SimpleIdentityServer Documentation

Release 1.0.0

habart thierry

Mar 24, 2017

Contents

CHAPTER 1

Topics:

Terminologies

Words	Definitions
Resource	Anything that needs to be protected for example : A user picture or an API operation
Folder	Resource with one or more resources.
File	Single resource.
Authorization policy	Contains one or more security rules and is assigned to one or several resources. Used by the UMA server to check if an incoming request can execute the requested operations (read, write or delete) against a 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.

Benchmark

The following table lists the differences between our product and others : SimpleIdentityServer (version 1.0.1), IdentityServer, Gluu server and OAUTH0. It has been made in "16-01-2017", if you noticed some differences don't hesitate to contact-us by email.

	SimpleIdentityServer	IdentityServer	Gluu server	AUTH0
Authors	Habart Thierry	Thinktecture	Gluu	Auth0
Start date	October 2015	January 2014	March 2014	November 2012
Workflow OAUTH2.0				
Client credentials	✓	✓	✓	✓
Password	✓	✓	✓	✓
Refresh token	✓	✓	✓	✓
Workflow OPENID				
Implicit	✓	✓	✓	✓
Hybrid	✓	✓	✓	✗
Other OPENID features				
Register a client	✓	✗	✓	✗
Sign token	✓	✓	✓	✓
Encrypt token	✓	✗	✓	✗
Invalidate session	✓	✓	✓	✗
Client authentication methods (RFC)				
client_secret_basic	✓	✓	✓	✓
client_secret_post	✓	✓	✓	✓
client_secret_jwt	✓	✗	✓	✗
private_key_jwt	✓	✗	✓	✗
Response modes				
query	✓	✓	✓	✓
fragment	✓	✓	✓	✓
form_post	✓	✓	✓	✓
Other parameters				
Claims	✓	✓	✓	✗

Continued on next page

Table 1.1 – continued from previous page

	SimpleIdentityServer	IdentityServer	Gluu server	AUTH0
Request	✓	✓	✓	✗
Quality				
Code coverage	84%	???	???	???
Number of UTs	633	???	???	???
UMA (RFC-UMA)				
UMA supported	✓	✗	✓	✗
SCIM2.0 (RFC-SCIM)				
SCIM2.0 supported	✓	✗	✓	✗
RFID ISO15693 and ISO14443A/B				
OpenId provider	✓	✗	✗	✗
UI				
UI exists	✓	✓	✓	✓
CRUD openid assets	✓	✓	✓	✓
CRUD uma assets	✓	✗	✓	✗
Organize resources by urls	✓	✗	✗	✗
Enable or disable external identity providers	✓	✗	✓	✓
Monitoring tool	Elastic Search & Kibana	Not specified	Custom tool	Custom tool
Two factor authentications	✓	✗	✓	✓
Deployment				
Deployment methods	Docker MSI	Docker Nuget packages	Manually	Manually SAAS
Others				
OPENID certifications	5	4	5	2
Preferred languages	C#	C#	Java	No preference

