E-book Series

Microsoft

A guided tour of Microsoft Entra Verified ID

Part 3 - A developer walkthrough to illustrate how to support verifiable credentials with your Azure Active Directory (Azure AD) B2C policies

aka.ms/verifyonce

This page is intentionally left blank.

Notice

Microsoft Entra Verified ID, formerly known as Azure Active Directory Verifiable Credentials, is part of the Microsoft Entra family of products. Learn more about the Microsoft Entra family of identity solutions.

Microsoft Entra Verified ID is included with any Azure Active Directory (Azure AD) subscription, including Azure AD Free, and is now generally available. Read the announcement.

The end-to-end illustrated walkthrough provided in the following pages is extrapolated from the documentation Microsoft Entra Verified ID documentation available on docs.microsoft.com. **Please refer to this documentation for the latest information that pertains to Microsoft Entra Verified ID**.

Publication: September 2022
Version 1.0a

About this guide

This third part of this guide is about illustrating how easy it is to integrate Microsoft Entra Verified ID with Azure AD B2C and the custom policies for your user journeys.

As introduced in the first part of this guide, Microsoft Entra Verified ID is a managed service for Verifiable Credentials (VCs). It provides organizations with a scalable, standards-based way to issue and verify digital credentials with end users or other organizations as part of an identity verification process, such as onboarding, secure access to apps, or account recovery. With Microsoft Entra Verified ID, organizations can easily and more securely verify workplace credentials, education status, certifications, or other unique identity attributes in seconds. This can be either Verified employee credentials as covered in the first part of this guide or Custom credentials as illustrated in the second part of the guide.

Plus, it respects the privacy of end users, who can own and control their own identity. Each of us needs a digital identity we own, one which securely and privately stores all elements of our digital identity. Microsoft Entra Verified ID makes portable, self-owned identity possible, and represents our commitment to an open, trustworthy, interoperable, and standards-based decentralized identity future for individuals and organizations. It allows individuals, organizations, and devices to decide what they share, when they share it, with whom they share it, and - when necessary - take it back.

Our focus is to make it easy for any company to use it every day, and to make it a great experience for the developer community to deliver innovative custom business solutions for endless numbers of use cases with our developer kit, APIs, and documentation, and customer-facing solutions are also paramount amongst them.

So, one question might arise:

How Microsoft Entra Verified ID intersect with Microsoft customer identity & access management (CIAM) solutions to define the various user journeys as part of these use cases? And in particular Azure Active Directory B2C (Azure AD B2C).

You said Azure AD B2C, Quès aco!?

See What is Azure Active Directory B2C? and Technical and feature overview to learn more.

Azure B2C provides business-to-customer (B2C) identity as a service (IDaaS). Your customers use their preferred social, enterprise, or local account identities to get single sign-on (SSO) access and single logout (SLO) capabilities to your web applications, mobile applications, and REST APIs, as well as for authorization such as access to REST API resources by authenticated users.

As such, Azure AD B2C is a white label, custom branded CIAM solution capable of supporting millions of users and billions of authentications per day. It takes care of the scaling and safety of the authentication platform, the capture of detailed analytics about sign-in behavior and sign-up conversion, the centralization/orchestration of the collection of user profile & preference information, the monitoring, and the automatically handling threats, such as denial-of-service (DoS), password spray, or brute force attacks.

Azure AD B2C uses standards-based authentication protocols notably including OpenID Connect (OIDC) by more than 95% of applications on the Internet, and also leveraged as part of the Microsoft Entra Verified ID as outlined in the previous guide. This allows Azure AD B2C to integrate with most modern applications and commercial off-the-shelf software.

Defining user journeys

See User flows and custom policies in Azure Active Directory B2C.

In Azure AD B2C, you can model your user journeys and define the business logic behind that users follow to gain access to your applications and REST APIs. For example, you can determine the sequence of steps users follow when they sign up, sign in, edit a profile and/or preference information, reset a password, etc. After completing the sequence, the user acquires a security token and gains access to your application.

These identity user experiences can be provided in two ways, i.e., with:

1. **User flows** that are predefined, built-in, configurable policies for the most common identity tasks, and thus enable to create sign-up, sign-in, and policy editing experiences in minutes.
2. **Custom policies** that enable you to create your own specific user journeys for (more) complex identity experience scenarios with the full spectrum of capabilities provided by the so-called **identity experience framework (IEF) orchestration platform** of Azure AD B2C.

The rest of this guide focuses on custom policies and how to leverage them in accordance with verifiable credentials (VCs).

So let's first consider what they are.

A first look at custom policies

See Azure Active Directory B2C custom policy overview.

Custom policies in Azure AD B2C are configuration files that define the behavior of your Azure AD B2C tenant user experience. Custom policies can be fully edited by an identity developer to complete many different crafted tasks.

As such, a custom policy is fully configurable and policy-driven. It orchestrates trust between entities in standard protocols such as the above-mentioned OpenID Connect (OIDC) protocol in the context of this guide, and a few non-standard ones, for example REST API-based system-to-system claims exchanges. The complete framework creates user-friendly, white-labeled experiences.

User journeys

Custom policies give you the ability to construct user journeys: each user journey is defined by a policy. You can build as many or as few policies as you need to enable the best user experience for your organization.

User journeys allows you to define the business logic with path through which user will follow to gain access to your applications or REST APIs, a.k.a. relying parties (RPs). The user is taken through the user journey to retrieve the claims that are to be presented to your application or REST API.

A user journey is built with any combination of steps that constitute a sequence of **orchestration steps** (See below) to interact with internal and external parties.

For example:

- Federate with other identity providers (IdP), a.k.a. claims providers,
- First-and third-party multifactor authentication (MFA) challenges,
- Collect and/or validate any user input,
- Integrate with external systems using REST API communication, i.e., a capability that will be leverage later in this guide to integrate with Microsoft Entra Verified ID.

This orchestration sequence must be followed through for a successful transaction. If any step fails, the transaction fails.

A relying party application or REST API calls a custom policy to execute such a specific user journey.

See User journeys in Azure Active Directory B2C and Sub journeys in Azure Active Directory B2C.

Building blocks

A custom policy is defined by several XML-formatted configuration files, which fer to each other in a hierarchical chain through an inheritance model, where the child policy at any level can inherit from the parent policy and extend it by adding new elements.

In terms of building blocks, the XML elements define the claims schema, claims transformations, content definitions, claims providers, technical profiles, validation technical profiles, user journey orchestration steps, and other aspects of the identity experience:

- **Claims schema**. A claim provides temporary storage of data during a policy execution. It can store information about the user, such as first name, last name, or any other claim obtained from the user or other systems (claims exchanges).

  The claim schema defines the claim types that can be referenced as part of the policy. The claims schema is the place where you declare your claim types. A claim type is similar to a variable in many programmatic languages. You can use the claim type to collect data from the user of your application, receive claims from social identity providers, send and receive data from a custom REST API, or store any internal data used by your custom policy. See ClaimsSchema: Azure Active Directory B2C.

- **Claims transformations**. A claims transformation converts a given claim into another one. In the claims transformation, you specify the transform method, for example adding an item to a string collection or changing the case of a string. See ClaimsTransformations - Azure Active Directory B2C.

- **Content definitions**. A content definition contains URLs to HTML5 templates that can be used in a user journey. The HTML5 page URI is used for a specified user interface step. For example, the sign-in or sign-up, password reset, or error pages.

  You can modify the look and feel and create new content definitions according to your needs. The identity experience framework orchestration platform of Azure AD B2C runs the related code in your customer's browser and uses a modern approach called Cross-Origin Resource Sharing (CORS). See ContentDefinitions - Azure AD B2C.

- **Claims providers**. A claims provider is an interface to communicate with different types of parties via its technical profiles (see below). Every claims provider must have one or more technical profiles that determine the (metadata) endpoints, exact claims exchange definitions, secrets, keys, and certificates as needed, and the protocols needed to communicate with the claims provider.

Multiple technical profiles may be defined because the claims provider supports multiple protocols, various endpoints with different capabilities, or releases different claims at different assurance levels. It may be acceptable to release sensitive claims in one user journey, but not in another. See ClaimsProviders - Azure Active Directory B2C.

- **Technical profiles**. A technical profile provides an interface to communicate with different types of parties. A user journey combines calling technical profiles via orchestration steps (see below) to define your business logic.

  All types of technical profiles share the same concept. You send input claims, run claims transformation, and communicate with the configured party. After the process is completed, the technical profile returns the output claims to claims bag. See Technical profiles - Azure AD B2C.

- **Validation technical profiles**. A validation technical profile is an ordinary technical profile from any protocol, such as Azure AD or a REST API. The validation technical profile returns output claims, or returns 4xx HTTP status code, with an error message. See Define a validation technical profile in a custom policy - Azure AD B2C.

- **User journey orchestration steps**. An orchestration step references to a method that implements its intended purpose or functionality. This method is called a technical profile. When your user journey needs branching to better represent the business logic, the orchestration step references to **sub journey**. A sub journey contains its own set of orchestration steps.

  A user must reach the last step to acquire a security token, but users may not need to travel through all of the orchestration steps. Orchestration steps can be conditionally executed based on **preconditions** defined in the orchestration step.

  After an orchestration step completes, the IEF orchestration platform of Azure AD B2C stores the outputted claims in the **claims bag.** The claims in the claims bag can be utilized by any further orchestration steps in the user journey.

  As such, When running through the orchestration steps, the IEF orchestration platform of Azure AD B2C sends and receives claims to and from internal and external parties and then sends a subset of these claims to your relying party (RP) application or REST API as part of the token.

At this point, you should hopefully have an understanding of what Azure AD B2C is and is and what custom policies are made of. Let's now consider how you can leverage this CIAM foundation with Microsoft Entra Verified ID.

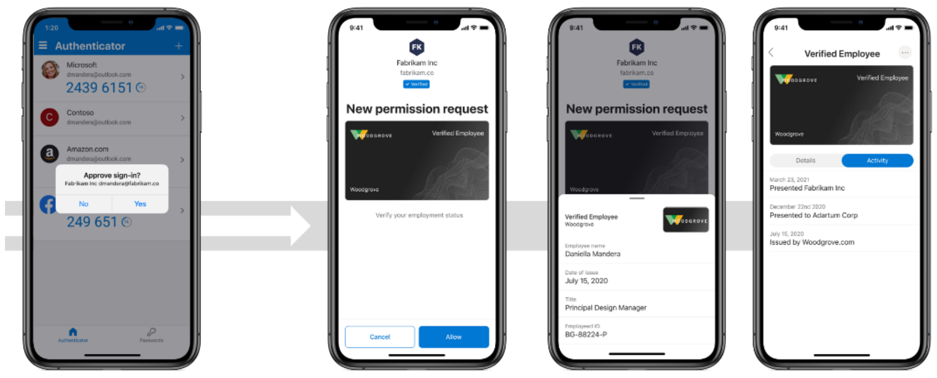
For a deeper look at the custom policies, and beyond the above-outlined articles from the Azure AD B2C documentation on doc.microsoft.com, you can consider the following series of documents that provides you with an end-to-end journey with the custom policies in Azure AD B2C, presenting in-depth the most common advanced identity scenarios:

- Custom policies introduction.
- Leverage custom policies in your tenant.
- Structure policies and manage keys.
- Bring your own identity and migrate users.
- Troubleshoot policies and audit access.
- Deep dive on custom policy schema.

It includes how to implement and manage custom policies for these scenarios and how to diagnose them with the available tooling. It also provides an in-depth understanding of how custom policies work and details how to fine-tune them to accommodate your own specific requirements.

Towards VC enabled identity experience in Azure AD B2C

We believe Microsoft Entra Verified ID a better way to verify identity.



easy to use and secure verifiable transparent convenient

This guide features the Microsoft Entra Verified ID and illustrates how work together with in order to have Verifiable Credentials (VCs) for B2C accounts, and thus how to integrate with the Azure AD B2C custom policies that exists in this repository.

Considering the above-mentioned capabilities, two general areas will frame the approach from a solution perspective. One is the REST API technical profile in Azure AD B2C and the other is the content definitions in Azure AD B2C, see:

- Define a RESTful technical profile in a custom policy.
- Customize the user interface with HTML templates.

As such, the former and the REST API makes it indeed possible to (call an API façade endpoint to build requests and in turn) invoke the Request Service REST API included with Microsoft Entra Verified ID. This API provides an abstraction layer and integration to the Microsoft Entra Verified ID service in order to both issue and verify VCs, see:

- How to call the Request Service REST API.
- Specify the Request Service REST API issuance request.
- Specify the Request Service REST API verify request.

It frees you from the need to understand the different protocols and encryption algorithms for Verifiable Credentials (VCs) and Decentralized IDs (DIDs). You rather need to simply understand how to format a JSON structure as parameter for the REST API.

This REST API callable from any programming language was used in the code sample application used in the first part resp. second part of this guide to both issue and verify your Verified employee credentials resp. Custom credentials.

In order to i) build both the issuance request(s) and the verify request(s) needed as part of the intended user journeys, and ii) expose the related endpoint(s) so that the IEF orchestration platform of Azure AD B2C can send claims and query presentation status, you will use here for the sake of the illustration a similar code sample application as the one used in the previous parts as a backend for Azure AD B2C to talk to, i.e., as a REST API facade endpoint if you prefer. This application/REST API façade is an ASP.NET core application.

In addition, as part of a custom policy definition, both an adequate claims provider and a RESTful technical profile must be specified for this to work.

The latter with custom HTML5 and self-asserted pages makes it possible to do ajax calls in JavaScript to generate the QR code. An adequate content definition as per the considered custom policy definition must also be provided.

This third part of this guide will cover all of the above, and throughout it, you will issue:

- Issue VCs for accounts in your Azure AD B2C tenant.
- Issue VCs during signup of new users in your Azure AD B2C tenant.
- Sign-in to B2C with your VCs by scanning a QR code.

Guide elements

For the sake of this walkthrough, we provide the following elements:

- **Hands-on instructions** containing the most important steps and their outputs. These are meant to show you the core elements.
- **Sample Azure AD B2C custom policies** that you can download, clone, or fork from the following GitHub repository: https://github.com/philber/entra-verifiedid-tour.
- **Sample code application** adapted to work together with Azure AD B2C as a callable REST API façade. You should use it when integrating with the sample Azure AD B2C custom policies. You can download, clone, or fork it from the previous GitHub repository

Guide prerequisites

We assume here that you have a Windows 10 local environment or above.

Install Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

You can download and install Git:

- Download the Git for Windows and run it.
- In the pop-up window, click **Install**. Follow the instructions.

Git will help us to clone the sample code project and related resources to complete this guide.

Clone the repo using Git

To clone both the set of Azure AD B2C custom policies and the sample code application's project on your Windows 11 local machine, open a PowerShell console, and run the following commands:

```
PS C:\> cd c:\
PS C:\> git clone https://github.com/philber/entra-verifiedid-tour.git
PS C:\> cd entra-verifiedid-tour\part-3
```

The sample Azure AD B2C custom policies are forked from the repo available at https://github.com/Azure-Samples/active-directory-verifiable-credentials/tree/main/B2C.

Likewise, the sample code application is forked from the repo available at https://github.com/Azure-Samples/active-directory-verifiable-credentials-dotnet. The related code is located in the *3-asp-net-core-api-b2c* folder.

Repository organization

The repo contains a series of custom policies for your Azure AD B2C tenant to issue resp. verify VCs as part of your signup resp. sign-in policies.

For the sake of this complementary walkthrough, under the folder where you cloned the repo, e.g., *C:\>entra-verifiedid-tour* in our illustration, we will mainly be interested in the *part-3* directory, and the folders underneath:

- *CredentialFiles:* contains VCs display and rules definition files.
    - The *display* definition file provides the content to control the branding of the credential and styling of the claims.
    - The *rules* definition file provides the content to describe important properties of VCs, and the rules that apply. In particular, it describes the claims that subjects (users) need to provide before a credential is issued for them.
- *Policies:* contains a series of (almost) ready-to-use custom policies for your Azure AD B2C tenant.

  This folder contains in turn a *html* subfolder with a series of content definitions for these custom policies.
- *wwwroot:* contains some html files for the code sample application front-end side.

The root of the *part-3* directory contains the adapted code sample application to work in conjunction with the above series of ready-to-use custom policies.

So, at this stage, you're all set! It's high time to move to the first module of this part.

Fulfill the prerequisites for your development environment

In order to complete this walkthrough, we assume that you already fulfilled the prerequisites expressed in the first and second parts of this guide.

The below prerequisites are indeed incremental. Let's consider them in order.

Fulfill the prerequisites for your Azure testing environment

Azure AD B2C tenant

In addition to your Azure AD tenant already being leveraged as part of this guide, an additional Azure AD B2C tenant is required to complete this third part. We assume at the stage that you already have an Azure subscription in place of which you're an administrator.

Create your Azure B2C AD tenant

See Tutorial - Create an Azure Active Directory B2C tenant for more details.

To get started, you first need an Azure B2C AD tenant if you don't have any for this walkthrough.

Unless noticed otherwise, all the required resources will be created in the FranceCentral region. If you want to setup the environment in another region, you will need to adapt the instructions accordingly.

Usage charges for Azure AD B2C are billed to an Azure subscription. You need to explicitly link your Azure subscription.

Be ready for B2C custom policies

See Tutorial - Create user flows and custom policies for more details.

You then need to make the Azure AD B2C tenant ready for the custom policies.

As already introduced, custom policies are a set of XML-formatted configuration files you upload to your Azure AD B2C tenant to define your user journeys, and thus the orchestration steps, the related technical profiles, etc.

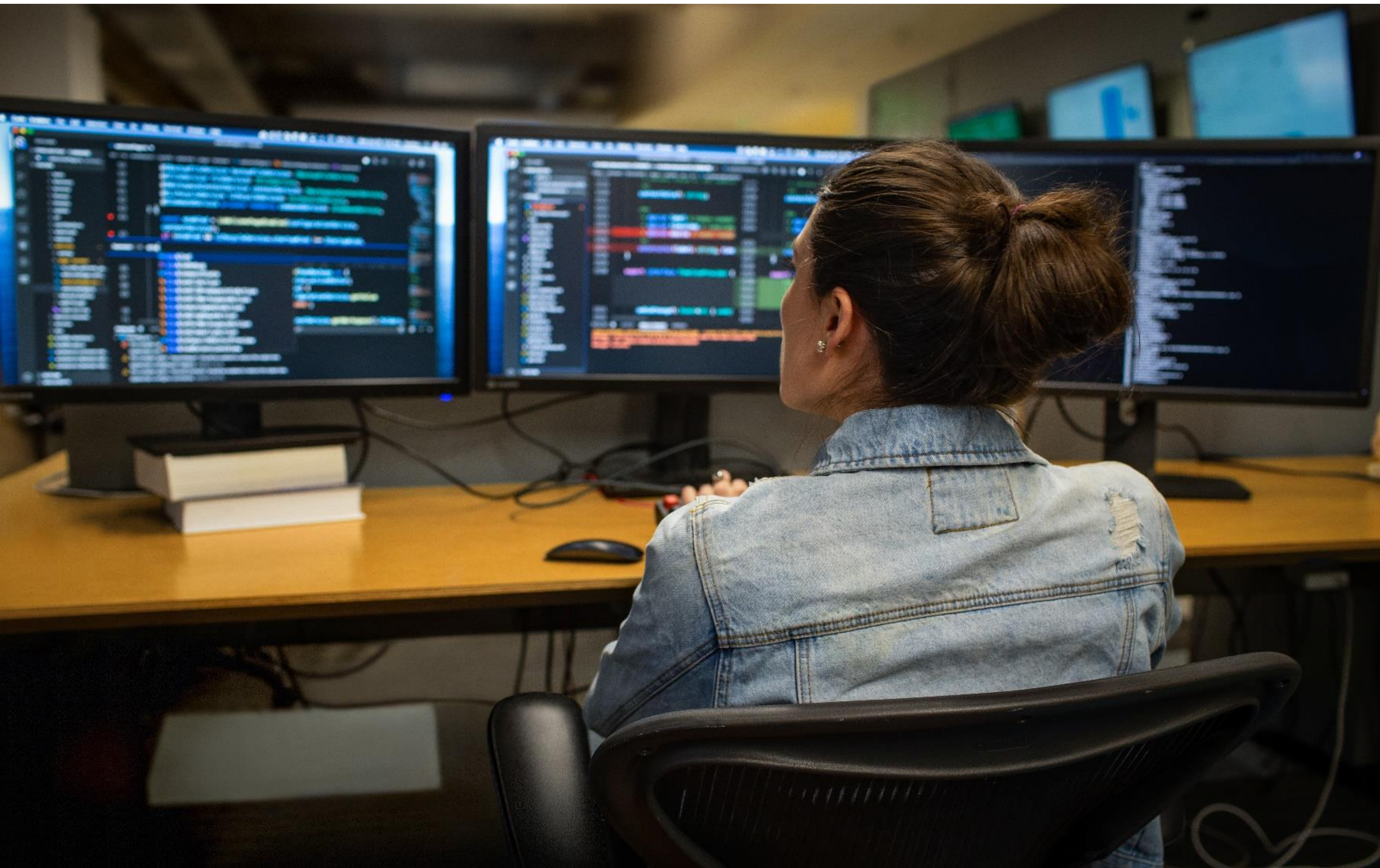
Interestingly enough, so-called starter packs are provided with several pre-built custom policies to get you going quickly. These starter packs are available here : https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack.

