Sommaire

[Terminologies 4](#_Toc461019094)

[Benchmark 5](#_Toc461019095)

[Architecture 6](#_Toc461019096)

[Use cases of the website 8](#_Toc461019097)

[Protect resources 9](#_Toc461019098)

[Action: Move resources to sub directory 11](#_Toc461019099)

[Action: Remove one or more resources 11](#_Toc461019100)

[Action: Rename a resource 11](#_Toc461019101)

[Action: Move resources 11](#_Toc461019102)

[Action: Duplicate one or more resources 11](#_Toc461019103)

[Action: How to access to a resource 12](#_Toc461019104)

[Action: Create a file 12](#_Toc461019105)

[Action: Create a folder 12](#_Toc461019106)

[Action: Upload one or more resources 12](#_Toc461019107)

[Action: Refresh current working directory 12](#_Toc461019108)

[Action: Display keyboard shortcuts 13](#_Toc461019109)

[Action: Change visualization 13](#_Toc461019110)

[Action  Search resources in the current working directory 13](#_Toc461019111)

[Action: Navigate to the previous / next folder 13](#_Toc461019112)

[Action: Add / Edit authorization rules 13](#_Toc461019113)

[Action : Select all resources 16](#_Toc461019114)

[Edit UMA & OPENID assets 17](#_Toc461019115)

[Action : Add a client 18](#_Toc461019116)

[Action : Delete one or more clients 18](#_Toc461019117)

[Action : Edit a client 18](#_Toc461019118)

[Action : Delete one or more scopes 20](#_Toc461019119)

[Action : Display scope information 20](#_Toc461019120)

[Action : Delete one or more resource owners 20](#_Toc461019121)

[Action : Edit a resource owner 20](#_Toc461019122)

[Action: Delete one or more authorization policies 22](#_Toc461019123)

[Action: Display authorization policy 22](#_Toc461019124)

[Action: Delete one or more UMA resources 22](#_Toc461019125)

[Action: Display UMA resource 22](#_Toc461019126)

[Identity providers 24](#_Toc461019127)

[Action: Display providers 24](#_Toc461019128)

[Action  Enable / disable a provider 25](#_Toc461019129)

[Action: Add a provider 25](#_Toc461019130)

[Action: Edit provider 26](#_Toc461019131)

[Parameters 27](#_Toc461019132)

[Use cases of the Visual Studio extension 28](#_Toc461019133)

[Configure parameters 28](#_Toc461019134)

[Generate one or more resources 28](#_Toc461019135)

[Refresh the resources 29](#_Toc461019136)

[Protect one or more resources 29](#_Toc461019137)

[Generate security proxy 30](#_Toc461019138)

[Action : Search one or more resources 30](#_Toc461019139)

[Action : Add a security proxy 31](#_Toc461019140)

[Scenarios 32](#_Toc461019141)

[First scenario: An heavy application wants to access to a protected API 32](#_Toc461019142)

[Identify and classify identities 33](#_Toc461019143)

[Add a client 34](#_Toc461019144)

[Add a resource 37](#_Toc461019145)

[Add authorization policy 37](#_Toc461019146)

[Assign marketing role to the resource owner 38](#_Toc461019147)

[Develop 39](#_Toc461019148)

[Second scenario : an API want to access to a resource 42](#_Toc461019149)

[Identify and classify identities 43](#_Toc461019150)

[Add a client 43](#_Toc461019151)

[Add a resource 44](#_Toc461019152)

[Add authorization policy 44](#_Toc461019153)

[Develop 45](#_Toc461019154)

[Third scenario: Limit access to certain website features 47](#_Toc461019155)

[Identify and classify identities 48](#_Toc461019156)

[Add a client 48](#_Toc461019157)

[Add a resource 49](#_Toc461019158)

[Add an authorization policy 49](#_Toc461019159)

[Add administrator role to the resource owner 50](#_Toc461019160)

[Develop 50](#_Toc461019161)

[Installation 52](#_Toc461019162)

[Mandatories steps 52](#_Toc461019163)

[Install GIT 52](#_Toc461019164)

[Install the certificate 52](#_Toc461019165)

[Manual installation 53](#_Toc461019166)

[Install with Docker 53](#_Toc461019167)

[Prequisistes 53](#_Toc461019168)

[Launch the deployment 54](#_Toc461019169)

[Technical documentation 55](#_Toc461019170)

[Monitoring 55](#_Toc461019171)

[Nuget packages 55](#_Toc461019172)

[SimpleIdentityServer.Proxy 55](#_Toc461019173)

[SimpleIdentityServer.UmaIntrospection.Authentication 55](#_Toc461019174)

[SimpleIdentityServer.Uma.Authorization 55](#_Toc461019175)

[Load balancing 56](#_Toc461019176)

# Terminologies

|  |  |
| --- | --- |
| Words | Definitions |
| Resource | It can be anything that needs to be protected for example : a picture of a user or an API operation. |
| Folder | Resource which can contains one or more resources. |
| File | Unique resource. |
| Authorization policy | Can contains one or more security rules and is assigned to one or several resources. It is used by the UMA server to determine if an incoming request can execute the requesting operations (read, write or delete) on a protected resource. |
| Security rule | Belongs to an authorization policy. |
| Resource owner | An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user. |
| Scope | List of resources which can be accessed by a client. |
| Client | An application making protected resource requests on behalf of the resource owner and with its authorization. The term “client” does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices) |
| UMA server | Server which is conformed to the OPENID-RFC [[1]](#footnote-1) |
| OPENID server | Server which is conformed to the UMA-RFC [[2]](#footnote-2) |
| Requesting Party Token (RPT) | An UMA access token associated with a set of authorization data, used by the client to gain access to protected resources at the UMA server. |

