

---

Article

[Daniel Kutac](#) · Aug 10, 2016 22m read

## InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2

Created by Daniel Kutac, Sales Engineer, InterSystems

Warning: if you get confused by URLs used: the original series used screens from machine called dk-gs2016. The new screenshots are taken from a different machine. You can safely treat url WIN-U9J96QBJSG as if it was dk-gs2016.

Part 2. Authorization server, OpenID Connect server

In the [previous part](#) of this short series, we have learned about simple use case – acting as an OAUTH[1] client. Now, it ' s time to bring our experience to a whole new level. We are going to build much more complex environment, where InterSystems IRIS is going to play all OAUTH roles.

We know already how to make a client, so let ' s concentrate on authorization server and even more, the OpenID Connect[2] provider.

As in the previous part, we need to prepare environment. This time it is going to be trickier, as there are more moving parts.

Before we go into the details of our example, we need to spend a few words about OpenID Connect.

As you may recall from previous part, we were asked – in order to be authorized by Google – to authenticate ourselves with Google first. The authentication is not part of OAUTH framework. In fact, there are many authentication frameworks around, independent of OAUTH. One of them is called OpenID. Started originally as an independent initiative, it recently leverages infrastructure provided by OAUTH framework, namely communication and data structures. Thus, OpenID Connect was born. In fact, many people call it OAUTH on steroids. Indeed, with OpenID Connect, you can not only authorize, but also authenticate using well known interfaces of OAUTH framework.

## Complex OpenID Connect demo

We will leverage much of the client code from the part 1. This saves us a lot of work, so we can concentrate on setting up environment.

### Prerequisites

This time we need to add, to already existing web server with SSL enabled, a PKI infrastructure. We need some cryptography required by OpenID Connect. If you want authenticate someone, you want to be absolutely sure that no-one else can impersonate the agent (client, auth server,...) who sends his/her confidential data over the network. This is where X.509 based cryptography comes in.

Please note: beginning with Cache 2017.1, it is no more needed to use X.509 Certificates to generate JWT / JWKS (JSON Web Key Set). We, for the backward compatibility and simplicity, use this option.

PKI

Strictly said, we do not need to use Caché PKI infrastructure at all, but it is more convenient way than using tools like openssl directly to generate all certificates.

We are not going into details of generating certificates here, as you can find details either within InterSystems IRIS [documentation](#) or elsewhere. As a result of generating certificates, we will create 3 public/private key pairs and associated certificates.

Let ' s call them

- rootca (rootca.cer) for our issuing certification authority
- auth (auth.cer and auth.key) for authorization & OpenID server
- client (client.cer and client.key) for client application server

## X.509 Credentials

We need to define X.509 credentials at individual servers so they can sign and validate JSON Web Tokens (JWT) exchanged during our demo

### Authorization & Authentication server configuration

Without going into details about how to define X.509 Credentials, we just show a screenshot of AUTHSERVER instance credentials.

**X.509 Credentials** Server: **dk-gs2016** Namespace: %SYS  
User: **dkutac** Licensed to: **ISC Sales Engineering** Instance: **AUTHSERVER**

Create New Credentials

The following sets of X.509 credentials are available to encrypt, decrypt, sign, and verify content (primarily for use w

Page size: 0 Max rows: 1000 Results: 2 Page: |< << 1 >> >| of 1

Alias	Owner List	Peer Names	Has Private Key	CAFile		
AuthServerConfig			1	C:\InterSystems\Cache\AUTHSERVER\mgr\cache.cer	<a href="#">Edit</a>	<a href="#">Delete</a>
ClientConfig			0	C:\InterSystems\Cache\AUTHSERVER\mgr\cache.cer	<a href="#">Edit</a>	<a href="#">Delete</a>

As image indicates, the AUTHSERVER owns its private key and certificate , whilst it only has certificate with public key of the CLIENT

### Client server configuration

And similarly, credentials defined at CLIENT instance

**X.509 Credentials** Server: **dk-gs2016** Namespace: %SYS  
User: **UnknownUser** Licensed to: **ISC Sales Engineering** Instance: **CLIENT**

Create New Credentials

The following sets of X.509 credentials are available to encrypt, decrypt, sign, and verify content (primarily for use

Page size: 0 Max rows: 1000 Results: 2 Page: |< << 1 >> >| of 1

Alias	Owner List	Peer Names	Has Private Key	CAFile		
AuthServerConfig			0	C:\InterSystems\Cache\CLIENT\mgr\cache.cer	<a href="#">Edit</a>	<a href="#">Delete</a>
ClientConfig			1	C:\InterSystems\Cache\CLIENT\mgr\cache.cer	<a href="#">Edit</a>	<a href="#">Delete</a>

Here the CLIENT owns private key and certificate, but only certificate with public key of AUTHSERVER.

## Resource server configuration

We do not need to define X509 credentials at RESSERVER instance in our example setup.

## OAUTH Configuration

Similarly to configuration described in part 1 of this series, we need to configure our servers for OAUTH. Let ' s start with AUTHSERVER instance, as this is the central component in overall OAUTH configuration.

### AUTHSERVER

In System Management Portal, navigate to System Administration > Security > OAuth 2.0 > Server Configuration.