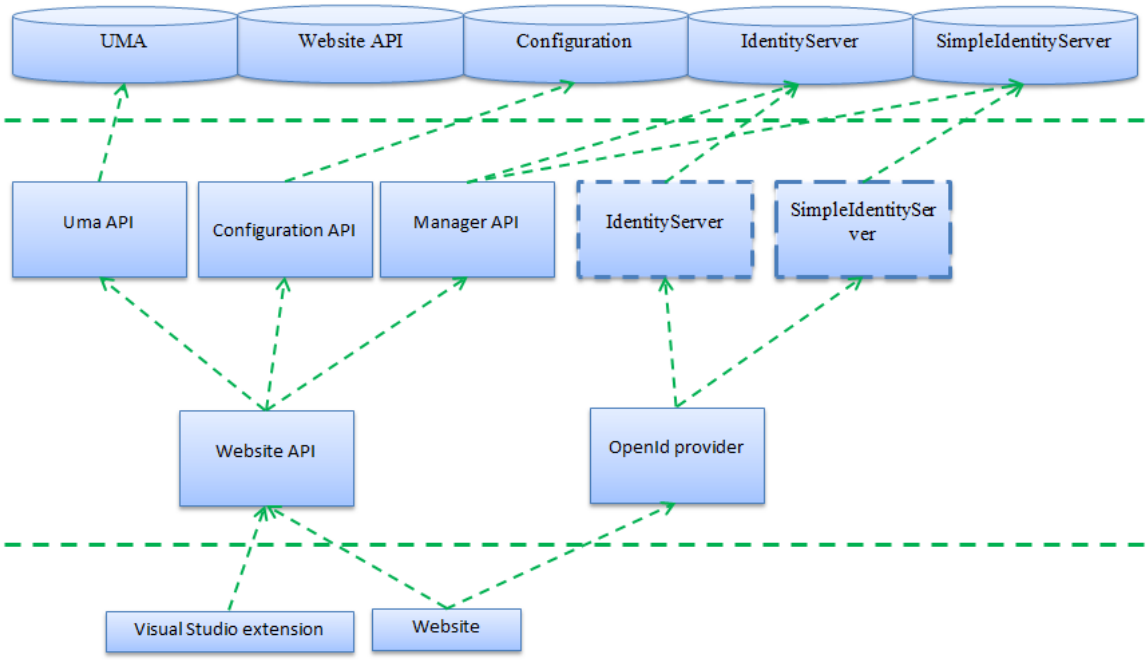
Continued on next page

Table 1.1 – continued from previous page

	SimpleIdentityServer	IdentityServer	Gluu server	AUTH0
Tools or methods used to easily interact with APIs	VSE and Nuget packages	Nuget packages	???	???

Architecture

The schema below shows the interactions between components



On a total of 13 there are : 6 APIs, one visual studio extension, one website and 5 databases :

- Manager API : used by the clients to execute CRUD operations on OPENID assets for examples : clients or resource owners.
- OpenId providers : you can choose between our provider (SimpleIdentityServer) or IdentityServer4. They both respect the OPENID RFC. For more information you can read the *benchmark*.
- UMA : Implementation of the User-Managed Access (UMA) profile.
- Configuration API : used by the clients to manage the OpenId configuration for examples : enable or disable external identity providers.
- WebSite API : An abstract layer which assigns URIs to resources.
- WebSite : 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.

Installation

We assumed that all the components are going to be installed on the same machine. Keep in mind that it's also possible to deploy them on separate servers. It can be useful to do it if you want a load balancing system.

Mandatories steps

There are two different ways to deploy the product, either manually with *MSI installer* or with *Docker*.

Whatever the methodology chosen the following steps are automatically performed, except the last one (needs to be done if Docker is chosen) an the third (needs to be done if the product is manually installed):

Install dotnet

Most of the components have been developed with .NET CORE and they can run on any environments such as : Windows, Linux or MAC.

The runtime version 1.0.1 and the SDK SDK version 1.0.0 - Preview 2 must be installed, you can refer to the website.

Install redis

Redis is an open source, in-memory data structure store, used as cache.

Follow the official guide to install it.

Install mongo

Mongo is a NoSql database used by the API to store the resources.

It can be downloaded from the website.

