Microsoft

# Azure AD B2C Custom Policies

## Structuring Policies and Managing Keys

Microsoft Corporation
Published: September 2018
Version: 0.9 (DRAFT)

Author: Philippe Beraud (Microsoft France)
Contributors/Reviewers: Marcelo di Iorio (Microsoft Spain), Kim Cameron, Brandon Murdoch, Ronny Bjones, Jose Rojas (Microsoft Corporation)

For the latest information on Azure Active Directory, please see
http://azure.microsoft.com/en-us/services/active-directory/



Abstract: Azure AD, the Identity Management as a Service (IDaaS) cloud multi-tenant service with proven ability to handle billions of authentications per day, has extended its capabilities to manage consumer identities with a service for Business-to-Consumer (B2C): Azure AD B2C.

Azure AD B2C is "IDaaS for Customers and Citizens" designed with Azure AD privacy, security, availability, and scalability for customer/citizen identity and access management (CIAM). It's a comprehensive, cloud-based, 100% policy driven solution where declarative policies encode the identity behaviors and experiences as well as the relationships of trust and authority inside a Trust Framework (TF).

Whilst the built-in policies in Azure AD B2C leverage a dedicated TF tailored by Microsoft, i.e. the "Microsoft Basic Trust Framework" in which you can set up for your configuration these predefined policies, the custom policies give you full control, and thus allows you to author and create your own Trust Framework through declarative policies. They thus provide you with all the requirements of an Identity "Hub".

This document is intended for IT professionals, system architects, and developers who are interested in understanding the most advanced capabilities Azure AD B2C provides, and more especially in this context how to manage their own (Trust Framework) custom policies along with all the associated keys.

# Table of Content

# Notice

This document covers the custom policies[1] now available for evaluation under public preview for all Azure Active Directory B2C (Azure AD B2C) customers. Custom policies are designed primarily for advanced identity pros/developers who need to address the most complex identity scenarios.

This feature set indeed requires developers to configure the Identity Experience Framework (mostly) directly via XML file editing. This method of configuration is powerful but more complex. Advanced identity pros/developers using the Identity Experience Framework should plan to invest some time completing walk-throughs and reading the online reference documentation beyond this series of documents.

For most scenarios, we recommend that you use the built-in policies[2] of Azure AD 2BC. Built-in policies are easier to set up for your configuration. You can use built-in and custom policies in the same Azure AD B2C tenant.

**As of this writing, custom policies are in public preview and may be substantially modified before GA. For information, see RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW[3].**

**This document will be updated to reflect the changes introduced at GA time for custom policies.**

**This document reflects current views and assumptions be of the date of development and is subject to change. Actual and future results and trends may differ materially from any forward-looking statements. Microsoft assumes no responsibility for errors or omissions in the materials.**

**THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.**

---

[1] AZURE ACTIVE DIRECTORY B2C: CUSTOM POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-overview-custom

[2] AZURE ACTIVE DIRECTORY B2C: BUILT-IN POLICIES: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-reference-policies

[3] RELEASE NOTES FOR AZURE ACTIVE DIRECTORY B2C CUSTOM POLICY PUBLIC PREVIEW: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-developer-notes-custom

# Introduction

Azure AD B2C is a cloud identity service for your consumer-facing web and mobile applications. Azure AD B2C is designed to solve the identity management challenges that have emerged, as economic and competitive pressures drive commercial enterprises, educational institutions, and government agencies to shift their service delivery channels from face-to-face engagements to online web and mobile applications.

Based on standardized protocols, Azure AD B2C is "IDaaS for Customers and Citizens" designed with Azure AD privacy, security, availability, and scalability for customer/citizen identity and access management (CIAM). The "secret sauce" of Azure AD B2C to achieve the above objectives resides in the 100% policy driven Identity Experience Framework that consume fit to purpose declarative policies.

Many of the most frequently used identity use cases can be addresses using the B2C extension in the Azure portal as the developer control surface. However, there some advanced features only available by writing custom user journeys which must be configured directly into policy XML files and uploaded to the B2C tenant. Access to this incremental feature set is available via the custom policies in Azure AD B2C as per currently available public preview.

> **Note** For a basic level of proficiency with the policy configuration available directly in the B2C Admin portal, see the introduction video Business-to-Consumer Identity Management with Azure Active Directory B2C[4] where all the relevant B2C Admin **portal** settings are.

## Objectives of this document

This third document is intended as an overview document for discovering and understanding how the advanced capabilities of Azure AD B2C can be leveraged to author and manage your own custom policies (along with the associated security keys) for your custom user journeys.

## Non-objectives of this paper

This series of documents is not intended as an overview document for the Azure AD offerings but rather focusses on this Azure AD B2C identity service, and more specifically on the custom policies that are now in public preview.

> **Note** For additional information, see the article Getting started with Azure AD[5]. As well as the whitepapers Active Directory from the on-premises to the cloud[6] and An overview of Azure AD[7] as part of the same series of documents.

[4] Business-to-Consumer Identity Management with Azure Active Directory B2C: https://channel9.msdn.com/Events/Build/2016/P423
[5] Getting started with Azure AD: https://docs.microsoft.com/en-us/azure/active-directory/get-started-azure-ad
[6] Active Directory from the on-premises to the cloud: https://aka.ms/aadpapers
[7] An overview of Azure AD: https://aka.ms/aadpapers

# Organization of this paper

To cover the aforementioned objectives, this document of the series is organized in the following four sections:

- IMPLEMENTING AND MANAGING YOUR POLICIES FOR CUSTOM USER JOURNEYS.
- MANAGING YOUR APPLICATION REGISTRATION.
- MANAGING YOUR KEYS FOR THE CUSTOM POLICIES.
- MANAGING USERS AND THEIR ATTRIBUTES.

These sections provide the information details necessary to understand the new capabilities introduced by the custom policies in Azure AD B2C, and successfully build and manage your own custom policies based on the already available features as per the currently available public preview.

# About the audience

This document is intended for IT professionals, system architects, and developers who are interested in understanding the advanced capabilities Azure AD B2C provides with all the requirements of an Identity "Hub", and in this context how to manage their own custom policies and all the associated keys.

# Implementing and managing your policies for custom user journeys

Azure AD B2C provides administrative interfaces to create and publish your custom policies to define your more advanced custom user journeys.

Based on the shared understanding of both the nature and structure of the policies that constitute a Trust Framework, this section describes all the operations that relate to them and provides guidance and best practices in this context.

## Understanding interaction with policies configured via portal UI

Custom policies in Azure AD B2C rely on your own Trust Framework configuration as discussed below and therefore share only a few characteristics with built-in policies that are configured via the B2C admin portal which rely on a common multi-tenant Trust Framework.

Custom policies inherit configuration settings from each other.

The portal configured built-in policies inherit from hidden files not available to the developer where as any custom policies based on the core templates of "Started Pack" (see the second document of this series) **inherit and extend** the settings in the custom base policy coming from the core template, e.g. the *TrustFrameworkBase.xml* policy XML file.

Beyond the above custom base policy file, each core template contains:

- A custom extensions policy file. This file named *TrustFrameworkExtensions.xml* is where most configuration changes are made for the policy as the way is structured.
- Relying party (RP) policy files. These are task-specific files called by your application: sign-up or sign-in file (*SignUpOrSignin.xml*), profile edit file (*ProfileEdit.xml)*, password reset file (*PasswordReset.xml*), etc.

**Note** For more information, see article AZURE ACTIVE DIRECTORY B2C: CUSTOM POLICIES[8] as well as the sixth document of this series.

**We DO recommend that you do not intermix custom and built-in policies.**

A web or mobile application, however, may invoke any type of policy, for example a custom policy for signup and a portal configured policy for sign-in. Claims providers, extended attributes, MFA on/off, and UI customization are a few of the features that must be configured separately between custom policies and portal configured policies.

[8] AZURE ACTIVE DIRECTORY B2C: CUSTOM POLICIES: https://aka.ms/aadb2ccustom

# Authoring your policies for the custom user journeys

Azure AD B2C is flexible enough to accommodate various situations in terms of identity use cases. The next sections deal with the most common situations you may encounter and the role you may play in such a context.

## Authoring your custom policies as an organization

Considering the above, let's illustrate a typical logical provisioning workflow. Let's suppose that, at this stage, your organization is in the process to define, create and deploy suitable policies to constitute its own Trust Framework.

### Leveraging the inheritance model

The various policies to define and create to accommodate various requirements, constraints, and needs should leverage the layering approach permitted by Azure AD B2C thanks to the inheritance model rather than defining everything in the policy XML file and starting again and again from scratch.

The inheritance model is as follows:

- The parent policy and child policy are of the same schema and flexibility.
- Child policy can inherit the parent policy functionality and '*extend it*'.
- Adding a new element effectively extends the parent policy.
- There is NO limit on the number of levels.

And, in terms of override:

- The child policy can override 'allowed' aspects of the parent policy.
- Same element in the child policy and parent policy implies, child policy is overriding that element of the parent policy.

**Important note** In terms of security boundary, because the parent policy and child policy could span across tenants, References to cryptographic key material (see section § *Managing your keys for the custom* policies later in this document) should be kept within the boundary of the organization that owns them.

Considering the above, when you subscribe to Azure AD B2C, and after configuring the "Starter Pack" in accordance to the second document of this series, you have uploaded in your B2C tenant the following policies:

1. A *B2C_1A_TrustFrameworkBase* base policy, e.g. the *TrustFrameworkBase.xml* policy XML file of the core templates of the "Starter Pack", that mainly defines the core elements that Azure AD B2C can basically leverages basically leverages for your custom policies that will inherit from it: core claims schema and transformations definitions, some key claims providers, and core user journeys.
2. A *B2C_1A_TrustFrameworkExtensions* custom policy, e.g. the *TrustFrameworkExtensions.xml* policy XML file of the core templates of the "Starter Pack", that simply inherits from the above policy. This

second policy constitutes a kind of isolation layer on which to ground any Trust Framework (custom policies). It constitutes the "exposed" base policy.

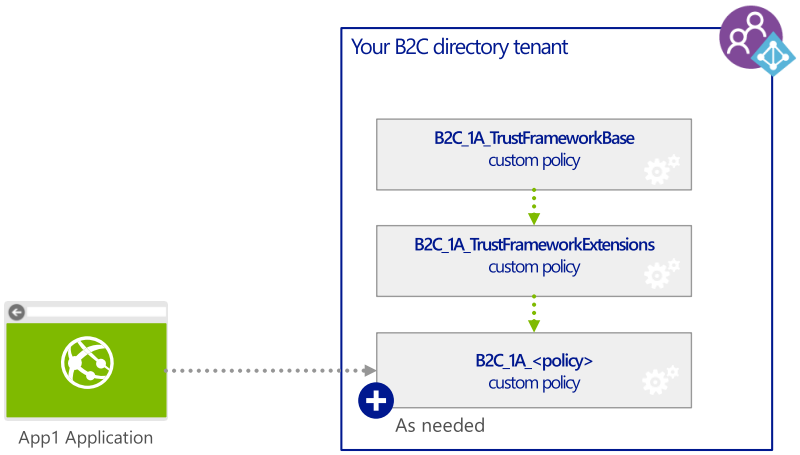
**The former *B2C_1A_TrustFrameworkBase* base policy certainly constitutes a more than valuable resource to learn how the various XML elements are defined in addition to this document and the sixth document of this series. We highly recommend leaving this file untouched.**

**The latter one, i.e. the *B2C_1A_TrustFrameworkExtensions* custom policy, should really be considered as the starting point of your own Trust Framework (policies). We advise to use it to all the core and common additional elements you may need for your own custom policies: common additional claims schema, key claims providers you will need, etc.**

The additional two or three custom policies that come with each core template of the "Starter Pack" inherit from this extensions policy, e.g.:

1. B2C_1A_ProfileEdit (*ProfileEdit.xml* policy XML file).
2. B2C_1A_PasswordReset (*PasswordReset.xml* policy XML file).
3. B2C_1A_signup_signin (*SignUpOrSignIn.xml* policy XML file).

As illustrated hereafter.




```
<TrustFrameworkPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
                      PolicySchemaVersion="0.3.0.0"
                      TenantId="litware369b2c.onmicrosoft.com"
                      PolicyId="B2C_1A_<policy>"
                      PublicPolicyUri="http://litware369b2c.onmicrosoft.com/B2C_1A_<policy>">
  …
```

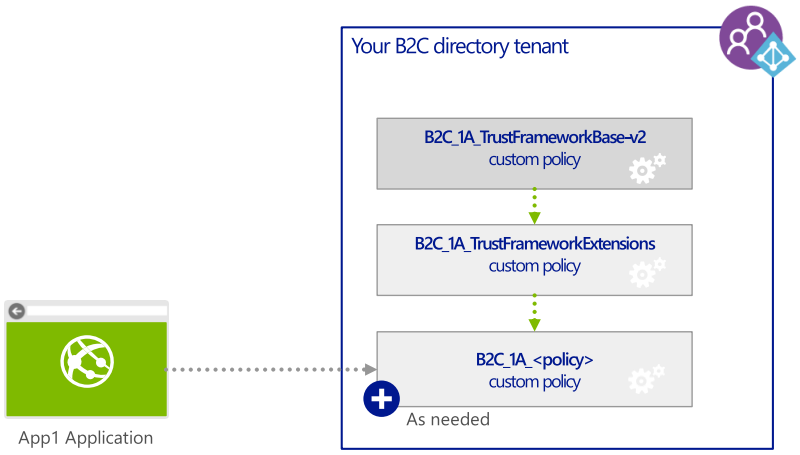
```
  <BasePolicy>
    <TenantId>litware369b2c.onmicrosoft.com</TenantId>
    <PolicyId>B2C_1A_TrustFrameworkExtensions</PolicyId>
  </BasePolicy>
```

```
  …
</TrustFrameworkPolicy>
```

In the lifecycle of a Trust Framework, the base policy indeed will certainly evolve over the time and there are consequently need for change version control. This may have significant consequences and should be indicated by a change in the version. You thus may require a new version to be published with the update

version of the *B2C_1A_TrustFrameworkBase* policy: let's' say a new *B2C_1A_TrustFrameworkBase-v2* policy instead of the *B2C_1A_TrustFrameworkBase* policy.

In such circumstances that will certainly occur over the time, having the *B2C_1A_TrustFrameworkExtensions* policy between the *B2C_1A_<policy>* policies and the *B2C_1A_TrustFrameworkBase* or now the *B2C_1A_TrustFrameworkBase-v2* policy can introduce a well-suited flexibility in your ability to version your foundational base policy. Having such a policy in place allows you simply changing the base policy reference in the *B2C_1A_TrustFrameworkBase* without having to update any of the *B2C_1A_<policy>* policies sitting below in the inheritance hierarchy in terms of reference to core definitions.




Considering the above, the recommended inheritance model will be as follows for a specific relying party (RP) *B2C_1A_<policy>* policy is the one suggested by the core templates of the "Starter Pack":

- The *B2C_1A_<policy>* inherits from the *B2C_1A_TrustFrameworkExtensions* policy.
- The *B2C_1A_TrustFrameworkExtensions* policy inherits in turn from the *B2C_1A_TrustFrameworkBase* policy.

## Authoring your policies as a Trust Framework provider

Considering what has been covered so far, Azure AD B2C has the ability address the unique identity management problems of different communities of interest. Industry has indeed many verticals that need to comply with specific regulations: e.g. applications dealing with financial transactions, applications dealing with patient health data, applications dealing with defense, etc. We often find the need to create policies that help those industries meet the required regulations.

Azure AD B2C achieves this versatility by being architected as you already know as a 100% policy driven Identity Experience Framework. Policies and counterpart policy XML files are designed to enable defining such policies, that various applications could use and thereby be compliant. **100% of everything is defined in the policy XML file (and its parent policy XML files). Nothing is hardcoded anywhere else outside of the policies.**

This engine can support a broad spectrum of digital identity ecosystems by enforcing different Trust Frameworks (TFs) tailored to the needs of individual communities of interest.

As such, Azure AD B2C provides thus the ability to manage data on behalf of participants in the above communities of interest. Federated identity provides a basis for achieving end user identity assurance at Internet scale. By delegating identity management to 3rd parties, a single digital identity for an end user can be re-used with multiple relying parties.

Let's consider how you can author a particular TF as a Trust Framework Provider (TFP) to allow governing a community of interest.

Multiple Trust Frameworks can be published by a TFP. **The intention could aim at establishing a catalog/marketplace of TFs that are suitable for a variety of communities of interest.**

> **Note** A TFP may choose to author a given TF for a single LOA and LOP in which case, the related policies may only include user journeys associated with that LOA and LOP. It should also be mentioned that a policy may have multiple user journeys for the same LOA and/or LOP simply because to achieve that LOA/LOP requirements, some claims providers can only be used with certain other claims providers, and thus user journeys may have to be formed for different combinations.

> **Important note** All the *certification program* aspects that may relate to a trust framework are outside the scope of Azure AD B2C. These aspects are covered using existing, out-of-band processes under the responsibility of the TFP that publishes a TF.

## Leveraging the inheritance model as a TFP

The aforementioned considerations for an organization that authors its own Trust Framework (policies) also apply here for the TFP.

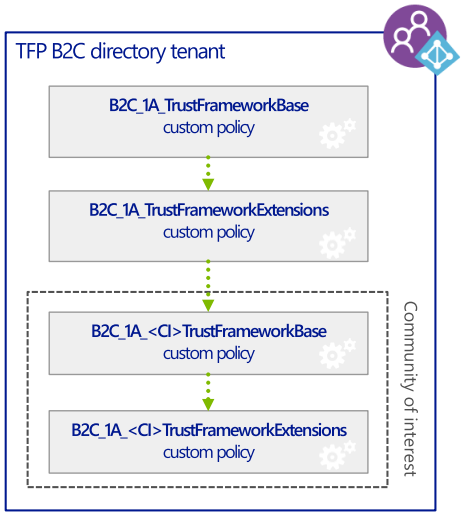
When the TFP subscribes to Azure AD B2C, and after configuring the "Starter Pack" in accordance to the second document of this series, the TFP will have uploaded in its B2C tenant the following two policies:

1. A *B2C_1A_TrustFrameworkBase* base policy, e.g. the *TrustFrameworkBase.xml* policy XML file of the core templates of the "Starter Pack", that mainly defines the core elements that Azure AD B2C can basically leverages.
2. A *B2C_1A_TrustFrameworkExtensions* custom policy, e.g. the *TrustFrameworkExtensions.xml* policy XML file of the core templates of the "Starter Pack", that simply inherits from the above policy and constitutes an isolation layer on which to ground any Trust Framework (policies) for the TFP.

To author a particular TF to allow governing a community of interest (CI), we do recommend for the TFP defining the following two policies:

1. A *B2C_1A_<CI>TrustFrameworkBase* base policy that inherits from the above *B2C_1A_TrustFrameworkExtensions* policy. This policy will mainly define all the core and common additional elements the TFP may need for a particular policy: common additional claims schema, key claims providers you will need, etc.
2. And in turn a *B2C_1A_<CI>TrustFrameworkExtensions* policy that will simply inherit from the above *B2C_1A_<CI>TrustFrameworkBase* policy. This second policy constitutes the same kind of isolation layer as the one mentioned above.

**The above policies are located in the TFP's B2C tenant as illustrated hereafter.**




# Authoring your policies as a relying party

As a participant in the community of interest, a specific relying party can express its technical policies for its application. These policies are referred as to *relying party (RP) policies*.

A relying party (RP) *B2C_1A_<policy>* policy will directly inherit from *B2C_1A_<CI> TrustFrameworkExtensions* policy located in the TFP B2C tenant - and thus leverages – all the elements defined in this policy provided by the TPF.

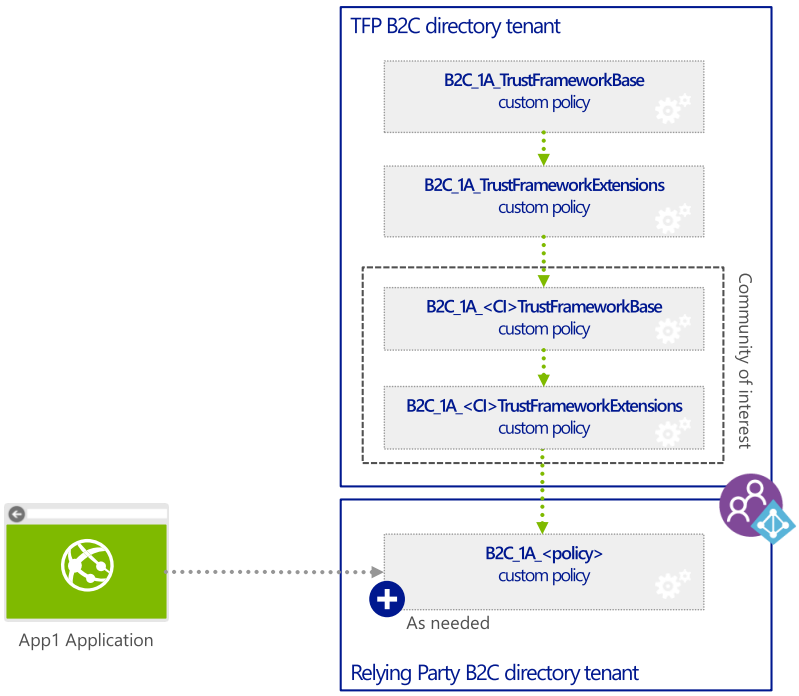
In other words, such a TF *B2C_1A_<policy>* policy builds on top of a technical policy in a given TFP.

```
<TrustFrameworkPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
                      PolicySchemaVersion="0.3.0.0"
                      TenantId="litware369b2c.onmicrosoft.com"
                      PolicyId="B2C_1A_<policy>"
                      PublicPolicyUri="http://litware369b2c.onmicrosoft.com/B2C_1A_<policy>">
  …
  <BasePolicy>
    <TenantId>"<TFP>.onmicrosoft.com</TenantId>
    <PolicyId>B2C_1A_<CI>TrustFrameworkExtensions</PolicyId>
  </BasePolicy>
  …
</TrustFrameworkPolicy>
```

They will allow the relying parties belonging to the community of interest to author their own relying party *B2C_1A_<policy>* policies.