Each of these starter packs contains the smallest number of technical profiles and user journeys needed to achieve the scenarios described:

Starter pack	Description
LocalAccounts	Enables the use of local accounts only
SocialAccounts	Enables the use of social (or federated) accounts only
SocialAndLocalAccounts	Enables the use of both local and social accounts
SocialAndLocalAccountsWithMfa	Enables social, local, and multi-factor authentication (MFA) options

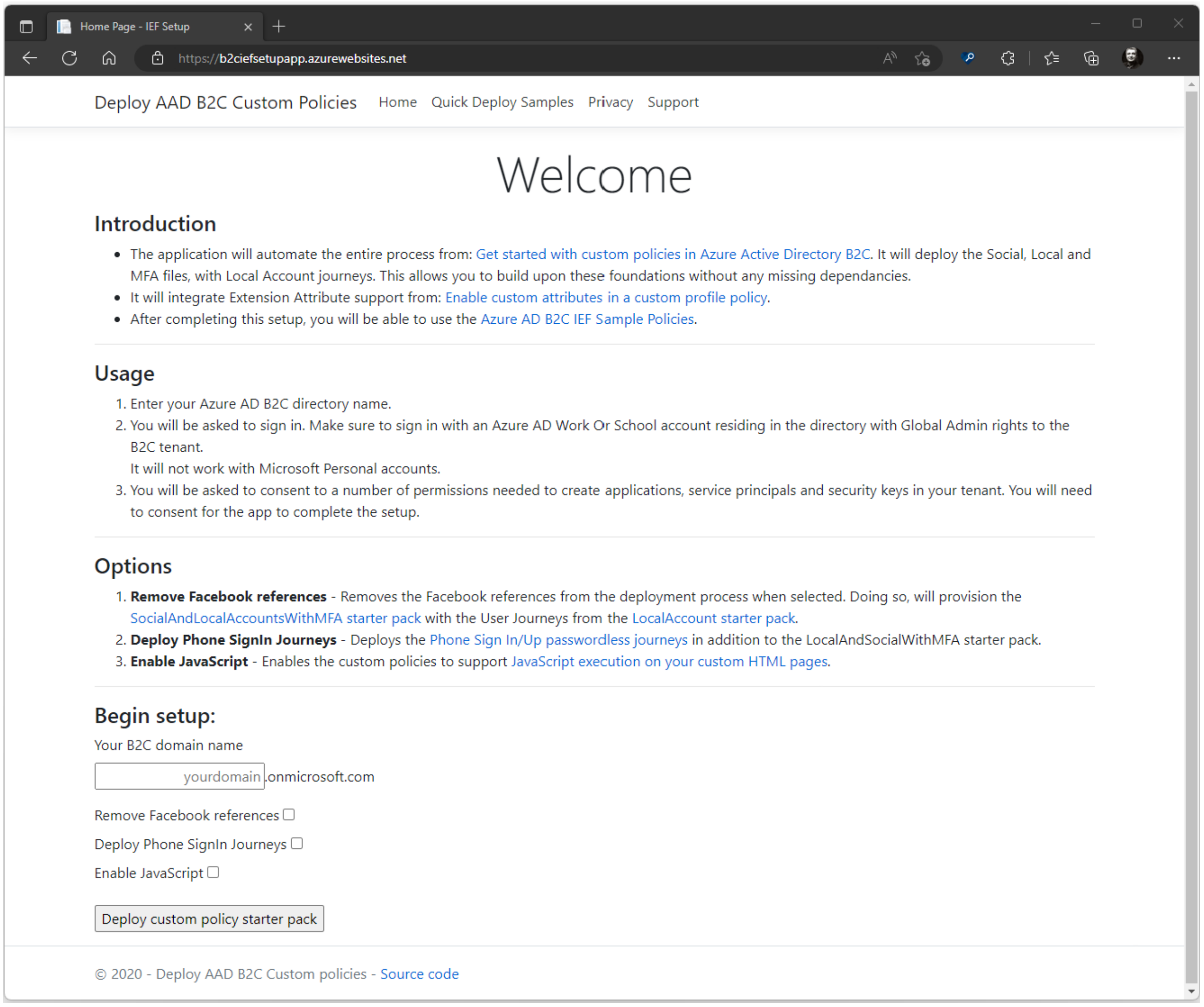
Each above starter pack contains:

File	Description
Base file	Few modifications are required to the base. Example: *TrustFrameworkBase.xml*
Localization file	This file is where localization changes are made. Example: *TrustFrameworkLocalization.xml*
Extension file	This file is where most configuration changes are made. Example: *TrustFrameworkExtensions.xml*
Relying party files	Task-specific files called by your application. Examples: *SignUpOrSignin.xml, ProfileEdit.xml, PasswordReset.xml*



To automate the entire process from Tutorial - Create user flows and custom policies, proceed with the following steps:

1. Open a browser session, navigate to the IEF Setup App at https://aka.ms/iefsetup, and follow the instructions.


Home Page - IEF Setup
https://b2ciefsetupapp.azurewebsites.net
Deploy AAD B2C Custom Policies
Home
Quick Deploy Samples
Privacy
Support

Welcome

Introduction

- The application will automate the entire process from: Get started with custom policies in Azure Active Directory B2C. It will deploy the Social, Local and MFA files, with Local Account journeys. This allows you to build upon these foundations without any missing dependancies.
- It will integrate Extension Attribute support from: Enable custom attributes in a custom profile policy.
- After completing this setup, you will be able to use the Azure AD B2C IEF Sample Policies.

Usage

1. Enter your Azure AD B2C directory name.
2. You will be asked to sign in. Make sure to sign in with an Azure AD Work Or School account residing in the directory with Global Admin rights to the B2C tenant.
   It will not work with Microsoft Personal accounts.
3. You will be asked to consent to a number of permissions needed to create applications, service principals and security keys in your tenant. You will need to consent for the app to complete the setup.

Options

1. **Remove Facebook references** - Removes the Facebook references from the deployment process when selected. Doing so, will provision the SocialAndLocalAccountsWithMFA starter pack with the User Journeys from the LocalAccount starter pack.
2. **Deploy Phone SignIn Journeys** - Deploys the Phone Sign In/Up passwordless journeys in addition to the LocalAndSocialWithMFA starter pack.
3. **Enable JavaScript** - Enables the custom policies to support JavaScript execution on your custom HTML pages.

Begin setup:

Your B2C domain name

yourdomain .onmicrosoft.com

Remove Facebook references ☐

Deploy Phone SignIn Journeys ☐

Enable JavaScript ☐

Deploy custom policy starter pack

© 2020 - Deploy AAD B2C Custom policies - Source code

2. Under **Begin setup**:
   a. Enter the domain name of your Azure AD B2C tenant. For example, litware369b2c in our illustration: litware369b2c.onmicrosoft.com.
   b. Check **Enable JavaScript**.
   c. Click **Deploy custom policy starter pack**. This will automatically deploy the above-mentioned SocialAndLocalAccountsWithMfa starter pack in your Azure AD B2C tenant, which will provide Sign Up and Sign In, Password Reset and Profile Edit journeys.
3. Sign-in with an account with admin privileges in your Azure AD B2C tenant - The account that was used to create the tenant has these by default –.
4. Azure AD B2C will ask you to consent to the application having the ability to create objects in your Azure AD B2C tenant (applications, keys).

5. Once you consent, the IEF Setup App will check whether your tenant has all the objects named in the above tutorial.
6. If these objects, do not exists, the IEF Setup App will create them:
   - Two applications: Azure AD B2C requires you to register two applications that it uses to sign up and sign in users with local accounts: i) **IdentityExperienceFramework**, a web API, and ii) **ProxyIdentityExperienceFramework**, a native app with delegated permission to the IdentityExperienceFramework app. Your users can sign up with an email address or username and a password to access your tenant-registered applications, which creates a "local account." Local accounts exist only in your Azure AD B2C tenant.
   - Two service principals for these two applications.
   - Two keys: Azure AD B2C requires you to create both i) a signing key named **B2C_1A_TokenSigningKeyContainer**, and ii) an encryption key named **B2C_1A_TokenEncryptionKeyContainer**.
7. The final screen will display the relevant application IDs needed in the IEF custom policies.
8. If the application did not exist already, the final screen will provide a URL link you should use to complete admin consent for the new applications to use each other item 9 in the above-mentioned Tutorial.
9. Use the Enterprise Apps option of the Azure portal's AAD blade to remove the B2CIEFSetup service principal from your Azure AD B2C tenant (optional).

For your convenience, the complete code of the IEF Setup App is shared on GitHub. A special thanks to Marius Rochon.

Fulfill the prerequisites for your local environment

Throughout this third part of this guide, we similarly assume you have a Windows 11 machine for the purpose of the illustration.

This said, please note that the same exact above prerequisites can be installed on a macOS or a Linux machine instead, and all the provided instructions apply in a similar way on such environments as well. For the sake of brevity, we neither cover these configurations in the rest of this guide nor the use of the Windows Subsystem for Linux 2 (WSL2) with a Linux distro (e.g., Ubuntu 20.04.4 LTS).

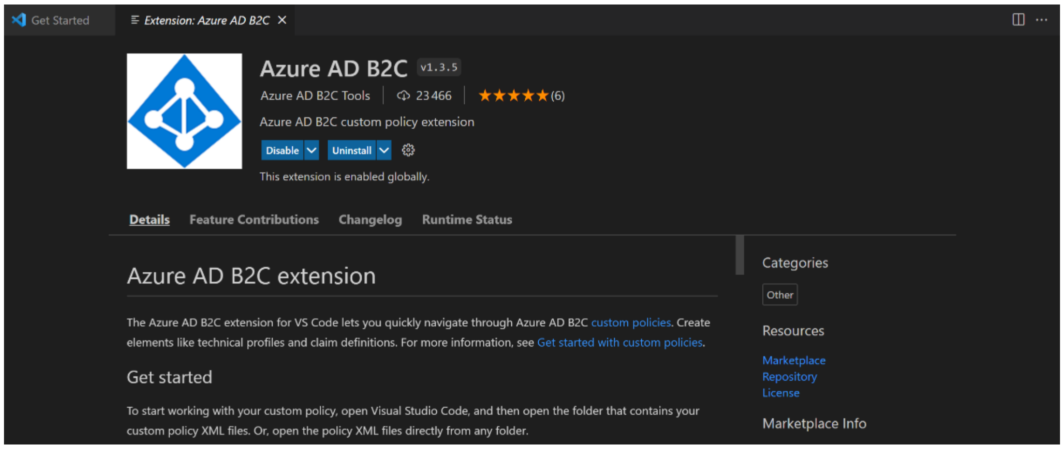
Let's see how to install the additional prerequisites. Let's consider first the edit of the XML-formatted files of your user journeys with the editor of your choice. For example, you can use Visual Studio Code, a lightweight cross-platform editor.

Install Azure AD B2C extension for Visual Studio Code

The Azure AD B2C extension for VS Code allows you navigate faster through you set of Azure AD B2C custom policies and create elements like technical profile, user journeys, claims definitions in your local environment and deploy them to your Azure tenant instead of using the Azure portal.

To install Azure AD B2C extension, proceed with the following steps:

1. Launch Visual Studio Code.
2. Navigate to **Extensions**.
3. In the Search bar, type "*Azure AD B2C*" and select the first extension.
4. Click **install**. Once completed, you should have a tab opening with the following view.



Azure AD B2C custom policy extension

This extension is enabled globally.

Azure AD B2C extension

The Azure AD B2C extension for VS Code lets you quickly navigate through Azure AD B2C custom policies. Create elements like technical profiles and claim definitions. For more information, see Get started with custom policies.

Get started

To start working with your custom policy, open Visual Studio Code, and then open the folder that contains your custom policy XML files. Or, open the policy XML files directly from any folder.

To start working with your custom policies, right from Visual Studio Code, you can open the folder that contains your XML-formatted files, e.g., *C:\>entra-verifiedid-tour\part-3\Policies*, or, open the XML-formatted files directly from any folder.

If the XML, or the XML Tools extensions for Visual Studio Code are installed and activated, the XML extension handles the XML completion within the edited files, See Troubleshoot custom policies and user flows in Azure Active Directory B2C.

In addition, you can also upload your custom policies from Visual Studio Code. This requires you to register a MS Graph delegated permissions application, see Upload a custom polices directly from Azure AD B2C vscode extension.

Install the IefPolicies PowerShell tools

To manage, i.e., create, extend, import, and export the XML-formatted files used for your own custom journeys in your Azure AD B2C tenant, you can leverage the so-called IefPolicies PowerShell tools.

The set of cmdlets is available in the PowerShell Gallery. Minimal PowerShell version is 7.0.

To install the IefPolicies cmdlets, proceed with the following steps:

1. Open a PowerShell 7 window with Administrative privileges.
2. Run the following command:

```
PS C:\> Set-PSRepository -Name "PSGallery" -InstallationPolicy Trusted
PS C:\> Install-Module -Name IefPolicies
```

Install ASP.NET Core Runtime 3.1

Unlike the first and second parts of this guide, you will leverage this time a code sample application's project in .NET Core 3.1.

The ASP.NET Core Runtime enables you to run web/server applications. On Windows, it's recommended to install the Hosting Bundle. It includes the .NET Core runtime, the ASP.NET Core runtime, and if installed on a machine with IIS it will also add the ASP.NET Core IIS Module. (If you only want the .NET Core or ASP.NET Core runtime, you'll find them in .NET Core 3.1 Downloads.)

Download the Hosting Bundle, i.e., the *dotnet-hosting-3.1.27-win.exe* file, execute it and follow the instructions.

The code sample application(s) used in this guide can be containerized and specific instructions are given in the repo you cloned to do so. For that purpose, you will need a Docker environment.

Install Docker Desktop for Windows

See Install Docker Desktop on Windows.

Docker Desktop is an application for Windows machines for the building and sharing of containerized applications and microservices. As such, it includes Developer tools, Kubernetes and version synchronization to production Docker Engines, i.e. the container runtimes.

It allows you to leverage certified images and templates and your choice of languages and tools. Development workflows leverage Docker Hub to extend your development environment to a secure repository for rapid auto-building, continuous integration and secure collaboration.

To install Docker for Windows on your local development environment, proceed with the following steps:

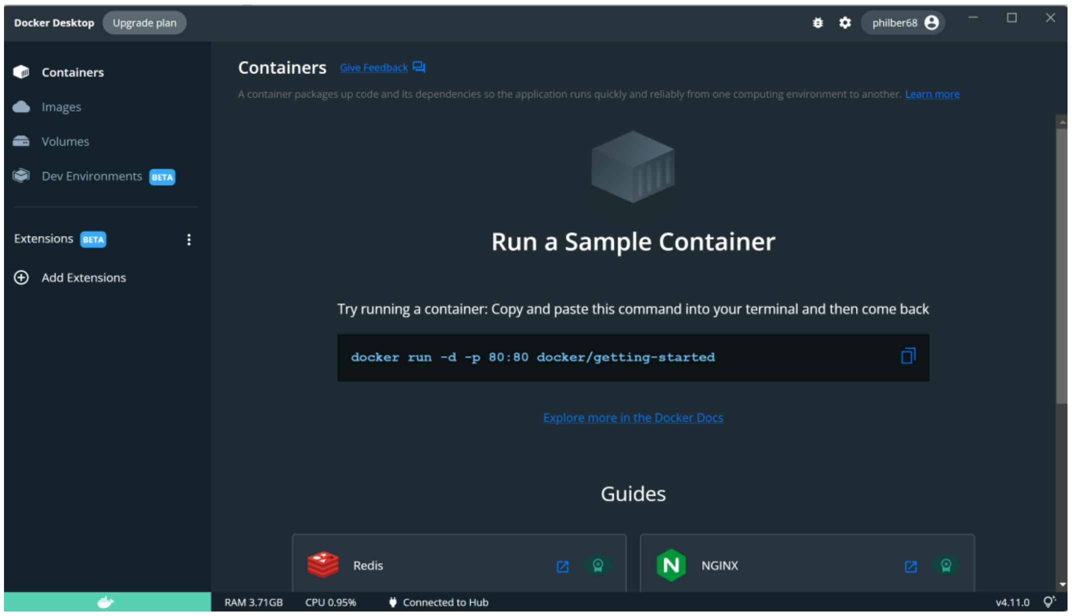
1. Download the setup file from the Docker Hub at https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe.
2. After downloading the *Docker Desktop Installer.exe* setup file, from the previous PowerShell 7window, run the following command:

```
PS C:\> Start-Process '.\Docker Desktop Installer.exe' -Wait install
```

3. Optionally, if your admin account is different to your user account, you must add the user to the docker-users group:
   a. From the PowerShell 7 windows run the following command:

```
PS C:\> compmgmt.msc
```

   b. The Computer Management MMC snap-in opens up. Navigate to **Local Users and Groups** and **Groups**, right-click **docker-users**.
   c. Click **Add...**
   d. Type your user account and click **OK** twice.
   e. Log out and log in again for the changes to take effect.
4. Docker Desktop does not start automatically after installation. To start Docker Desktop, search for Docker, and select **Docker Desktop** in the search results.
5. The Docker menu displays the Docker Subscription Service Agreement window - It includes a change to the terms of use for Docker Desktop -. Click the checkbox to indicate that you accept the updated terms and then click **Accept** to continue. Docker Desktop starts after you accept the terms.



A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Try running a container: Copy and paste this command into your terminal and then come back

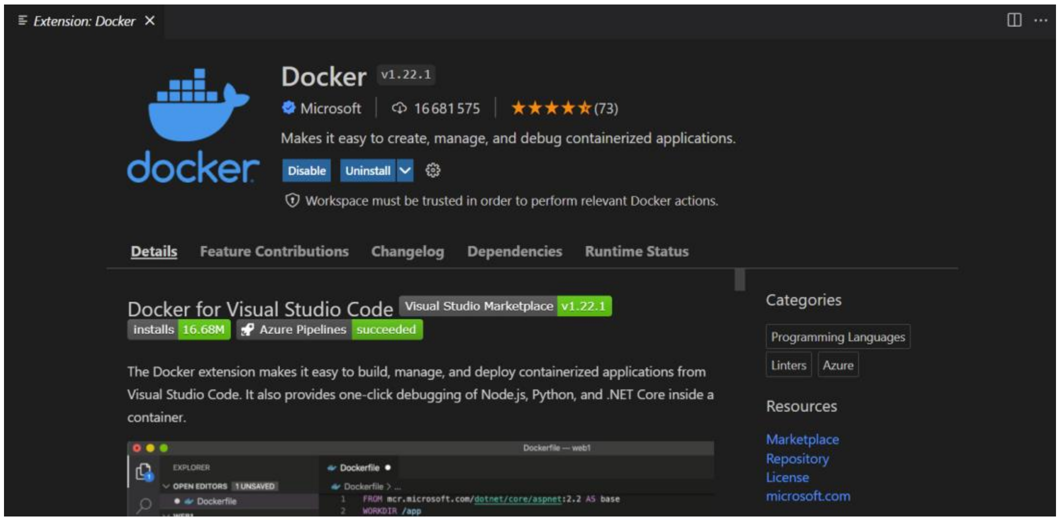
```
docker run -d -p 80:80 docker/getting-started
```

Install the Docker extension for Visual Studio Code

The [Docker]() extension makes it easy to build, manage, and deploy containerized applications in Visual Studio Code.

To install the extension, proceed with the following steps:

1. Launch Visual Studio Code.
2. Navigate to **Extensions**.
3. In the Search bar, type "*Docker*" and select the first extension.
4. Click **install**. Once completed, you should have a tab opening with the following view.



Makes it easy to create, manage, and debug containerized applications.

Workspace must be trusted in order to perform relevant Docker actions.

Docker for Visual Studio Code

The Docker extension makes it easy to build, manage, and deploy containerized applications from Visual Studio Code. It also provides one-click debugging of Node.js, Python, and .NET Core inside a container.