Click the menu link and fill form items:

- host name
- port (optional)
- prefix (optional) – these three fields compose Issuer endpoint
- specify conditions for return refresh token
- check supported grant types, for our demo just check all four types. However only Authorization code is used.
- optionally check Audience required – this adds aud property into authorization code and implicit requests
- optionally check Support user session - this means that an httpOnly cookie is used by the authorization server to keep the current user of this browser logged in. The second and subsequent requests for the access token will not prompt for user name and password.
- specify endpoint intervals
- define scopes supported by this server
- accept default or enter custom values of customization options –please note – change Generate token class valued from %OAuth2.Server.Generate to %OAuth2.Server.JWT so that a JWT is used as the access token rather than just an opaque token.
- provide name of registered SSL configuration to establish SSL over HTTP as required by OAuth 2.0
- Fill in settings for JSON Web Token (JWT)

Here is a screenshot of sample configuration

Menu

Home | About | Help | Logout

System > Security Management > OAuth 2.0 Authorization Server Configuration

OAuth 2.0 Authorization Server Configuration

Server: WIN-U9J96QBJSAG    Namespace: %SYS  
User: Administrator    Licensed to: ISC Sales Engineering    Instance: AUTHSERVER

Save

Cancel

Delete

Client Descriptions

Use the form below to edit the OAuth 2.0 authorization server configuration:

General

Scopes

Intervals

JWT Settings

Customization

Description

OpenID server for demo purposes

Issuer endpoint

The endpoint for this Authorization server.

https://WIN-U9J96QBJSAG/authserver/oauth2

Host name

WIN-U9J96QBJSAG

Required.

Port

Optional.

Prefix

authserver

Optional.

Audience required

☐

Support user session

☒

Return refresh token

Only as required by OpenID Connect ▾

Supported grant types (check at least one)

☒ Authorization code  
☒ Implicit  
☒ Resource owner password credentials  
☒ Client credentials

OpenID provider documentation

Service Documentation URL

Policy URL

Terms of service URL

SSL/TLS configuration

SSL4AUTHSERVER ▾

General

Scopes

Intervals

JWT Settings

Customization

Scopes

Supported scopes

Scope	Description	
scope1	First Scope	Edit
scope2	Second Scope	Edit
scope3	Third Scope	Edit
scope4	Fourth Scope	Edit

Require at least one supported scope.  
Add Supported Scope

☐ Allow unsupported scope

Default scope

General

Scopes

Intervals

JWT Settings

Customization

Endpoint intervals

Access token interval

900

Required. Enter interval in seconds.

Authorization code interval

60

Required. Enter interval in seconds.

Refresh token interval

2700

Required. Enter interval in seconds.

Session termination interval

86400

Required. Enter interval in seconds. Enter 0 for no automatic session termination.

Client secret expiration interval

0

Required. Enter interval in seconds. Enter 0 for no automatic client secret expiration.

General

Scopes

Intervals

JWT Settings

Customization

JSON Web Token (JWT) Settings

☒ Create JWT Settings from X509 credentials

X509 credentials

AuthServerConfig ▼

Required. Select credentials.

Private key password

.....

Signing algorithm

RS512: RSASSA-PKCS1-V1\_5 using SHA-512 ▼

Key management algorithm

RSA-OAEP: Encryption with RSAES-OAEP ▼

Content encryption algorithm

A256CBC-HS512: 256-bit AES in CBC mode with HMAC SHA-512 ▼

General

Scopes

Intervals

JWT Settings

Customization

Customization Options

Authenticate class

%OAuth2.Server.Authenticate

Required.

Validate user class

%OAuth2.Server.Validate

Required.

Session maintenance class

OAuth2.Server.Session

Required.

Generate token class

%OAuth2.Server.JWT

Required.

Customization namespace

%SYS ▼

Required.

Customization roles (select at least one)

Available

----- Select One or More -----

%All

%DB\_%DEFAULT

%DB\_CACHE

%DB\_CACHEAUDIT

%DB\_CACHELIB

%DB\_CACHETEMP

%DB\_DOCBOOK

%DB\_SAMPLES

%DB\_USER

%Developer

%Operator

%SecureBreak

%SQL

%SQLTuneTable

Selected

----- Select One or More -----

%DB\_CACHESYS

%Manager

Hold the [Shift] or [Ctrl] key while clicking to select multiple roles.

Having defined server configuration, we need to supply server client configuration. Within the page with server configuration form, click Client Configurations button and then press Create New Configuration for your CLIENT and RESSERVER instances.

This image shows CLIENT configuration.

General Client Credentials Client Information JWT Settings

Name   
Required.

Description

Client type ☒ Confidential ☐ Public ☐ Resource server  
Required.

Redirect URLs

Require at least one URL. Click an item in the list to edit or remove.

Supported grant types (check at least one) ☒ Authorization code  
☐ Implicit  
☒ Resource owner password credentials  
☒ Client credentials

Supported response types (check at least one) ☒ code  
☒ id\_token  
☒ id\_token token  
☒ token

Authentication type ☐ none ☒ basic ☐ form encoded body ☐ client secret JWT ☐ private key JWT

General Client Credentials Client Information JWT Settings

Client ID

Client secret [show](#)

General

Client Credentials

Client Information

JWT Settings

Launch URL:

https://WIN-U9J96QBJJSAG/client/csp/oauth2

Authorization display

This section contains the information to be displayed when requesting permissions from the end user.