# Benchmark

The following table lists the differences between our product and others: Lokit, Identity Server, Gluu server and AUTH0. It has been made in “29-08-2016”, if you noticed some differences don’t hesitate to contact-us by email.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Lokit** | **Identity Server** | **Gluu server** | **AUTH0** |
| Authors | Habart Thierry | Brock Allen & Dominick Baier | Gluu | Auth0 |
| Start date | October 2015 | January 2014 | March 2014 | November 2012 |
| Workflow Oauth2.0 | | | | |
| Client credentials | OK | OK | OK | OK |
| Password | OK | OK | OK | OK |
| Refresh token | OK | OK | OK | OK |
| Workflow OpenId | | | | |
| Implicit | OK | OK | OK | OK |
| Hybrid | OK | OK | OK | NOK |
| Other OPENID features | | | | |
| Register a client ([RFC](https://openid.net/specs/openid-connect-registration-1_0.html)) | OK | NOK | OK | NOK |
| Sign token ([JWS](https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41)) | OK | OK | OK | OK |
| Encrypt token ([JWE](https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-40)) | OK | NOK | NOK | NOK |
| Invalidate session | OK | OK | OK | NOK |
| Client authentication methods ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#ClientAuthentication)) | | | | |
| client\_secret\_basic | OK | OK | OK | OK |
| client\_secret\_post | OK | OK | OK | OK |
| client\_secret\_jwt | OK | NOK | NOK | NOK |
| private\_key\_jwt | OK | NOK | NOK | NOK |
| none | OK | NOK | NOK | NOK |
| Response modes | | | | |
| Query | OK | OK | OK | OK |
| Fragment | OK | OK | OK | OK |
| Form\_post | OK | OK | OK | OK |
| Other parameters | | | | |
| claims ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#ClaimsParameter)) | OK | OK | OK | NOK |
| request ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#RequestObject)) | OK | OK | OK | NOK |
| Quality | | | | |
| Code coverage | 84% | Unknown | Unknown | Unknown |
| Nombre UTs | 633 | Unknown | Unknown | Unknown |
| UMA ([RFC](http://openid.net/specs/openid-heart-uma-2015-12-09.html)) | | | |  |
| UMA supported | OK | NOK | OK | NOK |
| UI | | | | |
| UI exists | OK | OK | OK | OK |
| CRUD opened assets | OK | OK | OK | OK |
| CRUD uma assets | OK | NOK | OK | NOK |
| Resource organized by urls | OK | NOK | NOK | NOK |
| Enable or disable external identity providers | OK | NOK | OK | OK |
| Deployments | | | | |
| Deployment methods | Docker or manually | Manually | Manually | Manually or hosted on the cloud |
| Others | | | | |
| Number of OPENID certifications | 5 | 4 | 5 | 2 |
| Preferred languages | C# | C# | Java | No preference |
| Tools or methods used to easily interact with APIs | Visual Studio extensions and nugget packages | Nuget packages | Unknown | Unknown |

# Architecture

The schema below shows the interactions between components.

Configuration API

Manager API

Uma API

Openid API

WebSite API

WebSite

UMA DB

OpenId DB

Configuration DB

Visual Studio extension

WebSite API DB

On a total of 11 components, there are 5 APIs, one visual studio extension, one website and four databases :

* Manager API : used by the clients to execute CRUD operations on OPENID assets for examples : client or resource owners.
* OpenId et UMA API : They are conformed to the RFCs OPENID and UMA.
* Configuration API : used by the clients to manage the OpenId API configuration for example : enable or disable external identity providers.
* WebSite API : an abstract layer which assigns Uris to resources.
* WebSite : it is used by an administrator to manage resource access.
* Visual studio extension : used by a .NET developer to easily interact with the different components in code.

The website is used by the administrator to manage the UMA and OPENID assets. Whereas the visual studio extension will be used by developers to access / protect resources such as API operations.

By default the administrator credentials are :

* Login : administrator
* Password : password

Use cases of both roles have been identified and they are described in the two next chapters.

# Use cases of the website

## Protect resources

<<include>>

<<include>>

<<include>>

<<include>>

<<include>>

Create a folder

Create a file

Move resources

Create a resource

Duplicate one or more resources

How to access to a resource

Rename a resource

<<include>>

Remove one or more resources

Move resources to sub directory

Administrator

Upload one or more resources

Display keyboard shortcuts

Search resources in the current working directory

Change visualization

Refresh current working directory

Administrator

Navigate to the previous / next folder

Add / Edit authorization rules

Select all resources

### Action: Move resources to sub directory

Select one or more resources and click on the item “” displayed in the context menu. Selected resources will be moved to a new sub directory.

### Action: Remove one or more resources

Select some resources and execute one of the following actions :

* Keyboard Shortcut : DEL
* Action *Delete* in the contextual menu 

### Action: Rename a resource

A resource can be renamed by executing one of the following actions:

* Keyboard Shortcut : F2
* Action *Rename* in in the contextual menu 

### Action: Move resources

Select several resources and drag & drop them into an existing folder. A special icon will be displayed in the bottom-right left corner.



### Action: Duplicate one or more resources

Select a resource and duplicate it by executing one of the actions :

* Display the context menu and click on the actions *copy* & *paste* or *duplicate*.



* Use the shortcuts “CTRL+C & CTRL+V“.

