OAuth2lib

http://tools.ietf.org/html/ietf-oauth-v2-10 implementation

2 Septiembre 2010

OAuth2 - Assertion Profile Library

OAuth2lib is a library based on OAuth2 that implements the Assertion Profile of the OAuth2 draft.

OAuth2 is the evolution of the OAuth protocol that defines authorization flows in order that a Client application is able to request resources from a resource Server on behalf of an specific user.

The authorization flows depends on the system in which it's deployed: web applications, desktop applications, mobile phones, and living room devices.

In this library we develop the authorization flow called 'Assertion Profile' in the OAuth2 protocol.

OAuth2lib is free software; you can redistribute it and/or modify it under the GNU Lesser General Public License terms. For more license information, search for the LICENSE file inside the code package.

Documentation

OAuth2 Assertion Flow

In order to understand the Assertion Flow, it's necessary to describe the following concepts:

- **OAuth Client:** Application that uses OAuth to access the OAuth Server on behalf of the user or on his own behalf.
- **OAuth Authorization Server:** HTTP server that with an OAuth Client's request is capable of issuing tokens that give access to the resources.
- **OAuth Server:** HTTP Server that has got the protected resources and denies or grants the access to them depending on the token given within the Client's request.
- **Token:**A string representing an access grant issued to the client that has to be delivered to the OAuth Server in order to access to the resource.

OAuth 2 provides a protected resources authorization's delegation to different Trust Authorities.

The Client, in order to access a protected resource, must obtain first an Authorization Server's authorization, in the form of *tokens*.

This token is obtained by sending some credentials to the Authorization Server.

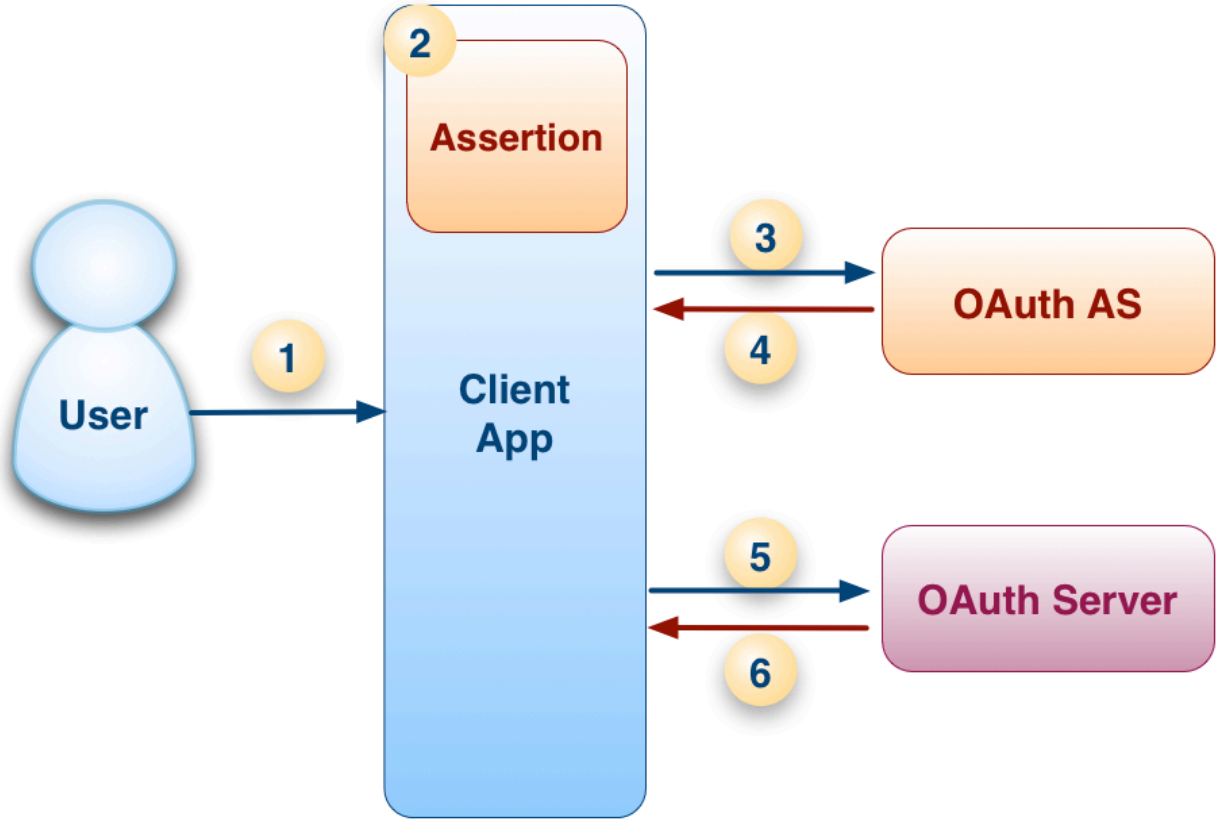
To get the resource, the client will have to give the resource Server the token obtained, and if it's a valid one, the Server returns the resource.

Authorization Flow

In this authorization flow, the peculiarity is that the credentials sent to the Authorization Server are assertions provided by an SP. So far, this library supports SAML2 and PAPI assertions.

The steps taken in order to obtain the protected resource are:

1. The user goes to a Client Application.
2. In the Client App, the user authenticates in an external SP that generates a SAML or PAPI assertion.
3. The Client App sends the assertion obtained to an Authorization Server. There, a token for a certain user, client, scope and lifetime is generated.
4. The Authorization Server sends the generated token to the Client App.
5. The Client App acts on behalf of the user and requests the resource to the Server. This token can be used more times until it expires.
6. The Server returns the resource if the token sent is a valid token.

