//
// code example
//
```

9. Atomia Identity - Technical information

This page should contain technical information about service.

10. Atomia Identity - FAQ

This page contains answers to the frequently asked questions.

11. Atomia Identity - Release log

This page will be written upon a first release of Atomia Identity.

12. Atomia Identity - Roadmap

This page will be written upon a first release of Atomia Identity.

13. Atomia Identity - License

Product	Component	Type of License	Should the license be distributed with our application?	Where is it located in our distribution?
Atomia Identity	Log4net	Apache License, Version 2.0	Yes	`%INSTALLDIR%\bin\log4net.license.txt`
Atomia Identity	Castle.Core	Apache License, Version 2.0	Yes	`%INSTALLDIR%\bin\Castle.DynamicProxy.license.txt`
Atomia Identity	Castle.DynamicProxy2	Apache License, Version 2.0	Yes	`%INSTALLDIR%\bin\Castle.DynamicProxy.license.txt`
Atomia Identity	Iesi.Collections	Apache License, Version 2.0	Yes	`%INSTALLDIR%\bin\Iesi.Collections.license.txt`
Atomia Identity	NHibernate	Apache License, Version 2.0	Yes	`%INSTALLDIR%\bin\log4net.license.txt`
Atomia Identity	Microsoft Enterprise Library	Microsoft Public License (Ms-PL)	No	-
Atomia Cloud Hosting Package - LiteSpeed agent	Zend Framework	New BSD License	Yes	`%INSTALLDIR%\libs\zend.license.txt`

Index