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-U9J96QBJSAG as if it was dk-gs2016.

Part 2. Authorization server, OpenID Connect server

In the <u>previous part</u> 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.

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 <u>documentation</u> or elsewhere. As a result of generating certificates, we will create 3 public/private key pairs and associated certificates.

Let 's call them

- · root<u>c</u>a (root<u>c</u>a.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: AUTH SERVER
Create New Credentials	
The following sets of X.509 credentials are a	available to encrypt, decrypt, sign, and verify content (primarily for use w
Page size: 0 Max rows: 1000 Results: 2 Pa	age:  < << 1 >>> >  of 1
Alias Owner List Peer Names Has	Private Key CAFile
AuthServerConfig	1 C:\InterSystems\Cache\AUTHSERVER\mgr\cache.cer Edit Delete
ClientConfig	0 C:\InterSystems\Cache\AUTHSERVER\mgr\cache.cer Edit Delete

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 User: UnknownUser	Namespace: % SYS 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:         (<<						
Alias	Owner List	Peer Names	Has Private Key	CAFile		
AuthServerConfig			0	C:\InterSystems\Cache\CLIENT\mgr\cache.cer Edit Delete		
ClientConfig			1	C:\InterSystems\Cache\CLIENT\mgr\cache.cer		

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

# InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2 Published on InterSystems Developer Community (https://community.intersystems.com)

uth 2.0 Authorization Se	erver Configuration	Server: WIN-U9J96QBJ User: Administrator ent Descriptions	ISAG Namespace: % SY Licensed to: ISC S	s ales Engineering	Instance: AUTH SERVER
e the form below to	edit the OAuth 2	2.0 authorization server config	guration:		
eneral	Scopes	Intervals	JWT Set	ings	Customization
	Description Issuer endpoint	OpenID server for demo purposes The endpoint for this Authorization	<b>server.</b> thserver/oauth2		]
		Host name Por WIN-U9J96QBJSAG Required. Opt	t ional.	Prefix authserver Optional.	
	Audience required				
Sup	oport user session				
Re	turn refresh token	Only as required by OpenID Conne	ect 🔻		
Supported grant types (c	heck at least one)	<ul> <li>Authorization code</li> <li>Implicit</li> <li>Resource owner password creden</li> <li>Client credentials</li> </ul>	tials		
OpenID provid	ler documentation	Service Documentation URL			
		Policy URL Terms of service URL			
			_		

General		Scopes	Intervals	JWT Settings	Customization
copes					
Supported	scopes				
Scope	Description				
scope1	First Scope	Edit			
scope2	Second Scope	Edit			
scope3	Third Scope	Edit			
scope4	Fourth Scope	Edit			
Require at l	least one supported	I scope.			
Add Sup	ported Scope				
🗆 Allow u	nsupported scope				
Default sco	ope				
1					

# InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2 Published on InterSystems Developer Community (https://community.intersystems.com)

General	Scopes	Intervals	JWT Settings	Cus
Endpoint intervals				
Access token interval				
900				
Required. Enter interval in second	nds.			
Authorization code interval				
60				
Required. Enter interval in second	nds.			
Refresh token interval				
2700				
Required. Enter interval in second	nds.			
Session termination interval				
86400				
Required. Enter interval in second	nds. Enter 0 for no automatic sessi	on termination.		
Client secret expiration interva	al			
0				
Required. Enter interval in second	nds. Enter 0 for no automatic client	secret expiration.		

General	Scopes	Intervals	JWT Settings	Customization
SON Web Token (JWT) S	ettings			
Create JWT Settings	from X509 credentials			
X509 credentials				
AuthServerConfig <b>v</b>				
Required. Select cred	entials.			
Private key passwore	d			
•••••				
Signing algorithm				
RS512: RSASSA-PKC	S1-V1_5 using SHA-512 V			
Key management algorit	thm			
RSA-OAEP: Encryption	n with RSAES-OAEP			
Content encryption algor	rithm			
A256CBC-HS512: 256-	bit AES in CBC mode with HMA	AC SHA-512 V		