Such an approach greatly facilitates the onboarding of a given relying party application onto the "identity information exchange network" of the community of interest. Such *B2C_1A_<policy>* policies not only operate with the constraints of the Trust Framework (*B2C_1A_<CI>TrustFrameworkExtensions* policy) but also enables the relying party to further refine its template-based application policy to accommodate its own requirements by for instance restricting its users' experiences to, e.g. a subset of claims that it is interested in, constraint on the claims providers that can be used, etc.

**The relying party *B2C_1A_<policy>* policies are located in the relying party's B2C tenant as illustrated hereafter.**




Azure AD B2C provides as part of its administrative PowerShell cmdlets interface the tooling for a relying party to manage in Azure AD B2C its own policies on that basis as expressed as per policy XML files. A relying party can publish any number of policy XML files referring to various definitions of the Trust Framework that the relying party has subscribed to. The relying party can update or delete its own policies at any time.

On that basis, developers and/or security officers are then able to configure the relying party's application(s) for automated digital identity information request processing by the Azure AD B2C service under the rules of the above policies.

# Manually managing your custom policies

This section depicts how to conduct the various operations that relate to the management of the custom policies.

As one can imagine, Azure AD B2C administrative interface provides standard CRUD features for publishing and updating policies. These operations are realized through the B2C admin portal.

## Uploading a specific custom policy

To upload a custom policy in your B2C tenant, proceed with the following steps:

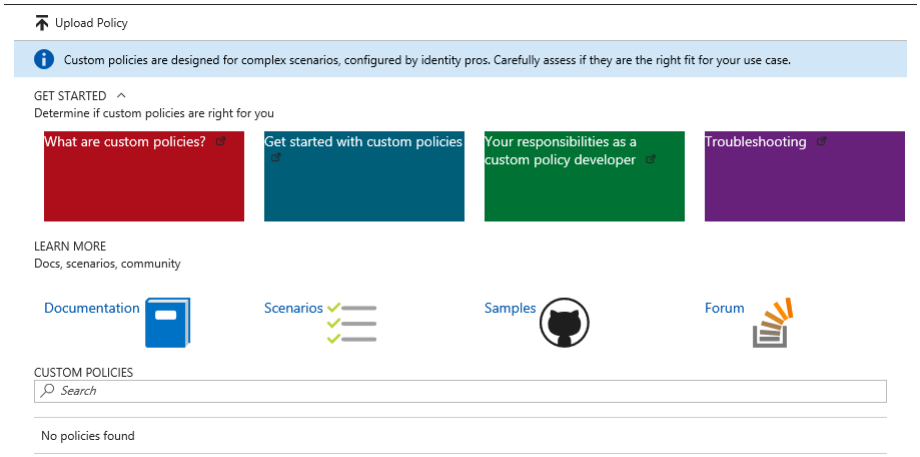
1. Open a browsing session and navigate to the Azure AD B2C blade using the following URL:

https://portal.azure.com/<*your_b2c_tenant*>.onmicrosoft.com/?Microsoft_AAD_B2CAdmin=true#blade/Microsoft_AAD_B2CAdmin/TenantManagementBlade/id/<*your_b2c_tenant*>.onmicrosoft.com

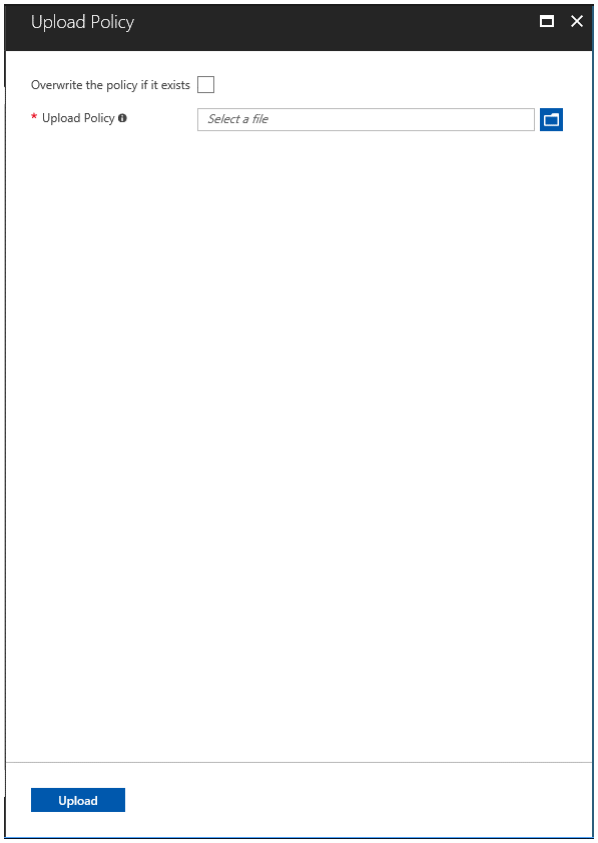
where *your_b2c_tenant* the name of your B2C tenant, for example:

https://portal.azure.com/litware369b2c.onmicrosoft.com/?Microsoft_AAD_B2CAdmin=true#blade/Microsoft_AAD_B2CAdmin/TenantManagementBlade/id/litware369b2c.onmicrosoft.com

2. Sign in with administrative credentials of that tenant, for example admin@litware369b2c.onmicrosoft.com in our illustration for the tenant admin.
3. Click **Browse** | **Azure AD B2C**. The Azure AD B2C blade opens up in the Azure portal.
4. Select **Identity Experience Framework**.




5. Click **Upload Policy** at the top of the blade. An eponym blade opens up.



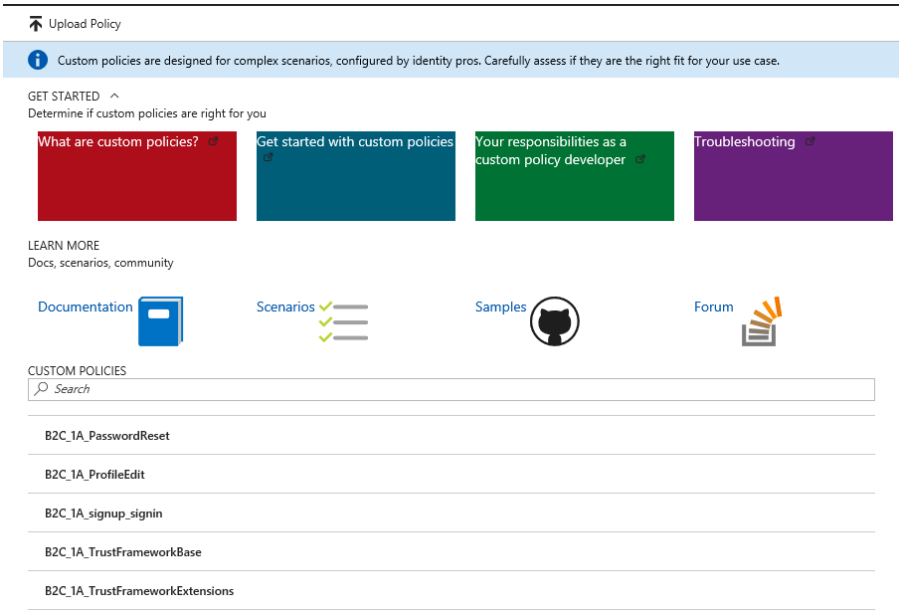

6. In **Upload Policy**:
    a. Press the folder icon to select the custom policy XML file to upload.

b. Leave the option to **overwrite the policy if it exists** unchecked to make sure you do not overwrite an existing policy with the same name unintentionally. Conversely, check **overwrite the policy if it exists** to overwrite it.

7. Click **Upload**.
8. When the custom policy is uploaded, the name is prepended with *B2C_1A_* for "Advanced". This is done to differentiate between the policies created by the Azure AD B2C blade and the handcrafted ones.

## Viewing all your custom policies

Once loaded, the custom policy are listed in the **Identity Experience Framework** blade.

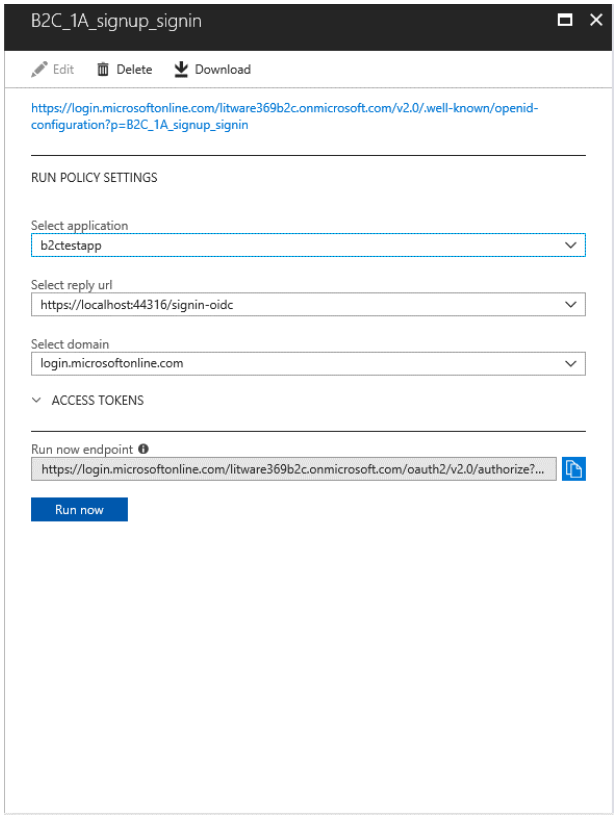



All your custom policies are prefixed by *B2C_1A_* with A standing for "Advanced".

## Downloading a specific custom policy

To download a specific custom policy, proceed with the following steps:

1. From the **Identity Experience Framework** blade in the Azure portal, click the custom policy to download. A related blade opens up.




2. Click **Download**.
3. Click **Save** or **Save as** to save the policy XML file.

You can refer to the fourth and sixth documents of this series for information, guidance, and a technical reference on how to edit and customize the policy XML file to model your advanced identity use case.

## Deleting a specific custom policy

To delete a specific custom policy, proceed with the following steps:

1. From the **Identity Experience Framework** blade in the Azure portal, click the custom policy to delete. A related blade opens up.
2. Click **Delete**.

# Programmatically managing your custom policies

Your custom policies can also be programmatically managed by using PowerShell or by using Graph API. The next two sections review in order these two options.

# Programmatically managing your custom policies using PowerShell

The following PowerShell code snippet illustrates how to programmatically add a custom policy and overwrite it if it already exists:

```
param(
[string] $tenantname = "litware369b2c.onmicrosoft.com",
[string] $policyPath = $(throw "-policyPath is required."),
[string] $username = $(throw "-username is required."),
[secureString] $password = $(Read-Host "Password" -AsSecureString)
)

$cred = New-Object System.Management.Automation.PSCredential ($username, $password)
Login-AzureRmAccount -Credential $cred

$context = Set-AzureRmContext -tenantId $tenantname -Name B2C -Force
$context | Select-AzureRmContext | Out-Null
$tenantId = $context.Tenant.tenantId
$token = $context.TokenCache.ReadItems() | Where-Object { $_.Resource -ilike "*/management.core.windows.net/*" -and
$_.AccessToken -ne $null -and $tenantId -ieq $_.Authority.Split('/')[3] } | sort -Property ExpiresOn -Descending | select
-First 1
$accessToken = $token.AccessToken

If (Test-Path $policyPath)
{
    Write-Output "Uploading policy: $($policyPath.Split('``\')[-1])"
    $policy = (Get-Content -Path $policyPath) -join "`n"

    Add-Type -AssemblyName System.Web
    $body = "<string
xmlns=`"http://schemas.microsoft.com/2003/10/Serialization/`">$([System.Web.HttpUtility]::HtmlEncode($policy))</string>"
    $headers = @{ "Authorization" = "Bearer $accessToken" }

    $response = $null
    $response = Invoke-WebRequest -Uri
"https://main.b2cadmin.ext.azure.com/api/trustframework?tenantId=$tenantname&overwriteIfExists=true&quot" -Method POST -
Body $body -ContentType "application/xml" -Headers $headers -UseBasicParsing
    if ($response.StatusCode -ge 200 -and $response.StatusCode -le 299)
    {
        Write-Output "Policy successfully uploaded"
    }
    else
    {
        Write-Output "Failed to upload policy"
    }
}
else
{
    Write-Error "Cannot find file: $policyPath"
}
```

The above snippet can be saved as *AddPolicy.ps1* PowerShell scripting file.

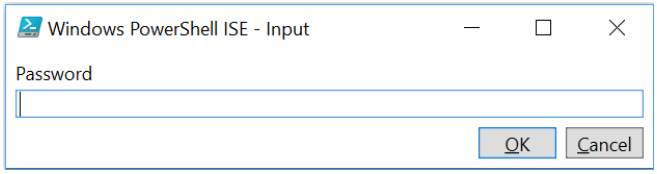
To upload the *SignUpOrSignin.xml* custom policy provided by the "Started Pack", proceed with the following steps:

1. Open a PowerShell command prompt or launch a Windows PowerShell ISE environment.
2. Run the following command:

```
.\AddPolicy.ps1 -policyPath "C:\Code\active-directory-b2c-custom-policy-
starterpack\SocialAndLocalAccounts\SignUpOrSignin.xml" -username philber@litware369b2c.onmicrosoft.com
```

Where the *policyPath* argument allows you to specify the policy file to upload and the *username* argument the user account to use for the operation.

3. When prompted, specify the credentials for the specified user account.




# Programmatically managing custom policies using Microsoft Graph API

## Introducing the Microsoft Graph API

The Microsoft Graph API enables you to build applications that connect users to their data. Like the Azure AD Graph API (see section § *Introducing the Azure AD Graph API* later in this document, it uses the relationships and objects defined in Active Directory and Azure AD, but can also operate over data held in Microsoft Office 365 applications such as Excel. You can use this API to build custom tools and utilities to help improve productivity. For example, you can use the Microsoft Graph API to view and maintain the associations between users and items such as their email, calendars, contacts, documents, devices, and other personal data sources.

The Microsoft Graph API facilitates building services that can automate tasks such as scheduling meetings, notify interested users if a document changes, analyze usage patterns over documents and data files, construct custom dashboards, and establish a user's organizational context (which department they work in, who is their manager, and so on).

**Note** For more information, see page WHAT CAN YOU DO WITH MICROSOFT GRAPH?[9].

This API has been updated to support the management of your custom policies in your B2C tenant.

To manage them, you send requests to the Microsoft Graph API using queries of the form:

https://graph.microsoft.com[/{*version*}/]{*resource*}?query-parameters

Where:

- *{version}* is the version of the API you are using. *v1.0* corresponds to the most recent implementation. However, in our specific context for custom policies, you do not specify any value.
- *{resource}* is with the object that you are referencing. Specify the **testcpimtf** endpoint the Microsoft Graph API: **testcpimtf/trustFrameworkpolicies** as per document Trust Framework (TF) policies[10]. This document exposes all the REST calls that are supported to manage custom policies in your B2C tenant.

As an example, this query lists all the custom policies uploaded in your B2C tenant:

https://graph.microsoft.com/v1.0/testcpimtf/trustFrameworkpolicies

[9] WHAT CAN YOU DO WITH MICROSOFT GRAPH?: https://developer.microsoft.com/en-us/graph/examples
[10] TRUSTFRAMEWORKPOLICIES: https://github.com/Azure-Samples/active-directory-b2c-graph-trustframework-policy/blob/master/Documentation-TrustFrameworkPolicy-API.pdf

You must provide a valid access token in the security header of the request.

**Note** For more information, see article GET ACCESS TOKENS TO CALL MICROSOFT GRAPH[11].


```
{
    "@odata.context": "https://graph.microsoft.com/testcpimtf/$metadata#trustFrameworkPolicies",
    "value": [
        {
            "id": "B2C_1A_PasswordReset"
        },
        {
            "id": "B2C_1A_ProfileEdit"
        },
        {
            "id": "B2C_1A_signup_signin"
        },
        {
            "id": "B2C_1A_TrustFrameworkBase"
        },
        {
            "id": "B2C_1A_TrustFrameworkExtensions"
        }
    ]
}
```


Likewise, this query gets the details of the B2C_1A_TrustFrameworkExtensions custom policy:

https://graph.microsoft.com/testcpimtf/trustFrameworkpolicies/B2C_1A_TrustFrameworkExtensions/$value

The content of the custom policy file is retrieved.

## Using the Microsoft Graph Explorer

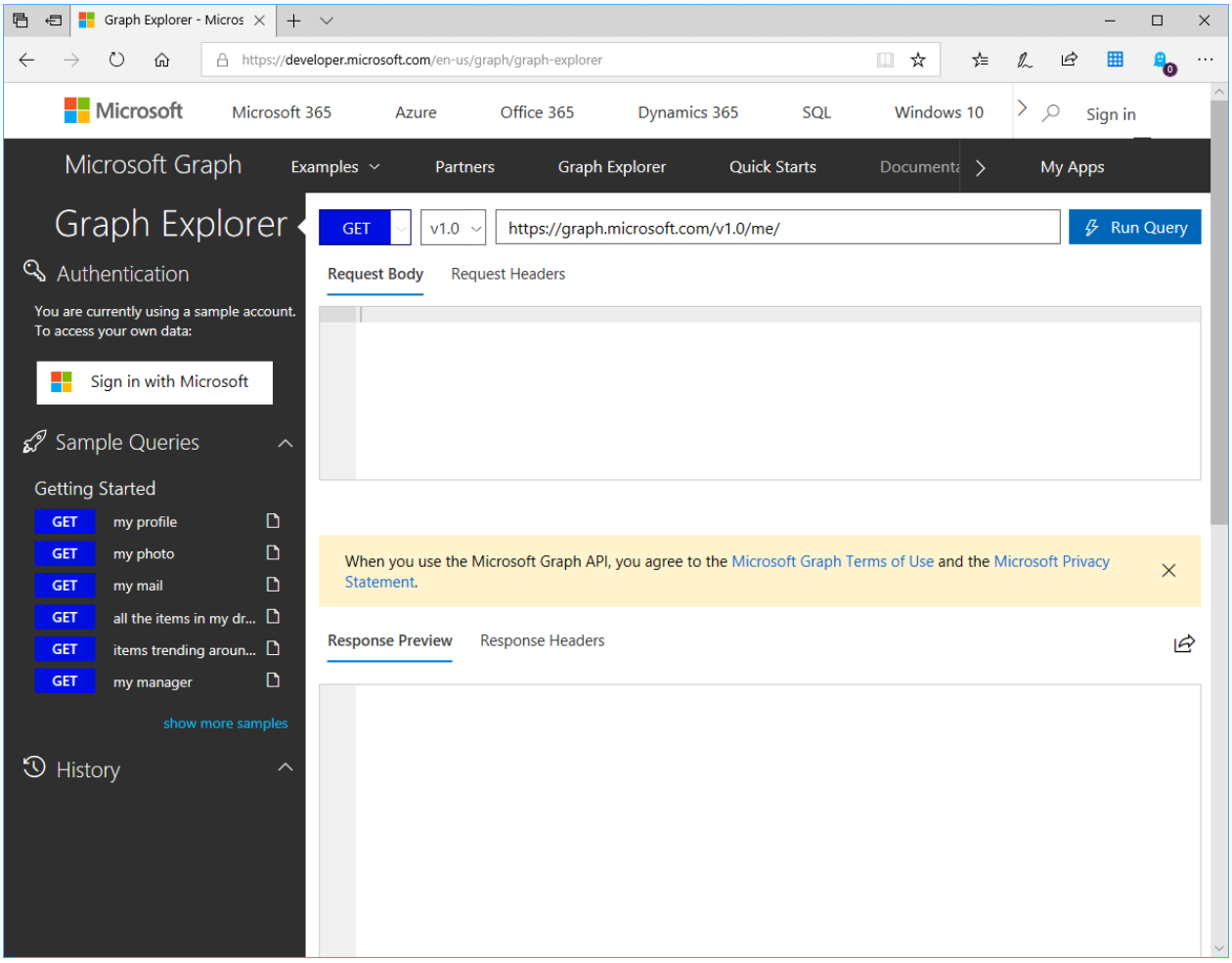
The Microsoft Graph Explorer is a web application that provides a friendly user interface to the Microsoft Graph API. It is available at https://developer.microsoft.com/graph/graph-explorer.

This web application basically sends HTTP requests to your B2C tenant and return results as an HTTP response. You provide your login credentials, and this information is included in the security header of each request as mentioned above.