Client name

MY DEMO CLIENT

Logo URL

http://WIN-U9J96QBJJSAG/authserver/csp/oauth2/authserver\_logo.gif

Client home page URL

http://WIN-U9J96QBJJSAG/csp/samples/menu.csp

Policy URL

http://WIN-U9J96QBJJSAG/csp/samples/inspector.csp

Terms of service URL

http://WIN-U9J96QBJJSAG/csp/samples/zipcode.csp

Contact emails (comma separated)

Default max age

Default scope

Leave JWT Token tab empty - with default values. As you can see, we populated fields with meaningless data, unlike in real application case.

And similarly, RESSERVER configuration

Client Description

Server: WIN-U9J96QBJJSAG Namespace: %SYS  
User: Administrator Licensed to: ISC Sales Engineering Instance: AUTHSERVER

Save

Cancel

Use the form below to edit an existing client description which has been registered with the OAuth 2.0 authorization server:

General

Client Credentials

Client Information

JWT Settings

Name

RESSERVER

Required.

Description

RESSERVER - server with protected resources

Client type

☒ Confidential ☐ Public ☐ Resource server

Required.

Redirect URLs

http://localhost

Require at least one URL. Click an item in the list to edit or remove.

Add URL

OK

Remove

Cancel

Supported grant types (check at least one)

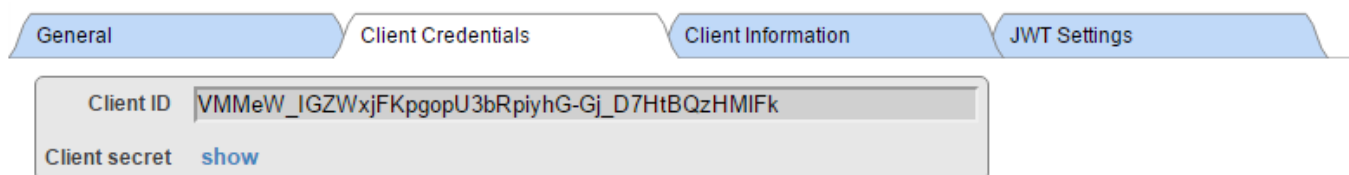
☒ Authorization code  
☐ Implicit  
☐ Resource owner password credentials  
☐ Client credentials

Supported response types (check at least one)

☒ code  
☒ id\_token  
☒ id\_token token  
☒ token

Authentication type

☐ none ☒ basic ☐ form encoded body ☐ client secret JWT ☐ private key JWT



The screenshot shows a configuration interface with four tabs: General, Client Credentials, Client Information, and JWT Settings. The Client Credentials tab is active. It contains two fields: 'Client ID' with the value 'VMMeW\_IGZWxjFKpgopU3bRpiyhG-Gj\_D7HtBQzHMIFk' and 'Client secret' with a 'show' button next to it.

Tab	Field	Value
Client Credentials	Client ID	VMMeW_IGZWxjFKpgopU3bRpiyhG-Gj_D7HtBQzHMIFk
	Client secret	show

As you can see, there is only very basic information needed for resource server, namely you need to set client type to Resource server. With CLIENT, you need to provide more information, the Client type (confidential as our client runs as web application capable of keeping client secret at server, not sending it to client agent).

## CLIENT

In SMP, navigate to System Administration > Security > OAuth 2.0 > Client Configurations.

Click Create Server Configuration button, fill the form and save it.



[Menu](#)   [Home](#) | [About](#) | [Help](#) | [Logout](#)   [System](#) > [Security Management](#) > [OAuth 2.0 Client](#) > [Server Description](#)

**Server Description**

Server: **WIN-U9J96QBJSAG**   Namespace: %SYS  
User: **UnknownUser**   Licensed to: **ISC Sales Engineering**   Instance: **CLIENT**

[Save](#)   [Cancel](#)   [Discover and Save](#)   [Edit](#)   [Update JWKS](#)

Use the form below to edit an existing OAuth 2.0 server description (entered manually):

Issuer endpoint

Required. Endpoint URL to be used to identify the authorization server.

SSL/TLS configuration

Required if SSL used for discovery.

Registration access token

Optional.

Authorization server

This section describes the authorization server to be used

Authorization endpoint  
  
Required.

Token endpoint  
  
Required.

Userinfo endpoint

Token introspection endpoint

Token revocation endpoint

JSON Web Token (JWT) Settings

Source other than dynamic registration

URL:  
  
Required. Enter an URL.

The following is a list of server metadata properties:

Name	Value
issuer	https://WIN-U9J96QBJSAG/authserver/oauth2
authorization_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/authorize
token_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/token
userinfo_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/userinfo
revocation_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/revocation
introspection_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/introspection
jwks_uri	https://WIN-U9J96QBJSAG/authserver/oauth2/jwks

Make sure that Issuer Endpoint corresponds to the value we defined earlier at the AUTHSERVER instance! You also need to modify authorization server endpoints according to your web server configuration. In our case we just embedded ' authserver ' into each input field.

Now, click Client Configurations link next to the newly created Issuer Endpoint and click Create Client Configuration button.

General

Client Information

JWT Settings

Client Credentials

Application name

demo client

Required. Local name of the client application.

