

Contents

Azure AD B2C Documentation

Overview

[About Azure AD B2C](#)

[Technical and feature overview](#)

Quickstarts

[Set up sign-in - ASP.NET](#)

[Set up sign-in - Desktop](#)

[Set up sign-in - Single page](#)

Tutorials

[1 - Create B2C tenant](#)

[2 - Register an application](#)

[3 - Create user flows](#)

[Add identity providers](#)

[Customize the UI](#)

[Authenticate users](#)

[ASP.NET](#)

[Desktop](#)

[Single page](#)

[Grant API access](#)

[ASP.NET](#)

[Desktop](#)

[Single page](#)

Samples

Concepts

[Application types](#)

[Authentication protocols](#)

[OAuth2 protocol](#)

[OpenID Connect protocol](#)

[Authorization Code grant flow](#)

[Implicit flow](#)

[Tokens](#)

[Request access token](#)

[User flow and policy](#)

[User flows](#)

[Custom policies](#)

[User accounts](#)

[How-to guides](#)

[App integration](#)

[Register an application](#)

[Register a SAML service provider](#)

[Register a Graph application](#)

[Add a web API application](#)

[Add a native client application](#)

[iOS ObjC using App Auth](#)

[Android using App Auth](#)

[User flow](#)

[Create a flow](#)

[Sign-up or sign-in](#)

[Resource owner password credentials](#)

[Set up self-serve password reset](#)

[UX customization](#)

[Customize the UI](#)

[JavaScript and page layouts](#)

[Customize language](#)

[Password complexity](#)

[Disable email verification](#)

[Enable MFA](#)

[External identity providers](#)

[Amazon](#)

[Azure AD \(Single-tenant\)](#)

[Microsoft Account](#)

- [Facebook](#)
- [GitHub](#)
- [Google](#)
- [LinkedIn](#)
- [QQ](#)
- [Twitter](#)
- [WeChat](#)
- [Weibo](#)
- [Generic identity provider](#)
- [Tokens and session management](#)
 - [Configure tokens](#)
 - [Configure session behavior](#)
 - [Configure age gating](#)
 - [Define custom attributes](#)
 - [Pass through external IdP token](#)
- [Custom policy](#)
 - [Create a policy](#)
 - [Get started with custom policies](#)
 - [Resource owner password credentials](#)
 - [Enable keep me signed in](#)
 - [Password change](#)
 - [Phone sign-up & sign-in](#)
- [UX customization](#)
 - [Configure user input](#)
 - [Customize the UI](#)
 - [Custom email](#)
 - [Enable JavaScript](#)
 - [Password complexity](#)
- [External identity providers](#)
 - [OIDC/OAuth providers](#)
 - [Amazon](#)
 - [Azure AD \(Single-tenant\)](#)

- Azure AD (Multi-tenant)
- Google
- LinkedIn
- Microsoft Account
- Twitter
- SAML providers
- ADFS
- Salesforce
- Tokens and session management
 - Customize tokens
 - Pass through external IdP token
- Adaptive experience
 - Set up direct sign-in
 - Add your own business logic
 - Validate user input
 - Obtain additional claims
 - Add your own RESTful API
 - Secure RESTful APIs with basic auth
 - Secure RESTful APIs with certificate auth
 - Define custom attributes
- Troubleshooting
 - Collect logs using Application Insights
 - Policy validation
- Usage analytics
- Reference
 - Trust Framework definition
 - TrustFrameworkPolicy
 - BuildingBlocks
 - ClaimsSchema
 - ClaimsTransformations
 - Boolean
 - Date

- [General](#)
- [Integer](#)
- [JSON](#)
- [Phone number](#)
- [External accounts](#)
- [StringCollection](#)
- [String](#)
- [Predicates](#)
- [ContentDefinitions](#)
- [Localization](#)
 - [Localization string IDs](#)
- [DisplayControls](#)
- [Verification](#)
- [ClaimsProviders](#)
- [TechnicalProfiles](#)
 - [About technical profiles](#)
 - [Azure Multi-Factor Authentication](#)
 - [Claim resolvers](#)
 - [Azure Active Directory](#)
 - [Claims transformation](#)
 - [JWT token issuer](#)
- [OAuth1](#)
- [OAuth2](#)
- [One-time password](#)
- [OpenID Connect](#)
- [REST](#)
- [SAML](#)
- [Self-asserted](#)
- [SSO session](#)
- [Validation](#)
- [UserJourneys](#)
- [RelyingParty](#)

[Use b2clogin.com](#)

[b2clogin.com overview](#)

[Migrate web API to b2clogin.com](#)

[Automation](#)

[Azure Monitor](#)

[Manage users - Microsoft Graph](#)

[Deploy with Azure Pipelines](#)

[Manage policies with PowerShell](#)

[Audit logs](#)

[Manage users - Azure portal](#)

[Secure API Management API](#)

[Compliance](#)

[User access](#)

[User data](#)

[Migration](#)

[Migrate users](#)

[Reference](#)

[Identity Experience Framework release notes](#)

[Code samples](#)

[Page layout versions](#)

[Cookie definitions](#)

[Error codes](#)

[Microsoft Graph API operations](#)

[Region availability & data residency](#)

[Billing model](#)

[Threat management](#)

[Extensions app](#)

[User flow versions](#)

[Resources](#)

[Azure Roadmap](#)

[Frequently asked questions](#)

[Getting help](#)

Pricing

[Pricing calculator](#)

Service updates

Solutions and training

Stack Overflow

Support

Videos

What is Azure Active Directory B2C?

1/28/2020 • 5 minutes to read • [Edit Online](#)

Azure Active Directory B2C provides business-to-customer identity as a service. Your customers use their preferred social, enterprise, or local account identities to get single sign-on access to your applications and APIs.

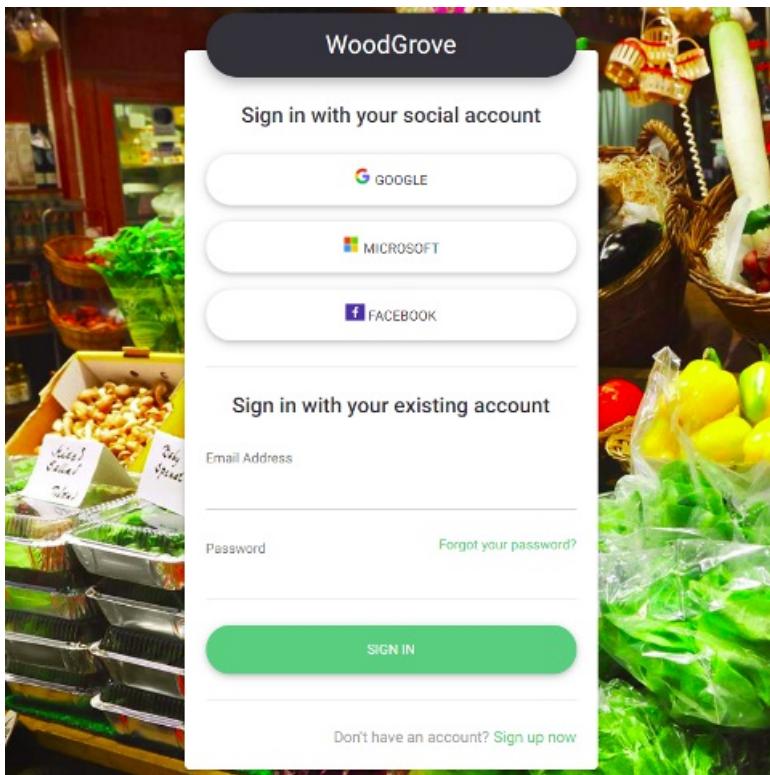


Azure Active Directory B2C (Azure AD B2C) is a customer identity access management (CIAM) solution capable of supporting millions of users and billions of authentications per day. It takes care of the scaling and safety of the authentication platform, monitoring and automatically handling threats like denial-of-service, password spray, or brute force attacks.

Custom-branded identity solution

Azure AD B2C is a white-label authentication solution. You can customize the entire user experience with your brand so that it blends seamlessly with your web and mobile applications.

Customize every page displayed by Azure AD B2C when your users sign up, sign in, and modify their profile information. Customize the HTML, CSS, and JavaScript in your user journeys so that the Azure AD B2C experience looks and feels like it's a native part of your application.



Single sign-on access with a user-provided identity

Azure AD B2C uses standards-based authentication protocols including OpenID Connect, OAuth 2.0, and SAML. It integrates with most modern applications and commercial off-the-shelf software.



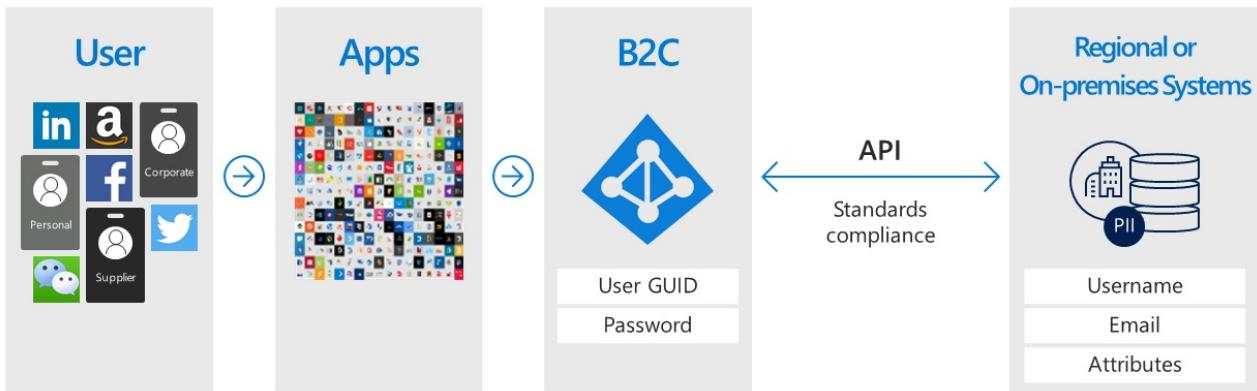
By serving as the central authentication authority for your web applications, mobile apps, and APIs, Azure AD B2C enables you to build a single sign-on (SSO) solution for them all. Centralize the collection of user profile and preference information, and capture detailed analytics about sign-in behavior and sign-up conversion.

Integrate with external user stores

Azure AD B2C provides a directory that can hold 100 custom attributes per user. However, you can also integrate

with external systems. For example, use Azure AD B2C for authentication, but delegate to an external customer relationship management (CRM) or customer loyalty database as the source of truth for customer data.

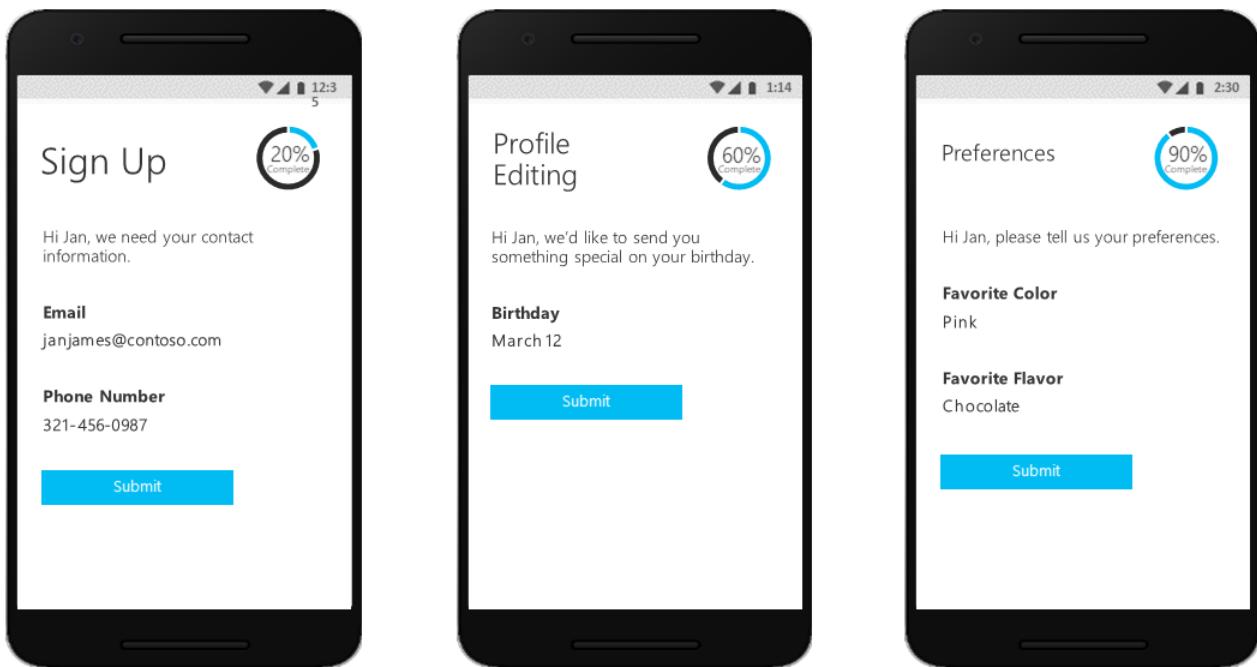
Another external user store scenario is to have Azure AD B2C handle the authentication for your application, but integrate with an external system that stores user profile or personal data. For example, to satisfy data residency requirements like regional or on-premises data storage policies.



Azure AD B2C can facilitate collecting the information from the user during registration or profile editing, then hand that data off to the external system. Then, during future authentications, Azure AD B2C can retrieve the data from the external system and, if needed, include it as a part of the authentication token response it sends to your application.

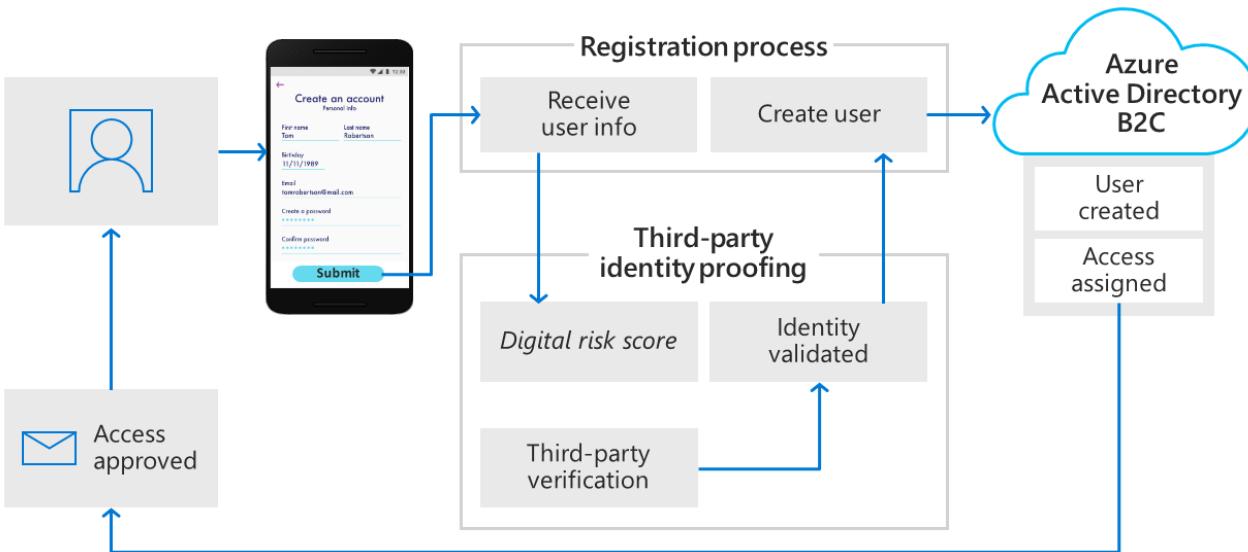
Progressive profiling

Another user journey option includes progressive profiling. Progressive profiling allows your customers to quickly complete their first transaction by collecting a minimal amount of information. Then, gradually collect more profile data from the customer on future sign-ins.



Third-party identity verification and proofing

Use Azure AD B2C to facilitate identity verification and proofing by collecting user data, then passing it to a third party system to perform validation, trust scoring, and approval for user account creation.



These are just some of the things you can do with Azure AD B2C as your business-to-customer identity platform. The following sections of this overview walk you through a demo application that uses Azure AD B2C. You're also welcome to move on directly to a more in-depth [technical overview of Azure AD B2C](#).

Example: WoodGrove Groceries

[WoodGrove Groceries](#) is a live web application created by Microsoft to demonstrate several Azure AD B2C features. The next few sections review some of the authentication options provided by Azure AD B2C to the WoodGrove website.

Business overview

WoodGrove is an online grocery store that sells groceries to both individual consumers and business customers. Their business customers buy groceries on behalf of their company, or businesses that they manage.

Sign-in options

WoodGrove Groceries offers several sign-in options based on the relationship their customers have with the store:

- **Individual** customers can sign up or sign in with individual accounts, such as with a social identity provider or an email address and password.
- **Business** customers can sign up or sign in with their enterprise credentials.
- **Partners** and suppliers are individuals who supply the grocery store with products to sell. Partner identity is provided by [Azure Active Directory B2B](#).

WoodGrove Groceries

Sign in

Language
Choose language

Individual customers
Order your fresh groceries with WoodGrove Groceries and our friendly drivers will deliver your grocery shopping to your home door.

Business customers
Order your fresh groceries with WoodGrove Groceries and our friendly drivers will deliver your grocery shopping to your office door.

Business partners
Manage your local produce in our inventory so we can deliver your fresh groceries to our customers.

[SIGN IN WITH YOUR PERSONAL ACCOUNT](#)

[SIGN UP WITH YOUR PERSONAL ACCOUNT](#)

[SIGN IN WITH YOUR WORK ACCOUNT](#)

[SIGN UP WITH YOUR WORK ACCOUNT](#)

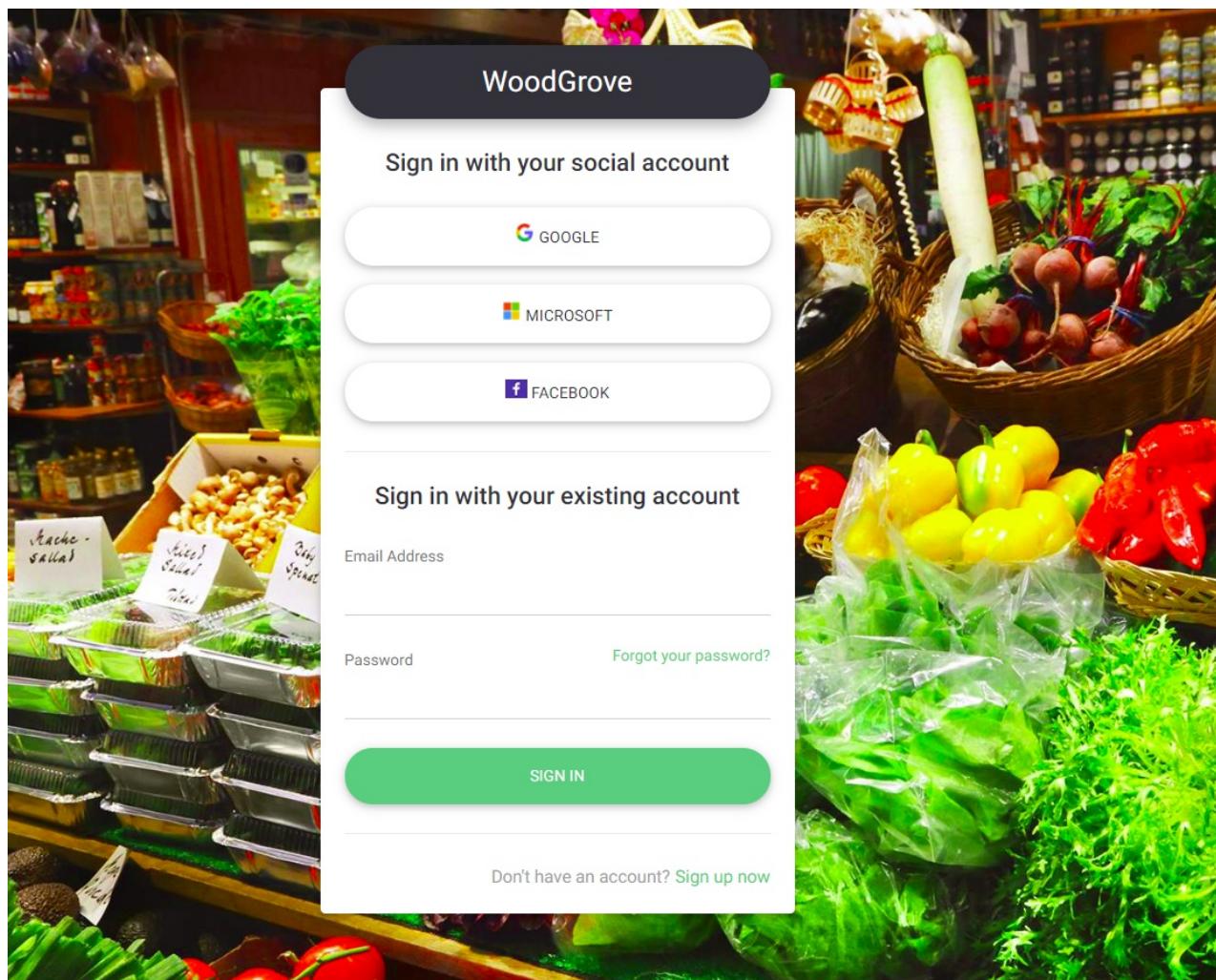
[SIGN IN WITH YOUR SUPPLIER ACCOUNT](#)

[SIGN UP TO BE A WOODGROVE SUPPLIER](#)



Authenticate individual customers

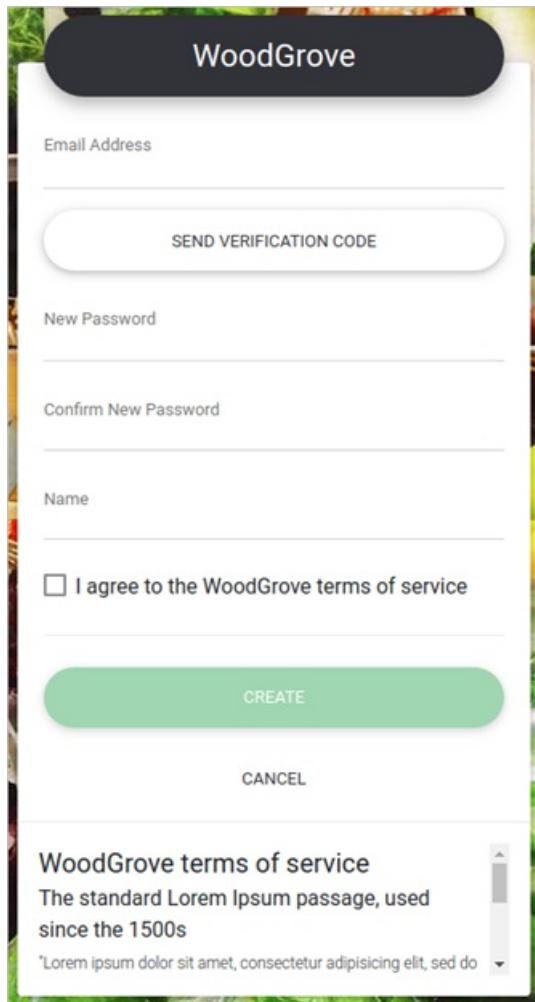
When a customer selects **Sign in with your personal account**, they're redirected to a customized sign-in page hosted by Azure AD B2C. You can see in the following image that we've customized the user interface (UI) to look and feel just like the WoodGrove Groceries website. WoodGrove's customers should be unaware that the authentication experience is hosted and secured by Azure AD B2C.



WoodGrove allows their customers to sign up and sign in by using their Google, Facebook, or Microsoft accounts

as their identity provider. Or, they can sign up by using their email address and a password to create what's called a *local account*.

When a customer selects **Sign up with your personal account** and then **Sign up now**, they're presented with a custom sign-up page.



After entering an email address and selecting **Send verification code**, Azure AD B2C sends them the code. Once they enter their code, select **Verify code**, and then enter the other information on the form, they must also agree to the terms of service.

Clicking the **Create** button causes Azure AD B2C to redirect the user back to the WoodGrove Groceries website. When it redirects, Azure AD B2C passes an OpenID Connect authentication token to the WoodGrove web application. The user is now signed-in and ready to go, their display name shown in the top-right corner to indicate they're signed in.

A screenshot of the WoodGrove Groceries website header. It features the "WoodGrove Groceries" logo, a "Catalog" link, a "Cart" icon showing "0" items, and a user profile for "Ellen Adams - Individual Customer". Below the header is a dark navigation bar with "Catalog" on the left and a search bar on the right.

Authenticate business customers

When a customer selects one of the options under **Business customers**, the WoodGrove Groceries website invokes a different Azure AD B2C policy than it does for individual customers.

This policy presents the user with an option to use their corporate credentials for sign-up and sign-in. In the WoodGrove example, users are prompted to sign in with any Office 365 or Azure AD account. This policy uses a [multi-tenant Azure AD application](#) and the `/common` Azure AD endpoint to federate Azure AD B2C with any Office

365 customer in the world.

Authenticate partners

The **Sign in with your supplier account** link uses Azure Active Directory B2B's collaboration functionality. Azure AD B2B is a family of features in Azure Active Directory to manage partner identities. Those identities can be federated from Azure Active Directory for access into Azure AD B2C-protected applications.

Learn more about Azure AD B2B in [What is guest user access in Azure Active Directory B2B?](#).

Next steps

Now that you have an idea of what Azure AD B2C is and some of the scenarios it can help with, dig a little deeper into its features and technical aspects.

[Azure AD B2C technical overview >](#)

Technical and feature overview of Azure Active Directory B2C

1/28/2020 • 14 minutes to read • [Edit Online](#)

A companion to [About Azure Active Directory B2C](#), this article provides a more in-depth introduction to the service. Discussed here are the primary resources you work with in the service, its features, and how these enable you to provide a fully custom identity experience for your customers in your applications.

Azure AD B2C tenant

In Azure Active Directory B2C (Azure AD B2C), a *tenant* represents your organization and is a directory of users. Each Azure AD B2C tenant is distinct and separate from other Azure AD B2C tenants. An Azure AD B2C tenant is different than an Azure Active Directory tenant, which you may already have.

The primary resources you work with in an Azure AD B2C tenant are:

- **Directory** - The *directory* is where Azure AD B2C stores your users' credentials and profile data, as well as your application registrations.
- **Application registrations** - You register your web, mobile, and native applications with Azure AD B2C to enable identity management. Also, any APIs you want to protect with Azure AD B2C.
- **User flows and custom policies** - The built-in (user flows) and fully customizable (custom policies) identity experiences for your applications.
 - Use *user flows* for quick configuration and enablement of common identity tasks like sign up, sign in, and profile editing.
 - Use *custom policies* to enable user experiences not only for the common identity tasks, but also for crafting support for complex identity workflows unique to your organization, customers, employees, partners, and citizens.
- **Identity providers** - Federation settings for:
 - *Social* identity providers like Facebook, LinkedIn, or Twitter that you want to support in your applications.
 - *External* identity providers that support standard identity protocols like OAuth 2.0, OpenID Connect, and more.
 - *Local* accounts that enable users to sign up and sign in with a username (or email address or other ID) and password.
- **Keys** - Add and manage encryption keys for signing and validating tokens.

An Azure AD B2C tenant is the first resource you need to create to get started with Azure AD B2C. Learn how in [Tutorial: Create an Azure Active Directory B2C tenant](#).

Accounts in Azure AD B2C

Azure AD B2C defines several types of user accounts. Azure Active Directory, Azure Active Directory B2B, and Azure Active Directory B2C share these account types.

- **Work account** - Users with work accounts can manage resources in a tenant, and with an administrator role, can also manage tenants. Users with work accounts can create new consumer accounts, reset passwords, block/unblock accounts, and set permissions or assign an account to a security group.
- **Guest account** - External users you invite to your tenant as guests. A typical scenario for inviting a guest user

to your Azure AD B2C tenant is to share administration responsibilities.

- **Consumer account** - Consumer accounts are the accounts created in your Azure AD B2C directory when users complete the sign-up user journey in an application you've registered in your tenant.

The screenshot shows the 'Users - All users' page in the Azure portal. The left sidebar includes links for Home, Users - All users, Deleted users, Password reset, User settings, Activity (Sign-ins, Audit logs, Bulk operation results (Preview)), Troubleshooting + Support (Troubleshoot, New support request), and Documentation. The main area displays a table of users with columns for NAME, USER NAME, USER TYPE, and SOURCE. The users listed are Colby Marble (Member, LinkedIn), Harriett Mosley (Member, Azure Active Directory), Julio Reuter (Member, Google), Katina Knowles (Member, Azure Active Directory), Kelly Shields (Member, Facebook), Rigoberto Dugas (Member, LinkedIn), Roslyn Fry (Member, Federated Azure Active Directory), Warren Boatright (Member, Azure Active Directory), and Young Underwood (Member, Microsoft Account). There are also search and filter options at the top of the table.

Figure: User directory within an Azure AD B2C tenant in the Azure portal

Consumer accounts

With a *consumer* account, users can sign in to the applications that you've secured with Azure AD B2C. Users with consumer accounts can't, however, access Azure resources, for example the Azure portal.

A consumer account can be associated with these identity types:

- **Local** identity, with the username and password stored locally in the Azure AD B2C directory. We often refer to these identities as "local accounts."
- **Social or enterprise** identities, where the identity of the user is managed by a federated identity provider like Facebook, Microsoft, ADFS, or Salesforce.

A user with a consumer account can sign in with multiple identities, for example username, email, employee ID, government ID, and others. A single account can have multiple identities, both local and social.

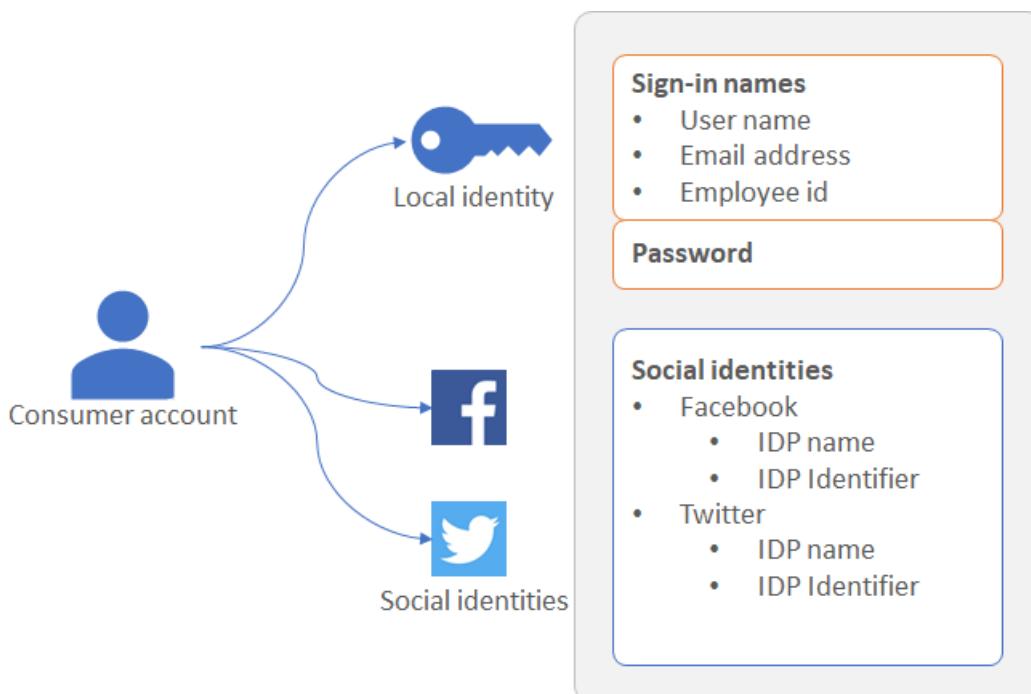


Figure: A single consumer account with multiple identities in Azure AD B2C

Azure AD B2C lets you manage common attributes of consumer account profiles like display name, surname, given name, city, and others. You can also extend the Azure AD schema to store additional information about your users. For example, their country or residency, preferred language, and preferences like whether they want to subscribe to a newsletter or enable multi-factor authentication.