To use the Microsoft Graph Explorer, perform the following steps:

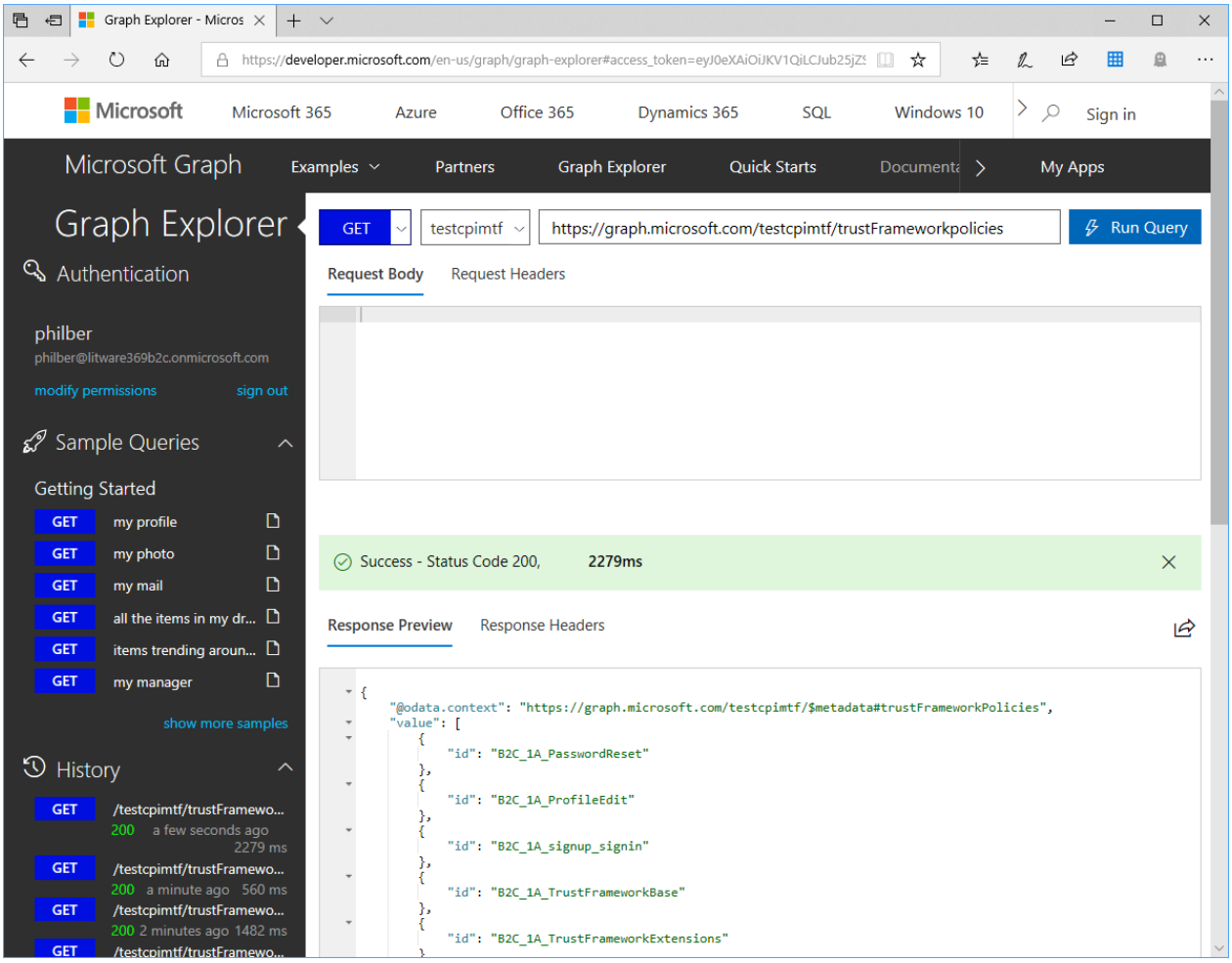
1. Open a browsing session and navigate to https://developer.microsoft.com/graph/graph-explorer.

[11] GET ACCESS TOKENS TO CALL MICROSOFT GRAPH: https://developer.microsoft.com/en-us/graph/docs/concepts/auth_overview




2. In the side bar, click **Sign in with Microsoft**. When invited, provide the credentials of an account with administrative access to your B2C tenant.
3. Click **Accept** to accept the permissions requested.
4. To run a query, select the **GET** verb, provide a URI that references the resources to find, such as https://graph.microsoft.com/v1.0/testcpimtf/trustFrameworkpolicies to find all custom policies in your B2C tenant, and then click **Run Query**.

The results will appear in the main body of the **Response Preview** window:

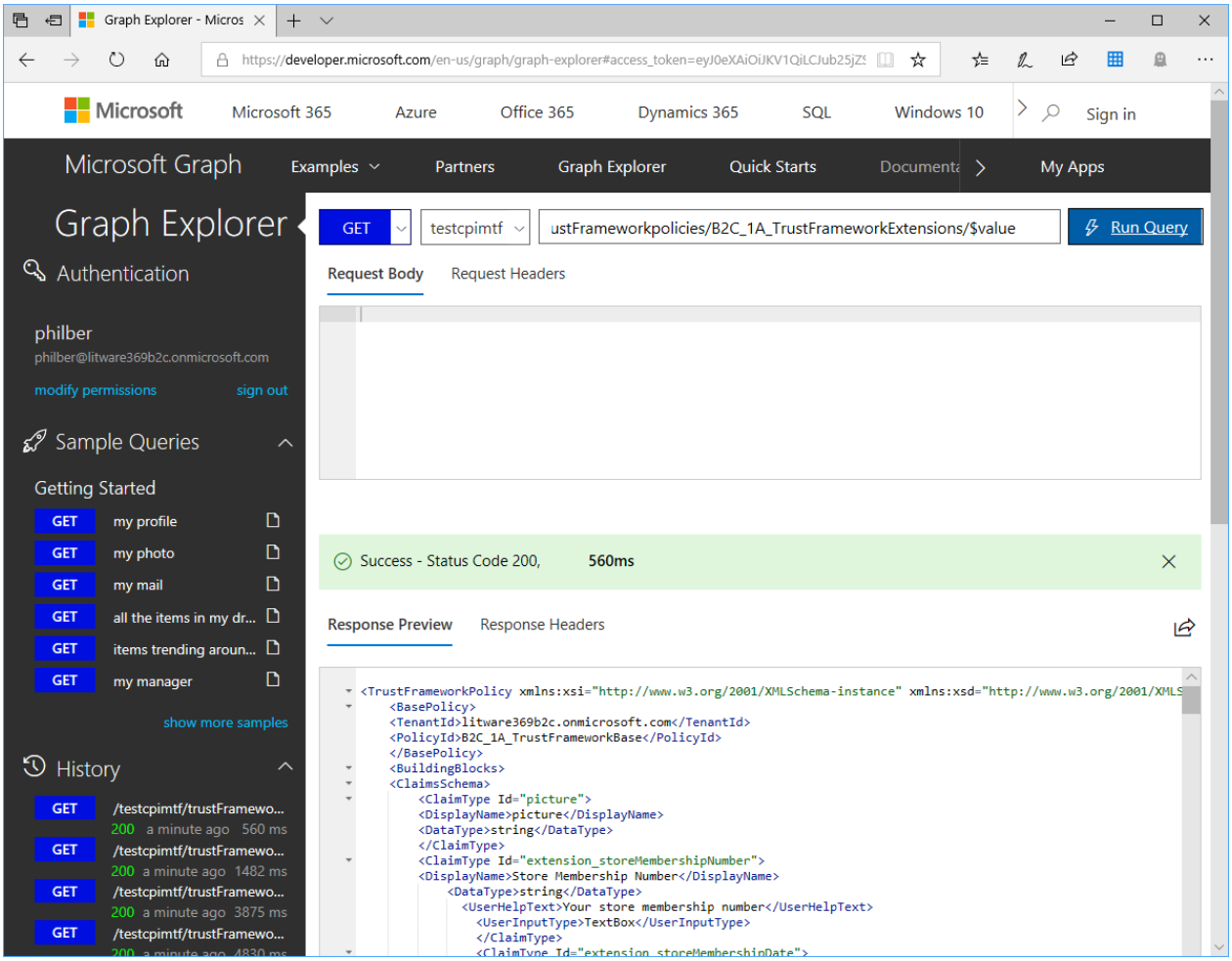



Note that you can see the HTTP header information included with the response in the **Response Headers** pane.

5. To perform an operation such as getting the content of an uploaded polices, specify for the URI of the resource:

https://graph.microsoft.com/testcpimtf/trustFrameworkpolicies/B2C_1A_TrustFrameworkExtensions/$value

For example, to get the B2C_1A_TrustFrameworkExtensions custom policy, and then click **Run Query**.




The above calls can also be made from a custom application. This is the purpose of the next section.

## Using the Microsoft Graph API from a custom application

For the sake of the illustration, we leverage here the B2CPolicyClient code sample that demonstrates managing custom policies. This code sample is available on GitHub: [Azure-Samples/active-directory-b2c-graph-trustframework-policy](Azure-Samples/active-directory-b2c-graph-trustframework-policy)[12].

It requires a global administrator account to run admin-level operations and to consent to application permissions.

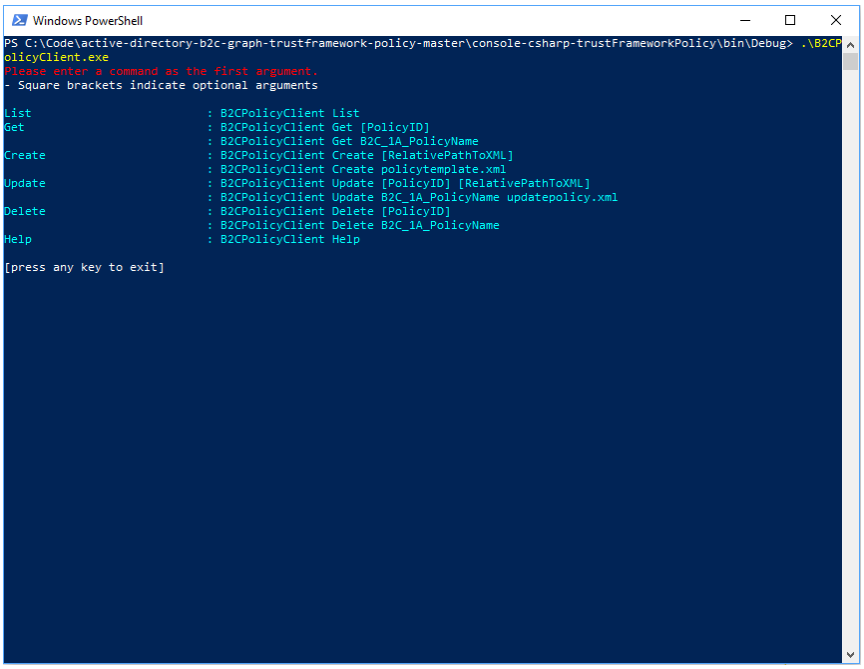
Proceed with the following steps:

1. Build the B2CPolicyClient code sample as per instructions in section § *Building the B2CPolicyClient code sample* in Appendix Building the code samples.
2. Open a PowerShell command prompt windows and navigate to the folder /bin/Debug for the B2CpolicyClient code sample.

[12] Azure-Samples/active-directory-b2c-graph-trustframework-policy**:** https://github.com/Azure-Samples/active-directory-b2c-graph-trustframework-policy

3. Run the following command:

```
PS> B2CPolicyClient.exe
```




4. This code sample demonstrates the followings:

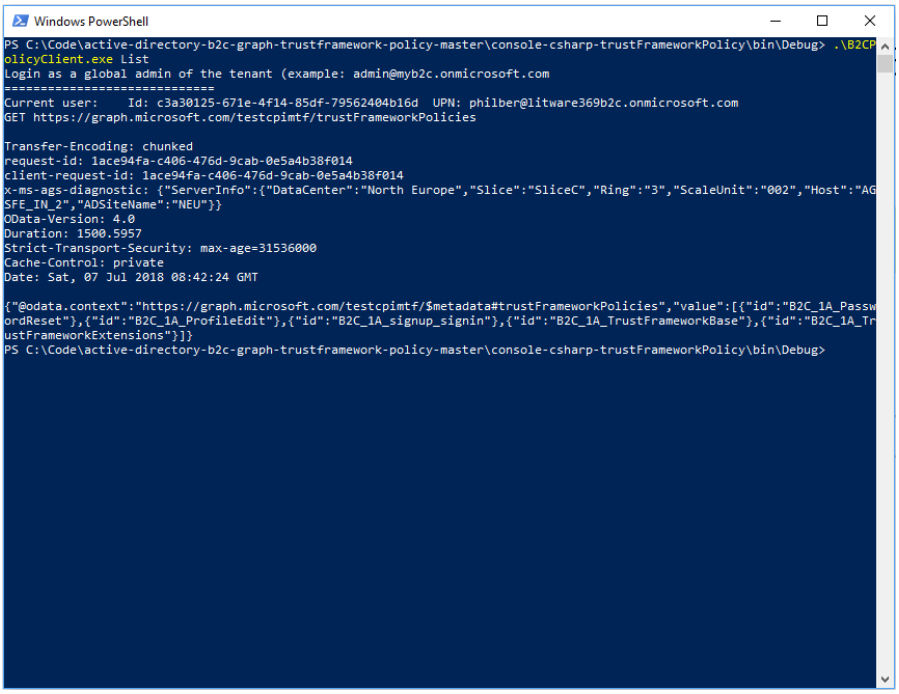
    a. List all custom policies (List).
    b. Read details of a custom policy (Get).
    c. Create a custom policy (Create).
    d. Update a custom policy (Update).
    e. Delete a custom policy (Delete).

5. Now run, for example, the following command:

```
PS> B2CPolicyClient.exe List
```

6. When prompted, sign in as a global administrator. For example, in our configuration philber@litware369b2c.onmicrosoft.com.

7. Click **Accept** to accept the requested authorizations. The output will show the list of policies.




This is the result of calling the Microsoft Graph API for custom policies, a.k.a. [Trust Framework (TF) policies][13].

[13] TRUSTFRAMEWORKPOLICIES: https://github.com/Azure-Samples/active-directory-b2c-graph-trustframework-policy/blob/master/Documentation-TrustFrameworkPolicy-API.pdf

# Managing your application registration

You have to register in your B2C tenant all the applications that will use custom policies and thus interact with the Identity Experience Framework in Azure AD B2C. This doesn't differ from "regular" Azure AD where any application that outsources authentication to Azure AD must be registered in a directory.

Application registration involves telling about your application, including the URL where it's located, the URL(s) to send replies after authentication, the URI to identify your application, and more.

This can be done both manually and programmatically. The next two sections consider both approaches. Let's start with the manual registration.

## Manually managing your application registration

This section shows in order how to view the registered applications and how to register a new application in your B2C tenant.

### Viewing all your registered applications

To list all the registered applications in your B2C tenant, proceed with the following steps:

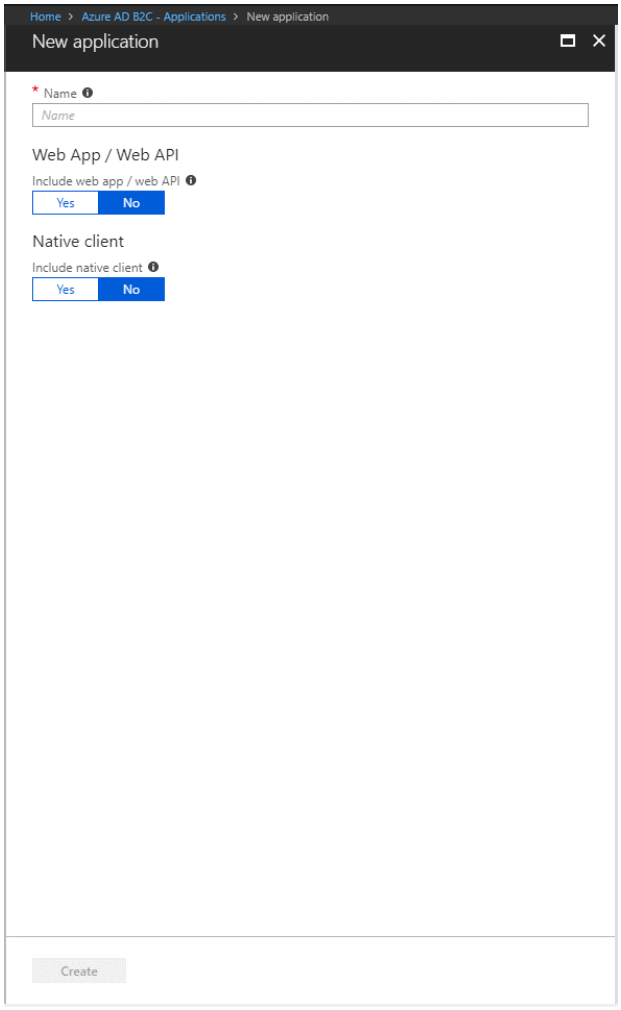
1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Applications**. A list of all the already registered applications in your B2C tenant is displayed.

| NAME | APPLICATION ID |
|---|---|
| jwt.ms website | 95b7953b-a499-48b9-bc1b-48bb095d6939 |
| b2ctestapp | 6c84237b-4fa1-47c0-a7eb-acea059ef99e |

### Registering a new application

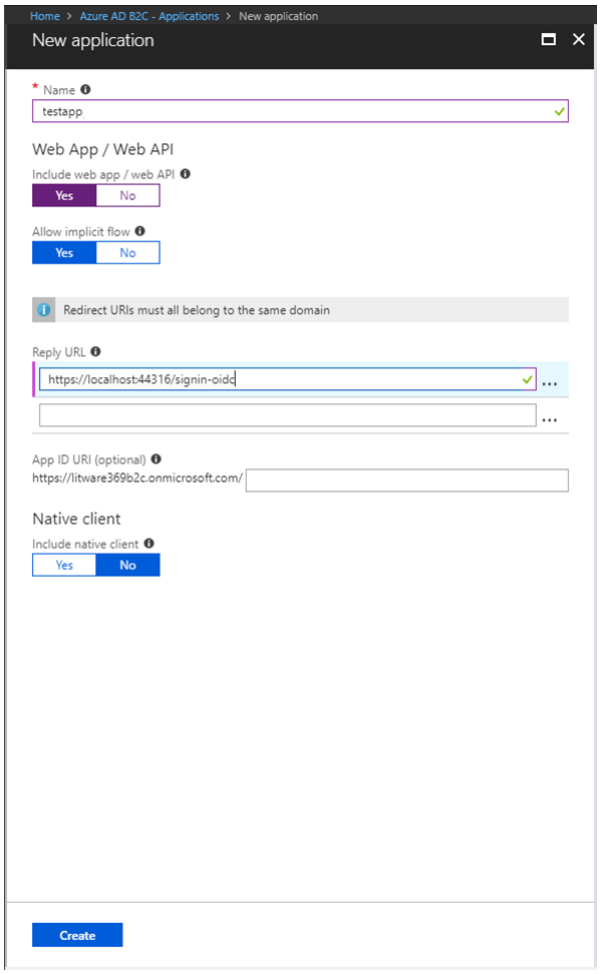
To register a new application, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Applications**.
3. Select **Add**. A **New application** blade shows up.




4. In **Name**, provide an application name that will describe your application, For example, in our illustration "*testapp*".
5. Depending on the nature of the applications, toggle the **Include web app / web API** and **Include native client** switches to the appropriated values, i.e. **Yes** or **No**. For example, in our illustration for a web application, **Include web app / web API** is set to **Yes**.
6. In **Redirect URIs**, specify the reply URL(s), i.e. the endpoint(s) where Azure AD B2C will return any tokens your application requests. For example, in our illustration "*http://localhost:44316/signin-oidc*".

**Note** If the application top register also includes a web API that needs to be secured, you'll want to create (and copy) an **Application Secret** as well by clicking **Generate key**.




7. Click **Create** to register your application.

# Programmatically managing your application registration

Programmatically managing your applications in your B2C tenant provide the ability to allow continuous integration / continuous deployment (CI/CD) to provision the applications in real time.

As of this writing, this can be achieved with the Beta endpoint the Microsoft Graph API (see section § *Introducing the Microsoft Graph API* earlier in this document).

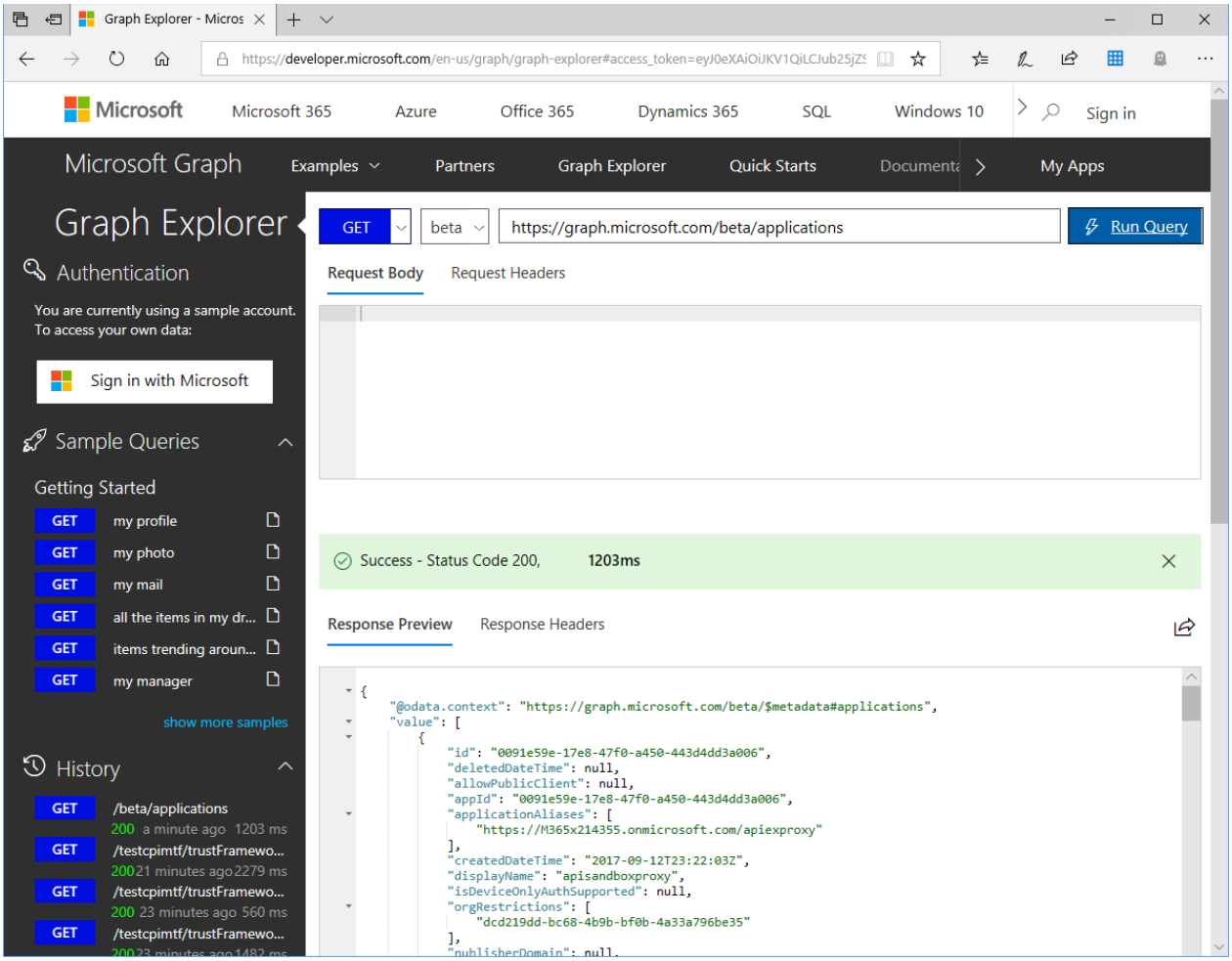
> **Note** The available documentation for beta is limited. For more information, see article APPLICATION RESOURCE TYPE[14].