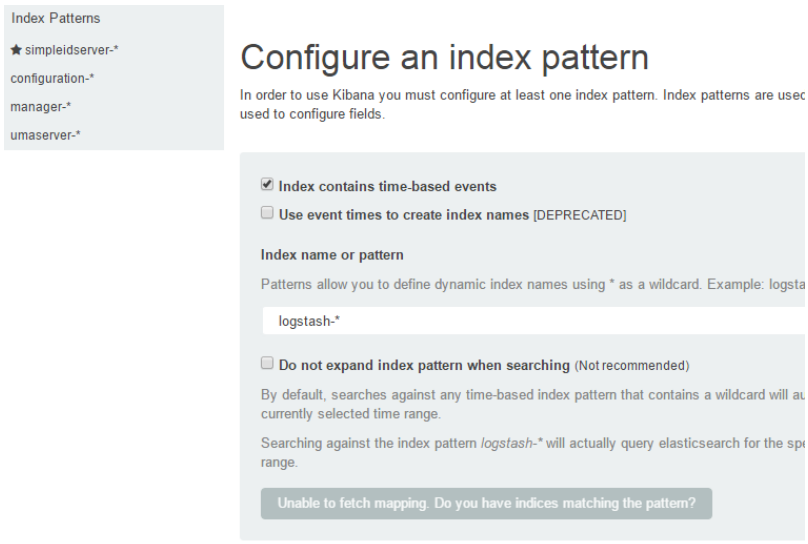
Install elastic search and kibana

Elastic search and Kibana must be installed to visualize the errors and logs. You can refer to the documentation to install both products (Elastic Search and Kibana).

When they are installed then the dashboard can be imported. First download the zip file and extract its content into a new directory named *Settings* then browse the Kibana website. Click on *Settings* > *Advanced* and import the *Setting / export-kibana.json* file.

In the second step navigate to **Settings > Indices** and add the following index patterns :

- *SimpleIdentityServer* : simpleidserver-*
- *Uma server* : umaserver-*
- *Manager server* : manager-*
- *Configuration* : configuration-*



Attention: It's necessary to first try to launch the product before starting to create the indexes. The reason is because they are created by the components during runtime. If the specified index doesn't exist then the message "*Unable to fetch mapping. Do you have indices matching the pattern*" is displayed.

Install NodeJs

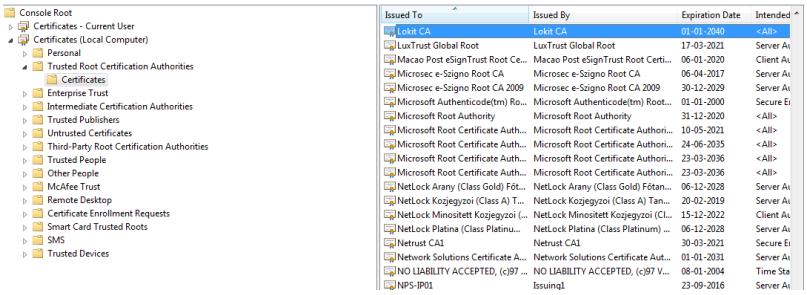
NodeJs is required to launch the website (developed with the framework EmberJs) and its API (developed with the framework Express).

The MSI can be downloaded from the official website.

Install the certificate

The certificate LokitCA.cer must be added to your certificate store **Local Computer Trusted CA**.

1. Execute the **mmc.exe** tool
2. Click on the menu item **File Add Remove snap in ...**
3. Below the available snap-ins select the **certificates** item and click on the **add** button
4. In the new window named **certificates snap-in** select the option **computer account** and click on **next** and **finish**
5. In the tree view located in the left panel, select the node **Console Root > Certificates (Local Computer) > Trusted Root Certification Authorities > Certificates**
6. Click on **more actions > all tasks > import ...**
7. In the new window named **certificate import wizard** select the certificate which has been downloaded and click on **Next**, **Next** and **Finish**



Attention: If you don't have a valid certificate installed on your machine then the website will not work and an error NOT TRUSTED CERTIFICATE will be displayed in your console. However it's still possible to work without it and to do that the certificate validation must be disabled in your favorite browser. It's a very dangerous step then don't leave the certificate validation disabled too long ! Refer to the documentation below :

- **Internet Explorer** : http://windowsitpro.com/windows/disabling-internet-explorer-feature-checks-server-certificate-revocation
- **Google Chrome** : chrome://flags/#remember-cert-error-decisions
- **FireFox** :http://smallbusiness.chron.com/disable-firefox-rejecting-certificates-59249.html

Manual installation

This article will show you how to manually install the product on Windows. First download the MSI from the URL and launch it as **administrator**.

Attention: SQLServer must be installed on your machine.

Sections:

Install all the required softwares

All the required softwares described in the previous part *Mandatories steps* are installed on your machine except **Kibana, Elastic Search and SQLServer**.