Learn more about the user account types in Azure AD B2C in [Overview of user accounts in Azure Active Directory B2C](#).

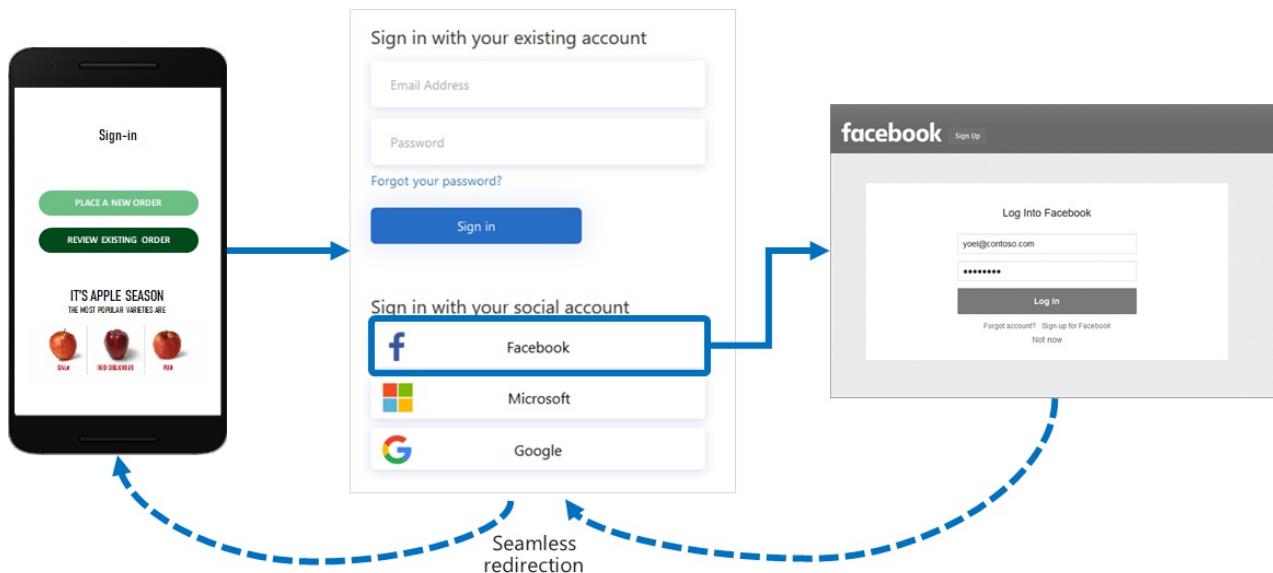
External identity providers

You can configure Azure AD B2C to allow users to sign in to your application with credentials from external social or enterprise identity providers (IdP). Azure AD B2C supports external identity providers like Facebook, Microsoft account, Google, Twitter, and any identity provider that supports OAuth 1.0, OAuth 2.0, OpenID Connect, SAML, or WS-Federation protocols.



With external identity provider federation, you can offer your consumers the ability to sign in with their existing social or enterprise accounts, without having to create a new account just for your application.

On the sign-up or sign-in page, Azure AD B2C presents a list of external identity providers the user can choose for sign-in. Once they select one of the external identity providers, they're taken (redirected) to the selected provider's website to complete the sign in process. After the user successfully signs in, they're returned back to Azure AD B2C for authentication of the account in your application.



To see how to add identity providers in Azure AD B2C, see [Tutorial: Add identity providers to your applications in Azure Active Directory B2C](#).

Identity experiences: user flows or custom policies

The extensible policy framework of Azure AD B2C is its core strength. Policies describe your users' identity experiences such as sign up, sign in, and profile editing.

In Azure AD B2C, there are two primary paths you can take to provide these identity experiences: user flows and custom policies.

- **User flows** are predefined, built-in, configurable policies that we provide so you can create sign-up, sign-in, and policy editing experiences in minutes.
- **Custom policies** enable you to create your own user journeys for complex identity experience scenarios.

Both user flows and custom policies are powered by the *Identity Experience Framework*, Azure AD B2C's policy orchestration engine.

User flow

To help you quickly set up the most common identity tasks, the Azure portal includes several predefined and configurable policies called *user flows*.

You can configure user flow settings like these to control identity experience behaviors in your applications:

- Account types used for sign-in, such as social accounts like a Facebook, or local accounts that use an email address and password for sign-in
- Attributes to be collected from the consumer, such as first name, postal code, or country of residency
- Azure Multi-Factor Authentication (MFA)
- Customization of the user interface
- Set of claims in a token that your application receives after the user completes the user flow
- Session management
- ...and more.

Most common identity scenarios for the majority of mobile, web, and single-page applications can be defined and implemented effectively with user flows. We recommend that you use the built-in user flows unless you have complex user journey scenarios that require the full flexibility of custom policies.

Learn more about user flows in [User flows in Azure Active Directory B2C](#).

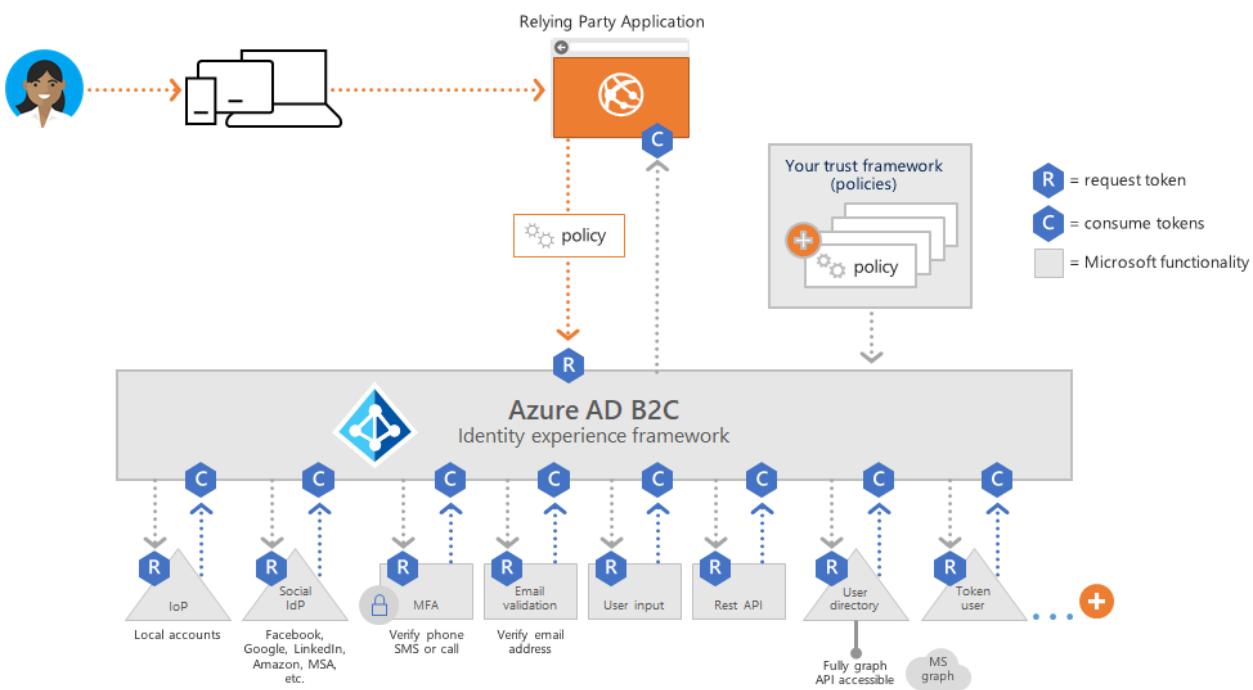
Custom policy

Custom policies unlock access to the full power of the Identity Experience Framework (IEF) orchestration engine. With custom policies, you can leverage IEF to build almost any authentication, user registration, or profile editing experience that you can imagine.

The Identity Experience Framework gives you the ability to construct user journeys with any combination of steps. For example:

- Federate with other identity providers
- First- and third-party multi-factor authentication (MFA) challenges
- Collect any user input
- Integrate with external systems using REST API communication

Each such user journey is defined by a policy, and you can build as many or as few policies as you need to enable the best user experience for your organization.



A custom policy is defined by several XML files that refer to each other in a hierarchical chain. The XML elements define the claims schema, claims transformations, content definitions, claims providers, technical profiles, user journey orchestration steps, and other aspects of the identity experience.

The powerful flexibility of custom policies is most appropriate for when you need to build complex identity scenarios. Developers configuring custom policies must define the trusted relationships in careful detail to include metadata endpoints, exact claims exchange definitions, and configure secrets, keys, and certificates as needed by each identity provider.

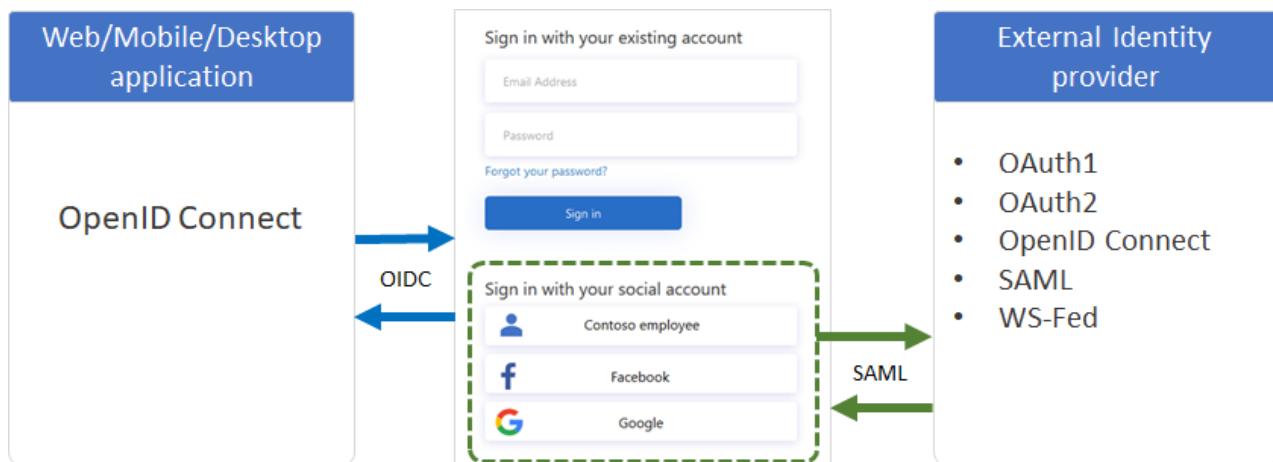
Learn more about custom policies in [Custom policies in Azure Active Directory B2C](#).

Protocols and tokens

Azure AD B2C supports the [OpenID Connect](#) and [OAuth 2.0 protocols](#) for user journeys. In the Azure AD B2C implementation of OpenID Connect, your application starts the user journey by issuing authentication requests to Azure AD B2C.

The result of a request to Azure AD B2C is a security token, such as an [ID token](#) or [access token](#). This security token defines the user's identity. Tokens are received from Azure AD B2C endpoints like the `/token` or `/authorize` endpoint. With these tokens, you can access claims that can be used to validate an identity and allow access to secure resources.

For external identities, Azure AD B2C supports federation with any OAuth 1.0, OAuth 2.0, OpenID Connect, SAML, and WS-Fed identity provider.

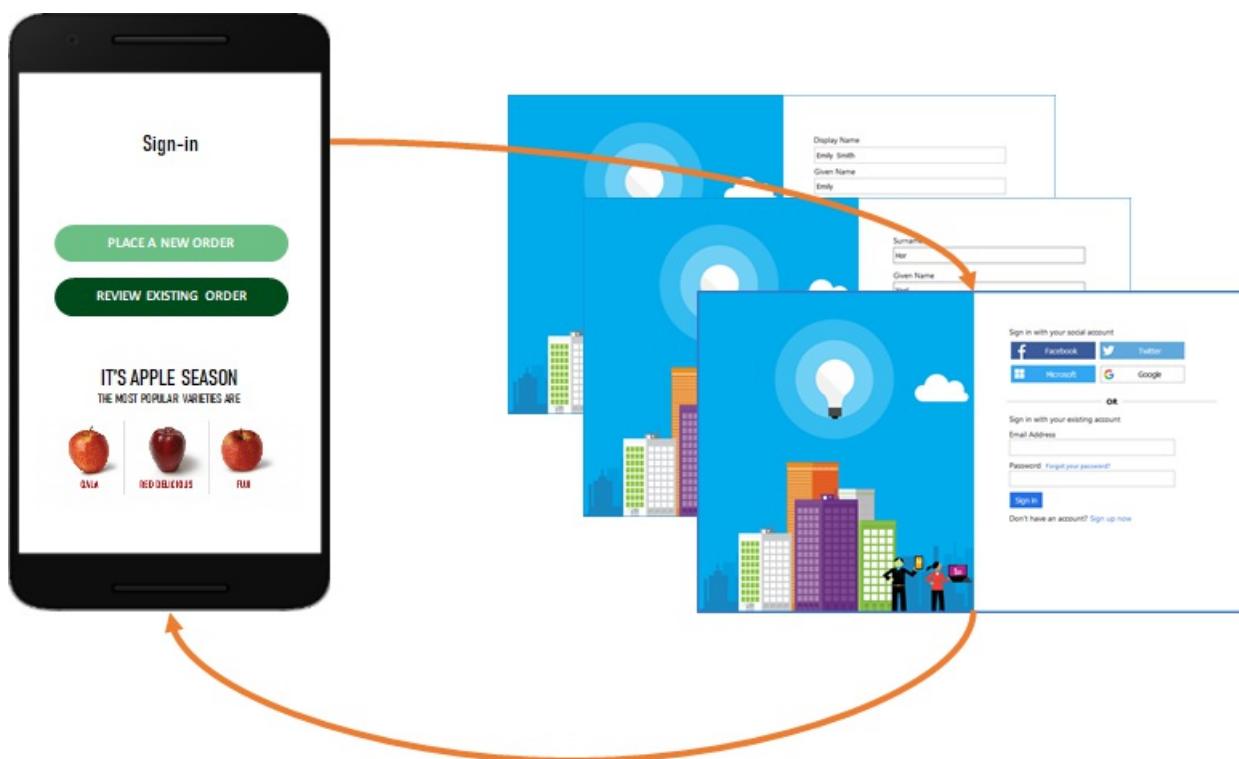


The preceding diagram shows how Azure AD B2C can communicate using variety of protocols within the same authentication flow:

1. The relying party application initiates an authorization request to Azure AD B2C using OpenID Connect.
2. When a user of the application chooses to sign in using an external identity provider that uses the SAML protocol, Azure AD B2C invokes the SAML protocol to communicate with that identity provider.
3. After the user completes the sign-in operation with the external identity provider, Azure AD B2C then returns the token to the relying party application using OpenID Connect.

Application integration

When a user wants to sign in to your application, whether it's a web, mobile, desktop, or single-page application (SPA), the application initiates an authorization request to a user flow- or custom policy-provided endpoint. The user flow or custom policy defines and controls the user's experience. When they complete a user flow, for example the *sign-up* or *sign-in* flow, Azure AD B2C generates a token, then redirects the user back to your application.

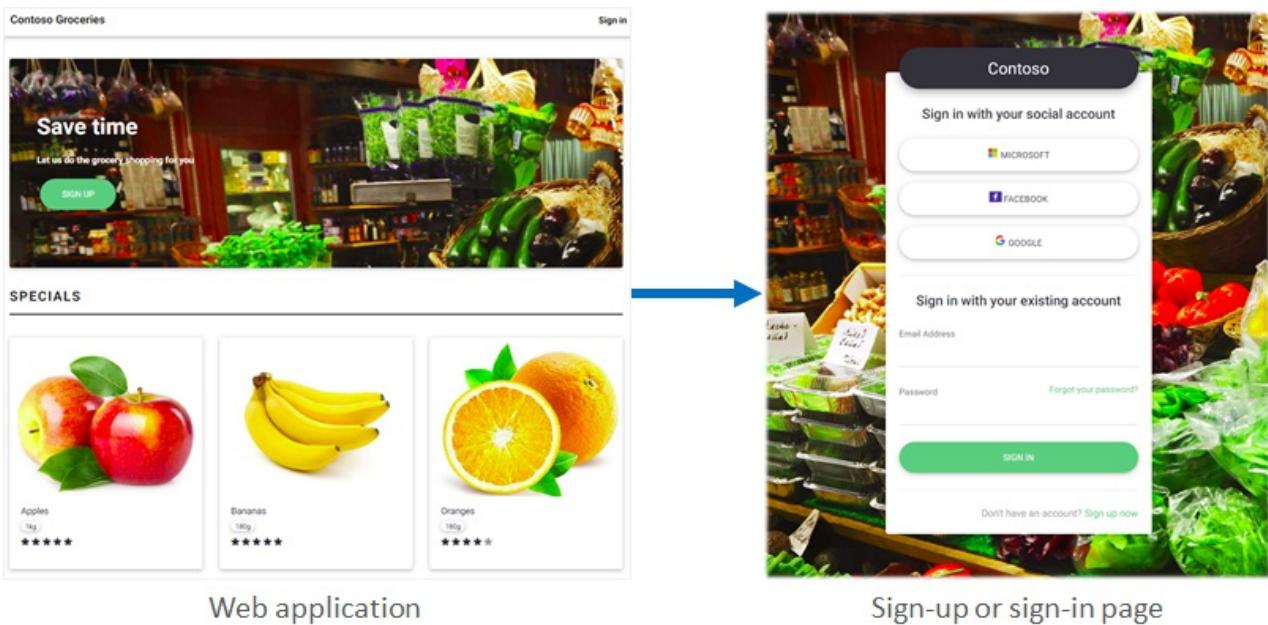


Multiple applications can use the same user flow or custom policy. A single application can use multiple user flows or custom policies.

For example, to sign in to an application, the application uses the *sign up* or *sign in* user flow. After the user has signed in, they may want to edit their profile, so the application initiates another authorization request, this time using the *profile edit* user flow.

Seamless user experiences

In Azure AD B2C, you can craft your users' identity experiences so that the pages they're shown blend seamlessly with the look and feel of your brand. You get nearly full control of the HTML and CSS content presented to your users when they proceed through your application's identity journeys. With this flexibility, you can maintain brand and visual consistency between your application and Azure AD B2C.



For information on UI customization, see [About user interface customization in Azure Active Directory B2C](#).

Localization

Language customization in Azure AD B2C allows you to accommodate different languages to suit your customer needs. Microsoft provides the translations for 36 languages, but you can also provide your own translations for any language. Even if your experience is provided for only a single language, you can customize any text on the pages.

Sign in with your existing account <input type="text"/> <input type="password"/> Forgot your password? <input type="button" value="Sign in"/>	Iniciar sesión con su cuenta existente <input type="text"/> <input type="password"/> ¿Olvidó su contraseña? <input type="button" value="Iniciar sesión"/>	अपने मोजूदा खाते के साथ साइन इन करें <input type="text"/> <input type="text"/> अपना पासवर्ड भूल गए हैं? <input type="button" value="साइन इन करें"/>
Sign in with your social account <div style="display: flex; justify-content: space-around;"> Microsoft Google Facebook </div>	Iniciar sesión con su cuenta social <div style="display: flex; justify-content: space-around;"> Microsoft Google Facebook </div>	अपने सोशल खाते के साथ साइन इन करें <div style="display: flex; justify-content: space-around;"> Microsoft Google Facebook </div>

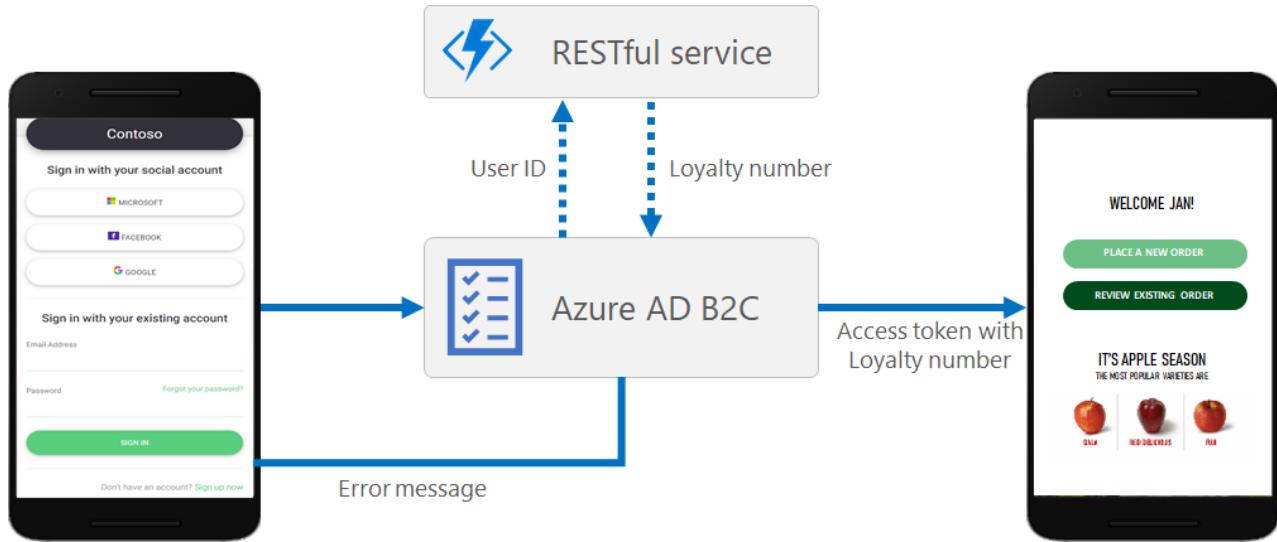
See how localization works in [Language customization in Azure Active Directory B2C](#).

Add your own business logic

If you choose to use custom policies, you can integrate with a RESTful API in a user journey to add your own business logic to the journey. For example, Azure AD B2C can exchange data with a RESTful service to:

- Display custom user-friendly error messages.
- Validate user input to prevent malformed data from persisting in your user directory. For example, you can modify the data entered by the user, such as capitalizing their first name if they entered it in all lowercase.
- Enrich user data by further integrating with your corporate line-of-business application.
- Using RESTful calls, you can send push notifications, update corporate databases, run a user migration process, manage permissions, audit databases, and more.

Loyalty programs are another scenario enabled by Azure AD B2C's support for calling REST APIs. For example, your RESTful service can receive a user's email address, query your customer database, then return the user's loyalty number to Azure AD B2C. The return data can be stored in the user's directory account in Azure AD B2C, then be further evaluated in subsequent steps in the policy, or be included in the access token.



You can add a REST API call at any step in the user journey defined by a custom policy. For example, you can call a REST API:

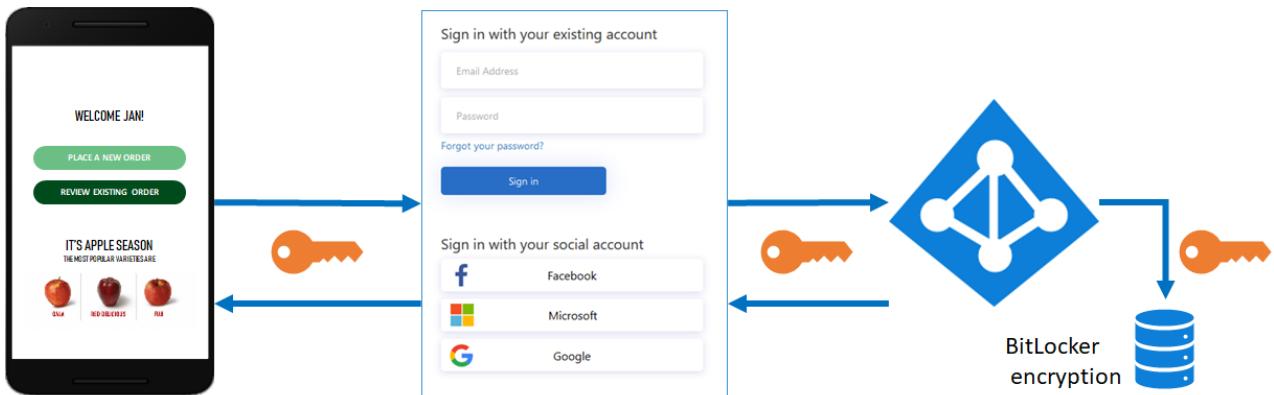
- During sign-in, just before Azure AD B2C validates the credentials
- Immediately after sign-in
- Before Azure AD B2C creates a new account in the directory
- After Azure AD B2C creates a new account in the directory
- Before Azure AD B2C issues an access token

To see how to use custom policies for RESTful API integration in Azure AD B2C, see [Integrate REST API claims exchanges in your Azure AD B2C user journey](#).

Protect customer identities

Azure AD B2C complies with the security, privacy, and other commitments described in the [Microsoft Azure Trust Center](#).

Sessions are modeled as encrypted data, with the decryption key known only to the Azure AD B2C Security Token Service. A strong encryption algorithm, AES-192, is used. All communication paths are protected with TLS for confidentiality and integrity. Our Security Token Service uses an Extended Validation (EV) certificate for TLS. In general, the Security Token Service mitigates cross-site scripting (XSS) attacks by not rendering untrusted input.



Data in used



Data in transit



Data at rest

Access to user data

Azure AD B2C tenants share many characteristics with enterprise Azure Active Directory tenants used for employees and partners. Shared aspects include mechanisms for viewing administrative roles, assigning roles, and auditing activities.

You can assign roles to control who can perform certain administrative actions in Azure AD B2C, including:

- Create and manage all aspects of user flows
- Create and manage the attribute schema available to all user flows
- Configure identity providers for use in direct federation
- Create and manage trust framework policies in the Identity Experience Framework (custom policies)
- Manage secrets for federation and encryption in the Identity Experience Framework (custom policies)

For more information about Azure AD roles, including Azure AD B2C administration role support, see [Administrator role permissions in Azure Active Directory](#).

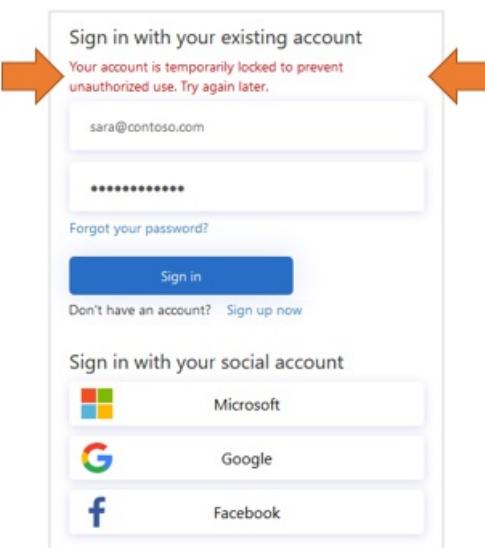
Multi-factor authentication (MFA)

Azure AD B2C multi-factor authentication (MFA) helps safeguard access to data and applications while maintaining simplicity for your users. It provides additional security by requiring a second form of authentication, and delivers strong authentication by offering a range of easy-to-use authentication methods. Your users may or may not be challenged for MFA based on configuration decisions that you can make as an administrator.

See how to enable MFA in user flows in [Enable multi-factor authentication in Azure Active Directory B2C](#).

Smart account lockout

To prevent brute-force password guessing attempts, Azure AD B2C uses a sophisticated strategy to lock accounts based on the IP of the request, the passwords entered, and several other factors. The duration of the lockout is automatically increased based on risk and the number of attempts.



For more information about managing password protection settings, see [Manage threats to resources and data in Azure Active Directory B2C](#).

Password complexity

During sign up or password reset, your users must supply a password that meets complexity rules. By default, Azure AD B2C enforces a strong password policy. Azure AD B2C also provides configuration options for specifying the complexity requirements of the passwords your customers use.

You can configure password complexity requirements in both [user flows](#) and [custom policies](#).

Auditing and logs

Azure AD B2C emits audit logs containing activity information about its resources, issued tokens, and administrator access. You can use these audit logs to understand platform activity and diagnose issues. Audit log entries are available soon after the activity that generated the event occurs.

In an audit log, which is available for your Azure AD B2C tenant or for a particular user, you can find information including:

- Activities concerning the authorization of a user to access B2C resources (for example, an administrator accessing a list of B2C policies)
- Activities related to directory attributes retrieved when an administrator signs in using the Azure portal
- Create, read, update, and delete (CRUD) operations on B2C applications
- CRUD operations on keys stored in a B2C key container
- CRUD operations on B2C resources (for example, policies and identity providers)
- Validation of user credentials and token issuance

Home > Azure AD B2C > Users - All users > Lisa Wood - Audit logs

Lisa Wood - Audit logs

User

Manage
Columns Refresh Download

Service: All
Category: All
Activity: All
Status: All
Target: Enter target name or upn

Initiated By (Actor): Enter actor name or upn
Date: Last 7 days Show dates as: Local UTC

Apply
Reset

DATE	SERVICE	CATEGORY	ACTIVITY	STATUS	TARGET(S)	INITIATED BY (ACTOR)
3/18/2019, 3:09:47 PM	B2C	Authentication	Verify phone number	Success	6ebce829-74bf-460f-8677-...	0239a9cc-309c-4d41-87f1-...
3/18/2019, 3:09:31 PM	B2C	Authentication	Send SMS to verify phone number	Success	6ebce829-74bf-460f-8677-...	0239a9cc-309c-4d41-87f1-...
3/18/2019, 3:09:25 PM	B2C	Authentication	Validate local account credentials	Success	a42005a0-72db-491d-a76a...	0239a9cc-309c-4d41-87f1-...
3/18/2019, 3:07:53 PM	B2C	Authentication	Validate local account credentials	Success	a42005a0-72db-491d-a76a...	0239a9cc-309c-4d41-87f1-...
3/18/2019, 3:07:04 PM	B2C	Authentication	Verify phone number	Success	6ebce829-74bf-460f-8677-...	0239a9cc-309c-4d41-87f1-...
3/18/2019, 3:06:46 PM	B2C	Authentication	Send SMS to verify phone number	Success	6ebce829-74bf-460f-8677-...	0239a9cc-309c-4d41-87f1-...