Client name

demo client

Global name to be used for dynamic registration.

Description

client for demo

Enabled

☒

Client Type

☒ Confidential ☐ Public ☐ Resource server

SSL/TLS configuration

SSL4CLIENT

Required.

Client redirect URL

The client URL to be specified to the authorization server to receive responses.

https://WIN-U9J96QBJ5AG/client/csp/sys/oauth2/OAuth2.Response.cls

Use TLS/SSL

☒

Host name

WIN-U9J96QBJ5AG

Required.

Port

Optional.

Prefix

client

Optional.

Required grant types (check at least one)

☒ Authorization code ☐ Implicit ☐ Resource owner password credentials ☐ Client credentials

Authentication type

☐ none ☒ basic ☐ form encoded body ☐ client secret JWT ☐ private key JWT

General

Client Information

JWT Settings

Client Credentials

Authorization display

This section contains the information to be displayed when requesting permissions from the end user.

Logo URL

Client home page URL

Policy URL

Terms of service URL

Default scope

scope1

Contact emails (comma separated)

Default max age (in seconds)

The screenshot shows the 'JSON Web Token (JWT) Settings' tab. It features a section for 'Create JWT Settings from X509 credentials' with a dropdown for 'X509 credentials' (set to 'ClientConfig'), a 'Required. Select credentials.' label, and a 'Private key password' field with masked characters. Below this are three tabs: 'Signing', 'Encryption', and 'Key'. The 'Signing' tab is active, showing dropdowns for 'IDToken algorithms' (RS256), 'Userinfo algorithms' (none), 'Access token algorithms' (none), and 'Request algorithms' (none). The 'Encryption' tab shows 'A256CBC-HS512' for IDToken and none for others. The 'Key' tab shows 'RSA-OAEP' for IDToken and none for others.

The screenshot shows the 'Client Credentials' tab. It contains a section titled 'This client's credentials' with the following fields: 'Client ID' (dkivZ4-KZ95rMsCmeWpEmQXcKpJZLjLH4ie5Ai-83YU), 'Client ID Issued At' (empty), 'Client secret' (ssssssssssssssssssssss), 'Client Secret Expires At' (empty), and 'Registration Client Uri' (empty). There are also 'Required.' and 'Required if client type is confidential.' labels.

Good! At this moment we have both CLIENT and AUTHSERVER configured. That could be enough for many use cases, as resource server may be just a namespace of AUTHSERVER, thus protected already. But let ' s consider that we want to cover a use case where an external doctor is trying to retrieve data from our internal clinical system. So in order to allow such doctor to retrieve data, we definitely want to store his account information INSIDE our resource server for auditing and forensic reasons. In that case, we need to continue and define configurations at RESSERVER.

## RESSERVER

In SMP, navigate to System Administration > Security > OAuth 2.0 > Client Configurations.

Click Create Server Configuration button, fill the form and save it.

Menu

Home | About | Help | Logout

System > Security Management > OAuth 2.0 Client > Server Description

Server Description

Server: WIN-U9J96QBJSAG    Namespace: %SYS  
User: Administrator    Licensed to: ISC Sales Engineering    Instance: RESSERVER

Save

Cancel

Discover and Save

Edit

Update JWKS

Use the form below to edit an existing OAuth 2.0 server description (created via Discovery):

Issuer endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2

Required. Endpoint URL to be used to identify the authorization server.

SSL/TLS configuration

SSL4RESSERVER

Required if SSL used for discovery.

Registration access token

Optional.

Authorization server

This section describes the authorization server to be used

Authorization endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2/authorize

Required.

Token endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2/token

Required.

Userinfo endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2/userinfo

Token introspection endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2/introspection

Token revocation endpoint

https://WIN-U9J96QBJSAG/authserver/oauth2/revocation

JSON Web Token (JWT) Settings

Source other than dynamic registration

JWKS from URL

URL:

https://WIN-U9J96QBJSAG/

Required. Enter an URL.

The following is a list of server metadata properties:

Name	Value
issuer	https://WIN-U9J96QBJSAG/authserver/oauth2
authorization_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/authorize
token_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/token
userinfo_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/userinfo
revocation_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/revocation
introspection_endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2/introspection
jwks_uri	https://WIN-U9J96QBJSAG/authserver/oauth2/jwks

We used discovery function here, this is a new feature implemented in Cache 2017.1

As you can see, this configuration is using the same data as corresponding configuration at CLIENT instance.

Now, click Client Configurations link next to the newly created Issuer Endpoint and click Create Client Configuration button.

The 'Client Information' tab is active. It contains the following fields and options:

- Application name:** RESSERVER resource (Required. Local name of the client application.)
- Client name:** RESSERVER\_resource (Global name to be used for dynamic registration.)
- Description:** demo resource server
- Enabled:** ☒
- Client Type:** ☐ Confidential ☐ Public ☒ Resource server
- SSL/TLS configuration:** SSL4RESSERVER (Required.)
- Authentication type:** ☐ none ☒ basic ☐ form encoded body ☐ client secret JWT ☐ private key JWT

The 'JWT Settings' tab is active. It contains the following fields and options:

- JSON Web Token (JWT) Settings:**
  - ☒ Create JWT Settings from X509 credentials
  - X509 credentials:** ResServerConfig (Required. Select credentials.)
  - Private key password:** .....
- Signing:** RS512
- Encryption:** A256CBC-HS512
- Key:** RSA-OAEP

Creating WT from X.509 Credentials is not recommended, but we used it for compatibility.