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS base
WORKDIR /app
```

Check out the [Docker in Visual Studio Code]() to get started. [The Docker extension wiki]() has troubleshooting tips and additional technical information.

At this stage, you are ready to further configure your Azure testing environment to integrate Microsoft Entra Verified ID with Azure AD B2C.

Configure your development environment

In order to complete this walkthrough, we assume that you already followed the first and second parts of this guide, and thus deployed your own Microsoft Entra Verified ID environment along with the code sample applications in ASP.NET Core, and ensured that everything worked fine as expected with the Verified employee credential and the provided Custom credential for your employees. This chapter considers such a foundation as a starting point.

If such a foundation isn't yet in place, please refer to these first and second parts of this guide to establish it in your Azure subscription.

In this chapter, you will now:

- Register Microsoft Authenticator in Azure AD B2C
- Configure and upload the provided Azure AD B2C custom policies located in the *Policies* folder underneath the *part-3* directory.
- Create a new B2C Custom credential that uses your Azure AD B2C tenant for issuance of VCs .
- Configure the code sample application to both issue and use your B2C Custom credential.
- Run the code sample application locally.
- Test it against the IEF orchestration platform.

These steps are covered in order in the next sections. Let's start with the registration of Microsoft Authenticator.

Register Microsoft Authenticator in Azure AD B2C

See Tutorial: Register a web application in Azure Active Directory B2C for more details.

Before any of your mobile applications, web applications, and REST APIs can interact with your Azure AD B2C tenant, they must be registered in it.

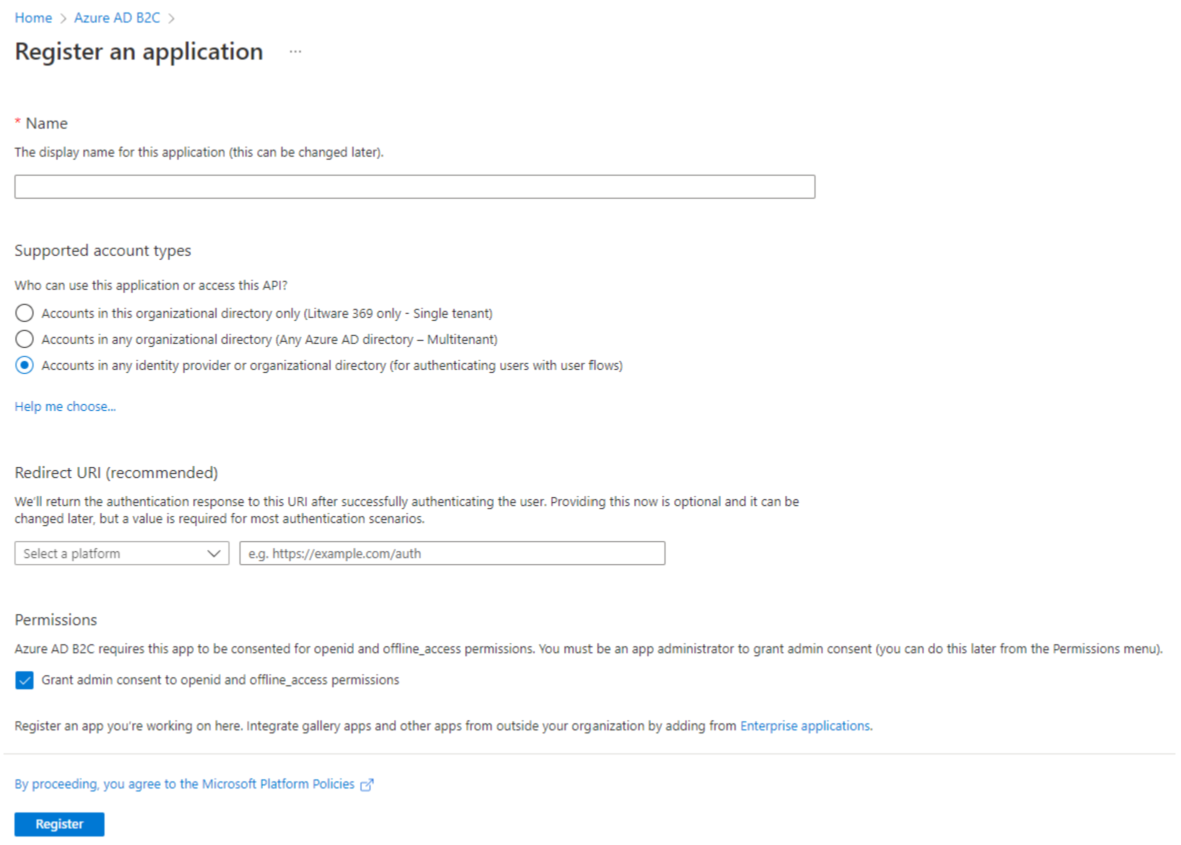
In our illustration, Microsoft Authenticator will be used to authenticated the user against your Azure AD B2C tenant in order to get in return an ID token from which the claims asserted by the tenant will be used in turn to issue a Custom credential.

Register Microsoft Authenticator in your B2C tenant

This is the exact same approach as the one taken in the second part of this guide as per Issuer service communication examples.

To register Microsoft Authenticator, proceed with the following steps:

1. Open a browser session, navigate to the Azure portal at https://portal.azure.com, and sign in with an account with admin privileges in your Azure AD B2C tenant - The account that was used to create the tenant has these by default -.
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directories + subscriptions** icon in the portal toolbar.
3. On the **Portal settings | Directories + subscriptions** page, find your Azure AD B2C directory in the **Directory** name list, and then select **Switch**.
4. In the Azure portal, search for and select **Azure AD B2C**.
5. Select **App registrations**, and then select **New registration**.



Register an application

* Name

The display name for this application (this can be changed later).

Supported account types

Who can use this application or access this API?

- ○ Accounts in this organizational directory only (Litware 369 only - Single tenant)
- ○ Accounts in any organizational directory (Any Azure AD directory – Multitenant)
- ● Accounts in any identity provider or organizational directory (for authenticating users with user flows)

Redirect URI (recommended)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Permissions

Azure AD B2C requires this app to be consented for openid and offline_access permissions. You must be an app administrator to grant admin consent (you can do this later from the Permissions menu).

✓ Grant admin consent to openid and offline_access permissions

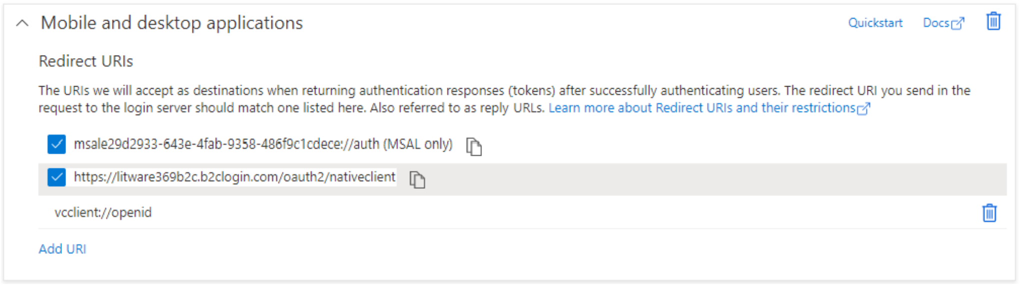
Register an app you're working on here. Integrate gallery apps and other apps from outside your organization by adding from Enterprise applications.

By proceeding, you agree to the Microsoft Platform Policies

   a. Under **Name**, specify "<*Issuer_Name*> *Verifiable Credential Service*" for the application name, where <*Issuer_Name*> is the name of your organization. For example, "*Litware 369 Inc Verifiable Credential Service*" in our illustration.
   b. Under **Supported account types**, select **Accounts in any identity provider or organizational directory (for authenticating users with user flows)**.
   c. Under **Redirect URI (recommended)**, select **Public client/native (mobile & desktop)**, and then set the redirect URI to `vcclient://openid`.
   d. Under **Permissions**, leave **Grant admin consent to openid and offline_access permissions** checked.
   e. Select **Register**.

6. Once registered, a new blade opens up for this application. In the left menu, under **Manage**, select **Authentication**.

a. Under **Mobile and desktop applications**, select the listed checkboxes in addition to the redirect URI you've specified above.



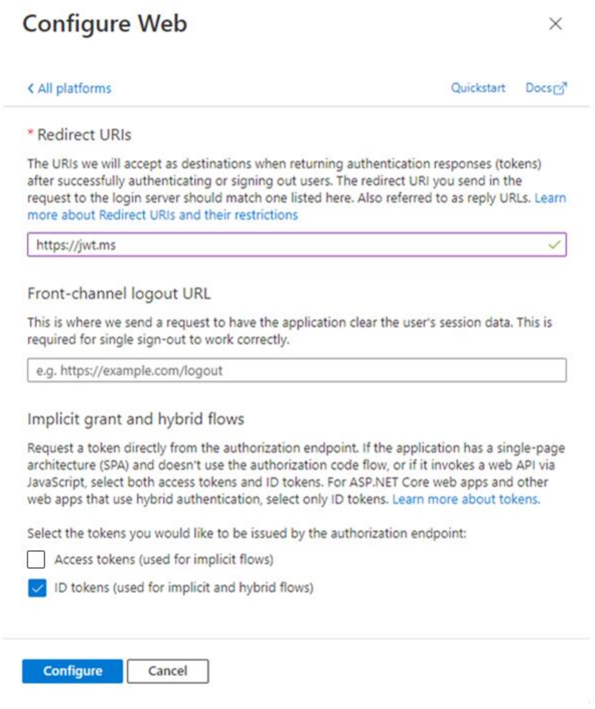
The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

- [✓] msale29d2933-643e-4fab-9358-486f9c1cdece://auth (MSAL only)
- [✓] https://litware369b2c.b2clogin.com/oauth2/nativeclient
- vcclient://openid

b. If you want to be able to later test what claims are in the ID token, do the following. Click **+ Add a platform**. In the **Configure platforms** blade, under **Web Applications**, select **Web**.

c. For **Redirect URIs**, enter "*https://jwt.ms*". This redirect URI is the endpoint to which the user is sent by the IEF orchestration platform of Azure AD B2C after completing its interaction with the user, and to which a security token is sent upon a successful user journey.

d. Check **ID tokens (used for implicit and hybrid flows)**.



* Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

https://jwt.ms

Front-channel logout URL

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

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

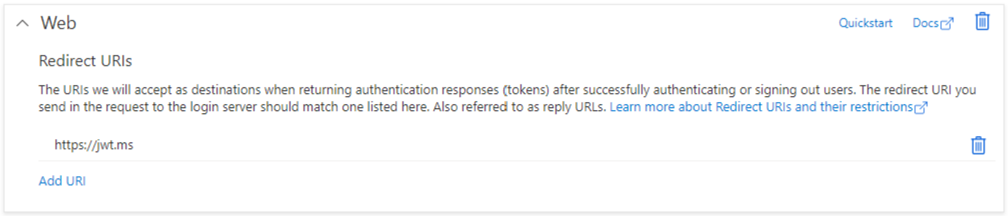
Implicit grant and hybrid flows

Request a token directly from the authorization endpoint. If the application has a single-page architecture (SPA) and doesn't use the authorization code flow, or if it invokes a web API via JavaScript, select both access tokens and ID tokens. For ASP.NET Core web apps and other web apps that use hybrid authentication, select only ID tokens. Learn more about tokens.

Select the tokens you would like to be issued by the authorization endpoint:

- [ ] Access tokens (used for implicit flows)
- [✓] ID tokens (used for implicit and hybrid flows)

e. Click **Configure**.



Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

https://jwt.ms

Add URI

f. Eventually click **Save**.

7. Now create a client secret for this newly registered application. The secret will be used by your application to exchange an authorization code for an access token. In the left menu, under **Manage**, select **Certificates & secrets**.

   a. Select **New client secret**.
   b. In **Description**, enter a description for the client secret. For example, "*clientsecret1*".
   c. Under **Expires**, select a duration for which the secret is valid, and then select **Add**.
   d. Record the secret's value for use in your client application code.

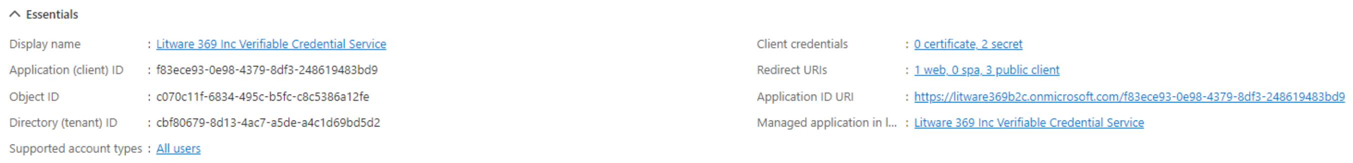
   This secret value is never displayed again after you leave this page. You use this value as the application secret in your application's code.

Take a note of your Client ID

The Client ID will be used shortly for the Custom Credential display and rules definitions to specify your Custom credential for the B2C users.

From the **Overview** page of your newly created application for Microsoft Authenticator, take a note of:

- The **Application (client) ID** value. For example `f83ece93-0e98-4379-8df3-248619483bd9` in our illustration.
- The **Directory (tenant) ID** value. For example `cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2` in our illustration.



Display name	: Litware 369 Inc Verifiable Credential Service	Client credentials	: 0 certificate, 2 secret
Application (client) ID	: f83ece93-0e98-4379-8df3-248619483bd9	Redirect URIs	: 1 web, 0 spa, 3 public client
Object ID	: c070c11f-6834-495c-b5fc-c8c5386a12fe	Application ID URI	: https://litware369b2c.onmicrosoft.com/f83ece93-0e98-4379-8df3-248619483bd9
Directory (tenant) ID	: cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2	Managed application in l...	: Litware 369 Inc Verifiable Credential Service
Supported account types	: All users		

Configure and upload the B2C custom policies

The policies for the B2C+VC integration are B2C custom policies with custom html, which is needed to generate the QR code.

You will therefore need to first upload the .html files to your Azure Storage account and then edit and upload the .xml custom policy files. See next section.

File	Id	Description
TrustFrameworkExtensionsVC.xml	B2C_1A_TrustFrameworkExtensionsVC	Extensions for VC's to the TrustFrameworkExtensions from the Starter Pack
SignUpVCOrSignin.xml	B2C_1A_VC_susi_issuevc	Standard Signup/Sign-in B2C policy but that issues a VC during user signup
SignUpOrSignInVC.xml	B2C_1A_signup_signin_VC	Standard Signup/Sign-in B2C policy but with VC added as a claims provider option (button)
SignupOrSigninVCQ.xml	B2C_1A_VC_susiq	Standard Signup/Sign-in B2C policy but with a QR code on sign-in page so you can scan it already there. Signup journey ends with issue the new user a VC via the id_token flow.
SigninVC.xml	B2C_1A_signin_VC	Policy that lets you sign-in to your B2C account via scanning the QR code and present your VC
SigninVCMFA.xml	B2C_1A_signinmfa_VC	Policy that uses VCs as a MFA after you've signed in with userid/password

Deploy the custom html templates

See Customize the user interface with HTML templates.

Set up an Azure Blob storage account for storing the custom .html files

Create a storage account

See Create a storage account.

Azure Blob Storage is an object storage solution for the cloud. Microsoft Entra Verified ID uses Azure Blob Storage to store the configuration files when the service is issuing verifiable credentials.

To create a Blob Storage account, proceed with the following steps:

1. Open a PowerShell 7 session and authenticate per section **Install Azure Az PowerShell module** in the first part of the guide.
2. Create a resource group:

```
PS C:\> New-AzResourceGroup -Name <yourResourceGroup> -Location "France Central"
```

Replace the placeholder value in brackets with your own value. For example in our illustration:

```
PS C:\> New-AzResourceGroup -Name "B2C_Configuration" -Location "France Central"
```

3. Next create the Blob Storage account:

```
PS C:\> New-AzStorageAccount -ResourceGroupName <yourResourceGroup> –AccountName <yourStorageAccount> –
Location "France Central" -SkuName Standard_LRS
```

Replace the placeholder values in brackets with your own values and *France Central* by the Azure region of your choice.

For example in our illustration:

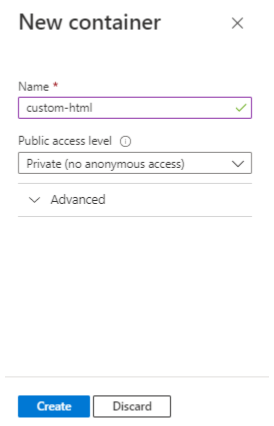
```
PS C:\> New-AzStorageAccount -ResourceGroupName "B2C_Configuration" –AccountName "litware369b2cstorage" –
Location "France Central" -SkuName Standard_LRS
```

After you've created the storage account, create a public container.

Create a public storage container

To create a public container in the above storage account, proceed with the following steps:

1. From the previous browser session, in the Azure portal, search for "*storage accounts*", and then click **Storage accounts**.
2. Click the blob storage account you created previously.
3. In the left pane for the storage account, scroll to the **Data storage** section, and select **Containers**.
4. Select **+ Container**. A **New container** blade pops up.
    a. Type a name for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character. For example, "custom-html" in our illustration.
    b. Set **Public access level** to **Blob (anonymous read access for blobs only)**.



Name * custom-html

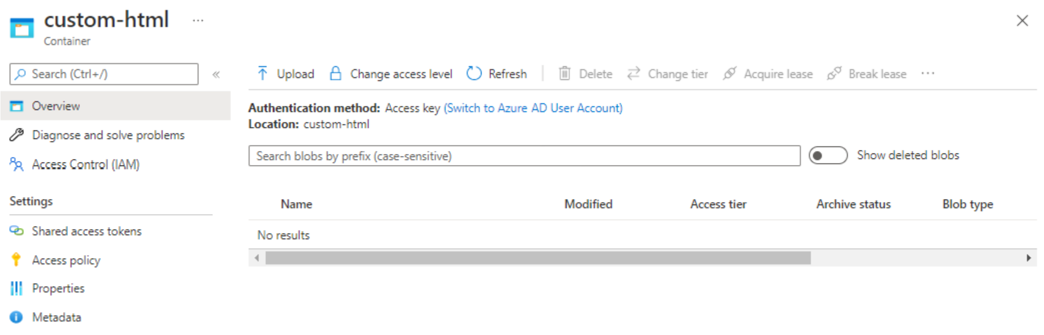
Public access level: Private (no anonymous access)

    c. Select **Create**.

Upload the custom html templates

Proceed with the following steps:

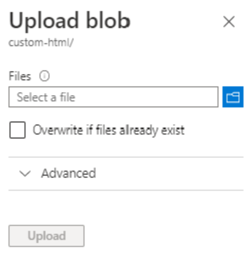
1. From the Azure portal, select the newly created container.



Authentication method: Access key (Switch to Azure AD User Account)
Location: custom-html

No results

2. Select **Upload**. An **Upload blob** blade pops up.



   a. Select the folder icon next to **Select a file**.
   b. From the location where you cloned the repo, navigate to the *html* subfolder under the *Policies* folder of the *part-3* directory, e.g., *c:\entra-verifiedid-tour\part-3\Policies\html*, and select the *selfAsserted.html* file.
      If you want to upload to a subfolder, expand **Advanced** and enter a folder name in **Upload to folder**.
   c. Select **Upload**.

3. Select the *selfAsserted.html* blob that you uploaded.
4. To the right of the **URL** text box, select the **Copy to clipboard** icon to copy the URL to your clipboard. For example, in our illustration:

   https://litware369b2cstorage.blob.core.windows.net/custom-html/selfAsserted.html

5. Open a browser session and navigate to this URL you copied to verify the blob you uploaded is accessible. If it is inaccessible, for example if you encounter a `ResourceNotFound` error, make sure the container access type is set to blob. The blob you uploaded is inaccessible, and you might encounter a `ResourceNotFound` error.
6. Repeat step 2 for the *unified.html* and *unifiedquick.html* files.

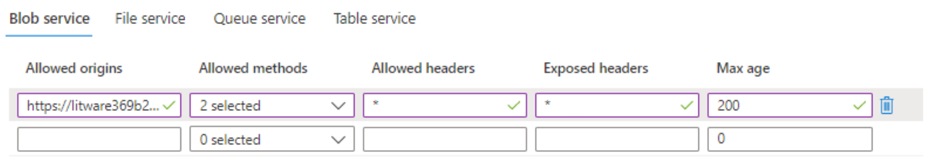
Configure CORS

You now need to configure your blob storage account for Cross-Origin Resource Sharing (CORS). CORS is an HTTP feature that enables a web application running under one domain to access resources in another domain. Web browsers implement a security restriction known as same-origin policy that prevents a web page from calling APIs in a different domain. CORS provides a secure way to allow one domain (the origin domain) to call APIs in another domain.

You can set CORS rules individually for each of the storage services, here the blob storage account. Once you set the CORS rules for your blob storage account, then a properly authenticated request made against the service from a different domain will be evaluated to determine whether it is allowed according to the rules you have specified.

Perform the following steps:

1. From your storage account in the Azure portal, in the left menu, under **Settings**, select **Resource sharing (CORS)**.



Blob service | File service | Queue service | Table service

Allowed origins	Allowed methods	Allowed headers	Exposed headers	Max age
https://litware369b2...	2 selected	*	*	200
	0 selected			0

   a. For **Allowed origins**, enter "*https://<yourTenantName>.b2clogin.com*". Replace *<yourTenantName>* with the name of your Azure AD B2C tenant. For example, https://litware369b2c.b2clogin.com in our illustration. Use all lowercase letters when entering your tenant name.
   b. For **Allowed Methods**, select both **GET** and **OPTIONS**.
   c. For **Allowed Headers**, enter an asterisk (*).
   d. For **Exposed Headers**, enter an asterisk (*).
   e. For **Max age**, enter "*200*".

2. At the top of the page, select **Save**.

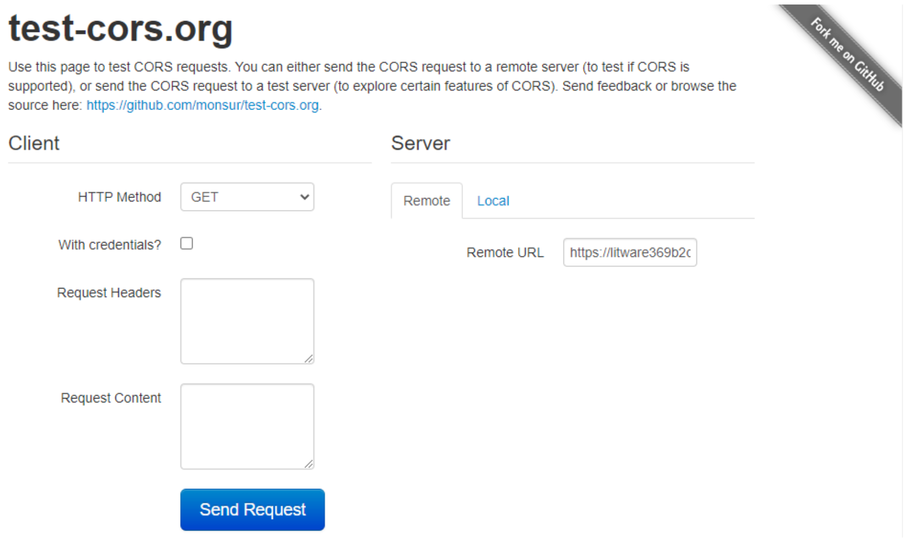
Test CORS

So let's validate at this stage that you're ready.

Perform the following steps:

1. Repeat the above configure CORS steps. For **Allowed origins**, enter "*https://www.test-cors.org*".
2. Navigate to https://www.test-cors.org.
3. In **Remote URL**, paste the above URL of your custom html file. For example, in our illustration:

   https://litware369b2cstorage.blob.core.windows.net/custom-html/selfAsserted.html



test-cors.org

Use this page to test CORS requests. You can either send the CORS request to a remote server (to test if CORS is supported), or send the CORS request to a test server (to explore certain features of CORS). Send feedback or browse the source here: https://github.com/monsur/test-cors.org.

Client

HTTP Method: GET

With credentials? ☐

Request Headers

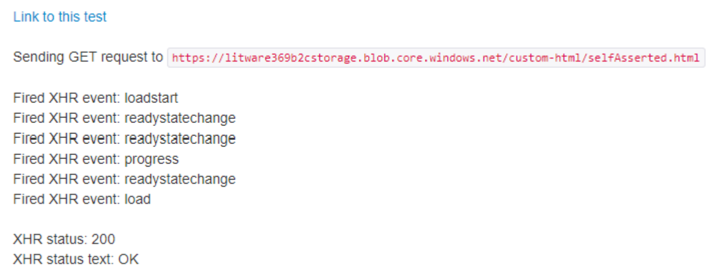
Request Content

Server

Remote | Local

Remote URL: https://litware369b2c

4. Click **Send Request**. The result should be `XHR status: 200` as illustrated hereafter.



Link to this test

Sending GET request to `https://litware369b2cstorage.blob.core.windows.net/custom-html/selfAsserted.html`

```
Fired XHR event: loadstart
Fired XHR event: readystatechange
Fired XHR event: readystatechange
Fired XHR event: progress
Fired XHR event: readystatechange
Fired XHR event: load

XHR status: 200
XHR status text: OK
```

If you receive an error, make sure that your CORS settings are correct. You might also need to clear your browser cache or open an in-private browsing session by pressing Ctrl+Shift+P.

Create the REST API key in the portal

See Secure APIs used as API connectors in Azure AD B2C.

The `REST-VC-PostIssuanceClaims` technical profile defined in the *TrustFrameworkExtensionsVC.xml* file, and that is used for the VC issuance during user signup, is configured to use an api-key for security reasons, namely `B2C_1A_RestApiKey`.

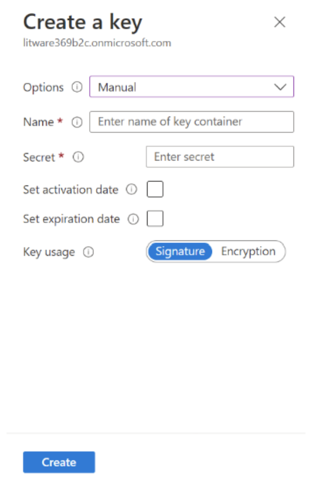
```
<TechnicalProfile Id="REST-VC-PostIssuanceClaims">
   <DisplayName>Verifiable Credentials Authentication Result</DisplayName>
   <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
   <Metadata>
      <!-- CHANGE THE BELOW LINE -->
      <Item Key="ServiceUrl">https://4f89-2a01-110-8012-1013-88b5-de4c-4192-
fb3a.ngrok.io/api/issuer/issuance-request-callback</Item>
      <Item Key="AuthenticationType">ApiKeyHeader</Item>
      <!-- change this to None for dev/debug -->
      <Item Key="SendClaimsIn">Body</Item>
      <Item Key="IncludeClaimResolvingInClaimsHandling">true</Item>
   </Metadata>
   <CryptographicKeys>
      <Key Id="x-api-key" StorageReferenceId="B2C_1A_RestApiKey" />
      <!-- remember to set this in the samples appSettings/etc -->
   </CryptographicKeys>
   <InputClaims>
      […omitted for the sake of brievity…]
   </InputClaims>
   <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

Therefore, you will need at this stage to create a policy key in the Azure AD B2C portal, and (manually) set the key value to something unique - You need to add this value to the code sample application's configuration also -. Otherwise, you will encounter an error in the trace when calling the API:

```
trce: AspNetCoreVerifiableCredentialsB2C.ApiVerifierController[0]
      {"id":"c1a4d8a3-8333-4a9f-b90c-a51262077599"}
fail: AspNetCoreVerifiableCredentialsB2C.ApiVerifierController[0]
      Missing header: x-api-key
```

Proceed with the following steps:

1. Open a browser session, navigate to the Azure portal, and sign in with an account with admin privileges in your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. In the left pane, under **Policies**, select **Identity Experience Framework**.
5. In the left pane, under **Manage**, select **Policy keys**.
6. Create a policy key in the B2C portal with the name `RestApiKey` and manually set the key value to something unique – It will be later referred as to `B2C_1A_RestApiKey` as defined in the above code snippet -. Click **Add** to add a new policy key. This opens up a **Create a key blade**.



   a. For **Option**, select **Manual**.
   b. For the name, enter "*RestApiKey*".
   c. For **Secret**, specify give something unique - a UUID for example, e.g., `e09f4443-b08f-4ed2-8717-7a202b990b40` in our illustration . For that purpose, you can use for example one of the numerous GUID generators available online such as https://guidgenerator.com/. The resulting value will be referred to as the secret value below.
   d. For **Key usage**, select **Encryption**.
   e. Click **Create**.