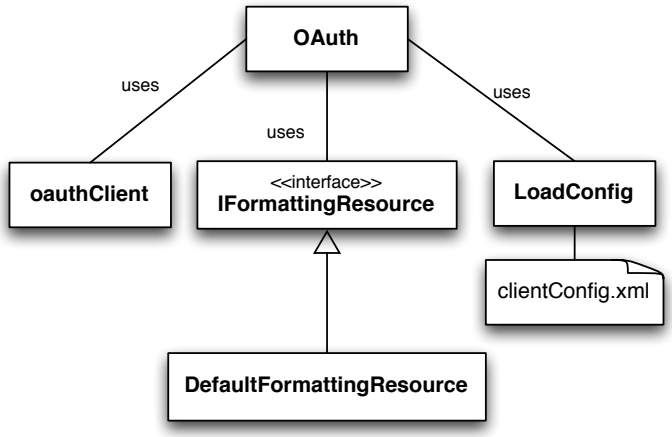

2
Assertion
3
OAuth AS
4
User
1
Client
App
5
OAuth Server
6

OAuth Client

The OAuth Client Application's goals are the following:

- To request a token from an Authorization Server with the obtained assertion.
- To request the resource on behalf of the user with the token.

OAuth Client's Architecture: 'oauth_client' directory:



'config' directory

- clientConfig.xml: File with the configuration of the ClientApp.

'src' directory

- OAuth.class.php: Configuration class that gives the developer an abstraction to the OAuth2 flow.
- OAuthClient.class.php: Class with the OAuth Client logic
- LoadConfig.class.php: Class that loads the Client App. Configuration.

`response_formats' directory

- DefaultFormattingResource.class.php: Example class of a formating resource class. It could exist different classes, depending on the scope of the request.
- IFormattingResource.class.php: Interface that models the resource depending on the response

Configuring the clientConfig.xml file

File wih the Client Application configuration. This configuration loads when the OAuth class is initialized. This parameters can be changed with the setters defined in the OAuth class.

The format of this file is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<OAuthClient>

    <!--Authorization Server Configuration-->
    <AuthServerConfig>
        <!-- Type of the assertion
           SAML2 = "urn:oasis:names:tc:SAML:2.0:assertion";
           PAPI = "urn:mace:rediris.es:papi";
        -->
        <AssertionType>urn:papi</AssertionType>
        <!--  The access grant type included in the request.
        In this library the type is "assertion". -->
        <GrantType>assertion</GrantType>
        <!-- Authorization Server URL.-->
        <AuthServerURL>
           url_of_the_auth_server/oauth_as/tokenEndpoint.php
           </AuthServerURL>
    </AuthServerConfig>

 <!--Resource Server Configuration*-->
    <ResServerConfig>
        <!--Format of the resource request:
            HEADER = "HTTP_Authorization_Header";
            GET = "URI_Query_Parameter";
            BODY = "Form-Encoded_Body_Parameter";
        -->
        <RequestType>
           HTTP_Authorization_Header
        </RequestType>
        <!--Resource Server URL.-->
        <ResServerURL>
           url_of_the_resource_server/oauth_server/serverEndpoint.php
        </ResServerURL>
        <ResponseFormats>
            <Scope id="http://www.rediris.es/sir/api/sps_available.php">
                <FormatClass>
                DefaultFormattingResource
                </FormatClass>
```

```
                <FormatFile>
                 /url/DefaultFormattingResource.class.php
                </FormatFile>
            </Scope>
        </ResponseFormats>
    </ResServerConfig>

 <!--Client App Configuration-->
    <ClientConfig>
        <ClientID>
           app_client_1
        </ClientID>
        <ClientSecret>
            example_key
        </ClientSecret>
        <!-- Error Response type.
        HTML or JSON.. -->
        <ErrorResponseType>
           HTML
        </ErrorResponseType>
        <DefaultScope>
           http://www.rediris.es/sir/api/sps_available.php
        </DefaultScope>
        <DebugActive>
           TRUE
        </DebugActive>
    </ClientConfig>
</OAuthClient>
```

Response Formats

We must give format to the different resources obtained. Each scope have one IFormattingResource Class defined.

The format of this archive will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseFormats>
    <Scope id="http://www.rediris.es/sir/api/sps_available.php">
        <FormatClass>DefaultFormattingResource</FormatClass>
        <FormatFile >DefaultFormattingResource.class.php</FormatFile
    </Scope>
</ResponseFormats>
```

Where:

- *Scope 'id'* will be the URI of the scope
- *FormatClass* will be the name of the class that implements IFormattingResource.
- *FormatFile* will be the name of the directory where it's defined the class that implements IFormattingResource.

DefaultFormattingResource Class

Description

Example class of a formating resource class. This formatting would depend on the scope of the request.

Implements IformattingResource Interface

Methods

public formatResource(services):String
Function that formats the default resource.

- String **services:** Default resource response.

IFormattingResource Interface

Description

Interface of a formating resource class.

Methods

public formatResource(services):String
Function that formats the resource.

- String **services:** Default resource response.

OAuth class

Description

Class that makes an abstraction to the OAuth Authorization Flow in order to simplify to developers the implementation.

Class Constants

- **HEADER:** HTTP_Authorization_Header
- **GET:** URI_Query_Parameter
- **BODY:** Form-Encoded_Body_Parameter
- **SAML2:** urn:oasis:names:tc:SAML:2.0:assertion
- **PAPI:** urn:mace:rediris.es:papi

- **HTML:** HTML
- **JSON:** JSON

Class Variables

- String **assertion_type:** Type of the assertion (Defined by the constants PAPI or SAML2). By default PAPI.
- String **client_id:** Client Identification
- String **client_secret:** Client Shared Secret
- Boolean **debug_active:** If true, the debug is active, inactive otherwise.
- String **error:** Error code-name
- String **error-type:** Error type. Defined by the constants HTML or JSON. By default HTML.
- String **as:** Authorization Server URL.
- String **rs:** Resource Server URL.
- String **request_type:** Type of request that the Client makes to the Resource Server (Defined by the constants HEADER, GET or BODY).By default HEADER.
- String **resource:** The obtained resource.
- String **grant_type:** The access grant type included in the request. In this library the type is "assertion".
- String **scope:** Scope of the request.
- String **conf:** LoadConfig object.