The 'Client Credentials' tab is active. It contains the following fields:

- This client's credentials:**
  - Client ID:** VMMeW\_IGZWxjFKpgopU3bRpiyhG-Gj\_D7HtBQzHMIFk (Required.)
  - Client ID Issued At:**
  - Client secret:** sssssssssssssssssssssssssssssssssssss (Required if client type is confidential.)
  - Client Secret Expires At:**
  - Registration Client Uri:**

Oh yes! Indeed, this was tedious, but necessary process. But now we can move forward, and start coding!

## Client Application

To keep things as simple as possible, we will recycle much of code from our Google example we described in part 1.

The client application is just two CSP pages, running in /csp/myclient application, with no security enforced – it just runs as unauthenticated user.

## Page 1

```
Class Web.OAUTH2.Cache1N Extends %CSP.Page
{

Parameter OAUTH2CLIENTREDIRECTURI = "https://dk-
gs2016/client/csp/myclient/Web.OAUTH2.Cache2N.cls";

Parameter OAUTH2APPNAME = "demo client";

ClassMethod OnPage() As %Status
{
    &html<<html>
<head>
    <style>

        .portalLogo {
            color: rgb(53,107,141);
            position: relative;
            font-weight: bold;
            font-size: 12pt;
            top: 0px;
            right: 0px;
            border: 2px solid rgb(53,107,141);
            padding: 2px;
            padding-left: 5px;
            padding-right: 5px;
            border-radius: 4px;
            background: #E0E0F0;
        }

        .portalLogoBox {
            position: static;
            padding: 10px;
            padding-bottom: 4px;
            padding-right: 30px;
            text-align: center;
        }

        .portalLogoSub {
            position: relative;
            color: #808080;
            font-size: 8pt;
            top: 3px;
            right: 0px;
        }

    </style>

</head>
<body>
    <h1>Authenticating and Authorizing against Cache&acute; OAuth2 provider</h1>
    <p>This page demo shows how to call Cache&acute; API functions using OAuth2 authori
zation.
    <p>We are going to call Cache&acute; authentication and authorization server to gra
nt our application access to data stored at another
    Cache&acute; server.
```

---

```
>

// Get the url for authorization endpoint with appropriate redirect and scopes.
// The returned url is used in the button below.

// DK: use 'dankut' account to authenticate!
set scope="openid profile scopel scope2"
set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
    ..#OAUTH2APPNAME,
    scope,
    ..#OAUTH2CLIENTREDIRECTURI,
    .properties,
    .isAuthorized,
    .sc)
if $$$ISERR(sc) {
    write "GetAuthorizationCodeEndpoint Error="
    write ..EscapeHTML($system.Status.GetErrorText(sc))_"<br>","!
}

&html<
<div class="portalLogoBox"><a class="portalLogo" href="#(url)#">Authorize for <b>IS
C</b></a></div>
</body></html>>
Quit $$$OK
}

ClassMethod OnPreHTTP() As %Boolean [ ServerOnly = 1 ]
{
    #dim %response as %CSP.Response
    set scope="openid profile scopel scope2"
    if ##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,scope,.accessTok
en,.idtoken,.responseProperties,.error) {
        set %response.ServerSideRedirect="Web.OAUTH2.Cache2N.cls"
    }
    quit 1
}

}
```