	/		JWI Settings	Customization	
Customization Options					
Authenticate class					
%OAuth2.Server.Authentica	te				
Required.					
Validate user class					
%OAuth2.Server.Validate					
Required.					
Session maintenance class					
OAuth2.Server.Session					
Required.					
Generate token class					
%OAuth2.Server.JW1					
Required.					
Required					
Customization roles (select a	t least one)				
Available	s	elected			
Select One or More	- Select C	ne or More			
%All	%DB CACH	IESYS			
%DB %DEFAULT	%Manager				
%DB CACHE	J				
%DB CACHEAUDIT					
%DB_CACHELIB					
%DB_CACHETEMP					
%DB_DOCBOOK					
%DB_SAMPLES					
%DB_USER					
%Developer					
%Operator					
%SecureBreak					
%SQL %SQLTupaTable					
%SQLTUNeTable		¥			
Hold the [Shift] or [Ctrl] key whi	le clicking to select mul	tiple 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 Credential	Client Information JWT Settings	
	Name	CLIENT Required.	
	Description	Client application	
	Client type	Confidential OPublic Resource server Required.	
		Redirect URLs https://WIN-U9J96QBJSAG/client/csp/sys/oauth2/OAuth2.Response.cls	
		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)	<ul> <li>✓ code</li> <li>✓ id_token</li> <li>✓ id_token token</li> <li>✓ token</li> </ul>	
	Authentication type	○ none ● basic ○ form encoded body ○ client secret JWT ○ private key JW	т

/	General	Client Credentials	Client Information	JWT Settings	
	Client ID	dkivZ4-KZ95rMsCmeWpEmQXcKpJZLjLH4ie5/	Ai-83YU		
	Client secret	show			

# InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2 Published on InterSystems Developer Community (https://community.intersystems.com)

General	t Credentials Client Information JWT Settings
Launch URL:	https://WIN-U9J96QBJSAG/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-U9J96QBJSAG/authserver/csp/oauth2/authserver_logo.gif Client home page URL http://WIN-U9J96QBJSAG/csp/samples/menu.csp
	http://WIN-U9J96QBJSAG/csp/samples/inspector.csp         Terms of service URL         http://WIN-U9J96QBJSAG/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 User:	WIN-U9J96QBJSAG Administrator	Namespace: % SY S Licensed to: ISC Sales Engineering	Instance: AUTH SERVER
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 Credentia	s Client Information JWT Settings
Name	RESSERVER
Description	RESSERVER - server with protected resources
Client type	Confidential Public Resource server Required.
	Redirect URLs
	nttp://iocainost
	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
	Resource owner password credentials     Client credentials
Supported response types (check at least one)	✓ code
	<ul> <li>✓ id_token token</li> <li>✓ token</li> </ul>
Authentication type	○ none

General	Client Credentials	Client Information	JWT Settings
Client ID	VMMeW_IGZWxjFKpgopU3bRpiyhG-Gj_D7Htt	BQzHMIFk	
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   Logou	Home   About   Help   Logout System > Security Management > OAuth 2.0 Client > Server Description			
Serv	er Description	Serve User:	r: WIN-U9J96QBJSAG UnknownUser	Namespace: % SY S Licensed to: ISC Sales Engineering	Instance: CLIENT
Sa	ave Cancel	Discover and Save	e Edit	Update JWKS	

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

Issuer endpoint	https://WIN-U9J96QBJSAG/authserver/oauth2
	Required. Endpoint URL to be used to identify the authorization server.
SSL/TLS configuration	SSL4CLIENT V
	Required if SSL used for discovery.
Registration access token	
Registration access token	Ontional
	optonu.
Authorization server	This section describes the authorization server to be used
	Authorization endpoint
	https://WIN-U9.J96QBJSAG/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
J SON Web Token (JWT) Settings	Source other than dynamic registration
	JWKS from URL V
	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