APIs under the /beta version in Graph API are in preview and are subject to breaking change. Use of these APIs in production applications is NOT supported.

[14] APPLICATION RESOURCE TYPE: https://developer.microsoft.com/en-us/graph/docs/api-reference/beta/resources/application

To get the list of applications in your B2C tenant, run the following query:

https://graph.microsoft.com/beta/applications




**If you are interested in trying it out, please email aadb2cpreview@microsoft.com (please make sure to add the feature name, i.e. "Programmatically register B2C applications" ).**

# Managing your keys for the custom policies

This section deals with the key containers for which references have been specified in the policy XML files. It introduces and presents in this context what a key container is and details on how to upload keys in it.

## Introducing the key containers support in Azure AD B2C

As already introduced in this document as part of the technical profiles for claims providers and claims issuers, the Identity Experience Framework in Azure AD B2C leverages the RFC 7517 JSON Web Key (JWK)[15] standard for the key containers for a uniform key usage cross the service with, as a such, a support for different kind of cryptographic keys such as binary secrets, RSA keys, and X509 certificates.

JWK is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This aforementioned specification also defines a JWK Set JSON data structure that represents a set of JWKs. Azure AD B2C implements the basic readers, writers and sterilizers for the JWK standard.

### Understanding key set

A key set is a collection of keys. These keys can be binary secrets or RSA keys.

The following JSON snippet provides an example of a key set that contains binary secrets:

```
{"keys":
    [
      {"kty":"oct",
       "k":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
       "use":"sig",
       "kid":"FacebookClientSecret"
      },
      {"kty":"oct",
       "k":"AABCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7E3",
       "use":"sig",
       "kid":"FacebookClientSecret"
      }
    ]
}
```

This key set contains two key objects placed in the array “keys”. They are both of the key type “oct” which means binary secret. All keys are serialized in base64 Url encoding.

[15] JSON Web Key (JWK): https://tools.ietf.org/html/rfc7517

The following JSON snippet provides an example of a key set that contains a RSA private key:

```
{"keys":
    [
      {"kty":"RSA",
       "kid":"JwtTokenSigning",
       "use":"sig",
       "e":"AQAB",
       "d":"G2VGBWW2er3GgDY sh3M7oZyTDJQugOuIvWXU9dr9aXiSpt6rqeYMYJscb2KswL3YbZhc2BdIlFc5DMGyxR8bcRLOWZto-
           VbmWYvxE9P332VCZU4OHAincU3WzlU-gDsWNE21GV0hEG30wFJV2xCgARJxq35XUK33ro6XwDs7V5E",
      "n":"lPX72e7g2exGDDSwk9JxZP-RaLc8MennVuctuNi2ajKCnHp6eqxixxakZ4uGf11qP9zhEEseAONK3m-q2M96y7HJJ261f0Dom-
           lngR73iS3v0UWMjR_SGjnhv0HJ8zA7IaDCHAgdARJaUQ6Y6RP4Ft54Vfw39QJTrXMObQz7DrM",
      "p":"-vfl4t8R40NiiY_Bk7hNOYZqb6wt2zk-uvQzEeQ6VTGPRVTX52DGYGCU-bzCgwb4rMV2LmQS72gbgQEAgxwvSQ",
      "q":"l_KHk_syueUwJnYoUUprcYnHrZj2hQcXaBJk_8WpXAl6fFxbopsa4MMqGbLlHfMZgw01DyodeZAqT-HSrgeCGw",
      "dp":"PhA2Ao_lrY0JfjlVxsEqH9K02Z77zB3w_GzNgFwvbMgPxTaBGhAdntHjpYrfk2WS8671Ve2bnSbivllQFhxPOQ",
      "dq": "AChcQ6cLEm5OcnvE5vCw09lUv7Vc7B5pw83hFPLbjiDkY6Sk47JNgNYzAPvSiQlriVtGJtHgXfPsV1DgqxWO6Q",
      "qi":"Wg3mWikHWlpecQj9MA9UtDBs8KL0uaXe1DGatEsT1-gOWznbaUF_PIoSC8Xow3TC_TXdmdISfj7A5pJT_cOggw"},
    ]
}
```

This key set contains an RSA private key with all key parameters needed to perform efficient RSA operations.

It is always advisable to include all key components in the key because lacking one of them reduces performance in so far as all key parameters need to be generated during the key loading.

**As stated above, "oct" and "RSA" are currently the only key types supported.**

The following table describes the main parameters of a key. For the additional one, please refer to the standard specification:

| Parameter | Description |
|---|---|
| *Kty* | Indicate the type of key. This parameter is mandatory.<br>Values: "oct" or "RSA". |
| *Kid* | Provide a specific key an identifier which can be later. This parameter is optional.<br>In Azure AD B2C, kid corresponds to the *StorageReferenceId* value of *the Key* XML element, i.e. the machine understandable identifier that is used to uniquely identify a particular storage key container and reference it from other XML elements in the policy XML file. |
| *Use* | Specify the key's usage. This parameter is optional.<br>As of this writing, Azure AD B2C usage of oct and RSA keys is currently limited to signatures. |

Interestingly enough, key set allows to support rolling keys. Rolling key are used for signing keys and allow you to define in Azure AD B2C key sets with different generations of keys. A new key can be generated and published way ahead. Depending on policy and expiration, the old key will no longer be used and removed. Azure AD B2C can switch to the new key and a new generation key can be generated.

This same approach can also be used to revoke a key.

Now that you have an understanding of the key containers structure, let's consider how to manage them in Azure AD B2C.

# Managing your key containers

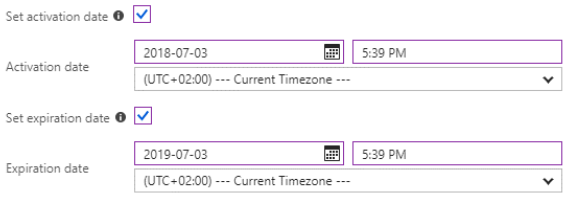
Key containers in Azure AD B2C are managed via the B2C admin portal.

**The second document of the series depicts how to proceed to get started with the "Starter Pack" for the custom policies in Azure AD B2C.**

## Generating a RSA key

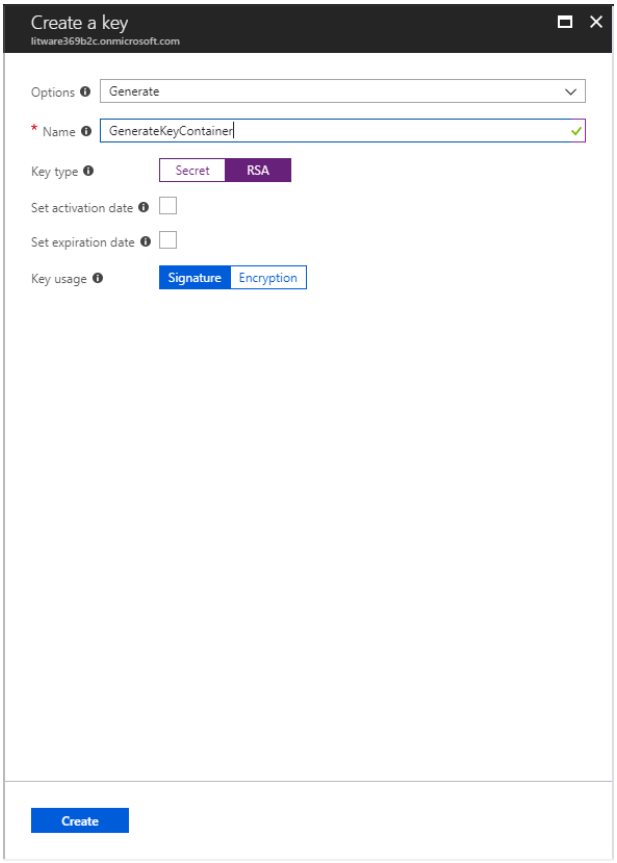
To generate a new key container in your B2C tenant of type RSA with the related RSA key, a.k.a. an asymmetric key, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Select **Add**.
    a. Select **Generate**.
    b. For **Name**, specify the name of the name of the container. For example, in our illustration "*GenerateKeyContainer*". The prefix "B2C_1A_" might be added automatically for "Advanced".
    c. For **Key type**, use **RSA**.
    d. For **Set activation date** and **Set expiration date**, use the defaults or specify the relevant dates if any in your context. These parameters are timestamps indicating when the key can become active and when it will expire.




    For our illustration, we will create the RSA key without any time limits.

    e. For **Key usage**, select the correct usage. For example, in our illustration, **Signature**.

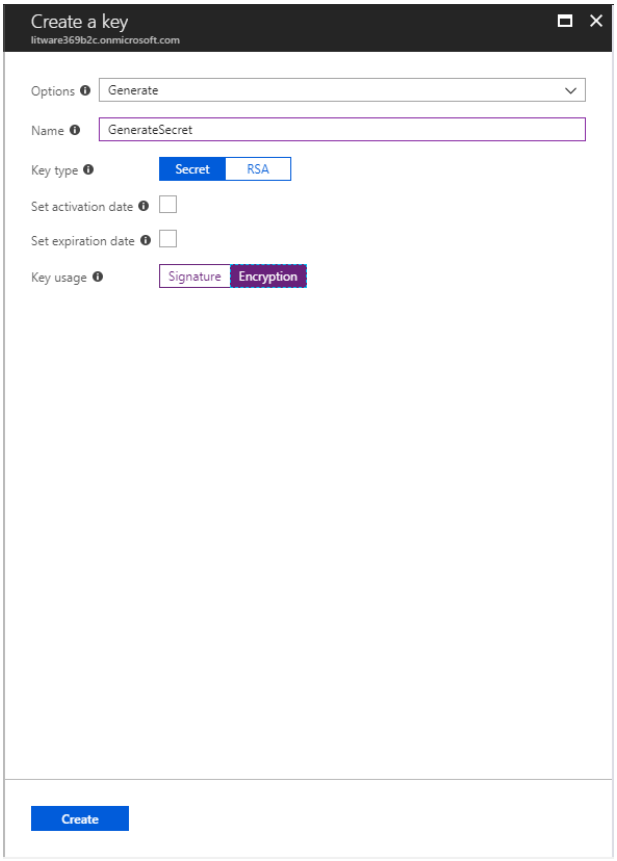



f. Select **Create**.

## Generating a binary secret

To generate a new key container in your B2C tenant of type oct with a binary secret, a.k.a. a symmetric key, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Select **Add**.
    a. Select **Generate**.
    b. In **Name**, specify the name of the name of the container. For example, in our illustration "*GenerateSecret*". The prefix "B2C_1A_" might be added automatically.
    c. For **Key type**, use **Secret**.
    d. For **Set activation date** and **Set expiration date**, use the defaults or specify the relevant dates. These parameters are timestamps indicating when the secret can become active and when it will expire. For our illustration, we will create the secret without any time limits.
    e. For **Key usage**, use **Encryption**.

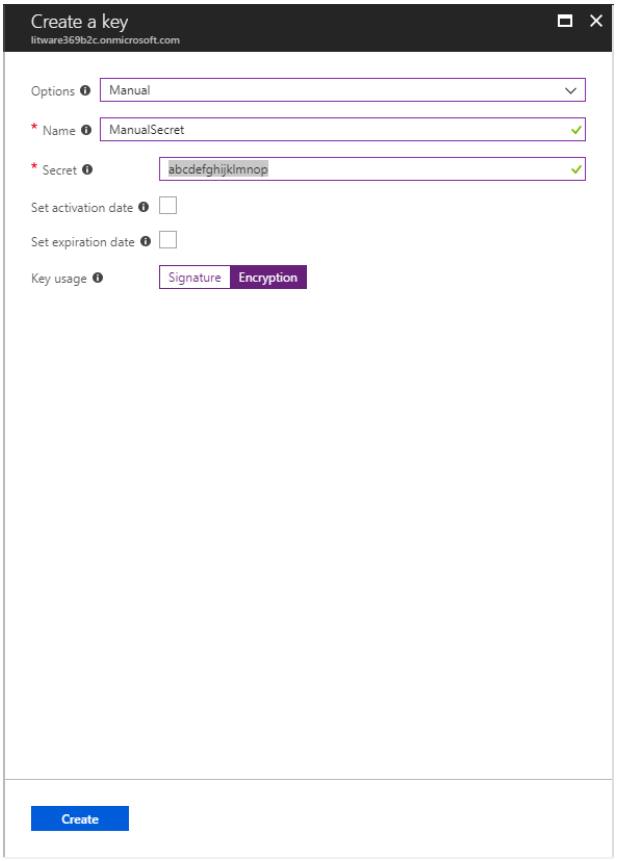



f. Select **Create**.

## Adding a binary secret

To manually add a new key container in your B2C tenant of type oct and specify the fixed key for the key container, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Select **Add**.
    a. Select **Manual**.
    b. In **Name**, specify the name of the name of the container. For example, in our illustration "*ManualSecret*". The prefix "B2C_1A_" might be added automatically.
    c. For **Key type**, use **Secret**.
    d. In **Secret**, specify the secret. For example, in our illustration, "abcdefghijklmnop".
    e. For **Set activation date** and **Set expiration date**, use the defaults or specify the relevant dates. These parameters are timestamps indicating when the secret can become active and when it will expire. For our illustration, we will create the secret without any time limits.
    f. For **Key usage**, use **Encryption**.




g. Select **Create**.

# Uploading a X509 certificate

As X509 certificate, you can upload the following certificate formats:

- A base64 encoded certificate (i.e. .cer file).

```
MIIKHQIBAzCCCdkGCSqGSIb3DQEHAaCCCcoEggnGMIIJwj……
```

- A PEM (Privacy Enhanced Mail) file (i.e. .pem, .key, .cer, or .cert file). This file format for certificate is governed by the RFC 7468 TEXTUAL ENCODINGS OF PKIX, PKCS, AND CMS STRUCTURES[16] and is used preferentially by open-source software.

```
-----BEGIN CERTIFICATE-----
MIIGmDCCBICgAwIBAgITWgAG4Q5s/s5qNTWX8AABAAbhDjANBgkqhkiG9w0B…….
-----END CERTIFICATE-----
```

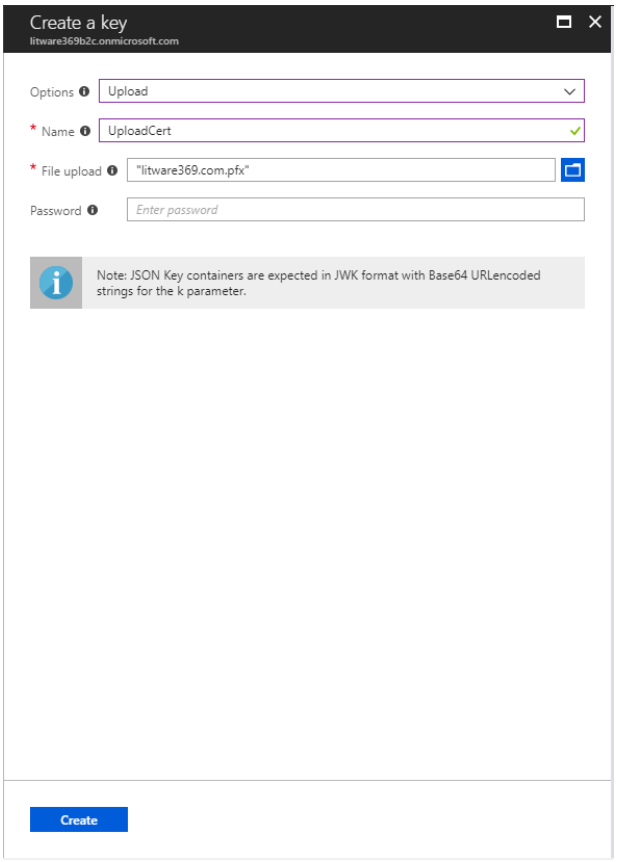
- A PKCS12 file (i.e. .pkcs12, .pfx, or .p12 file). This file format was originally defined by RSA in the Public-Key Cryptography Standards (abbreviated PKCS). The "12" variant was originally enhanced by Microsoft, and later submitted as RFC 7292 PKCS #12: PERSONAL INFORMATION EXCHANGE SYNTAX

[16] RFC 7468 TEXTUAL ENCODINGS OF PKIX, PKCS, AND CMS STRUCTURES: https://tools.ietf.org/html/rfc7468

v1.1[17]. This is a passworded container format that contains both public and private certificate pairs. Unlike PEM files, this container is fully encrypted.

To upload a certificate file - The file can optionally be protected by a password with a PKCS12 file -, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Select **Add**.
    a. Select **Upload**.
    b. In **Name**, specify the name of the name of the container. For example, in our illustration "*UploadCert*". The prefix "B2C_1A_" might be added automatically.
    c. In **File upload**, specifies the path of certificate file. For example, in our illustration the *litware369.com.pfx* file.
    d. In **Password**, specify the password if any that protects the file being specified.




    e. Select **Create**.

The expiration and not before dates will be the same as the uploaded certificate.

[17] RFC 7292 PKCS #12: PERSONAL INFORMATION EXCHANGE SYNTAX V1.1: PKCS #12: Personal Information Exchange Syntax v1.1

# Uploading a key

As a key, you can upload the following formats:

- A JSON key (i.e. .json file) as per aforementioned RFC 7517.

```
{
   "kty": "oct",
   "k": "UVVGQ1FrTkRSRVJGUlVaR01…",
   "use": "sig"
}
```

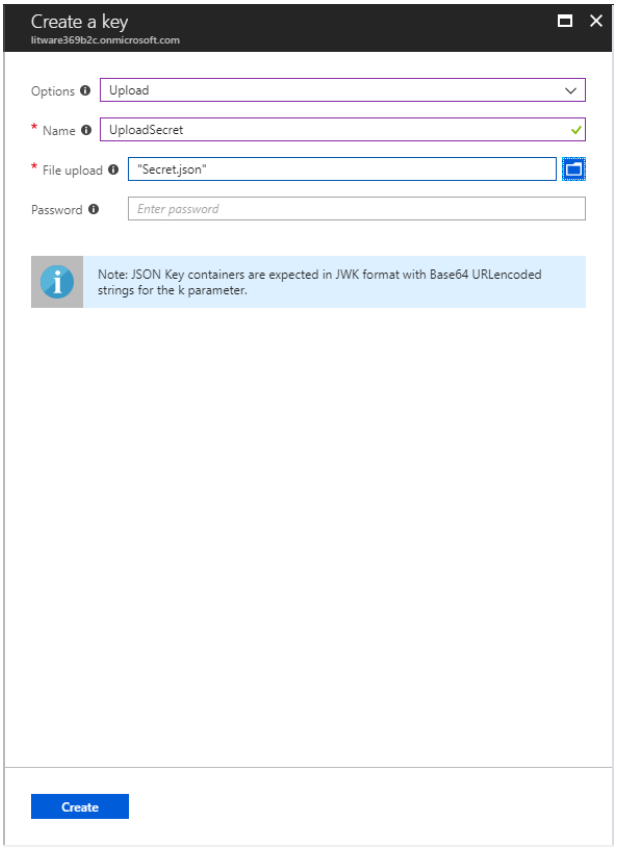
- A JSON key container (i.e. .json file) as per RFC 7517, which allows to control the full key container.

```
{
   "keys": [
   {       "kty": "oct",
           "k": "MTliZmZkMTJiN…..",
           "use": "sig"
   }  ]
}
```

As illustrated above, it is expected in JWK format with base64 Url encoded strings for the k parameter

To upload a key, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Select **Add**.
    a. Select **Upload**.
    b. In **Name**, specify the name of the name of the container. For example, in our illustration "*UploadSecret*". The prefix "B2C_1A_" might be added automatically.
    c. In **File upload**, specifies the path of key (container) file. For example, in our illustration the *Secret.json* file.

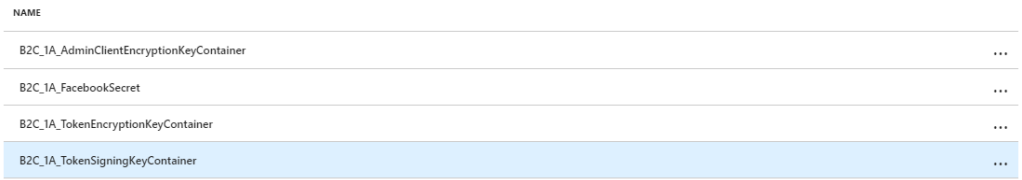



d. Select **Create**.

## Viewing all your keys

To list all the policy keys in your B2C tenant, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**. A list of all the keys in your B2C tenant is displayed.




The uploaded certificate and secrets (see previous sections) are all encrypted by means of the JSON Web Encryption (JWE) as per eponym RFC 7516[18]. This requires the use of a modern browser like Microsoft Edge or Google Chrome.