The softwares that will be installed are :

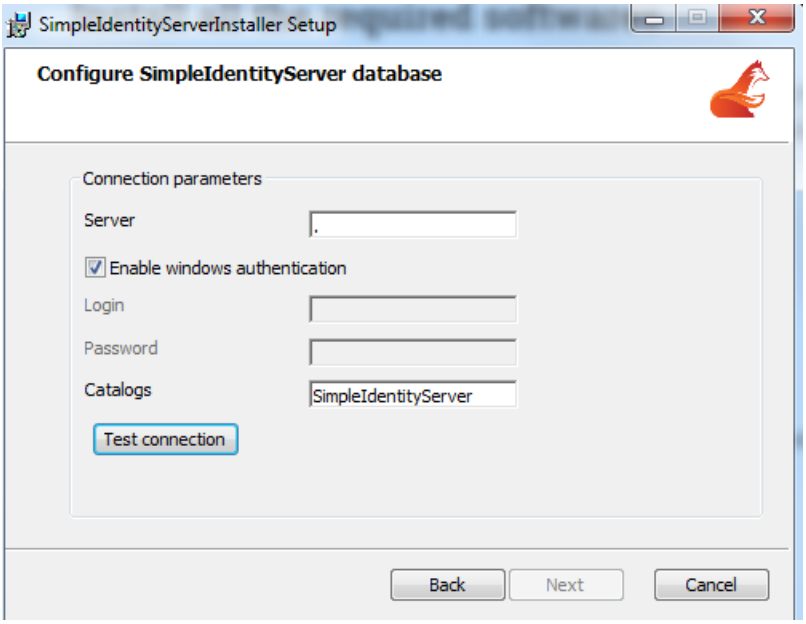
1. Redis version 3.2.100
2. DotNetCore Runtime version 1.0.1
3. DotNetCore SDK version 1.0.0.preview2-003133
4. NodeJs version 4.6.0
5. MongoDb version 3.2.10

When the installation is completed then the MSI product is launched.

Install the product

The MSI is made of six screens :

1. The classical Welcome window
2. An end-user license agreement
3. Four screens where the connection strings need to be specified. In each of them there is a *test connection* button. Fill-in the fields (server, credentials and catalog) and test the connection by clicking on the button. If it successes then the Next button is enabled otherwise an error message is displayed.



Attention: At the moment only SqlServer is supported by the Installer. However the appsettings.json files can be updated to use other database engines such as PostGre or SqlLite. For more information you can contact-us or refer to the technical documentation.

The four connection strings are :

1. SimpleIdentityServer : it contains all the assets of the SimpleIdentityServer OpenId provider.
2. Uma database : it contains all the assets of the UMA server.
3. Configuration database : it contains some settings (expiration time ...) and the external authentication providers.
4. Identity server database : it contains all the assets of the IdentityServer OpenId provider.

The next part consists to install Elastic Search and Kibana on your machine, refer to the chapter *Install Kibana*.

When everything is installed jump to the next section *Getting started*

Install with Docker

1. *Install the certificate*.
2. Install Docker.

3. Open Virtual box, select the *default* machine and add the following forwarding port rules

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Api	TCP	127.0.0.1	5445		5445
OpenId	TCP	127.0.0.1	5443		5443
Uma	TCP	127.0.0.1	5444		5444
WebSite	TCP	127.0.0.1	4200		4200
Kibana	TCP	127.0.0.1	5601		5601

4. Fetch the GIT repository https://github.com/thabart/SimpleIdentityServerDocker.git into a new folder named "Docker".
5. Open a command prompt and navigate to the folder "Docker".

If you choose our OpenId provider *SimpleIdentityServer* then execute the command :

```
docker-compose up
```

Otherwise if you choose *IdentityServer4* then execute the following command :

```
docker-compose -f Docker-Compose-IdServer.yml up
```

Getting Started

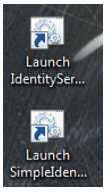
In this part we are going to present the most important features and how to quickly secure an API.

Sections:

If the application has been installed with Docker then jump to the next section *Install the Visual Studio extension* otherwise continue to the next one.