### Action: How to access to a resource

Select a resource and display its contextual menu. When the option “*how to access?*” is selected then a new window is displayed in front of the file explorer.



### Action: Create a file

Display the contextual menu of your current working directory and click on the option *new file* .

### Action: Create a folder

Do the same than before but instead of clicking on *new file* select the action *new folder* .

### Action: Upload one or more resources

Resources can be uploaded via different ways:

* Open a windows explorer and select several files, drag and drop them to the page “Protect your resources”.
* Click on the option upload files displayed in the contextual menu



### Action: Refresh current working directory

Either press F5 or click on the action *reload* in the contextual menu 

### Action: Display keyboard shortcuts

List of shortcuts can be displayed by pressing F1.



### Action: Change visualization

You can choose between vertical or horizontal resources visualization, to do that click on the icon  displayed in the menu bar.

### Action  Search resources in the current working directory

Accessible by pressing the keyboard shortcut CTRL+F or by typing your text in the search bar positioned at the top right-hand corner of the window.



### Action: Navigate to the previous / next folder

The navigation history is stored into your browser. Navigating to the previous / next folder can be done by executing one of the following actions :

* Use the buttons  displayed in the search bar
* Keyboard shortcut *Back* or *CTRL+←* : navigate to the previous folder
* Keyboard shortcut *CTRL+→* : navigate to the next folder.

### Action: Add / Edit authorization rules

If a resource is selected then it’s possible to add or edit its authorization rules. Click on the icon  or press the keyboard shortcut *CTRL+P*.

Rules and authorization policy were mentioned several times but they have never been clearly explained properly.

An authorization policy must contains at least one rule otherwise an error is displayed when attempting to create an empty one. They are used by the UMA server during the authorization process. Indeed when a client wants to access to a protected resource, he asked to the UMA resource an RPT token. The server internally decides to grant client access to a protected resource by executing the policy. The workflow is described in UMA website [[3]](#footnote-3).

Rules are exclusive and if at least one of them is satisfied then authorization is granted by the policy. Here the authorization policy formula is : . A rule is satisfied if the received claims values and / or clients are correct.

Policy structure overview :

Policy

In order to help you identifying policy and their rules, you can read the scenario below.

Problematic : Only the end-user thabart of the application Sample Client is allowed to view his bank account N°12345.

Solution : In the first place the identities should be identified and classified by their nature. There are four important information in the scenario, they are underlined in the text. Thanks to the decision table it was possible to classified them.

|  |  |
| --- | --- |
| Question | Nature |
| What is the identity of your resource to protect ? | Resource identifier |
| What is the identity of your client ? | Client Identifier |
| Which actions do-you want to perform on the resource ? | Permissions |
| Do-you want to restrict access to one or more users ? If yes, which information can be used to identify them ? For example : role or geographical position ? | Claims |
| Can-you identify the nature of your resource ? | Nature |

Classified information :

* Resource identifier : N°12345
* Client Id : Sample Client
* Permissions : view / read
* Claims : thabart
* Nature : BankAccount

When you have finished with the resource classification, you can go to the second step.

A resource needs to be created, first add a “BankAccount” folder into the “resources” directory, it should match with the nature of a resource.

Navigate to the new directory and create a file named “N°12345”, same value as the resource identifier.



Finally open the authorization policy editor and add a new rule. Below *allowed clients* select the authorized clients, in our case the client is “Sample Client”.

Under the sub-title “allowed claims” select “sub” et fill-in the field with “thabart”, then click on “Add” to add the claim into the list.

Persist your policy by clicking on *add rule* and on *save*.



A red square is displayed around a resource with an authorization policy applied on it.



### Action : Select all resources

To select all resources press *CTRL+A*.

You probably noticed that the actions available in the assets directory have not been described, it will be the object of the next chapter.

## Edit UMA & OPENID assets

<<include>>

<<include>>

Delete clients

Add a client

Administrator

Edit a client

Display scope information

Delete one or more scopes

Delete one or more resource owners

Edit resource owner

Delete or more authorization policies

Display authorization policy

Delete one or more UMA resources

Display UMA resource

### Action : Add a client

Open the folder *assets >> openid >> clients* and display its contextual menu.

Click on *add client * and fill-in the callback urls separated by a comma.

*Warning : urls should begin with https [[4]](#footnote-4)*

### Action : Delete one or more clients

Select several clients, display the contextual menu and click on *remove client*

.

### Action : Edit a client

Display the client information by clicking on *client information*  option in the contextual menu.



The client identity, secret and its callback urls are displayed on the new window.

Click on the link *Edit* to edit them.



The parameters are explained in the OPENID RFC [[5]](#footnote-5).

In most of the cases, advanced parameters don’t need to be updated except if you want to change the workflows. Here some uncommon scenarios in which advanced settings need to be updated:

* The client have a JWKS url, it is used by the OPENID server to decrypt and / or check the signature of the *request [[6]](#footnote-6)* parameter.
* Update the client authentication method.
* Change information displayed in the consent view.

If you want more information about the other advanced settings, you can contact-us or read the OPENID documentation[[7]](#footnote-7).

### Action : Delete one or more scopes

Open the directory *assets>>openid>>scopes*, select several scopes, display the context menu and click on remove scope .

### Action : Display scope information

Select one scope, open its contextual menu and click on *scope information* . The following information are displayed :

* Nature : Is-it an openid scope?
* Visibility : Is-it visible in the consent screen ?
* Contract : Is-is returned by the openid configuration endpoint */.well-known/openid-configuration*.