Methods

OAuth __construct ([$dir = ""])
Public OAuth Class Constructor.

- String **dir:** Directory where the client configuration is located.

Return an OAuth Object

PUBLIC doOAuthFlow($assertion): boolean
Function that gets the resource with an OAuth2 flow and stores it in the 'resource' parameter. (And it could be accesed by the method getResource)

Return a boolean: True if the flow went ok, false otherwise. The error description is stored in the 'error' parameter

- String **assertion:** Assertion provided

PRIVATE error($string): void
Function that shows the errors in the error_log if $debug_active is TRUE.

- String **string:** String showed in the error_log.

PUBLIC getAs(): string
Returns the Authorization Server URL.

PUBLIC getAssertion_type(): string
Returns the type of the assertion. It could be PAPI or SAML2.

PUBLIC getClient_id(): string
Returns the Client Identificator.

PUBLIC getClient_secret(): string
Returns the Client Secret.

PUBLIC getDefault_scope(): string
Returns the default_scope.

PUBLIC getError(): string
Returns the error description.

PUBLIC getError_type(): string
Returns the error type.

PUBLIC getGrant_type(): string
Returns the grant_type.

PUBLIC getRequest_type(): string
Returns the request type.

PUBLIC getResource(): string
Returns the resource.

PUBLIC getRs(): string
Returns the resource server URL.

PUBLIC getScope(): string
Returns the scope.

PUBLIC returnError($oauth): string
Function that given an OAuthClient object, formats the obtained error depending on the selected type in the OAuth class: If it is HTML returns an html with the message inside of the div element:

```
<div class="error"> $error_msg <div>
```

If it is JSON returns a json element with the following format:

```
{"error":"error_description"}
```

- OAuthClient **oauth:** OAuthClient object.

PUBLIC returnResource($oauth): string
Function that given an OAuthClient object, formats the corresponding response depending on the scope of the request. Returns a String with the formatted response.

- OAuthClient **oauth:** OAuthClient object.

PUBLIC setAs($url_as): void
Sets the Authorization Server URL

- String **url_as:** URL of the OAuth authorization server.

PUBLIC setBODYResourceRequest(): void
Sets the resource request type to a POST request.

PUBLIC setGETResourceRequest(): void
Sets the resource request type to a GET request.

PUBLIC setGrant_type($grant_type): void
Sets the grant_type with the parameter $grant_type.The access grant type must be one of "authorization-code", "basic-credentials", "assertion", "refresh-token", or "none".

- String **grant_type:** Grant type

PUBLIC setHEADERResourceRequest(): void
Sets the resource request type to a Authorization HEADER request.

PUBLIC setHTMLErrorResponse(): void
Set the error response type (error_type parameter) to HTML

PUBLIC setJSONErrorResponse(): void
Sets the error response type (error_type parameter) to JSON

PUBLIC setPAPIAssertionType(): void
Sets the assertion_type parameter to PAPI

PUBLIC setRs($url_rs): void
Sets the Resource Server URL.

- String **url_rs:** URL of the OAuth resource server.

PUBLIC setSAML2AssertionType(): void
Sets the assertion_type parameter to SAML2

PUBLIC setScope($scope): void
Sets the scope with the $scope parameter

- String **scope:** URI of the scope of the resource.

OAuthClient class

Description

Class with the OAuth Client Application logic.

Class Constants

- **HEADER:** HTTP_Authorization_Header
- **GET:** URI_Query_Parameter
- **BODY:** Form-Encoded_Body_Parameter

Class Variables

- String **access_token:** Access Token generated by the Authorization Server
- String **client_id:** Client Identification
- String **client_secret:** Client Shared Secret
- Boolean **debug_active:** If TRUE, the debug is active, inactive otherwise.
- String **error:** Error code-name
- Integer **expires_in:** Lifetime of the access token. 3600s by default.
- String **request_type:** Type of request that the Client makes to the Resource Server (Defined by the constants HEADER, GET or BODY).
- String **resource:** The obtained resource.
- String **scope-ret:** Scope parameter returned by the Authorization Server.

Methods

PUBLIC OAuthClient __construct($clientid, $clientsecret, [$debug = false])
OAuthClient class Constructor

- String **clientid:** Client Identification
- String **clientsecret:** Client Shared Secret.
- Boolean **debug:** If TRUE actives the debug. FALSE by default./li>

Return an OAuthClient Object

PRIVATE cleanHeader($string): string
Auxiliar function that clean the server response header.

Returns the cleaned response.

- String **string:** Header.

PRIVATE doAccessTokenRequest($as, $scope, $assertion, $assertion_type, [$grant_type = "assertion"]): Boolean
Function that mades theaccess token request to the AS.

Returns TRUE if the request obtained an Access Token, FALSE otherwise.

- String **as:** The Authorization Server URL
- String **scope:** The scope of the access request.
- String **assertion** The assertion
- String **assertion_type:** The format of the assertion as defined by the authorization server.
- String **grant_type:** The access grant type included in the request.

PRIVATE doATRequest($as, $request): Boolean
Makes the HTTP POST CURL connection to request the access token from the authorization server.

Stores the access token in the protected param 'access_token'. If an error occurs, it stores the error in the protected param 'error'.

Returns True if the Auth server response has an access token

- String **as:** The Authorization Server URL
- Array **request:** The request data

PRIVATE doBodyResRequest($rs, $request): Boolean
Makes connection to request the resource with a Form-Encoded Body Parameter.

When including the access token in the HTTP request entity-body, the client adds the access token to the request body using the "oauth_token" parameter. The entity-body can include other request-specific parameters, in which case, the "oauth_token" parameters SHOULD be appended following the request-specific parameters, properly separated by an "&".

Returns TRUE if the request obtained the resource, FALSE otherwise.

- String **rs:** The Resource Server URL
- Array **request:** The request data

PRIVATE doGetResRequest($rs, $request): Boolean
Makes connection to request the resource with a URI Query Parameter.