Launch SimpleIdentityServer / IdentityServer

Normally two shortcuts have been added to your Desktop : *Launch-IdServer.cmd* and *Launch-SimpleIdentityServer.cmd*. The difference between both is the OpenId Provider. Indeed you can choose between our implementation *SimpleIdentityServer* or IdentityServer4. To help you making a decision, you can read the *Benchmark*.

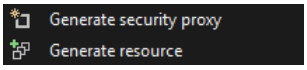


Install the Visual Studio Extension

A Visual Studio Extension is available and can be downloaded here. It provides some useful features and thanks to it any developers can easily protect an API and create client without requiring strong knowledge in OpenId, Uma and OAUTH2.0.

> **Attention:** Unfortunately the tool is working only with Visual Studio 2015, ASP.NET CORE and C# projects. Previous versions and other languages will be supported in future releases.

Once it has been installed, create an empty project and display its contextual menu. You'll see two new items in it :

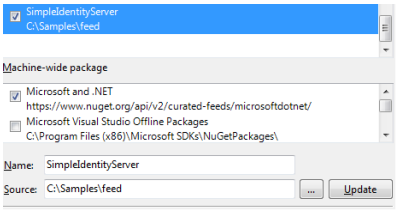


- The first-one **Generate security proxy** generates C# functions into the selected project. They are called by a client / API consumer to retrieve an RPT token. This one is passed into an HTTP Request Header to access to a protected resource.
- The second option **Generate resource(s)** displays the API operations of the selected project. The developer selects some operations to protect and submits his choice by clicking on **Protect**. At the end the resources are automatically created based on a naming convention.

In the next sections we will run the second Scenario that consists to call a protected API via an API. Fetch the GitHub project https://github.com/thabart/SimpleIdentityServer.Samples.git into a new directory named "Samples".

Add Nuget feed & open the solution

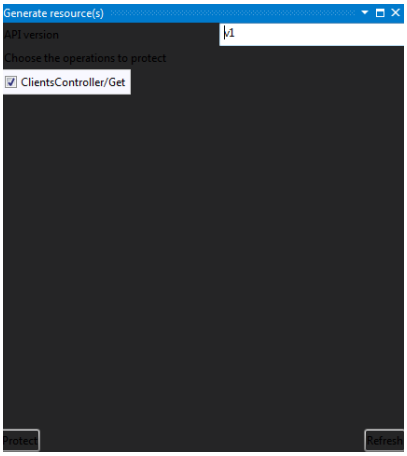
Download the zip file and extract its content into a new directory named *Feed*. Start a new Visual Studio instance and add this folder as a new Nuget feed. To do that open the *Package Sources* window by clicking on **Tools > Options > Nuget Package Manager > Package sources**. Fill-in the field **name** with SimpleIdentityServer and specify the full path of the feed as **source**. At the end you should obtain something like :



Still on the same Visual Studio instance, open the solution "Samples/Scenario2/MarketingClient.sln", restore the Nuget packages and build the solution. You are now ready to run the solution !

Protect the operation

Select the project "ClientApi" and display its contextual menu. Click on the item **Generate resource(s)** and wait some minutes before the list is displayed. Select the operation "ClientsController/Get" and click on **Protect**.

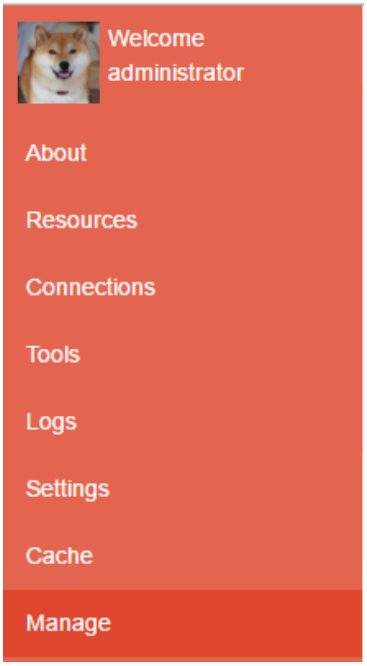


Now the resource has been added, its permissions can be edited via the website.