### Action : Delete one or more resource owners

Select some resource owners, display the context menu and click on remove resource owner .

### Action : Edit a resource owner

Select a resource owner, display its context menu and click on *resource owner info* button .

In the new window displayed in front, roles can be assigned to the resource owner. To do that, a local account needs to be created otherwise the message “not a local account” is displayed.

Why is-it useful to add roles ? Sometime it’s difficult to make an authorization policy based on the claims returned by external identity providers. If you want to limit the access of a resource only to certain roles but the claims are coming from facebook, the solution cannot be implemented simply because roles are not returned. But thanks to the website, claims returned by facebook can be enriched with roles. If we replace facebook by ADFS, additional roles don’t need to be specified because they are already returned.

Enrichment workflow :

Facebook

Hotmail

OPENID server

Client

Claims

Enriched claims

As we mentioned earlier a resource owner account can have two states :

External

Internal

Create a local account

Confirm his local account

Authenticate with external identity provider

When a resource owner is authenticated against one of the external identity providers : « Twitter », « Hotmail » or « GitHub » then an external account is automatically created.



To switch to a local account, click on the link *create a local account*:



### Action: Delete one or more authorization policies

Open the folder *assets>>uma>>authorization policies*, select several authorization policies, display the context menu and click on *remove authorization policy* .

### Action: Display authorization policy

Select an authorization policy, display its contextual menu and click on *authorization policy* . List of resources impacted by the policy and number of rules are displayed in the window.



### Action: Delete one or more UMA resources

Open the folder *assets>>uma>>resources*, select one or more resources, display the contextual menu and click on *remove resource* .

### Action: Display UMA resource

Display the contextual menu and click on resource info . Scopes and resource are displayed in the window :



## Identity providers

In the OpenId server, the end user can choose between one of the identity provider to authenticate himself: Hotmail, ADFS, GitHub or the Belgium id card.

When the claims are returned, they are used by the authorization policies to grant a request access to a protected resource.

If some claims are missing in the result, it’s always possible to enrich them with roles (refer to the chapter “edit a resource owner”).

List of actions available in the “connections” screen:

Enable / disable a provider

Edit a provider

Add a provider

Display providers

The following providers are configured by default: Microsoft, Linkedin, Google, GitHub, Facebook and the Belgium id card.

### Action: Display providers

Click on the connections tab to display the providers.



### Action  Enable / disable a provider

Providers can be enabled / disabled from the authentication page. To do that click on the button “On / Off” next to each provider.

By default an authentication page looks like:



New authentication page after the Facebook provider has been disabled:



Providers are managed by the administrator without worrying about restarting application.

### Action: Add a provider

Click on the *new identity provider* button to add a new one. When the window is displayed, try to fill-in all the fields. Use the decision table below to find all the values. When you have finished with it, click on the button *create* to persist the new provider.

|  |  |
| --- | --- |
| Fields | Description |
| Name | Name displayed is the authentication page |
| Callback path | Technical data used by the OPENID server. The value should start with « /signin-\* » otherwise there is no button displayed |
| Choose an identity provider type | Choose between OPENID, OAUTH2.0 or WS-Federation |

If there is no corresponding identity provider type, you can submit a new “enhancement” ticket to our website. The feature will be developed and deployed to your environment via Docker or XCOPY (refer to the Installation chapter)

If you want to use the resource owners which are stored in your database. We can help you writing some scripts to migrate from the old schema to the new one or create a new OPENID identity provider which takes your database as source.

### Action: Edit provider

Click on the provider title to edit it. You will be redirected to a new window where all the parameters are displayed. You will probably notice some differences between types, they are listed in the table:

|  |  |  |
| --- | --- | --- |
| Type | Parameters | Options |
| OAUTH2.0 | Callback path | TokenEndpoint |
| Namespace | AuthorizationEndpoint |
| Class | Scope |
| Code | ClientId |
| UserInformationEndpoint |
| ClientSecret |
| OPENID | Callback path | ClientId |
| Scope |
| ClientSecret |
| WellKnownConfigurationEndPoint |
| WSFEDERATION | Callback path | Realm |
| Namespace | IdEndpoint |
| Class |
| Code |

All parameters except namespace, code and class can be deduced with the configuration of your external identity providers for examples: the client identifier or secret. Those specific parameters are used by the OPENID server to parse xml or json and returns OPENID claims.

Those specific settings are not present in OPENID for a very simple reason, because a parser is not needed when OPENID is correctly implemented. The complete list of applications compliant with the standard is available online [[8]](#footnote-8) .

The parameter “code” presents in OAUTH2.0 and WS-Federation should contains a class with a method that respects a certain signature:

*public List<Claim> Process(JObject jObj) : OAUTH2.0*

*public List<Claim> Process(XmlNode node) : WS-Federation*

When the code has been written, the fields namespace and class name can be filled-in with their corresponding values. If we have the code below then the namespace is “Parser” and class name is “Example” :

*namespace Parser*

*{*

*public class Example*

*{*

*public List<Claim> Process(XmlNode node) { }*

*}*

*}*

## Parameters

Parameters are accessible via the “settings” tab. The token / authorization code expiration times can be updated.