[18] RFC 7516 JSON Web Encryption (JWE): https://tools.ietf.org/html/rfc7516

The Encryption is done in JavaScript using the B2C_1A_AdminClientEncryptionKeyContainer with the following algorithms:

- AES-GCM (256 bits key).
- RSA-OAEP (2048 bits key).

## Deleting a key container

To delete a key container in your B2C tenant, proceed with the following steps:

1. Open the **Identity Experience Framework** blade in your Azure AD B2C tenant settings.
2. Select **Policy Keys**.
3. Right-click the key container to delete and select **Delete**.

## Backing up and restoring a key container

Every mutation to a key container results in a backup file which can be restored.

# Managing users and their attributes

Azure AD exposes directory administrative and programming surfaces for querying, adding, updating, and deleting objects in the directory, and thus, for users, sustaining the identities lifecycle management as a whole. Azure AD B2C doesn't make an exception to that.

The next sections consider the various options available to manage users in your B2C tenant.

## Using the Graph API to manage users

Microsoft provides two APIs for managing information stored in Azure AD:

1. The Azure AD Graph API which was designed to operate over tenants hosted by using Azure AD,
2. And the more recent Microsoft Graph API as already introduced (see section § *Introducing the Microsoft Graph API* earlier in this document).

As of this writing, new development of the Azure AD Graph API has stopped, and Microsoft is focusing on the Microsoft Graph API. The Microsoft Graph API indeed unifies many of the features available through the Azure Graph API with other services and Active Directory running on-premises. This unification has resulted in significant overlap between the two APIs.

In most cases, this means that while existing code that runs using the Azure AD Graph API will still be supported, you should consider using the Microsoft Graph API for new applications wherever possible. However, there are some situations where the functionality of the Azure AD Graph API is not yet available in the Microsoft Graph API.

**As of this now, one example especially concerns Azure AD B2C for user management. So, if you are working with B2C as in the context of this series of document, you should continue to use for user management the Azure AD Graph API, at least for the moment.**

> **Note** For more information, see article AZURE AD B2C: USE THE AZURE AD GRAPH API[19].

**Typical use cases of the Azure AD Graph API in the context of Azure AD B2C for user management are profile lookup, call center, and migration.**

The migration use case is specifically covered in the fourth document of this series.

### Introducing the Azure AD Graph API

An Azure AD tenant can contain a large number of objects (users, groups, profiles, permissions, and other items). While it might be feasible to perform some tasks manually, such as adding a new user, implementing an operation that spans a large set of objects in the directory (such as setting a permission for all users) is better performed programmatically. This approach can save time and reduce inconsistencies and errors that might otherwise occur. This is the role of the Azure AD Graph API.

[19] AZURE AD B2C: USE THE AZURE AD GRAPH API: https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-devquickstarts-graph-dotnet

The Azure AD Graph API is a service that provides a programmatic interface to Azure AD. With this API, you can perform for example the following common operations for a user object:

- Create a new local account in your B2C tenant.
- Get user's detailed properties, such as the name, the sign-in name, etc.
- Update user's properties, such as their location and phone number, or change the password.
- Disable the user account.

The above operations are performed on a tenant through the Azure AD Graph API by submitting HTTP requests to the Graph API endpoint at https://graph.windows.net. All requests are authenticated, and you must provide a valid access token in the request header.

In the context of this paper, the general form of a request is:

https://graph.windows.net/{*your_b2c_tenant*}/{*resource_path*}?{*api_version*}[*odata_query_parameters*]

Where

- {*your_b2c_tenant*} is the name of your Azure AD tenant,
- {*resource_path*} the resource that you are querying, modifying, adding, or deleting,
- {*api_version*} should typically be *api-version=1.6* for the version number,

If a request requires additional parameters, you specify them using OData operators, such as *$filter*, *$orderby*, *$expand*, *$top*, and *$format*.

For example, to retrieve a list of users whose names start with "B" in your B2C tenant, you can send this HTTP GET request:

https://graph.windows.net/myorganization/users?api-version=1.6&$filter=startswith(displayName,'B')

The result is an HTTP response message. The body contains a JSON array containing the matching details.

> **Note** You can use the aliases **myorganization** to refer to the tenant that you are currently signed in to, and **me** to refer to the currently signed-in user.

To create a new user, you would submit a POST request like this:

https://graph.windows.net/myorganization/users?api-version=1.6

The request body must contain the details of the new user, for example:

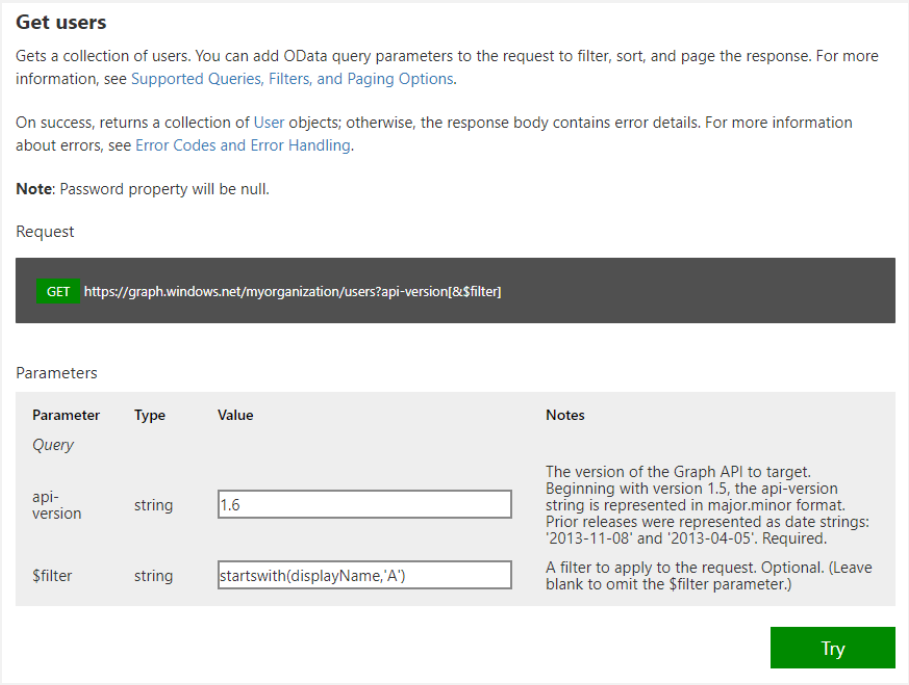
```
{
   "accountEnabled": true,
   "displayName": "John Doe",
   "mailNickname": "johndoe",
   "passwordProfile": {
      "password": "Test1234",
      "forceChangePasswordNextLogin": false
   },
   "userPrincipalName": "johndoe@litware369b2c.onmicrosoft.com"
}
```

The Azure AD Graph API provides access to a web service that enables you to manage tenants in Azure AD (B2C). You can interact with the Azure AD Graph API from any environment that enables you to compose and submit HTTP requests.

Consequently, you can invoke Azure AD Graph API requests directly from the Graph Explorer, the PowerShell command line, and of course from a custom application. The next sections will successively consider the three situations.

You can find a full list of the HTTP requests that you can end to the Azure AD Graph API in the article SUPPORTED QUERIES, FILTERS, AND PAGING OPTIONS | GRAPH API CONCEPTS[20].

**Note** The article AZURE AD GRAPH API REFERENCE[21] enables you to try out some of the available operations (currently only on user and group entities, and the me alias) and see the request and responses inline, right from within the online documentation.




**Note** The Graph API provides full text search capabilities in preview. Until now, searching Azure AD entities was limited to *$filter* searches and had a number of limitations developers found constraining. With this preview functionality, more powerful search mechanisms are added to the Graph API. For more details, see the blog post FULL TEXT SEARCH CAPABILITIES IN AZURE AD GRAPH API (PREVIEW)[22].

[20] SUPPORTED QUERIES, FILTERS, AND PAGING OPTIONS | GRAPH API CONCEPTS: https://msdn.microsoft.com/library/azure/ad/graph/howto/azure-ad-graph-api-supported-queries-filters-and-paging-options#CommonQueries

[21] AZURE AD GRAPH API REFERENCE: https://msdn.microsoft.com/en-us/library/azure/ad/graph/api/api-catalog

[22] FULL TEXT SEARCH CAPABILITIES IN AZURE AD GRAPH API (PREVIEW): http://blogs.msdn.com/b/aadgraphteam/archive/2015/04/10/full-text-search-capabilities-in-azure-ad-graph-api-preview.aspx

The following table roughly depicts the user object model:

| Property | Local account | Social account |
|---|---|---|
| *userPrincipalName* | Unused in Azure AD B2C | |
| *signInNames* | <type, value> collection:<br>• emailAddress, username<br>• phoneNumber – *coming soon*<br>• Other types – *possible* | N/A |
| *creationType* | LocalAccount | N/A |
| *passwordPolicies* | DisablePasswordExpiration, DisableStrongPassword | N/A |
| *AlternativeSecurityIds (hidden)* | | Collection (Account Linking - *future*):<br>5::facebook::<*facebook_uid*> |
| *StrongAuthenticationEmail (hidden)* | Verified email address. Used for password reset | Stored only if email address is changed & verified |
| *StrongAuthenticationPhone (hidden)* | Verified phone number (used for MFA) | |
| *OtherMails* | Communication email | Stored only if email address is unchanged |
| *Custom attributes (a.k.a. Extension properties)* | extension_facec33f7cb944e6823259ff65de4243_storeMembershipNumber | |

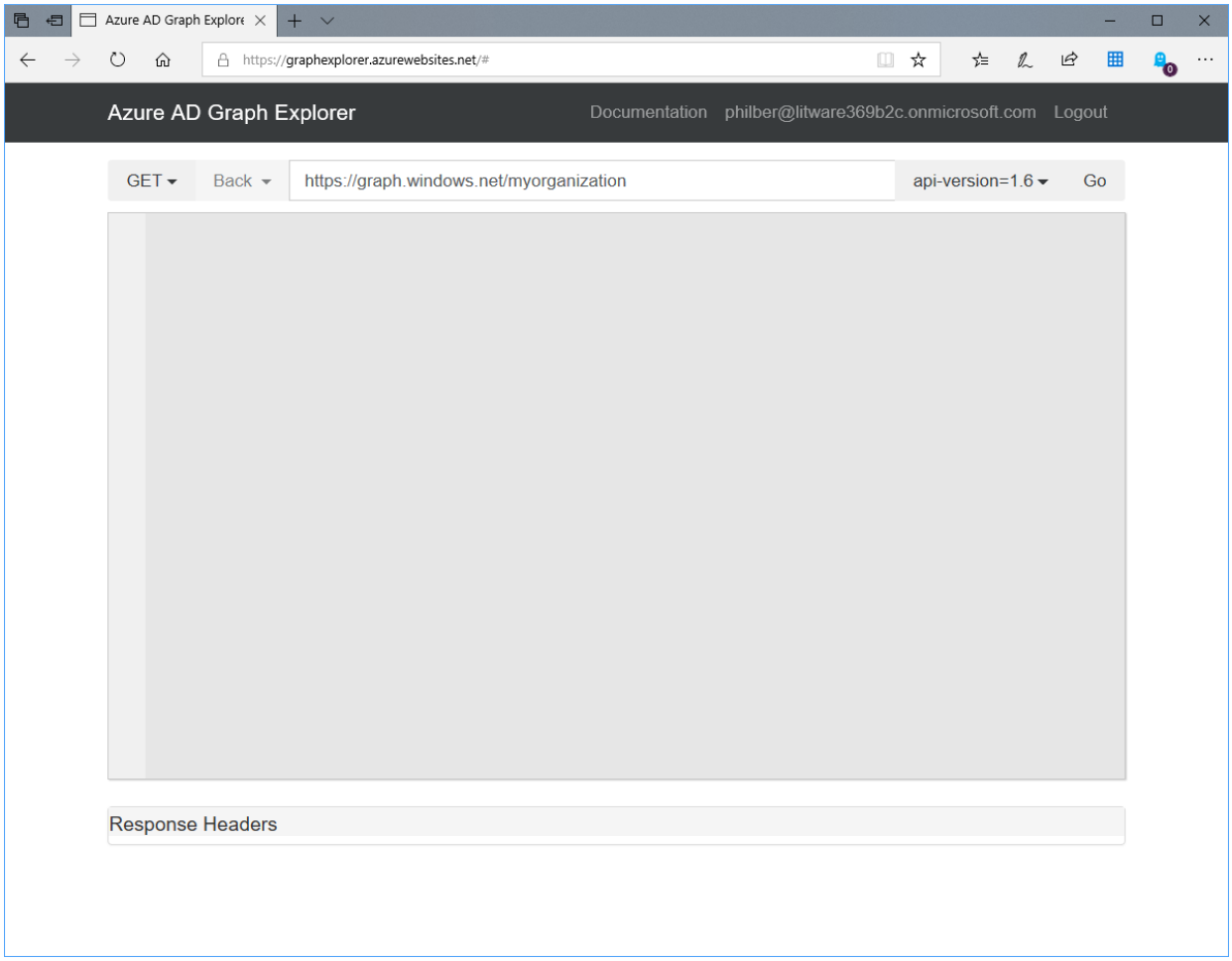
# Using the Azure AD Graph Explorer

The Azure AD Graph Explorer is a web application that provides a friendly user interface to the aforementioned Microsoft Graph API. It is available at https://graphexplorer.azurewebsites.net.

This web application basically sends requests to your B2C tenant and return results as an HTTP response. You provide your login credentials, and this information is included in the security header of each request.

To use the Azure AD Graph Explorer, perform the following steps:

1. Open a browsing session and navigate to https://graphexplorer.azurewebsites.net.
2. In the title bar, select **Login**.
3. Provide the credentials of an account with administrative privileges to your B2C tenant.

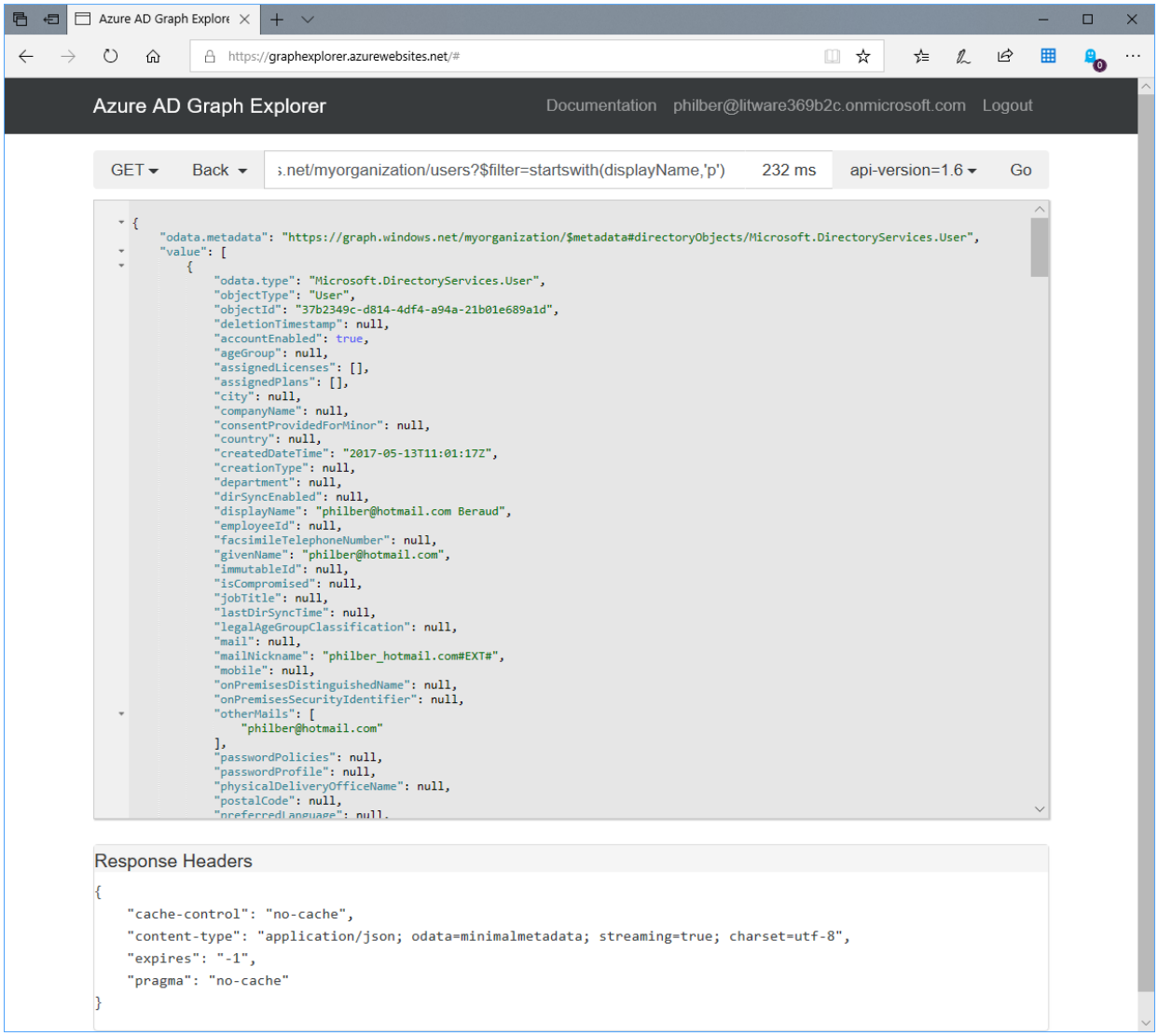



4. To run a query, select the **GET** verb, provide a URI that references the resources to find, such as:

   https://graph.windows.net/myorganization/users?$filter=startswith(displayName,'P')

   to find all users whose name starts with the letter "P", select the API version to use, and then select **Go**.

The results will appear in the main body of the window:




Note that you can expand the Response Headers pane to see the HTTP header information included with the response.

5. To perform an operation such as adding a new user, select the POST verb, specify the URI of the resource that you are adding, provide the details of the resource as the message body, and then select **Go**. For example, to create a new user, specify a URI such as:

   https://graph.windows.net/myorganization/users

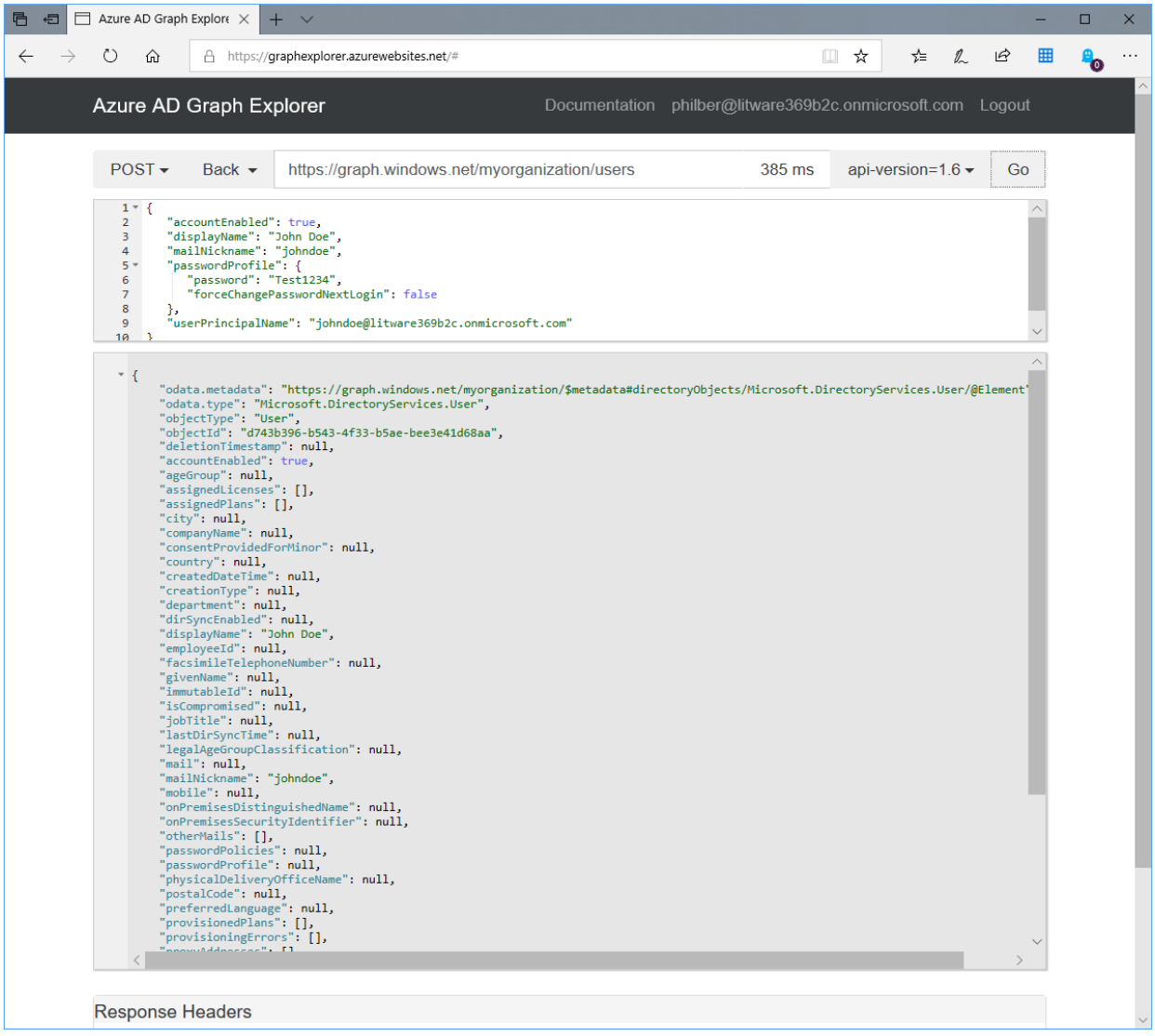
   and enter the information for the new user as the message body, in JSON format.