Add permission

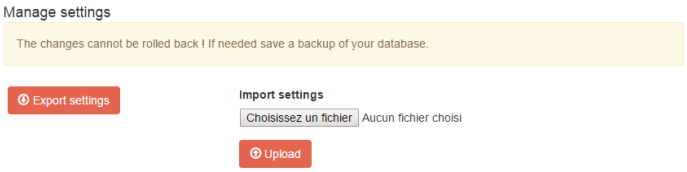
Browse the URL of the website and submit the following credentials. If they are correct then you'll see more options in the left panel:

```
Login: administrator
Password: password
```



Download the zip file which contains all the settings and extract its content into a new folder named *Settings*. Click on **manage** and import one of them depending on your OpenId Provider nature :

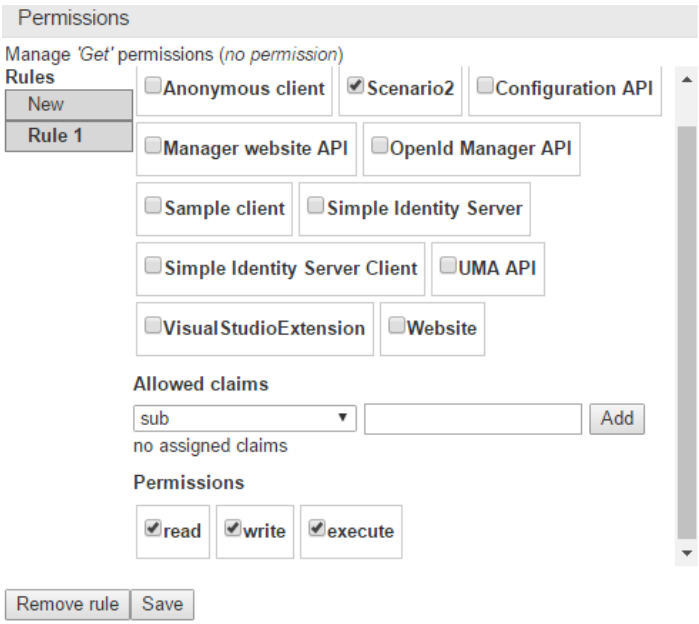
- SimpleIdentityServer : import *Settings / export-simpleidserver.json*
- IdentityServer: import *Settings / export-idserver.json*



Click on **resources** and navigate to the folder *Apis* > *ClientApi* > *v1* > *ClientsController*. Display the contextual menu of the resource *Get* and click on **Permissions**. In the new window add a new rule and persist the changes by executing the following actions :

1. Under allowed clients select *Scenario2*
2. Select the permissions *read, write, execute*
3. Click on **add rule**
4. Persist the changes by clicking on **save**

At the end you should obtain something like this :



Execute the application

In the solution explorer set *ClientApi* and *MarketingClient* as startup projects and run them by pressing **F5**. Open the URL http://localhost:5103/api/ratings with your preferred browser. The list of clients should be returned by the API :

```
[{"clientName":"client_1","value":5},{"clientName":"client_2","value":5}]
```

Scenarios

In this chapter we will focus on the most common problems that can be found in enterprises and we will try to solve them.

Samples can be found here : https://github.com/thabart/SimpleIdentityServer.Samples.git

A heavy application wants to access to a protected API

Sections:

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 view the list.

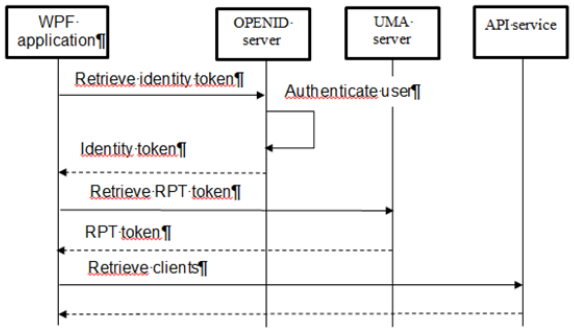
Problem

How the application can access to the protected operation ?