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 Infor	nation	JWT Settings		Client Cre	edentials	
Ap	oplication name	demo client Required. Local na	me of the client a	application.			
	Client name	demo client Global name to be i	used for dynamic	c registration.			
	Description	client for demo					
	Enabled						
	Client Type	Confidential	O Public O F	lesource serve	r		
\$ \$L/TL	S configuration	SSL4CLIENT Required.		▼			
Clie	nt redirect URL	The client URL to	be specified to th	ne authorization	n server to receiv	e responses.	
		https://WIN-U9J96QBJSAG/client/csp/sys/oauth2/OAuth2.Response.cls					
		Use TLS/SSL					
		Host name	P	ort		Prefix	
		WIN-U9J96QBJ	SAG	- P 1		client	
		Requirea.	0	puonal.		Opuonal.	
Required grant types (che	ck at least one)	Authorization co	ode				
		Implicit					
		Resource owner password credentials					
		Client credentia	lls				
Auth	entication type	onone obasi	ic 🕞 form enc	oded body 🔅	) client secret JW	/T 🔵 private ke	y JWT

General	t Information JWT Settings Client Credentials
Authorization display	This section contains the information to be displayed when requesting permissions from the end user.
	Client home page URL
	Policy URL
	Terms of service URL
Default scope	scope1
Contact emails (comma separated)	
Default max age (in seconds)	

#### InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2 Published on InterSystems Developer Community (https://community.intersystems.com)

General	lient Information	ttings Client Crede	ntials
J SON Web Token (JWT) Settings	Create JWT Settings from X509 of X509 credentials ClientConfig ▼ Required. Select credentials. Private key password 	Encryption	Кеу
IDToken algorithms	RS256 V	A256CBC-HS512 V	RSA-OAEP 🔻
Userinfo algorithms	none 🔻	none 🔻	none 🔻
Access token algorithms	none 🔻	none 🔻	none 🔻
Request algorithms	none 🔻	none 🔻	none 🔻

General	Client Information JWT Settings Client Credentials
This client's credentials	Client ID dkivZ4-KZ95rMsCmeWpEmQXcKpJZLjLH4ie5Ai-83YU Required. Client ID Issued At Client secret
	ssssssssssssssssssssssssssssssssssssss

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.

N	lenu	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: RES		Instance: RESSERVER						
	Save	Cancel	Di	scovera	and Save	Edit		Update JWKS		

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

Issuer endpoint SSL/TLS configuration Registration access token	https://WIN-U9J96QBJSAG/authserver/oauth2         Required. Endpoint URL to be used to identify the authorization server.         SSL4RESSERVER       ▼         Required if SSL used for discovery.         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.						
	https://win-09J96QBJSAG/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 V						
	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
the second se	

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.

ſ	General	Client Information JWT Settings Client Credentials	
	Application name	RESSERVER resource	
	Client name	Required. Local name of the client application.  RESSERVER_resource	
	Description	demo resource server	
	Enabled		
	Client Type	◯ Confidential ◯ Public ● Resource server	
	SSL/TLS configuration	SSL4RESSERVER T Required.	
	Authentication type	○ none ● basic ○ form encoded body ○ client secret JWT ○ private key JWT	

General	lient Information JWT Settings Client Credentials
JSON Web Token (JWT) Settings	Create JWT Settings from X509 credentials X509 credentials ResServerConfig ▼ Required. Select credentials. Private key password 
	Signing Encryption Key
Userinfo algorithms	RS512 <b>V</b> A256CBC-HS512 <b>V</b> RSA-OAEP <b>V</b>

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

General	Client Information JWT Settings Client Credentials
This client's credentials	Client ID VMMeW_IGZWxjFKpgopU3bRpiyhG-Gj_D7HtBQzHMIFk Required. Client ID Issued At Client secret ssssssssssssssssssssssssssssssssssss

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.CachelN 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>
  <hl>Authenticating and Authorizing against Cache&acute; OAuth2 provider</hl>
  This page demo shows how to call Cache´ API functions using OAuth2 authori
zation.
  We are going to call Cache´ authentication and authorization server to gra
nt our application access to data stored at another
  Cache´ 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 scope1 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 scope1 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

>

```
<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/pre>
r>",!
       }
       write "",!
       Call the resource server and display result.
       // option 1 - resource server - by definition - trusts data coming from autho
rization server,
       11
              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!",!
        write "<a href='Web.OAUTH2.CachelN.cls'>Authorize me</a>"
    }
    &html<</body></html>>
    Quit $$$OK
}
}
```

Following screenshots illustrate processing:

Authorization / OpenID Connect Authentication server login page at AUTHSERVER instance

