

## Give it a REST! Part 2: Retrieving and Using Azure AD OAuth Tokens in a Rule App

Warning: in the InRule Blog, we have a wide variety of posts - some are targeted towards business users, and others are targeted towards the more technical side of our customer base. This post is the second in a two-part series about interacting with REST endpoints from a Rule App, and they are decidedly for the latter audience.

In the first part [of this two-part series](#), we created a Rule App that calls out to an irServer REST Rule Execution Service (RES) – if you haven’t already, I’d highly recommend reading through that first.

In this part, we’re going to briefly touch on how to configure an Azure App Service with App Service Authentication and Azure AD, and then we’re going to update our Rule App to retrieve an OAuth token from Azure AD to use it in an execution request placed to the execution service protected with App Service Authentication.

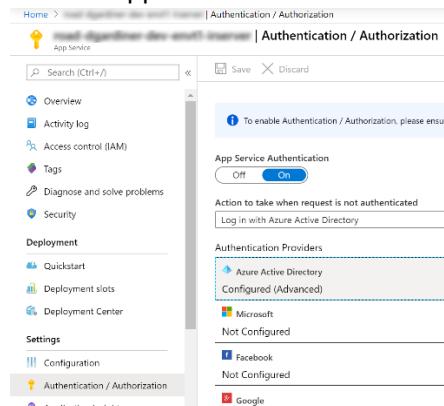
Got another cup of coffee? Let’s go!

### Configuring Azure App Services with Azure AD Authentication

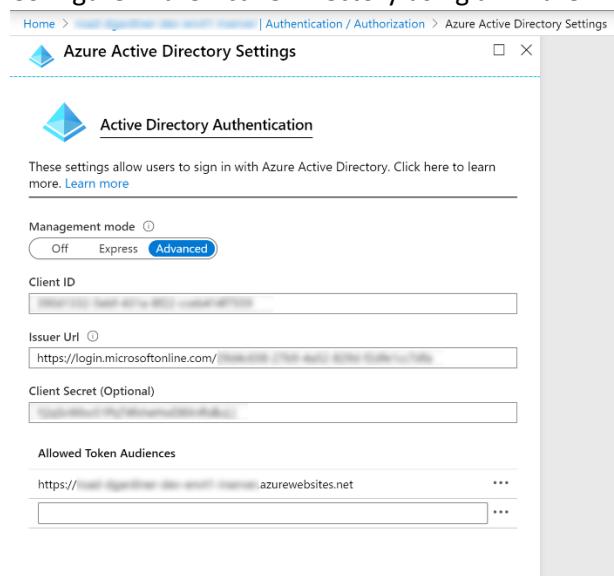
InRule’s standard irServer Rule Execution service is shipped as a generic service, which allows customers to put any kind of authentication on top of it. Our general guidance for the easiest authentication to layer on (when deployed to Azure) is to use Azure App Service Authentication, which is simple to enable on the Azure Portal.

This process is pretty well documented here <https://docs.microsoft.com/en-us/azure/app-service/configure-authentication-provider-aad>, so I’ll just touch on the highlights.

#### 1. Turn on App Service Authentication



#### 2. Configure Azure Active Directory using an Azure AD App Registration



### 3. Note the App Registration's Application ID, Directory ID, and Application ID URI

Home > Infile Technology, Inc. | App registrations > Infile Technology, Inc. | Overview

Display name: Infile Technology, Inc.

Application (client) ID: 39000000-0000-0000-0000-000000000000

Directory (tenant) ID: 72f988bf-86f1-4648-9b51-f204db0977d4

Object ID: 39000000-0000-0000-0000-000000000000

Supported account types: My organization only

Redirect URIs: 1 web, 0 public client

Application ID URI: https://39000000-0000-0000-0000-000000000000.azurewebsites.net

Managed application in ...

### 4. Enable Access Tokens on the App Registration

Home > Infile Technology, Inc. | App registrations > Infile Technology, Inc. | Authentication

Save Discard Switch to the old experience Got feedback?

Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

+ Add a platform

**Web**

Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. [Learn more about redirect URIs and the restrictions](#)

https://39000000-0000-0000-0000-000000000000.azurewebsites.net/.auth/login/aad/callback

Add URI

Logout URL

This is where we send a request to have the application clear the user's session data. This is required for single sign-out to work correctly.

e.g. https://myapp.com/logout

Implicit grant

Allows an application to request a token directly from the authorization endpoint. Recommended only if the application has a single page architecture (SPA), has no backend components, or invokes a Web API via JavaScript. [Learn more about the implicit grant flow](#)

To enable the implicit grant flow, select the tokens you would like to be issued by the authorization endpoint:

Access tokens

ID tokens

### 5. Add a Client Secret (for use in the next section) and make note of it – you will not be able to view the full Secret from the Azure Portal again.

Dashboard > Infile Technology, Inc. | App registrations > Infile Technology, Inc. | Certificates & secrets

Search (Ctrl+F)

Overview Quickstart Manage Authentication Certificates & secrets Token configuration API permissions Expose an API Owners Roles and administrators (Preview) Manifest Support + Troubleshooting Troubleshooting New support request

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

+ Upload certificate

Thumbprint Start date Expires

No certificates have been added for this application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value
Integration Training	1/3/2022	XXXXXXXXXXXX
No description	1/3/2030	Hidden

### 6. Grant the App Registration API Permissions to Azure AD Graph and the RES App Service

Home > Infile Technology, Inc. | App registrations > Infile Technology, Inc. | API permissions

Search (Ctrl+F)

Overview Quickstart Manage Authentication Certificates & secrets Token configuration API permissions Expose an API Owners Roles and administrators (Preview) Manifest Support + Troubleshooting Troubleshooting

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission

Grant admin consent for Infile Technology, Inc.

API / Permissions name	Type	Description	Admin consent req...	Status
Azure Active Directory Graph (1)				
User.Read	Delegated	Sign in and read user profile	-	...
user_impersonation	Delegated	Access	-	...

Once App Service Authentication is enabled and configured for Azure AD, you'll need to include either an APIKEY or Bearer token authentication header when placing requests to the RES.

## Retrieving and Using an OAuth Token in a Rule App

The final step in our authentication sandwich (and the original reason I started writing this post) is retrieving an OAuth token using a rule app. While in this example I finish by making a request to the RES using the method described in part one of the series, the same process would apply to placing requests to any resource behind OAuth authentication.

Since getting an OAuth token is fundamentally just another REST request, the structure should look familiar:

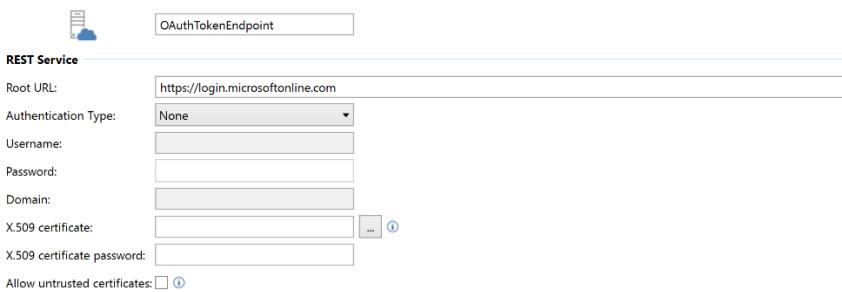
### 1. Define the Response object

We only care about two properties from the response, so it's a pretty simple entity structure.



### 2. Define the REST Service Endpoint

This is a generic URL for any Azure AD tenant

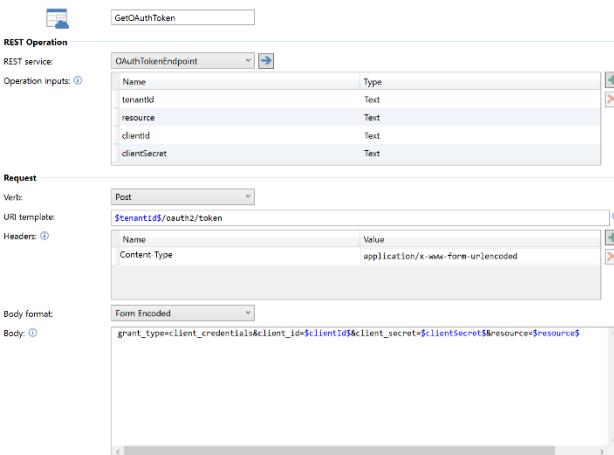


A screenshot of the 'REST Service' configuration screen. It shows the following fields:

- Root URL: `https://login.microsoftonline.com`
- Authentication Type: `None`
- Username: (empty)
- Password: (empty)
- Domain: (empty)
- X.509 certificate: (empty)
- X.509 certificate password: (empty)
- Allow untrusted certificates:

### 3. Define the REST Data Operation

This is the important bit. All parts of the Request are important here, and the parameter values are what determine the capabilities of the resulting Token.