When including the access token in the HTTP request uri, the client adds the access token to the request URI query component as defined by [RFC3986] using the "oauth_token" parameter. The HTTP request URI query can include other request-specific parameters, in which case, the "oauth_token" parameters SHOULD be appended following the request-specific parameters, properly separated by an "&".

Returns TRUE if the request obtained the resource, FALSE otherwise.

- String **rs:** The Resource Server URL
- Array **request:** The request data

PRIVATE doHeaderResRequest($rs, $request): Boolean
Makes connection to request the resource with an Authorization Request Header Field.

The "Authorization" request header field is used by clients to make authenticated token requests. The client uses the "token" attribute to include the access token in the request.

Returns TRUE if the request obtained the resource, FALSE otherwise.

- String **rs:** The Resource Server URL

- Array **request:** The request data

PRIVATE doResourceRequest($rs, $request_type, $request): Boolean
Function that makes the resource request to the Resource server.

Returns TRUE if the request obtained the resource, FALSE otherwise.

- String **rs:** The Resource Server URL
- Array **request:** The request data
- String **request_type:** The request type. It could be GET, BODY, or HEADER.

PRIVATE error($string): void
Function that shows the errors in the error_log if $debug_active is TRUE.

- String **string:** String showed in the error_log.

PRIVATE generateATRequest($scope, $assertion, $assertion_type, $grant_type): Array
Generates an access token request.

Returns the request Array.

- String **scope:** The scope of the access request.
- String **assertion** The assertion
- String **assertion_type:** The format of the assertion as defined by the authorization server.
- String **grant_type:** The access grant type included in the request.

PRIVATE generateResourceRequest($extra): Array
Generates the array of the resource request.

Returns the parameters of the request.

- Array **extra** Array with extra parameters.

PUBLIC getAccess_token(): String
Returns the obtained access token.

PUBLIC getExpires_in(): Integer
Returns the lifetime of the token.

PUBLIC getHTMLError(): String
Returns the error in HTML format.

PUBLIC getJSONError(): String
Returns the error in JSON format.

PUBLIC getResource(): string
Returns the resource.

PRIVATE isntHTTPS($url): Boolean
Function that checks if and url is https or http

Returns TRUE if it is http, FALSE if it is https.

- String **url:**URL to check.

PRIVATE processAuthServerResponse($info, $output): Boolean
Manages the Auth server response.

Returns TRUE if the Auth server response has an access token, FALSE otherwise.

- Array **info:**Info of the CURL response.
- String **output:**Output of the CURL response.

PRIVATE processResServerResponse($info, $output): Boolean
Manages the resource server response.

Returns TRUE if the Auth server response has a resource, FALSE otherwise.

- Array **info:**Info of the CURL response.
- String **output:**Output of the CURL response.

PUBLIC requestResource($rs, $request_type, [$extra = null])): Boolean
Function that manages the request to the resource server.

Returns TRUE if the request obtained the resource, FALSE otherwise.

- String **rs:** The Resource Server URL
- Array **extra:** Extra parameters added in case of necessity. Initialized by default to null.
- String **request_type:** The request type. It could be GET, BODY, or HEADER.

OAuth AS

The OAuth Authorization Server's goals are the following:

- Given an assertion, to check if it's a valid one and to generate a token for an specific scope.

OAuth Authorization Server's Architecture: 'oauth_as' directory

- tokenEndpoint.php**:** Authorization Server Endpoint

'config' directory

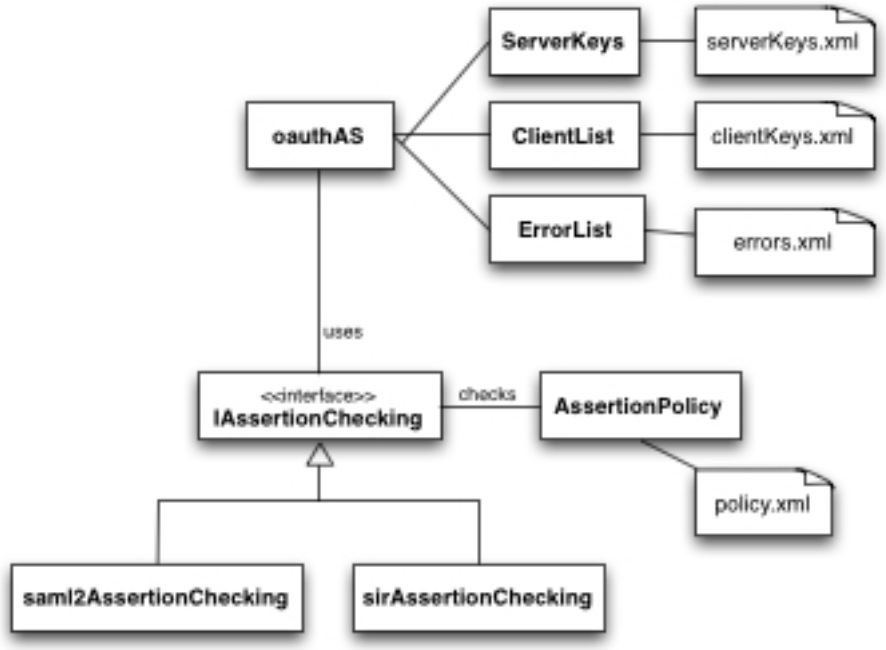
- clientKeys.xml**:** File with the reference of the registered client.
- errors.xml**:** File with the reference of the errors supported by the OAuth2 protocol for the OAuth Authorization Server.
- policies.xml**:** File with the reference oth the Authorization policies.
- serverKeys.xml: File with the reference of the registered servers.

'src' directory

- oauthAS class: Class that has the logic of the Authorization Server. It generates the token and manage the Client's requests.
- ClientList class: Class that permit to load the client list from *keys.xml*.
- ErrorList class: Class that permit to load the errors list from *errors.xml*.
- Server Keys class: Class that permit to load the server list of related scopes and keys from serverKeys.xml.