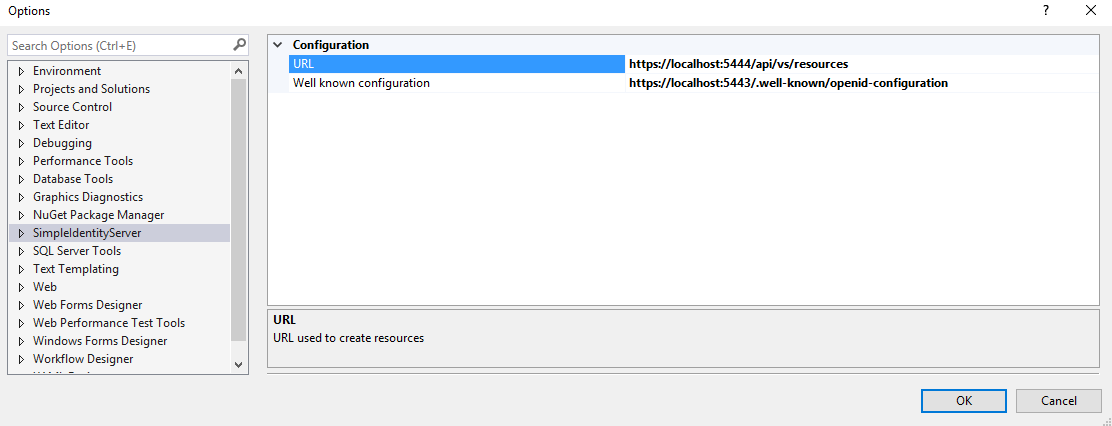
# Use cases of the Visual Studio extension

The extension is working only with Visual Studio version 2015. The previous versions will be supported in future release.

## Configure parameters

The configuration window is accessible via “Tools >> Options >> SimpleIdentityServer”. There are two URLs, the first one is used by the “generate resource” feature to generate a resource for each selected actions.

The other url is mostly used to retrieve an access token via client credentials, it is passed in the Authorization header to access to protected operations.



## Generate one or more resources

Choose any API project for which you want to protect its operations and display its contextual menu, then click on the option *generate resource* 

Don’t worry if it takes some times to load the operations. If after a while they are not displayed then close the panel window, select the project and reopen it again.

Here some features available in the window:

Refresh the resources

Protect one or more resources

Developer

### Refresh the resources

To refresh the resources, just click on the button *refresh*. It will take some times before they are displayed.



### Protect one or more resources

Select the resources that you wish to protect and indicate the API version. It’s very important to mention it because several versions of the same API can be deployed and they can present different contracts. The number should be incremented if there are any breaking changes with the previous version.

When the form has been filled-in then click on the button *protect* to protect the resources.

Behind the scene the following sub-tasks are performed :

* Create a resource by following a naming convention : “*Api\<assembly name>\<version number>\<controller name>\<action>*”
* Add Nuget packages : “SimpleIdentityServer.UmaIntrospection.Authentication” and “SimpleIdentityServer.Uma.Authorization”
* Add a test file “Startup\_Sample.cs”, it can be reused to protect the operations.



Technical details are described in the « technical documentation »».

## Generate security proxy

An application / client who wants to access to protected resources, needs to retrieve an RPT token first and passes it to the Authorization request header. To do that the option *generate security proxy* can be used by the developer.

The following actions are available in the window:

Search one or more resources

Add security proxy

Developer

### Action : Search one or more resources

Type the name in the search bar and click on “search” to confirm.



### Action : Add a security proxy

Select one resource and click on *Add*. The following sub-tasks are performed :

* Install the Nuget package : “SimpleIdentityServer.Proxy”
* Add files to the selected project :
  + SecurityProxy\_\*.cs : contains a static method which can be used to retrieve an RPT token
  + AuthProvider.cs : can be used to retrieve an identity token via resource owner credentials

# Scenarios

In this chapter we will focus on the most common problems in enterprise, and we will try to solve them.

Samples can be found here :

*https://github.com/thabart/SimpleIdentityServer.Samples.git*

## First scenario: An heavy application wants to access to a protected API

**Context** An e-commerce enterprise has internally developed a tool used by his marketing team, to retrieve information about his most loyal clients.

The application has been developed in WPF and interact with a RESTFUL API to retrieve the clients. Only this application and users that belong to the “marketing” group are authorized to retrieve the list.

**Problem**: How the application can access to the protected operation?

**Solution**:

The workflow is made of three big steps:

* Identity token: retrieve an identity token with implicit grant-type [[9]](#footnote-9). The token is returned to the client in a callback parameter.
* *RPT token* The identity and access token (valid for the scope uma\_authorization) are passed in the request to retrieve the RPT one [[10]](#footnote-10). When it is received by the WPF application, the token is passed in the Authorization header to retrieve the loyal clients. Both parameters are required by the authorization policy.
* *Check RPT token*: The token is checked against the introspection endpoint [[11]](#footnote-11), this endpoint is offered by the UMA server.

WPF application

API service

OPENID server

Retrieve identity token

Identity token

UMA server

Authenticate user

Retrieve RPT token

RPT token

Retrieve clients

We spared you the implementation details, otherwise it will be too much difficult to understand. The workflow is normally much more complex and contains more intermediate steps.

Before going further, we are going to prepare the environment by following the steps:

Identify and classify identities

Add a client

Add a resource

Develop

Add authorization policy

Assign marketing role to the resource owner

### Identify and classify identities

The decision table can help you identify and classify them.

|  |  |
| --- | --- |
| Questions | Type |
| Which application wants to access to the resource ? | Client |
| Which operation do-you want to protected ? Identify the service name, his version, the business entity (client, product) et operation. | Resource : Concatenation of service name, version number, business entity and operation. |
| Which applications are authorized to access ? | Authorized clients |
| Which resource owner informations are accepted ? | Claims |

Result:

* Client: WPF application
* Resource  ClientApi / v1 / Clients / Get
* Authorized clients : WPF application
* Claims : role marketing.