Details

Activity	Target(s)	Modified Properties
ACTIVITY	INITIATED BY (ACTOR)	ADDITIONAL DETAILS
DATE	3/18/2019, 3:09:25 PM	Type Application TenantId sunflowersdemo.onmicrosoft.com
ACTIVITY TYPE	Validate local account credentials	PolicyId B2C_1_profile_edit
CORRELATION ID	8174cfdf-df35-4bbe-b2d5-9d4e948c9e5c	APP ID 2ed4b543-8aa7-40b7-9ba8-9cf42aeeefaf ApplicationId 0239a9cc-309c-4d41-87f1-31288feb2e82
CATEGORY	Authentication	Service Principal ID Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0
STATUS	Success	Service Principal Name 0239a9cc-309c-4d41-87f1-31288feb2e82 Client LocalAccountUsername
STATUS REASON	N/A	ClientIpAddress

For additional details on audit logs, see [Accessing Azure AD B2C audit logs](#).

Usage insights

Azure AD B2C allows you to discover when people sign up or sign in to your web app, where your users are located, and what browsers and operating systems they use. By integrating Azure Application Insights into Azure AD B2C by using custom policies, you can gain insight into how people sign up, sign in, reset their password or edit their profile. With such knowledge, you can make data-driven decisions for your upcoming development cycles.

Find out more about usage analytics in [Track user behavior in Azure Active Directory B2C using Application Insights](#).

Next steps

Now that you have deeper view into the features and technical aspects of Azure Active Directory B2C, get started with the service by creating a B2C tenant:

[Tutorial: Create an Azure Active Directory B2C tenant >](#)

Quickstart: Set up sign in for an ASP.NET application using Azure Active Directory B2C

1/28/2020 • 3 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) provides cloud identity management to keep your application, business, and customers protected. Azure AD B2C enables your applications to authenticate to social accounts and enterprise accounts using open standard protocols. In this quickstart, you use an ASP.NET application to sign in using a social identity provider and call an Azure AD B2C protected web API.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Visual Studio 2019](#) with the **ASP.NET and web development** workload.
- A social account from Facebook, Google, or Microsoft.
- [Download a zip file](#) or clone the sample web application from GitHub.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnet-webapp-and-webapi.git
```

There are two projects are in the sample solution:

- **TaskWebApp** - A web application that creates and edits a task list. The web application uses the **sign-up or sign-in** user flow to sign up or sign in users.
- **TaskService** - A web API that supports the create, read, update, and delete task list functionality. The web API is protected by Azure AD B2C and called by the web application.

Run the application in Visual Studio

1. In the sample application project folder, open the **B2C-WebAPI-DotNet.sln** solution in Visual Studio.
2. For this quickstart, you run both the **TaskWebApp** and **TaskService** projects at the same time. Right-click the **B2C-WebAPI-DotNet** solution in Solution Explorer, and then select **Set StartUp Projects**.
3. Select **Multiple startup projects** and change the **Action** for both projects to **Start**.
4. Click **OK**.
5. Press **F5** to debug both applications. Each application opens in its own browser tab:
 - `https://localhost:44316/` - The ASP.NET web application. You interact directly with this application in the quickstart.
 - `https://localhost:44332/` - The web API that's called by the ASP.NET web application.

Sign in using your account

1. Click **Sign up / Sign in** in the ASP.NET web application to start the workflow.

The screenshot shows a web browser window with the URL <https://localhost:44316>. The page has a dark header bar with the text "Application name", "Home", "Claims", "To-Do List", and "Sign up / Sign in". The main content area features a large "ASP.NET" logo and the text "ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript." Below this is a blue button labeled "Learn more »".

The sample supports several sign-up options including using a social identity provider or creating a local account using an email address. For this quickstart, use a social identity provider account from either Facebook, Google, or Microsoft.

2. Azure AD B2C presents a sign-in page for a fictitious company called Fabrikam for the sample web application. To sign up using a social identity provider, click the button of the identity provider you want to use.

The screenshot shows a browser window titled "Sign up or sign in" with the URL fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_susi/oauth2/v2.0/auth.... The page features a blue background with a lightbulb icon and a city skyline illustration. It displays three sign-in options: "Sign in with your social account" (Facebook, Google, Microsoft), "Sign in with your existing account" (Email Address, Password), and a "Sign in" button. At the bottom, there is a link "Don't have an account? [Sign up now](#)".

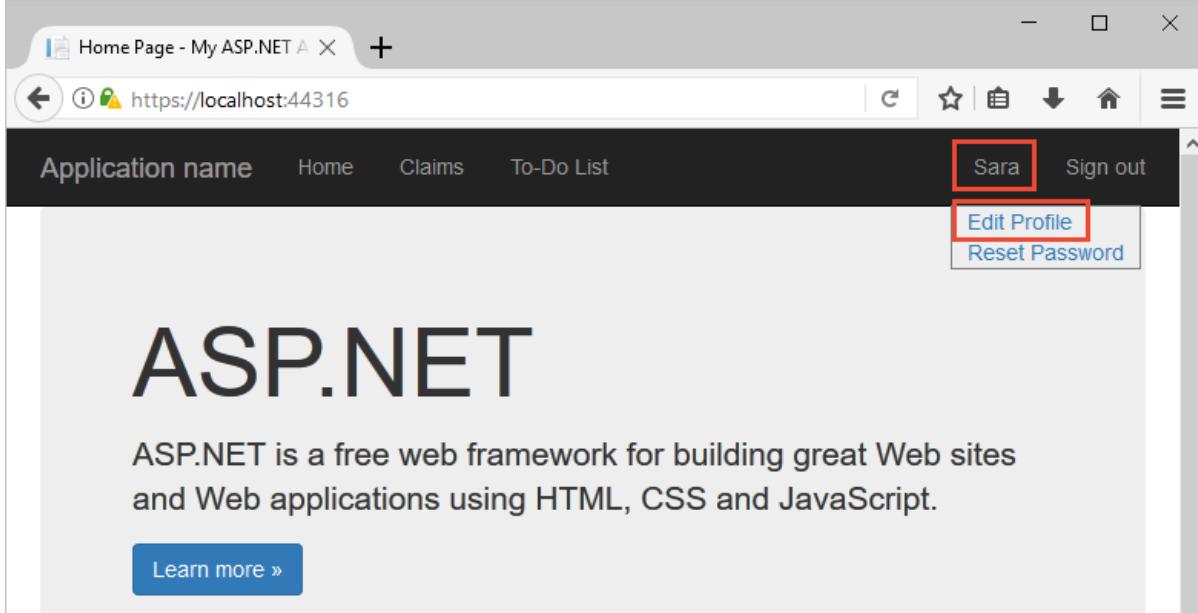
You authenticate (sign in) using your social account credentials and authorize the application to read information from your social account. By granting access, the application can retrieve profile information from the social account such as your name and city.

3. Finish the sign-in process for the identity provider.

Edit your profile

Azure Active Directory B2C provides functionality to allow users to update their profiles. The sample web app uses an Azure AD B2C edit profile user flow for the workflow.

1. In the application menu bar, click your profile name and select **Edit profile** to edit the profile you created.

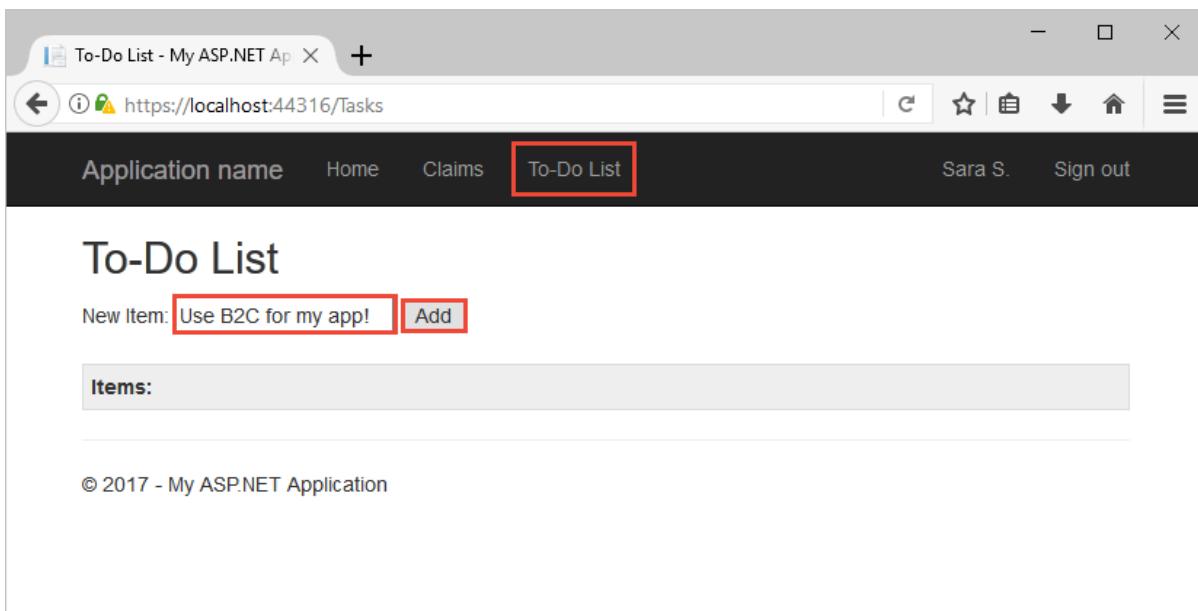


2. Change your **Display name** or **City**, and then click **Continue** to update your profile.

The change is displayed in the upper right portion of the web application's home page.

Access a protected API resource

1. Click **To-Do List** to enter and modify your to-do list items.
2. Enter text in the **New Item** text box. Click **Add** to call the Azure AD B2C protected web API that adds a to-do list item.



The ASP.NET web application includes an Azure AD access token in the request to the protected web API resource to perform operations on the user's to-do list items.

You've successfully used your Azure AD B2C user account to make an authorized call an Azure AD B2C protected web API.

Clean up resources

You can use your Azure AD B2C tenant if you plan to try other Azure AD B2C quickstarts or tutorials. When no longer needed, you can [delete your Azure AD B2C tenant](#).

Next steps

In this quickstart, you used a sample ASP.NET application to:

- Sign in with a custom login page
- Sign in with a social identity provider
- Create an Azure AD B2C account
- Call a web API protected by Azure AD B2C

Get started creating your own Azure AD B2C tenant.

[Create an Azure Active Directory B2C tenant in the Azure portal](#)

Quickstart: Set up sign-in for a desktop app using Azure Active Directory B2C

1/29/2020 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) provides cloud identity management to keep your application, business, and customers protected. Azure AD B2C enables your applications to authenticate to social accounts and enterprise accounts using open standard protocols. In this quickstart, you use a Windows Presentation Foundation (WPF) desktop application to sign in using a social identity provider and call an Azure AD B2C protected web API.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Visual Studio 2019](#) with the **ASP.NET and web development** workload.
- A social account from either Facebook, Google, or Microsoft.
- [Download a zip file](#) or clone the [Azure-Samples/active-directory-b2c-dotnet-desktop](#) repository from GitHub.

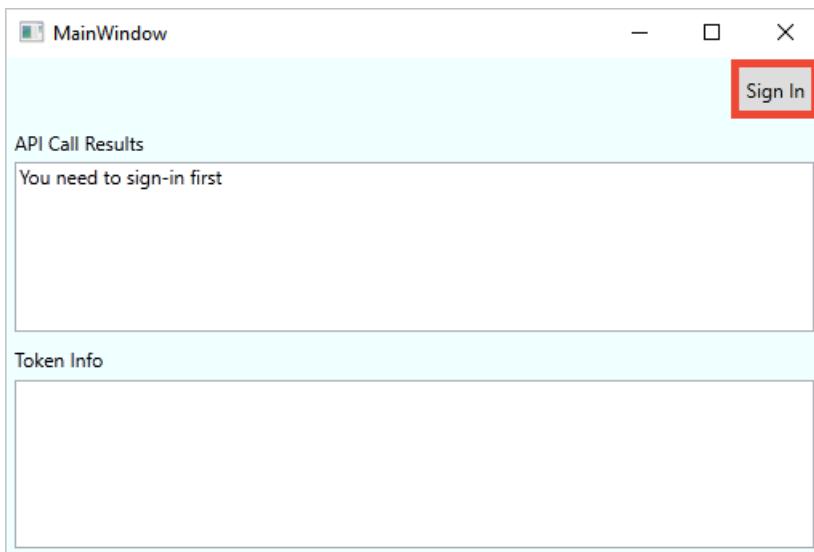
```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnet-desktop.git
```

Run the application in Visual Studio

1. In the sample application project folder, open the **active-directory-b2c-wpf.sln** solution in Visual Studio.
2. Press **F5** to debug the application.

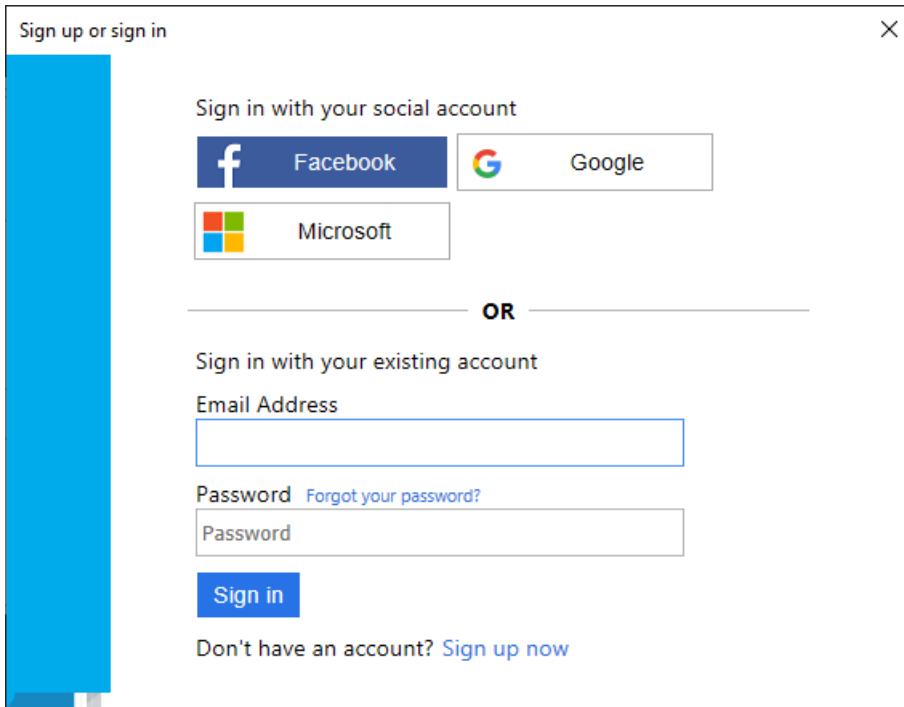
Sign in using your account

1. Click **Sign in** to start the **Sign Up or Sign In** workflow.



The sample supports several sign-up options. These options include using a social identity provider or creating a local account using an email address. For this quickstart, use a social identity provider account from either Facebook, Google, or Microsoft.

2. Azure AD B2C presents a sign-in page for a fictitious company called Fabrikam for the sample web application. To sign up using a social identity provider, click the button of the identity provider you want to use.



You authenticate (sign in) using your social account credentials and authorize the application to read information from your social account. By granting access, the application can retrieve profile information from the social account such as your name and city.

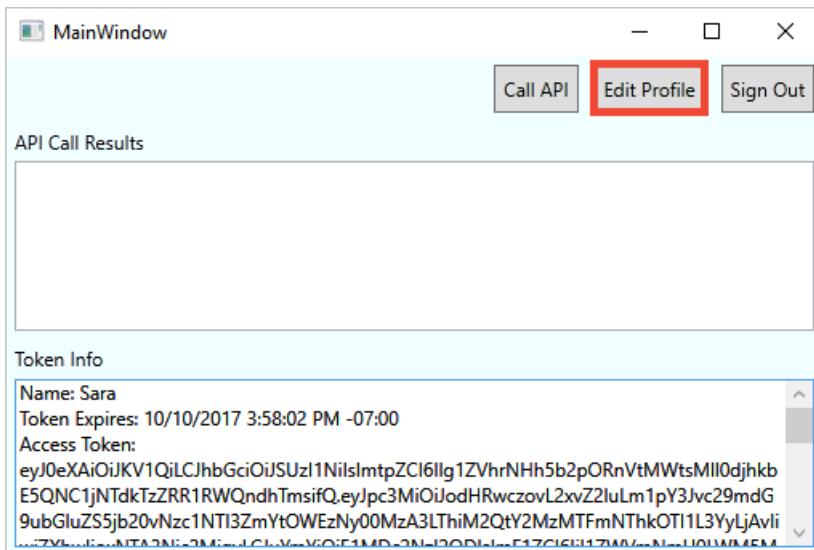
3. Finish the sign-in process for the identity provider.

Your new account profile details are pre-populated with information from your social account.

Edit your profile

Azure AD B2C provides functionality to allow users to update their profiles. The sample web app uses an Azure AD B2C edit profile user flow for the workflow.

1. In the application menu bar, click **Edit profile** to edit the profile you created.



2. Choose the identity provider associated with the account you created. For example, if you used Facebook as the identity provider when you created your account, choose Facebook to modify the associated profile

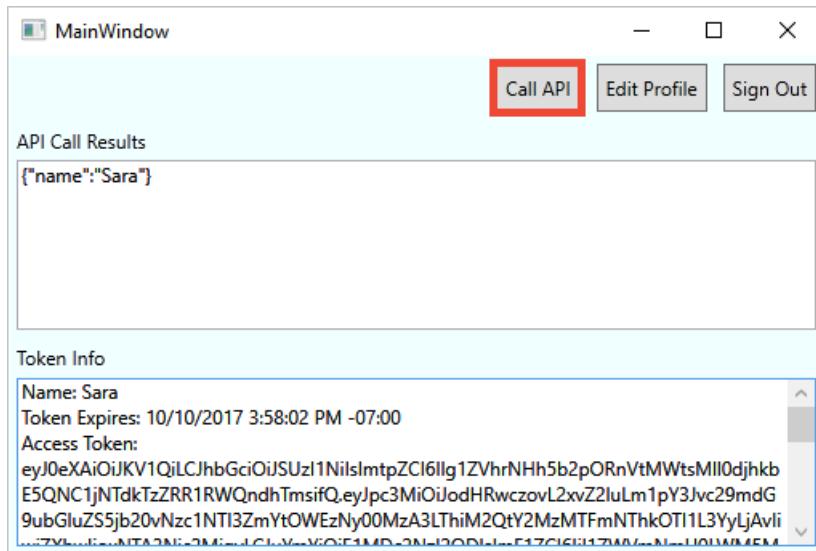
details.

3. Change your **Display name** or **City**, and then click **Continue**.

A new access token is displayed in the *Token info* text box. If you want to verify the changes to your profile, copy and paste the access token into the token decoder <https://jwt.ms>.

Access a protected API resource

Click **Call API** to make a request to the protected resource.



The application includes the Azure AD access token in the request to the protected web API resource. The web API sends back the display name contained in the access token.

You've successfully used your Azure AD B2C user account to make an authorized call an Azure AD B2C protected web API.

Clean up resources

You can use your Azure AD B2C tenant if you plan to try other Azure AD B2C quickstarts or tutorials. When no longer needed, you can [delete your Azure AD B2C tenant](#).

Next steps

In this quickstart, you used a sample desktop application to:

- Sign in with a custom login page
- Sign in with a social identity provider
- Create an Azure AD B2C account
- Call a web API protected by Azure AD B2C

Get started creating your own Azure AD B2C tenant.

[Create an Azure Active Directory B2C tenant in the Azure portal](#)

Quickstart: Set up sign-in for a single-page app using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) provides cloud identity management to keep your application, business, and customers protected. Azure AD B2C enables your applications to authenticate to social accounts, and enterprise accounts using open standard protocols. In this quickstart, you use a single-page application to sign in using a social identity provider and call an Azure AD B2C protected web API.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Visual Studio 2019](#) with the **ASP.NET and web development** workload
- [Node.js](#)
- Social account from Facebook, Google, or Microsoft
- Code sample from GitHub: [active-directory-b2c-javascript-msal-singlepageapp](#)

You can [download the zip archive](#) or clone the repository:

```
git clone https://github.com/Azure-Samples/active-directory-b2c-javascript-msal-singlepageapp.git
```

Run the application

1. Start the server by running the following commands from the Node.js command prompt:

```
cd active-directory-b2c-javascript-msal-singlepageapp  
npm install && npm update  
node server.js
```

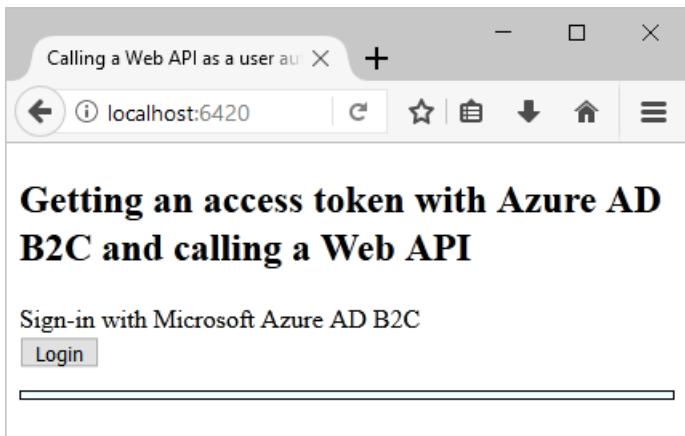
Server.js outputs the port number it's listening on at localhost.