Username	
Password	
Sign in	
Cancel	

# Sign in with your Account

#### User consent page at AUTHSERVER

User: dankut	CACHÉ - AUTHSERVER	Policy Terms of service
	<b>CLIENT</b> is requesting these permissions	:
1. OpenID C	Connect	
2. User defa	ult profile information	
3. First Sco	be	
4. Second Scope		
	Accept	
	Cancel	

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-|VgQmU","username":"dankut","token\_type":"bearer","exp":1464680155,"lat":1464679255,"nbf":1464679255,"sub":"dankut","aud":"Jyb4714iJ2kCF6EAyYd5frvAeSJ7iTFTf1wP-|VgQmU","iss":"https://dk-gs2016/authserver/oauth2"}

Valid IDToker

Valid D10ken = {"iss":"https://dk-JDToken = {"iss":"https://dk-gs2016/authserver/oauth2","sub":"dankut","exp":1464680155,"auth\_time":1464679255,"iat":1464679255,"nonce":"shnTRQihvx0qY2sXxfJt26NLZL8","at\_hash":"e-3jKuShHNqdmALhnevTFLijGy7G9G005X1Q7xDCq2w","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.

InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 2 Published on InterSystems Developer Community (https://community.intersystems.com)

Edit: /csp/portfolio	D2 Server; dk-gs2016 Namespace:%SYS User: dkutac Licensed to: ISC Sales Engineering Instance:RESSERVER
Save	
Edit definition for w	eb application /csp/portfolio2:
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     CInbound     Web Services     DeepSee     iKnow
Permitted Classes	1"oauth2test.demoResource"
Security Settings	Resource Required <ul> <li>Group By ID</li> </ul> <li>Allowed Authentication Methods</li> <li>Unauthenticated</li> <li>Password</li> <li>Kerberos</li> <li>Delegated</li> <li>Login Cookie</li>
Session Settings	Session Timeout 900 seconds Event Class
	Use Cookie for Session Always V Session Cookie Path /csp/portfolio2/ V
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 "<h3>Hello from Cach&eacute; server: <i>/csp/portfolio2</i> applicat
ion!</h3>"
           w "running code as <b>$username = "_$username_"</b> with following <b>
$roles = "_$roles_"</b> at node <b>"_$p($zu(86),"*",2)_"</b>."
        }
    } else {
        w "<h3>NOT AUTHORIZED!</h3>"
        w ""
       W
```

```
i $d(%objlasterror) d $system.OBJ.DisplayError()
    w ""
}
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 binterSystems IRIS.

Here is an example of ZAUTHENTICATE routine

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

```
set tRes=$SYSTEM.Status.Error($$$AccessDenied)
     try {
         If ServiceName="%Service_CSP" {
            set accessToken=##class(%SYS.OAuth2.AccessToken).GetAccessTokenFromReques
t(.sc)
            if $$$ISOK(sc) {
                set sc=##class(%SYS.OAuth2.AccessToken).GetIntrospection("RESSERVER r
esource",accessToken,.jsonObject)
                if $$$ISOK(sc) {
                    // todo: watch out for potential collision between standard accou
nt and delegated (openid) one!
                    set Username=jsonObject.username
                    set Credentials("scope")=$p(jsonObject.scope,"openid profile ",2)
                    set Credentials("namespace")=Namespace
                    // temporary hack
                    //set Password="xxx"
                    set Password=$tr(Credentials("scope")," ",",")
                    set tRes=$SYSTEM.Status.OK()
                } else {
                    set tRes=$SYSTEM.Status.Error($$$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 requ
est 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 in
teresting
    // data based on the request parameters.
    w "<h3>Hello from Cach&eacute; server: <i>/csp/portfolio</i> application!</h3>
п
    w "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/portfoli	O Server: dk-gs2016 Namespace:% SYS
Save Can	Cel
Edit definition for w	/eb application /csp/portfolio:
General	Application Roles Matching Roles
Name	csp/portfolio Required. (e.g. /csp/appname)
Description	
Namespace	PORTFOLIO V 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 - <u>https://tools.ietf.org/html/rfc6749</u>. 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:<u>https://community.intersystems.com/post/intersystems-iris-open-authorization-framework-oauth-20-implementation-part-2</u>