For example, to execute our previous illustration to create the John Doe user account:

```
{
   "accountEnabled": true,
   "displayName": "John Doe",
   "mailNickname": "johndoe",
   "passwordProfile": {
      "password": "Test1234",
      "forceChangePasswordNextLogin": false
   },
   "userPrincipalName": "johndoe@litware369b2c.onmicrosoft.com"
}
```

As already outlined, you can find the details of the properties that you can specify for the different types of objects that you can create in Azure AD (B2C) by using the Azure Graph API in the Azure AD Graph API Reference[23].

If the operation is successful, the response message shows the details of the object that was created:




[23] AZURE AD GRAPH API REFERENCE: https://msdn.microsoft.com/en-gb/library/azure/ad/graph/api/api-catalog

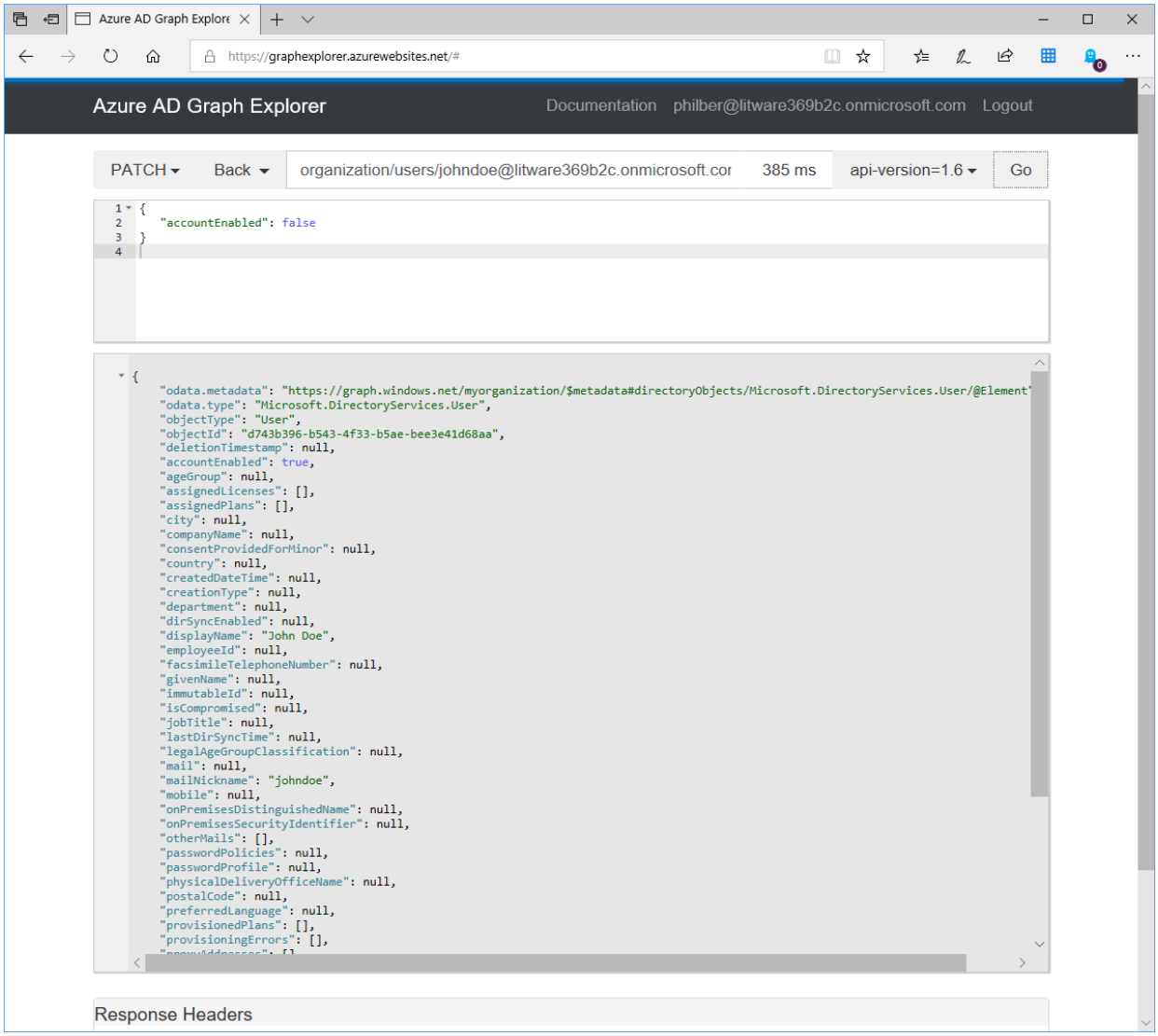
Note that you must be logged in as an account that has administrative rights in the domain to create, delete, and modify objects in your B2C tenant.

6. To modify an object, select the **PATCH** verb, provide the URI of the resource that you wish to modify, and specify the new details for the resource as the message body. The following example disables the account just created):

   https://graph.windows.net/myorganization/users/johndoe@litware369b2c.onmicrosoft.com

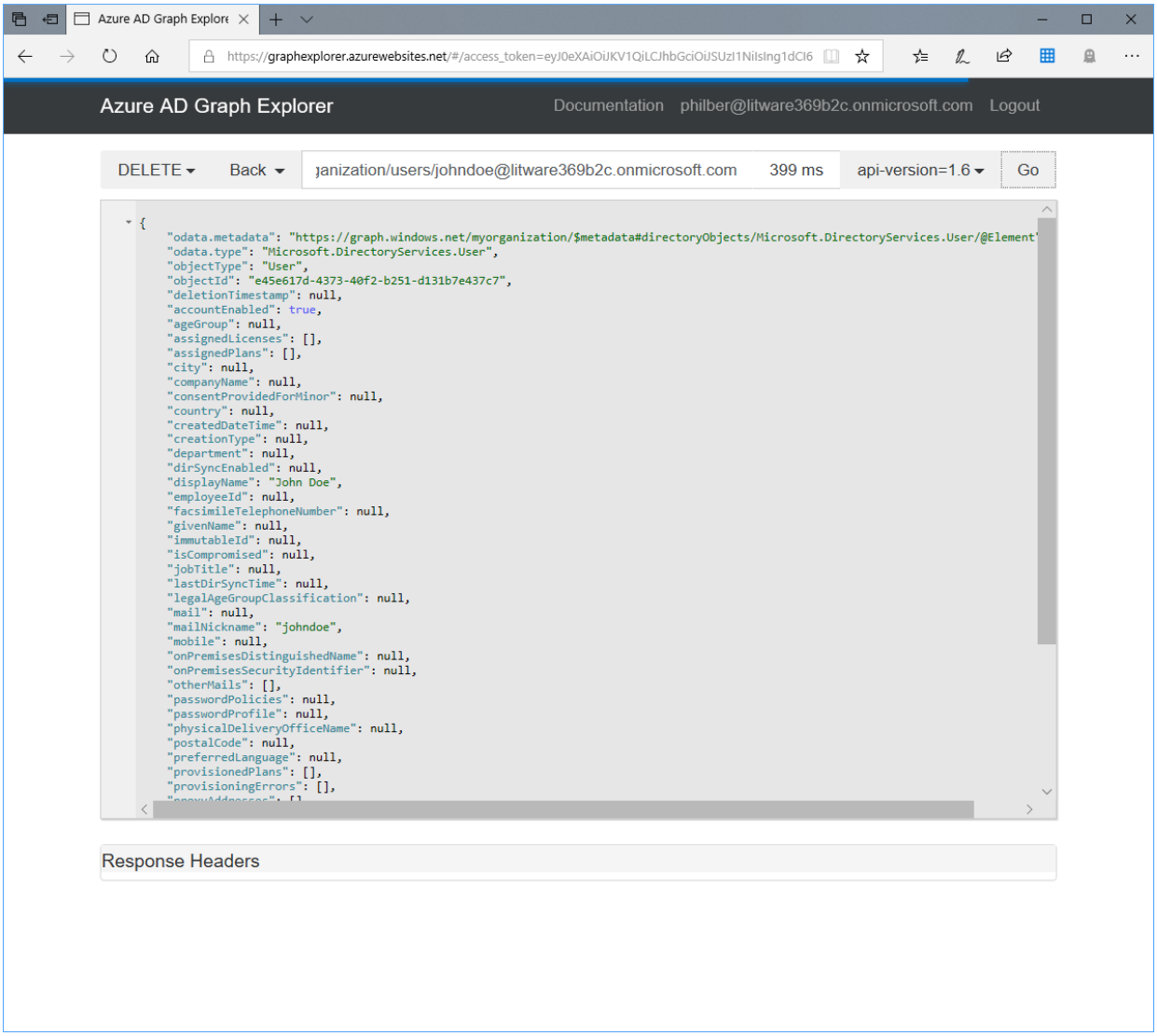
   by setting the **accountEnabled** attribute to **false**:

```
{
   "accountEnabled": false
}
```




7. To remove an object, select the **DELETE** verb, and provide the URI of the object:

   https://graph.windows.net/myorganization/users/johndoe@litware369b2c.onmicrosoft.com




# Using the Azure AD Graph API from PowerShell

You can perform operations using the Azure AD Graph API by submitting HTTP requests through a tool such as PowerShell. This approach is useful for performing ad-hoc operations against Azure AD.

The following steps illustrate this technique:

**Note** For more information, see blog post WORKING WITH AZURE ACTIVE DIRECTORY GRAPH API FROM POWERSHELL[24].

1. Install the .NET DLLs for ADAL v3 if not already done.
    a. Open a PowerShell command prompt with administrative privileges.

[24] WORKING WITH AZURE ACTIVE DIRECTORY GRAPH API FROM POWERSHELL:
https://blogs.technet.microsoft.com/paulomarques/2016/03/21/working-with-azure-active-directory-graph-api-from-powershell/

b. Run the following command:

```
PS> Install-Module -Name ADAL.PowerShell -RequiredVersion 3.19.4.1
```

It installs the *Microsoft.IdentityModel.Clients.ActiveDirectory.dll* file under the folder *C:\Program Files\WindowsPowerShell\Modules\ADAL.PowerShell\3.19.4.1* folder.

2. Now open a PowerShell command prompt or a PowerShell ISE environment.
3. Create the following PowerShell function:

```
function GetAuthToken
{   param
    (
          [Parameter(Mandatory=$true)]
          $TenantName
    )
    $adal =
"${env:ProgramFiles}\WindowsPowerShell\Modules\ADAL.PowerShell\3.19.4.1\Microsoft.IdentityModel.Clients.ActiveDirectory.d
ll"
    [System.Reflection.Assembly]::LoadFrom($adal) | Out-Null
    $clientId = "1950a258-227b-4e31-a9cf-717495945fc2"
    $redirectUri = "urn:ietf:wg:oauth:2.0:oob"
    $resourceAppIdURI = "https://graph.windows.net"
    $authority = "https://login.windows.net/$TenantName"
    $PlatformParameters = New-Object Microsoft.IdentityModel.Clients.ActiveDirectory.PlatformParameters(1)
    $tokenCache = New-Object Microsoft.IdentityModel.Clients.ActiveDirectory.TokenCache
    $authContext = New-Object "Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext" -ArgumentList
$authority, $tokenCache
    $authResult = $authContext.AcquireTokenAsync($resourceAppIdURI, $clientId, $redirectUri,
$PlatformParameters).GetAwaiter().GetResult()
    return $authResult
}
```

The **GetAuthToken** function prompts you to log in to the Azure AD (B2C) tenant specified by the **$TenantName** parameter and returns the authentication token if the login is successful.

4. Create a variable that references the name of your Azure AD tenant. Replace <your_B2C_tenant> with the name of your Azure AD tenant):

```
$tenant = "<your_B2C_tenant>.onmicrosoft.com"
```

5. Run the **GetAuthToken** function and extract the authentication token from the result:

```
$token = GetAuthToken -TenantName $tenant
```

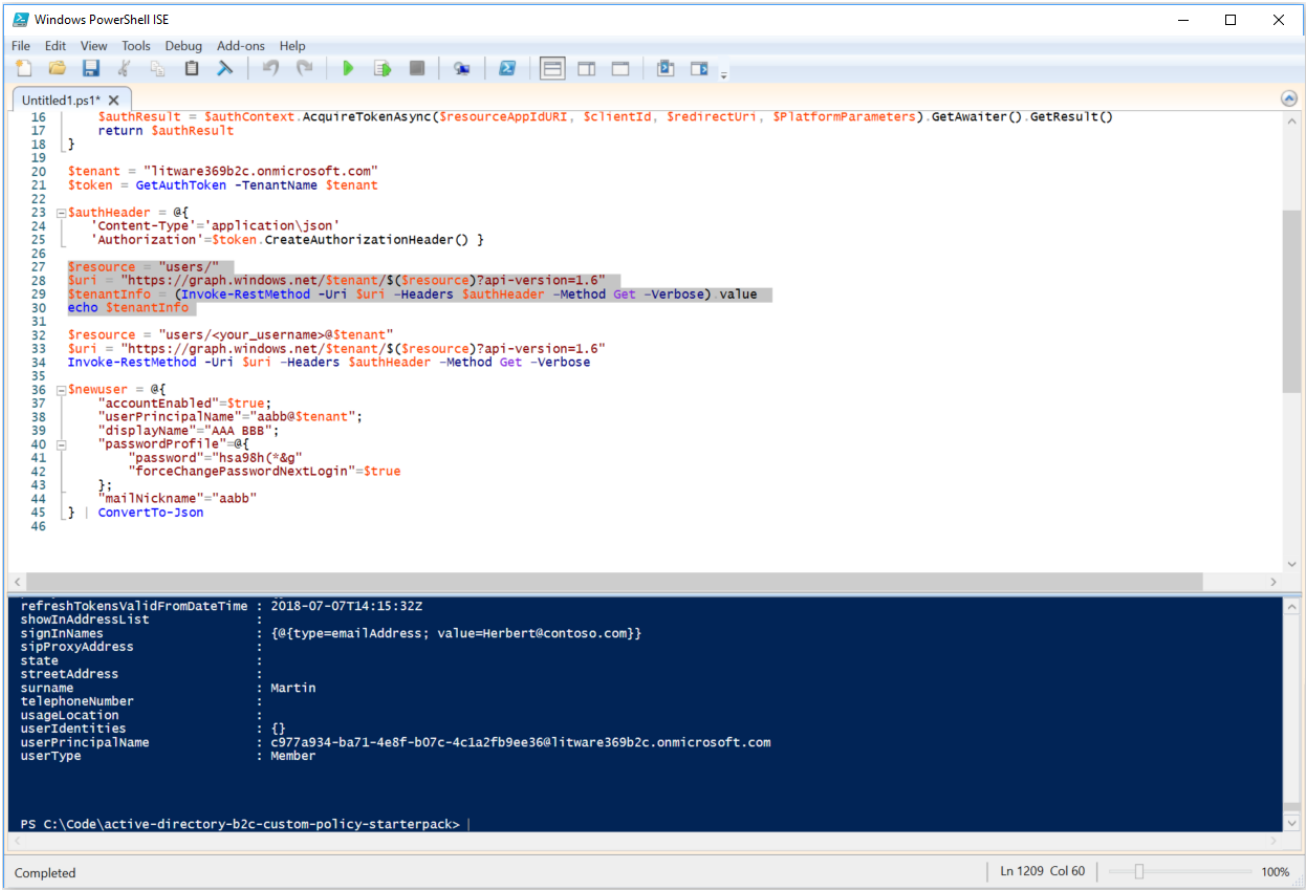
6. Construct an HTTP authorization header object that contains the bearer token in the **$token** variable:

```
$authHeader = @{
   'Content-Type'='application\json'
   'Authorization'=$token.CreateAuthorizationHeader() }
```

7. To retrieve the list of users from your tenant, run the following code:

```
$resource = "users/"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
$tenantInfo = (Invoke-RestMethod -Uri $uri –Headers $authHeader –Method Get –Verbose).value
echo $tenantInfo
```

This code constructs the URI of the **/users** resource in your tenant, and then uses the **InvokeRestMethod** function to send a GET request to this URI. The request includes an authentication header that contains the bearer token. The **echo** statement displays the results; these could be lengthy if your tenant contains many users.




8. To retrieve a single user, set the **$resource** variable as follows (replace <your_username> with the name of the user to fetch) and then run the query again:

```
$resource = "users/<your_username>@$tenant"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
Invoke-RestMethod -Uri $uri -Headers $authHeader -Method Get -Verbose
```

9. To create a new user, construct a JSON object that contains the attributes for the user:

```
$newuser = @{
    "accountEnabled"=$true;
    "userPrincipalName"="aabb@$tenant";
    "displayName"="AAA BBB";
    "passwordProfile"=@{
        "password"="hsa98h(*&g"
        "forceChangePasswordNextLogin"=$true
    };
    "mailNickname"="aabb"
} | ConvertTo-Json
```

10. Send an HTTP POST message to the **users** resource. Specify the JSON object as the message body:

```
$resource = "users/"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
Invoke-RestMethod -Uri $uri –Headers $authHeader –Method Post -Body
$newuser -ContentType "application/json" –Verbose
```

11. To modify a user, for example, to disable the user's account, create a JSON object that contains the details to be changed:

```
$changes = @{
    "accountEnabled"=$false
} | ConvertTo-Json
```

12. Send an HTTP PATCH message to the resource corresponding to the user account to modify. Specify the JSON object as the message body:

```
$resource = "users/aaa@$tenant"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
Invoke-RestMethod -Uri $uri –Headers $authHeader –Method Patch -Body
$changes -ContentType "application/json" –Verbose
```

13. To remove a user, send an HTTP DELETE request to the **users** resource that corresponds to that user:

```
$resource = "users/aaa@$tenant"
$uri = "https://graph.windows.net/$tenant/$($resource)?api-version=1.6"
Invoke-RestMethod -Uri $uri –Headers $authHeader –Method Delete – Verbose
```

# Using the Azure AD Graph API from a custom application

If you are performing many tasks manually, such as migrating a large number of users, the chances of making an error are high. A more considered approach is creating your own custom application that automates the tasks required.

For that purpose, you can use the Microsoft.IdentityModel.Clients.ActiveDirectory[25] NuGet package in your own custom applications to construct and send requests programmatically to the Azure AD Graph API endpoint.

For the sake of the illustration, we will use the sample application from B2CGraphClient sample application that shows how to use this package in a C# application. It can be downloaded at https://github.com/AzureADQuickStarts/B2C-GraphAPI-DotNet/archive/master.zip.

## Building the B2CGraphClient sample application

The section § *Building the B2CGraphClient code sample* in the Appendix Building the code samples depicts how to get and build this sample application. Please refer to it before moving to the next section.

[25] Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package:
https://www.nuget.org/packages/Microsoft.IdentityModel.Clients.ActiveDirectory/

## Using the B2CGraphClient sample application

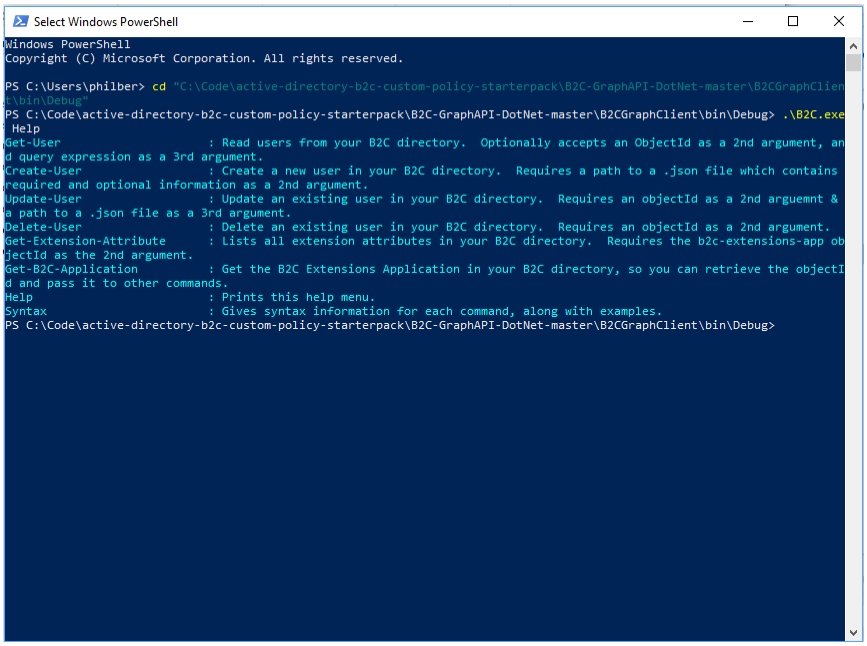
This section more especially shows how to perform local account CRUD operations with the B2CGraphClient sample command line application. It assumes that you successfully performed all the steps required to build and configure B2CGraphClient  sample application

To run the B2CGraphClient sample command line application, proceed with the following steps:

1. Open a PowerShell command prompt window and navigate to the *B2C-GraphAPI-DotNetmaster\B2CGraphClient\bin\Debug* folder in the **Starter-Pack** folder.
2. From the command line window, type the following command:

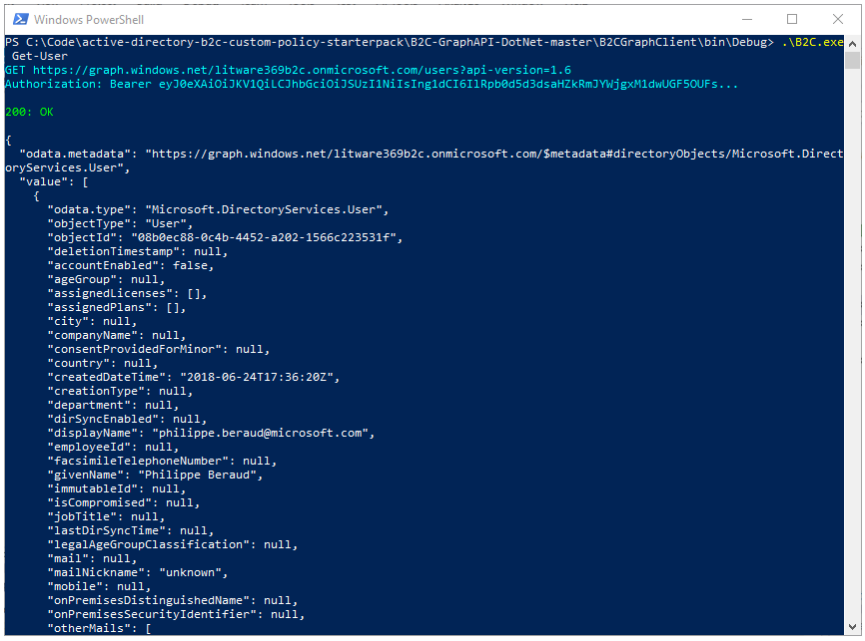
```
PS> .\B2C.exe Help
```

The various CRUD operations supported by the B2CGraphClient sample command line application are listed.