```
Listening on port 6420...
```

2. Browse to the URL of the application. For example, <http://localhost:6420>.

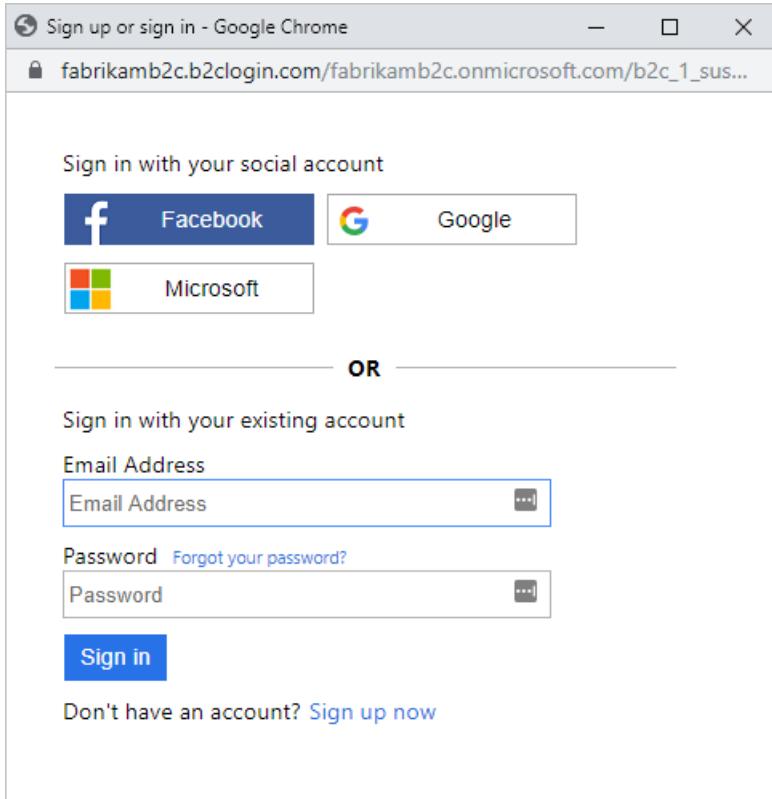
Sign in using your account

1. Click **Login** to start the workflow.



The sample supports several sign-up options including using a social identity provider or creating a local account using an email address. For this quickstart, use a social identity provider account from either Facebook, Google, or Microsoft.

2. Azure AD B2C presents a sign-in page for a fictitious company called Fabrikam for the sample web application. To sign up using a social identity provider, click the button of the identity provider you want to use.



You authenticate (sign in) using your social account credentials and authorize the application to read information from your social account. By granting access, the application can retrieve profile information from the social account such as your name and city.

3. Finish the sign-in process for the identity provider.

Access a protected API resource

Click **Call Web API** to have your display name returned from the Web API call as a JSON object.



The sample single-page application includes an access token in the request to the protected web API resource.

Clean up resources

You can use your Azure AD B2C tenant if you plan to try other Azure AD B2C quickstarts or tutorials. When no longer needed, you can [delete your Azure AD B2C tenant](#).

Next steps

In this quickstart, you used a sample single-page application to:

- Sign in with a custom login page
- Sign in with a social identity provider
- Create an Azure AD B2C account
- Call a web API protected by Azure AD B2C

Get started creating your own Azure AD B2C tenant.

[Create an Azure Active Directory B2C tenant in the Azure portal](#)

Tutorial: Create an Azure Active Directory B2C tenant

1/23/2020 • 3 minutes to read • [Edit Online](#)

Before your applications can interact with Azure Active Directory B2C (Azure AD B2C), they must be registered in a tenant that you manage.

In this article, you learn how to:

- Create an Azure AD B2C tenant
- Link your tenant to your subscription
- Switch to the directory containing your Azure AD B2C tenant
- Add the Azure AD B2C resource as a **Favorite** in the Azure portal

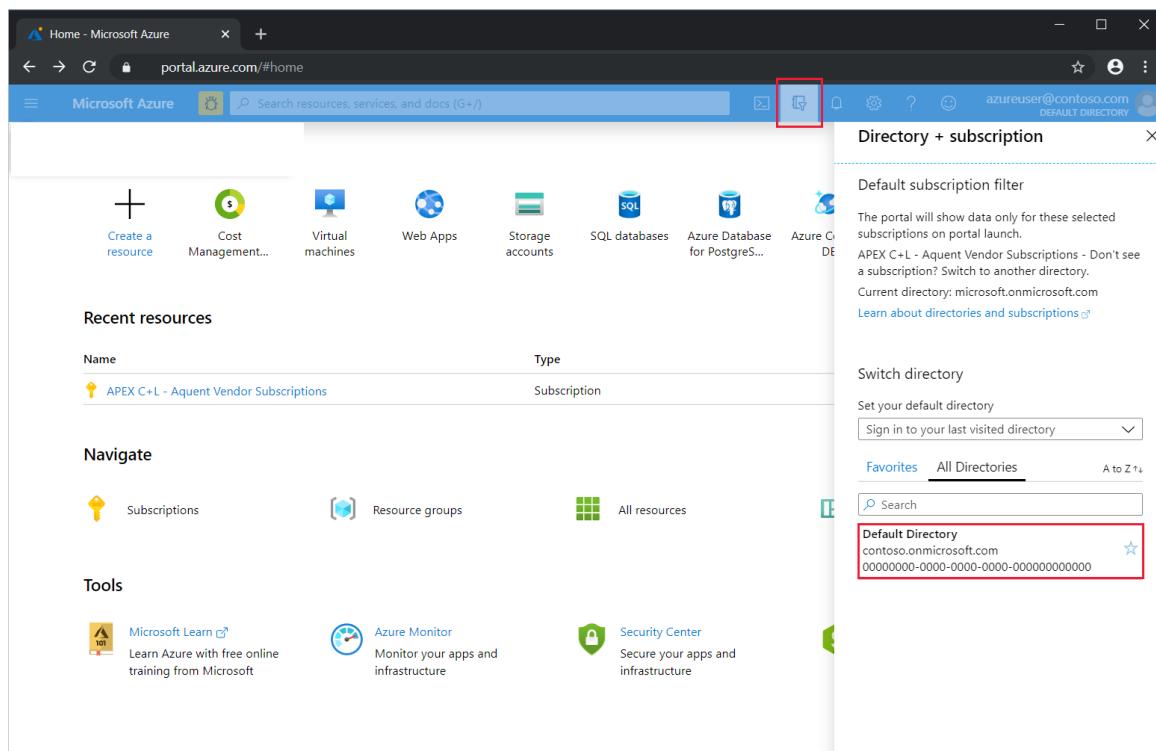
You learn how to register an application in the next tutorial.

If you don't have an Azure subscription, create a [free account](#) before you begin.

Create an Azure AD B2C tenant

1. Sign in to the [Azure portal](#). Sign in with an Azure account that's been assigned at least the [Contributor](#) role within the subscription or a resource group within the subscription.
2. Select the directory that contains your subscription.

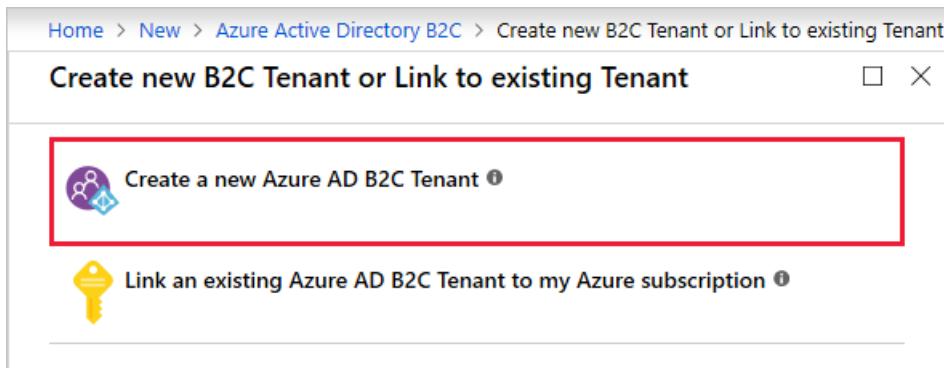
In the Azure portal toolbar, select the **Directory + Subscription** icon, and then select the directory that contains your subscription. This directory is different from the one that will contain your Azure AD B2C tenant.



3. On the Azure portal menu or from the **Home** page, select **Create a resource**.

4. Search for **Azure Active Directory B2C**, and then select **Create**.

5. Select **Create a new Azure AD B2C Tenant**.



6. Enter an **Organization name** and **Initial domain name**. Select the **Country or region** (it can't be changed later), and then select **Create**.

The domain name is used as part of your full tenant domain name. In this example, the tenant name is *contosob2c.onmicrosoft.com*:

A screenshot of the 'Azure AD B2C Create Tenant' page. The URL in the address bar shows the path: ... > Azure Active Directory B2C > Create new B2C Tenant or Link to existing Tenant > Azure AD B2C Create Tenant. The page has several input fields: 'Organization name' (Contoso B2C), 'Initial domain name' (contosob2c), and 'Country or region' (United States). Below the fields, a message says 'Directory creation will take about one minute.' At the bottom, a large blue 'Create' button is highlighted with a red border.

7. Once the tenant creation is complete, select the **Create new B2C Tenant or Link to existing Tenant** link at the top of the tenant creation page.

A screenshot of the 'Azure AD B2C Create Tenant' page, showing the state after tenant creation. The URL in the address bar shows the path: ... > Create new B2C Tenant or Link to existing Tenant > Azure AD B2C Create Tenant. The page displays the same form fields as in step 6: 'Organization name' (Contoso B2C).

8. Select **Link an existing Azure AD B2C Tenant to my Azure subscription**.

Home > New > Azure Active Directory B2C > Create new B2C Tenant or Link to existing Tenant

Create new B2C Tenant or Link to existing Tenant

Create a new Azure AD B2C Tenant

Link an existing Azure AD B2C Tenant to my Azure subscription

9. Select the **Azure AD B2C Tenant** that you created, then select your **Subscription**.

For **Resource group**, select **Create new**. Enter a **Name** for the resource group that will contain the tenant, select the **Resource group location**, and then select **Create**.

Azure AD B2C Resource

* Azure AD B2C Tenant

contosob2c.onmicrosoft.com

Azure AD B2C Resource name

contosob2c.onmicrosoft.com

* Subscription

Contoso Subscription

* Resource group

(New) contosob2cRG

Create new

* Resource group location

East US

Create

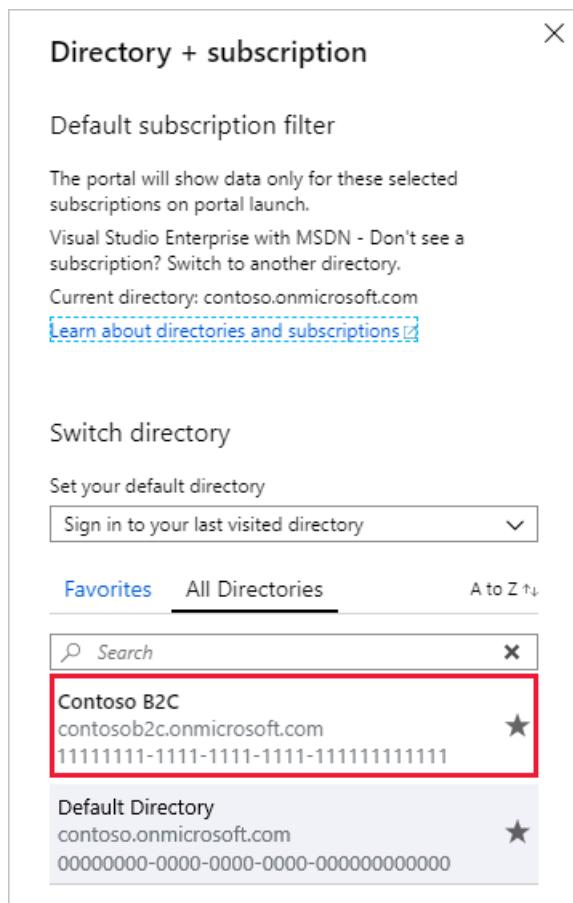
You can link multiple Azure AD B2C tenants to a single Azure subscription for billing purposes.

Select your B2C tenant directory

To start using your new Azure AD B2C tenant, you need to switch to the directory that contains the tenant.

Select the **Directory + subscription** filter in the top menu of the Azure portal, then select the directory that contains your Azure AD B2C tenant.

If at first you don't see your new Azure B2C tenant in the list, refresh your browser window, then select the **Directory + subscription** filter again in the top menu.



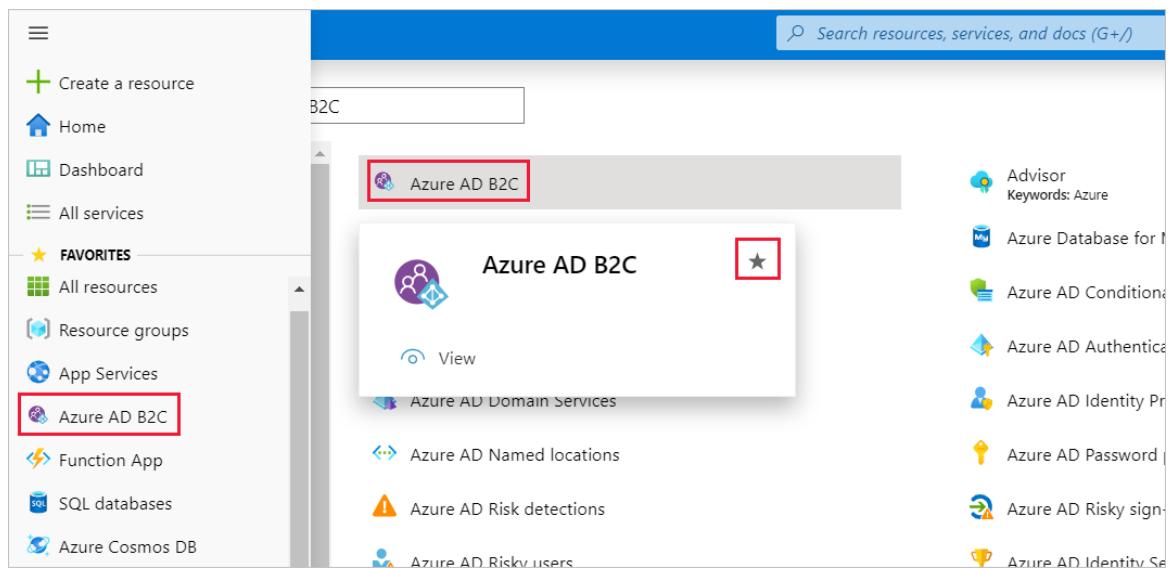
Add Azure AD B2C as a favorite (optional)

This optional step makes it easier to select your Azure AD B2C tenant in the following and all subsequent tutorials.

Instead of searching for *Azure AD B2C* in **All services** every time you want to work with your tenant, you can instead favorite the resource. Then, you can select it from the portal menu's **Favorites** section to quickly browse to your Azure AD B2C tenant.

You only need to perform this operation once. Before performing these steps, make sure you've switched to the directory containing your Azure AD B2C tenant as described in the previous section, [Select your B2C tenant directory](#).

1. Sign in to the [Azure portal](#).
2. In the Azure portal menu, select **All services**.
3. In the **All services** search box, search for **Azure AD B2C**, hover over the search result, and then select the star icon in the tooltip. **Azure AD B2C** now appears in the Azure portal under **Favorites**.
4. If you want to change the position of your new favorite, go to the Azure portal menu, select **Azure AD B2C**, and then drag it up or down to the desired position.



Next steps

In this article, you learned how to:

- Create an Azure AD B2C tenant
- Link your tenant to your subscription
- Switch to the directory containing your Azure AD B2C tenant
- Add the Azure AD B2C resource as a **Favorite** in the Azure portal

Next, learn how to register a web application in your new tenant.

[Register your applications >](#)

Tutorial: Register an application in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

Before your [applications](#) can interact with Azure Active Directory B2C (Azure AD B2C), they must be registered in a tenant that you manage. This tutorial shows you how to register a web application using the Azure portal.

In this article, you learn how to:

- Register a web application
- Create a client secret

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

If you haven't already created your own [Azure AD B2C Tenant](#), create one now. You can use an existing Azure AD B2C tenant.

Register a web application

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *webapp1*.
6. For **Include web app/ web API** and **Allow implicit flow**, select **Yes**.
7. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. For example, you could set it to listen locally at `https://localhost:44316`. If you don't yet know the port number, you can enter a placeholder value and change it later.

For testing purposes like this tutorial you can set it to `https://jwt.ms` which displays the contents of a token for inspection. For this tutorial, set the **Reply URL** to `https://jwt.ms`.

The following restrictions apply to reply URLs:

- The reply URL must begin with the scheme `https`.
- The reply URL is case-sensitive. Its case must match the case of the URL path of your running application. For example, if your application includes as part of its path `.../abc/response-oidc`, do not specify `.../ABC/response-oidc` in the reply URL. Because the web browser treats paths as case-sensitive,

cookies associated with `.../abc/response-oidc` may be excluded if redirected to the case-mismatched `.../ABC/response-oidc` URL.

8. Select **Create** to complete the application registration.

Create a client secret

If your application exchanges an authorization code for an access token, you need to create an application secret.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. In the **Azure AD B2C - Applications** page, select the application you created, for example *webapp1*.
2. Select **Keys** and then select **Generate key**.
3. Select **Save** to view the key. Make note of the **App key** value. You use this value as the application secret in your application's code.

Next steps

In this article, you learned how to:

- Register a web application
- Create a client secret

Next, learn how to create user flows to enable your users to sign up, sign in, and manage their profiles.

[Create user flows in Azure Active Directory B2C >](#)

Tutorial: Create user flows in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

In your applications you may have [user flows](#) that enable users to sign up, sign in, or manage their profile. You can create multiple user flows of different types in your Azure Active Directory B2C (Azure AD B2C) tenant and use them in your applications as needed. User flows can be reused across applications.

In this article, you learn how to:

- Create a sign-up and sign-in user flow
- Create a profile editing user flow
- Create a password reset user flow

This tutorial shows you how to create some recommended user flows by using the Azure portal. If you're looking for information about how to set up a resource owner password credentials (ROPC) flow in your application, see [Configure the resource owner password credentials flow in Azure AD B2C](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

[Register your applications](#) that are part of the user flows you want to create.

Create a sign-up and sign-in user flow

The sign-up and sign-in user flow handles both sign-up and sign-in experiences with a single configuration. Users of your application are led down the right path depending on the context.

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.

The screenshot shows the Microsoft Azure portal interface. In the top right corner, there is a user profile for 'contoso@contoso.com'. Below the header, there's a search bar and a 'Default subscription filter' sidebar. The sidebar shows 'Directory + subscription' set to 'contoso@contoso.com' and includes a note about APEX C+L - Agent Vendor Subscriptions. It also has a link to 'Learn about directories and subscriptions'.

Azure services

- Create a resource
- Virtual machines
- Web Apps
- Storage accounts
- SQL databases
- Azure Database for PostgreSQL
- Azure Cosmos DB
- Kubernetes

Navigate

- Subscriptions
- Resource groups
- All resources

Tools

- Microsoft Learn
- Azure Monitor
- Security Center

Useful links

- Technical Documentation
- Azure Services
- Recent Azure Updates
- Azure Migration Tools
- Find an Azure expert
- Quickstart Center

3. In the Azure portal, search for and select **Azure AD B2C**.
4. Under **Policies**, select **User flows (policies)**, and then select **New user flow**.

The screenshot shows the 'Azure AD B2C - User flows (policies)' page. On the left, there's a navigation menu with 'Overview', 'Manage' (Applications, Identity providers, User attributes, Users), and 'Policies' (User flows (policies), Identity Experience Framework). The 'User flows (policies)' item is highlighted with a red box. On the right, there's a search bar and a table for managing user flows. A large red box highlights the '+ New user flow' button at the top right of the table area.

Azure AD B2C - User flows (policies)
contoso0926Tenant.onmicrosoft.com

+ New user flow

User flow name	User flow type	
Search using user flow name	Filter by user flow type	
NAME	TYPE	MFA
No user flows found.		

5. On the **Recommended** tab, select the **Sign up and sign in** user flow.

Home > Azure AD B2C - User flows (policies) > Create a user flow

Create a user flow

Select a user flow type

A user flow is a series of pages for your users to interact with to sign up or sign into their account. Choose one to start with and you can create multiple user flows to define your entire authentication experience for your app. [Learn more about user flow types.](#)

Recommended [Preview](#) [All](#)

Recommended for most applications.

Sign up and sign in Lets a user register for or log into their account	Profile editing Lets the user configure their user attributes	Password reset Allows a user to choose a new password after verifying their email
--	---	---

6. Enter a **Name** for the user flow. For example, *signupsignin1*.

7. For **Identity providers**, select **Email signup**.

Home > Azure AD B2C - User flows (policies) > Create a user flow > Create

Create

Sign up and sign in

[Select a different type of user flow](#)

Get started with your user flow with a few basic selections. Don't worry about getting everything right here, you can modify your user flow after you've created it.

1. Name *

The unique string used to identify this user flow in requests to Azure AD B2C. This cannot be changed after a user flow has been created.

* B2C_1_ 

2. Identity providers *

Identity providers are the different types of accounts your users can use to log into your application. You need to select at least one for a valid user flow and you can add more in the Identity providers section for your directory. [Learn more about identity providers.](#)

Please select at least one identity provider

 Email signup

3. Multifactor authentication

Enabling multifactor authentication (MFA) requires your users to verify their identity with a second factor before allowing them into your application. [Learn more about multifactor authentication.](#)

Multifactor authentication [Enabled](#) [Disabled](#)

8. For **User attributes and claims**, choose the claims and attributes that you want to collect and send from the user during sign-up. For example, select **Show more**, and then choose attributes and claims for **Country/Region**, **Display Name**, and **Postal Code**. Click **OK**.

Home > Azure AD B2C - User flows

Create

Create

Sign up and sign in

[Select a different type of user flow](#)

Get started with your user flow with a template or create a new user flow after you've created it.

1. Name *

The unique string used to identify this user flow. It must be unique across all user flows in your directory.

* B2C_1_signupsignin1

2. Identity providers *

Identity providers are the different types of accounts that can be used for a valid user flow and you can add more later.

Please select at least one identity provider.

Email signup

3. Multifactor authentication

Enabling multifactor authentication (MFA) adds another layer of security to your application. [Learn more about MFA](#)

Multifactor authentication [Enabled](#)

4. User attributes and claims

User attributes are values collected on sign up. Claims are values about the user returned to the application in the token. You can create custom attributes for use in your directory.

	Collect attribute	Return claim
City <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Country/Region <small>i</small>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Display Name <small>i</small>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Email Address <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Email Addresses <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Given Name <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Identity Provider <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Identity Provider Access Token <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Job Title <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Postal Code <small>i</small>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
State/Province <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Street Address <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
Surname <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
User is new <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>
User's Object ID <small>i</small>	<input type="checkbox"/>	<input type="checkbox"/>

[Show more...](#)

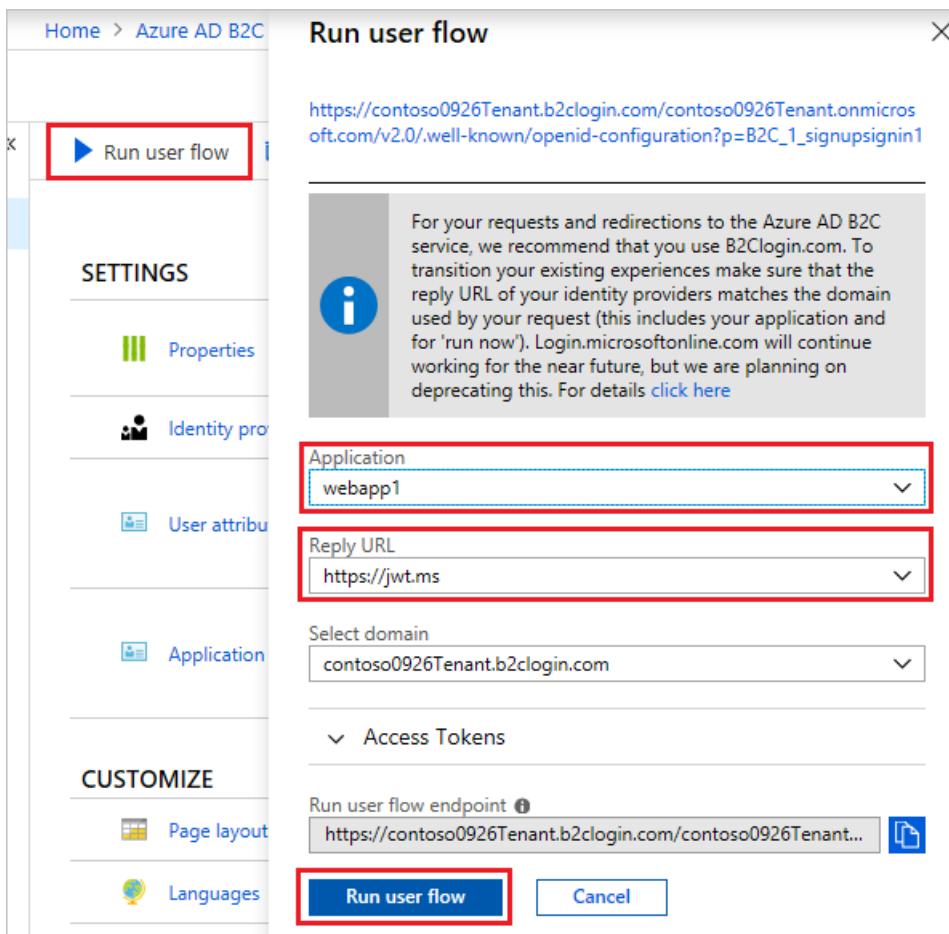
[Create](#)

Ok

9. Click **Create** to add the user flow. A prefix of *B2C_1* is automatically appended to the name.

Test the user flow

1. Select the user flow you created to open its overview page, then select **Run user flow**.
2. For **Application**, select the web application named *webapp1* that you previously registered. The **Reply URL** should show <https://jwt.ms>.
3. Click **Run user flow**, and then select **Sign up now**.



4. Enter a valid email address, click **Send verification code**, enter the verification code that you receive, then select **Verify code**.
5. Enter a new password and confirm the password.
6. Select your country and region, enter the name that you want displayed, enter a postal code, and then click **Create**. The token is returned to `https://jwt.ms` and should be displayed to you.
7. You can now run the user flow again and you should be able to sign in with the account that you created. The returned token includes the claims that you selected of country/region, name, and postal code.

Create a profile editing user flow

If you want to enable users to edit their profile in your application, you use a profile editing user flow.

1. In the menu of the Azure AD B2C tenant overview page, select **User flows (policies)**, and then select **New user flow**.
2. Select the **Profile editing** user flow on the **Recommended** tab.
3. Enter a **Name** for the user flow. For example, *profileediting1*.
4. For **Identity providers**, select **Local Account SignIn**.
5. For **User attributes**, choose the attributes that you want the customer to be able to edit in their profile. For example, select **Show more**, and then choose both attributes and claims for **Display name** and **Job title**. Click **OK**.
6. Click **Create** to add the user flow. A prefix of *B2C_1* is automatically appended to the name.

Test the user flow

1. Select the user flow you created to open its overview page, then select **Run user flow**.
2. For **Application**, select the web application named *webapp1* that you previously registered. The **Reply URL** should show `https://jwt.ms`.

3. Click **Run user flow**, and then sign in with the account that you previously created.
4. You now have the opportunity to change the display name and job title for the user. Click **Continue**. The token is returned to `https://jwt.ms` and should be displayed to you.

Create a password reset user flow

To enable users of your application to reset their password, you use a password reset user flow.

1. In the Azure AD B2C tenant overview menu, select **User flows (policies)**, and then select **New user flow**.
2. Select the **Password reset** user flow on the **Recommended** tab.
3. Enter a **Name** for the user flow. For example, `passwordreset1`.
4. For **Identity providers**, enable **Reset password using email address**.
5. Under Application claims, click **Show more** and choose the claims that you want returned in the authorization tokens sent back to your application. For example, select **User's Object ID**.
6. Click **OK**.
7. Click **Create** to add the user flow. A prefix of `B2C_1` is automatically appended to the name.

Test the user flow

1. Select the user flow you created to open its overview page, then select **Run user flow**.
2. For **Application**, select the web application named `webapp1` that you previously registered. The **Reply URL** should show `https://jwt.ms`.
3. Click **Run user flow**, verify the email address of the account that you previously created, and select **Continue**.
4. You now have the opportunity to change the password for the user. Change the password and select **Continue**. The token is returned to `https://jwt.ms` and should be displayed to you.

Next steps

In this article, you learned how to:

- Create a sign-up and sign-in user flow
- Create a profile editing user flow
- Create a password reset user flow

Next, learn about adding identity providers to your applications to enable user sign-in with providers like Azure AD, Amazon, Facebook, GitHub, LinkedIn, Microsoft, or Twitter.

[Add identity providers to your applications >](#)

Tutorial: Add identity providers to your applications in Azure Active Directory B2C

1/28/2020 • 6 minutes to read • [Edit Online](#)

In your applications, you may want to enable users to sign in with different identity providers. An *identity provider* creates, maintains, and manages identity information while providing authentication services to applications. You can add identity providers that are supported by Azure Active Directory B2C (Azure AD B2C) to your [user flows](#) using the Azure portal.

In this article, you learn how to:

- Create the identity provider applications
- Add the identity providers to your tenant
- Add the identity providers to your user flow

You typically use only one identity provider in your applications, but you have the option to add more. This tutorial shows you how to add an Azure AD identity provider and a Facebook identity provider to your application. Adding both of these identity providers to your application is optional. You can also add other identity providers, such as [Amazon](#), [GitHub](#), [Google](#), [LinkedIn](#), [Microsoft](#), or [Twitter](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

[Create a user flow](#) to enable users to sign up and sign in to your application.

Create applications

Identity provider applications provide the identifier and key to enable communication with your Azure AD B2C tenant. In this section of the tutorial, you create an Azure AD application and a Facebook application from which you get identifiers and keys to add the identity providers to your tenant. If you're adding just one of the identity providers, you only need to create the application for that provider.

Create an Azure Active Directory application

To enable sign-in for users from Azure AD, you need to register an application within the Azure AD tenant. The Azure AD tenant is not the same as your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your Azure AD tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
4. Select **New registration**.
5. Enter a name for your application. For example, `Azure AD B2C App`.
6. Accept the selection of **Accounts in this organizational directory only** for this application.
7. For the **Redirect URI**, accept the value of **Web** and enter the following URL in all lowercase letters, replacing `your-B2C-tenant-name` with the name of your Azure AD B2C tenant.

<https://your-B2C-tenant-name.b2clogin.com/your-B2C-tenant-name.onmicrosoft.com/oauth2/authresp>

For example, <https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp>.

All URLs should now be using [b2clogin.com](#).

8. Select **Register**, then record the **Application (client) ID** which you use in a later step.
9. Under **Manage** in the application menu, select **Certificates & secrets**, then select **New client secret**.
10. Enter a **Description** for the client secret. For example, [Azure AD B2C App Secret](#).
11. Select the expiration period. For this application, accept the selection of **In 1 year**.
12. Select **Add**, then record the value of the new client secret which you use in a later step.

Create a Facebook application

To use a Facebook account as an identity provider in Azure AD B2C, you need to create an application at Facebook. If you don't already have a Facebook account, you can get it at <https://www.facebook.com/>.

1. Sign in to [Facebook for developers](#) with your Facebook account credentials.
2. If you haven't already done so, you need to register as a Facebook developer. To do this, select **Get Started** on the upper-right corner of the page, accept Facebook's policies, and complete the registration steps.
3. Select **My Apps** and then **Create App**.
4. Enter a **Display Name** and a valid **Contact Email**.
5. Click **Create App ID**. This may require you to accept Facebook platform policies and complete an online security check.
6. Select **Settings > Basic**.
7. Choose a **Category**, for example [Business and Pages](#). This value is required by Facebook, but isn't used by Azure AD B2C.
8. At the bottom of the page, select **Add Platform**, and then select **Website**.
9. In **Site URL**, enter <https://your-tenant-name.b2clogin.com/> replacing [your-tenant-name](#) with the name of your tenant.
10. Enter a URL for the **Privacy Policy URL**, for example <http://www.contoso.com/>. The privacy policy URL is a page you maintain to provide privacy information for your application.
11. Select **Save Changes**.
12. At the top of the page, record the value of **App ID**.
13. Next to **App Secret**, select **Show** and record its value. You use both the App ID and App Secret to configure Facebook as an identity provider in your tenant. **App Secret** is an important security credential which you should store securely.
14. Select the plus sign next to **PRODUCTS**, then under **Facebook Login**, select **Set up**.
15. Under **Facebook Login** in the left-hand menu, select **Settings**.
16. In **Valid OAuth redirect URIs**, enter
<https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp>. Replace [your-tenant-name](#) with the name of your tenant. Select **Save Changes** at the bottom of the page.
17. To make your Facebook application available to Azure AD B2C, click the **Status** selector at the top right of the page and turn it **On** to make the Application public, and then click **Confirm**. At this point, the Status should change from **Development** to **Live**.

Add the identity providers

After you create the application for the identity provider that you want to add, you add the identity provider to

your tenant.

Add the Azure Active Directory identity provider

1. Make sure you're using the directory that contains Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your Azure AD B2C tenant.
2. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
3. Select **Identity providers**, and then select **New OpenID Connect provider**.
4. Enter a **Name**. For example, enter *Contoso Azure AD*.
5. For **Metadata url**, enter the following URL replacing `your-AD-tenant-domain` with the domain name of your Azure AD tenant:

```
https://login.microsoftonline.com/your-AD-tenant-domain/.well-known/openid-configuration
```

For example, `https://login.microsoftonline.com/contoso.onmicrosoft.com/.well-known/openid-configuration`.

6. For **Client ID**, enter the application ID that you previously recorded.
7. For **Client secret**, enter the client secret that you previously recorded.
8. Leave the default values for **Scope**, **Response type**, and **Response mode**.
9. (Optional) Enter a value for **Domain_hint**. For example, *ContosoAD*. **Domain hints** are directives that are included in the authentication request from an application. They can be used to accelerate the user to their federated IdP sign-in page. Or they can be used by a multi-tenant application to accelerate the user straight to the branded Azure AD sign-in page for their tenant.
10. Under **Identity provider claims mapping**, enter the following claims mapping values:

- **User ID**: *oid*
- **Display name**: *name*
- **Given name**: *given_name*
- **Surname**: *family_name*
- **Email**: *unique_name*

11. Select **Save**.

Add the Facebook identity provider

1. Select **Identity providers**, then select **Facebook**.
2. Enter a **Name**. For example, *Facebook*.
3. For the **Client ID**, enter the App ID of the Facebook application that you created earlier.
4. For the **Client secret**, enter the App Secret that you recorded.
5. Select **Save**.

Update the user flow

In the tutorial that you completed as part of the prerequisites, you created a user flow for sign-up and sign-in named *B2C_1_signupsignin1*. In this section, you add the identity providers to the *B2C_1_signupsignin1* user flow.

1. Select **User flows (policies)**, and then select the *B2C_1_signupsignin1* user flow.
2. Select **Identity providers**, select the **Facebook** and **Contoso Azure AD** identity providers that you added.
3. Select **Save**.

Test the user flow

1. On the Overview page of the user flow that you created, select **Run user flow**.
2. For **Application**, select the web application named *webapp1* that you previously registered. The **Reply URL** should show `https://jwt.ms`.
3. Select **Run user flow**, and then sign in with an identity provider that you previously added.
4. Repeat steps 1 through 3 for the other identity providers that you added.

If the sign in operation is successful, you're redirected to `https://jwt.ms` which displays the Decoded Token, similar to:

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "<key-ID>"  
}.{  
  "exp": 1562346892,  
  "nbf": 1562343292,  
  "ver": "1.0",  
  "iss": "https://your-b2c-tenant.b2clogin.com/10000000-0000-0000-0000-000000000000/v2.0/",  
  "sub": "20000000-0000-0000-0000-000000000000",  
  "aud": "30000000-0000-0000-0000-000000000000",  
  "nonce": "defaultNonce",  
  "iat": 1562343292,  
  "auth_time": 1562343292,  
  "name": "User Name",  
  "idp": "facebook.com",  
  "postalCode": "12345",  
  "tfp": "B2C_1_signupsignin1"  
}.[Signature]
```

Next steps

In this article, you learned how to:

- Create the identity provider applications
- Add the identity providers to your tenant
- Add the identity providers to your user flow

Next, learn how to customize the UI of the pages shown to users as part of their identity experience in your applications:

[Customize the user interface of your applications in Azure Active Directory B2C](#)

Tutorial: Customize the interface of user experiences in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

For more common user experiences, such as sign-up, sign-in, and profile editing, you can use [user flows](#) in Azure Active Directory B2C (Azure AD B2C). The information in this tutorial helps you to learn how to [customize the user interface \(UI\)](#) of these experiences using your own HTML and CSS files.

In this article, you learn how to:

- Create UI customization files
- Update the user flow to use the files
- Test the customized UI

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

Create a [user flow](#) to enable users to sign up and sign in to your application.

Create customization files

You create an Azure storage account and container and then place basic HTML and CSS files in the container.

Create a storage account

Although you can store your files in many ways, for this tutorial, you store them in [Azure Blob storage](#).

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure subscription. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your subscription. This directory is different than the one that contains your Azure B2C tenant.
3. Choose All services in the top-left corner of the Azure portal, search for and select **Storage accounts**.
4. Select **Add**.
5. Under **Resource group**, select **Create new**, enter a name for the new resource group, and then click **OK**.
6. Enter a name for the storage account. The name you choose must be unique across Azure, must be between 3 and 24 characters in length, and may contain numbers and lowercase letters only.
7. Select the location of the storage account or accept the default location.
8. Accept all other default values, select **Review + create**, and then click **Create**.
9. After the storage account is created, select **Go to resource**.

Create a container

1. On the overview page of the storage account, select **Blobs**.
2. Select **Container**, enter a name for the container, choose **Blob (anonymous read access for blobs only)**, and then click **OK**.

Enable CORS

Azure AD B2C code in a browser uses a modern and standard approach to load custom content from a URL that you specify in a user flow. Cross-origin resource sharing (CORS) allows restricted resources on a web page to be requested from other domains.

1. In the menu, select **CORS**.
2. For **Allowed origins**, enter `https://your-tenant-name.b2clogin.com`. Replace `your-tenant-name` with the name of your Azure AD B2C tenant. For example, `https://fabrikam.b2clogin.com`. You need to use all lowercase letters when entering your tenant name.
3. For **Allowed Methods**, select `GET`, `PUT`, and `OPTIONS`.
4. For **Allowed Headers**, enter an asterisk (*).
5. For **Exposed Headers**, enter an asterisk (*).
6. For **Max age**, enter 200.

ALLOWED ORIGINS	ALLOWED METHODS	ALLOWED HEADERS	EXPOSED HEADERS	MAX AGE
<code>fabrikam.b2clogin.com</code>	<code>GET,OPTIONS</code>	<code>*</code>	<code>*</code>	200

7. Click **Save**.

Create the customization files

To customize the UI of the sign-up experience, you start by creating a simple HTML and CSS file. You can configure your HTML any way you want, but it must have a **div** element with an identifier of `api`. For example, `<div id="api"></div>`. Azure AD B2C injects elements into the `api` container when the page is displayed.

1. In a local folder, create the following file and make sure that you change `your-storage-account` to the name of the storage account and `your-container` to the name of the container that you created. For example, `https://store1.blob.core.windows.net/b2c/style.css`.

```

<!DOCTYPE html>
<html>
  <head>
    <title>My B2C Application</title>
    <link rel="stylesheet" href="https://your-storage-account.blob.core.windows.net/your-
container/style.css">
  </head>
  <body>
    <h1>My B2C Application</h1>
    <div id="api"></div>
  </body>
</html>

```

The page can be designed any way that you want, but the `api` div element is required for any HTML customization file that you create.

2. Save the file as *custom-ui.html*.
3. Create the following simple CSS that centers all elements on the sign-up or sign-in page including the elements that Azure AD B2C injects.

```

h1 {
  color: blue;
  text-align: center;
}
.intro h2 {
  text-align: center;
}
.entry {
  width: 300px ;
  margin-left: auto ;
  margin-right: auto ;
}
.divider h2 {
  text-align: center;
}
.create {
  width: 300px ;
  margin-left: auto ;
  margin-right: auto ;
}

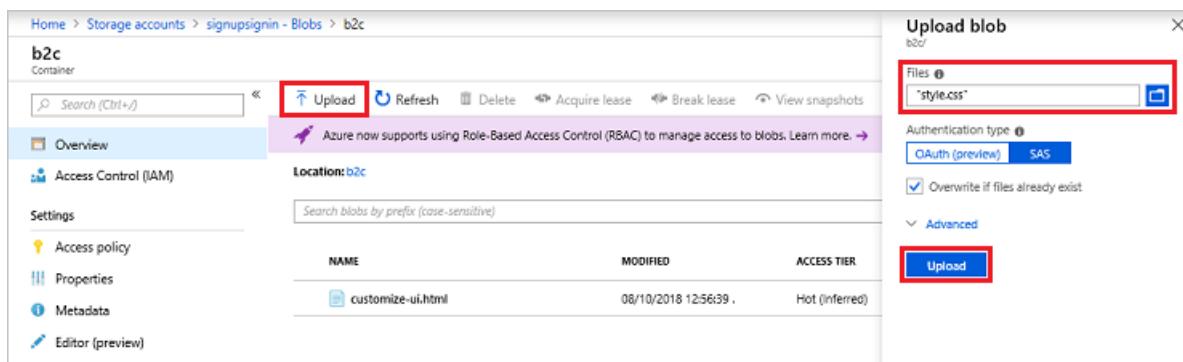
```

4. Save the file as *style.css*.

Upload the customization files

In this tutorial, you store the files that you created in the storage account so that Azure AD B2C can access them.