Configure the B2C custom policies

Run ngrok

if you use ngrok, a tip to leverage here is to start it and leave it running so that your publicly available hostname for the REST API facade exposed by the code sample application will NOT change for the rest of this document.

This will allow you to specify in the custom policy XML-formatted files a publicly available fully qualified name on the Internet, and this name will remain valid after uploading these files in your Azure AD B2C tenant.

As already introduced in the previous parts of this guide, ngrok acts as a reverse proxy to “read” your code sample application.

Proceed with the following steps:

1. Open a PowerShell 7 window.
2. Start a tunnel with ngrok.

```
PS C:\> ngrok http 5002
```

   More specifically, this command runs ngrok to set up a URL on 5002 where the code sample application’s endpoint will be later listening on, and make it publicly available on the Internet. For example https://0a12-93-3-247-151.ngrok.io in our illustration.

3. **Leave the PowerShell 7 window opened for the rest of this part.**

Edit and update the B2C custom policies

Proceed with the following steps:

1. Open a PowerShell 7 window and navigate to the location where you cloned the repo, navigate to the *Policies* folder of the *part-3* directory, e.g., *c:\entra-verifiedid-tour\part-3\Policies*.
2. Open Visual Studio Code.

```
PS C:\> code .
```

3. Search and replace with the command **Replace in Files** (Ctrl+Shift+H) in the set of all .xml custom policy files for *yourtenant*.onmicrosoft.com to the real fully qualified name of your B2C tenant. For example, litware369b2c.onmicrosoft.com in our illustration.
4. Then do the following changes to the *TrustFrameworkExtensionsVC.xml* file:
   a. Find all places where `LoadUri` references your blob storage account and update the value of the url for the .html custom template files you uploaded in the section **Deploy the custom html templates** to reflect your blob storage account.

      For example, you will need to replace yourstorageaccount.blob.core.windows.net/b2cux/ by litware369b2cstorage.blob.core.windows.net/custom-html in our illustration.

```
 <!-- pages uses custom html -->
    <ContentDefinition Id="api.selfasserted.vc">
       <!-- CHANGE THE BELOW LINES -->
       <LoadUri> https://litware369b2cstorage.blob.core.windows.net/custom-
html/selfAsserted.html</LoadUri>
       <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
       <DataUri>urn:com:microsoft:aad:b2c:elements:contract:selfasserted:2.1.0</DataUri>
       <Metadata>
          <Item Key="DisplayName">Verifiable Credentials Signin</Item>
       </Metadata>
    </ContentDefinition>
    <ContentDefinition Id="api.signuporsignin">
       <LoadUri> https://litware369b2cstorage.blob.core.windows.net/custom-html/unified.html</LoadUri>
       <DataUri>urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:2.1.0</DataUri>
       <LocalizedResourcesReferences MergeBehavior="Prepend">
          <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.signuporsignin.en"
/>
       </LocalizedResourcesReferences>
    </ContentDefinition>
    <ContentDefinition Id="api.signuporsignin.quick">
       <LoadUri> https://litware369b2cstorage.blob.core.windows.net/custom-
html/unifiedquick.html</LoadUri>
       <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
       <DataUri>urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:2.1.0</DataUri>
       <LocalizedResourcesReferences MergeBehavior="Prepend">
          <LocalizedResourcesReference Language="en"
LocalizedResourcesReferenceId="api.signuporsigninquick.en" />
        </LocalizedResourcesReferences>
    </ContentDefinition>
```

b. Update all places where `VCServiceUrl` references your local deployment of the code sample application via ngrok. For example, you will need to replace df4a-158-174-131-118.ngrok.io by 0a12-93-3-247-151.ngrok.io in our illustration.

c. Update all places where `ServiceUrl` references your local deployment of the code sample application via ngrok. For example, you will need to replace df4a-158-174-131-118.ngrok.io by 0a12-93-3-247-151.ngrok.io in our illustration.

5. Do the following changes to the *SignupOrSigninVCQ.xml* and *SignUpVCOrSignin.xml* files:

File	Changes
SignupOrSigninVCQ.xml	`OrchestrationStep` 7, 8, and 9 should be changed to 9, 10, and 11
SignUpVCOrSignin.xml	`OrchestrationStep` 7, 8, and 9 should be changed to 9, 10, and 11

As illustrated hereafter with the first file:

```
  <OrchestrationSteps>
      <!-- …omitted for brievity… -->
      </OrchestrationStep>
      <!-- If it's a new user, pass the claims to the VC sample to prepare for issuing a VC -->
      <!-- change 7 to 9 for SocialAndLocalAccountsWithMfa -->
      <OrchestrationStep Order="9" Type="ClaimsExchange">
        <Preconditions>
          <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
            <Value>newUser</Value>
            <Action>SkipThisOrchestrationStep</Action>
          </Precondition>
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="PostVCIssuanceClaims" TechnicalProfileReferenceId="REST-VC-
PostIssuanceClaims" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <!-- If it's a new user, show the QR code to the user in order to issue the VC -->
      <!-- change 8 to 10 for SocialAndLocalAccountsWithMfa -->
      <OrchestrationStep Order="10" Type="ClaimsExchange">
        <Preconditions>
          <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
            <Value>newUser</Value>
            <Action>SkipThisOrchestrationStep</Action>
          </Precondition>
        </Preconditions>
        <ClaimsExchanges>
          <ClaimsExchange Id="UXVCIssuanceClaims" TechnicalProfileReferenceId="SelfAsserted-VCIssuance" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <!-- NOTE! If you are using the starter pack SocialAndLocalAccountsWithMfa, change 9 to  11 -->
      <OrchestrationStep Order="11" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer" />
   </OrchestrationSteps>
```

The provided set of B2C custom policies in this repo were indeed initially created using the starter pack SocialAndLocalAccounts. If you were using the IEF Setup App as instructed in section **Be ready for B2C custom policies**, this app has deployed instead the starter pack SocialAndLocalAccountsWithMfa with a different *TrustFrameworkBase.xml* file. So, you need to modify these two files since the orchestration step numbers are different between the two starter pack base files. Otherwise, you will encounter errors when uploading these two policies files in the next section below.

6. Save all the .xml custom policy files.

Upload the B2C custom policies

Eventually, upload the above set of .xml custom policy files to your Azure AD B2C tenant, either via the Azure portal, the IefPolicies cmdlets or directly from Visual Studio Code, starting with *TrustFrameworkExtensionsVC.xml* first. Overwrite the custom policy if it already exists.

Please note that you already should have uploaded the *TrustFrameworkBase.xml* and *trustFrameworkExtensions.xml* in a previous step, see section **Be ready for B2C custom policies**.

Test the Microsoft Authenticator registration

At this stage, you can test your prior registration for Microsoft Authenticator and, if you've enabled support for redirecting to https://jwt.ms, you can get an ID token by running the following in a browser session.

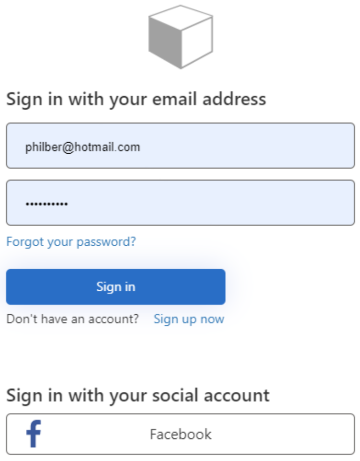
Proceed with the following steps:

1. Open a browser session and navigate to:

https://<*yourTenant*>.b2clogin.com/<*yourTenantId*>/B2C_1A_SIGNUP_SIGNIN/v2.0/authorize?client_id=<*yourClientId*>&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid%20profile&response_type=id_token&prompt=login

Replace <*yourTenant*> with the name of your tenant, and <*yourTenantId*> resp. <*yourClientId*> with your tenant ID resp. your client ID you took a note of. For example, in our illustration:

https://litware369b2c.b2clogin.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/B2C_1A_SIGNUP_SIGNIN/oauth2/v2.0/authorize?client_id=f83ece93-0e98-4379-8df3-248619483bd9&nonce=defaultNonce&redirect_uri=https%3A%2F%2Fjwt.ms&scope=openid%20profile&response_type=id_token&prompt=login



Sign in with your email address

philber@hotmail.com

••••••••••

Forgot your password?

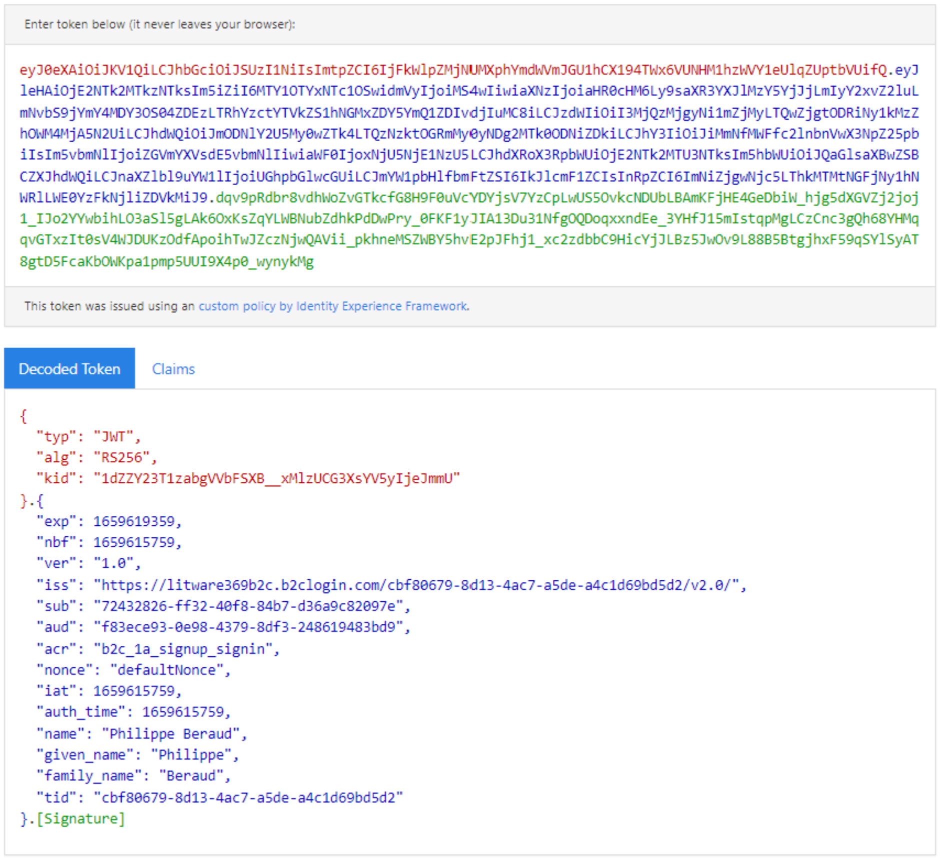
Sign in

Don't have an account? Sign up now

Sign in with your social account

Facebook

2. Click **Sign in** and follow the instructions to sign-in with the previously created test account.



Enter token below (it never leaves your browser):

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjFkWlpZMjNUMXphYmdWVmJGU1hCX194TWx6VUNHM1hzWVY1eUlqZUptbVUifQ.eyJ
leHAiOjE2NTk2MTkzNTksIm5iZiI6MTY1OTYxNTc1OSwidmVyIjoiMS4wIiwiaXNzIjoiaHR0cHM6Ly9saXR3YXJlMzY5YjJjLmIyY2xvZ2luL
mNvbS9jYmY4MDY3OS04ZDEzLTRhYzctYTVkZS1hNGMxZDY5YmQ1ZDIvdjIuMC8iLCJzdWIiOiI3MjQzMjgyNi1mZjMyLTQwZjgtODRiNy1kMzZ
hOWM4MjA5N2UiLCJhdWQiOiJmODNlY2U5My0wZTk4LTQzNzktOGRmMy0yNDg2MTk0ODNiZDkiLCJhY3IiOiJiMmNfMWFfc2lnbnVwX3NpZ25pb
iIsIm5vbmNlIjoiZGVmYXVsdE5vbmNlIiwiaWF0IjoxNjU5NjE1NzU5LCJhdXRoX3RpbWUiOjE2NTk2MTU3NTksIm5hbWUiOiJQaGlsaXBwZSB
CZXJhdWQiLCJnaXZlbl9uYW1lIjoiUGhpbGlwcGUiLCJmYW1pbHlfbmFtZSI6IkJlcmF1ZCIsInRpZCI6ImNiZjgwNjc5LThkMTMtNGFjNy1hN
WRlLWE0YzFkNjliZDVkMiJ9.dqv9pRdbr8vdhWoZvGTkcfG8H9F0uVcYDYjsV7YzCpLwUS5OvkcNDUbLBAmKFjHE4GeDbiW_hjg5dXGVZj2joj
1_IJo2YYwbihLO3aSl5gLAk6OxKsZqYLWBNubZdhkPdDwPry_0FKF1yJIA13Du31NfgOQDoqxxndEe_3YHfJ15mIstqpMgLCzCnc3gQh68YHMq
qvGTxzIt0sV4WJDUKzOdfApoihTwJZczNjwQAVii_pkhneMSZWBY5hvE2pJFhj1_xc2zdbbC9HicYjJLBz5JwOv9L88B5BtgjhxF59qSYlSyAT
8gtD5FcaKbOWKpa1pmp5UUI9X4p0_wynykMg
```

This token was issued using an custom policy by Identity Experience Framework.

Decoded Token Claims

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1659619359,
  "nbf": 1659615759,
  "ver": "1.0",
  "iss": "https://litware369b2c.b2clogin.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "72432826-ff32-40f8-84b7-d36a9c82097e",
  "aud": "f83ece93-0e98-4379-8df3-248619483bd9",
  "acr": "b2c_1a_signup_signin",
  "nonce": "defaultNonce",
  "iat": 1659615759,
  "auth_time": 1659615759,
  "name": "Philippe Beraud",
  "given_name": "Philippe",
  "family_name": "Beraud",
  "tid": "cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2"
}.[Signature]
```

To get the extra claims, you need to have profile as part of the scope.

Please note that Azure AD B2C has built-in support in the Azure portal for testing your B2C custom policies via the **Run user flow** functionality. You will leverage this capability in the next chapter.

With the B2C custom policies in place in your B2C tenant, let's now create the Custom credential you will be using for the B2C integration.

Create your credentials that uses your B2C tenant for issuance of VCs

Create a new Custom credential

See Tutorial - Issue Microsoft Entra Verified ID credentials from an application for more details.

To create a new custom (verifiable) credential with Microsoft Entra Verified ID, we will need to specify the *display* and *rules* definitions as already illustrated in the second part of this guide. As such, and as reminder:

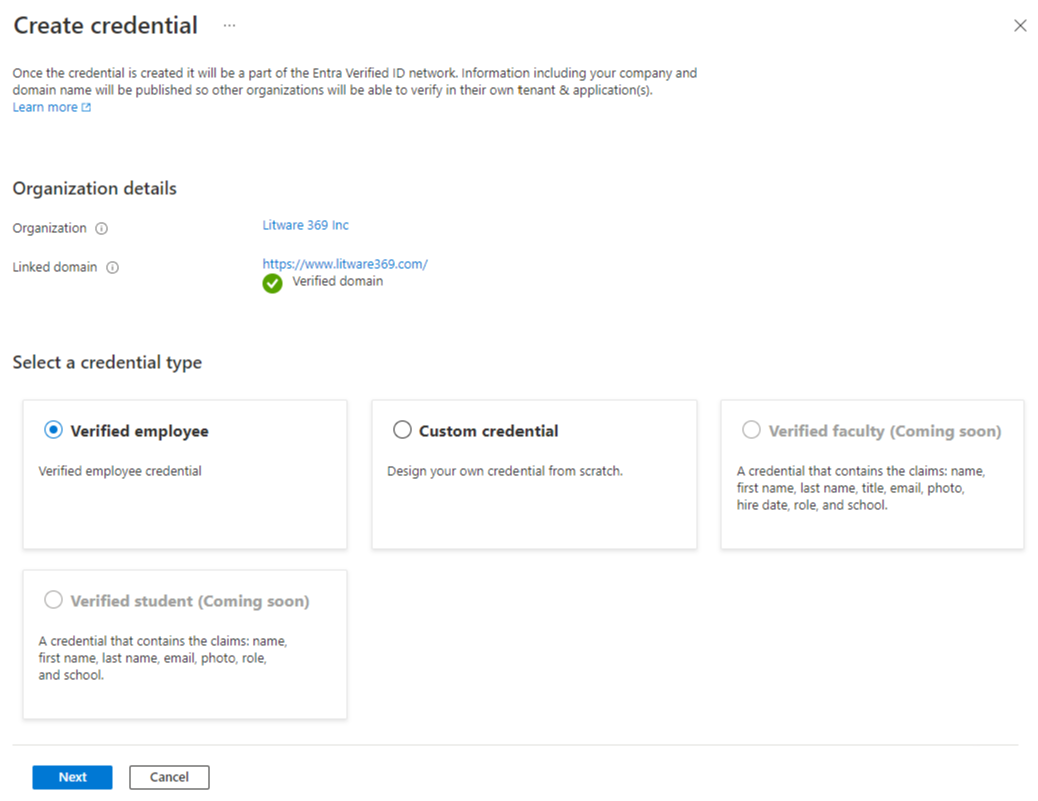
- The *display* definition controls the branding of the credential and styling of the claims.
- The *rules* definition describes important properties of the custom credential. In particular, it describes the claims that subjects (users) need to provide before such a custom credential is issued for them.

Proceed with the following steps:

1. Using the Azure portal, search for "*verified id*", and then click **Verified ID** to open the eponym blade.

   Please note that you can start using the **unified Microsoft Entra admin center** instead at https://entra.microsoft.com/. The admin center represents an integrated, easy-to-use approach to managing your entire identity infrastructure. Click **Verified ID**.

2. Click **Credentials**, and click **+ Add a credential**. A **Create credential** blade opens up.



Create credential

Once the credential is created it will be a part of the Entra Verified ID network. Information including your company and domain name will be published so other organizations will be able to verify in their own tenant & application(s).
Learn more

Organization details

Organization: Litware 369 Inc

Linked domain: https://www.litware369.com/ Verified domain

Select a credential type

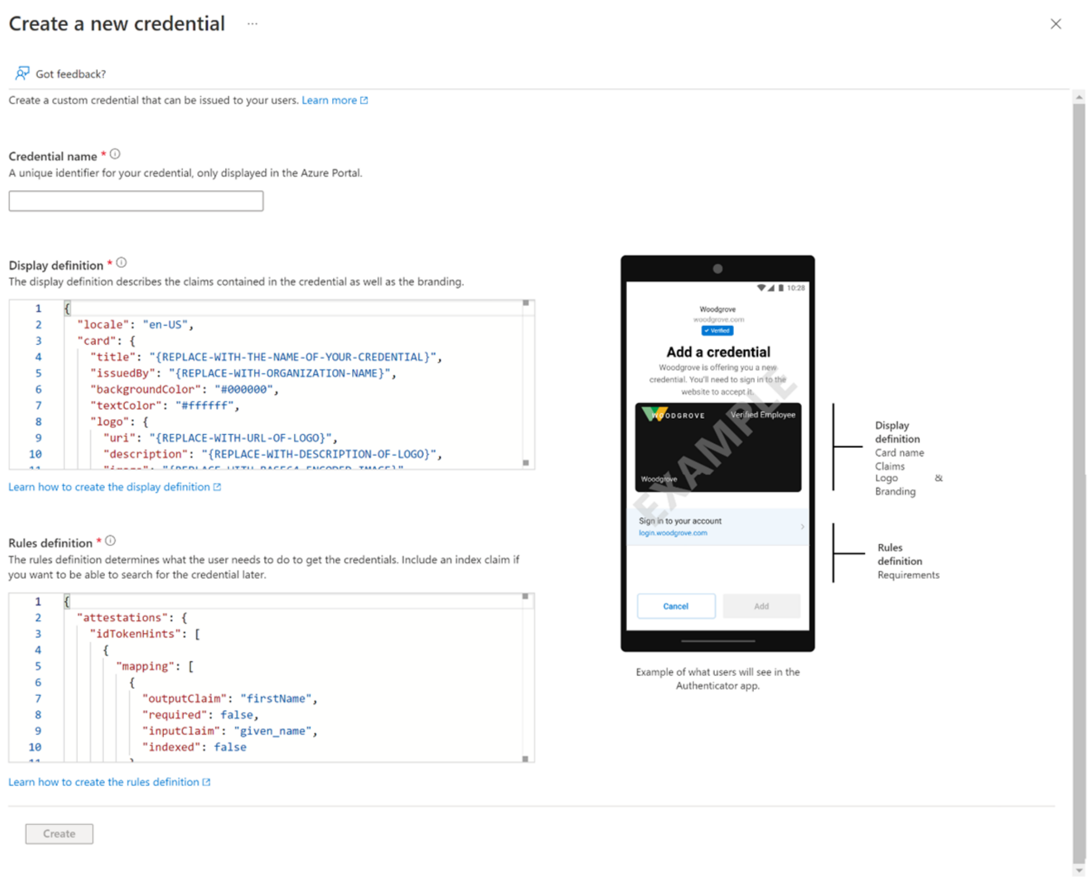
[✓ Verified employee] Verified employee credential

Custom credential: Design your own credential from scratch.

Verified faculty (Coming soon): A credential that contains the claims: name, first name, last name, title, email, photo, hire date, role, and school.

Verified student (Coming soon): A credential that contains the claims: name, first name, last name, email, photo, role, and school.

You get the option to launch two available quick starts, with more to come. Select **Custom Credential** and select **Next**. A **Create a new credential** blade opens up.



Create a new credential ···

Got feedback?

Create a custom credential that can be issued to your users. Learn more

Credential name * ⓘ
A unique identifier for your credential, only displayed in the Azure Portal.

Display definition * ⓘ
The display definition describes the claims contained in the credential as well as the branding.

```
{
  "locale": "en-US",
  "card": {
    "title": "{REPLACE-WITH-THE-NAME-OF-YOUR-CREDENTIAL}",
    "issuedBy": "{REPLACE-WITH-ORGANIZATION-NAME}",
    "backgroundColor": "#000000",
    "textColor": "#ffffff",
    "logo": {
      "uri": "{REPLACE-WITH-URL-OF-LOGO}",
      "description": "{REPLACE-WITH-DESCRIPTION-OF-LOGO}",
```

Learn how to create the display definition

Rules definition * ⓘ
The rules definition determines what the user needs to do to get the credentials. Include an index claim if you want to be able to search for the credential later.

```
{
  "attestations": {
    "idTokenHints": [
      {
        "mapping": [
          {
            "outputClaim": "firstName",
            "required": false,
            "inputClaim": "given_name",
            "indexed": false
```

Learn how to create the rules definition

Example of what users will see in the Authenticator app.