When you have finished, the identities can be added.

### Add a client

Add a new client and edit his properties (for more information refer to the use cases “add a client” and “edit a client”). In the new window update as many properties as you can. There are parameters very easy to update like the displayed name and callback urls, however some of them are not obvious. Try to guess the grant-types value, they are needed by the client to interact with the other components. If you succeed, then other values can be easily deduced.

The better way to choose grant-types is by identifying all interactions with the OPENID server. Only one interaction has been identified in the workflow. An identity and access token (valids for the scopes “uma\_protection” and “uma\_authorization”) are returned to the client. Keep in mind that whatever the situation, when a client is trying to retrieve an RPT token, an access token valids for both scopes should be retrieved.

WPF application

API service

OPENID server

Retrieve Identity Token

& access token

Identity token & access token

UMA server

Authenticate user

Retrieve RPT token

RPT token

Retrieve clients

Depending on the technical implementation of the client, the grant-type can be different. Relationships between implementation type and grant-types are listed:

|  |  |  |  |
| --- | --- | --- | --- |
| Implementation | Grant-Type | Advantages | Disadvantages |
| Embedded browser | implicit | Authentication is delegated to OPENID | No control over the look and feel |
| Create formula | client\_credentials | Control look and feel | The code must be obfuscated, because the client identifier and secret must be hidden from malicious users.  Don’t choose this approach if there is no trust relationship between the client and server. |

When you have the grant-types then the other parameters can be deduced. Just answer the questions of the next two tables:

|  |  |
| --- | --- |
| Grant type | Response types |
| Authorization code | Authorization code |
| Implicit flow | Token |
| Authorization code |
| Identity token |
| Client credentials |  |
| Password |  |
| Refresh token |  |

Mappings between grant-types and response-types

|  |  |
| --- | --- |
| Type token | Scopes |
| RPT token | uma\_authorization |
| uma\_protection |
| Identity token | Openid |
| Profile |
| Role |
| *Complete list can be found in the documentation [[12]](#footnote-12)* |

Mappings between tokens and scopes

When all the parameters have been found, the edit page can be filled-in:

* Callback uris : <https://client.com>
* Grant types  implicit
* Response types: token, id\_token
* Scopes: openid, profile, role, uma\_authorization, uma\_protection



### Add a resource

There are different two ways to add a resource, either with the website (refer to the use case “create a folder”) or either with the Visual Studio extension (refer to the use case “generate one or more resources”). In both cases, the name must respect a certain convention which has been decided by you and it must be consistent with the other resources. For example, imagine there are two pictures, one “Thierry\picture.png” and an another “Lokit \ picture.png”. At first glance this organisation seems to be awkward, and it can be easily reorganized in something cleaner: “images\thierry-picture.png & images\lokit-picture.png”.

If your resource is an API operation, then we suggest to respect this convention: “*Apis \ <application name> \ <version number> \ <business entity> \ <operation>*”

In our scenario the resource name is: « *Apis \ ClientApi \ v1 \ ClientsController \ Get* ». If you are working with the Visual Studio Extension you don’t have to be worried about the name because the convention is respected.

We really insist on the fact that it’s very important to have a good architecture since the beginning. If later the structure is modified, then all the resource consumers will be impacted and they must be updated and redeployed again.

### Add authorization policy

When the client and resource have been created then the authorization policy can be assigned. If you encounter some difficulties to identity clients and claims, you can always refer to the use case “Add / Edit authorization rules”.

* Allowed clients: WpfClient
* Allowed claims: role marketing
* Permissions: execute



### Assign marketing role to the resource owner

The marketing role have to be assigned to the resource owner, otherwise the authorization policy will never pass. Choose a resource owner, edit his properties and assign the role.



### Develop

When you have finished with the initial setup, you can start to implement the changes.

#### API

There are two different kinds of authorization policies:

* Conventional: the URL of the resource must match the structure of the project and also the API version. The last value can be set to the property “ConventionalUmaOptions.Version”.
* Individual: limit the access to one specific resource by passing the URL and scopes.

The technical implementation is running only on ASP.NET CORE projects. If you are working on a previous version, we can help you with the implementation.

Previous versions will be supported in future release.

The Nuget packages

« *SimpleIdentityServer.UmaIntrospection.Authentication* » and « *SimpleIdentityServer.Uma.Authorization* »  must be installed on your API project.

Enabling the conventional authorization is pretty simple. Insert the code below into the method “ConfigurationServices” of the Startup class.

// Authorization policy

services.AddAuthorization(options =>

{

// Add conventional uma authorization

options.AddPolicy("uma", policy =>

{

// policy.Requirements.Add(new ConventionalUmaAuthorizationRequirementTst(null));

policy.AddConventionalUma();

// options.AddPolicy("resourceSet", policy => policy.AddResourceUma("<url>", "<read>","<update>"));

});

});

L’opération « ClientController/Get » doit ensuite être décorée par l’attribut : 

#### WPF application

In first place, add the Nuget package “SimpleIdentityServer.Proxy” to your client. You need it get an RPT token to retrieve the list of clients.

An authentication window must be displayed by your application. There are two different ways to do it, either by displaying the OPENID authentication webpage inside an embedded browser or with a WPF formula. Whatever the methodology you have chosen, at the end the identity and access tokens must be retrieved and stored.

Our preference of course is to use an embedded browser, because we don’t trust the OPENID server.