## Page 2

```
Class Web.OAUTH2.Cache2N Extends %CSP.Page
{

Parameter OAUTH2APPNAME = "demo client";

Parameter OAUTH2ROOT = "https://dk-gs2016/resserver";

Parameter SSLCONFIG = "SSL4CLIENT";

ClassMethod OnPage() As %Status
{
    &html<<html>
<head>
</head>
```

```
<style>
body { font-family: verdana; }

h4 { color:#2080E0 ;}
</style>

<body>>

    // Check if we have an access token from oauth2 server
    set isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,
"scope1 scope2",.accessToken,.idtoken,.responseProperties,.error)

    // Continue with further checks if an access token exists.
    // Below are all possible tests and may not be needed in all cases.
    // The JSON object which is returned for each test is just displayed.
    if isAuthorized {
        write "<h3>Authorized!</h3>",!

        // Validate and get the details from the access token, if it is a JWT.
        set valid=##class(%SYS.OAuth2.Validation).ValidateJWT(..#OAUTH2APPNAME,access
Token,"scope1 scope2",,.jsonObject,.securityParameters,.sc)
        if $$$ISOK(sc) {
            if valid {
                write "Valid JWT"_"<br>",!
            } else {
                write "Invalid JWT"_"<br>",!
            }
            write "Access token="
            do jsonObject.%ToJSON()
            write "<br>",!
        } else {
            write "JWT Error="_"..EscapeHTML($system.Status.GetErrorText(sc))_"<br>",!
        }
        write "<br>",!

        // Call the introspection endpoint and display result -- see RFC 7662.
        set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection(..#OAUTH2APPNAME,acc
essToken,.jsonObject)
        if $$$ISOK(sc) {
            write "Introspection="
            do jsonObject.%ToJSON()
            write "<br>",!
        } else {
            write "Introspection Error="_"..EscapeHTML($system.Status.GetErrorText(sc)
)"_"<br>",!
        }
        write "<br>",!

        if idtoken'="" {
            // Validate and display the IDToken -- see OpenID Connect Core specificat
ion.

            set valid=##class(%SYS.OAuth2.Validation).ValidateIDToken(
                ..#OAUTH2APPNAME,
                idtoken,
                accessToken,,
                .jsonObject,
                .securityParameters,
```



```
.sc)
if $$$ISOK(sc) {
    if valid {
        write "Valid IDToken_"<br>",!
    } else {
        write "Invalid IDToken_"<br>",!
    }
    write "IDToken="
    do jsonObject.%ToJSON()
    write "<br>",!
} else {
    write "IDToken Error="_..EscapeHTML($system.Status.GetErrorText(sc))_
"<br>",!
}
} else {
    write "No IDToken returned_"<br>",!
}
write "<br>",!

// not needed for the application logic, but provides information about user
that we can pass to Delegated authentication

// Call the userinfo endpoint and display the result -- see OpenID Connect Co
re specification.
set sc=##class(%SYS.OAuth2.AccessToken).GetUserinfo(
    ..#OAUTH2APPNAME,
    accessToken,,
    .jsonObject)
if $$$ISOK(sc) {
    write "Userinfo="
    do jsonObject.%ToJSON()
    write "<br>",!
} else {
    write "Userinfo Error="_..EscapeHTML($system.Status.GetErrorText(sc))_<b
r>",!
}
write "<p>",!

/*****
*
*   Call the resource server and display result.
*
*****/

// option 1 - resource server - by definition - trusts data coming from autho
rization server,
//     so it serves data to whoever is asking
//   as long as access token passed to resource server is valid

// option 2 - alternatively, you can use delegated authentication (OpenID Con
nect)
//   and call into another CSP application (with delegated authentication prot
ection)
//   - that's what we do here in this demo

write "<h4>Call resource server (delegated auth)","</h4>",!
set httpRequest=##class(%Net.HttpRequest).%New()
// AddAccessToken adds the current access token to the request.
```

```
set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(
    httpRequest,,
    ..#SSLCONFIG,
    ..#OAUTH2APPNAME)
if $$$ISOK(sc) {
    set sc=httpRequest.Get(..#OAUTH2ROOT_"/csp/portfolio/oauth2test.demoResou
rce.cls")
}
if $$$ISOK(sc) {
    set body=httpRequest.HttpResponse.Data
    if $isobject(body) {
        do body.Rewind()
        set body=body.Read()
    }
    write body,"<br>",!
}
if $$$ISERR(sc) {
    write "Resource Server Error="_.EscapeHTML($system.Status.GetErrorText(s
c))_"<br>",!
}
write "<br>",!

write "<h4>Call resource server - no auth, just token validity check","</h4>"
,!

set httpRequest=##class(%Net.HttpRequest).%New()
// AddAccessToken adds the current access token to the request.
set sc=##class(%SYS.OAuth2.AccessToken).AddAccessToken(
    httpRequest,,
    ..#SSLCONFIG,
    ..#OAUTH2APPNAME)
if $$$ISOK(sc) {
    set sc=httpRequest.Get(..#OAUTH2ROOT_"/csp/portfolio2/oauth2test.demoReso
urce.cls")
}
if $$$ISOK(sc) {
    set body=httpRequest.HttpResponse.Data
    if $isobject(body) {
        do body.Rewind()
        set body=body.Read()
    }
    write body,"<br>",!
}
if $$$ISERR(sc) {
    write "Resource Server Error="_.EscapeHTML($system.Status.GetErrorText(s
c))_"<br>",!
}
write "<br>",!
} else {
    write "Not Authorized!<p>",!
    write "<a href='Web.OAUTH2.Cache1N.cls'>Authorize me</a>"
}
&html<</body></html>>
Quit $$$OK
}

}
```

Following screenshots illustrate processing:

Authorization / OpenID Connect Authentication server login page at AUTHSERVER instance

### Sign in with your Account

User consent page at AUTHSERVER

User:  
dankut

CACHÉ - AUTHSERVER

[Policy](#)  
[Terms of service](#)

**[MY DEMO CLIENT](#) is requesting these permissions:**

1. OpenID Connect

2. User default profile information

3. First Scope

4. Second Scope

And, finally, resulting page

#### Authorized!

Valid JWT

```
Access token= {"jti": "https://dk-gs2016/authserver/oauth2.7zqYM6psc5MBHZRu42ZF_3rbd-w", "iss": "https://dk-gs2016/authserver/oauth2", "sub": "dankut", "exp": 1464680155, "aud": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU"}
```

```
Introspection= {"active": true, "scope": "openid profile scope1 scope2", "client_id": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU", "username": "dankut", "token_type": "bearer", "exp": 1464680155, "iat": 1464679255, "nbf": 1464679255, "sub": "dankut", "aud": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU", "iss": "https://dk-gs2016/authserver/oauth2"}
```