1. Choose **All services** in the top-left corner of the Azure portal, search for and select **Storage accounts**.
2. Select the storage account you created, select **Blobs**, and then select the container that you created.
3. Select **Upload**, navigate to and select the *custom-ui.html* file, and then click **Upload**.



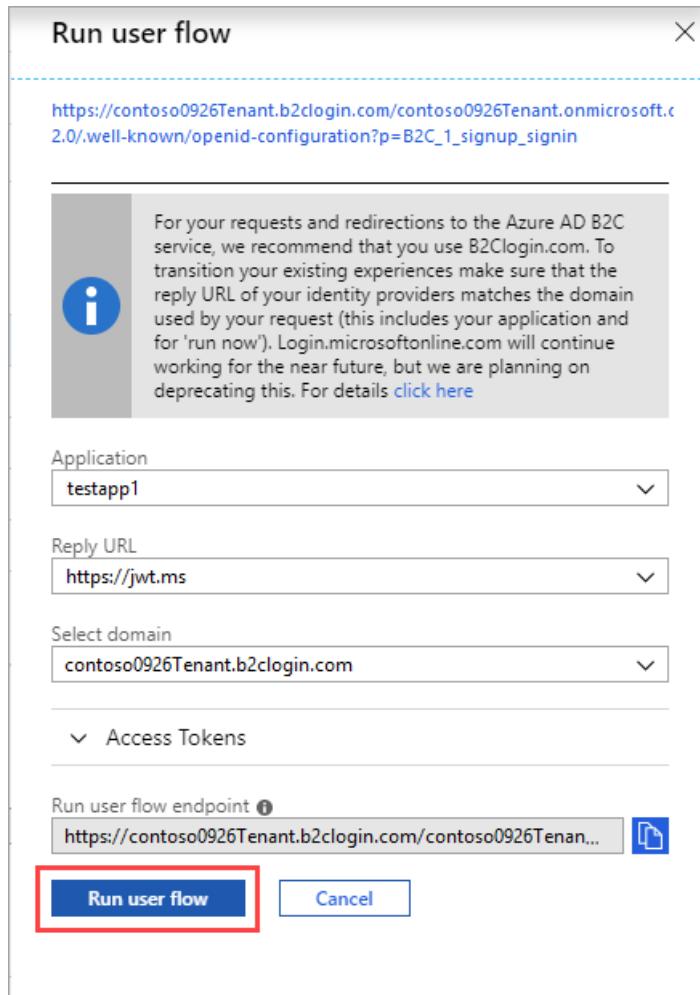
4. Copy the URL for the file that you uploaded to use later in the tutorial.
5. Repeat step 3 and 4 for the *style.css* file.

Update the user flow

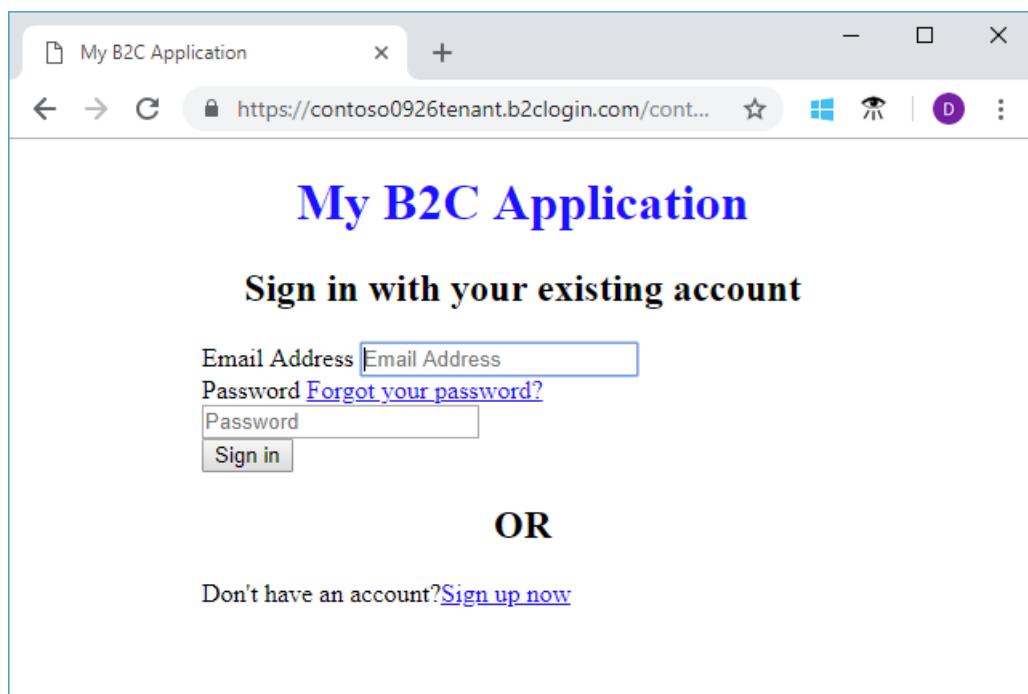
1. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
2. Select **User flows (policies)**, and then select the *B2C_1_signupsignin1* user flow.
3. Select **Page layouts**, and then under **Unified sign-up or sign-in page**, click **Yes** for **Use custom page content**.
4. In **Custom page URI**, enter the URI for the *custom-ui.html* file that you recorded earlier.
5. At the top of the page, select **Save**.

Test the user flow

1. In your Azure AD B2C tenant, select **User flows** and select the *B2C_1_signupsignin1* user flow.
2. At the top of the page, click **Run user flow**.
3. Click the **Run user flow** button.



You should see a page similar to the following example with the elements centered based on the CSS file that you created:



Next steps

In this article, you learned how to:

- Create UI customization files
- Update the user flow to use the files
- Test the customized UI

[Language customization in Azure Active Directory B2C](#)

Tutorial: Enable authentication in a web application using Azure Active Directory B2C

1/28/2020 • 5 minutes to read • [Edit Online](#)

This tutorial shows you how to use Azure Active Directory B2C (Azure AD B2C) to sign in and sign up users in an ASP.NET web application. Azure AD B2C enables your applications to authenticate to social accounts, enterprise accounts, and Azure Active Directory accounts using open standard protocols.

In this tutorial, you learn how to:

- Update the application in Azure AD B2C
- Configure the sample to use the application
- Sign up using the user flow

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Create user flows](#) to enable user experiences in your application.
- Install [Visual Studio 2019](#) with the **ASP.NET and web development** workload.

Update the application registration

In the tutorial that you completed as part of the prerequisites, you registered a web application in Azure AD B2C. To enable communication with the sample in this tutorial, you need to add a redirect URI and create a client secret (key) for the registered application.

Add a redirect URI (reply URL)

You can use the current **Applications** experience or our new unified **App registrations (Preview)** experience to update the application. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **Applications**, and then select the *webapp1* application.
5. Under **Reply URL**, add `https://localhost:44316`.
6. Select **Save**.
7. On the properties page, record the application ID for use in a later step when you configure the web application.

Create a client secret

Next, create a client secret for the registered web application. The web application code sample uses this to prove its identity when requesting tokens.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Under **API ACCESS**, select **Keys**.
2. Enter a description for the key in the **Key description** box. For example, *clientsecret1*.
3. Select a validity **Duration** and then select **Save**.
4. Record the key's **VALUE**. You use this value for configuration in a later step.

Configure the sample

In this tutorial, you configure a sample that you can download from GitHub. The sample uses ASP.NET to provide a simple to-do list. The sample uses [Microsoft OWIN middleware components](#). Download a zip file or clone the sample from GitHub. Make sure that you extract the sample file in a folder where the total character length of the path is less than 260.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnet-webapp-and-webapi.git
```

The following two projects are in the sample solution:

- **TaskWebApp** - Create and edit a task list. The sample uses the **sign-up or sign-in** user flow to sign up and sign in users.
- **TaskService** - Supports the create, read, update, and delete task list functionality. The API is protected by Azure AD B2C and called by TaskWebApp.

You change the sample to use the application that's registered in your tenant, which includes the application ID and the key that you previously recorded. You also configure the user flows that you created. The sample defines the configuration values as settings in the *Web.config* file.

Update the settings in the *Web.config* file to work with your user flow:

1. Open the **B2C-WebAPI-DotNet** solution in Visual Studio.
2. In the **TaskWebApp** project, open the **Web.config** file.
 - a. Update the value of `ida:Tenant` and `ida:AadInstance` with the name of the Azure AD B2C tenant that you created. For example, replace `fabrikamb2c` with `contoso`.
 - b. Replace the value of `ida:ClientId` with the application ID that you recorded.
 - c. Replace the value of `ida:ClientSecret` with the key that you recorded. If the client secret contains any predefined XML entities, for example less than (`<`), greater than (`>`), ampersand (`&`), or double quote (`"`), you must escape those characters by XML-encoding the client secret before adding it to your *Web.config*.
 - d. Replace the value of `ida:SignUpSignInPolicyId` with `b2c_1_signupsignin1`.
 - e. Replace the value of `ida>EditProfilePolicyId` with `b2c_1_profileediting1`.
 - f. Replace the value of `ida:ResetPasswordPolicyId` with `b2c_1_passwordreset1`.

Run the sample

1. In Solution Explorer, right-click the **TaskWebApp** project, and then click **Set as StartUp Project**.
2. Press **F5**. The default browser launches to the local web site address <https://localhost:44316/>.

Sign up using an email address

1. Select **Sign up / Sign in** to sign up as a user of the application. The **b2c_1_signupsignin1** user flow is used.
2. Azure AD B2C presents a sign-in page with a sign-up link. Since you don't have an account yet, select **Sign up now**. The sign-up workflow presents a page to collect and verify the user's identity using an email address. The sign-up workflow also collects the user's password and the requested attributes defined in the

user flow.

3. Use a valid email address and validate using the verification code. Set a password. Enter values for the requested attributes.

Email Address
joe@contoso.com

Send verification code

New Password
••••••••

Confirm New Password
••••••••

Postal Code
98029

Display Name
Joe Contoso

Create Cancel

4. Select **Create** to create a local account in the Azure AD B2C tenant.

The application user can now use their email address to sign in and use the web application.

However, the **To-Do List** feature won't function until you complete the next tutorial in the series, [Tutorial: Use Azure AD B2C to protect an ASP.NET web API](#).

Next steps

In this tutorial, you learned how to:

- Update the application in Azure AD B2C
- Configure the sample to use the application
- Sign up using the user flow

Now move on to the next tutorial to enable the **To-Do List** feature of the web application. In it, you register a web API application in your own Azure AD B2C tenant, and then modify the code sample to use your tenant for API authentication.

[Tutorial: Use Azure Active Directory B2C to protect an ASP.NET web API >](#)

Tutorial: Authenticate users in a native desktop client using Azure Active Directory B2C

1/28/2020 • 5 minutes to read • [Edit Online](#)

This tutorial shows you how to use Azure Active Directory B2C (Azure AD B2C) to sign in and sign up users in an Windows Presentation Foundation (WPF) desktop application. Azure AD B2C enables your applications to authenticate to social accounts, enterprise accounts, and Azure Active Directory accounts using open standard protocols.

In this tutorial, you learn how to:

- Add the native client application
- Configure the sample to use the application
- Sign up using the user flow

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Create user flows](#) to enable user experiences in your application.
- Install [Visual Studio 2019](#) with **.NET desktop development** and **ASP.NET and web development** workloads.

Add the native client application

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select **Add**.
 5. Enter a name for the application. For example, *nativeapp1*.
 6. For **Native client**, select **Yes**.
 7. Enter a **Custom Redirect URI** with a unique scheme. For example,

`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`. There are two important considerations when choosing a redirect URI:

- **Unique:** The scheme of the redirect URI must be unique for every application. In the example `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`, `com.onmicrosoft.contosob2c.exampleapp` is the scheme. This pattern should be followed. If two applications share the same scheme, the user is given a choice to choose an application. If the user chooses incorrectly, the sign-in fails.
- **Complete:** The redirect URI must have both a scheme and a path. The path must contain at least one forward slash after the domain. For example, `//oauth/` works while `//oauth` fails. Don't include special characters in the URI, for example, underscores.

8. Select **Create**.

Record the **Application (client) ID** for use in a later step.

Configure the sample

In this tutorial, you configure a sample that you can download from GitHub. The sample WPF desktop application demonstrates sign-up, sign-in, and can call a protected web API in Azure AD B2C. [Download a zip file, browse the repo](#), or clone the sample from GitHub.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnet-desktop.git
```

To update the application to work with your Azure AD B2C tenant and invoke its user flows instead of those in the default demo tenant:

1. Open the **active-directory-b2c-wpf** solution (`active-directory-b2c-wpf.sln`) in Visual Studio.
2. In the **active-directory-b2c-wpf** project, open the `App.xaml.cs` file and find the following variable definitions. Replace `{your-tenant-name}` with your Azure AD B2C tenant name and `{application-ID}` with the application ID that you recorded earlier.

```
private static readonly string Tenant = "{your-tenant-name}.onmicrosoft.com";
private static readonly string AzureAdB2CHostname = "{your-tenant-name}.b2clogin.com";
private static readonly string ClientId = "{application-ID}";
```

3. Update the policy name variables with the names of the user flows that you created as part of the prerequisites. For example:

```
public static string PolicySignUpSignIn = "B2C_1_signupsignin1";
public static string PolicyEditProfile = "B2C_1_profileediting1";
public static string PolicyResetPassword = "B2C_1_passwordreset1";
```

Run the sample

Press **F5** to build and run the sample.

Sign up using an email address

1. Select **Sign In** to sign up as a user. This uses the **B2C_1_signupsignin1** user flow.
2. Azure AD B2C presents a sign in page with a **Sign up now** link. Since you don't yet have an account, select the **Sign up now** link.
3. The sign-up workflow presents a page to collect and verify the user's identity using an email address. The sign-up workflow also collects the user's password and the requested attributes defined in the user flow.

Use a valid email address and validate using the verification code. Set a password. Enter values for the requested attributes.

Email Address
lisa@contoso.com

New Password
••••••••

Confirm New Password
••••••••

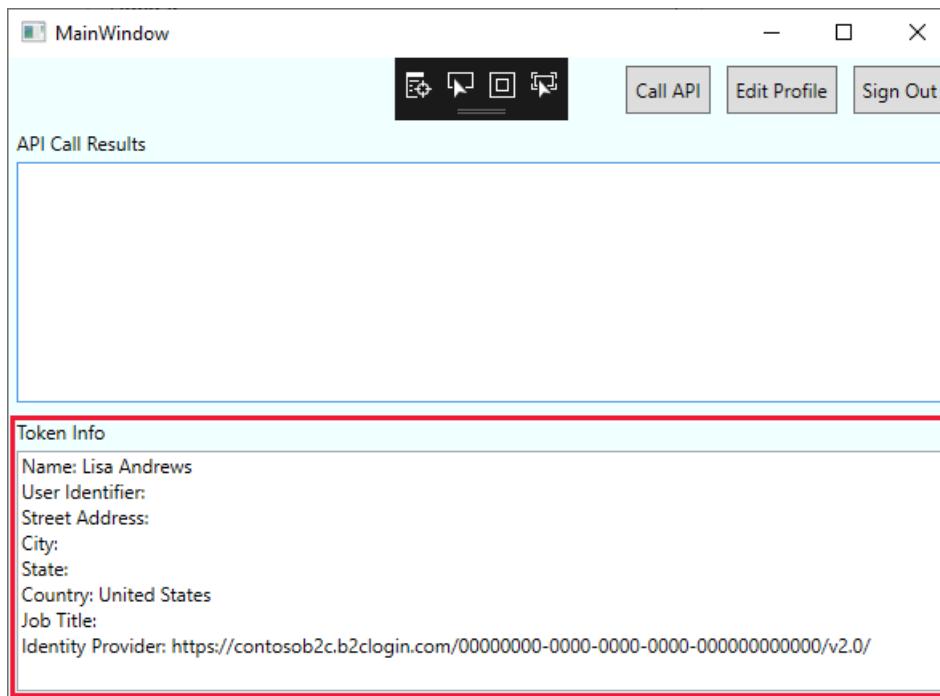
Postal Code
98029

Display Name
Lisa Andrews

Create Cancel

4. Select **Create** to create a local account in the Azure AD B2C tenant.

The user can now use their email address to sign in and use the desktop application. After a successful sign-up or sign-in, the token details are displayed in the lower pane of the WPF app.



If you select the **Call API** button, an **error message** is displayed. You encounter the error because, in its current state, the application is attempting to access an API protected by the demo tenant, fabrikamb2c.onmicrosoft.com. Since your access token is valid only for your Azure AD B2C tenant, the API call is therefore unauthorized.

Continue to the next tutorial to register a protected web API in your own tenant and enable the **Call API** functionality.

Next steps

In this tutorial, you learned how to:

- Add the native client application
- Configure the sample to use the application
- Sign up using the user flow

Next, to enable the **Call API** button functionality, grant the WPF desktop application access to a web API registered in your own Azure AD B2C tenant:

[Tutorial: Grant access to a Node.js web API from a desktop app >](#)

Tutorial: Enable authentication in a single-page application using Azure Active Directory B2C (Azure AD B2C)

1/28/2020 • 5 minutes to read • [Edit Online](#)

This tutorial shows you how to use Azure Active Directory B2C (Azure AD B2C) to sign in and sign up users in a single-page application (SPA). Azure AD B2C enables your applications to authenticate to social accounts, enterprise accounts, and Azure Active Directory accounts using open standard protocols.

In this tutorial, you learn how to:

- Update the application in Azure AD B2C
- Configure the sample to use the application
- Sign up using the user flow

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

You need the following Azure AD B2C resources in place before continuing with the steps in this tutorial:

- [Azure AD B2C tenant](#)
- [Application registered](#) in your tenant
- [User flows created](#) in your tenant

Additionally, you need the following in your local development environment:

- Code editor, for example [Visual Studio Code](#) or [Visual Studio 2019](#)
- [.NET Core SDK 2.2](#) or later
- [Node.js](#)

Update the application

In the second tutorial that you completed as part of the prerequisites, you registered a web application in Azure AD B2C. To enable communication with the sample in the tutorial, you need to add a redirect URI to the application in Azure AD B2C.

You can use the current **Applications** experience or our new unified **App registrations (Preview)** experience to update the application. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
 3. Select **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select the *webapp1* application.
 5. Under **Reply URL**, add `http://localhost:6420`.

6. Select **Save**.
7. On the properties page, record the **Application ID**. You use the app ID in a later step when you update the code in the single-page web application.

Get the sample code

In this tutorial, you configure a code sample that you download from GitHub. The sample demonstrates how a single-page application can use Azure AD B2C for user sign-up and sign-in, and to call a protected web API.

[Download a zip file](#) or clone the sample from GitHub.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-javascript-msal-singlepageapp.git
```

Update the sample

Now that you've obtained the sample, update the code with your Azure AD B2C tenant name and the application ID you recorded in an earlier step.

1. Open the `index.html` file in the root of the sample directory.
2. In the `msalConfig` definition, modify the **clientId** value with the Application ID you recorded in an earlier step. Next, update the **authority** URI value with your Azure AD B2C tenant name. Also update the URI with the name of the sign-up/sign-in user flow you created in one of the prerequisites (for example, `B2C_1_signupsignin1`).

```
var msalConfig = {
  auth: {
    clientId: "00000000-0000-0000-0000-000000000000", //This is your client ID
    authority: "https://fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_susi", //This is
    your tenant info
    validateAuthority: false
  },
  cache: {
    cacheLocation: "localStorage",
    storeAuthStateInCookie: true
  }
};
```

The name of the user flow used in this tutorial is **B2C_1_signupsignin1**. If you're using a different user flow name, specify that name in the `authority` value.

Run the sample

1. Open a console window and change to the directory containing the sample. For example:

```
cd active-directory-b2c-javascript-msal-singlepageapp
```

2. Run the following commands:

```
npm install && npm update
node server.js
```

The console window displays the port number of the locally running Node.js server:

```
Listening on port 6420...
```

3. Go to <http://localhost:6420> in your browser to view the application.

The sample supports sign-up, sign-in, profile editing, and password reset. This tutorial highlights how a user signs up using an email address.

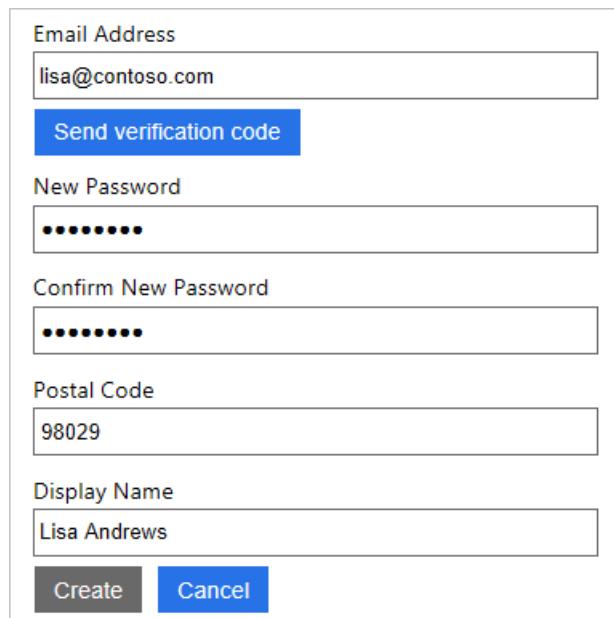
Sign up using an email address

WARNING

After sign-up or sign-in, you might see an [insufficient permissions error](#). Due to the code sample's current implementation, this error is expected. This issue will be resolved in a future version of the code sample, at which time this warning will be removed.

1. Select **Login** to initiate the *B2C_1_signupsignin1* user flow you specified in an earlier step.
2. Azure AD B2C presents a sign-in page with a sign-up link. Since you don't yet have an account, select the **Sign up now** link.
3. The sign-up workflow presents a page to collect and verify the user's identity using an email address. The sign-up workflow also collects the user's password and the requested attributes defined in the user flow.

Use a valid email address and validate using the verification code. Set a password. Enter values for the requested attributes.



Email Address
lisa@contoso.com

Send verification code

New Password
•••••••

Confirm New Password
•••••••

Postal Code
98029

Display Name
Lisa Andrews

Create Cancel

4. Select **Create** to create a local account in the Azure AD B2C directory.

When you select **Create**, the sign up page closes and the sign in page reappears.

You can now use your email address and password to sign in to the application.

Error: insufficient permissions

After you sign in, the application may return an insufficient permissions error:

```
ServerError: AADB2C90205: This application does not have sufficient permissions against this web resource to perform the operation.  
Correlation ID: ce15bbcc-0000-0000-0000-494a52e95cd7  
Timestamp: 2019-07-20 22:17:27Z
```

You receive this error because the web application is attempting to access a web API protected by the demo directory, *fabrikamb2c*. Because your access token is valid only for your Azure AD directory, the API call is unauthorized.

To fix this error, continue on to the next tutorial in the series (see [Next steps](#)) to create a protected web API for your directory.

Next steps

In this article, you learned how to:

- Update the application in Azure AD B2C
- Configure the sample to use the application
- Sign up using the user flow

Now move on to the next tutorial in the series to grant access to a protected web API from the SPA:

[Tutorial: Grant access to an ASP.NET Core web API from an SPA using Azure AD B2C >](#)

Tutorial: Grant access to an ASP.NET web API using Azure Active Directory B2C

1/28/2020 • 7 minutes to read • [Edit Online](#)

This tutorial shows you how to call a protected web API resource in Azure Active Directory B2C (Azure AD B2C) from an ASP.NET web application.

In this tutorial, you learn how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Configure the sample to use the application

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

Complete the steps and prerequisites in [Tutorial: Enable authenticate in a web application using Azure Active Directory B2C](#).

Add a web API application

Web API resources need to be registered in your tenant before they can accept and respond to protected resource requests by client applications that present an access token.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
 3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select **Add**.
 5. Enter a name for the application. For example, *webapi1*.
 6. For **Include web app/ web API**, select **Yes**.
 7. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. In this tutorial, the sample runs locally and listens at `https://localhost:44332`.
 8. For **App ID URI**, enter the identifier used for your web API. The full identifier URI including the domain is generated for you. For example, `https://contosotenant.onmicrosoft.com/api`.
 9. Click **Create**.
 10. On the properties page, record the application ID that you'll use when you configure the web application.

Configure scopes

Scopes provide a way to govern access to protected resources. Scopes are used by the web API to implement

scope-based access control. For example, users of the web API could have both read and write access, or users of the web API might have only read access. In this tutorial, you use scopes to define read and write permissions for the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**.
2. Select the *webapi1* application to open its **Properties** page.
3. Select **Published scopes**. Published scopes can be used to grant a client application certain permissions to the web API.
4. For **SCOPE**, enter `demo.read`, and for **DESCRIPTION**, enter `Read access to the web API`.
5. For **SCOPE**, enter `demo.write`, and for **DESCRIPTION**, enter `Write access to the web API`.
6. Select **Save**.

Grant permissions

To call a protected web API from an application, you need to grant your application permissions to the API. In the prerequisite tutorial, you created a web application in Azure AD B2C named *webapp1*. You use this application to call the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**, and then select the web application that should have access to the API. For example, *webapp1*.
2. Select **API access**, and then select **Add**.
3. In the **Select API** dropdown, select the API to which web application should be granted access. For example, *webapi1*.
4. In the **Select Scopes** dropdown, select the scopes that you defined earlier. For example, `demo.read` and `demo.write`.
5. Select **OK**.

Your application is registered to call the protected web API. A user authenticates with Azure AD B2C to use the application. The application obtains an authorization grant from Azure AD B2C to access the protected web API.

Configure the sample

Now that the web API is registered and you have scopes defined, you configure the web API to use your Azure AD B2C tenant. In this tutorial, you configure a sample web API. The sample web API is included in the project you downloaded in the prerequisite tutorial.

There are two projects in the sample solution:

- **TaskWebApp** - Create and edit a task list. The sample uses the **sign-up or sign-in** user flow to sign up or sign in users.
- **TaskService** - Supports the create, read, update, and delete task list functionality. The API is protected by Azure AD B2C and called by TaskWebApp.

Configure the web application

1. Open the **B2C-WebAPI-DotNet** solution in Visual Studio.
2. In the **TaskWebApp** project, open **Web.config**.
3. To run the API locally, use the localhost setting for **api:TaskServiceUrl**. Change the Web.config as follows:

```
<add key="api:TaskServiceUrl" value="https://localhost:44332/" />
```

4. Configure the URI of the API. This is the URI the web application uses to make the API request. Also, configure the requested permissions.

```
<add key="api:ApiIdentifier" value="https://<Your tenant name>.onmicrosoft.com/api/" />
<add key="api:ReadScope" value="demo.read" />
<add key="api:WriteScope" value="demo.write" />
```

Configure the web API

1. In the **TaskService** project, open **Web.config**.
2. Configure the API to use your tenant.

```
<add key="ida:AadInstance" value="https://<Your tenant name>.b2clogin.com/{0}/{1}/v2.0/.well-known/openid-configuration" />
<add key="ida:Tenant" value="<Your tenant name>.onmicrosoft.com" />
```

3. Set the client ID to the Application ID of your registered web API application, *webapi1*.

```
<add key="ida:ClientId" value="<application-ID>"/>
```

4. Update the user flow setting with the name of your sign-up and sign-in user flow, *B2C_1_signupsignin1*.

```
<add key="ida:SignUpSignInPolicyId" value="B2C_1_signupsignin1" />
```

5. Configure the scopes setting to match those you created in the portal.

```
<add key="api:ReadScope" value="demo.read" />
<add key="api:WriteScope" value="demo.write" />
```

Run the sample

You need to run both the **TaskWebApp** and **TaskService** projects.

1. In Solution Explorer, right-click on the solution and select **Set StartUp Projects....**
2. Select **Multiple startup projects**.
3. Change the **Action** for both projects to **Start**.
4. Click **OK** to save the configuration.
5. Press **F5** to run both applications. Each application opens in its own browser window.
 - `https://localhost:44316/` is the web application.
 - `https://localhost:4432/` is the web API.
6. In the web application, select **sign-up / sign-in** to sign in to the web application. Use the account that you previously created.
7. After you sign in, select **To-do list** and create a to-do list item.

When you create a to-do list item, the web application makes a request to the web API to generate the to-do list

item. Your protected web application is calling the web API protected by Azure AD B2C.

Next steps

In this tutorial, you learned how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Configure the sample to use the application

[Tutorial: Add identity providers to your applications in Azure Active Directory B2C](#)

Tutorial: Grant access to a Node.js web API from a desktop app using Azure Active Directory B2C

1/28/2020 • 7 minutes to read • [Edit Online](#)

This tutorial shows you how to call a Node.js web API protected by Azure Active Directory B2C (Azure AD B2C) from a Windows Presentation Foundation (WPF) desktop app, also protected by Azure AD B2C.

In this tutorial, you learn how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Update the sample to use the application

Prerequisites

Complete the steps and prerequisites in [Tutorial: Authenticate users in a native desktop client](#).

Add a web API application

Web API resources need to be registered in your tenant before they can accept and respond to protected resource requests by client applications that present an access token.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select **Add**.
 5. Enter a name for the application. For example, *webapi1*.
 6. For **Web App / Web API**, select **Yes**.
 7. For **Allow implicit flow**, select **Yes**.
 8. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. In this tutorial, the sample runs locally and listens at `https://localhost:5000`.
 9. For **App ID URI**, add an API endpoint identifier to the URI shown. For this tutorial, enter `api`, so that the full URI is similar to `https://contosob2c.onmicrosoft.com/api`.
 10. Select **Create**.
 11. Record the **APPLICATION ID** for use in a later step.

Configure scopes

Scopes provide a way to govern access to protected resources. Scopes are used by the web API to implement scope-based access control. For example, some users could have both read and write access, whereas other users might have read-only permissions. In this tutorial, you define read and write permissions for the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**.
2. Select the *webapi1* application to open its **Properties** page.
3. Select **Published scopes**. Published scopes can be used to grant a client application certain permissions to the web API.
4. For **SCOPE**, enter `demo.read`, and for **DESCRIPTION**, enter `Read access to the web API`.
5. For **SCOPE**, enter `demo.write`, and for **DESCRIPTION**, enter `Write access to the web API`.
6. Select **Save**.

Record the value under **SCOPES** for the `demo.read` scope to use in a later step when you configure the desktop application. The full scope value is similar to `https://contosob2c.onmicrosoft.com/api/demo.read`.

Grant permissions

To call a protected web API from a native client application, you need to grant the registered native client application permissions to the web API you registered in Azure AD B2C.

In the prerequisite tutorial, you registered a native client application named *nativeapp1*. The following steps configure that native application registration with the API scopes you exposed for *webapi1* in the previous section. This allows the desktop application to obtain an access token from Azure AD B2C that the web API can use to verify and provide scoped access to its resources. You configure and run both the desktop application and web API code samples later in the tutorial.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**, and then select *nativeapp1*.
2. Select **API access**, and then select **Add**.
3. In the **Select API** dropdown, select *webapi1*.
4. In the **Select Scopes** dropdown, select the scopes that you defined earlier. For example, `demo.read` and `demo.write`.
5. Select **OK**.

A user authenticates with Azure AD B2C to use the WPF desktop application. The desktop application obtains an authorization grant from Azure AD B2C to access the protected web API.

Configure the samples

Now that the web API is registered and you have scopes and permissions configured, you configure the desktop application and web API samples to use your Azure AD B2C tenant.

Update the desktop application

In a prerequisite for this article, you modified a [WPF desktop application](#) to enable sign-in with a user flow in your Azure AD B2C tenant. In this section, you update that same application to reference the web API you registered earlier, *webapi1*.

1. Open the **active-directory-b2c-wpf** solution (`active-directory-b2c-wpf.sln`) in Visual Studio.
2. In the **active-directory-b2c-wpf** project, open the *App.xaml.cs* file and find the following variable definitions.
 - a. Replace the value of the `ApiScopes` variable with the value you recorded earlier when you defined the

demo.read scope.

- b. Replace the value of the `ApiEndpoint` variable with the **Redirect URI** you recorded earlier when you registered the web API (for example, `webapi1`) in your tenant.

Here's an example:

```
public static string[] ApiScopes = { "https://contosob2c.onmicrosoft.com/api/demo.read" };  
public static string ApiEndpoint = "http://localhost:5000";
```

Get and update the Node.js API sample

Next, get the Node.js web API code sample from GitHub and configure it to use the web API you registered in your Azure AD B2C tenant.

[Download a zip file](#) or clone the sample web app from GitHub.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-javascript-nodejs-webapi.git
```

The Node.js web API sample uses the Passport.js library to enable Azure AD B2C to protect calls to the API.