'assertions' directory

- AssertionPolicy class: Class that manages the policy.xml file, wich one has got the authorization policy for each type of assertion.
- IAssertionChecking interface: Interface that defines the methods for an Assertion Checking class.
- saml2AssertionChecking: Class that checks if an assertion in saml2 format fulfill the defined policies.
- sirAssertionChecking:Class that checks if an assertion in saml2 format fulfill the defined policies.



Configuring the Keys.xml file

We must give the Client Applications a shared secret that will be used to reinforce the security of the application. This shared secret or *key* will be registered in the `keys.xml` file. The format of this archive will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<Keys>
<Key id="client_id" value="key"/>
<Key id="client_id2" value="Key2"/>
</Keys>
```

Where:

- *client* will be the identificator of the Client Application
- *key* will be the shared secret.

This shared secret has to be the same in the keys.xml file of the Authorization Server and of the Resource Server.

OAuth AS Policy

Definition of the authorization policy xml file

This xml file define what kind of user can access to the resources.

The main structure looks like this one:

```
<AssertionList>
  <Assertion ...>
                .
                .
```

```
  </Assertion>
</AssertionList>
```

Where,

- AssertionList: this element defines the list of assertion types.
- Assertion: this element defines the policies for an specific assertion.

Defining an Assertion Policy

The next example shows how can be defined an authorization policy for an specific assertion.

SAML2 Assertion example

```
<Assertion type="saml2">
    <Policies>
        <Policy>
            <Attributes check="all" >
               <Attribute name="def:eduPersonScopedAffiliation"
value="staff@rediris.es" />
            </Attributes>
        </Policy>
    </Policies>
</Assertion>
```

PAPI Assertion example

```
<Assertion type="saml2">
    <Policies>
        <Policy>
            <Attributes check="all" >
               <Attribute name="ePA" value="staff" />
            </Attributes>
        </Policy>
        <Policy>
            <Attributes check="any" >
               <Attribute name="sHO" value="rediris.es" />
               <Attribute name="sHO" value="fecyt.es" />
            </Attributes>
        </Policy>
    </Policies>
</Assertion>
```

Where,

- Inside the element <Policies>, you can add one element <Policy> per policy that you may need.

- The element <Attributes> inside a <Policy> defines how the user's attributes should be checked. This is specified in its attribute check, where allowed values are:
    - all: it says that all the rules have to be satisfied.
    - any: it says that one or more rules have to be satisfied.
    - none: it says that none of all rules has to be satisfied.
- The element <Attribute> inside the one <Attributes> defines a rule over the user's attributes, where name is the name of the attribute and value is its value.

Configuring the ServerKeys.xml file

We must give the Authorization Servers a Resource Server's shared secret or *key* that will be used to encrypt the token and reinforce the security of the resource servers. These keys will be registered in the `serverKeys.xml` file. The format of this archive will be:

```
<AuthServer id="authServerID" url="authServerURL">
          <ResourceServer id="res_serv_id1">
              <Scopes>
                  <Scope>scope1</Scope>
                  <Scope>scope2</Scope>
              </Scopes>
              <Key>example_key</Key>
          </ResourceServer>