3. Type the following command to run the sample application and send an HTTP GET request to the **users** resource. You should see a list of users from your B2C tenant (in JSON format).

```
PS> .\B2C.exe Get-User
```



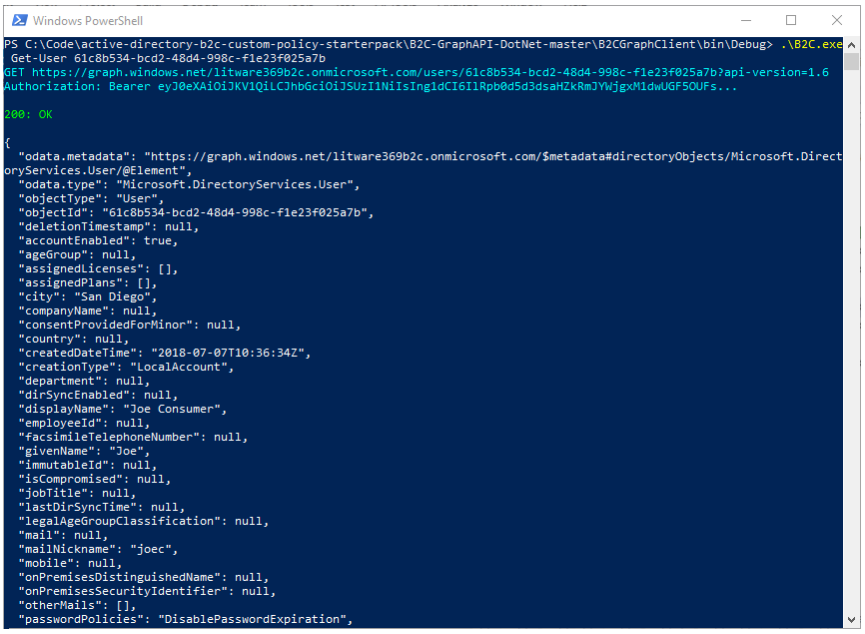

To create new username-based and email-based local accounts in your B2C tenant, proceed with the following steps:

1. Copy the file *usertemplate-email.json* located in the folder *B2C-graphAPI-DotNet-master* under the **Starter-Pack** folder in the same directory of the B2C.exe executable.
2. Copy the file *usertemplate-username.json* located in the folder *B2C-graphAPI-DotNet-master* under the **Starter-Pack** folder in the same directory of the B2C.exe executable.
3. Type the following commands from the above command prompt window:

```
PS> .\B2C.exe Create-User usertemplate-email.json
PS> .\B2C.exe Create-User usertemplate-username.json
```

4. Save the object IDs of the users you just created.
5. Run the previous **Get-User** command to see the new local accounts in your B2C tenant.

To search for specific users in your B2C tenant, type the following command (use the object ID of one of the local accounts created in the previous step 3:



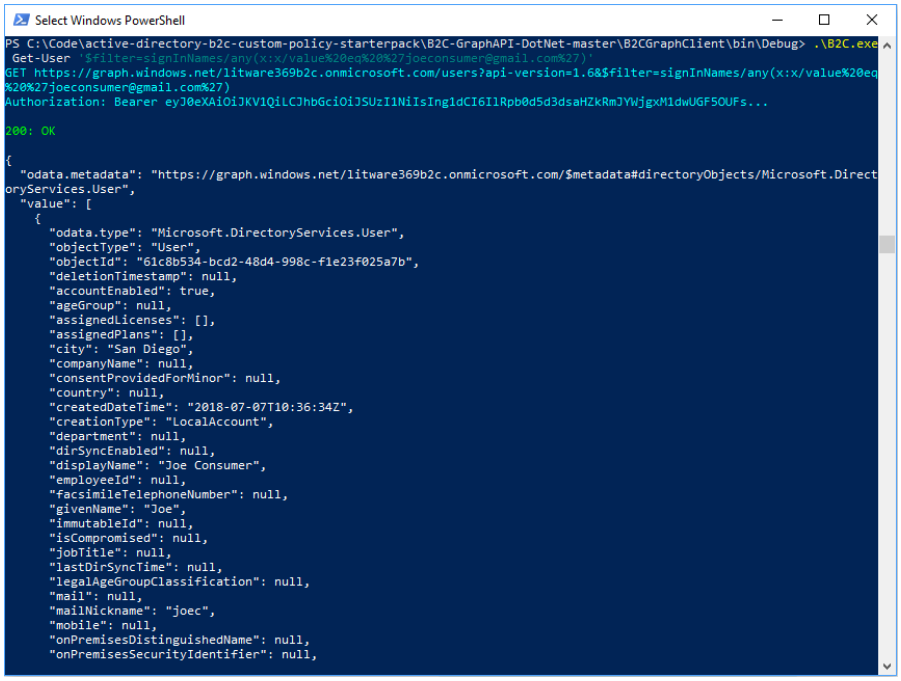

```
PS> .\B2C.exe Get-User 61c8b534-bcd2-48d4-998c-f1e23f025a7b
GET https://graph.windows.net/litware369b2c.onmicrosoft.com/users/61c8b534-bcd2-48d4-998c-f1e23f025a7b?api-version=1.6
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlRpb0d5d3dsaHZkRmJYWjgxM1dwUGF5OUFs...

200: OK
```


```
{
  "odata.metadata":
"https://graph.windows.net/litware369b2c.onmicrosoft.com/$metadata#directoryObjects/Microsoft.DirectoryServices
.User/@Element",
  "odata.type": "Microsoft.DirectoryServices.User",
  "objectType": "User",
  "objectId": "61c8b534-bcd2-48d4-998c-f1e23f025a7b",
  "deletionTimestamp": null,
  "accountEnabled": true,
  "ageGroup": null,
  "assignedLicenses": [],
  "assignedPlans": [],
  "city": "San Diego",
  "companyName": null,
  "consentProvidedForMinor": null,
  "country": null,
  "createdDateTime": "2018-07-07T10:36:34Z",
  "creationType": "LocalAccount",
  "department": null,
  "dirSyncEnabled": null,
  "displayName": "Joe Consumer",
  "employeeId": null,
  "facsimileTelephoneNumber": null,
  "givenName": "Joe",
  "immutableId": null,
  "isCompromised": null,
  "jobTitle": null,
  "lastDirSyncTime": null,
  "legalAgeGroupClassification": null,
  "mail": null,
  "mailNickname": "joec",
  "mobile": null,
  "onPremisesDistinguishedName": null,
  "onPremisesSecurityIdentifier": null,
  "otherMails": [],
  "passwordPolicies": "DisablePasswordExpiration",
  "passwordProfile": null,
  "physicalDeliveryOfficeName": null,
  "postalCode": "92130",
```

```
  "preferredLanguage": null,
  "provisionedPlans": [],
  "provisioningErrors": [],
  "proxyAddresses": [],
  "refreshTokensValidFromDateTime": "2018-07-07T10:36:33Z",
  "showInAddressList": null,
  "signInNames": [
    {
      "type": "emailAddress",
      "value": "joeconsumer@gmail.com"
    }
  ],
  "sipProxyAddress": null,
  "state": "California",
  "streetAddress": null,
  "surname": "Consumer",
  "telephoneNumber": null,
  "usageLocation": null,
  "userIdentities": [],
  "userPrincipalName": "b10e1833-7785-4ced-bc4f-eaa91d5ace51@litware369b2c.onmicrosoft.com",
  "userType": "Member"
}
PS> _
```

You can also leverage the filtering capabilities of the OData standard as stated in the introduction. Run the following command:



```
PS> .\B2C.exe Get-User '$filter=signInNames/any(x:x/value%20eq%20%27joeconsumer@gmail.com%27)'
GET https://graph.windows.net/litware369b2c.onmicrosoft.com/users?api-
version=1.6&$filter=signInNames/any(x:x/value%20eq%20%27joeconsumer@gmail.com%27)
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlRpb0d5d3dsaHZkRmJYWjgxM1dwUGF5OUFs...

200: OK

{
  "odata.metadata":
"https://graph.windows.net/litware369b2c.onmicrosoft.com/$metadata#directoryObjects/Microsoft.DirectoryServices.User",
  "value": [
    {
      "odata.type": "Microsoft.DirectoryServices.User",
      "objectType": "User",
      "objectId": "61c8b534-bcd2-48d4-998c-f1e23f025a7b",

      …

      "userPrincipalName": "b10e1833-7785-4ced-bc4f-eaa91d5ace51@litware369b2c.onmicrosoft.com",
      "userType": "Member"
    }
  ]
}
PS> _
```

**Note** The B2CGraphClient application also enables you to update and delete users, using the **UpdateUser** and **Delete-User** options. However, for these commands you must specify the object ID of the user to be modified or deleted. You can find this information by using the **Get-User** option.

# Using the Azure AD PowerShell cmdlets to manage users

The new V2 version of the Azure AD PowerShell cmdlets aim at providing a close alignment of the PowerShell functionality with the Graph API capabilities. Microsoft is moving towards a faster and more agile release process for new or updated functionality of these cmdlets.

As of this writing, these new PowerShell cmdlets are currently still in public preview. You will see regular new functionality updates to this preview release until the complete replacement is available.

**Note** For more information, see the article MICROSOFT AZURE ACTIVE DIRECTORY POWERSHELL MODULE VERSION RELEASE HISTORY[26].

To install this new version, proceed with the following steps:

1. Open a Windows PowerShell command prompt with administrative privileges.

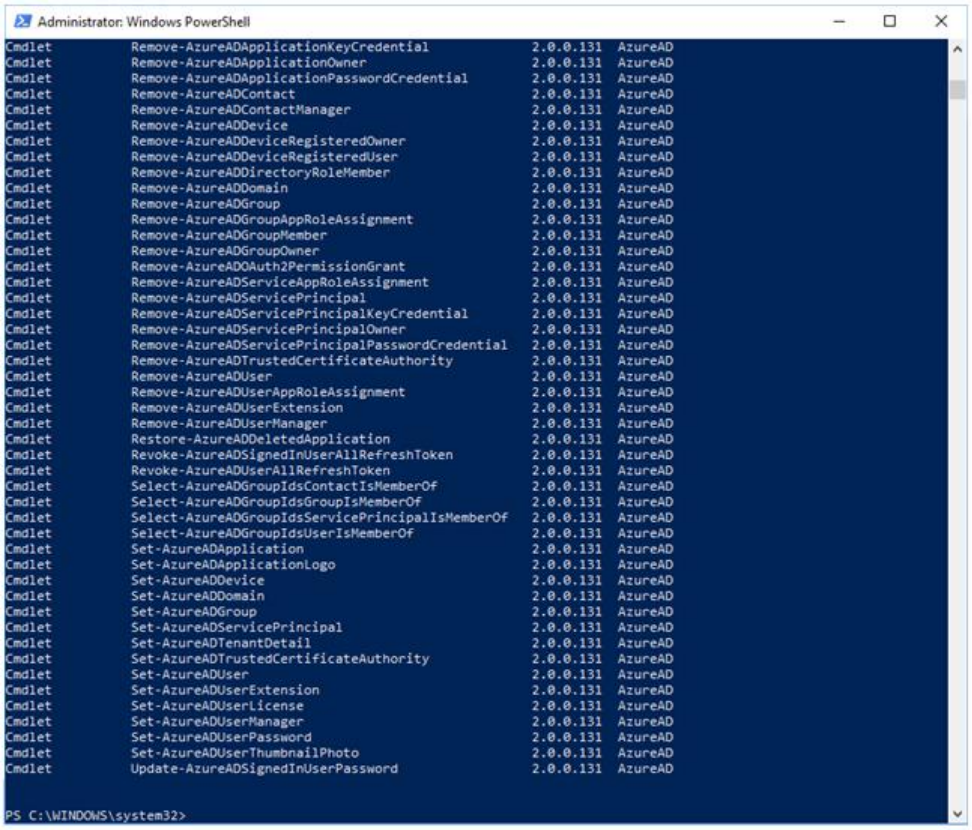
[26] MICROSOFT AZURE ACTIVE DIRECTORY POWERSHELL MODULE VERSION RELEASE HISTORY: http://social.technet.microsoft.com/wiki/contents/articles/28552.microsoft-azure-active-directory-powershell-module-version-release-history.aspx

2. Install the Public Preview release of Azure Active Directory V2 PowerShell Module from the PowerShell Gallery[27] by running the following command:

```
PS> Install-Module -Name AzureADPreview
```

3. Start taking advantage of the new cmdlets. For detail descriptions and examples of these cmdlets, type the following command to get all the cmdlets:

```
PS> Get-Command *AzureAD*
```




As one of the above cmdlets development's objectives consists in having a close alignment with the Graph API functionality, the names of objects and parameters are kept as close as possible to what is used in the Graph API.

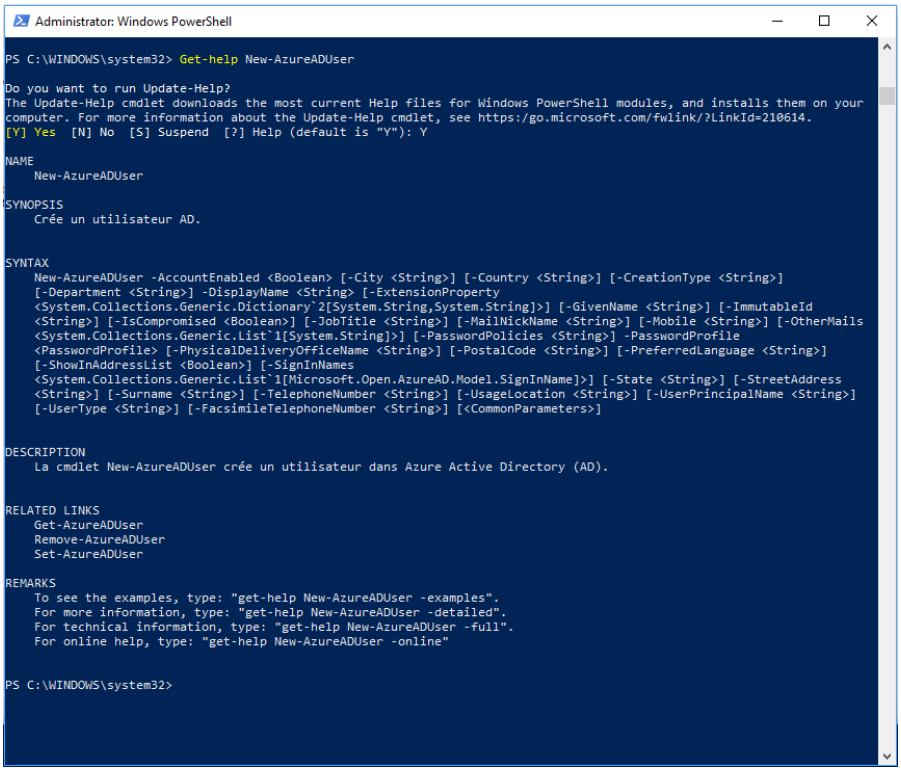
4. To see how these cmdlets work, type the following command to get the details of a specific cmdlet *<cmdlet name>*:

```
PS> Get-help <cmdlet name>
```

[27] Azure Active Directory PowerShell for Graph - Public Preview Release 2.0.1.18: https://www.powershellgallery.com/packages/AzureADPreview/2.0.1.18

For example:

```
PS> Get-help New-AzureADUser
```




**This concludes this third document of this series.**

# Appendix Building the code samples

## Building the B2CPolicyClient code sample

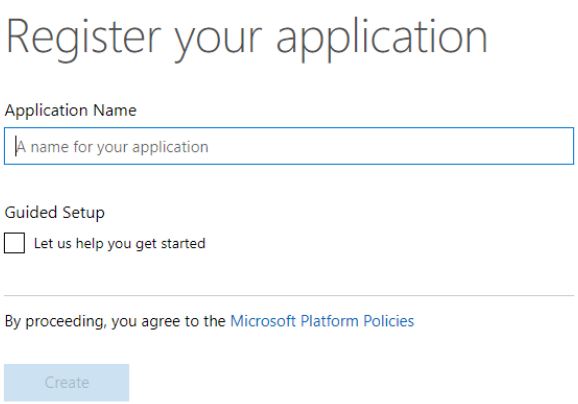
This section depicts how to build the B2CPolicyClient sample command line application to illustrate how to use the Graph API to call TrustFrameworkPolicy operations in your B2C test tenant in order to manage your custom policies.

This section depicts how to build the B2CPolicyClient sample command line application to illustrate how to use the Microsoft Graph API to call TrustFrameworkPolicy operations in your B2C test tenant in order to manage your custom policies.

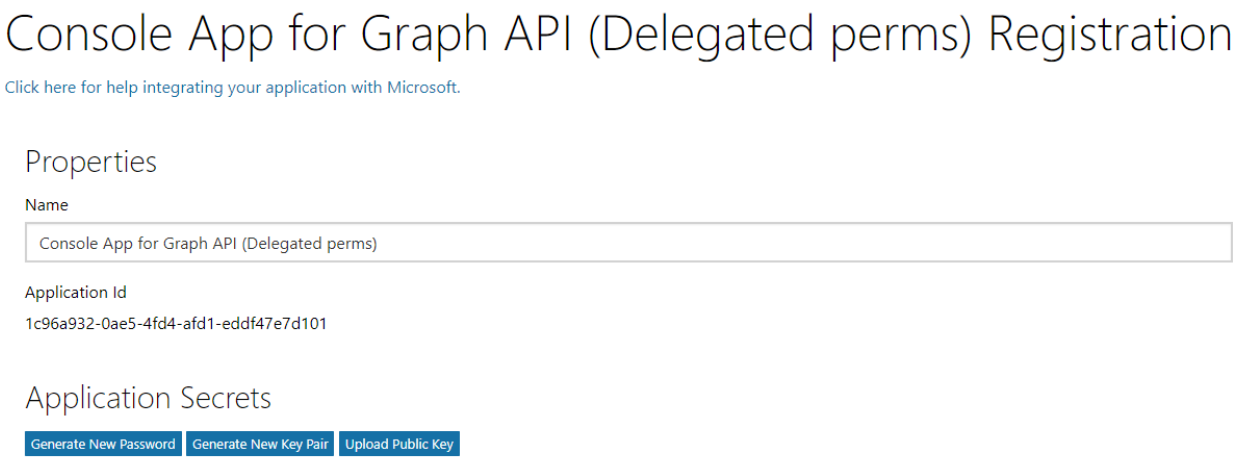
### Registering the delegated permissions application

Proceed with the following steps:

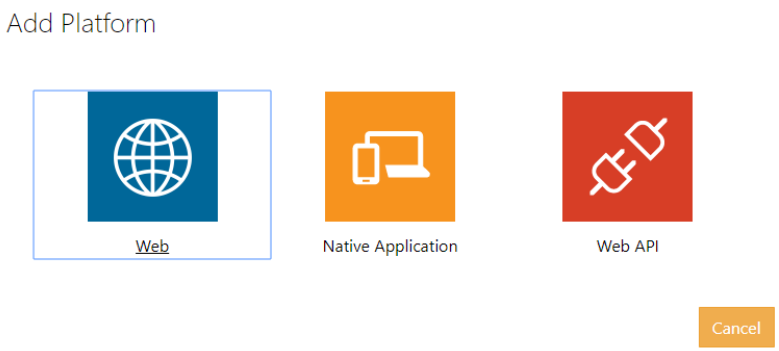
1. Open a browsing session and navigate to the Application Registration Portal at https://apps.dev.microsoft.com/.
2. Log in using your Microsoft account.
3. Select **Add an app**.



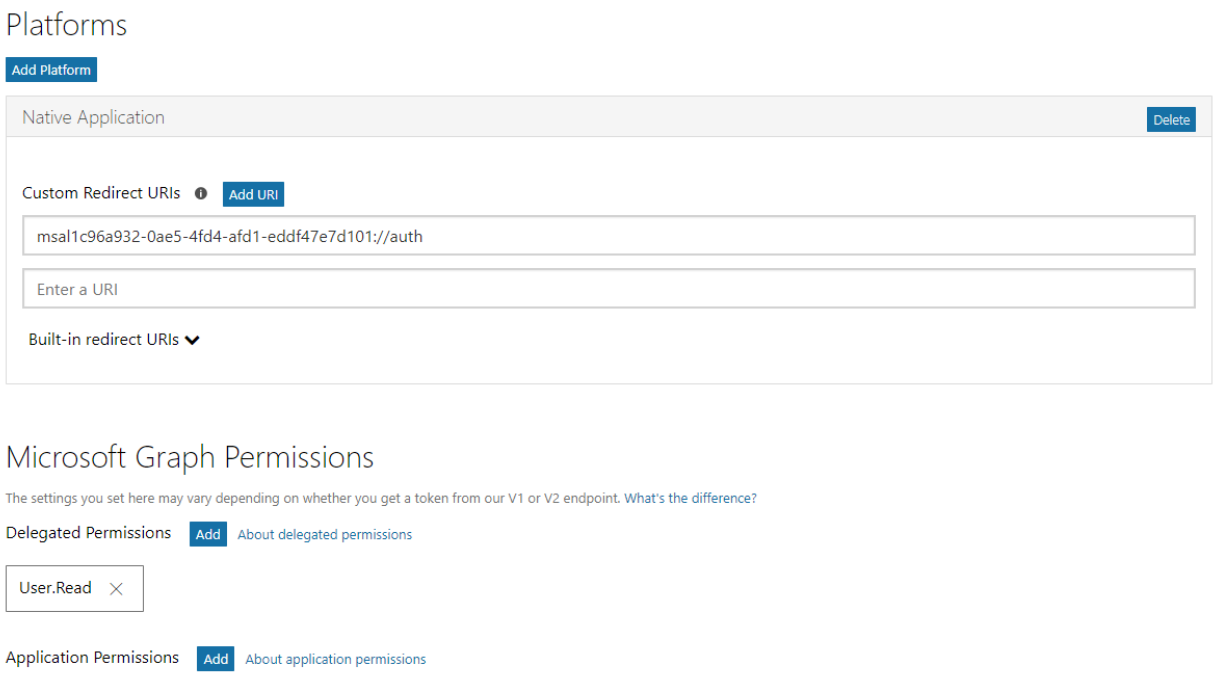

1. Enter a friendly name for the application, for example, in our illustration "*Console App for Graph API (Delegated perms)*".
2. Click **Create**.