1. Open the `index.js` file.
2. Update these variable definitions with the following values. Change `<web-API-application-ID>` to the **Application (client) ID** of the web API you registered earlier (`webapi1`). Change `<your-b2c-tenant>` to the name of your Azure AD B2C tenant.

```
var clientID = "<web-API-application-ID>";  
var b2cDomainHost = "<your-b2c-tenant>.b2clogin.com";  
var tenantIdGuid = "<your-b2c-tenant>.onmicrosoft.com";  
var policyName = "B2C_1_signupsignin1";
```

3. Since you're running the API locally, update the path in the route for the GET method to `/` instead of the demo app's location of `/hello`:

```
app.get("/",
```

Run the samples

Run the Node.js web API

1. Launch a Node.js command prompt.
2. Change to the directory containing the Node.js sample. For example
3. Run the following commands:

```
npm install && npm update
```

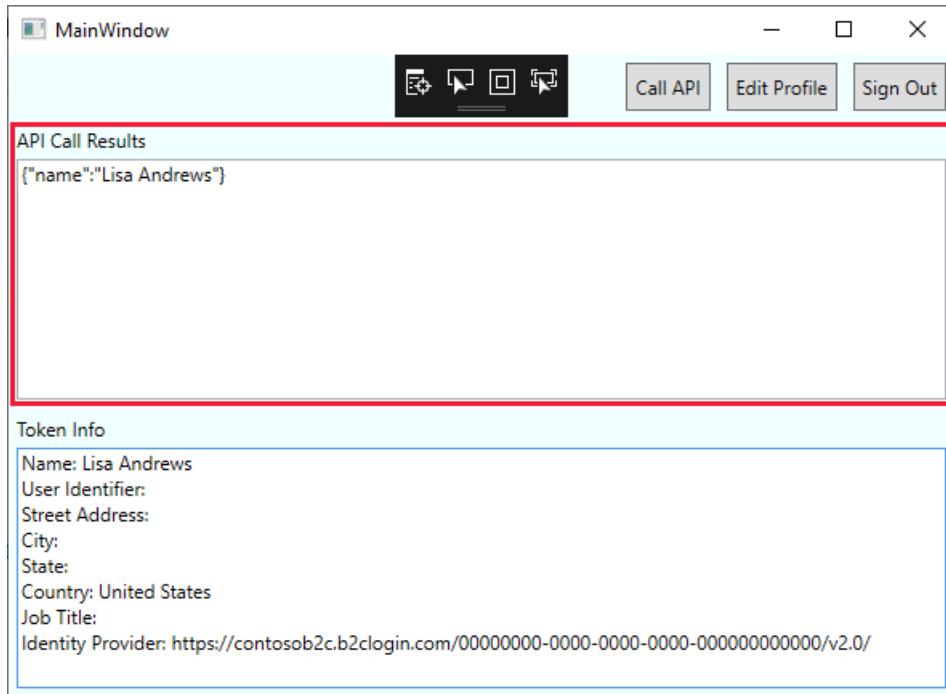
```
node index.js
```

Run the desktop application

1. Open the **active-directory-b2c-wpf** solution in Visual Studio.
2. Press **F5** to run the desktop app.

3. Sign in using the email address and password used in [Authenticate users with Azure Active Directory B2C in a desktop app tutorial](#).
4. Select the **Call API** button.

The desktop application makes a request to the locally running web API, and upon verification of a valid access token, shows the signed-in user's display name.



Your desktop application, protected by Azure AD B2C, is calling the locally running web API that is also protected by Azure AD B2C.

Next steps

In this tutorial, you learned how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Update the sample to use the application

[Tutorial: Add identity providers to your applications in Azure Active Directory B2C](#)

Tutorial: Grant access to an ASP.NET Core web API from a single-page application using Azure Active Directory B2C

1/28/2020 • 9 minutes to read • [Edit Online](#)

This tutorial shows you how to call an Azure Active Directory B2C (Azure AD B2C)-protected ASP.NET Core web API resource from a single-page application.

In this tutorial, you learn how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Configure the sample to use the application

Prerequisites

- Complete the steps and prerequisites in [Tutorial: Enable authentication in a single-page application using Azure Active Directory B2C](#).
- Visual Studio 2019 or later, or Visual Studio Code
- .NET Core 2.2 or later
- Node.js

Add a web API application

Web API resources need to be registered in your tenant before they can accept and respond to protected resource requests by client applications that present an access token.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select **Add**.
 5. Enter a name for the application. For example, *webapi1*.
 6. For **Web App / Web API**, select **Yes**.
 7. For **Allow implicit flow**, select **Yes**.
 8. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. In this tutorial, the sample runs locally and listens at `https://localhost:5000`.
 9. For **App ID URI**, add an API endpoint identifier to the URI shown. For this tutorial, enter `api`, so that the full URI is similar to `https://contosob2c.onmicrosoft.com/api`.
 10. Select **Create**.

11. Record the **APPLICATION ID** for use in a later step.

Configure scopes

Scopes provide a way to govern access to protected resources. Scopes are used by the web API to implement scope-based access control. For example, some users could have both read and write access, whereas other users might have read-only permissions. In this tutorial, you define both read and write permissions for the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**.
2. Select the *webapi1* application to open its **Properties** page.
3. Select **Published scopes**. Published scopes can be used to grant a client application certain permissions to the web API.
4. For **SCOPE**, enter `demo.read`, and for **DESCRIPTION**, enter `Read access to the web API`.
5. For **SCOPE**, enter `demo.write`, and for **DESCRIPTION**, enter `Write access to the web API`.
6. Select **Save**.

Record the value under **SCOPES** for the `demo.read` scope to use in a later step when you configure the single-page application. The full scope value is similar to `https://contosob2c.onmicrosoft.com/api/demo.read`.

Grant permissions

To call a protected web API from another application, you need to grant that application permissions to the web API.

In the prerequisite tutorial, you created a web application named *webapp1*. In this tutorial, you configure that application to call the web API you created in a previous section, *webapi1*.

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Select **Applications**, and then select the web application that should have access to the API. For example, *webapp1*.
 2. Select **API access**, and then select **Add**.
 3. In the **Select API** dropdown, select the API to which web application should be granted access. For example, *webapi1*.
 4. In the **Select Scopes** dropdown, select the scopes that you defined earlier. For example, `demo.read` and `demo.write`.
 5. Select **OK**.

Your single-page web application is registered to call the protected web API. A user authenticates with Azure AD B2C to use the single-page application. The single-page app obtains an authorization grant from Azure AD B2C to access the protected web API.

Configure the sample

Now that the web API is registered and you have scopes defined, you configure the web API code to use your Azure AD B2C tenant. In this tutorial, you configure a sample .NET Core web application you download from GitHub.

[Download a *.zip archive](#) or clone the sample web API project from GitHub.

```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnetcore-webapi.git
```

Configure the web API

1. Open the *B2C-WebApi/appsettings.json* file in Visual Studio or Visual Studio Code.
2. Modify the `AzureAdB2C` block to reflect your tenant name, the application ID of the web API application, the name of your sign-up/sign-in policy, and the scopes you defined earlier. The block should look similar to the following example (with appropriate `Tenant` and `ClientId` values):

```
"AzureAdB2C": {  
    "Tenant": "<your-tenant-name>.onmicrosoft.com",  
    "ClientId": "<webapi-application-ID>",  
    "Policy": "B2C_1_signupsignin1",  
  
    "ScopeRead": "demo.read",  
    "ScopeWrite": "demo.write"  
},
```

Enable CORS

To allow your single-page application to call the ASP.NET Core web API, you need to enable [CORS](#) in the web API.

1. In *Startup.cs*, add CORS to the `ConfigureServices()` method.

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddCors();
```

2. Also within the `ConfigureServices()` method, set the `jwtOptions.Authority` value to the following token issuer URI.

Replace `<your-tenant-name>` with the name of your B2C tenant.

```
jwtOptions.Authority = $"https://<your-tenant-  
name>.b2clogin.com/{Configuration["AzureAdB2C:Tenant"]}/{Configuration["AzureAdB2C:Policy"]}/v2.0";
```

3. In the `Configure()` method, configure CORS.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)  
{  
    app.UseCors(builder =>  
        builder.WithOrigins("http://localhost:6420").AllowAnyHeader().AllowAnyMethod());
```

4. (Visual Studio only) Under **Properties** in the Solution Explorer, open the *launchSettings.json* file, then find the `iisExpress` block.
5. (Visual Studio only) Update the `applicationURL` value with the port number you specified when you registered the *webapi1* application in an earlier step. For example:

```
"iisExpress": {  
    "applicationUrl": "http://localhost:5000/",  
    "sslPort": 0  
}
```

Configure the single-page application

The single-page application (SPA) from the [previous tutorial](#) in the series uses Azure AD B2C for user sign-up and sign-in, and calls the ASP.NET Core web API protected by the *frabrikamb2c* demo tenant.

In this section, you update the single-page application to call the ASP.NET Core web API protected by *your* Azure AD B2C tenant and which you run on your local machine.

To change the settings in the SPA:

1. Open the *index.html* file in the [active-directory-b2c-javascript-msal-singlepageapp](#) project you downloaded or cloned in the previous tutorial.
2. Configure the sample with the URI for the *demo.read* scope you created earlier and the URL of the web API.
 - a. In the `appConfig` definition, replace the `b2cScopes` value with the full URI for the scope (the **SCOPE** value you recorded earlier).
 - b. Change the `webApi` value to the redirect URI you added when you registered the web API application in an earlier step.

The `appConfig` definition should look similar to the following code block (with your tenant name in the place of `<your-tenant-name>`):

```
// The current application coordinates were pre-registered in a B2C tenant.
var appConfig = {
  b2cScopes: ["https://<your-tenant-name>.onmicrosoft.com/api/demo.read"],
  webApi: "http://localhost:5000/"
};
```

Run the SPA and web API

Finally, you run both the ASP.NET Core web API and the Node.js single-page application on your local machine. Then, you sign in to the single-page application and press a button to initiate a request to the protected API.

Although both applications run locally in this tutorial, they use Azure AD B2C for secure sign-up/sign-in and to grant access to the protected web API.

Run the ASP.NET Core web API

In Visual Studio, press **F5** to build and debug the *B2C-WebAPI.sln* solution. When the project launches, a web page is displayed in your default browser announcing the web API is available for requests.

If you prefer to use the `dotnet` CLI instead of Visual Studio:

1. Open a console window and change to the directory containing the `*.csproj` file. For example:

```
cd active-directory-b2c-dotnetcore-webapi/B2C-WebApi
```

2. Build and run the web API by executing `dotnet run`.

When the API is up and running, you should see output similar to the following (for the tutorial, you can safely ignore any `NETSDK1059` warnings):

```
$ dotnet run
Hosting environment: Production
Content root path: /home/user/active-directory-b2c-dotnetcore-webapi/B2C-WebApi
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Run the single page app

1. Open a console window and change to the directory containing the Node.js sample. For example:

```
cd active-directory-b2c-javascript-msal-singlepageapp
```

2. Run the following commands:

```
npm install && npm update  
node server.js
```

The console window displays the port number of where the application is hosted.

```
Listening on port 6420...
```

3. Navigate to `http://localhost:6420` in your browser to view the application.
4. Sign in using the email address and password you used in the [previous tutorial](#). Upon successful login, you should see the `User 'Your Username' logged-in` message.
5. Select the **Call Web API** button. The SPA obtains an authorization grant from Azure AD B2C, then accesses the protected web API to display the contents of its index page:

```
Web API returned:  
<html><r><n><head><r><n> <title>Azure AD B2C API Sample</title><r><n> ...
```

Next steps

In this tutorial, you learned how to:

- Add a web API application
- Configure scopes for a web API
- Grant permissions to the web API
- Configure the sample to use the application

Now that you've seen an SPA request a resource from a protected web API, gain a deeper understanding of how these application types interact with each other and with Azure AD B2C.

[Application types that can be used in Active Directory B2C >](#)

Azure Active Directory B2C code samples

2/26/2020 • 2 minutes to read • [Edit Online](#)

The following tables provide links to samples for applications including iOS, Android, .NET, and Node.js.

Mobile and desktop apps

SAMPLE	DESCRIPTION
ios-swift-native-msal	An iOS sample in Swift that authenticates Azure AD B2C users and calls an API using OAuth 2.0
android-native-msal	A simple Android app showcasing how to use MSAL to authenticate users via Azure Active Directory B2C, and access a Web API with the resulting tokens.
ios-native-appauth	A sample that shows how you can use a third party library to build an iOS application in Objective-C that authenticates Microsoft identity users to our Azure AD B2C identity service.
android-native-appauth	A sample that shows how you can use a third party library to build an Android application that authenticates Microsoft identity users to our B2C identity service and calls a web API using OAuth 2.0 access tokens.
dotnet-desktop	A sample that shows how a Windows Desktop .NET (WPF) application can sign in a user using Azure AD B2C, get an access token using MSAL.NET and call an API.
xamarin-native	A simple Xamarin Forms app showcasing how to use MSAL to authenticate users via Azure Active Directory B2C, and access a Web API with the resulting tokens.

Web apps and APIs

SAMPLE	DESCRIPTION
dotnet-webapp-and-webapi	A combined sample for a .NET web application that calls a .NET Web API, both secured using Azure AD B2C.
dotnetcore-webapp	An ASP.NET Core web application that can sign in a user using Azure AD B2C, get an access token using MSAL.NET and call an API.
openidconnect-nodejs	A Node.js app that provides a quick and easy way to set up a Web application with Express using OpenID Connect.
javascript-nodejs-webapi	A small node.js Web API for Azure AD B2C that shows how to protect your web api and accept B2C access tokens using passport.js.

SAMPLE	DESCRIPTION
ms-identity-python-webapp	Demonstrate how to Integrate B2C of Microsoft identity platform with a Python web application.

Single page apps

SAMPLE	DESCRIPTION
javascript-msal-singlepageapp	A single page application (SPA) calling a Web API. Authentication is done with Azure AD B2C by leveraging MSAL.js.
javascript-hellojs-singlepageapp	A single page app, implemented with an ASP.NET Web API backend, that signs up & signs in users using Azure AD B2C and calls the web API using OAuth 2.0 access tokens.

Application types that can be used in Active Directory B2C

2/20/2020 • 6 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) supports authentication for a variety of modern application architectures. All of them are based on the industry standard protocols [OAuth 2.0](#) or [OpenID Connect](#). This article describes the types of applications that you can build, independent of the language or platform you prefer. It also helps you understand the high-level scenarios before you start building applications.

Every application that uses Azure AD B2C must be registered in your [Azure AD B2C tenant](#) by using the [Azure portal](#). The application registration process collects and assigns values, such as:

- An **Application ID** that uniquely identifies your application.
- A **Reply URL** that can be used to direct responses back to your application.

Each request that is sent to Azure AD B2C specifies a **user flow** (a built-in policy) or a **custom policy** that controls the behavior of Azure AD B2C. Both policy types enable you to create a highly customizable set of user experiences.

The interaction of every application follows a similar high-level pattern:

1. The application directs the user to the v2.0 endpoint to execute a [policy](#).
2. The user completes the policy according to the policy definition.
3. The application receives a security token from the v2.0 endpoint.
4. The application uses the security token to access protected information or a protected resource.
5. The resource server validates the security token to verify that access can be granted.
6. The application periodically refreshes the security token.

These steps can differ slightly based on the type of application you're building.

Web applications

For web applications (including .NET, PHP, Java, Ruby, Python, and Node.js) that are hosted on a server and accessed through a browser, Azure AD B2C supports [OpenID Connect](#) for all user experiences. In the Azure AD B2C implementation of OpenID Connect, your web application initiates user experiences by issuing authentication requests to Azure AD. The result of the request is an `id_token`. This security token represents the user's identity. It also provides information about the user in the form of claims:

```
// Partial raw id_token  
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImtyaU1QZG1Cd...  
  
// Partial content of a decoded id_token  
{  
    "name": "John Smith",  
    "email": "john.smith@gmail.com",  
    "oid": "d9674823-dffc-4e3f-a6eb-62fe4bd48a58"  
    ...  
}
```

Learn more about the types of tokens and claims available to an application in the [Azure AD B2C token reference](#).

In a web application, each execution of a [policy](#) takes these high-level steps:

1. The user browses to the web application.
2. The web application redirects the user to Azure AD B2C indicating the policy to execute.
3. The user completes policy.
4. Azure AD B2C returns an `id_token` to the browser.
5. The `id_token` is posted to the redirect URI.
6. The `id_token` is validated and a session cookie is set.
7. A secure page is returned to the user.

Validation of the `id_token` by using a public signing key that is received from Azure AD is sufficient to verify the identity of the user. This process also sets a session cookie that can be used to identify the user on subsequent page requests.

To see this scenario in action, try one of the web application sign-in code samples in our [Getting started section](#).

In addition to facilitating simple sign-in, a web server application might also need to access a back-end web service. In this case, the web application can perform a slightly different [OpenID Connect flow](#) and acquire tokens by using authorization codes and refresh tokens. This scenario is depicted in the following [Web APIs section](#).

Web APIs

You can use Azure AD B2C to secure web services such as your application's RESTful web API. Web APIs can use OAuth 2.0 to secure their data, by authenticating incoming HTTP requests using tokens. The caller of a web API appends a token in the authorization header of an HTTP request:

```
GET /api/items HTTP/1.1
Host: www.mywebapi.com
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6...
Accept: application/json
...
...
```

The web API can then use the token to verify the API caller's identity and to extract information about the caller from claims that are encoded in the token. Learn more about the types of tokens and claims available to an app in the [Azure AD B2C token reference](#).

A web API can receive tokens from many types of clients, including web applications, desktop and mobile applications, single page applications, server-side daemons, and other web APIs. Here's an example of the complete flow for a web application that calls a web API:

1. The web application executes a policy and the user completes the user experience.
2. Azure AD B2C returns an (OpenID Connect) `id_token` and an authorization code to the browser.
3. The browser posts the `id_token` and authorization code to the redirect URI.
4. The web server validates the `id_token` and sets a session cookie.
5. The web server asks Azure AD B2C for an `access_token` by providing it with the authorization code, application client ID, and client credentials.
6. The `access_token` and `refresh_token` are returned to the web server.
7. The web API is called with the `access_token` in an authorization header.
8. The web API validates the token.
9. Secure data is returned to the web application.

To learn more about authorization codes, refresh tokens, and the steps for getting tokens, read about the [OAuth 2.0 protocol](#).

To learn how to secure a web API by using Azure AD B2C, check out the web API tutorials in our [Getting started](#)

section.

Mobile and native applications

Applications that are installed on devices, such as mobile and desktop applications, often need to access back-end services or web APIs on behalf of users. You can add customized identity management experiences to your native applications and securely call back-end services by using Azure AD B2C and the [OAuth 2.0 authorization code flow](#).

In this flow, the application executes [policies](#) and receives an `authorization_code` from Azure AD after the user completes the policy. The `authorization_code` represents the application's permission to call back-end services on behalf of the user who is currently signed in. The application can then exchange the `authorization_code` in the background for an `access_token` and a `refresh_token`. The application can use the `access_token` to authenticate to a back-end web API in HTTP requests. It can also use the `refresh_token` to get a new `access_token` when an older one expires.

Current limitations

Unsupported application types

Daemons/server-side applications

Applications that contain long-running processes or that operate without the presence of a user also need a way to access secured resources such as web APIs. These applications can authenticate and get tokens by using the application's identity (rather than a user's delegated identity) and by using the OAuth 2.0 client credentials flow. Client credential flow is not the same as on-behalf-flow and on-behalf-flow should not be used for server-to-server authentication.

Although client credential flow is not currently supported by Azure AD B2C, you can set up client credential flow using Azure AD. An Azure AD B2C tenant shares some functionality with Azure AD enterprise tenants. The client credential flow is supported using the Azure AD functionality of the Azure AD B2C tenant.

To set up client credential flow, see [Azure Active Directory v2.0 and the OAuth 2.0 client credentials flow](#). A successful authentication results in the receipt of a token formatted so that it can be used by Azure AD as described in [Azure AD token reference](#).

Web API chains (on-behalf-of flow)

Many architectures include a web API that needs to call another downstream web API, where both are secured by Azure AD B2C. This scenario is common in native clients that have a Web API back-end and calls a Microsoft online service such as the Microsoft Graph API.

This chained web API scenario can be supported by using the OAuth 2.0 JWT bearer credential grant, also known as the on-behalf-of flow. However, the on-behalf-of flow is not currently implemented in the Azure AD B2C.

Faulted apps

Do not edit Azure AD B2C applications in these ways:

- On other application management portals such as the [Application Registration Portal](#).
- Using Graph API or PowerShell.

If you edit the Azure AD B2C application outside of the Azure portal, it becomes a faulted application and is no longer usable with Azure AD B2C. Delete the application and create it again.

To delete the application, go to the [Application Registration Portal](#) and delete the application there. In order for the application to be visible, you need to be the owner of the application (and not just an admin of the tenant).

Next steps

Find out more about the built-in policies provided by [User flows in Azure Active Directory B2C](#).

Azure AD B2C: Authentication protocols

1/28/2020 • 4 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) provides identity as a service for your apps by supporting two industry standard protocols: OpenID Connect and OAuth 2.0. The service is standards-compliant, but any two implementations of these protocols can have subtle differences.

The information in this guide is useful if you write your code by directly sending and handling HTTP requests, rather than by using an open source library. We recommend that you read this page before you dive into the details of each specific protocol. But if you're already familiar with Azure AD B2C, you can go straight to [the protocol reference guides](#).

The basics

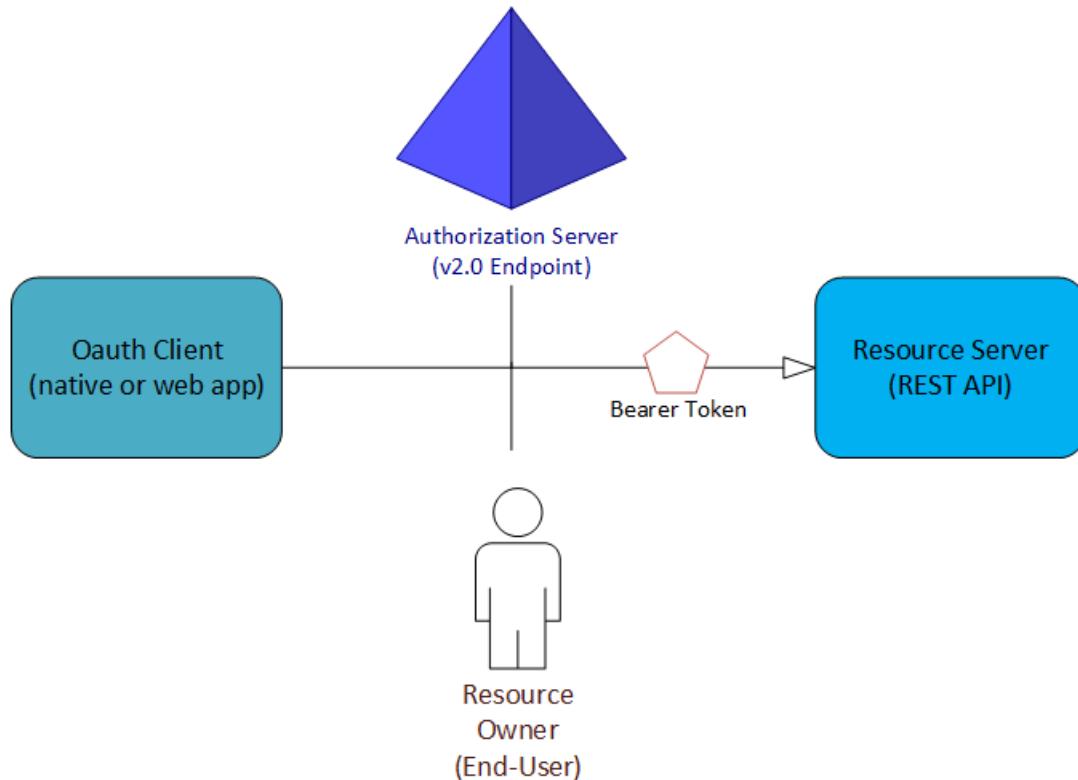
Every app that uses Azure AD B2C needs to be registered in your B2C directory in the [Azure portal](#). The app registration process collects and assigns a few values to your app:

- An **Application ID** that uniquely identifies your app.
- A **Redirect URI** or **package identifier** that can be used to direct responses back to your app.
- A few other scenario-specific values. For more information, learn [how to register your application](#).

After you register your app, it communicates with Azure Active Directory (Azure AD) by sending requests to the endpoint:

```
https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/oauth2/v2.0/authorize  
https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/oauth2/v2.0/token
```

In nearly all OAuth and OpenID Connect flows, four parties are involved in the exchange:



- The **authorization server** is the Azure AD endpoint. It securely handles anything related to user information and access. It also handles the trust relationships between the parties in a flow. It is responsible for verifying the user's identity, granting and revoking access to resources, and issuing tokens. It is also known as the identity provider.
- The **resource owner** is typically the end user. It is the party that owns the data, and it has the power to allow third parties to access that data or resource.
- The **OAuth client** is your app. It's identified by its Application ID. It's usually the party that end users interact with. It also requests tokens from the authorization server. The resource owner must grant the client permission to access the resource.
- The **resource server** is where the resource or data resides. It trusts the authorization server to securely authenticate and authorize the OAuth client. It also uses bearer access tokens to ensure that access to a resource can be granted.

Policies and user flows

Arguably, Azure AD B2C policies are the most important features of the service. Azure AD B2C extends the standard OAuth 2.0 and OpenID Connect protocols by introducing policies. These allow Azure AD B2C to perform much more than simple authentication and authorization.

To help you set up the most common identity tasks, the Azure AD B2C portal includes predefined, configurable policies called **user flows**. User flows fully describe consumer identity experiences, including sign-up, sign-in, and profile editing. User flows can be defined in an administrative UI. They can be executed by using a special query parameter in HTTP authentication requests.

Policies and user flows are not standard features of OAuth 2.0 and OpenID Connect, so you should take the time to understand them. For more information, see the [Azure AD B2C user flow reference guide](#).

Tokens

The Azure AD B2C implementation of OAuth 2.0 and OpenID Connect makes extensive use of bearer tokens, including bearer tokens that are represented as JSON web tokens (JWTs). A bearer token is a lightweight security token that grants the "bearer" access to a protected resource.

The bearer is any party that can present the token. Azure AD must first authenticate a party before it can receive a bearer token. But if the required steps are not taken to secure the token in transmission and storage, it can be intercepted and used by an unintended party.

Some security tokens have built-in mechanisms that prevent unauthorized parties from using them, but bearer tokens do not have this mechanism. They must be transported in a secure channel, such as a transport layer security (HTTPS).

If a bearer token is transmitted outside a secure channel, a malicious party can use a man-in-the-middle attack to acquire the token and use it to gain unauthorized access to a protected resource. The same security principles apply when bearer tokens are stored or cached for later use. Always ensure that your app transmits and stores bearer tokens in a secure manner.

For additional bearer token security considerations, see [RFC 6750 Section 5](#).

More information about the different types of tokens that are used in Azure AD B2C are available in [the Azure AD token reference](#).

Protocols

When you're ready to review some example requests, you can start with one of the following tutorials. Each

corresponds to a particular authentication scenario. If you need help determining which flow is right for you, check out [the types of apps you can build by using Azure AD B2C](#).

- [Build mobile and native applications by using OAuth 2.0](#)
- [Build web apps by using OpenID Connect](#)
- [Build single-page apps using the OAuth 2.0 implicit flow](#)

Web sign-in with OpenID Connect in Azure Active Directory B2C

1/28/2020 • 15 minutes to read • [Edit Online](#)

OpenID Connect is an authentication protocol, built on top of OAuth 2.0, that can be used to securely sign users in to web applications. By using the Azure Active Directory B2C (Azure AD B2C) implementation of OpenID Connect, you can outsource sign-up, sign-in, and other identity management experiences in your web applications to Azure Active Directory (Azure AD). This guide shows you how to do so in a language-independent manner. It describes how to send and receive HTTP messages without using any of our open-source libraries.

[OpenID Connect](#) extends the OAuth 2.0 *authorization* protocol for use as an *authentication* protocol. This authentication protocol allows you to perform single sign-on. It introduces the concept of an *ID token*, which allows the client to verify the identity of the user and obtain basic profile information about the user.

Because it extends OAuth 2.0, it also enables applications to securely acquire *access tokens*. You can use access tokens to access resources that are secured by an [authorization server](#). OpenID Connect is recommended if you're building a web application that's hosted on a server and accessed through a browser. For more information about tokens, see the [Overview of tokens in Azure Active Directory B2C](#).

Azure AD B2C extends the standard OpenID Connect protocol to do more than simple authentication and authorization. It introduces the [user flow parameter](#), which enables you to use OpenID Connect to add user experiences to your application, such as sign-up, sign-in, and profile management.

Send authentication requests

When your web application needs to authenticate the user and run a user flow, it can direct the user to the `/authorize` endpoint. The user takes action depending on the user flow.

In this request, the client indicates the permissions that it needs to acquire from the user in the `scope` parameter, and specifies the user flow to run. To get a feel for how the request works, try pasting the request into a browser and running it. Replace `{tenant}` with the name of your tenant. Replace `90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6` with the app ID of the application you've previously registered in your tenant. Also change the policy name (`{policy}`) to the policy name that you have in your tenant, for example `b2c_1_sign_in`.