The following code is used to retrieve an RPT token. Don’t forget to update the properties like UmaConfigurationUrl, OpenIdConfigurationUrl and RootManagerApiUrl.

public static async Task<string> GetRptToken(

string idToken,

string umaProtectionToken,

string umaAuthorizationToken)

{

var factory = new SecurityProxyFactory();

var proxy = factory.GetProxy(new SecurityOptions

{

ClientId = Constants.ClientInfo.ClientId,

ClientSecret = Constants.ClientInfo.ClientSecret,

UmaConfigurationUrl = "https://localhost:5445/.well-known/uma-configuration",

OpenidConfigurationUrl = "https://localhost:5443/.well-known/openid-configuration",

RootManageApiUrl = "https://localhost:5444/api"

});

try

{

var result = await proxy.GetRpt("resources/Apis/ClientApi/v1/ClientsController/Get", idToken, umaProtectionToken, umaAuthorizationToken, new List<string>

{

"execute"

});

return result;

}

catch (Exception ex)

{

return null;

}

}

Invoke the method GetRptToken and pass the tokens as parameters. When the RPT is received, it can be passed to the authorization header.

var request = new HttpRequestMessage

{

Method = HttpMethod.Get,

RequestUri = new Uri("http://localhost:5100/api/clients")

};

request.Headers.Add("Authorization", $"Bearer {rptToken}");

If you want to launch the application, open the solution “Scenario1 \ WpfClient” from the repository “SimpleIdentityServer.Samples”. Open the two files ”UI\WpfClient\SecurityProxyClientApi”, “UI\WpfClient\Constants” and update their properties.

Launch the application and enters the credentials of a resource owner with marketing role.



If the authentication succeeds, then the list of clients is displayed:



## Second scenario : an API want to access to a resource

**Context**: A new RESTFUL service has been developed to retrieve the most loyal clients. Those data must be enriched with marketing information, they have been gathered by the marketing team since the creation of the enterprise.

At the end consolidated data must be returned by the API. They will be consumed by analytics tools to follow the evolution of client satisfactions.

**Problem** : How the API can access to the clients ?

**Solution**:

The workflow presents here is very to similar to the one in the first scenario. There is only one difference, the identity token is not returned to the client. The reason is simple, if the client is correct then an RPT token is granted by the UMA server. User information are not needed by the authorization policy then by deduction the identity token doesn’t have to be passed in the request.

Retrieve RPT token

API

API

OPENID server

UMA server

Retrieve actions

The workflow that we are going to follow to implement the solution looks like the one in the previous scenario. The task “assigned marketing role to the resource owner” have been removed.

Identify and classify identities

Add a client

Add a resource

Develop

Add authorization policy

### Identify and classify identities

After using the decision table (refer to the previous scenario) we obtained :

* Client : API
* Resource : ClientApi / v1 / Clients / Get
* Authorized clients : API
* Claims : none

### Add a client

Most of the parameters can be easily deduced including the grant-type ! Which grant-type will you choose by taking into account that there is a trust relationship with the OPENID server and there is no identity token ?

The correct answer is … “client\_credentials”. Indeed there is no danger to embed the client credentials into the code because the API is hosted in a server, so the code cannot be accessed by malicious users.

Client parameters :

* Grant-types : client credentials
* Scopes : uma\_authorization et uma\_protection



### Add a resource

The resource already exists (refer to the first scenario).

### Add authorization policy

Edit the authorization policy of the resource « resources \ Apis \ ClientApi \ v1 \ ClientsController \ Get » and add a new rule. If you have some doubts about the parameters, we invite you to re-read again the use case « add / edit authorization policy rules ». It’s very important to guess by yourself the parameters, at the beginning it’s difficult but you will quickly take the habit.

Rule parameters :

* Allowed clients : Marketing API
* Allowed claims : empty
* Permissions : execute



### Develop

#### Client (API)

Install the Nuget package « SimpleIdentityServer.Proxy » to your client.

Retrieve an access token valids for the scope “uma\_protection” and “uma\_authorization” by using the grant-type client\_credentials.

private async Task<GrantedToken> GetAccessToken()

{

return await \_identityServerClientFactory.CreateTokenClient()

.UseClientSecretBasicAuth(\_clientId, \_clientSecret)

.UseClientCredentials("uma\_authorization", "uma\_protection")

.ResolveAsync("https://localhost:5443/.well-known/openid-configuration");

}

Invoke the function « GetRptToken » and pass the access token value as parameters (umaProtectionToken and umaAuthorizationToken). The result can be used to retrieve the clients.

public static async Task<string> GetRptToken(

string umaProtectionToken,

string umaAuthorizationToken)

{

var factory = new SecurityProxyFactory();

var proxy = factory.GetProxy(new SecurityOptions

{

UmaConfigurationUrl = "https://localhost:5445/.well-known/uma-configuration",

OpenidConfigurationUrl = "https://localhost:5443/.well-known/openid-configuration",

RootManageApiUrl = "https://localhost:5444/api"

});

try

{

var result = await proxy.GetRpt("resources/Apis/ClientApi/v1/ClientsController/Get", umaProtectionToken, umaAuthorizationToken, new List<string>

{

"execute"

});

return result;

}

catch (Exception ex)

{

return null;

}

}

If you want to launch the complete example, open the solution « Scenario2 \ MarketingClient » from the repository « SimpleIdentityServer.Samples » and update the properties.

Launch both projects “ClientApi” and “MarketingClient” and browse the URL :

<http://localhost:5103/api/ratings>. The clients will be displayed in the browser.