           <ResourceServer id="res_serv_id2">
              <Scopes>
                  <Scope>scope2_1</Scope>
                  <Scope>scope2_2</Scope>
                  <Scope>scope2_3</Scope>
              </Scopes>
              <Key>example_key2</Key>
          </ResourceServer>
</AuthServer>
```

Where *ResourceServers* contains a set of resource servers. Each of one will have a set of *Scopes* and a *Key* that will be the shared secret.

This shared secret has to be the same in the asKeys.xml file of the Resource Server.

ServerKeys Class

Description

Class that load the server keys and scopes list from the serverKeys.xml file.

Class Variables

- Array **keys:** List of keys ordered by scopes.

Methods

PUBLIC ServerKeys __construct([$dir = ""])
ServerKeys class Constructor

- String **dir:** Directory where the client configuration is located.

PUBLIC hasScope($scope): boolean
Function that returns TRUE if exist the scope send in the parameter.

- String **scope:** Scope of the request.

PUBLIC getKey($scope): void
Function that returns the key of the related scope send in the parameter.

- String **scope:** Scope of the request.

PRIVATE loadKeys($file): string
Auxiliar function that loads the keys and scopes from the file.

- String **file:** file where the keys are defined

OAuthAS Class

Description

Class with the OAuth Authorization Server logic.

Class Constants

- **SAML2:** urn:oasis:names:tc:SAML:2.0:assertion
- **PAPI:** urn:mace:rediris.es:papi

Class Variables

- String **error:** Error code-name
- Boolean **debug_active:** If TRUE, the debug is active, inactive otherwise.
- ClientList **clients:** ClientList object
- ServerKeys **servers:** ServerKeys object
- String **assertion:** The assertion of the request.
- String **assertion_type:** Type of the assertion of the request.
- String **access_token:** The access token generated.

- ErrorList **errors:** ErrorList object
- String **scope:** Scope of the Request.
- String **client_id:** Client Identification
- IAssertionChecking **assertion_checking:** IAssertionChecking element.
- Integer **lifetime:** the default lifetime of the access tokens.

Methods

PUBLIC oauthAS __construct([$dir=””])
oauthAS class Constructor

- String **dir:** Directory where the configuration files are located.

Return an OAuthAS Object

PRIVATE error($string): void
Function that shows the errors in the error_log if $debug_active is TRUE.

- String **string:** String showed in the error_log.

PUBLIC getError(): string
Returns the error description.

PRIVATE generateAccessToken(): void
Function that generates an access token from the parameters of the request.

PRIVATE isValidAssertion(): boolean
Function that ckecks the assertion depending of the assertion type (SAML2, PAPI).

TRUE if is a valid one, FALSE otherwise.

PRIVATE isValidClient(): boolean
Function that ckecks if the OAuth Client making the request is registered.

TRUE if is a valid one, FALSE otherwise.

PRIVATE isValidFormatRequest(): boolean
Function that ckecks if the format request of the OAuth Client is valid.

TRUE if is a valid one, FALSE otherwise.

PRIVATE isValidScope(): boolean
Function that ckecks if the Scope of the request is authorized for the user.

TRUE if is a valid one, FALSE otherwise.

PRIVATE manageASErrorResponse(): string

Responds an error If the token request is invalid or unauthorized by adding the following parameter to the entity body of the HTTP response using the "application/json" media type with the following format:

- *error* REQUIRED. A single error code
- *error_description* OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.
- *error_uri* OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the end-user with additional information about the error.

PRIVATE manageASResponse(): string
Function that returns the resource, making use of the Resource Class deployed in the server.

PUBLIC manageRequest(): string
Function that manages the request of the app client and return an appropiate response.

ClientList Class

Description

Class that load the client list from the keys.xml file.

Class Variables

- Array **clients:** List of clients.

Methods

PUBLIC ClientList __construct([$dir = ""])
ClientList class Constructor

- String **dir:** Directory where the configuration files are located.

PUBLIC isClient($client_id): void
Function that returns TRUE if exist the client identification send in the parameter.

- String **client_id:** Client App ID.

PUBLIC getSecret($client_id): void
Function that returns the Secret of the client send in the parameter.

- String **client_id:** Client App ID.

PRIVATE loadClients($file): string
Auxiliar function that loads the clients from the file.

- String **file:** file where the clients are defined

ErrorList Class

Description

Class that load the error list.

Class Variables

- Boolean **debug_active:** If TRUE, the debug is active, inactive otherwise.
- Array **errors:** List of error's.
- Array **uris:** List of error's URIs.
- Array **descriptions:** List of error's descriptions.

Methods

PUBLIC ErrorList __construct([$dir=””])
ErrorList class Constructor

- String **dir:** Directory where the configuration is located.

PRIVATE error($string): void
Function that shows the errors in the error_log if $debug_active is TRUE.

- String **string:** String showed in the error_log.

PUBLIC hasError($error): void
Function that returns TRUE if the exist the error code send in the parameter.

- String **error:** Error code defined in the OAuth2 protocol.

PUBLIC hasURI($error): void
Function that returns TRUE if the exist the URI of the error code send in the parameter.

- String **error:** Error code defined in the OAuth2 protocol.

PUBLIC hasDescription($error): void
Function that returns TRUE if the exist the Description of the error code send in the parameter.

- String **error:** Error code defined in the OAuth2 protocol.

PUBLIC getDescription($error): void

Function that returns the Description of the error code send in the parameter.

- String **error:** Error code defined in the OAuth2 protocol.

PUBLIC getURI($error): void
Function that returns the URI of the error code send in the parameter.

- String **error:** Error code defined in the OAuth2 protocol.

PRIVATE loadErrors($file): string
Auxiliar function that loads the errors from the file.

- String **file:** file where the errors are defined

IAssertionChecking Interface

Description

Interface that must implement the class that checks an assertion.

Methods

PUBLIC getError():String
Function that gets the error. Returns null if there's no error.

PUBLIC checkAssertion(assertion):String
Function that checks if the assertion is a valid one.

- String **assertion:** The assertion.

sirAssertionChecking Class

Description

Class that checks a PAPI assertion.

Implements IAssertionChecking Interface

Methods

PUBLIC getError():String
Function that gets the error. Returns null if there's no error.

PUBLIC checkAssertion(assertion):String
Function that checks if the PAPI assertion is a valid one.

- String **assertion:** The PAPI assertion.

saml2AssertionChecking Class

Description

Class that checks a SAML2 assertion.

Implements IAssertionChecking Interface

Methods

PUBLIC getError():String
Function that gets the error. Returns null if there's no error.

PUBLIC checkAssertion(assertion):String
Function that checks if the SAML2 assertion is a valid one.

- String **assertion:** The SAML2 assertion.

OAuth Server

The OAuth resource Server's goals are the following:

- Given a token, to check if it's a valid one and to return the requested resource.

OAuth Resource Server's Architecture:

'oauth_server' directory

- **serverEndpoint.php:** Resource Server Endpoint

'config' directory

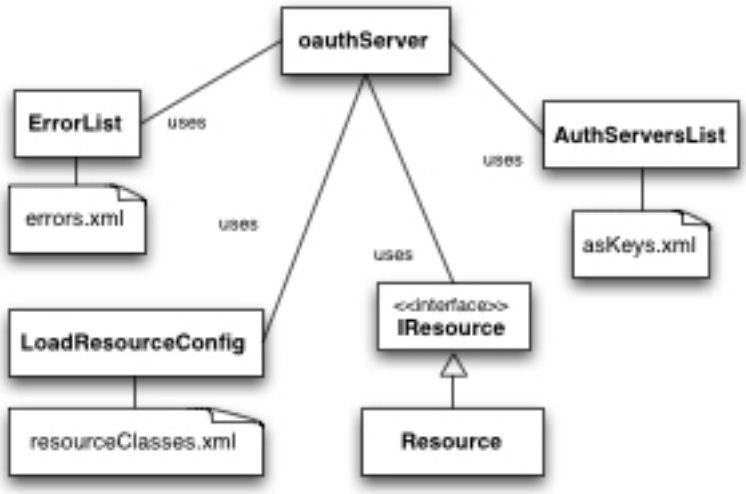
- **asKeys.xml:** Fiile with the reference of the registered Authorization Servers.
- **errors.xml:** File with the reference of the errors supported by the OAuth2 protocol for the OAuth Authorization Server.
- **resourceClasses.xml**: File with the reference of the registered resources and the location of the files where they are defined

'src' directory

- oauthRS class: Class that has the logic of the Resource Server. It checks the token and manage the Client's requests.
- AuthServerList class: Class that permit to load the authorization server list from *asKeys.xml*.
- ErrorList class: Class that permit to load the errors list from *errors.xml*.
- LoadResourceConfig class: Class that permit to load the resources config from resourceClasses*.xml*.

'resources' directory

- IServerResource interface: nterface that defines the methods for a Resource class.
- Resource class: Class that gets the resource if the token is a valid one. Implements the interface IResource.



Configuring the asKeys.xml file

Each Resource Server will have a shared secret or *key* that will be used to encrypt the token and reinforce the security of the resource servers. These keys will be registered in the `asKeys.xml` file. The format of this archive will be:

```
<AuthServers>
 <AuthServer id="authServerID" url="authServerURL">
      <Key>example_key</Key>
  </AuthServer>
  <AuthServer id="authServerID2" url="authServerURL2">
      <Key>example_key2</Key>
  </AuthServer>
</AuthServers>
```

Where *AuthServers* contains a set of authorization servers. Each of one will have a *Key* that will be the shared secret.

This shared secret has to be the same in the authServers.xml file of the Resource Server.

OAuthRS Class

Description

Class with the OAuth Resource Server logic.

Class Variables

- Boolean **debug_active:** If TRUE, the debug is active, inactive otherwise.
- String **error:** Error code-name
- String **resource:** The obtained resource.
- String **scope:** Scope of the Request.
- AuthServerList **authservers:** AuthServerList object
- ErrorList **errors:** ErrorList object
- Array **extra:** Extra parameters send in the Client's request.
- String **token:** The access token of the request.

Methods

PUBLIC oauthRS __construct([$dir=””])
oauthRS class Constructor

- String **dir:** Directory where the configuration files are located.

Return an OAuthRS Object

PRIVATE checkPersonScope($person_id): string

Function that checks if the scope included in the request is a valid one.

TRUE if is a valid one, FALSE otherwise.

- String **$person_id:** Identifier of the user.

PRIVATE error($string): void
Function that shows the errors in the error_log if $debug_active is TRUE.

- String **string:** String showed in the error_log.

PRIVATE isValidFormatGETorPOSTRequest($request): string
Function that checks if the request (GET or POST) is a valid one.

TRUE if is a valid one, FALSE otherwise.

- String **$request:** The GET or POST request data.

PRIVATE isValidFormatHeaderRequest($request): string
Function that checks if the request (Authorization Header) is a valid one.

TRUE if is a valid one, FALSE otherwise.

- String **$request:** The HEADER request data.

PRIVATE isValidFormatRequest(): string
Function that checks the format of the request, depending on the method: GET, POST or Authorization Header.

TRUE if is a valid one, FALSE otherwise.

PRIVATE isValidToken(): string
Function that checks if the token given in the request is a valid one.

TRUE if is a valid one, FALSE otherwise.

PRIVATE manageRSErrorResponse(): string
Function that manage a negative response. If the error is insufficient_scope, sends a HTTP 403. If the error is a invalid_request, sends a HTTP 400. If the error is a invalid_token, sends a HTTP 401. Other types of errors returns an HTTP 401.

Uses the "application/json" media type with the following format:

- *error* REQUIRED. A single error code
- *error_description* OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.
- *error_uri* OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the end-user with additional information about the error.

PRIVATE manageRSResponse(): string
Function that returns the resource, making use of the Resource Class deployed in the server.

PUBLIC manageRequest(): string
Function that manages the request of the app client and return an appropiate response.

Checks the format of the request depending on the method: GET, POST or header and if the given token is a valid one.

Returns a string with an Error or a Resource

IServerResource Interface

Description

Interface that must implement the class that obtains the resource.

Methods

public getResource(scope, extra):String
Function that gets the resource described by an scope.

- String **scope:** Scope of the resource request.
- Array **extra:** Extra parameters that we may need to get the resource.

public checkScope(scope, person_id):String
Function that checks if the user (person_id) is authorized to get the resources described by an scope.

- String **scope:** Scope of the resource request.
- String **person_id:** ID of the user.

ServerResource Class

Description

Example class of a resource class.

Implements IServerResource Interface

Methods

public getResource(scope, extra):String
Function that gets the resource described by an scope.

- String **scope:** Scope of the resource request.
- Array **extra:** Extra parameters that we may need to get the resource.

public checkScope(scope, person_id):String
Function that checks if the user (person_id) is authorized to get the resources described by an scope.

- String **scope:** Scope of the resource request.
- String **person_id:** ID of the user.

AuthServerLists Class

Description

Class that load the Authorization Server's keys from the asKeys.xml file.

Class Variables

- Array **keys:** List of keys.

Methods

PUBLIC AuthServerLists __construct([$dir= ""])
AuthServerLists class Constructor

- String **dir:** Directory where the configuration files are located.

PUBLIC checkTokenKey($token,$digest): boolean
Function that returns FALSE if doesn't decrypt the message with the registered Authorization Server's keys or TRUE otherwise.

- String **token:** plain token information.
- String **digest:** encrypted token information.

PRIVATE loadASs($file): string
Auxiliar function that loads the keys from the file.

- String **file:** file where the keys of the Authorization Servers are defined

Installation Guide

Download the code of oauth2lib v12 here or by command line with SVN:

```
svn checkout https://forja.rediris.es/svn/oauth2lib
```

Package Structure

- PAPI_oauth2_client directory: Client Application example with PAPI configuration.
- SAML_oauth2_client directory: Client Application example with SAML configuration.
- oauth_as directory: OAuth2 Authorization Server code.
- oauth_server directory: OAuth2 Resource Server code.
- oauth_client directory: OAuth2 Client code.

To implement one of the elements above you must include in your PHP library directory the oauth2lib code (directories: 'oauth_as', 'oauth_server' and 'oauth_client_directory').

OAuth Client

Structure

- *config* directory: Templates of the configuration files.
  - clientConfig.template.xml
- *src* directory:
  - LoadConfig.class.php
  - OAuth.class.php
  - OAuthClient.class.php
  - *response_formats* directory:
    - DefaultFormattingResource.class.php Example code with classes that implement the IFormattingResource Interface
    - IFormattingResource.class.php

Configuration

Step1: Instantiate the OAuth class.

In your application code, you must create an OAuth object.