Solution

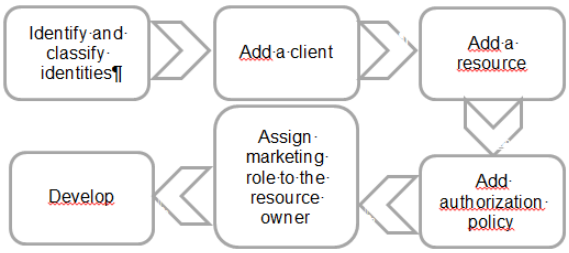
The workflow is made of three big steps

- **Identity token** : retrieve an identity token with implicit grant type. The token is returned to the client as a callback parameter.
- **RPT token** : the identity and access tokens (valid for the scope *uma_authorization*) are passed in the request to retrieve the RPT one. 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, this one is offered by the UMA server.



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

The decision table can help you to identify and classify the identities :

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

Result :

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

When the identities have been identified then they can be added.

Add a client

Add a new client and edit his properties. In the new window update as many properties as you can. Some parameters are rather easy to update like : displayed name and callbackurls, contrary to the *grant-types* parameter. If you can guess the grant-types then jump to the next section otherwise follow the methodology presented below to identify them.

Choose the grant-types

The grant-types must be chosen according to two factors :

- **Type of client** : API, website or WPF application
- **How the authentication page has been implemented** ? Redirect to the OpenId provider or create a login formula.

The relationships between grant-types and those factors are listed in the following table :

Application	Implementation	Grant type	+ / -
WebsiteWPF application	Redirect to OpenId provider	implicit	Delegate to openid No control over look & feel
	Login formula	password	Control look & feel Obfuscate source code Trust relationship
API		client_credentials	

Fill-in the parameters

Once the grant-types have been identified then the other parameter values can be deduced. Read the two following tables and pick-up the correct values.

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(*)
	website_api(*) (NS)
	uma(*) (NS)
Identity token	OpenId
	Profile
	Email
	Address
	Phone
	Role (NS)

Mappings between tokens and scopes

Legend

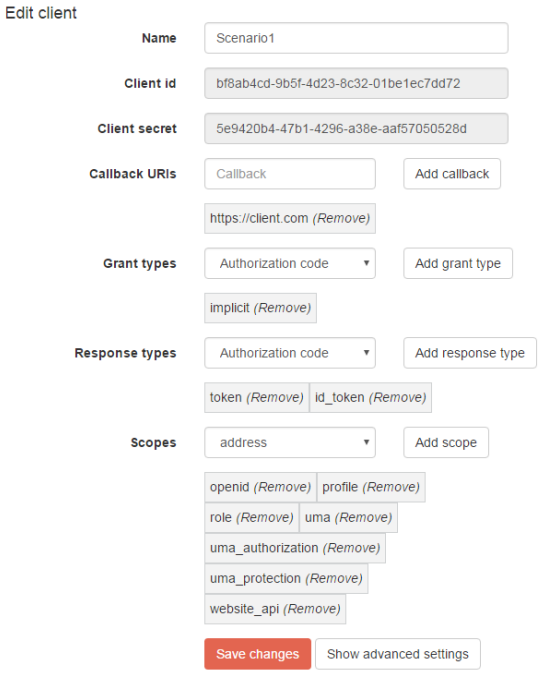
- (*) : mandatories scopes
- (NS) : not conformed to OPENID & UMA

Important: The scope **website_api** is required by the client to access to the WebSite API operations such as : retrieve a resource by its url.

The other scope **uma** is used by the protected API operations to introspect the received access token.

When all the parameters have been found then the edit page can be filled-in like this :

- Callback uris : https://client.com
- GrantTypes : implicit
- Response types : token and id_token
- Scopes: openid, profile, role, uma_authorization, uma_protection, uma, website_api