Valid IDToken

```
IDToken= {"iss": "https://dk-gs2016/authserver/oauth2", "sub": "dankut", "exp": 1464680155, "auth_time": 1464679255, "iat": 1464679255, "nonce": "shnTRQihvx0qY2sXxfJt26NLZL8", "at_hash": "e-3jKu5hHNqdmALhnevTFLijGy7G9G005X1Q7xDCq2w", "aud": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU", "azp": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU"}
```

```
Userinfo= {"sub": "dankut", "iss": "https://dk-gs2016/authserver/oauth2", "aud": "Jyb4714IJ2kCF6EAyYd5frvAeSJ7ITFTf1wP-IVgQmU", "name": "Dan Kutac", "preferred_username": "dankut", "updated_at": 1463386129}
```

#### Call resource server (delegated auth)

**Hello from Caché server: /csp/portfolio application!**

running code as `$username = dankut` with following `$roles = scope1,scope2` at node **RESSERVER**.

#### Call resource server - no auth, just token validity check

**Hello from Caché server: /csp/portfolio2 application!**

running code as `$username = UnknownUser` with following `$roles = %All` at node **RESSERVER**.

As you could see, reading through the code, indeed, there is almost no difference to client code we showed in part 1. There is something new that comes with page 2. This is some debugging information, and checking validity of JWT. Once we validated returned JWT, we could introspect data coming from AUTHSERVER about the user identity. We simply presented this information to the page output, but we can do more with it. As in the above mentioned use case of an external doctor, we can use the identity information and pass it to the resource server for the authentication purposes if required. Or just passing this information as parameter to API call to resource server.

Next paragraphs will describe how we used the user identity information, in more details.

## Resource Application

The resource server can be the same server as authorization / authentication server and in many cases that would be the case. But in our demo, we made the two servers separate InterSystems IRIS instances.

So, we have two possible cases, how to work with security context on the resource server.

### Alternative 1 – no authentication

This is the simple case. Authorization/ authentication server are just the same Caché instance. In this case we can simply pass access token to a csp application, which is specifically created for a single purpose – serve data to client applications that use OAUTH to authorize them to ask for data.

The configuration of the resource csp application (we called it /csp/portfolio2) can look like the screenshot below.

Edit: /csp/portfolio2

Server: dk-gs2016  
User: dkutac

Namespace: %SYS  
Licensed to: ISC Sales Engineering

Instance: RESSERVER

Save

Cancel

Edit definition for web application /csp/portfolio2:

Application saved.

General

Application Roles

Matching Roles

Name

/csp/portfolio2

Required. (e.g. /csp/appname)

Description

Namespace

PORTFOLIO2

Default Application for PORTFOLIO2: /csp/portfolio2

☒ Namespace Default Application

Enabled

☒ Application ☒ CSP/ZEN ☒ Inbound Web Services  
☐ DeepSee ☐ iKnow

Permitted Classes

1"oauth2test.demoResource"

Security Settings

Resource Required

Group By ID

Allowed Authentication Methods

☒ Unauthenticated ☐ Password ☐ Kerberos ☐ Delegated ☐ Login Cookie

Session Settings

Session Timeout

900

seconds

Event Class

Use Cookie for Session

Always

Session Cookie Path

/csp/portfolio2/

Dispatch Class

CSP File Settings

Serve Files

No

Serve Files Timeout

3600

seconds

CSP Files Physical Path

c:\intersystems\cache\resserver\csp\portfolio2\

Browse...

Package Name

Default Superclass

CSP Settings

☒ Recurse ☒ Auto Compile ☒ Lock CSP Name

Custom Pages

Login Page

Change Password Page

Custom Error Page

We put just minimum security into the application definition – allowing only specific CSP page to be executed.

Alternatively, the resource server can provide a REST API instead of classic web pages. In real life scenarios, this is all up to the user to fine-tune security context.

An example of source code:

```
Class oauth2test.demoResource Extends %CSP.Page
{
ClassMethod OnPage() As %Status
{
    set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromRequest(.sc)
    if $$$ISOK(sc) {
        set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("RESSERVER resource",accessToken,.jsonObject)
        if $$$ISOK(sc) {
            // optionally validate against fields in jsonObject

            w "<p><h3>Hello from Cach&eacute; server: <i>/csp/portfolio2</i> applicat
ion!</h3>"
            w "<p>running code as <b>$username = \"_username_\"</b> with following <b>
$roles = \"_roles_\"</b> at node <b>\"_$(zup($zu(86),\"\",2))_\"</b>."
        }
    } else {
        w "<h3>NOT AUTHORIZED!</h3>"
        w "<pre>"
        w
    }
}
```

```
        i $d(%objlasterror) d $system.OBJ.DisplayError()  
        w "</pre>"  
    }  
    Quit $$$OK  
}  
  
}
```

## Alternative 2 – delegated authentication

This is another extreme case, we want to utilize user 's identity at resource server to the maximum possible extent – as if the user was working with equal security context as internal users of resource server.

One of possible potions we have, is using delegated authentication.

To get this setup running, we need to perform a few more steps to configure the resource server.

- Enable Delegated Authentication
- Provide ZAUTHENTICATE routine
- Configure Web application (in our case we called in /csp/portfolio)

The ZAUTHENTICATE routine implementation is very simple and straightforward, as we trust the AUTHSERVER who provided user identity as well as his/her scope (security profile), so we simply accept whatever username is coming and pass it along with scope to resource server user database (with necessary translation between OAUTH scope and InterSystems IRIS roles). That 's it. The rest is done seamlessly by InterSystems IRIS.