```
GET https://{tenant}.b2clogin.com/{tenant}.onmicrosoft.com/{policy}/oauth2/v2.0/authorize?  
client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6  
&response_type=code+id_token  
&redirect_uri=https%3A%2F%2Faadb2cplayground.azurewebsites.net%2F  
&response_mode=form_post  
&scope=openid%20offline_access  
&state=arbitrary_data_you_can_receive_in_the_response  
&nonce=12345
```

PARAMETER	REQUIRED	DESCRIPTION
<code>{tenant}</code>	Yes	Name of your Azure AD B2C tenant

PARAMETER	REQUIRED	DESCRIPTION
{policy}	Yes	The user flow to be run. Specify the name of a user flow you've created in your Azure AD B2C tenant. For example: <code>b2c_1_sign_in</code> , <code>b2c_1_sign_up</code> , or <code>b2c_1_edit_profile</code> .
client_id	Yes	The application ID that the Azure portal assigned to your application.
nonce	Yes	A value included in the request (generated by the application) that is included in the resulting ID token as a claim. The application can then verify this value to mitigate token replay attacks. The value is typically a randomized unique string that can be used to identify the origin of the request.
response_type	Yes	Must include an ID token for OpenID Connect. If your web application also needs tokens for calling a web API, you can use <code>code+id_token</code> .
scope	Yes	A space-separated list of scopes. The <code>openid</code> scope indicates a permission to sign in the user and get data about the user in the form of ID tokens. The <code>offline_access</code> scope is optional for web applications. It indicates that your application will need a <i>refresh token</i> for extended access to resources.
prompt	No	The type of user interaction that's required. The only valid value at this time is <code>login</code> , which forces the user to enter their credentials on that request.
redirect_uri	No	The <code>redirect_uri</code> parameter of your application, where authentication responses can be sent and received by your application. It must exactly match one of the <code>redirect_uri</code> parameters that you registered in the Azure portal, except that it must be URL encoded.
response_mode	No	The method that is used to send the resulting authorization code back to your application. It can be either <code>query</code> , <code>form_post</code> , or <code>fragment</code> . The <code>form_post</code> response mode is recommended for best security.

PARAMETER	REQUIRED	DESCRIPTION
state	No	A value included in the request that's also returned in the token response. It can be a string of any content that you want. A randomly generated unique value is typically used for preventing cross-site request forgery attacks. The state is also used to encode information about the user's state in the application before the authentication request occurred, such as the page they were on.

At this point, the user is asked to complete the workflow. The user might have to enter their username and password, sign in with a social identity, or sign up for the directory. There could be any other number of steps depending on how the user flow is defined.

After the user completes the user flow, a response is returned to your application at the indicated `redirect_uri` parameter, by using the method that's specified in the `response_mode` parameter. The response is the same for each of the preceding cases, independent of the user flow.

A successful response using `response_mode=fragment` would look like:

```
GET https://aad2cplayground.azurewebsites.net/
id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCIk5HVEZ2ZEstZnl0aEV1Q...
&code=AwABAAAAvPM1KaPlrEqdFSBzjqFTGBCmLdgfSTLEMPGYuNHSUYBrq...
&state=arbitrary_data_you_can_receive_in_the_response
```

PARAMETER	DESCRIPTION
id_token	The ID token that the application requested. You can use the ID token to verify the user's identity and begin a session with the user.
code	The authorization code that the application requested, if you used <code>response_type=code+id_token</code> . The application can use the authorization code to request an access token for a target resource. Authorization codes typically expire after about 10 minutes.
state	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The application should verify that the <code>state</code> values in the request and response are identical.

Error responses can also be sent to the `redirect_uri` parameter so that the application can handle them appropriately:

```
GET https://aad2cplayground.azurewebsites.net/
error=access_denied
&error_description=the+user+canceled+the+authentication
&state=arbitrary_data_you_can_receive_in_the_response
```

PARAMETER	DESCRIPTION
error	A code that can be used to classify the types of errors that occur.
error_description	A specific error message that can help identify the root cause of an authentication error.
state	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The application should verify that the <code>state</code> values in the request and response are identical.

Validate the ID token

Just receiving an ID token is not enough to authenticate the user. Validate the ID token's signature and verify the claims in the token per your application's requirements. Azure AD B2C uses [JSON Web Tokens \(JWTs\)](#) and public key cryptography to sign tokens and verify that they are valid. There are many open-source libraries that are available for validating JWTs, depending on your language of preference. We recommend exploring those options rather than implementing your own validation logic.

Azure AD B2C has an OpenID Connect metadata endpoint, which allows an application to get information about Azure AD B2C at runtime. This information includes endpoints, token contents, and token signing keys. There is a JSON metadata document for each user flow in your B2C tenant. For example, the metadata document for the `b2c_1_sign_in` user flow in `fabrikamb2c.onmicrosoft.com` is located at:

```
https://fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_sign_in/v2.0/.well-known/openid-configuration
```

One of the properties of this configuration document is `jwks_uri`, whose value for the same user flow would be:

```
https://fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_sign_in/discovery/v2.0/keys
```

To determine which user flow was used in signing an ID token (and from where to get the metadata), you have two options. First, the user flow name is included in the `acr` claim in the ID token. Your other option is to encode the user flow in the value of the `state` parameter when you issue the request, and then decode it to determine which user flow was used. Either method is valid.

After you've acquired the metadata document from the OpenID Connect metadata endpoint, you can use the RSA 256 public keys to validate the signature of the ID token. There might be multiple keys listed at this endpoint, each identified by a `kid` claim. The header of the ID token also contains a `kid` claim, which indicates which of these keys was used to sign the ID token.

To verify the tokens from Azure AD B2C, you need to generate the public key using the exponent(e) and modulus(n). You need to determine how to do this in your respective programming language accordingly. The official documentation on Public Key generation with the RSA protocol can be found here:

<https://tools.ietf.org/html/rfc3447#section-3.1>

After you've validated the signature of the ID token, there are several claims that you need to verify. For instance:

- Validate the `nonce` claim to prevent token replay attacks. Its value should be what you specified in the sign-in request.
- Validate the `aud` claim to ensure that the ID token was issued for your application. Its value should be the application ID of your application.

- Validate the `iat` and `exp` claims to make sure that the ID token hasn't expired.

There are also several more validations that you should perform. The validations are described in detail in the [OpenID Connect Core Spec](#). You might also want to validate additional claims, depending on your scenario. Some common validations include:

- Ensuring that the user/organization has signed up for the application.
- Ensuring that the user has proper authorization/privileges.
- Ensuring that a certain strength of authentication has occurred, such as Azure Multi-Factor Authentication.

After you validate the ID token, you can begin a session with the user. You can use the claims in the ID token to obtain information about the user in your application. Uses for this information include display, records, and authorization.

Get a token

If you need your web application to only run user flows, you can skip the next few sections. These sections are applicable only to web applications that need to make authenticated calls to a web API and are also protected by Azure AD B2C.

You can redeem the authorization code that you acquired (by using `response_type=code+id_token`) for a token to the desired resource by sending a `POST` request to the `/token` endpoint. In Azure AD B2C, you can [request access tokens for other APIs](#) as usual by specifying their scope(s) in the request.

You can also request an access token for your app's own back-end Web API by convention of using the app's client ID as the requested scope (which will result in an access token with that client ID as the "audience"):

```
POST {tenant}.onmicrosoft.com/{policy}/oauth2/v2.0/token HTTP/1.1
Host: {tenant}.b2clogin.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6&scope=90c0fe63-bcf2-44d5-8fb7-
b8bbc0b29dc6
offline_access&code=AwABAAAAvPM1KaPlrEqdFSBzjqfTGBCmLdgfSTLEMPGYuNHSUYBrq...&redirect_uri=urn:ietf:wg:oauth:2
.0:oob
```

PARAMETER	REQUIRED	DESCRIPTION
{tenant}	Yes	Name of your Azure AD B2C tenant
{policy}	Yes	The user flow that was used to acquire the authorization code. You can't use a different user flow in this request. Add this parameter to the query string, not to the POST body.
client_id	Yes	The application ID that the Azure portal assigned to your application.

PARAMETER	REQUIRED	DESCRIPTION
client_secret	Yes, in Web Apps	The application secret that was generated in the Azure portal . Client secrets are used in this flow for Web App scenarios, where the client can securely store a client secret. For Native App (public client) scenarios, client secrets cannot be securely stored, therefore not used on this flow. If using a client secret, please change it on a periodic basis.
code	Yes	The authorization code that you acquired in the beginning of the user flow.
grant_type	Yes	The type of grant, which must be <code>authorization_code</code> for the authorization code flow.
redirect_uri	Yes	The <code>redirect_uri</code> parameter of the application where you received the authorization code.
scope	No	A space-separated list of scopes. The <code>openid</code> scope indicates a permission to sign in the user and get data about the user in the form of <code>id_token</code> parameters. It can be used to get tokens to your application's own back-end web API, which is represented by the same application ID as the client. The <code>offline_access</code> scope indicates that your application needs a refresh token for extended access to resources.

A successful token response looks like:

```
{
  "not_before": "1442340812",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIngldCI6Ik5HVEZ2ZEstZn10aEV1Q...",
  "scope": "90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6 offline_access",
  "expires_in": "3600",
  "refresh_token": "AAQfQmvuDy8WtUv-sd0TBwVVQs1rC-Lfxa_NDkLqpg50Cxp5Dxj0VPF1mx2Z..."
}
```

PARAMETER	DESCRIPTION
not_before	The time at which the token is considered valid, in epoch time.
token_type	The token type value. <code>Bearer</code> is the only type that is supported.
access_token	The signed JWT token that you requested.

PARAMETER	DESCRIPTION
scope	The scopes for which the token is valid.
expires_in	The length of time that the access token is valid (in seconds).
refresh_token	An OAuth 2.0 refresh token. The application can use this token to acquire additional tokens after the current token expires. Refresh tokens can be used to retain access to resources for extended periods of time. The scope <code>offline_access</code> must have been used in both the authorization and token requests in order to receive a refresh token.

Error responses look like:

```
{
  "error": "access_denied",
  "error_description": "The user revoked access to the app.",
}
```

PARAMETER	DESCRIPTION
error	A code that can be used to classify types of errors that occur.
error_description	A message that can help identify the root cause of an authentication error.

Use the token

Now that you've successfully acquired an access token, you can use the token in requests to your back-end web APIs by including it in the `Authorization` header:

```
GET /tasks
Host: mytaskwebapi.com
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...
```

Refresh the token

ID tokens expire in a short period of time. Refresh the tokens after they expire to continue being able to access resources. You can refresh a token by submitting another `POST` request to the `/token` endpoint. This time, provide the `refresh_token` parameter instead of the `code` parameter:

```
POST {tenant}.onmicrosoft.com/{policy}/oauth2/v2.0/token HTTP/1.1
Host: {tenant}.b2clogin.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6&scope=openid
offline_access&refresh_token=AwABAAAAvPM1KaPlrEqdFSBzjqfTGBcmLdgfSTLEMPGYuNHSUYBrq...&redirect_uri=urn:ietf:w
g:oauth:2.0:oob
```

PARAMETER	REQUIRED	DESCRIPTION
{tenant}	Yes	Name of your Azure AD B2C tenant
{policy}	Yes	The user flow that was used to acquire the original refresh token. You can't use a different user flow in this request. Add this parameter to the query string, not to the POST body.
client_id	Yes	The application ID that the Azure portal assigned to your application.
client_secret	Yes, in Web Apps	The application secret that was generated in the Azure portal . Client secrets are used in this flow for Web App scenarios, where the client can securely store a client secret. For Native App (public client) scenarios, client secrets cannot be securely stored, therefore not used on this call. If using a client secret, please change it on a periodic basis.
grant_type	Yes	The type of grant, which must be a refresh token for this part of the authorization code flow.
refresh_token	Yes	The original refresh token that was acquired in the second part of the flow. The <code>offline_access</code> scope must be used in both the authorization and token requests in order to receive a refresh token.
redirect_uri	No	The <code>redirect_uri</code> parameter of the application where you received the authorization code.
scope	No	A space-separated list of scopes. The <code>openid</code> scope indicates a permission to sign in the user and get data about the user in the form of ID tokens. It can be used to send tokens to your application's own back-end web API, which is represented by the same application ID as the client. The <code>offline_access</code> scope indicates that your application needs a refresh token for extended access to resources.

A successful token response looks like:

```
{
  "not_before": "1442340812",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
  "scope": "90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6 offline_access",
  "expires_in": "3600",
  "refresh_token": "AAQfQmvuDy8WtUv-sd0TBwWVQs1rC-Lfxa_NDkLqpg50Cxp5Dxj0VPF1mx2Z...",
}
```

PARAMETER	DESCRIPTION
not_before	The time at which the token is considered valid, in epoch time.
token_type	The token type value. <code>Bearer</code> is the only type that is supported.
access_token	The signed JWT token that was requested.
scope	The scope for which the token is valid.
expires_in	The length of time that the access token is valid (in seconds).
refresh_token	An OAuth 2.0 refresh token. The application can use this token to acquire additional tokens after the current token expires. Refresh tokens can be used to retain access to resources for extended periods of time.

Error responses look like:

```
{
  "error": "access_denied",
  "error_description": "The user revoked access to the app.",
}
```

PARAMETER	DESCRIPTION
error	A code that can be used to classify types of errors that occur.
error_description	A message that can help identify the root cause of an authentication error.

Send a sign-out request

When you want to sign the user out of the application, it isn't enough to clear the application's cookies or otherwise end the session with the user. Redirect the user to Azure AD B2C to sign out. If you fail to do so, the user might be able to reauthenticate to your application without entering their credentials again.

To sign out the user, redirect the user to the `end_session` endpoint that is listed in the OpenID Connect metadata document described earlier:

```
GET https://[tenant].b2clogin.com/[tenant].onmicrosoft.com/[policy]/oauth2/v2.0/logout?
post_logout_redirect_uri=https%3A%2F%2Fjwt.ms%2F
```

PARAMETER	REQUIRED	DESCRIPTION
{tenant}	Yes	Name of your Azure AD B2C tenant
{policy}	Yes	The user flow that you want to use to sign the user out of your application.
id_token_hint	No	A previously issued ID token to pass to the logout endpoint as a hint about the end user's current authenticated session with the client. The <code>id_token_hint</code> ensures that the <code>post_logout_redirect_uri</code> is a registered reply URL in your Azure AD B2C application settings.
client_id	No*	The application ID that the Azure portal assigned to your application. <i>*This is required when using Application isolation SSO configuration and Require ID Token in logout request is set to No.</i>
post_logout_redirect_uri	No	The URL that the user should be redirected to after successful sign out. If it isn't included, Azure AD B2C shows the user a generic message. Unless you provide an <code>id_token_hint</code> , you should not register this URL as a reply URL in your Azure AD B2C application settings.
state	No	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The application should verify that the <code>state</code> values in the request and response are identical.

Secure your logout redirect

After logout, the user is redirected to the URI specified in the `post_logout_redirect_uri` parameter, regardless of the reply URLs that have been specified for the application. However, if a valid `id_token_hint` is passed, Azure AD B2C verifies that the value of `post_logout_redirect_uri` matches one of the application's configured redirect URIs before performing the redirect. If no matching reply URL was configured for the application, an error message is displayed and the user is not redirected.

External identity provider sign-out

Directing the user to the `end_session` endpoint clears some of the user's single sign-on state with Azure AD B2C, but it doesn't sign the user out of their social identity provider (IDP) session. If the user selects the same IDP during a subsequent sign-in, they are reauthenticated without entering their credentials. If a user wants to sign out of the application, it doesn't necessarily mean they want to sign out of their Facebook account. However, if local accounts are used, the user's session ends properly.

OAuth 2.0 authorization code flow in Azure Active Directory B2C

1/28/2020 • 11 minutes to read • [Edit Online](#)

You can use the OAuth 2.0 authorization code grant in apps installed on a device to gain access to protected resources, such as web APIs. By using the Azure Active Directory B2C (Azure AD B2C) implementation of OAuth 2.0, you can add sign-up, sign-in, and other identity management tasks to your mobile and desktop apps. This article is language-independent. In the article, we describe how to send and receive HTTP messages without using any open-source libraries.

The OAuth 2.0 authorization code flow is described in [section 4.1 of the OAuth 2.0 specification](#). You can use it for authentication and authorization in most [application types](#), including web applications and natively installed applications. You can use the OAuth 2.0 authorization code flow to securely acquire access tokens and refresh tokens for your applications, which can be used to access resources that are secured by an [authorization server](#). The refresh token allows the client to acquire new access (and refresh) tokens once the access token expires, typically after one hour.

This article focuses on the **public clients** OAuth 2.0 authorization code flow. A public client is any client application that cannot be trusted to securely maintain the integrity of a secret password. This includes mobile apps, desktop applications, and essentially any application that runs on a device and needs to get access tokens.

NOTE

To add identity management to a web app by using Azure AD B2C, use [OpenID Connect](#) instead of OAuth 2.0.

Azure AD B2C extends the standard OAuth 2.0 flows to do more than simple authentication and authorization. It introduces the [user flow](#). With user flows, you can use OAuth 2.0 to add user experiences to your application, such as sign-up, sign-in, and profile management. Identity providers that use the OAuth 2.0 protocol include [Amazon](#), [Azure Active Directory](#), [Facebook](#), [GitHub](#), [Google](#), and [LinkedIn](#).

To try the HTTP requests in this article:

1. Replace `{tenant}` with the name of your Azure AD B2C tenant.
2. Replace `90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6` with the app ID of an application you've previously registered in your Azure AD B2C tenant.
3. Replace `{policy}` with the name of a policy you've created in your tenant, for example `b2c_1_sign_in`.

1. Get an authorization code

The authorization code flow begins with the client directing the user to the `/authorize` endpoint. This is the interactive part of the flow, where the user takes action. In this request, the client indicates in the `scope` parameter the permissions that it needs to acquire from the user. The following three examples (with line breaks for readability) each use a different user flow.

```

GET https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/{{policy}}/oauth2/v2.0/authorize?
client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6
&response_type=code
&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob
&response_mode=query
&scope=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6%20offline_access
&state=arbitrary_data_you_can_receive_in_the_response

```

PARAMETER	REQUIRED?	DESCRIPTION
{tenant}	Required	Name of your Azure AD B2C tenant
{policy}	Required	The user flow to be run. Specify the name of a user flow you've created in your Azure AD B2C tenant. For example: <code>b2c_1_sign_in</code> , <code>b2c_1_sign_up</code> , or <code>b2c_1_edit_profile</code> .
client_id	Required	The application ID assigned to your app in the Azure portal .
response_type	Required	The response type, which must include <code>code</code> for the authorization code flow.
redirect_uri	Required	The redirect URI of your app, where authentication responses are sent and received by your app. It must exactly match one of the redirect URIs that you registered in the portal, except that it must be URL-encoded.
scope	Required	A space-separated list of scopes. A single scope value indicates to Azure Active Directory (Azure AD) both of the permissions that are being requested. Using the client ID as the scope indicates that your app needs an access token that can be used against your own service or web API, represented by the same client ID. The <code>offline_access</code> scope indicates that your app needs a refresh token for long-lived access to resources. You also can use the <code>openid</code> scope to request an ID token from Azure AD B2C.
response_mode	Recommended	The method that you use to send the resulting authorization code back to your app. It can be <code>query</code> , <code>form_post</code> , or <code>fragment</code> .

PARAMETER	REQUIRED?	DESCRIPTION
state	Recommended	A value included in the request that can be a string of any content that you want to use. Usually, a randomly generated unique value is used, to prevent cross-site request forgery attacks. The state also is used to encode information about the user's state in the app before the authentication request occurred. For example, the page the user was on, or the user flow that was being executed.
prompt	Optional	The type of user interaction that is required. Currently, the only valid value is <code>login</code> , which forces the user to enter their credentials on that request. Single sign-on will not take effect.

At this point, the user is asked to complete the user flow's workflow. This might involve the user entering their username and password, signing in with a social identity, signing up for the directory, or any other number of steps. User actions depend on how the user flow is defined.

After the user completes the user flow, Azure AD returns a response to your app at the value you used for `redirect_uri`. It uses the method specified in the `response_mode` parameter. The response is exactly the same for each of the user action scenarios, independent of the user flow that was executed.

A successful response that uses `response_mode=query` looks like this:

```
GET urn:ietf:wg:oauth:2.0:oob?
code=AwABAAA... // the authorization_code, truncated
&state=arbitrary_data_you_can_receive_in_the_response // the value provided in the request
```

PARAMETER	DESCRIPTION
code	The authorization code that the app requested. The app can use the authorization code to request an access token for a target resource. Authorization codes are very short-lived. Typically, they expire after about 10 minutes.
state	See the full description in the table in the preceding section. If a <code>state</code> parameter is included in the request, the same value should appear in the response. The app should verify that the <code>state</code> values in the request and response are identical.

Error responses also can be sent to the redirect URI so that the app can handle them appropriately:

```
GET urn:ietf:wg:oauth:2.0:oob?
error=access_denied
&error_description=The+user+has+cancelled+entering+self-asserted+information
&state=arbitrary_data_you_can_receive_in_the_response
```

PARAMETER	DESCRIPTION
error	An error code string that you can use to classify the types of errors that occur. You also can use the string to react to errors.
error_description	A specific error message that can help you identify the root cause of an authentication error.
state	See the full description in the preceding table. If a <code>state</code> parameter is included in the request, the same value should appear in the response. The app should verify that the <code>state</code> values in the request and response are identical.

2. Get a token

Now that you've acquired an authorization code, you can redeem the `code` for a token to the intended resource by sending a POST request to the `/token` endpoint. In Azure AD B2C, you can [request access tokens for other API's](#) as usual by specifying their scope(s) in the request.

You can also request an access token for your app's own back-end Web API by convention of using the app's client ID as the requested scope (which will result in an access token with that client ID as the "audience"):

```
POST https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/{{policy}}/oauth2/v2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6&scope=90c0fe63-bcf2-44d5-8fb7-
b8bbc0b29dc6
offline_access&code=AwABAAAAvPM1KaPlrEqdFSBzjqfTGBcmLdgfSTLEMPGYuNHSUYBrq...&redirect_uri=urn:ietf:wg:oauth:2
.0:oob
```

PARAMETER	REQUIRED?	DESCRIPTION
{tenant}	Required	Name of your Azure AD B2C tenant
{policy}	Required	The user flow that was used to acquire the authorization code. You cannot use a different user flow in this request.
client_id	Required	The application ID assigned to your app in the Azure portal .
client_secret	Yes, in Web Apps	The application secret that was generated in the Azure portal . Client secrets are used in this flow for Web App scenarios, where the client can securely store a client secret. For Native App (public client) scenarios, client secrets cannot be securely stored, and therefore are not used in this call. If you use a client secret, please change it on a periodic basis.

PARAMETER	REQUIRED?	DESCRIPTION
grant_type	Required	The type of grant. For the authorization code flow, the grant type must be <code>authorization_code</code> .
scope	Recommended	A space-separated list of scopes. A single scope value indicates to Azure AD both of the permissions that are being requested. Using the client ID as the scope indicates that your app needs an access token that can be used against your own service or web API, represented by the same client ID. The <code>offline_access</code> scope indicates that your app needs a refresh token for long-lived access to resources. You also can use the <code>openid</code> scope to request an ID token from Azure AD B2C.
code	Required	The authorization code that you acquired in the first leg of the flow.
redirect_uri	Required	The redirect URI of the application where you received the authorization code.

A successful token response looks like this:

```
{
  "not_before": "1442340812",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZn10aEV1Q...",
  "scope": "90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6 offline_access",
  "expires_in": "3600",
  "refresh_token": "AAQfQmvuDy8WtUv-sd0TBwWVQs1rC-Lfxa_NDkLqpg50Cxp5Dxj0VPF1mx2Z...",
}
```

PARAMETER	DESCRIPTION
not_before	The time at which the token is considered valid, in epoch time.
token_type	The token type value. The only type that Azure AD supports is Bearer.
access_token	The signed JSON Web Token (JWT) that you requested.
scope	The scopes that the token is valid for. You also can use scopes to cache tokens for later use.
expires_in	The length of time that the token is valid (in seconds).

PARAMETER	DESCRIPTION
refresh_token	An OAuth 2.0 refresh token. The app can use this token to acquire additional tokens after the current token expires. Refresh tokens are long-lived. You can use them to retain access to resources for extended periods of time. For more information, see the Azure AD B2C token reference .

Error responses look like this:

```
{
  "error": "access_denied",
  "error_description": "The user revoked access to the app.",
}
```

PARAMETER	DESCRIPTION
error	An error code string that you can use to classify the types of errors that occur. You also can use the string to react to errors.
error_description	A specific error message that can help you identify the root cause of an authentication error.

3. Use the token

Now that you've successfully acquired an access token, you can use the token in requests to your back-end web APIs by including it in the `Authorization` header:

```
GET /tasks
Host: mytaskwebapi.com
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...
```

4. Refresh the token

Access tokens and ID tokens are short-lived. After they expire, you must refresh them to continue to access resources. To do this, submit another POST request to the `/token` endpoint. This time, provide the `refresh_token` instead of the `code`:

```
POST https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/{{policy}}/oauth2/v2.0/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6&scope=90c0fe63-bcf2-44d5-8fb7-
b8bbc0b29dc6
offline_access&refresh_token=AwABAAAAvPM1KaPlrEqdFSBzjqfTGBCmLdgfSTLEMPGYuNHSUYBrq...&redirect_uri=urn:ietf:w
g:oauth:2.0:oob
```

PARAMETER	REQUIRED?	DESCRIPTION
{tenant}	Required	Name of your Azure AD B2C tenant

Parameter	Required?	Description
{policy}	Required	The user flow that was used to acquire the original refresh token. You cannot use a different user flow in this request.
client_id	Required	The application ID assigned to your app in the Azure portal .
client_secret	Yes, in Web Apps	The application secret that was generated in the Azure portal . Client secrets are used in this flow for Web App scenarios, where the client can securely store a client secret. For Native App (public client) scenarios, client secrets cannot be securely stored, and therefore are not used in this call. If you use a client secret, please change it on a periodic basis.
grant_type	Required	The type of grant. For this leg of the authorization code flow, the grant type must be <code>refresh_token</code> .
scope	Recommended	A space-separated list of scopes. A single scope value indicates to Azure AD both of the permissions that are being requested. Using the client ID as the scope indicates that your app needs an access token that can be used against your own service or web API, represented by the same client ID. The <code>offline_access</code> scope indicates that your app will need a refresh token for long-lived access to resources. You also can use the <code>openid</code> scope to request an ID token from Azure AD B2C.
redirect_uri	Optional	The redirect URI of the application where you received the authorization code.
refresh_token	Required	The original refresh token that you acquired in the second leg of the flow.

A successful token response looks like this:

```
{
  "not_before": "1442340812",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZn10aEV1Q...",
  "scope": "90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6 offline_access",
  "expires_in": "3600",
  "refresh_token": "AAQfQmvuDy8WtUv-sd0TBwWVQs1rC-Lfxa_NDkLqpg50Cxp5Dxj0VPF1mx2Z...",
}
```

PARAMETER	DESCRIPTION
not_before	The time at which the token is considered valid, in epoch time.
token_type	The token type value. The only type that Azure AD supports is Bearer.
access_token	The signed JWT that you requested.
scope	The scopes that the token is valid for. You also can use the scopes to cache tokens for later use.
expires_in	The length of time that the token is valid (in seconds).
refresh_token	An OAuth 2.0 refresh token. The app can use this token to acquire additional tokens after the current token expires. Refresh tokens are long-lived, and can be used to retain access to resources for extended periods of time. For more information, see the Azure AD B2C token reference .

Error responses look like this:

```
{
  "error": "access_denied",
  "error_description": "The user revoked access to the app."
}
```

PARAMETER	DESCRIPTION
error	An error code string that you can use to classify types of errors that occur. You also can use the string to react to errors.
error_description	A specific error message that can help you identify the root cause of an authentication error.

Use your own Azure AD B2C directory

To try these requests yourself, complete the following steps. Replace the example values we used in this article with your own values.

1. [Create an Azure AD B2C directory](#). Use the name of your directory in the requests.
2. [Create an application](#) to obtain an application ID and a redirect URI. Include a native client in your app.
3. [Create your user flows](#) to obtain your user flow names.

Single-page sign in using the OAuth 2.0 implicit flow in Azure Active Directory B2C

1/28/2020 • 15 minutes to read • [Edit Online](#)

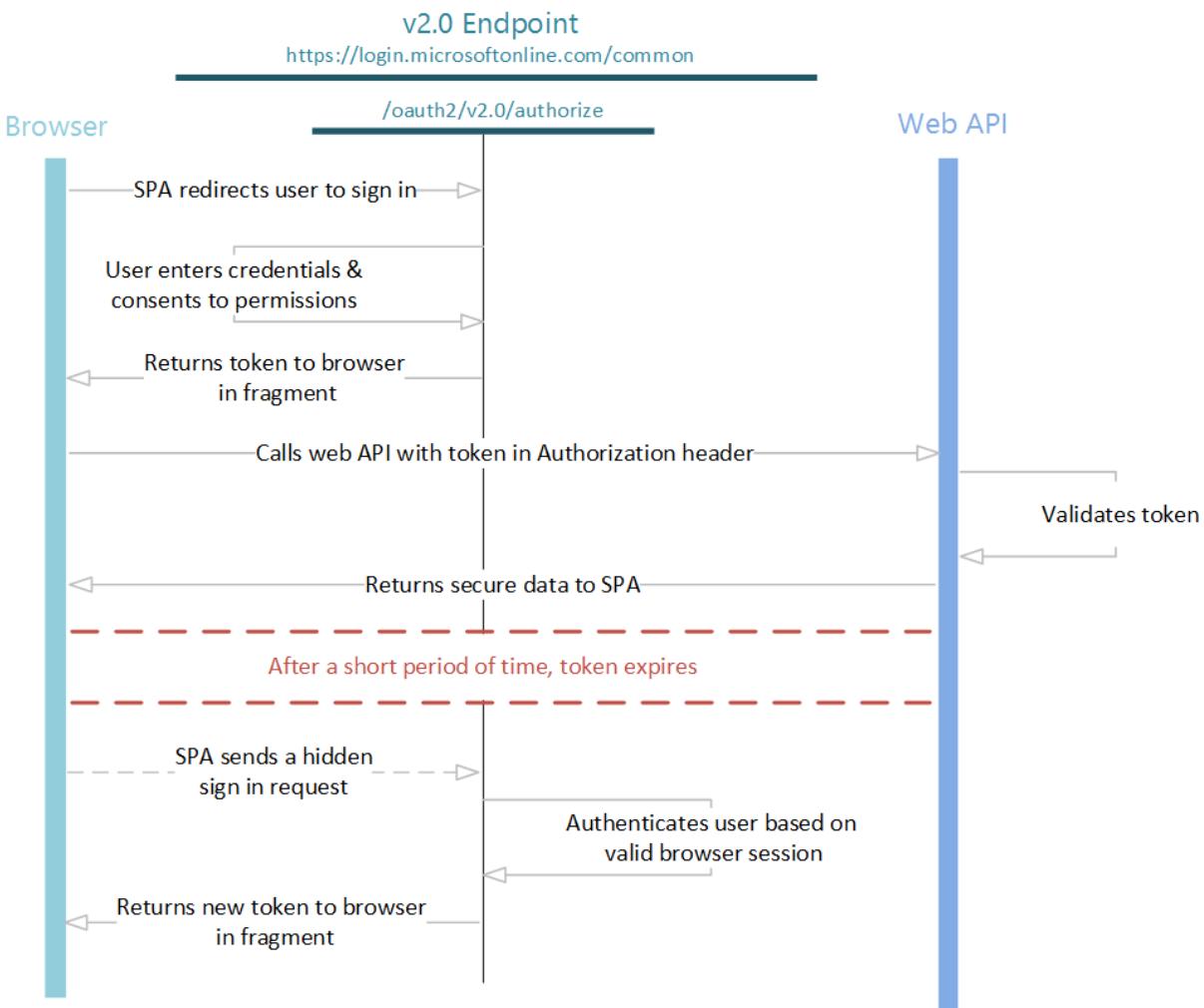
Many modern applications have a single-page app front end that is written primarily in JavaScript. Often, the app is written by using a framework like React, Angular, or Vue.js. Single-page apps and other JavaScript apps that run primarily in a browser have some additional challenges for authentication:

- The security characteristics of these apps are different from traditional server-based web applications.
- Many authorization servers and identity providers do not support cross-origin resource sharing (CORS) requests.
- Full-page browser redirects away from the app can be invasive to the user experience.

To support these applications, Azure Active Directory B2C (Azure AD B2C) uses the OAuth 2.0 implicit flow. The OAuth 2.0 authorization implicit grant flow is described in [section 4.2 of the OAuth 2.0 specification](#). In implicit flow, the app receives tokens directly from the Azure Active Directory (Azure AD) authorize endpoint, without any server-to-server exchange. All authentication logic and session handling is done entirely in the JavaScript client with either a page redirect or a pop-up box.

Azure AD B2C extends the standard OAuth 2.0 implicit flow to more than simple authentication and authorization. Azure AD B2C introduces the [policy parameter](#). With the policy parameter, you can use OAuth 2.0 to add policies to your app, such as sign-up, sign-in, and profile management user flows. In the example HTTP requests in this article, **{tenant}.onmicrosoft.com** is used as an example. Replace `{tenant}` with the name of your tenant if you have one and have also created a user flow.

The implicit sign-in flow looks something like the following figure. Each step is described in detail later in the article.



Send authentication requests

When your web application needs to authenticate the user and run a user flow, it can direct the user to the `/authorize` endpoint. The user takes action depending on the user flow.

In this request, the client indicates the permissions that it needs to acquire from the user in the `scope` parameter and the user flow to run. To get a feel for how the request works, try pasting the request into a browser and running it. Replace `{tenant}` with the name of your Azure AD B2C tenant. Replace `90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6` with the app ID of the application you've previously registered in your tenant. Replace `{policy}` with the name of a policy you've created in your tenant, for example `b2c_1_sign_in`.