A screenshot of the 'REST Operation' configuration screen. It shows the following fields:

- REST service: `OAuthTokenEndpoint`
- Operation inputs: 

Name	Type
tenantId	Text
resource	Text
clientId	Text
clientSecret	Text
- Request:
  - Verb: `Post`
  - URI template: `$tenantId/oauth2/token`
  - Headers:
    - Name: Content-Type, Value: `application/x-www-form-urlencoded`
  - Body format: `Form Encoded`
  - Body:

```
grant_type=client_credentials&client_id=$clientId$&client_secret=$clientSecret$&resource=$resource$
```

### 4. Execute the REST Data Operation with appropriate parameter values

Tenant ID (Directory ID), Client ID (Application ID), and Resource (Application ID URI) are all available in the App Registration Overview; the Secret was generated and noted during the App Registration setup. In my example, I'm using an entity for passing around those 4 pieces of information, but that is not required.

**Language Rule**

**Take the following actions:**

set **Credentials Tenant Id** to "00000000-0000-0000-0000-000000000000"  
set **Credentials Resource** to "https://appnamegoeshere.azurewebsites.net"  
set **Credentials Client Id** to "00000000-0000-0000-000000000000"  
set **Credentials Client Secret** to "00000000000000000000000000000000"  
[add action]

**Execute REST Service**

REST operation: **GetOAuthToken**

Inputs:

Name	Type	Expression	...
tenantId	String	Credentials.TenantId	...
resource	String	Credentials.Resource	...
clientId	String	Credentials.ClientId	...
clientSecret	String	Credentials.ClientSecret	...

Assign return to: **TokenResultString**

## 5. Map the result into the Response Object

Just like before, we're going to take the string response and map it to the Response object we defined in step 1 – then, we're able to get to the bits we need to use in requests.


ParseTokenResultString
 Enabled

---

**Map Data**

Source type:

Source expression:

Target expression:








---

Ignore data shape errors:  

Ignore casting errors:  

Case insensitive match:  




6. When placing requests, pass in the appropriate “Authorization” header value

The header should be used as “{token\_type} {access\_token}” – which will look something like “Bearer 000...000”.

	<input type="text" value="SetValue1"/>	<input checked="" type="checkbox"/> Enabled
<b>Set Value</b>		
Show name in tree:	<input type="checkbox"/>	
Field:	<input type="text" value="Token"/>	
Expression:	<pre>Concat(TokenResult.token_type, " ", TokenResult.access_token)</pre> 	

The screenshot shows the IrServer interface with the following configuration:

- REST Operation** section:
  - REST service: **IrServerRestEndpoint**
  - Operation inputs: **ApplyRules** (selected in a dropdown menu)
  - Table of inputs:
 

Name	Type
authorizationHeader	Text
ruleAppName	Text
entityName	Text
entityState	Text
- Request** section:
  - Verb: **Post**
  - URI template: **ApplyRules**
  - Headers:
 

Name	Value
Authorization	<b>\$authorizationHeader\$</b>
Accept	<b>application/json</b>
  - Body format: **JSON**
  - Body content (JSON):
 

```
{
    "RuleApp": {
        "RepositoryRuleAppRevisionSpec": {
            "RuleApplicationName": "$ruleAppName$"
        }
    },
    "EntityName": "$entityName$",
    "EntityState": "$entityState$"
}
```

...and that's it! Using that process, you can retrieve a token to be used later in your Rule App to authenticate API requests to protected endpoints.

Hopefully this series has been helpful, and has been sufficiently technical without being overwhelming. If you run into issues, don't hesitate to reach out to our Support team or engage ROAD Services, and we'll make sure you're able to implement the functionality your business needs.