## Third scenario: Limit access to certain website features

**Context**: Only users with « administrator » role can see the button « administrate » whereas the others cannot.

**Problème** : How to limit access to certain features based on the users roles ?

**Solution** :

Compared to other scenarios, the workflow is pretty simple :

* End users authenticate themselves to the OPENID server. If authentication succeeds then identity and access tokens are returned to the website.
* The tokens are used to retrieve the permissions from the UMA server.

Retrieve permissions

Identity & access tokens

Authenticate

End-User

UMA server

WebSite

OPENID server

Like the other scenarios, we are going to follow a set of steps to implement the solutions.

Identify and classify identities

Add a client

Add a resource

Develop

Add authorization policy

Add « administrator » role to resource owner

### Identify and classify identities

After using the decision table we obtained :

* Client: WebSite
* Resources: WebSite \ Scenario3 \ Home \ Admin
* Authorized clients : WebSite
* Claims : administrator role

### Add a client

Once again the parameters are fairly easy to deduce except the grant-types … Your choice depends on your technical implementation, if you have your own authentication page then the logical grant type is “password”.

If this grant-type has been chosen then you have to be careful to hide the client credentials because the application doesn’t trust the OPENID server.

In the other side, if you are using the OPENID authentication page then the “implicit” grant-type should be used. The user-agent will be redirected to OPENID , if the authentication succeeds then the tokens are returned to the website. Our preference is the implicit one because it’s easier to implement and more reliable.

Don’t forget the scopes « uma\_authorization » & « uma\_protection » because there is an interaction with the UMA server.

Client parameters :

* Callback url : <https://localhost:5105/Authenticate/Callback>
* Grant types : implicit
* Response types : token & id\_token
* Scopes : uma\_protection, uma\_authorization, role, openid, profile



### Add a resource

Add the resource « WebSite \ Scenario 3 \ Home \ Admin ».



### Add an authorization policy

Assign an authorization policy to the resource “WebSite \ Scenario3 \ Home \ Admin” and add the following rule :

* Allowed clients : WebApplicationScenario3
* Permissions : read
* Allowed claims : administrator role



### Add administrator role to the resource owner

Edit the resource owner properties and add a new “administrator” role :



### Develop

If you want to launch the full example, open solution “Scenario3 \ WebApplication” from the repository “SimpleIdentityServer.Samples”. Update the parameters and launch the application.

Authenticate with an administrator account, you will see four permissions. Try to connect with an another none administrator account, there are only three permissions because the Admin resource is not accessible.



Permissions of an administrator



Permissions of a none administrator user

# Installation

We assumed that everything will be installed on the same machine. Keep in mind that it’s also possible to deploy the components on separate machines ! It can be useful if you want load balancing.

## Mandatories steps

Whatever the methodology (manual or Docker) you have chosen to install the product, the following perquisites must be done.

### Install GIT

GIT must be installed, follow the official guide ([link](https://git-scm.com/downloads)) for more information.



### Install the certificate

The certificate « LokitCA.cer » must be added to the certificate store “Local User \ Trusted CA”. Without it the website is not going to work.



## Manual installation

TODO

## Install with Docker

### Install docker

Install Docker by following the official guide ([lien](https://docs.docker.com/engine/installation/))



### Configure Virtual Box

By default Docker is using Virtual Box to launch his virtual machines. Some forwarding port rules must be added to the “default” machine :

* Open VirtualBox and select the default machine
* Click on Configuration
* Click on “connection” tab
* Click on the button “port redirection” and add those rules :



Note : A rule is missing in the screenshot, add dit :

Nom : Kibana

Protocol : TCP

IP hôte : 127.0.0.1

Port hôte : 5601

Port invité : 5601

### Launch the deployment

In a command prompt execute the instruction « docker-compose  up ». When the application is installed and deployed, you can browse the website via

<http://localhost:4200>.

# Technical documentation

## Monitoring

TODO



## Nuget packages

### SimpleIdentityServer.Proxy

TODO

### SimpleIdentityServer.UmaIntrospection.Authentication

TODO

### SimpleIdentityServer.Uma.Authorization

TODO

## Load balancing

Load balancer

OpenId #2

OpenId #1

Load balancer

Configuration #1

Configuration #2

Load balancer

WebSiteAPI #2

WebSiteAPI #1

Load balancer

UMA #1

UMA #2

Load balancer

Manager #1

Manager #2

1. RFC OPENID : http://openid.net/specs/openid-connect-core-1\_0.html [↑](#footnote-ref-1)
2. RFC UMA : https://docs.kantarainitiative.org/uma/rec-uma-core.html [↑](#footnote-ref-2)
3. RPT token : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.5.1 [↑](#footnote-ref-3)
4. Redirect\_uri validation rules : <https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata> [↑](#footnote-ref-4)
5. Client parameters : <https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata> [↑](#footnote-ref-5)
6. Request parameter : <http://openid.net/specs/openid-connect-core-1_0.html#JWTRequests> [↑](#footnote-ref-6)
7. OPENID documentation : <http://openid.net/specs/openid-connect-core-1_0.html> [↑](#footnote-ref-7)
8. Applications compliants with OPENID : http://openid.net/certification/ [↑](#footnote-ref-8)
9. Implicit grant type : http://openid.net/specs/openid-connect-implicit-1\_0.html [↑](#footnote-ref-9)
10. RPT token : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.5.1 [↑](#footnote-ref-10)
11. Introspection endpoint : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.4.1 [↑](#footnote-ref-11)
12. Liste des scopes : http://openid.net/specs/openid-connect-core-1\_0.html#ScopeClaims [↑](#footnote-ref-12)