3. In **Name**, specify a name for your custom credential. For example, "CustomCredentialB2C". This name is used in the portal to identify your Custom (verifiable) credential. It's included as part of the verifiable credentials contract.

   Now let's move with the configuration of the above-described the display and *rules* definitions. You can use the content of the provided *CustomCredentialB2CDisplay.json* and *CustomCredentialB2CRules.json* files for these definitions We recommend to first update these files locally to reflect the specificities of your configuration, and then to copy and paste the updated contents in the related definitions textboxes.

4. In the **Display definition** textbox, copy the JSON content of the *CustomCredentialB2CDisplay.json* file and paste it to replace the provider skeleton.

```
{
  "locale": "en-US",
  "card": {
    "backgroundColor": "#000000",
    "description": "Use your verified credential to prove to anyone that you know all about verifiable
credentials.",
    "issuedBy": "Litware369 Inc.",
    "textColor": "#ffffff",
    "title": "Custom Credential B2C",
    "logo": {
      "description": "Litware 369 Inc. Logo",
      "uri": "https://litwarestorageaccount.blob.core.windows.net/images/litware369inc-logo.png"
    }
  },
  "consent": {
    "instructions": "Sign in with your account to get your card.",
    "title": "Do you want to get your Verified Credential?"
  },
  "claims": [
    {
      "claim": "vc.credentialSubject.userName",
      "label": "User name",
      "type": "String"
    },
    {
      "claim": "vc.credentialSubject.displayName",
      "label": "Display name",
      "type": "String"
    },
    {
      "claim": "vc.credentialSubject.firstName",
      "label": "First name",
      "type": "String"
    },
    {
      "claim": "vc.credentialSubject.lastName",
      "label": "Last name",
      "type": "String"
    },
    {
      "claim": "vc.credentialSubject.tid",
      "label": "tid",
      "type": "String"
    },
    {
      "claim": "vc.credentialSubject.oid",
      "label": "oid",
      "type": "String"
    }
  ]
}
```

5. In the **Rules definition**, copy the JSON content of the *CustomCredentialB2CRules.json* file and paste it to replace the provider skeleton. As you can see, the *rules* definition corresponds to an id_token flow where one will have to priorly authenticate against your Azure AB B2C tenant to have this custom credential be issued with their information.

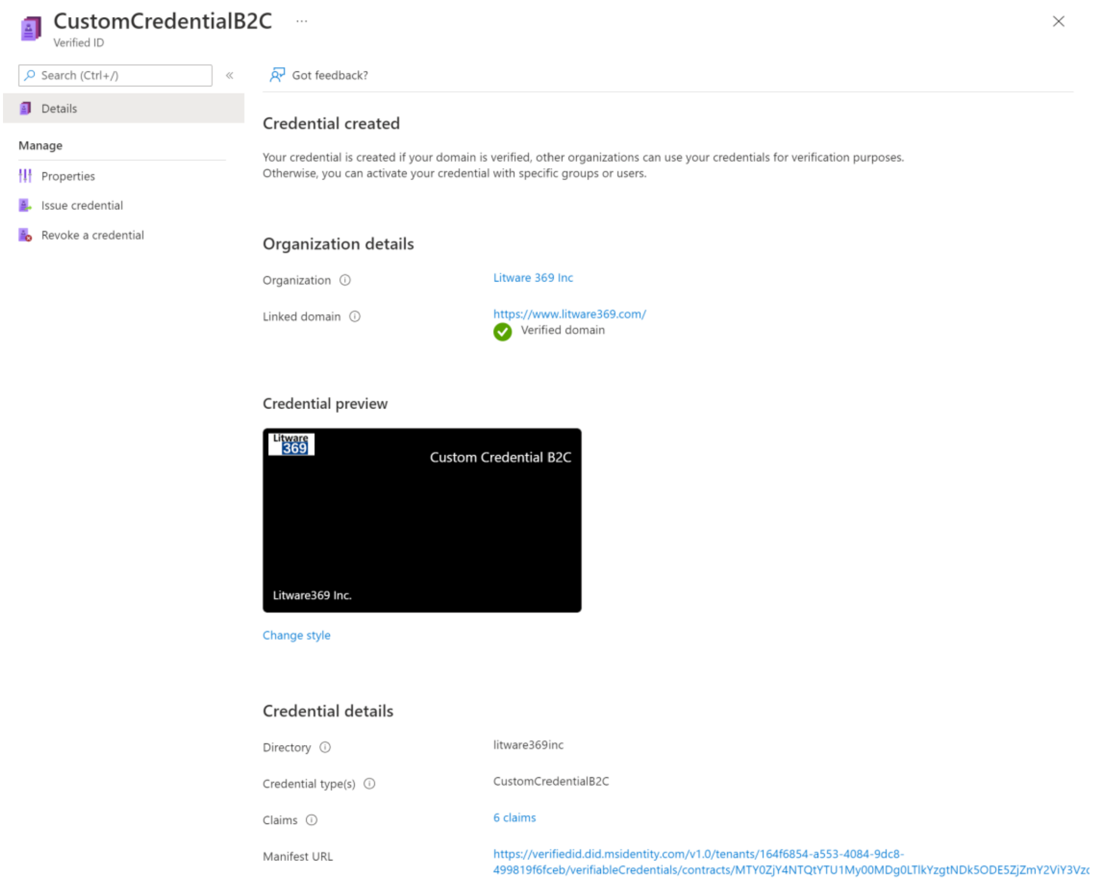
```
{
  "attestations": {
    "idTokens": [
      {
        "clientId": "<yourClientID>",
        "configuration":
"https://<yourTenant>.b2clogin.com/<yourTenant>.onmicrosoft.com/B2C_1A_SIGNUP_SIGNIN/v2.0/.well-known/openid-configuration",
        "redirectUri": "vcclient://openid",
        "scope": "openid profile email",
        "mapping": [
          {
            "outputClaim": "userName",
            "required": true,
            "inputClaim": "userName",
            "indexed": false
          },
          {
            "outputClaim": "displayName",
            "required": true,
            "inputClaim": "displayName",
            "indexed": false
          },
          {
            "outputClaim": "firstName",
            "required": true,
            "inputClaim": "firstName",
            "indexed": false
          },
          {
            "outputClaim": "lastName",
            "required": true,
            "inputClaim": "lastName",
            "indexed": true
          },
          {
            "outputClaim": "tid",
            "required": true,
            "inputClaim": "tid",
            "indexed": false
          },
          {
            "outputClaim": "oid",
            "required": true,
            "inputClaim": "oid",
            "indexed": false
          }
        ],
        "required": false
      }
    ]
  },
  "validityInterval": 2592000,
  "vc": {
    "type": [
      "CustomCredentialB2C"
    ]
  }
}
```

Replace `clientId`, `configuration`, and `vc type` by your own values depending on your configuration.

a. `clientId`: specify in lieu of <*youClientID*> the app (client) ID of the application you previously registered in your Azure AD B2C tenant for Microsoft Authenticator as per section **Register Microsoft Authenticator in Azure AD B2C**. For example, `f83ece93-0e98-4379-8df3-248619483bd9` in our illustration.
b. `Configuration:` ensure that the configuration endpoint reflects to the metadata of your deployed custom policies in Azure AD B2C as per previous section. More particularly, you will need to replace <*yourTenant*> by the name of your B2C tenant. For example, in our illustration:

https://litware369b2c.b2clogin.com/litware369b2c.onmicrosoft.com/B2C_1A_VC_SUSIQ/v2.0/.well-known/openid-configuration

6. Eventually click **Create**.


CustomCredentialB2C ...
Verified ID

Credential created

Your credential is created if your domain is verified, other organizations can use your credentials for verification purposes.
Otherwise, you can activate your credential with specific groups or users.

Organization details

Organization	Litware 369 Inc
Linked domain	https://www.litware369.com/ ✓ Verified domain

Credential preview

Credential details

Directory	litware369inc
Credential type(s)	CustomCredentialB2C
Claims	6 claims
Manifest URL	https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViY3Vz

Please note that you can update the above *display* and *rule* definition at anytime by selecting **Properties**. for the considered Custom credential for your BC2 users.

Configure the code sample application

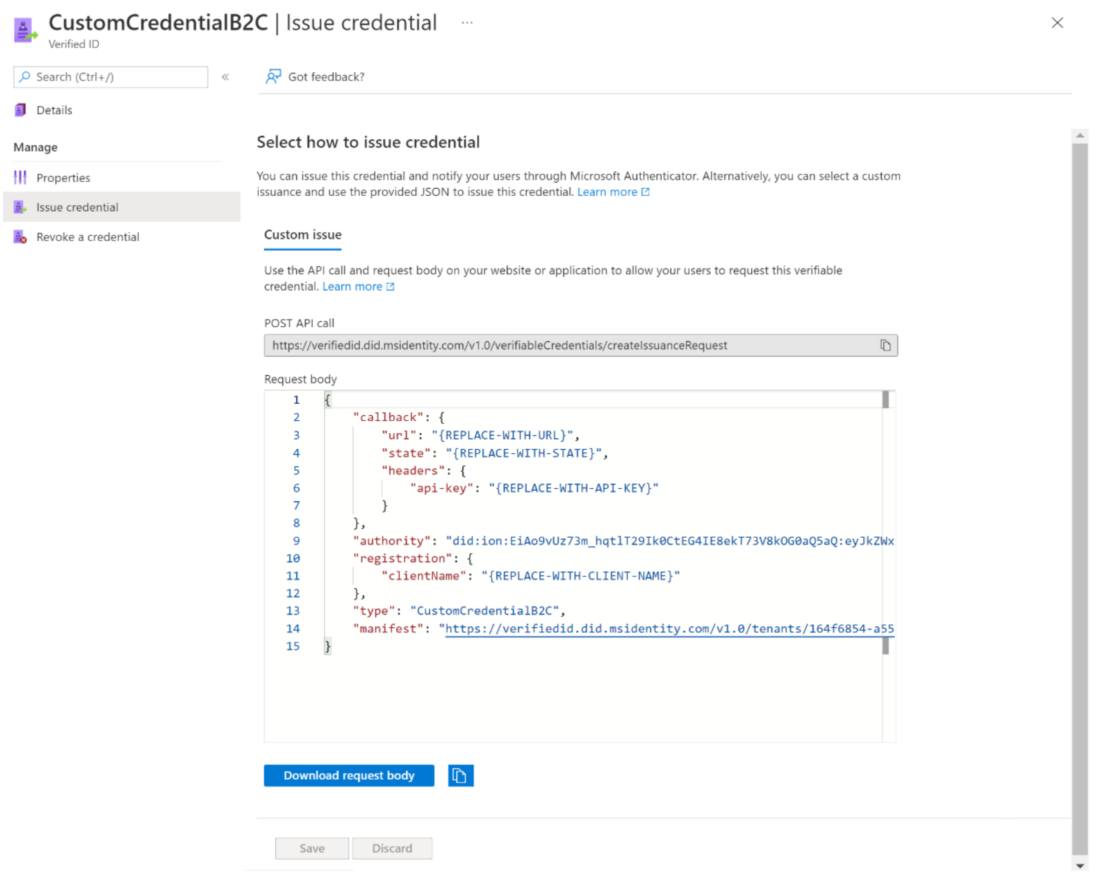
It's now time to configure the code sample application. Let's start by gathering the required information.

Gather the custom credential details

All you need here is your issuer DID for your Azure AD tenant, the credential type, and the manifest url to your credential. The easiest way to find these values for a managed credential is to view the credential in the portal.

Proceed with the following steps:

1. Still from the above **CustomCredentialB2C** blade, under **Manage**, select **Issue credential** and switch to **Custom issue**.


CustomCredentialB2C | Issue credential

Verified ID

Search (Ctrl+/)

Details

Manage

Properties

Issue credential

Revoke a credential

Got feedback?

Select how to issue credential

You can issue this credential and notify your users through Microsoft Authenticator. Alternatively, you can select a custom issuance and use the provided JSON to issue this credential. Learn more

Custom issue

Use the API call and request body on your website or application to allow your users to request this verifiable credential. Learn more

POST API call

https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/createIssuanceRequest

Request body

```
{
    "callback": {
        "url": "{REPLACE-WITH-URL}",
        "state": "{REPLACE-WITH-STATE}",
        "headers": {
            "api-key": "{REPLACE-WITH-API-KEY}"
        }
    },
    "authority": "did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx
    "registration": {
        "clientName": "{REPLACE-WITH-CLIENT-NAME}"
    },
    "type": "CustomCredentialB2C",
    "manifest": "https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a55
}
```

Download request body

Save Discard

These steps bring up a textbox with a skeleton JSON payload for the Request Service REST API:

```
{
    "callback": {
        "url": "{REPLACE-WITH-URL}",
        "state": "{REPLACE-WITH-STATE}",
        "headers": {
            "api-key": "{REPLACE-WITH-API-KEY}"
        }
    },
    "authority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYW
NlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1
NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6Qj
F1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0
aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbW
FpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERv
bWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdH
kuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVw
ZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOn
siZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVu
dCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
    "registration": {
        "clientName": "{REPLACE-WITH-CLIENT-NAME}"
    },
    "type": "VerifiableCredentialB2C",
    "manifest": "https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpYWJsZ
WNyZWRlbnRpYWxzYjJj/manifest"
}
```

2. In this payload, take a note of the following values that you will need to reflect in your code sample application's configuration files :
    a. `authority`: `authority`: this value corresponds to the issuer's DID (`did:ion..`). You will need to specify this value in the `VerifierAuthority` and `IssuerAuthority` properties of the *appsettings.json* file of the ASP.NET Core code sample application.
    b. `type`: the credential type is `CustomCredentialB2C` as specified above. You will also later need to ensure that this value corresponds to the one set in the `CredentialType` property of the *appsettings.json* of the code sample application in ASP.NET Core.

The `manifest` setting specifies the complete URL to the DID manifest used for both issuance and presentation is as follows:

https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpYWJsZWNyZWRlbnRpYWxzYjJj/manifest

It uses the new hostname verifiedid.did.msidentity.com introduced in July'22 that avoids to specify .eu. for an EU tenant, see What's new for Microsoft Entra Verified ID (preview).

You will need to specify this URL in the `DidManifest` property of the *appsettings.json* of the ASP.NET Core code sample application.

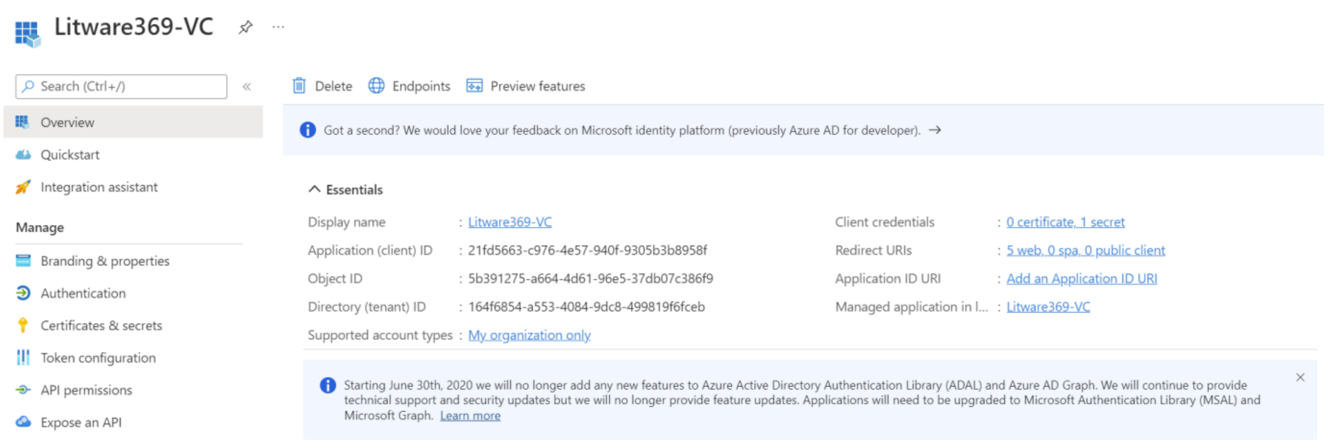
Gather environment details

Now that you have a new Custom credential in place for your (future) B2C users, you're going to gather some information about your environment.

Proceed with the following steps:

1. From the Azure portal, make sure you're using the directory that contains your Azure AD tenant. Select the **Directories + subscriptions** icon in the portal toolbar.
2. On the **Portal settings | Directories + subscriptions** page, find your Azure AD directory in the **Directory** name list, and then select **Switch**.
3. In the Azure portal, search for and select **Azure Active Directory**.
    a. In the left pane, click **App registrations**.
    a. Select the application you previously registered in the first part of this guide. This is the application that has the VC `permission VerifiableCredential.Create.All` and that has access to your VC Azure KeyVault.

       For example, Litware369-VC in our illustration. An eponym blade pops up.



Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Display name : Litware369-VC
Application (client) ID : 21fd5663-c976-4e57-940f-9305b3b8958f
Object ID : 5b391275-a664-4d61-96e5-37db07c386f9
Directory (tenant) ID : 164f6854-a553-4084-9dc8-499819f6fceb
Supported account types : My organization only
Client credentials : 0 certificate, 1 secret
Redirect URIs : 5 web, 0 spa, 0 public client
Application ID URI : Add an Application ID URI
Managed application in l... : Litware369-VC

Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

b. On the application overview page, copy and take a note of the **Application (client) ID** and **Directory (tenant) ID** values for later.

Update the code sample application app

Like the prior NodeJS project(s) in the first and second parts of this guide, this ASP.Net Core 3.1 project is also divided in 2 parts, one for issuing a VC and yet another one for verifying a VC.

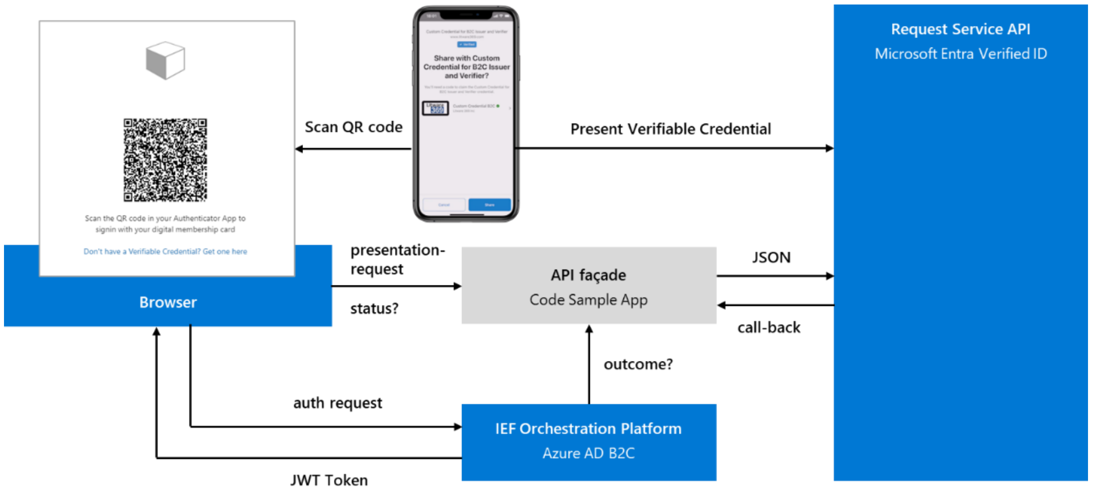
To later verify if your environment is completely working, you can use both parts to issue the above CustomCredentialB2C VC and verify that as well.

Issuance	Description
wwwroot/issuer.html	The basic webpage containing the JavaScript to call the APIs for issuance.
ApiIssuerController.cs	This is the controller which contains the API called from the webpage. It calls the Request Service REST API after getting an access token through the Microsoft Authentication Library (MSAL).
issuance_request_config.json	The sample payload send to the server to start issuing a VC.
Verification	**Description**
wwwroot/verifier.html	The website acting as the verifier of the verifiable credential.
ApiVerifierController.cs	This is the controller which contains the API called from the webpage. It calls the Request Service REST API after getting an access token through MSAL and helps verifying the presented verifiable credential.
presentation_request_config.json	The sample payload send to the server to start issuing a vc.

The code sample application is designed to work together with Azure AD B2C in order to have Verifiable Credentials), i.e., Custom credentials, for B2C accounts.

Even though you could use it for testing generic contracts, such as the prior VerifiedEmployee in the first part or the CustomCredentialTest in the second part of this guide, this application is intended to be used for the integration with the Azure AD B2C.

You will later use it as a REST API façade when integrating with the Azure AD B2C sample custom policies that come along with this third part of the guide, and the CustomCredentialB2C VC you created with the ID token flow. As such, it exposes in particular two REST API endpoints suffixed by `-b2c` specifically meant for the integration with Azure AD B2C.


Scan the QR code in your Authenticator App to signin with your digital membership card
Don't have a Verifiable Credential? Get one here
Browser
Scan QR code
Share with Custom Credential for B2C Issuer and Verifier?
Present Verifiable Credential
Request Service API
Microsoft Entra Verified ID
presentation-request
status?
API façade
Code Sample App
JSON
call-back
outcome?
auth request
IEF Orchestration Platform
Azure AD B2C
JWT Token

To this extend, this application indeed uses two general areas to form the solution: i) one is the custom html and self asserted pages in Azure AD B2C, and ii) the other is the REST API technical profile in Azure AD B2C:

1. The custom html makes it possible to do ajax calls in JavaScript to generate the QR code to issue or verify a VC.
2. The REST API technical profile makes it possible to integrate calls to the code sample application that exposes a REST API facade so that the IEF orchestration platform of Azure AD B2C can send claims and query presentation status. For this to work, you need this application as a backend for Azure AD B2C to talk to.

See section **Under the hood** below for more details.

As such, the code sample application all its configuration in the *appsettings.json* file, and you need to update it before you run the application to expose the REST API façade.

In the *appsettings.json* file, there are a few settings, but the ones listed below needs your attention. First, `TenantId`, `ClientId` and `ClientSecret` are used to acquire an access_token so you can authorize the Request Service REST API of your Microsoft Entra Verified ID service to your Azure Key Vault. This is extensively covered in the first part of the guide.