```
$client = new OAuth($dir);
```

Where $dir will be the absolute path to the directory where is located your configuration file. For example:

```
$client = new OAuth(dirname(__FILE__)."/own_config/");
```

Note: The default path is the 'config' directory that we could find in the oauth_client directory.

Step2: Configure the parameters of the OAuth Client class

We load the default configuration parameters with the `clientConfig.xml file`.

We can modify this parameters dynamically with the OAuth class, too. These are defined in the client's documentation and are the following:

- Authorization Server
  - PUBLIC setAs($url_as): void
- Resource Server
  - PUBLIC setRs($url_rs): void
- Scope
  - PUBLIC setScope($scope): void
- Assertion Type: Type of the assertion (Defined by the constants PAPI or SAML2). By default PAPI.
  - PUBLIC setPAPIAssertionType(): void
  - PUBLIC setSAML2AssertionType(): void
- Error Type: Error type. Defined by the constants HTML or JSON. By default HTML.
  - PUBLIC setHTMLErrorResponse(): void
  - PUBLIC setJSONErrorResponse(): void
- Request type: Type of request that the Client makes to the Resource Server (Defined by the constants HEADER, GET or BODY). By default HEADER.
  - PUBLIC setBODYResourceRequest(): void
  - PUBLIC setGETResourceRequest(): void
  - PUBLIC setHEADERResourceRequest(): void
- Client Secret and Client ID: Defined in the constructor of the OAuth Class
  - OAuth __construct ([$file = ""])

The dynamic configuration will be made with these methods. For example:

```
$as="https://oauth-server.rediris.es/oauth2_12/oauth_as/
tokenEndpoint.php";
$rs="https://oauth-server.rediris.es/oauth2_12/oauth_server/
serverEndpoint.php";
$client->setAs($as);
$client->setRs($rs);
$client->setGETResourceRequest();
$client->setJSONErrorResponse();
$client->setSAML2AssertionType();
```

Step3: Start the authorization flow

```
$dev = $client->doOAuthFlow($assertion)
```

Where $assertion will be the specific assertion for an user. It could be an PAPI assertion or a SAML2 assertion.

In the case of a PAPI assertion, we send in $assertion the information stored in `$_SESSION['userdata']`, obtained with the phpPoA. For more information about phpPoA, see its web page

In the SAML2 case, we send in $assertion the result of call the method `$as->getAttributes();` of SimpleSAMLphp. For more information about SimpleSAMLphp, see its web page

Step4: Getting the resource (or the error)

```
if(!$dev){
      echo $client->getError();
}else{
      echo $client->getResource();
}
```

We get the resource with the method getResource().

To know if exists an error, we must check if the result of the *doOAuthFlow* method returns FALSE or not.

To know which error has happened, we can use the getError() method, that returns a string with the information.

Step5: Formatting the resource

To show the resources to the user, you must implement a class to visualize them properly.

For example, in the use case example, we've implemented a method that gets the resource, an xml string, and format it properly, to show the information in a readable way.

We can format the resources in different ways depending on the scope of the resource, thanks to the method returnResource() of the OAuth class. This function, given an OAuthClient object, formats the corresponding response depending on the scope of the request.

In order to format a Response we must do the following steps:

Step 5.1: Formatting the resource
To format a resource you must to develop a Class that implements the interface IFormattingResource. An example of this type of class is the DefaultFormattingResource.

Step 5.2: Register the format to an specific Scope
To register the format, you must configure the format responses by scopes in the clientConfig.xml.

OAuth Authorization Server

Structure

- *config* directory: Templates of the Authorization Server configuration files.
  - clientKeys.template.xml
  - serverKeys.template.xml
  - errors.template.xml
  - policies.template.xml
- *src* directory:
  - oauthAS class
  - ServerKeys class
  - ClientList class
  - ErrorList class
  - *assertions* directory:
    - AssertionPolicy class:
    - IAssertionChecking interface
    - saml2AssertionChecking
    - sirAssertionChecking
- *tokenEndpoint.php*

Configuration

Step1: Configuring the file serverKeys.xml
The Authorization Server has to know the Servers that are going to read its tokens. To define this, we use the serverKeys.xml file.

Step2: Configuring the file clientKeys.xml
The Authorization Server has to know with whose Clients are communicating. To define this, we use the keys.xml file.

Step3: Configuring the file policies.xml
The Authorization Server has to know which assertions are valid ones. To define the authorization policy, we use the policies.xml file.

Step4: Endpoint
With the current code organization, the Token Endpoint that we must give to the Client App will be accessible in the php file: tokenEndpoint.php. In this step you will need to change the directory where your configuration is located, for example:

```
$config_dir = dirname(__FILE__)."/own_config/";
$as = new oauthAS($config_dir);
```

OAuth Resource Server

Structure

- *config* directory: Templates of the Resource Server configuration files.
  - asKeys.template.xml
  - errors.template.xml
  - resourceClasses.template.xml
- *src* directory:
  - oauthRS class
  - AuthServerList class
  - LoadResourceConfig class
  - ErrorList class
  - *resources* directory:
    - Resource Template class
    - IServerResource interface
- *serverEndpoint.php*

Configuration

Step1: Configuring the file asKeys.xml
The Resource Server has to know with whose Authorization Servers are communicating. To define this, we use the asKeys.xml file.

Step2: To get the resources
To get the resources from the server, you must develop a class to get them.

This class must implement the IServerResource interface.

Step3: Configuring the file resourceClasses.xml
To get the correct resources for each request, you must configure with scope is related to which resource.

To define this, we use the resourceClasses.xml file.

Step4: Endpoint
With the current code organization, the Server Endpoint that we must give to the Client App will be accessible in the php file: serverEndpoint.php. In this step you will need to change the directory where your configuration is located, for example:

```
$config_dir = dirname(__FILE__)."/own_config/";
$rs = new oauthRS($config_dir);
```