Add a resource

There are two different ways to add a resource, either with the website or either with the Visual Studio Extension. 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>
```

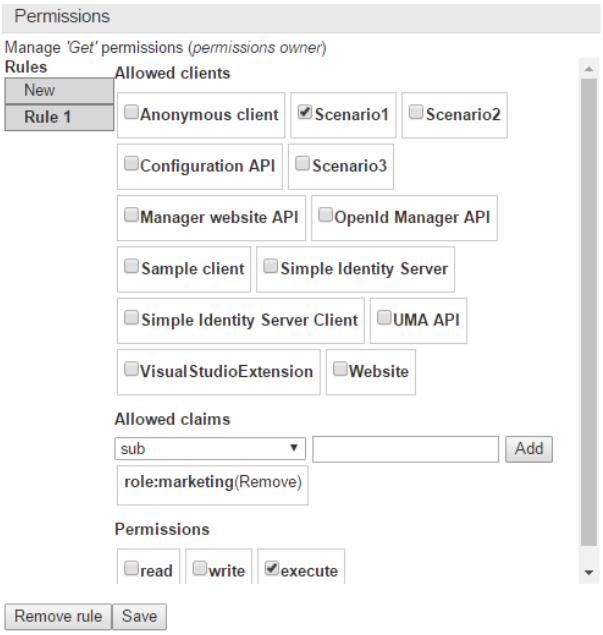
In our scenario the resource name is : "Apis > ClientApi > v1 > ClientsController > Get". If you are working with the Visual Studio Extension you don't have to be worried about the name because the convention is respected.

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 consumers of the resources will be impacted and they must be updated and redeployed again.

Add authorization policy

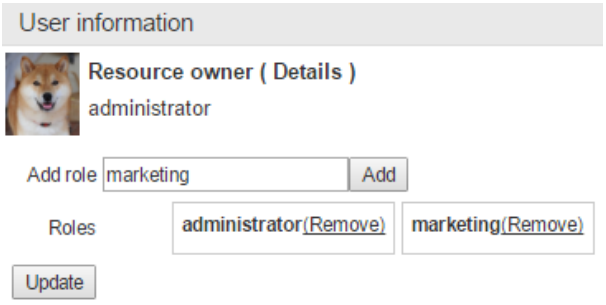
When the client and resource have been created then the authorization policy can be assigned.

- Allowed clients : Scenario1
- Allowed claims : role => marketing
- Permissions : execute



Assign marketing role to the resource owner

The marketing role must 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 changed.

API

There are two different kinds of authorization mechanisms :

- **Conventional**: the URL of the resource must match the structure of the project and also the API version. The last value can be set as a property "ConventionalUmaOptions.Versions".
- **Individual**: Limit the access to one specific resource by passing the URL and scopes

The Nuget packages : *SimpleIdentityServer.UmaIntrospection.Authentication* and *SimpleIdentityServer.Uma.Authorization* must be installed on your API project.

Enabling the conventional authorization is pretty straightforward. Insert the code below into the method *ConfigurationServices* of your 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>"));
  });
});
```

Then decorate the operation "ClientsController > Get" with the attribute : `[Authorize("uma")]`

WPF application

1. Add the Nuget package *SimpleIdentityServer.Proxy* to your client.
2. Retrieve an RPT token.

```
public static async Task<string> GetRptToken(
    string idToken,
    string umaProtectionToken,
    string umaAuthorizationToken,
    string resourceToken)
{
    var factory = new SecurityProxyFactory();
    var proxy = factory.GetProxy(new SecurityOptions
    {
        UmaConfigurationUrl = Constants.UmaConfigurationUrl,
        OpenidConfigurationUrl = Constants.OpenidConfigurationUrl,
        RootManageApiUrl = Constants.RootManageApiUrl
    });
    try
    {
        var result = await proxy.GetRpt("resources/Apis/ClientApi/v1/
↪ClientsController/Get", idToken, umaProtectionToken, umaAuthorizationToken,␣
↪resourceToken, new List <string>
        {
            "execute"
        });
        return result;
    }
    catch (Exception ex)
    {
        return null;
    }
}
```

3. Pass the RPT token to the Authorization header

```
var httpClient = new HttpClient();
var request = new HttpRequestMessage
{
    Method = HttpMethod.Get,
    RequestUri = new Uri("http://localhost:5100/api/clients")
};
request.Headers.Add("Authorization", $"Bearer {rptToken}");
var response = await httpClient.SendAsync(request);
```

An API wants to access to a resource

TODO

Limit access to certain features of the website

TODO

Use cases