With that, proceed with the following steps:

1. Launch a code editor of your choice. For example Visual Studio Code in our illustration.
2. Click **File**, and select **Open Folder**. Select the directory *part-3* under the folder where you previously cloned the code sample repo, e.g., *C:\>entra-verifiedid-tour\part-3* in our illustration. Click **Select the directory**.
3. At the root of the folder, select the *appsettings.json* file.

```
[…omitted for brievety…]
"AppSettings": {
   "ApiEndpoint": "https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/",
   "Authority": "https://login.microsoftonline.com/{0}",
   "scope": "3db474b9-6a0c-4840-96ac-1fceb342124f/.default",
   "TenantId": "",
   "ClientId": "",
   "ClientSecret": "",
   "ApiKey": "",
   "CookieKey": "state",
   "CookieExpiresInSeconds": 7200,
    "CacheExpiresInSeconds": 300,
    "client_name": "Custom Credential for B2C Issuer and Verifier",
    "Purpose": "To prove your identity",
    "VerifierAuthority": "did:ion:...your DID...",
    "IssuerAuthority": "did:ion:...your DID...",
    "CredentialType": "<yourVCName>",
    "DidManifest":
"https://verifiedid.did.msidentity.com/v1.0/<yourTenantId>/verifiableCredential/contracts/<yourVCName>",
    "IssuancePinCodeLength": 0,
    "B2C1ARestApiKey": "your-b2c-app-key
}
[…omitted for brievety…]
```

4. Update the following properties with the information as the following:

   a. `ApiEndpoint`: specify the Request Service API endpoint.
   b. `TenantId`: specify the ID of the technical Azure AD tenant that you have setup Microsoft Entra Verified ID in as per first and second parts of this guide. See section Gather environment details above.

      This isn't NOT the newly created Azure AD B2C tenant.

   c. `ClientId`: specify your application (client) ID corresponding to the application you priorly registered for the VC in the first part of the guide. See section Gather environment details above.
   d. `ClientSecret`: enter the client secret corresponding to this application. See section Gather environment details in the first part of this guide.
   e. `VerifierAuthority`: specify the DID for your Microsoft Entra Verified ID service as per section Gather the custom credential details above. You can find in your VC blade in the Azure portal or in the unified Microsoft Entra portal.
   f. `IssuerAuthority`: specify again the DID for your Microsoft Entra Verified ID service.
   g. `CredentialType`: specify whatever you have as type in the Rules definition of your Custom credential. The default is CustomCredentialB2C to align with the provided definition files in the repo.
   h. `DidManifest`: specify the complete URL to the DID manifest as per section Gather the custom credential details above. It is used to set the attribute manifest and it is used for both issuance and presentation.
   i. `IssuancePinCodeLength`: if you want your issuance process to use the pin code method, specify how many digits the pin code should have. A value of zero will not use the pin code method.
   j. `B2C1ARestApiKey`: specify the value of your key as per section Create the REST API key in the portal.

```
[…omitted for brievety…]
"AppSettings": {
    "ApiEndpoint": "https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/",
    "Authority": "https://login.microsoftonline.com/{0}",
    "scope": "3db474b9-6a0c-4840-96ac-1fceb342124f/.default",
    "TenantId": "164f6854-a553-4084-9dc8-499819f6fceb",
    "ClientId": "21fd5663-c976-4e57-940f-9305b3b8958f",
    "ClientSecret": "vGa7Q~crZdB8FVk9q6Rn1GxNSTvJuLd4OlR4F",
    "ApiKey": "",
    "CookieKey": "state",
    "CookieExpiresInSeconds": 7200,
    "CacheExpiresInSeconds": 300,
    "client_name": "Custom Credential for B2C Issuer and Verifier",
    "Purpose": "To prove your identity",
    "VerifierAuthority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYW
NlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1
NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6Qj
F1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0
aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbW
FpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERv
bWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdH
kuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVw
ZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOn
siZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVu
dCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
    "IssuerAuthority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYW
NlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1
NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6Qj
F1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0
aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbW
FpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERv
bWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdH
kuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVw
ZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOn
siZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVu
dCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
    "CredentialType": "CustomCredentialB2C",
    "DidManifest": "https://verifiedid.did.msidentity.com/v1.0/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredential/contracts/CustomCredentialB2C",
    "IssuancePinCodeLength": 0,
    "B2C1ARestApiKey": "e09f4443-b08f-4ed2-8717-7a202b990b40"
}
[…omitted for brievety…]
```

5. Save the *appsettings.json* file by pressing CTRL+S.

   The Request Service REST API is called with special payloads for issuing verifiable credentials. The sample payload files are modified by code by copying the correct values from the *appsettings.json* file.

   f you want to modify the *payloads issuance_request_config.json* and *presentation_request_config.json* files yourself, make sure you comment out the code overwriting the values in the *VerifierController.cs* and *ApiIssuerController.cs* files. The code overwrites the `Authority`, `Manifest` and `trustedIssuers` values. The callback URI is modified in code to match your hostname.

   For issuance you don't need to change anything, for verifying make sure you follow the instructions in this guide and copy paste the correct payload to the *presentation_request_config.json*.

   Make sure you copy the `TenantId`, `ClientId`, and `ClientSecret` you copied when creating the app registration to the *appsettings.json* as well.

The required configuration file is now updated.

Run the code sample application locally

Let's first reflect your own environment configuration in the provided code sample. This code sample demonstrates how to use Microsoft Entra Verified ID to issue here and later consume Verifiable Credentials (VCs).

You can either run the code sample standalone or with Docker.

Run the code sample application standalone

To run the code sample standalone, perform the following steps:

From VS code environment, open a new terminal, and run the following command:

1. From the terminal window in Visual Studio Code, go to the directory *part-3* under the folder where you cloned the source code of the code sample application. For example, *c:\entra-verifiedid-tour\part-3* in our illustration.
2. Run the following command:

```
PS C:\> dotnet build "AspNetCoreVerifiableCredentialsB2C.csproj" -c Debug -o .\bin\Debug\netcoreapp3.1
```

   The above command will build the code sample application with all dependencies that the code needs (it can takes few minutes).

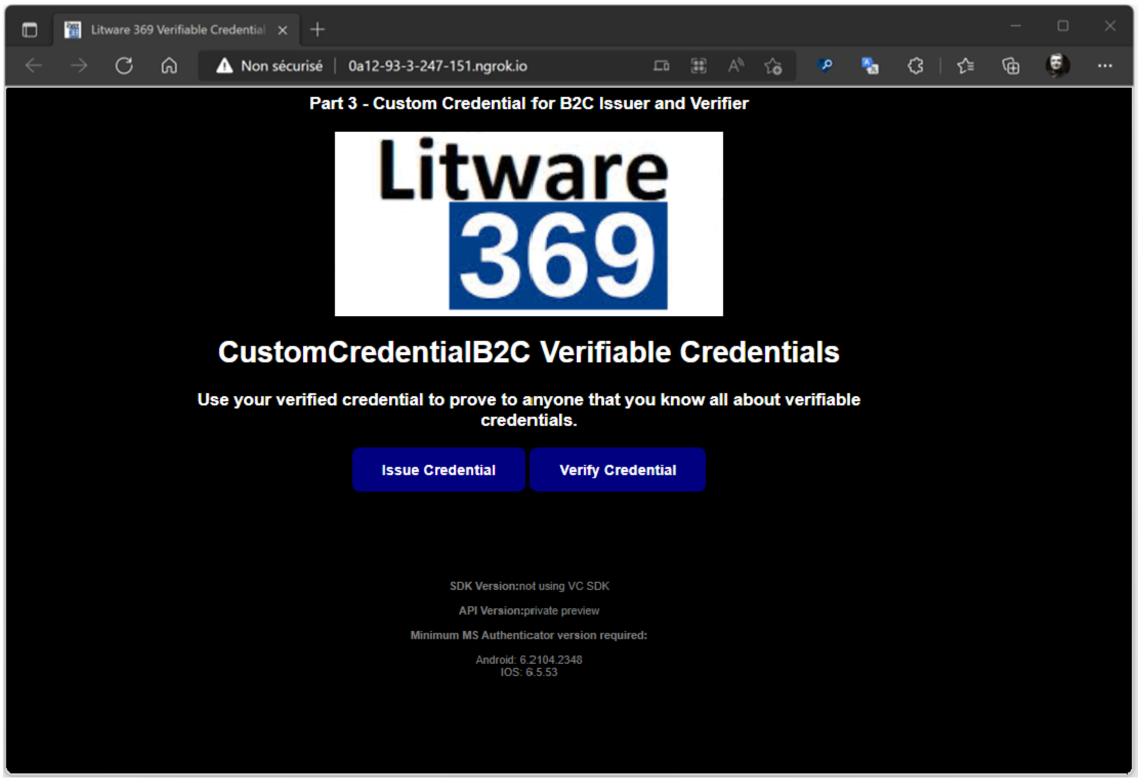
3. Launch the code sample application with the following command:

```
PS C:\> dotnet run
```

4. Open the HTTPS URL priorly generated by ngrok in a browser session. For example, in our illustration:

   https://0a12-93-3-247-151.ngrok.io/

   You should see the following screenshot:


Litware 369 Verifiable Credential
Non sécurisé | 0a12-93-3-247-151.ngrok.io
Litware 369
Issue Credential
Verify Credential

Part 3 - Custom Credential for B2C Issuer and Verifier

CustomCredentialB2C Verifiable Credentials

Use your verified credential to prove to anyone that you know all about verifiable credentials.

SDK Version:not using VC SDK

API Version:private preview

Minimum MS Authenticator version required:

Android: 6.2104.2348
IOS: 6.5.53

Run the code sample application with Docker (Optional)

To optionally run the code sample application in a Docker container, you assume at this stage that the prerequisites are fulfilled with both Docker Desktop for Windows and the VS Code Docker extension being installed on your local development machine. Same is true for ASP.NET Core that is also required on your computer and that should be already installed.

With that, let's start by adding the Docker to the code sample application project.

Add Docker files to the project

Proceed with the following steps:

1. If not already done, launch Visual Studio Code.
2. Click **File**, and select **Open Folder**. Select the directory *part-3* under the folder where you previously cloned the code sample repo, e.g., *C:\>entra-verifiedid-tour\part-3* in our illustration. Click **Select the directory** to open the code sample application.
3. Now open the Command Palette (Ctrl+Shift+P) and use **Docker: Add Docker Files to Workspace...** command.
4. Select **dotnet** when prompted for the application platform.
5. Select **package.json**.
6. Select either **Yes** or **No** when prompted to include Docker Compose files. Compose is typically used when running multiple containers at once.
7. Enter "*5002*" when prompted for the application port.

The Docker extension for Visual Studio code helps you author Docker files by using IntelliSense to provide auto-completions and contextual help. It lists all available Dockerfile instructions and describes the syntax.

Build the service image

Proceed with the following steps

1. Now open the Command Palette (Ctrl+Shift+P) and use the **Docker Images: Build Image...** command. You can instead run the following command:

```
PS C:\> docker build -t aspnetcoreverifiablecredentialsb2cdotnet:v1.0 .
```

2. Open the Docker Explorer and verify that the new image `aspnetcoreverifiablecredentialsb2cdotnet:v1.0` is visible in the *Images* tree.

Run the service container

Proceed with the following steps:

1. Right-click on the image built in the previous section and select **Run** or **Run Interactive**. The container should start and you should be able to see it in the Docker Containers tree. You instead run the following command:

```
PS C:\> docker run --rm -it -p 5002:80 aspnetcoreverifiablecredentialsb2cdotnet:v1.0
```

2. Open a browser session and navigate to https://0a12-93-3-247-151.ngrok.io. You should see the same page as the previous one.

Test the code sample application against the IEF orchestration platform

Once you have the code sample application up and running, and you can issue and verify VC, you can verify that it also will respond as the API façade that the IEF orchestration platform of Azure AD B2C will call as part the sequence of orchestration steps, as per executed custom policy.

To do this, you need the id (guid) that is unique for the browser session. You can either find it in the trace output in the console window of the running sample app, or you can find it in the browsers developer console (F12 in Edge/Chrome, then Console) after you have completed a verification in the browser.

To test the API façade exposed by adapted code sample application, proceed with the following steps:

1. Open a PowerShell 7 window.
2. Run the following command:

```
PS C:\> $id="<yourID>"
PS C:\> $ngrokUrl="<yourNgrokUrl>"
PS C:\> $response = Invoke-WebRequest -Uri "$ngrokUrl/api/verifier/presentation-response-b2c" `
             -Method Post -Body (@{id=$id;}|ConvertTo-json) -ContentType "application/json"
PS C:\> $response.Content | ConvertFrom-json
```

Replace the placeholder value in brackets with the above value. For example, in our illustration:

```
PS C:\> $id="2fb8dce0-9d69-481a-8a2b-f64b684a8aae"
PS C:\> $ngrokUrl="https://0a12-93-3-247-151.ngrok.io"
PS C:\> $response = Invoke-WebRequest -Uri "$ngrokUrl/api/verifier/presentation-response-b2c" `
             -Method Post -Body (@{id=$id;}|ConvertTo-json) -ContentType "application/json"
PS C:\> $response.Content | ConvertFrom-json
```

At this stage, your code sample application is ready to build and issue your Custom credential for B2C, once called from the IEF orchestration platform.

So, it's now high time to see how to issue and verify your own customers' credentials as part of your user journeys in Azure AD B2C. This is the purpose of the next chapter.

Leverage the customers' credentials for your user journeys

You are now ready to issue and verify Verifiable Credentials as part of your customers' user journeys in Azure AD B2C, i.e.:

- **Issue a VC during signup of new users in your Azure AD B2C tenant.**
- **Sign-in to B2C with a VC by scanning a QR code.**

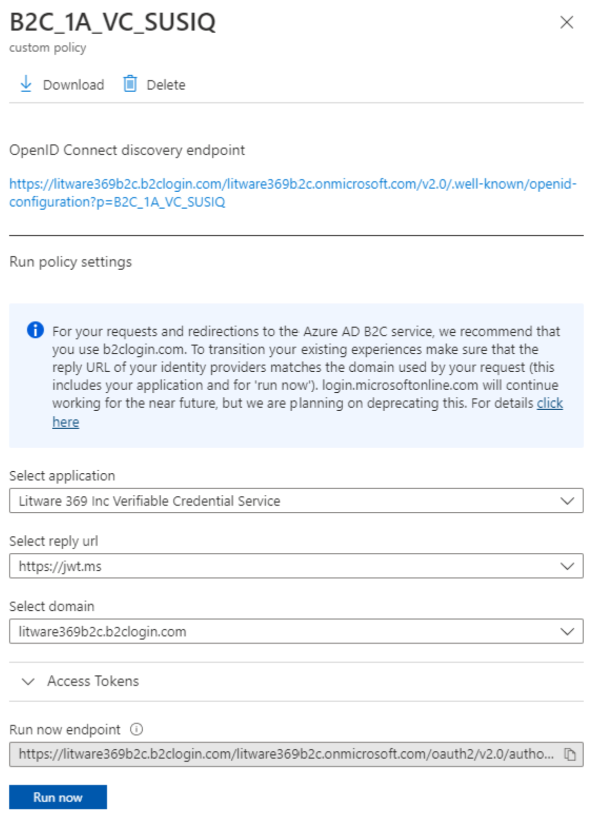
Test the custom policies

Issue a VC during signup

During the signup flow you will issue this user a VC by scanning the QR code after the signup. For that purpose, you will need to run the *SignupOrSigninVCQ.xml* (a.k.a., B2C_1A_VC_susiq) custom policy.

Proceed with the following steps:

1. Open a browser session, navigate to the Azure portal at https://portal.azure.com, and sign in with an account with admin privileges in your Azure AD B2C tenant - The account that was used to create the tenant has these by default -.
2. Make sure you're using the directory that contains your Azure AD B2C tenant. If needed:
    a. Select the **Directories + subscriptions** icon in the portal toolbar.
    b. On the **Portal settings | Directories + subscriptions** page, find your Azure AD B2C directory in the **Directory** name list, and then select **Switch**.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Under **Policies**, select **Identity Experience Framework**.
5. Under **Custom policies**, select the B2C_1A_VC_SUSIQ custom policy. An eponym blade opens up.
    a. In **Select application**, select the application you registered for Microsoft Authenticator. For example, **Litware 369 Inc Verifiable Credential Service** in our illustration.
    b. In **Select reply url**, select **https://jwt.ms**.
    c. In **Select domain**, leave the selected domain. For example, **litware369b2c.onmicrosoft.com** in our illustration.


B2C_1A_VC_SUSIQ
custom policy

OpenID Connect discovery endpoint

https://litware369b2c.b2clogin.com/litware369b2c.onmicrosoft.com/v2.0/.well-known/openid-configuration?p=B2C_1A_VC_SUSIQ

Run policy settings

For your requests and redirections to the Azure AD B2C service, we recommend that you use b2clogin.com. To transition your existing experiences make sure that the reply URL of your identity providers matches the domain used by your request (this includes your application and for 'run now'). login.microsoftonline.com will continue working for the near future, but we are planning on deprecating this. For details click here

Select application: Litware 369 Inc Verifiable Credential Service

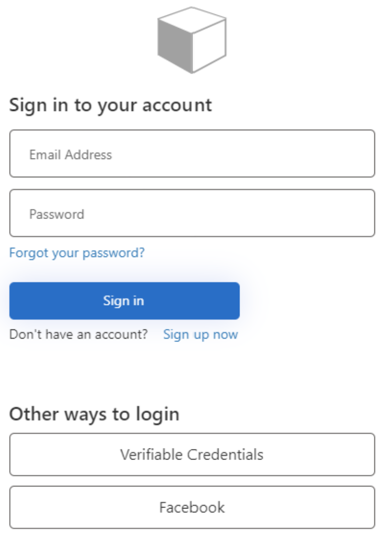
Select reply url: https://jwt.ms

Select domain: litware369b2c.b2clogin.com

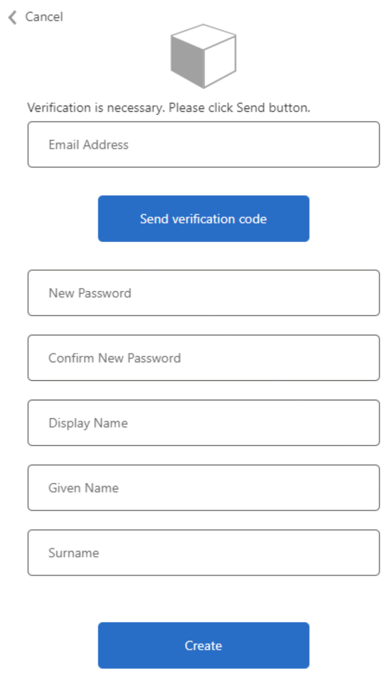
Access Tokens

Run now endpoint: https://litware369b2c.b2clogin.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/autho...

6. Click **Run now**. A new tab opens up in your browser session.


Sign in to your account

7. Signup a new user. Click **Sign up now** under the **Sign in** button.



Cancel

Verification is necessary. Please click Send button.

Email Address

Send verification code

New Password

Confirm New Password

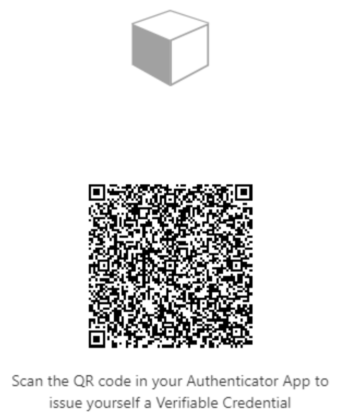
Display Name

Given Name

Surname

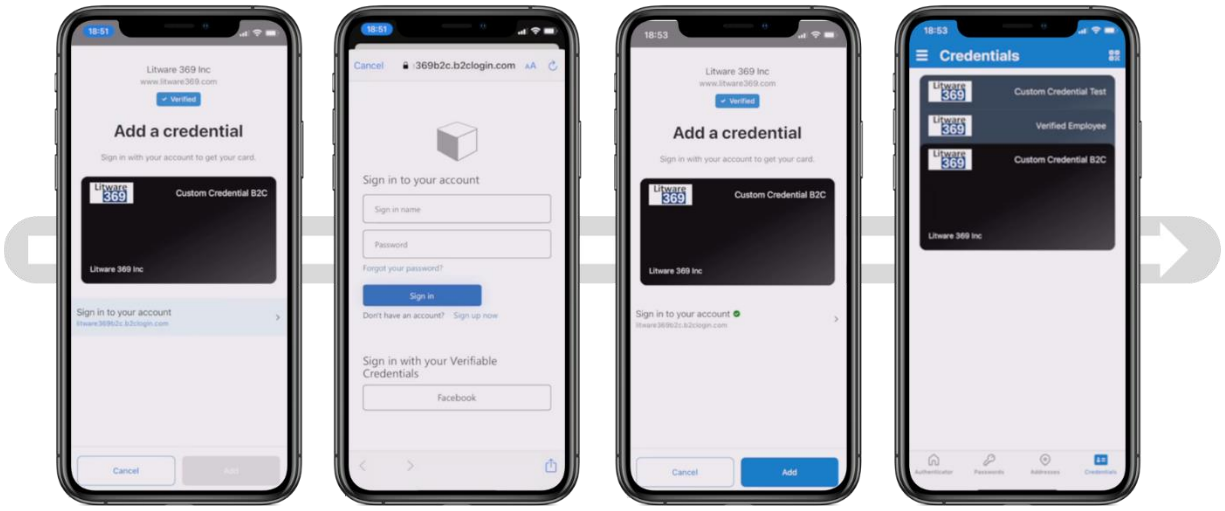
Create

8. After you have validated the email, set the password, etc, and created a new user, the final step in the user journey aims at issuing the new user with a VC.

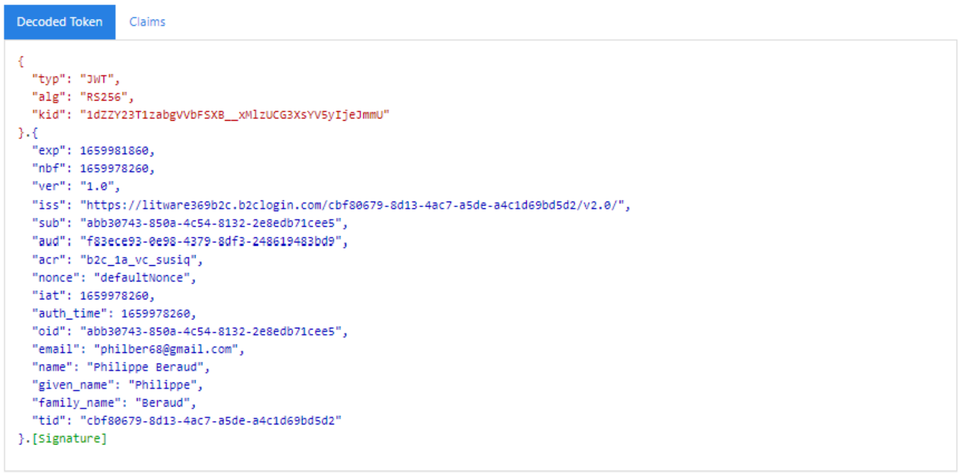


Scan the QR code in your Authenticator App to issue yourself a Verifiable Credential

Scan the QR code with Microsoft Authenticator to initiate the issuance.



The journey ends up by loading the received id_token into https://jwt.ms and decoding it.