4. On the Application Registration page, under **Properties**, make a note of the value of the **Application Id**. For example, in our configuration: 1c96a932-0ae5-4fd4-afd1-eddf47e7d101.
5. Under Click **Add Platform**.




6. Select the **Native App** tile.




Leave the Microsoft Graph permissions as they are. You indeed don't need to assign any permissions to the application on this page in so far as the delegated permissions operations in this code sample use permissions that are specified directly in code (see file *AuthenticationHelper.cs*).

7. Click **Save** at the bottom of the screen.

# Downloading the B2CPolicyClient code sample

Proceed with the following steps:

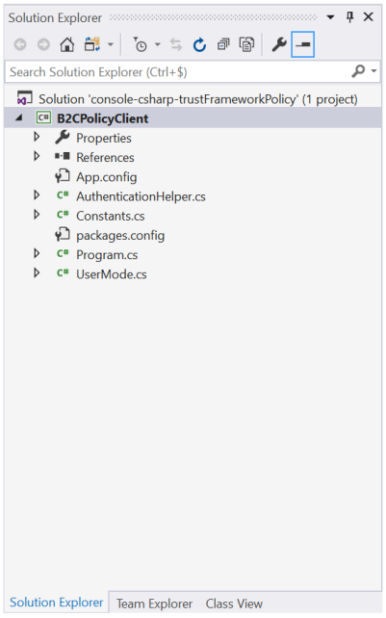
1. Download the archive file from GitHub at https://github.com/Azure-Samples/active-directory-b2c-graph-trustframework-policy/archive/master.zip.
2. Save the *active-directory-b2c-graph-trustframework-policy-master.zip* file on your local machine, and then extract the content of the *active-directory-b2c-graph-trustframework-policy-master.zip* file on your local disk, for example under the **Starter-Pack** folder. This will create an *active-directory-b2c-graph-trustframework-policy-master* folder underneath.

# Building and running the B2CPolicyClient code sample

Proceed with the following steps:

1. Navigate to the folder *active-directory-b2c-graph-trustframework-policy-master* folder and open the *console-csharp-trustFrameworkPolicy.sln* sample solution in Visual Studio 2017.

   If necessary, Visual Studio 2017 will download all the required NuGet package and resolve all the dependencies.




2. Open the *Constants.cs* file.

```
namespace console_csharp_trustframeworkpolicy
{
    internal class Constants
    {
        // TODO: update "ClientIdForUserAuthn" with your app guid and "Tenant" with your tenant name
        //       see README.md for instructions

        // Client ID is the application guid used uniquely identify itself to the v2.0 authentication endpoint
        public const string ClientIdForUserAuthn = "ENTER_YOUR_CLIENT_ID";
        // Your tenant Name, for example "myb2ctenant.onmicrosoft.com"
        public const string Tenant = "ENTER_YOUR_TENANT_NAME";

        // leave these as-is - URIs used for auth
        public const string AuthorityUri = "https://login.microsoftonline.com/" + Tenant + "/oauth2/v2.0/token";
        public const string RedirectUriForAppAuthn = "https://login.microsoftonline.com";

        …
    }
}
```

3. Replace ENTER_YOUR_CLIENT_ID with the above value of Application Id for this application. In our illustration, the value 1c96a932-0ae5-4fd4-afd1-eddf47e7d101.
4. Replace ENTER_YOUR_TENANT_NAME with the name of your B2C tenant. For example, in our configuration, litware369b2c.onmicrosoft.com.
5. On the **Build** menu, select **Rebuild Solution**. Verify that the solution compiles without any errors.

# Building the B2CGraphClient code sample

This section depicts how to build the B2CGraphClient sample command line application to illustrate how to use the Graph API to query, add, update, and delete users in your B2C tenant.

# Creating the service application in the B2C tenant

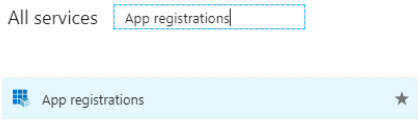
To communicate with the Azure AD Graph API, you first need to have a service account with administrative privileges. In Azure AD, you can do this by registering an application and authenticating to Azure AD. The application credentials are the application ID and the application secret. (The application acts as itself, not as a user, to call the Azure AD Graph API.)

To do so, you have to:

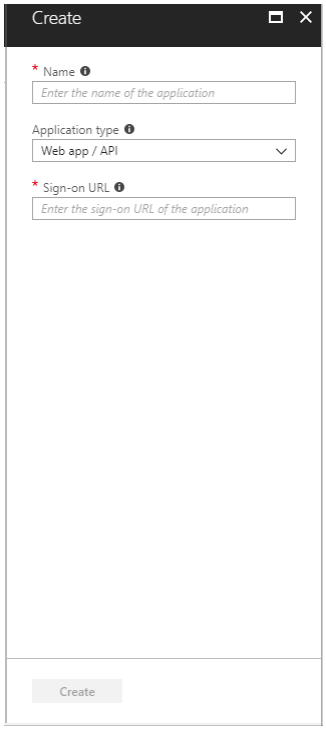
- Register your application in your B2C tenant.
- Copy the application ID.
- Create application secret.
- Set read and write permission to your application.

To create the service application, proceed with the following steps:

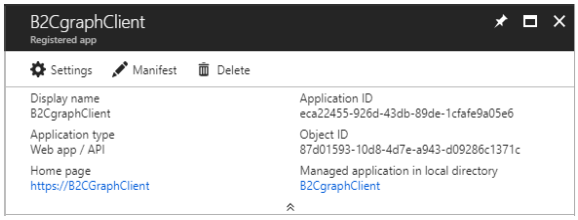
1. Log into the Azure portal as an account with administrative privileges.
2. Select **All services**, type **App registrations** in the search field, and then select **App registrations**.



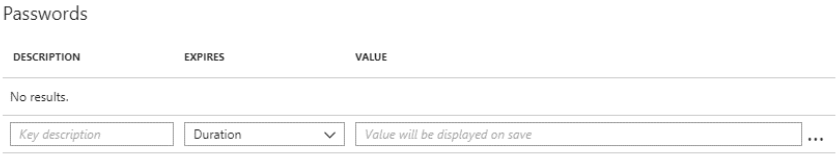

3. Select **+ New application registration**. A new blade opens.



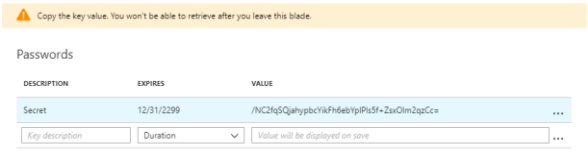

   a. In **Name**, enter "*B2CgraphClient*".
   b. Leave **Application type set** to **Web app/API**.
   c. in **Sign-on URL**, enter "*https://B2CGraphClient*" - this can actually be anything as it is just a placeholder -, and then select **Create**.



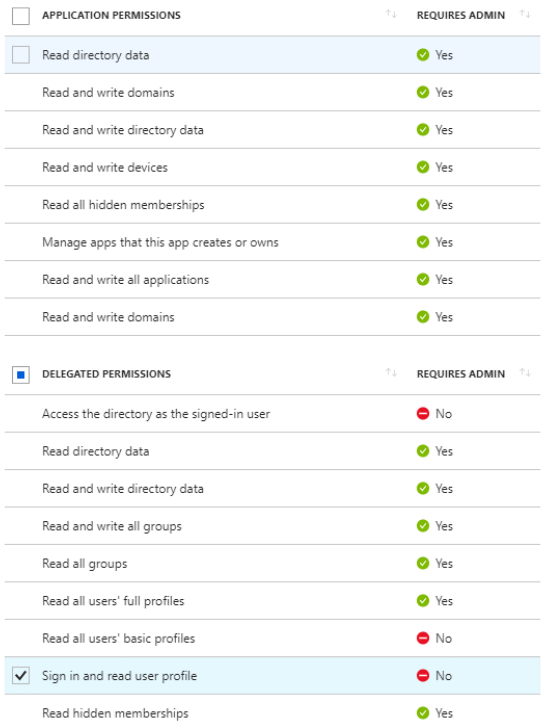

4. Make a note of the value of **Application ID**, here: eca22455-926d-43db-89de-1cfafe9a05e6.
5. Click **Settings**. A new blade opens.
6. Under **API ACCESS**, click **Keys**.



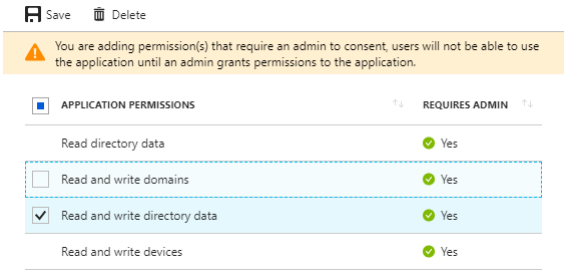

7. Create a new key named **Secret** that never expires, click **Save**.



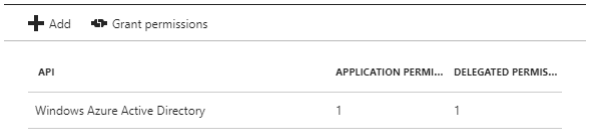

8. Make a note of the key value, here: /NC2fqSQjahypbcYikFh6ebYpIPls5f+ZsxOlm2qzCc=.
9. Under **API ACCESS**, now click **Required permissions**, and then select **Windows Azure Active Directory**.



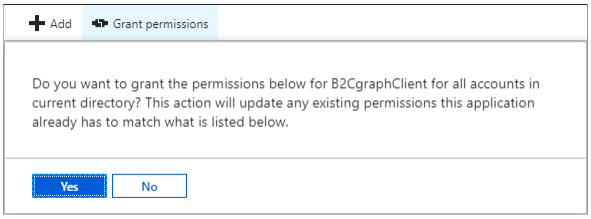

10. Select **Read and write directory data**




11. Click **Save**.




12. Select **Grant Permissions**, and then select **Yes** to confirm**.**




Please note that read and write directory data permission does NOT include the ability to delete users or change their password.

If you want to give your application the ability to delete users, or change password, you need to do extra steps that involve PowerShell to set User Account Administrator permissions as follows. This not required for the B2CGraphClient, so you can directly skip to the next section if you are not interested in these steps.

Following is a PowerShell code snippet to configure User Account Administrator permissions for your application:

```
$AppId = <your_application_ID>
```

```
# Fetch Azure AD application to assign role
$roleMember = Get-AzureADServicePrincipal -Filter "AppId eq '$AppId'"

# Fetch User Account Administrator role instance
$role = Get-AzureADDirectoryRole | WhereObject {$_.displayName eq 'User Account Administrator'}

# If role instance dos not exist, instantiate it based on the role template
If ($role -eq $null) {
   # Instantiate an instance of the role template
   $roleTemplate = Get-AzureADDirectoryRoleTemplate | WhereObject {$_.displayName eq 'User Account Administrator'}
   Enable-AzureADDirectoryRole -RoleTemplateId $roleTemplate.ObjectId

   # Fecth user Account Administrator role instance again
   $role = Get-AzureADDirectoryRole | WhereObject {$_.displayName eq 'User Account Administrator'}
}

# Add application to role
Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $roleMember.ObjectId

# Fecth role membership for role to confirm
Get-AzureADDirectoryRoleMember -ObjectId $role.ObjectId
```

The $AppID value must be changed to your application ID.

The above PowerShell code snippet requires to first connect to your B2C tenant with the Connect-AzureAD cmdlet. This cmdlet is part of the version 2 of the Azure AD PowerShell cmdlet.

## Downloading the B2CGraphClient code sample

Proceed with the following steps:

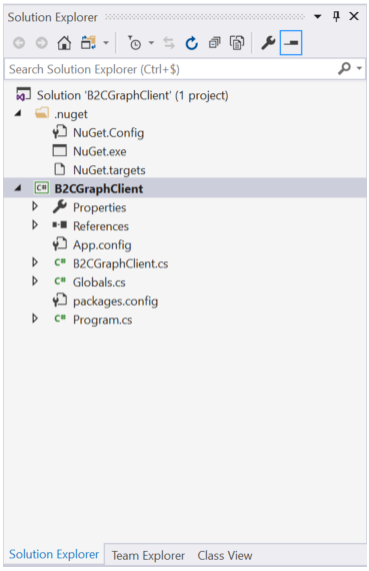
1. Download the B2CGraphClient code sample command line application as a .zip file from GitHub at https://github.com/AzureADQuickStarts/B2C-GraphAPI-DotNet/archive/master.zip.
2. Save the *B2C-GraphAPI-DotNet-master.zip* file on your local machine, and then extract the content of the *B2C-GraphAPI-DotNet-master.zip* file on your local disk, for example under the **Starter-Pack** folder. This will create a *B2C-GraphAPI-DotNet-master* folder underneath.

## Building and running the B2CGraphClient code sample

To build the B2CGraphClient code sample, proceed the following steps – explanation of the code principles is provided as well -:

1. Navigate to the folder *B2CGraphClient* within the above *B2C-graphAPI-DotNet-master* folder and open the *B2CGraphClient.sln* Visual Studio solution.

   If necessary, Visual Studio 2017 will download all the required NuGet package and resolve all the dependencies.




2. In Solution Explorer, in the **B2CGraphClient** project, open the *App.config* file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
   </startup>
   <appSettings>
```

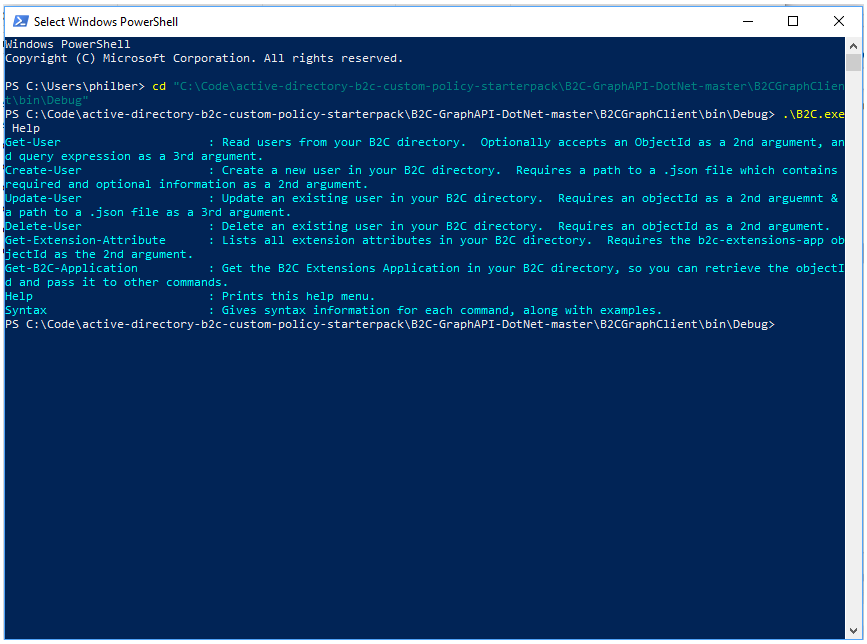
```
      <add key="b2c:Tenant" value="[Enter tenant name, e.g. contoso.onmicrosoft.com]" />
      <add key="b2c:ClientId" value="[Enter the client ID (a.k.a AppPrincipalId) as obtained from the Azure AD
Powershell, e.g. 82692da5-a86f-44c9-9d53-2f88d52b478b]" />
      <add key="b2c:ClientSecret" value="[Enter the client secret that you generated, e.g.
ONHJGaI232VenJIboyg8hmTlyNXh0Ef0brRRRNWBRfc=]" />
```

```
    </appSettings>
</configuration>
```

3. Replace the following app settings with your own values (generated earlier). For example, in our configuration:
    a. Tenant: litware369b2c.onmicrosoft.com
    b. ClientId: eca22455-926d-43db-89de-1cfafe9a05e6
    c. ClientSecret: /NC2fqSQjahypbcYikFh6ebYplPls5f+ZsxOlm2qzCc=
4. On the **Build** menu, select **Rebuild Solution**. Verify that the solution compiles without any errors
5. Open a PowerShell command line prompt window, type the following command:

```
PS> cd B2CGraphClient\bin\Debug
PS> .\B2C.exe Help
```

The various CRUD operations supported by the B2CGraphClient application are listed.




6. In Solution Explorer, select the file *B2CGraphClient.cs* to examine the source code for the application.
7. In the Code window, find the **B2CGraphClient** constructor. It looks like this:

```
public B2CGraphClient(string clientId, string clientSecret, string tenant)
{
    // The client_id, client_secret, and tenant are pulled in from the App.config file
    this.clientId = clientId;
    this.clientSecret = clientSecret;
    this.tenant = tenant;

    // The AuthenticationContext is ADAL's primary class, in which you indicate the direcotry to use.
    this.authContext = new AuthenticationContext("https://login.microsoftonline.com/" + tenant);

    // The ClientCredential is where you pass in your client_id and client_secret, which are
    // provided to Azure AD in order to receive an access_token using the app's identity.
    this.credential = new ClientCredential(clientId, clientSecret);
}
```

This code runs when the application starts up. It uses the values in the file *App.config* to connect to your B2C tenant and saves the authentication token that is returned. It is analogous to the **GetAuthToken** function used in the PowerShell example (see section § *Using the Azure AD Graph API from PowerShell*).

4. Scroll to the end of the file and find the **SendGraphRequest** function:

```
public async Task<string> SendGraphGetRequest(string api, string query) {
    // First, use ADAL to acquire a token using the app's identity (the credential)
    // The first parameter is the resource we want an access_token for; in this case, the Graph API.
    AuthenticationResult result = authContext.AcquireToken("https://graph.windows.net", credential);

    // For B2C user managment, be sure to use the 1.6 Graph API version.
    HttpClient http = new HttpClient();
    string url = "https://graph.windows.net/" + tenant + api + "?" + Globals.aadGraphVersion;
    if (!string.IsNullOrEmpty(query))
    {
       url += "&" + query;
    }

    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("GET " + url);
    Console.WriteLine("Authorization: Bearer " + result.AccessToken.Substring(0, 80) + "...");
    Console.WriteLine("");

    // Append the access token for the Graph API to the Authorization header of the request, using the Bearer scheme.
    HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Get, url);
    request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", result.AccessToken);
    HttpResponseMessage response = await http.SendAsync(request);
    if (!response.IsSuccessStatusCode)
    {
       string error = await response.Content.ReadAsStringAsync();
       object formatted = JsonConvert.DeserializeObject(error);
       throw new WebException("Error Calling the Graph API: \n" + JsonConvert.SerializeObject(formatted,
Formatting.Indented));
    }

    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine((int)response.StatusCode + ": " + response.ReasonPhrase);
    Console.WriteLine("");
    return await response.Content.ReadAsStringAsync();
}
```

The application uses this function to send HTTP GET requests to the Azure AD Graph API. The **api** parameter to this function contains the resource to query (such as **users** or **groups**), and the **query** parameter contains any OData options, such as **$filter**, **$top**, and so on. The function constructs a URI referencing the data to fetch, adds the bearer token to the request, and then submits it to the Azure AD Graph API endpoint. It captures the result and returns it as a string.

5. Find the **SendPostRequest** method:

```
private async Task<string> SendGraphPostRequest(string api, string json)
{
    // NOTE: This client uses ADAL v2, not ADAL v4
    AuthenticationResult result =
    authContext.AcquireToken(Globals.aadGraphResourceId, credential);
    HttpClient http = new HttpClient();
    string url = Globals.aadGraphEndpoint + tenant + api + "?" + Globals.aadGraphVersion;

    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("POST " + url);
    Console.WriteLine("Authorization: Bearer " + result.AccessToken.Substring(0, 80) + "...");
    Console.WriteLine("Content-Type: application/json");
    Console.WriteLine("");
    Console.WriteLine(json);
    Console.WriteLine("");

    HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post, url);
    request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", result.AccessToken);
    request.Content = new StringContent(json, Encoding.UTF8, "application/json");
```

```
    HttpResponseMessage response = await http.SendAsync(request);
    if (!response.IsSuccessStatusCode)
    {
       string error = await response.Content.ReadAsStringAsync();
       object formatted = JsonConvert.DeserializeObject(error);
       throw new WebException("Error Calling the Graph API: \n" +
       JsonConvert.SerializeObject(formatted, Formatting.Indented));
    }

    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine((int)response.StatusCode + ": " + response.ReasonPhrase);
    Console.WriteLine("");

    return await response.Content.ReadAsStringAsync();
}
```

This method sends HTTP POST requests to create new resources. The parameters are the URI identifying the type of resource to create, and a string containing the JSON data that represents the object.

6. Find and examine the **SendGraphPatchRequest** and **SendGraphDeleteRequest** methods. These methods follow a similar pattern, constructing HTTP PATCH and HTTP DELETE requests to modify and remove objects.