Here is an example of ZAUTHENTICATE routine

```
#include %occErrors  
#include %occInclude  
  
ZAUTHENTICATE(ServiceName, Namespace, Username, Password, Credentials, Properties) PUBLIC  
{  
    set tRes=$SYSTEM.Status.OK()  
    try {  
        set Properties("FullName")="OAuth account "_Username  
        //set Properties("Roles")=Credentials("scope")  
        set Properties("Username")=Username  
        //set Properties("Password")=Password  
        // temporary hack as currently we can't pass Credentials array from GetCredentials()  
        // method  
        set Properties("Password")="xxx" // we don't really care about oauth2 account password  
        set Properties("Roles")=Password  
    } catch (ex) {  
        set tRes=$SYSTEM.Status.Error($$$AccessDenied)  
    }  
    quit tRes  
}  
  
GetCredentials(ServiceName, Namespace, Username, Password, Credentials) Public  
{  
    set ts=$zts
```

```
set tRes=$SYSTEM.Status.Error($$$AccessDenied)

try {
  If ServiceName="%Service_CSP" {
    set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromRequest(.sc)

    if $$$ISOK(sc) {
      set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("RESOURCE resource",accessToken,.jsonObject)
      if $$$ISOK(sc) {
        // todo: watch out for potential collision between standard account and delegated (openid) one!
        set Username=jsonObject.username
        set Credentials("scope")=$p(jsonObject.scope,"openid profile ",2)
        set Credentials("namespace")=Namespace
        // temporary hack
        //set Password="xxx"
        set Password=$tr(Credentials("scope")," ","")
        set tRes=$SYSTEM.Status.OK()
      } else {
        set tRes=$SYSTEM.Status.Error($$$GetCredentialsFailed)
      }
    }
  } else {
    set tRes=$SYSTEM.Status.Error($$$AccessDenied)
  }
} catch (ex) {
  set tRes=$SYSTEM.Status.Error($$$GetCredentialsFailed)
}
Quit tRes
}
```

The CSP page itself can then be very simple:

```
Class OAuth2test.DemoResource Extends %CSP.Page
{

ClassMethod OnPage() As %Status
{
  // access token authentication is performed by means of Delegated authentication!
  // no need to do it, again, here

  // This is a dummy resource server which just gets the access token from the request and
  // uses the introspection endpoint to ensure that the access token is valid.
  // Normally the response would not be security related, but would contain some interesting
  // data based on the request parameters.
  w "<p><h3>Hello from Caché server: <i>/csp/portfolio</i> application!</h3>"
  "
  w "<p>running code as <b>$username = \"_username_\"</b> with following <b>$roles = \"_roles_\"</b> at node <b>\"_p($zu(86),\"\",2)\"</b>."
  Quit $$$OK
}
}
```

And, lastly, the Web application configuration for /csp/portfolio

Edit: /csp/portfolio

Server: **dk-gs2016** Namespace: %SYS  
User: **dkutac** Licensed to: **ISC Sales Engineering** Instance: **RESSERVER**

Save Cancel

Edit definition for web application /csp/portfolio:

GeneralApplication RolesMatching Roles

Name: /csp/portfolio  
Required. (e.g. /csp/appname)

Description:

Namespace: PORTFOLIO Default Application for PORTFOLIO: /csp/portfolio ☒ Namespace Default Application

Enabled: ☒ Application ☒ CSP/ZEN ☒ Inbound Web Services  
☐ DeepSee ☐ iKnow

Permitted Classes:

Security Settings

Resource Required: Group By ID:

Allowed Authentication Methods: ☐ Unauthenticated ☒ Password ☐ Kerberos ☒ Delegated ☐ Login Cookie

Session Settings

Session Timeout: 900 seconds Event Class:

Use Cookie for Session: Always Session Cookie Path: /csp/portfolio/

Dispatch Class:

CSP File Settings

Serve Files: Always Serve Files Timeout: 3600 seconds

CSP Files Physical Path: c:\intersystems\cache\resserver\csp\portfolio\ Browse...

Package Name: Default Superclass:

CSP Settings: ☒ Recurse ☒ Auto Compile ☒ Lock CSP Name

Custom Pages

Login Page: Change Password Page:

Custom Error Page:

If you were really paranoid, you could set Permitted classes as we did in first variant. Or, again, use REST API. But all this is way beyond the scope of our topic.

Next time, we are going to explain individual classes, introduced by the InterSystems IRIS OAUTH framework. We will describe their APIs, and when / where to call them.

[1] Whenever we mention OAUTH we mean OAuth 2.0 as specified in RFC 6749 - <https://tools.ietf.org/html/rfc6749>. We use shortcut OAUTH just for simplicity.

[2] OpenID Connect is maintained by OpenID Foundation – <http://openid.net/connect>

[#Authentication](#) [#Access control](#) [#Best Practices](#) [#OAuth2](#) [#Security](#) [#Caché](#) [#Ensemble](#) [#InterSystems IRIS](#)

---

Source  
URL: <https://community.intersystems.com/post/intersystems-iris-open-authorization-framework-oauth-20-implementation-part-2>