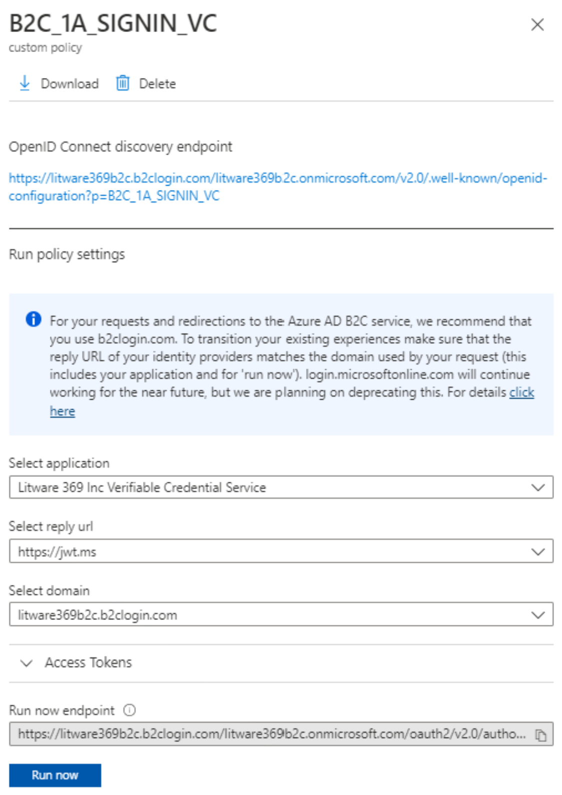

Decoded Token Claims

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1659981860,
  "nbf": 1659978260,
  "ver": "1.0",
  "iss": "https://litware369b2c.b2clogin.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "abb30743-850a-4c54-8132-2e8edb71cee5",
  "aud": "f83ece93-0e98-4379-8df3-248619483bd9",
  "acr": "b2c_1a_vc_susiq",
  "nonce": "defaultNonce",
  "iat": 1659978260,
  "auth_time": 1659978260,
  "oid": "abb30743-850a-4c54-8132-2e8edb71cee5",
  "email": "philber68@gmail.com",
  "name": "Philippe Beraud",
  "given_name": "Philippe",
  "family_name": "Beraud",
  "tid": "cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2"
}.[Signature]
```

Sign-in to B2C with a VC

Proceed with the following steps :

1. Now run either the B2C_1A_VC_SUSIQ custom policy, or the B2C_1A_SIGNIN_VC custom policy instead. For example, under **Custom Policies**, select the B2C_1A_SIGNIN_VC custom policy. An eponym blade opens up.
    a. In **Select application**, select the application you registered for Microsoft Authenticator. For example, **Litware 369 Inc Verifiable Credential Service** in our illustration.
    b. In **Select reply url**, select **https://jwt.ms**.
    c. In **Select domain**, leave the selected domain. For example, **litware369b2c.onmicrosoft.com** in our illustration.


B2C_1A_SIGNIN_VC

custom policy

OpenID Connect discovery endpoint

https://litware369b2c.b2clogin.com/litware369b2c.onmicrosoft.com/v2.0/.well-known/openid-configuration?p=B2C_1A_SIGNIN_VC

Run policy settings

For your requests and redirections to the Azure AD B2C service, we recommend that you use b2clogin.com. To transition your existing experiences make sure that the reply URL of your identity providers matches the domain used by your request (this includes your application and for 'run now'). login.microsoftonline.com will continue working for the near future, but we are planning on deprecating this. For details click here

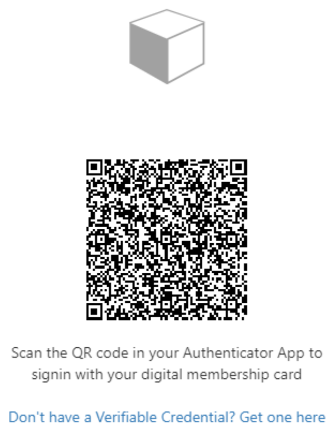
Select application: Litware 369 Inc Verifiable Credential Service

Select reply url: https://jwt.ms

Select domain: litware369b2c.b2clogin.com

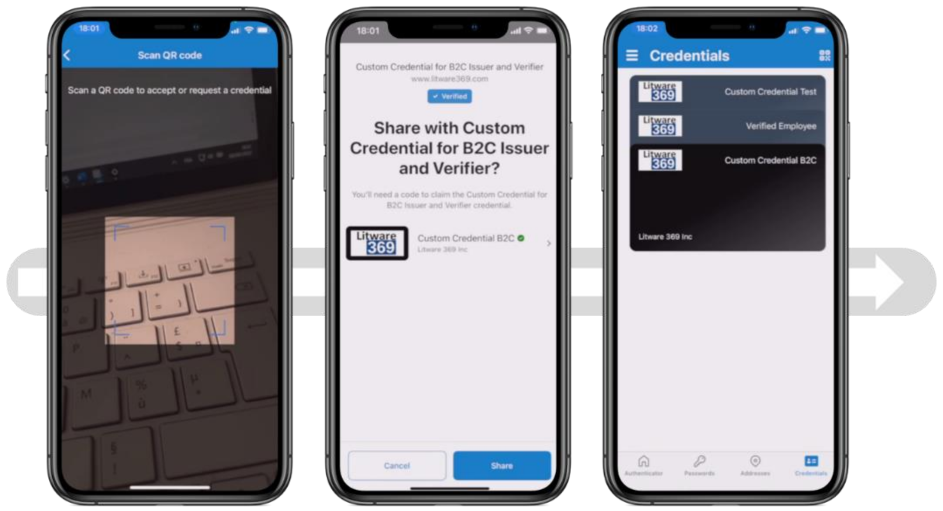
Run now endpoint: https://litware369b2c.b2clogin.com/litware369b2c.onmicrosoft.com/oauth2/v2.0/autho...

2. Click **Run now**. A new tab opens up in your browser session.

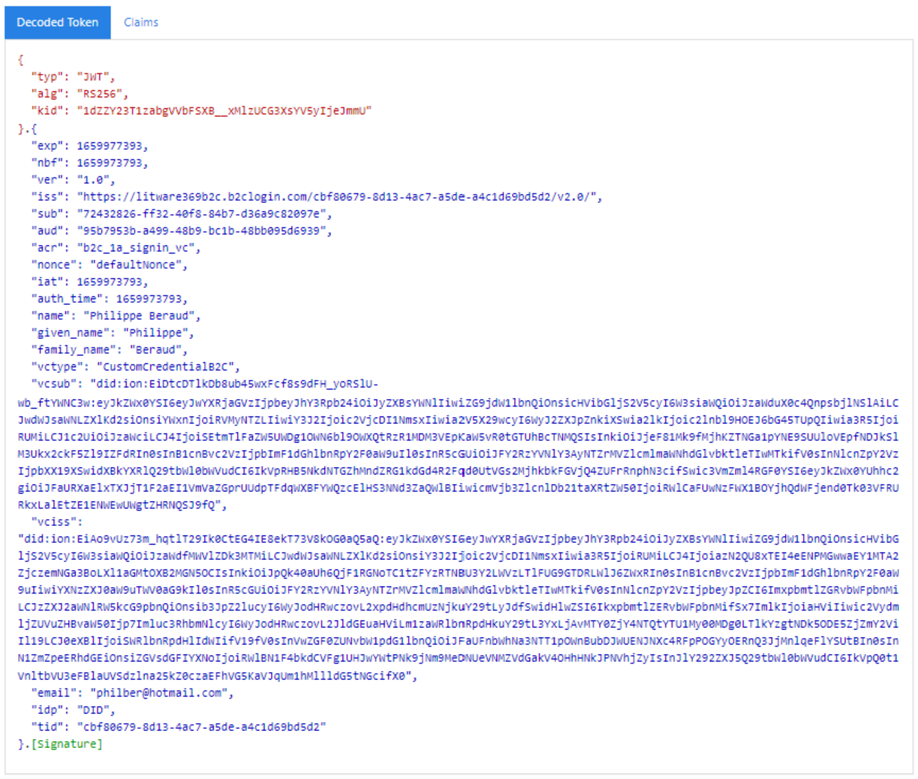

Scan the QR code in your Authenticator App to signin with your digital membership card

Don't have a Verifiable Credential? Get one here

3. Scan the QR code with Microsoft Authenticator, and click **Share** to signin to your B2C account using your CustomCredentialB2C VC.


Scan QR code
Scan a QR code to accept or request a credential
Custom Credential for B2C Issuer and Verifier
Verified
Share with Custom Credential for B2C Issuer and Verifier?
Custom Credential B2C
Cancel
Share
Credentials
Custom Credential Test
Verified Employee
Custom Credential B2C
Litware 369 Inc

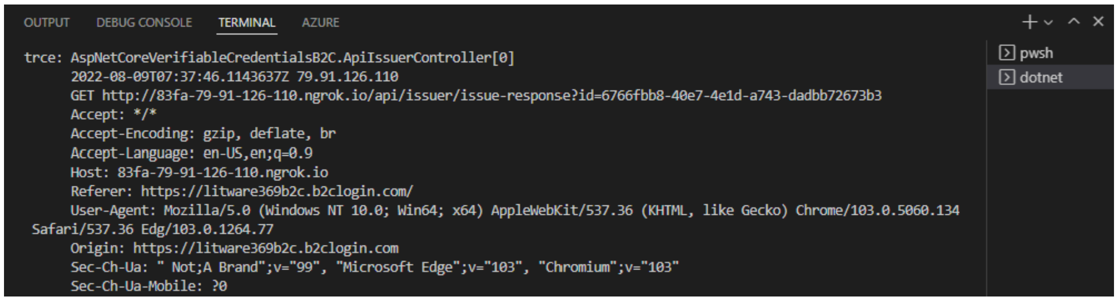
The journey similarly ends up by loading the received id_token into https://jwt.ms and decoding it.



```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1659977393,
  "nbf": 1659973793,
  "ver": "1.0",
  "iss": "https://litware369b2c.b2clogin.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "72432826-ff32-40f8-84b7-d36a9c82097e",
  "aud": "95b7953b-a499-48b9-bc1b-48bb095d6939",
  "acr": "b2c_1a_signin_vc",
  "nonce": "defaultNonce",
  "iat": 1659973793,
  "auth_time": 1659973793,
  "name": "Philippe Beraud",
  "given_name": "Philippe",
  "family_name": "Beraud",
  "vctype": "CustomCredentialB2C",
  "vcsub": "did:ion:EiDtcDTlkDb8ub45wxFcf8s9dFH_yoRSlU-
wb_ftYWNC3w:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWduX0c4QnpsbjlNSlAiLC
JwdWJsaWNLZXlKd2siOnsiYWxnIjoiRVMyNTZLIiwiY3J2Ijoic2VjcDI1NmsxIiwia2V5X29wcyI6WyJ2ZXJpZnkiXSwia2lkIjoic2lnbl9HOEJ6bG45TUpQIiwia3R5Ijoi
RUMiLCJ1c2UiOiJzaWciLCJ4IjoiSEtmTlFaZW5UWDg1OWN6bl9OWXQtRzR1MDM3VEpKaW5vR0tGTUhBcTNMQSIsInkiOiJjeF81Mk9fMjhKZTNGa1pYNE9SUUloVEpfNDJkSl
M3Ukx2ckF5Zl9IZFdRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2Vz
IjpbXX19XSwidXBkYXRlQ29tbWl0bWVudCI6IkVpRHB5NkdNTGZhMndZRG1kdGd4R2FqdUtVGs2MjhkbkFGVjQ4ZUFrRnphN3cifSwic3VmZml4RGF0YSI6eyJkZWx0YUhhc2
giOiJFaURXaElxTXJjT1F2aEI1VmVaZGprUUdpTFdqWXBFYWQzcElHS3NNd3ZaQWlBIiwicmVjb3ZlcnlDb21taXRtZW50IjoiRWlCaFUwNzFWX1BOYjhQdWFjend0Tk03VFRU
RkxLaletZE1ENWEwUWgtZHRNQSJ9fQ",
  "vciss":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG9jdW1lbnQiOnsicHVibG
ljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2
ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW
9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMi
LCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2Vydm
ljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2Vi
Il19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sIn
N1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1
VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMlllldG5tNGcifX0",
  "email": "philber@hotmail.com",
  "idp": "DID",
  "tid": "cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2"
}.[Signature]
```

LogLevel Trace

If you set the `LogLevel` to Trace in the *appsettings.json* file, then the ASP.NET Core code sample application will output all HTTP requests in the Visual Studio Code terminal, which will make it convenient for you to study the interaction between components.



```
trce: AspNetCoreVerifiableCredentialsB2C.ApiIssuerController[0]
      2022-08-09T07:37:46.1143637Z 79.91.126.110
      GET http://83fa-79-91-126-110.ngrok.io/api/issuer/issue-response?id=6766fbb8-40e7-4e1d-a743-dadbb72673b3
      Accept: */*
      Accept-Encoding: gzip, deflate, br
      Accept-Language: en-US,en;q=0.9
      Host: 83fa-79-91-126-110.ngrok.io
      Referer: https://litware369b2c.b2clogin.com/
      User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134
 Safari/537.36 Edg/103.0.1264.77
      Origin: https://litware369b2c.b2clogin.com
      Sec-Ch-Ua: " Not;A Brand";v="99", "Microsoft Edge";v="103", "Chromium";v="103"
      Sec-Ch-Ua-Mobile: ?0
```

This leads us directly to the next section.

Under the hood

As already mentioned in the two previous parts of this guide, Microsoft Entra Verified ID includes the Request Service REST API.

As such, this API provides an abstraction and integration layer to the Microsoft Entra Verified ID service in order to both issue and verify VCs. This API callable from any programming language is unsurprisingly used here in the code sample application to issue your Verified employee credentials as well as the Custom credentials you created.

About the Azure AD B2C custom policies for issuance and verification

As already introduced, custom policies in Azure AD B2C are XML-formatted configuration files that define the behavior of your Azure AD B2C tenant.

In this third and last part of the guide, we use a custom policy to signup and issue a Custom (verifiable) credential, and in turn to also sign-in with the issued custom credential and verify it.

About the REST API claims provider and related technical profiles

The code sample application that runs locally and is available on the Internet thank to ngrok exposes a REST API endpoint to both issue and verify the above Custom credential. For example, in our illustration:

https://0a12-93-3-247-151.ngrok.io/api

So, it is simultaneously a web application and more importantly a REST API while we are here only interested in the REST API façade that will be invoked by the IEF orchestration platform of Azure AD B2C as per executed custom policy. In other words, from an Azure AD B2C perspective, this is a REST API to invoke.

Leveraging the IEF orchestration platform's advanced capabilities, you can indeed add your own business logic to a user journey by calling your own REST API. As a state machine, the IEF orchestration platform can send and receive data from your REST API, here the code sample application, to exchange claims. As such, for example, you can:

1. **Use external identity data source to validate user input data**. For example, you can verify that the email address provided by the user exists in your customer's database, and if not, present an error. You can as well think of API connectors as a way of supporting outbound webhooks because the call is made when an event occurs e.g. a sign up.

2. **Process claims**. If a user enters their first name in all lowercase or all uppercase letters, your REST API can format the name with only the first letter capitalized and return it to Azure AD B2C. However, when using a custom policy, `ClaimsTransformations` is preferred over calling a REST API.
3. **Dynamically enrich user data by further integrating with corporate line-of-business applications**. Your REST API can receive the user's email address, query the customer's database, and return the user's loyalty number to Azure AD B2C. Then return claims can be stored in the user's Azure AD account, evaluated in the next orchestration steps, or included in the access token.
4. **Run custom business logic**. You can send push notifications, update corporate databases, run a user migration process, manage permissions, audit databases, and perform any other workflows.

In this specific case, the code sample application as a REST API façade allows to both issue and verify the previously created Custom credential. We falls in this fourth category of usage.

In order to interact with a REST API, a claims provider along with some technical profiles must be defined in the custom policy. Technical profiles enable to interact with our API routes according to the request.

In the *TrustFrameworkExtensionsVC.xml,* we have thus two technical profiles being defined:

1. The first one named `REST-VC-PostIssuanceClaims` is used to send an issuance request to the REST API to issue in turn a Custom (verifiable) credential.

```
<TechnicalProfile Id="REST-VC-PostIssuanceClaims">
  <DisplayName>Verifiable Credentials Authentication Result</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="ServiceUrl">https://0a12-93-3-247-151.ngrok.io/api/issuer/issuance-claims-b2c</Item>
    <Item Key="AuthenticationType">ApiKeyHeader</Item> <!-- change this to None for dev/debug -->
    <Item Key="SendClaimsIn">Body</Item>
    <Item Key="IncludeClaimResolvingInClaimsHandling">true</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="bf64e602-f72b-11ec-b939-0242ac120002" StorageReferenceId="B2C_1A_RestApiKey" /> <!-- remember
to set this in the samples appSettings/etc -->
  </CryptographicKeys>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="correlationId"  PartnerClaimType="id"
DefaultValue="{Context:CorrelationId}" AlwaysUseDefaultValue="true" />
    <InputClaim ClaimTypeReferenceId="IPAddress" DefaultValue="{Context:IPAddress}"
AlwaysUseDefaultValue="true" />
    <InputClaim ClaimTypeReferenceId="tenantId" PartnerClaimType="tid"
DefaultValue="{Policy:TenantObjectId}" AlwaysUseDefaultValue="true"  />
    <InputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="oid" />
    <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="username" />
    <InputClaim ClaimTypeReferenceId="displayName" />
    <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
    <InputClaim ClaimTypeReferenceId="surName" PartnerClaimType="lastName" />
  </InputClaims>
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

2. The second one named `REST-VC-GetAuthResult` is used to handle a presentation request to authenticate the user.

```
<TechnicalProfile Id="REST-VC-GetAuthResult">
  <DisplayName>Verifiable Credentials Authentication Result</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
```

```
  <Metadata>
    <Item Key="ServiceUrl">https://0a12-93-3-247-151.ngrok.io/api/verifier/presentation-response-
b2c</Item>
    <Item Key="AuthenticationType">None</Item>
    <Item Key="SendClaimsIn">Body</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="VCStateId" PartnerClaimType="id" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="vcCredentialType" PartnerClaimType="vcType" />
    <OutputClaim ClaimTypeReferenceId="vcSubject" PartnerClaimType="vcSub" />
    <OutputClaim ClaimTypeReferenceId="vcIssuer" PartnerClaimType="vcIss" />
    <OutputClaim ClaimTypeReferenceId="vcKey" PartnerClaimType="vcKey" />
    <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="oid" />
    <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="username" />
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
    <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="lastName" />
    <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="DID" AlwaysUseDefaultValue="true"
/>
   </OutputClaims>
   <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

In the two above `Metadata` sections, please note the ServiceUrl value:

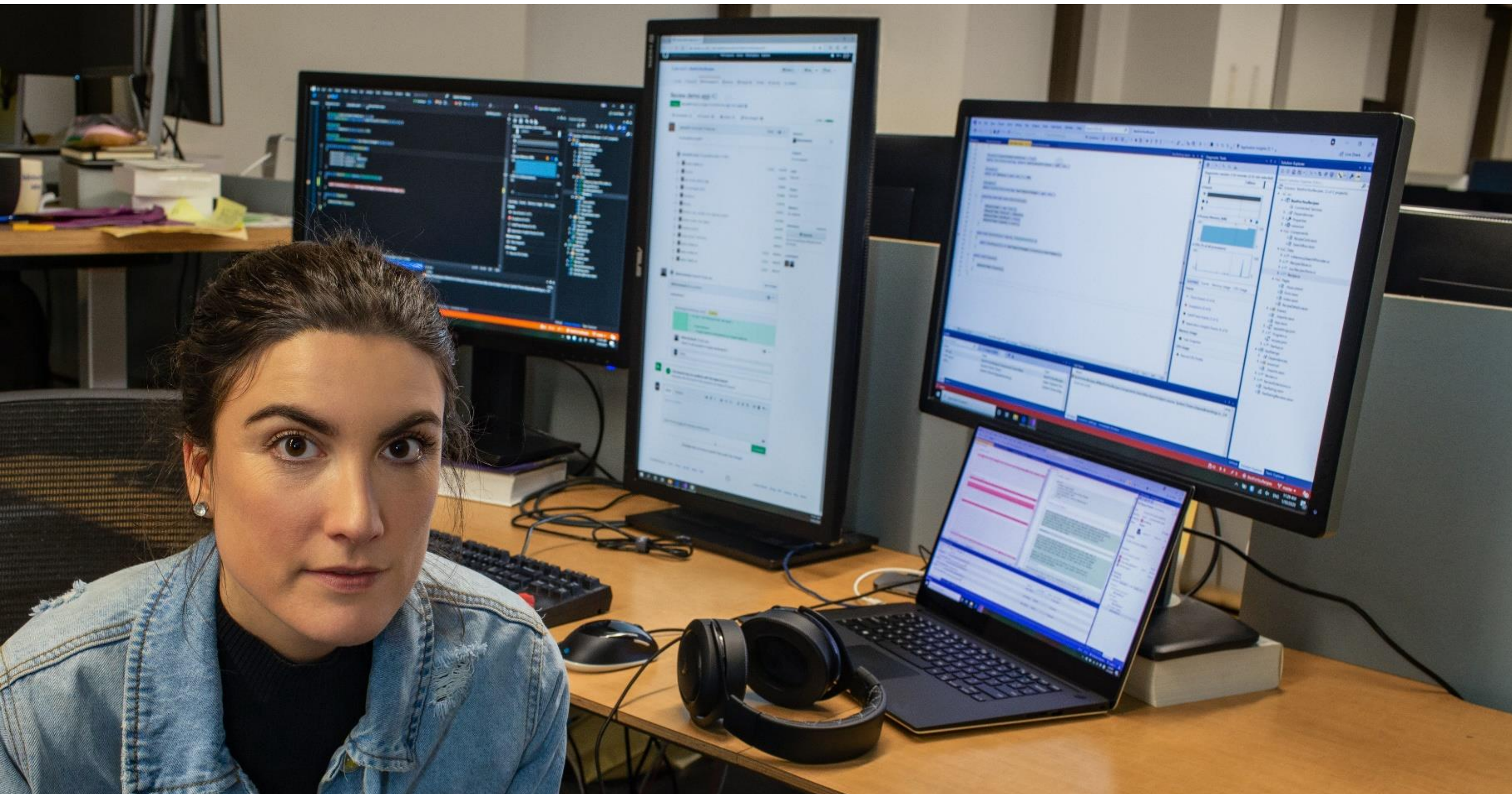
- For the issuance request:

https://0a12-93-3-247-151.ngrok.io/api/**issuer/issuance-claims-b2c**

- Conversely for the presentation request:

https://0a12-93-3-247-151.ngrok.io/api/**verifier/presentation-response-b2c**

As previously emphasized, the REST API façade, i.e., the code sample application's API, exposes these two endpoints to both issue and verify custom (verifiable) credentials through the IEF orchestration platform of Azure AD B2C.



Let's now consider for that the related orchestration steps in the supplied custom policies.

About the orchestration steps

As mentioned above, an orchestration step is a reference to a method that implements its intended purpose or functionality. The method represents a technical profile.

To call the technical profiles defined in the *TrustFrameworkExtensionsVC.xml,* you will need to create an orchestration step. You will be more particularly interested in the orchestration step #7 defined in the *SignupOrSigninVCQ.xml custom* policy file.

```
<OrchestrationStep Order="7" Type="ClaimsExchange">
   <Preconditions>
     <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
        <Value>newUser</Value>
        <Action>SkipThisOrchestrationStep</Action>
     </Precondition>
   </Preconditions>
   <ClaimsExchanges>
    <ClaimsExchange Id="PostVCIssuanceClaims" TechnicalProfileReferenceId="REST-VC-PostIssuanceClaims" />
   </ClaimsExchanges>
</OrchestrationStep>
```

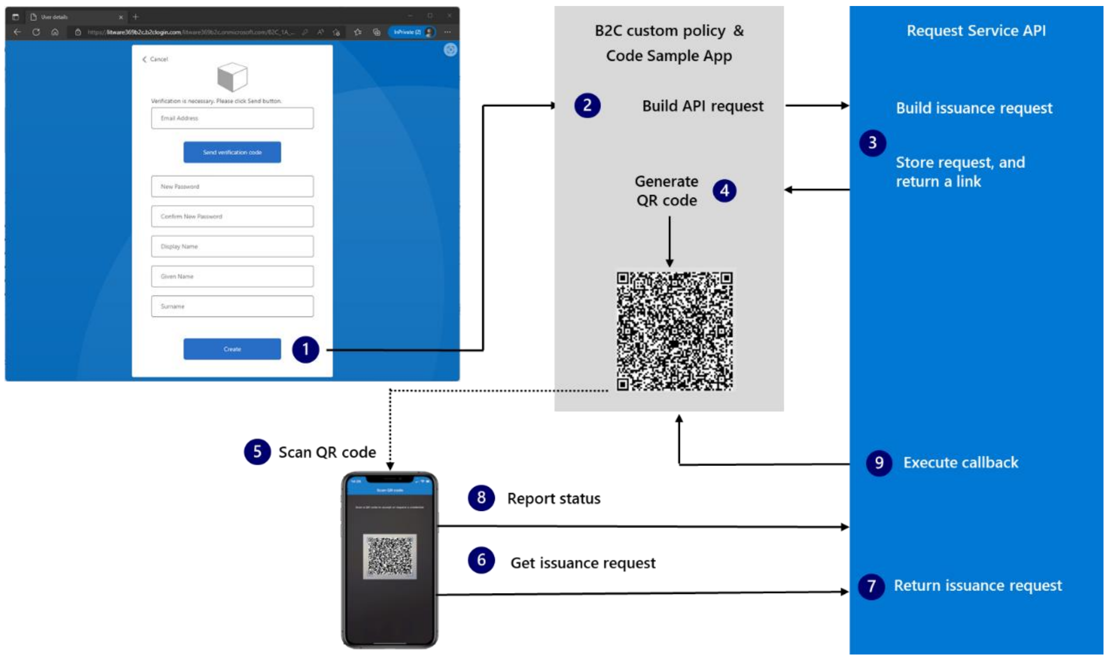
As you can see, this orchestration step passes the claims to the code sample application to prepare for issuing a Custom credential if it's a new user. You can see that we have here a claim exchange which refers to the technical profile with id `REST-VC-PostIssuanceClaims` outlined above.

About the calls to the Request Service REST API during the signup flow

The following figure explains what's going on during the *issuance* flow between the executed custom policy (defined in Azure AD B2C), the local code sample application used as a REST API façade, and the Request Service REST API of your Microsoft Entra Verified ID service.

The code sample application uses some .html files as a user interface, a.k.a. a content definition in the Azure AD B2C jargon, in addition to the custom policy XML-formatted files.

The first one to consider is the *selfAsserted.html* file, which displays the QR Code for issuance request (right) after a successful signup and the creation of the user.



The Request Service REST API is called, and a QR code or deep link is returned after this call. Microsoft Authenticator retrieves the request from the Microsoft Entra Verified ID service. When the QR code is scanned, the service calls back to the code sample application using the B2C_1A_VC_SUSIQ custom policy.

The flow details are as follows:

- Signup a new user.
- After the signup process, the custom policy calls a request implemented in the code sample application to build an issuance request by calling the Request Service REST API with the correct payload. See Specify the Request Service REST API issuance request.
- The service creates the request in a JSON Web Token (JWT) format, and stores the request. The Rest API typically returns the deep link to the request and the QR code with this address encoded. The request is stored in the service
- The browser generates the QR code using the requests called in the *selfAsserted.html* file.
- The user scans the QR code using Microsoft Authenticator. The QR code informs Microsoft Authenticator where it can retrieve the issuance request.
- The issuance request is downloaded from the address pointing to the Microsoft Entra Verified ID service.
- After downloading, the request on the service is deleted. When the request is downloaded, the service sends a notification to the call-back endpoint. The call-back informs the code sample application that the QR code has been scanned.
- Microsoft Authenticator follows the regular flow to add the VC. On completion, Microsoft Authenticator reports the status to the service.
- At this moment, after the issuance, there is also in turn a call-back to inform the code sample application.

As per section **Issue a VC during signup** above, the signup process is as follows :

1. The IEF orchestration platform of Azure AD B2C instantiates the B2C_1A_VC_SUSIQ custom policy and executes in order all the defined orchestration steps taking care of the preconditions. At this

point, the signup policy being executed is the same as any other signup policy like the one that comes with the Azure AD B2C starter pack.

At some point, the execution flow runs the *unified.html* located in your storage account by using the following xml in the *TrustFrameworkExtensionsVC.xml*.

```
 <ContentDefinition Id="api.signuporsignin">
   <LoadUri>https://<your_storage_account>.blob.core.windows.net/<your_container>/unified.html</LoadUri>
   <DataUri>urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:2.1.0</DataUri>
   <LocalizedResourcesReferences MergeBehavior="Prepend">
     <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.signuporsignin.en" />
    </LocalizedResourcesReferences>
 </ContentDefinition>
```

By using https://jwt.ms as the redirect URI for the application you created in your Azure B2C AD tenant, the decoded token after the signup is like the following:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "1dZZY23T1zabgVVbFSXB__xMlzUCG3XsYV5yIjeJmmU"
}.{
  "exp": 1657187439,
  "nbf": 1657183839,
  "ver": "1.0",
  "iss": "https://litware369b2c.b2clogin.com/cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2/v2.0/",
  "sub": "e621a4e1-f4ef-4b8e-bfd1-aea6b531fbc7",
  "aud": "f83ece93-0e98-4379-8df3-248619483bd9",
  "acr": "b2c_1a_vc_susiq",
  "nonce": "defaultNonce",
  "iat": 1657183839,
  "auth_time": 1657183839,
  "signInName": "johndoevc@gmail.com",
  "oid": "e621a4e1-f4ef-4b8e-bfd1-aea6b531fbc7",
  "email": "dialloakh@gmail.com",
  "name": "John Doe VC",
  "given_name": "John Doe",
  "family_name": "Doe",
  "tid": "cbf80679-8d13-4ac7-a5de-a4c1d69bd5d2"
}
```

2. After the signup, it's time to issue a VC for the new user. The user is redirected to the VC issuance page. This is a GET request which interacts with the API endpoint exposed by the code sample application. This request will displays a QR code that you need to scan with Microsoft Authenticator.

    a. **On the custom policies side**. This request runs the following content definition in the *TrustFrameworkExtensionsVC.xml* file to render the related html content:

```
 <ContentDefinition Id="api.signuporsignin.quick">

<LoadUri>https://<your_storage_account>.blob.core.windows.net/<your_container>/unifiedquick.html</LoadUri>
   <DataUri>urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:2.1.0</DataUri>
   <LocalizedResourcesReferences MergeBehavior="Prepend">
     <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.signuporsigninquick.en"
/>
    </LocalizedResourcesReferences>
 </ContentDefinition>
```

The above content definition will then execute some orchestration steps in the *SignupOrSigninVCQ.xml* file in order to process the issuance request against the API endpoint exposed by the code sample application.

```
<OrchestrationStep Order="7" Type="CombinedSignInAndSignUp"
                    ContentDefinitionReferenceId="api.signuporsignin.quick">
   <ClaimsProviderSelections>
      <ClaimsProviderSelection TargetClaimsExchangeId="PostVCIssuanceClaims" />
   </ClaimsProviderSelections>
 </OrchestrationStep>
 <OrchestrationStep Order="8" Type="ClaimsExchange">
   <Preconditions>
     <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
        <Value>newUser</Value>
        <Action>SkipThisOrchestrationStep</Action>
     </Precondition>
   </Preconditions>
   <ClaimsExchanges>
    <ClaimsExchange Id="PostVCIssuanceClaims" TechnicalProfileReferenceId="REST-VC-PostIssuanceClaims" />
   </ClaimsExchanges>
  </OrchestrationStep>
```

b. **On the API façade side**. The API endpoint allows to both issue and verify VC. More specifically, in the *ApiIssuerController.cs* file, the previous orchestrations will execute the following GET request in order to start the issuance of the credential for the new user.

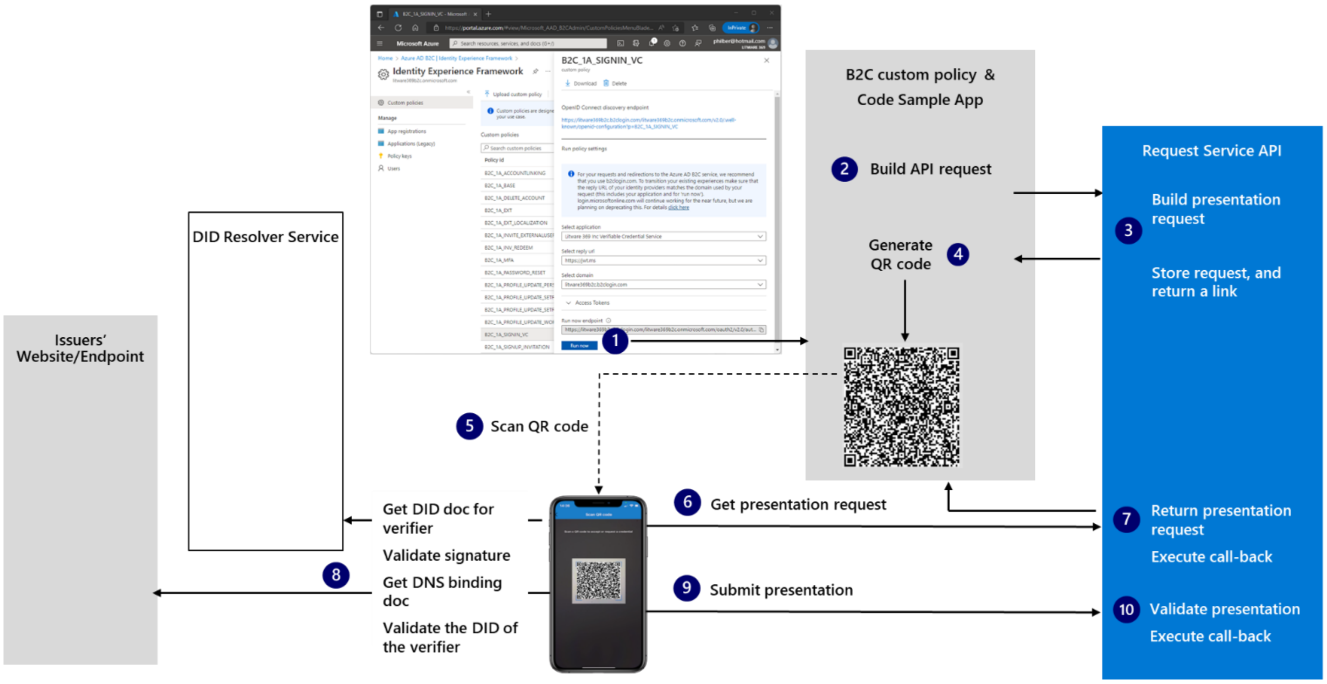
```
[HttpGet("/api/issuer/issue-request")]
public ActionResult IssuanceReference() {
   TraceHttpRequest();
   […omitted for brievity…]
}
```

If the call contains the query string parameter `id`, it should be the same id as passed to the below api/issuer/issuance-claims-b2c endpoint.

```
/// <summary>
/// Azure AD B2C REST API Endpoint for storing claims for future VC issuance request
/// HTTP POST comes from Azure AD B2C
/// body : The InputClaims from the B2C policy. The 'id' is B2C's correlationId
///        Other claims are claims that may be used in the VC (see issue-request above)
/// </summary>
/// <returns>200 OK, 401 (api-key) or 404 (missing id/oid)</returns>
[HttpPost("/api/issuer/issuance-claims-b2c")]
public ActionResult IssuanceClaimsB2C() {
   TraceHttpRequest();
   try {
      string body = GetRequestBody();
      _log.LogTrace(body);
      // if the appSettings has an API key for B2C, make sure B2C passes it
      if ( !VerifyB2CApiKey() ) {
         return ReturnUnauthorized( "invalid x-api-key" );
      }
      // make sure B2C passed the 'id' claim (correlationId) that we use for caching
      // (without it we will never be able to find these claims again)
      JObject b2cClaims = JObject.Parse(body);
      string correlationId = b2cClaims["id"].ToString();
      if (string.IsNullOrEmpty(correlationId)) {
         return ReturnErrorMessage("Missing claim 'id'");
      }
      // make sure B2C atleast passes the oid claim as that is the key for identifying a B2C user from a
VC
      string oid = b2cClaims["oid"].ToString();
      if (string.IsNullOrEmpty(oid)) {
         return ReturnErrorMessage("Missing claim 'oid'");
      }
      CacheObjectWithExpiery(correlationId, b2cClaims );
      return new OkResult();
  } catch (Exception ex) {
       return ReturnErrorMessage(ex.Message);
  }
}
```

About the calls to the Request Service REST API during the signing flow

The sign-in flow is a verification process which implies both the local API verifier and one of the following custom policies: either B2C_1A_VC_SUSIQ (and select to sign-in with VC), or B2C_1A_SIGNIN_VC.



The flow details are as follows:

- Select and run the custom policy to sign-in.
- The code sample application creates a presentation request by calling the Request Service REST API with the correct payload. See Specify the Request Service REST API verify request.
- The service creates the request in the correct format (JWT) and stores the request. The Request Service REST API returns the deep link to the request and optionally the QR code with this address encoded. The request is stored in our service.
- The browser generates the QR code.
- Microsoft Authenticator scans the QR code - With a mobile app, it instead opens the deep link -.
- The presentation request is downloaded from the address pointing to the Microsoft Entra Verified ID service.
- After downloading, the request on the service is deleted.
- The next part of the flow validates at the Microsoft Authenticator level the signature of the verifier and the DNS binding.

- Microsoft Authenticator submits the presentation to the REST API, and thus presents the requested VC(s) to the service.
- The service validates the VC(s). The result of the verification is sent back to the call-back endpoint specified in 2 and optionally sends back the content of the VC(s) as well.

Like the signup flow, the custom policy runs a GET request implemented in the *ApiVerifierController.cs* file.

a. **On the custom policies side**. The execution of one of the above custom policies triggers a GET request that will be processed in the *ApiVerifierController.cs* file of the code sample application, and in turn a QR code will be displayed. The user will need to scan this QR code in order to present their VC. The user journey being called to sign-in with VC is the following one.

```
<UserJourney Id="SigninVC">
      <OrchestrationSteps>
        <OrchestrationStep Order="1" Type="ClaimsExchange">
          <ClaimsExchanges>
            <ClaimsExchange Id="LocalAccountSigninEmailExchange"
TechnicalProfileReferenceId="SelfAsserted-VCSignin" />
          </ClaimsExchanges>
        </OrchestrationStep>
        <OrchestrationStep Order="2" Type="SendClaims" CpimIssuerTechnicalProfileReferenceId="JwtIssuer"
/>
      </OrchestrationSteps>
      <ClientDefinition ReferenceId="DefaultWeb" />
 </UserJourney>
```

b. **On the API facade side**. As aforementioned, the code for the presentation request and the subsequent verification is implemented in the *ApiVerifierController.cs* file of the code sample application. the following request is run when the previous policy is executed:

```
[HttpGet("/api/verifier/presentation-request")]
public ActionResult PresentationReference() {
   TraceHttpRequest();
   […]
}
```

The QR code is in turn obtained by the content definition by calling the following method:

```
/// <summary>
/// This method is called from the B2C HTML/javascript to get a QR code deeplink that
/// points back to this API instead of the VC Request Service API.
/// You need to pass in QueryString parameters such as 'id' or 'StateProperties' which both
/// are the B2C CorrelationId. StateProperties is a base64 encoded JSON structure.
/// </summary>
/// <returns>JSON deeplink to this API</returns>
[HttpGet("/api/verifier/presentation-request-link")]
public ActionResult StaticPresentationReferenceGet() {
   TraceHttpRequest();
   try {
      string correlationId = this.Request.Query["id"];
      string stateProp = this.Request.Query["StateProperties"]; // may come from SETTINGS.transId
      if (string.IsNullOrWhiteSpace(correlationId) && !string.IsNullOrWhiteSpace(stateProp) ) {
         stateProp = stateProp.PadRight(stateProp.Length + (stateProp.Length % 4), '=');
         JObject spJson = JObject.Parse( Encoding.UTF8.GetString(Convert.FromBase64String(stateProp)) );
         correlationId = spJson["TID"].ToString();
      }
      if ( string.IsNullOrWhiteSpace(correlationId) ) {
         correlationId = Guid.NewGuid().ToString();
      }
      RemoveCacheValue( correlationId );
      var resp = new {
         requestId = correlationId,
         url = string.Format("openid://vc/?request_uri={0}/presentation-request-proxy?id={1}",
GetApiPath(), correlationId),
         expiry = (int)(DateTime.UtcNow.AddDays(1) - new DateTime(1970, 1, 1)).TotalSeconds,
         id = correlationId
      };
      string respJson = JsonConvert.SerializeObject(resp);
      _log.LogTrace("API static request Response\n{0}", respJson );
      return ReturnJson( respJson );
   } catch (Exception ex) {
       return ReturnErrorMessage(ex.Message);
   }
}
```

and its callback URI will runs a POST request in order to verify the presented VC.

```
/// <summary>
/// Azure AD B2C REST API Endpoint for retrieveing the VC presentation response
/// HTTP POST comes from Azure AD B2C
/// body : The InputClaims from the B2C policy.It will only be one claim named 'id'
/// </summary>
/// <returns>returns a JSON structure with claims from the VC presented</returns>
[HttpPost("/api/verifier/presentation-response-b2c")]
public ActionResult PresentationResponseB2C() {
   TraceHttpRequest();
   try {
      string body = GetRequestBody();
      _log.LogTrace(body);
      // if the appSettings has an API key for B2C, make sure the caller passes it
      if (!VerifyB2CApiKey()) {
         return ReturnUnauthorized("invalid x-api-key");
      }
      JObject b2cRequest = JObject.Parse(body);
      string correlationId = b2cRequest["id"].ToString();
      if (string.IsNullOrEmpty(correlationId)) {
         return ReturnErrorMessage("Missing argument 'id'");
      }
      VCCallbackEvent callback = null;
      if (!GetCachedObject<VCCallbackEvent>(correlationId, out callback)) {
         return ReturnErrorB2C("Verifiable Credentials not presented"); // 409
      }
      // remove cache data now, because if we crash, we don't want to get into an infinite loop of
crashing
      RemoveCacheValue(correlationId);
      // setup the response that we are returning to B2C
      var obj = new {
         vcType = callback.issuers[0].type[callback.issuers[0].type.Length - 1], // last
         vcIss = callback.issuers[0].authority,
         vcSub = callback.subject,
         // key is intended to be user in user's profile 'identities' collection as a signInName,
         // and it can't have colons, therefor we modify the value (and clip at following :)
         vcKey = callback.subject.Replace("did:ion:", "did.ion.").Split(":")[0]
      };
      JObject b2cResponse = JObject.Parse(JsonConvert.SerializeObject(obj));
      // add all the additional claims in the VC as claims to B2C
      foreach (KeyValuePair<string, string> kvp in callback.issuers[0].claims) {
          b2cResponse.Add(new JProperty(kvp.Key, kvp.Value));
      }
      string resp = JsonConvert.SerializeObject(b2cResponse);
      _log.LogTrace(resp);
      return ReturnJson( resp );
   } catch (Exception ex) {
        return ReturnErrorMessage(ex.Message);
   }
}
```

About the code

Since the Request Service Rest API is a multi-tenant API it needs to receive an access token when it's called. The old endpoint of the API is https://beta.(eu.)did.msidentity.com/v1.0/{*yourTenantID*}/verifiablecredentials/request

The new endpoint https://verifiedid.did.msidentity.com/v1.0/{*yourTenantID*}/verifiablecredentials/request

To get an access token we are using MSAL as library. MSAL supports the creation and caching of access token which are used when calling Azure AD protected resources like the Request Service REST API. Typically calling the library looks something like this in the *ApiBaseVCController.cs* file, see the `GetAccessToken` method:

```
app = ConfidentialClientApplicationBuilder.Create(AppSettings.ClientId)
    .WithClientSecret(AppSettings.ClientSecret)
    .WithAuthority(new Uri(AppSettings.Authority))
    .Build();
```

And creating an access token:

```
result = await app.AcquireTokenForClient(scopes)
                  .ExecuteAsync();
```

As of this writing, the scope needs to be `3db474b9-6a0c-4840-96ac-1fceb342124f/.default`. This might change in the future.

Calling the Request Service REST API looks like this:

```
HttpClient client = new HttpClient();
var defaultRequestHeaders = client.DefaultRequestHeaders;
defaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", result.AccessToken);

HttpResponseMessage res = await client.PostAsync(AppSettings.ApiEndpoint, new StringContent(jsonString,
Encoding.UTF8, "application/json"));
response = await res.Content.ReadAsStringAsync();
```

As a conclusion

This concludes this walkthrough and this third and last part of this guide.

We hope you have enjoyed the additional tour with Azure AD B2C, the potential and the power of the custom policies, etc., and you are not too tired with all those technical details.

As part of this third part of our guided tour, we have outlined all the steps required to setup your own demo environment in Azure with Azure AD B2C, to customize and upload B2C custom policies along with the setup of an API façade for the integration with Microsoft Entra Verified ID, and ultimately to issue, and verify custom Verifiable Credentials (VCs) via B2C custom policies along with some additional explanations and related rational.

Furthermore, we have also some system integration partners ready to help you quickly do (further) prototyping, proof of concepts, pilots, etc.

Going beyond

To continue learning about the passionate subject of verifiable credentials and more generally speaking of decentralized identity, please check out these additional resources:

- Get customer stories and verifiable credentials resources at http://aka.ms/verifyonce.
- Understand the basics of decentralized identity via our five-part blog series.
- Quick overview: http://aka.ms/didexplained.
- Learn more about the decentralized identity movement at aka.ms/ownyouridentity.
- Documentation for developers: http://aka.ms/didfordevs.
- Get involved with http://identity.foundation , the industry working group for all things Decentralized ID (DID).

Verify once. use everywhere. Start your decentralized identity journey today with verifiable credentials in Microsoft Entra.

aka.ms/verifyonce