```

GET https://'{tenant}'.b2clogin.com/{tenant}.onmicrosoft.com/{policy}/oauth2/v2.0/authorize?
client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6
&response_type=id_token+token
&redirect_uri=https%3A%2F%2Faadb2cplayground.azurewebsites.net%2F
&response_mode=fragment
&scope=openid%20offline_access
&state=arbitrary_data_you_can_receive_in_the_response
&nonce=12345
  
```

PARAMETER	REQUIRED	DESCRIPTION
<code>{tenant}</code>	Yes	Name of your Azure AD B2C tenant

PARAMETER	REQUIRED	DESCRIPTION
{policy}	Yes	The user flow to be run. Specify the name of a user flow you've created in your Azure AD B2C tenant. For example: <code>b2c_1_sign_in</code> , <code>b2c_1_sign_up</code> , or <code>b2c_1_edit_profile</code> .
client_id	Yes	The application ID that the Azure portal assigned to your application.
response_type	Yes	Must include <code>id_token</code> for OpenID Connect sign-in. It also can include the response type <code>token</code> . If you use <code>token</code> , your app can immediately receive an access token from the authorize endpoint, without making a second request to the authorize endpoint. If you use the <code>token</code> response type, the <code>scope</code> parameter must contain a scope that indicates which resource to issue the token for.
redirect_uri	No	The redirect URI of your app, where authentication responses can be sent and received by your app. It must exactly match one of the redirect URIs that you registered in the portal, except that it must be URL-encoded.
response_mode	No	Specifies the method to use to send the resulting token back to your app. For implicit flows, use <code>fragment</code> .
scope	Yes	A space-separated list of scopes. A single scope value indicates to Azure AD both of the permissions that are being requested. The <code>openid</code> scope indicates a permission to sign in the user and get data about the user in the form of ID tokens. The <code>offline_access</code> scope is optional for web apps. It indicates that your app needs a refresh token for long-lived access to resources.
state	No	A value included in the request that also is returned in the token response. It can be a string of any content that you want to use. Usually, a randomly generated, unique value is used, to prevent cross-site request forgery attacks. The state is also used to encode information about the user's state in the app before the authentication request occurred, like the page they were on.

PARAMETER	REQUIRED	DESCRIPTION
nonce	Yes	A value included in the request (generated by the app) that is included in the resulting ID token as a claim. The app can then verify this value to mitigate token replay attacks. Usually, the value is a randomized, unique string that can be used to identify the origin of the request.
prompt	No	The type of user interaction that's required. Currently, the only valid value is <code>login</code> . This parameter forces the user to enter their credentials on that request. Single sign-on doesn't take effect.

At this point, the user is asked to complete the policy's workflow. The user might have to enter their username and password, sign in with a social identity, sign up for the directory, or any other number of steps. User actions depend on how the user flow is defined.

After the user completes the user flow, Azure AD returns a response to your app at the value you used for `redirect_uri`. It uses the method specified in the `response_mode` parameter. The response is exactly the same for each of the user action scenarios, independent of the user flow that was executed.

Successful response

A successful response that uses `response_mode=fragment` and `response_type=id_token+token` looks like the following, with line breaks for legibility:

```
GET https://aadb2cplayground.azurewebsites.net/#  
access_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...  
&token_type=Bearer  
&expires_in=3599  
&scope="90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6 offline_access",  
&id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...  
&state=arbitrary_data_you_sent_earlier
```

PARAMETER	DESCRIPTION
access_token	The access token that the app requested.
token_type	The token type value. The only type that Azure AD supports is Bearer.
expires_in	The length of time that the access token is valid (in seconds).
scope	The scopes that the token is valid for. You also can use scopes to cache tokens for later use.
id_token	The ID token that the app requested. You can use the ID token to verify the user's identity and begin a session with the user. For more information about ID tokens and their contents, see the Azure AD B2C token reference .

PARAMETER	DESCRIPTION
state	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The app should verify that the <code>state</code> values in the request and response are identical.

Error response

Error responses also can be sent to the redirect URI so that the app can handle them appropriately:

```
GET https://aadb2cplayground.azurewebsites.net/#  
error=access_denied  
&error_description=the+user+canceled+the+authentication  
&state=arbitrary_data_you_can_receive_in_the_response
```

PARAMETER	DESCRIPTION
error	A code used to classify types of errors that occur.
error_description	A specific error message that can help you identify the root cause of an authentication error.
state	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The app should verify that the <code>state</code> values in the request and response are identical.

Validate the ID token

Receiving an ID token is not enough to authenticate the user. Validate the ID token's signature, and verify the claims in the token per your app's requirements. Azure AD B2C uses [JSON Web Tokens \(JWTs\)](#) and public key cryptography to sign tokens and verify that they are valid.

Many open-source libraries are available for validating JWTs, depending on the language you prefer to use. Consider exploring available open-source libraries rather than implementing your own validation logic. You can use the information in this article to help you learn how to properly use those libraries.

Azure AD B2C has an OpenID Connect metadata endpoint. An app can use the endpoint to fetch information about Azure AD B2C at runtime. This information includes endpoints, token contents, and token signing keys. There is a JSON metadata document for each user flow in your Azure AD B2C tenant. For example, the metadata document for the b2c_1_sign_in user flow in the fabrikamb2c.onmicrosoft.com tenant is located at:

```
https://fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_sign_in/v2.0/.well-known/openid-configuration
```

One of the properties of this configuration document is the `jwks_uri`. The value for the same user flow would be:

```
https://fabrikamb2c.b2clogin.com/fabrikamb2c.onmicrosoft.com/b2c_1_sign_in/discovery/v2.0/keys
```

To determine which user flow was used to sign an ID token (and where to fetch the metadata from), you have two options. First, the user flow name is included in the `acr` claim in `id_token`. For information about how to parse the claims from an ID token, see the [Azure AD B2C token reference](#). Your other option is to encode the user flow in the value of the `state` parameter when you issue the request. Then, decode the `state` parameter to determine

which user flow was used. Either method is valid.

After you've acquired the metadata document from the OpenID Connect metadata endpoint, you can use the RSA-256 public keys (located at this endpoint) to validate the signature of the ID token. There might be multiple keys listed at this endpoint at any given time, each identified by a `kid`. The header of `id_token` also contains a `kid` claim. It indicates which of these keys was used to sign the ID token. For more information, including learning about [validating tokens](#), see the [Azure AD B2C token reference](#).

After you validate the signature of the ID token, several claims require verification. For example:

- Validate the `nonce` claim to prevent token replay attacks. Its value should be what you specified in the sign-in request.
- Validate the `aud` claim to ensure that the ID token was issued for your app. Its value should be the application ID of your app.
- Validate the `iat` and `exp` claims to ensure that the ID token has not expired.

Several more validations that you should perform are described in detail in the [OpenID Connect Core Spec](#). You might also want to validate additional claims, depending on your scenario. Some common validations include:

- Ensuring that the user or organization has signed up for the app.
- Ensuring that the user has proper authorization and privileges.
- Ensuring that a certain strength of authentication has occurred, such as by using Azure Multi-Factor Authentication.

For more information about the claims in an ID token, see the [Azure AD B2C token reference](#).

After you have validated the ID token, you can begin a session with the user. In your app, use the claims in the ID token to obtain information about the user. This information can be used for display, records, authorization, and so on.

Get access tokens

If the only thing your web apps needs to do is execute user flows, you can skip the next few sections. The information in the following sections is applicable only to web apps that need to make authenticated calls to a web API, and which are protected by Azure AD B2C.

Now that you've signed the user into your single-page app, you can get access tokens for calling web APIs that are secured by Azure AD. Even if you have already received a token by using the `token` response type, you can use this method to acquire tokens for additional resources without redirecting the user to sign in again.

In a typical web app flow, you would make a request to the `/token` endpoint. However, the endpoint does not support CORS requests, so making AJAX calls to get a refresh token is not an option. Instead, you can use the implicit flow in a hidden HTML iframe element to get new tokens for other web APIs. Here's an example, with line breaks for legibility:

```
https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/{{policy}}/oauth2/v2.0/authorize?  
client_id=90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6  
&response_type=token  
&redirect_uri=https%3A%2F%2Faadb2cplayground.azurewebsites.net%2F  
&scope=https%3A%2F%2Fapi.contoso.com%2Ftasks.read  
&response_mode=fragment  
&state=arbitrary_data_you_can_receive_in_the_response  
&nonce=12345  
&prompt=none
```

PARAMETER	REQUIRED?	DESCRIPTION
{tenant}	Required	Name of your Azure AD B2C tenant
{policy}	Required	The user flow to be run. Specify the name of a user flow you've created in your Azure AD B2C tenant. For example: <code>b2c_1_sign_in</code> , <code>b2c_1_sign_up</code> , or <code>b2c_1_edit_profile</code> .
client_id	Required	The application ID assigned to your app in the Azure portal .
response_type	Required	Must include <code>id_token</code> for OpenID Connect sign-in. It might also include the response type <code>token</code> . If you use <code>token</code> here, your app can immediately receive an access token from the authorize endpoint, without making a second request to the authorize endpoint. If you use the <code>token</code> response type, the <code>scope</code> parameter must contain a scope that indicates which resource to issue the token for.
redirect_uri	Recommended	The redirect URI of your app, where authentication responses can be sent and received by your app. It must exactly match one of the redirect URIs you registered in the portal, except that it must be URL-encoded.
scope	Required	A space-separated list of scopes. For getting tokens, include all scopes that you require for the intended resource.
response_mode	Recommended	Specifies the method that is used to send the resulting token back to your app. For implicit flow, use <code>fragment</code> . Two other modes can be specified, <code>query</code> and <code>form_post</code> , but do not work in the implicit flow.
state	Recommended	A value included in the request that is returned in the token response. It can be a string of any content that you want to use. Usually, a randomly generated, unique value is used, to prevent cross-site request forgery attacks. The state also is used to encode information about the user's state in the app before the authentication request occurred. For example, the page or view the user was on.

PARAMETER	REQUIRED?	DESCRIPTION
nonce	Required	A value included in the request, generated by the app, that is included in the resulting ID token as a claim. The app can then verify this value to mitigate token replay attacks. Usually, the value is a randomized, unique string that identifies the origin of the request.
prompt	Required	To refresh and get tokens in a hidden iframe, use <code>prompt=none</code> to ensure that the iframe does not get stuck on the sign-in page, and returns immediately.
login_hint	Required	To refresh and get tokens in a hidden iframe, include the username of the user in this hint to distinguish between multiple sessions the user might have at a given time. You can extract the username from an earlier sign-in by using the <code>preferred_username</code> claim (the <code>profile</code> scope is required in order to receive the <code>preferred_username</code> claim).
domain_hint	Required	Can be <code>consumers</code> or <code>organizations</code> . For refreshing and getting tokens in a hidden iframe, include the <code>domain_hint</code> value in the request. Extract the <code>tid</code> claim from the ID token of an earlier sign-in to determine which value to use (the <code>profile</code> scope is required in order to receive the <code>tid</code> claim). If the <code>tid</code> claim value is <code>9188040d-6c67-4c5b-b112-36a304b66dad</code> , use <code>domain_hint=consumers</code> . Otherwise, use <code>domain_hint=organizations</code> .

By setting the `prompt=none` parameter, this request either succeeds or fails immediately, and returns to your application. A successful response is sent to your app at the indicated redirect URI, by using the method specified in the `response_mode` parameter.

Successful response

A successful response by using `response_mode=fragment` looks like this example:

```
GET https://aad2cplayground.azurewebsites.net/#access_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZn10aEV1Q...&state=arbitrary_data_you_sent_earlier&token_type=Bearer&expires_in=3599&scope=https%3A%2F%2Fapi.contoso.com%2Ftasks.read
```

PARAMETER	DESCRIPTION
access_token	The token that the app requested.
token_type	The token type will always be Bearer.
state	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The app should verify that the <code>state</code> values in the request and response are identical.
expires_in	How long the access token is valid (in seconds).
scope	The scopes that the access token is valid for.

Error response

Error responses also can be sent to the redirect URI so that the app can handle them appropriately. For `prompt=none`, an expected error looks like this example:

```
GET https://aad2cplayground.azurewebsites.net/#  
error=user_authentication_required  
&error_description=the+request+could+not+be+completed+silently
```

PARAMETER	DESCRIPTION
error	An error code string that can be used to classify types of errors that occur. You also can use the string to react to errors.
error_description	A specific error message that can help you identify the root cause of an authentication error.

If you receive this error in the iframe request, the user must interactively sign in again to retrieve a new token.

Refresh tokens

ID tokens and access tokens both expire after a short period of time. Your app must be prepared to refresh these tokens periodically. To refresh either type of token, perform the same hidden iframe request we used in an earlier example, by using the `prompt=none` parameter to control Azure AD steps. To receive a new `id_token` value, be sure to use `response_type=id_token` and `scope=openid`, and a `nonce` parameter.

Send a sign-out request

When you want to sign the user out of the app, redirect the user to Azure AD to sign out. If you don't redirect the user, they might be able to reauthenticate to your app without entering their credentials again because they have a valid single sign-on session with Azure AD.

You can simply redirect the user to the `end_session_endpoint` that is listed in the same OpenID Connect metadata document described in [Validate the ID token](#). For example:

```
GET https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/{{policy}}/oauth2/v2.0/logout?  
post_logout_redirect_uri=https%3A%2F%2Faadb2cplayground.azurewebsites.net%2F
```

PARAMETER	REQUIRED	DESCRIPTION
{tenant}	Yes	Name of your Azure AD B2C tenant
{policy}	Yes	The user flow that you want to use to sign the user out of your application.
post_logout_redirect_uri	No	The URL that the user should be redirected to after successful sign out. If it isn't included, Azure AD B2C shows the user a generic message.
state	No	If a <code>state</code> parameter is included in the request, the same value should appear in the response. The application should verify that the <code>state</code> values in the request and response are identical.

NOTE

Directing the user to the `end_session_endpoint` clears some of the user's single sign-on state with Azure AD B2C. However, it doesn't sign the user out of the user's social identity provider session. If the user selects the same identity provider during a subsequent sign-in, the user is re-authenticated, without entering their credentials. If a user wants to sign out of your Azure AD B2C application, it does not necessarily mean they want to completely sign out of their Facebook account, for example. However, for local accounts, the user's session will be ended properly.

Next steps

Code sample: hello.js with Azure AD B2C

[Single-page application built on hello.js with Azure AD B2C \(GitHub\)](#)

This sample on GitHub is intended to help get you started with Azure AD B2C in a simple web application built on `hello.js` and using pop-up-style authentication.

Overview of tokens in Azure Active Directory B2C

1/28/2020 • 12 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) emits several types of security tokens as it processes each [authentication flow](#). This document describes the format, security characteristics, and contents of each type of token.

Token types

Azure AD B2C supports the [OAuth 2.0 and OpenID Connect protocols](#), which makes use of tokens for authentication and secure access to resources. All tokens used in Azure AD B2C are [JSON web tokens \(JWTs\)](#) that contain assertions of information about the bearer and the subject of the token.

The following tokens are used in communication with Azure AD B2C:

- *ID token* - A JWT that contains claims that you can use to identify users in your application. This token is securely sent in HTTP requests for communication between two components of the same application or service. You can use the claims in an ID token as you see fit. They are commonly used to display account information or to make access control decisions in an application. ID tokens are signed, but they are not encrypted. When your application or API receives an ID token, it must validate the signature to prove that the token is authentic. Your application or API must also validate a few claims in the token to prove that it's valid. Depending on the scenario requirements, the claims validated by an application can vary, but your application must perform some common claim validations in every scenario.
- *Access token* - A JWT that contains claims that you can use to identify the granted permissions to your APIs. Access tokens are signed, but they aren't encrypted. Access tokens are used to provide access to APIs and resource servers. When your API receives an access token, it must validate the signature to prove that the token is authentic. Your API must also validate a few claims in the token to prove that it is valid. Depending on the scenario requirements, the claims validated by an application can vary, but your application must perform some common claim validations in every scenario.
- *Refresh token* - Refresh tokens are used to acquire new ID tokens and access tokens in an OAuth 2.0 flow. They provide your application with long-term access to resources on behalf of users without requiring interaction with those users. Refresh tokens are opaque to your application. They are issued by Azure AD B2C and can be inspected and interpreted only by Azure AD B2C. They are long-lived, but your application shouldn't be written with the expectation that a refresh token will last for a specific period of time. Refresh tokens can be invalidated at any moment for a variety of reasons. The only way for your application to know if a refresh token is valid is to attempt to redeem it by making a token request to Azure AD B2C. When you redeem a refresh token for a new token, you receive a new refresh token in the token response. Save the new refresh token. It replaces the refresh token that you previously used in the request. This action helps guarantee that your refresh tokens remain valid for as long as possible.

Endpoints

A [registered application](#) receives tokens and communicates with Azure AD B2C by sending requests to these endpoints:

- <https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/oauth2/v2.0/authorize>
- <https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/oauth2/v2.0/token>

Security tokens that your application receives from Azure AD B2C can come from the `/authorize` or `/token` endpoints. When ID tokens are acquired from the `/authorize` endpoint, it's done using the [implicit flow](#), which is often used for users signing in to JavaScript-based web applications. When ID tokens are acquired from the `/token` endpoint, it's done using the [authorization code flow](#), which keeps the token hidden from the browser.

Claims

When you use Azure AD B2C, you have fine-grained control over the content of your tokens. You can configure [user flows](#) and [custom policies](#) to send certain sets of user data in claims that are required for your application. These claims can include standard properties such as **displayName** and **emailAddress**. Your applications can use these claims to securely authenticate users and requests.

The claims in ID tokens are not returned in any particular order. New claims can be introduced in ID tokens at any time. Your application shouldn't break as new claims are introduced. You can also include [custom user attributes](#) in your claims.

The following table lists the claims that you can expect in ID tokens and access tokens issued by Azure AD B2C.

NAME	CLAIM	EXAMPLE VALUE	DESCRIPTION
Audience	<code>aud</code>	<code>90c0fe63-bcf2-44d5-8fb7-b8bbc0b29dc6</code>	Identifies the intended recipient of the token. For Azure AD B2C, the audience is the application ID. Your application should validate this value and reject the token if it doesn't match. Audience is synonymous with resource.
Issuer	<code>iss</code>	<code>https://{{tenant}}.b2clogin.com/{{tenant}}.onmicrosoft.com/v2.0/.well-known/openid-configuration</code>	Identifies the security token service (STS) that constructs and returns the token. It also identifies the directory in which the user was authenticated. Your application should validate the issuer claim to make sure that the token came from the appropriate endpoint.
Issued at	<code>iat</code>	<code>1438535543</code>	The time at which the token was issued, represented in epoch time.
Expiration time	<code>exp</code>	<code>1438539443</code>	The time at which the token becomes invalid, represented in epoch time. Your application should use this claim to verify the validity of the token lifetime.

NAME	CLAIM	EXAMPLE VALUE	DESCRIPTION
Not before	nbf	1438535543	The time at which the token becomes valid, represented in epoch time. This time is usually the same as the time the token was issued. Your application should use this claim to verify the validity of the token lifetime.
Version	ver	1.0	The version of the ID token, as defined by Azure AD B2C.
Code hash	c_hash	SGCPtt01wxwfgnYZy2VJtQ	A code hash included in an ID token only when the token is issued together with an OAuth 2.0 authorization code. A code hash can be used to validate the authenticity of an authorization code. For more information about how to perform this validation, see the OpenID Connect specification .
Access token hash	at_hash	SGCPtt01wxwfgnYZy2VJtQ	An access token hash included in an ID token only when the token is issued together with an OAuth 2.0 access token. An access token hash can be used to validate the authenticity of an access token. For more information about how to perform this validation, see the OpenID Connect specification
Nonce	nonce	12345	A nonce is a strategy used to mitigate token replay attacks. Your application can specify a nonce in an authorization request by using the <code>nonce</code> query parameter. The value you provide in the request is emitted unmodified in the <code>nonce</code> claim of an ID token only. This claim allows your application to verify the value against the value specified on the request. Your application should perform this validation during the ID token validation process.

NAME	CLAIM	EXAMPLE VALUE	DESCRIPTION
Subject	sub	884408e1-2918-4cz0-b12d-3aa027d7563b	The principal about which the token asserts information, such as the user of an application. This value is immutable and cannot be reassigned or reused. It can be used to perform authorization checks safely, such as when the token is used to access a resource. By default, the subject claim is populated with the object ID of the user in the directory.
Authentication context class reference	acr	Not applicable	Used only with older policies.
Trust framework policy	tfp	b2c_1_signupsignin1	The name of the policy that was used to acquire the ID token.
Authentication time	auth_time	1438535543	The time at which a user last entered credentials, represented in epoch time. There is no discrimination between that authentication being a fresh sign-in, a single sign-on (SSO) session, or another sign-in type. The auth_time is the last time the application (or user) initiated an authentication attempt against Azure AD B2C. The method used to authenticate is not differentiated.
Scope	scp	Read	The permissions granted to the resource for an access token. Multiple granted permissions are separated by a space.
Authorized Party	azp	975251ed-e4f5-4efd-abcb-5f1a8f566ab7	The application ID of the client application that initiated the request.

Configuration

The following properties are used to [manage lifetimes of security tokens](#) emitted by Azure AD B2C:

- **Access & ID token lifetimes (minutes)** - The lifetime of the OAuth 2.0 bearer token used to gain access to a protected resource. The default is 60 minutes. The minimum (inclusive) is 5 minutes. The maximum (inclusive) is 1440 minutes.
- **Refresh token lifetime (days)** - The maximum time period before which a refresh token can be used

to acquire a new access or ID token. The time period also covers acquiring a new refresh token if your application has been granted the `offline_access` scope. The default is 14 days. The minimum (inclusive) is one day. The maximum (inclusive) is 90 days.

- **Refresh token sliding window lifetime (days)** - After this time period elapses the user is forced to reauthenticate, irrespective of the validity period of the most recent refresh token acquired by the application. It can only be provided if the switch is set to **Bounded**. It needs to be greater than or equal to the **Refresh token lifetime (days)** value. If the switch is set to **Unbounded**, you cannot provide a specific value. The default is 90 days. The minimum (inclusive) is one day. The maximum (inclusive) is 365 days.

The following use cases are enabled using these properties:

- Allow a user to stay signed in to a mobile application indefinitely, as long as the user is continually active on the application. You can set **Refresh token sliding window lifetime (days)** to **Unbounded** in your sign-in user flow.
- Meet your industry's security and compliance requirements by setting the appropriate access token lifetimes.

These settings are not available for password reset user flows.

Compatibility

The following properties are used to [manage token compatibility](#):

- **Issuer (iss) claim** - This property identifies the Azure AD B2C tenant that issued the token. The default value is `https://<domain>/<B2C tenant GUID>/v2.0/`. The value of `https://<domain>/tfp/<B2C tenant GUID>/<Policy ID>/v2.0/` includes IDs for both the Azure AD B2C tenant and the user flow that was used in the token request. If your application or library needs Azure AD B2C to be compliant with the [OpenID Connect Discovery 1.0 spec](#), use this value.
- **Subject (sub) claim** - This property identifies the entity for which the token asserts information. The default value is **ObjectID**, which populates the `sub` claim in the token with the object ID of the user. The value of **Not supported** is only provided for backward-compatibility. It's recommended that you switch to **ObjectID** as soon as you are able to.
- **Claim representing policy ID** - This property identifies the claim type into which the policy name used in the token request is populated. The default value is `tfp`. The value of `acr` is only provided for backward-compatibility.

Pass-through

When a user journey starts, Azure AD B2C receives an access token from an identity provider. Azure AD B2C uses that token to retrieve information about the user. You [enable a claim in your user flow](#) or [define a claim in your custom policy](#) to pass the token through to the applications that you register in Azure AD B2C. Your application must be using a [v2 user flow](#) to take advantage of passing the token as a claim.

Azure AD B2C currently only supports passing the access token of OAuth 2.0 identity providers, which include Facebook and Google. For all other identity providers, the claim is returned blank.

Validation

To validate a token, your application should check both the signature and claims of the token. Many open-source libraries are available for validating JWTs, depending on your preferred language. It's recommended that you explore those options rather than implement your own validation logic.

Validate signature

A JWT contains three segments, a *header*, a *body*, and a *signature*. The signature segment can be used to validate the authenticity of the token so that it can be trusted by your application. Azure AD B2C tokens are signed by using industry-standard asymmetric encryption algorithms, such as RSA 256.

The header of the token contains information about the key and encryption method used to sign the token:

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "GvnPApfWMdLRi8PDmisFn7bprKg"  
}
```

The value of the **alg** claim is the algorithm that was used to sign the token. The value of the **kid** claim is the public key that was used to sign the token. At any given time, Azure AD B2C can sign a token by using any one of a set of public-private key pairs. Azure AD B2C rotates the possible set of keys periodically. Your application should be written to handle those key changes automatically. A reasonable frequency to check for updates to the public keys used by Azure AD B2C is every 24 hours.

Azure AD B2C has an OpenID Connect metadata endpoint. Using this endpoint, applications can request information about Azure AD B2C at runtime. This information includes endpoints, token contents, and token signing keys. Your Azure AD B2C tenant contains a JSON metadata document for each policy. The metadata document is a JSON object that contains several useful pieces of information. The metadata contains **jwks_uri**, which gives the location of the set of public keys that are used to sign tokens. That location is provided here, but it's best to fetch the location dynamically by using the metadata document and parsing **jwks_uri**:

```
https://contoso.b2clogin.com/contoso.onmicrosoft.com/discovery/v2.0/keys?p=b2c_1_signupsignin1
```

The JSON document located at this URL contains all the public key information in use at a particular moment. Your app can use the **kid** claim in the JWT header to select the public key in the JSON document that is used to sign a particular token. It can then perform signature validation by using the correct public key and the indicated algorithm.

The metadata document for the **B2C_1_signupsignin1** policy in the **contoso.onmicrosoft.com** tenant is located at:

```
https://contoso.b2clogin.com/contoso.onmicrosoft.com/v2.0/.well-known/openid-configuration?  
p=b2c_1_signupsignin1
```

To determine which policy was used to sign a token (and where to go to request the metadata), you have two options. First, the policy name is included in the **acr** claim in the token. You can parse claims out of the body of the JWT by base-64 decoding the body and deserializing the JSON string that results. The **acr** claim is the name of the policy that was used to issue the token. The other option is to encode the policy in the value of the **state** parameter when you issue the request, and then decode it to determine which policy was used. Either method is valid.

A description of how to perform signature validation is outside the scope of this document. Many open-source libraries are available to help you validate a token.

Validate claims

When your applications or API receives an ID token, it should also perform several checks against the claims in the ID token. The following claims should be checked:

- **audience** - Verifies that the ID token was intended to be given to your application.
- **not before** and **expiration time** - Verifies that the ID token hasn't expired.
- **issuer** - Verifies that the token was issued to your application by Azure AD B2C.
- **nonce** - A strategy for token replay attack mitigation.

For a full list of validations your application should perform, refer to the [OpenID Connect specification](#).

Next steps

Learn more about how to [use access tokens](#).

Request an access token in Azure Active Directory B2C

2/2/2020 • 3 minutes to read • [Edit Online](#)

An *access token* contains claims that you can use in Azure Active Directory B2C (Azure AD B2C) to identify the granted permissions to your APIs. When calling a resource server, an access token must be present in the HTTP request. An access token is denoted as **access_token** in the responses from Azure AD B2C.

This article shows you how to request an access token for a web application and web API. For more information about tokens in Azure AD B2C, see the [overview of tokens in Azure Active Directory B2C](#).

NOTE

Web API chains (On-Behalf-Of) is not supported by Azure AD B2C. - Many architectures include a web API that needs to call another downstream web API, both secured by Azure AD B2C. This scenario is common in clients that have a web API back end, which in turn calls another service. This chained web API scenario can be supported by using the OAuth 2.0 JWT Bearer Credential grant, otherwise known as the On-Behalf-Of flow. However, the On-Behalf-Of flow is not currently implemented in Azure AD B2C.

Prerequisites

- [Create a user flow](#) to enable users to sign up and sign in to your application.
- If you haven't already done so, [add a web API application to your Azure Active Directory B2C tenant](#).

Scopes

Scopes provide a way to manage permissions to protected resources. When an access token is requested, the client application needs to specify the desired permissions in the **scope** parameter of the request. For example, to specify the **Scope Value** of `read` for the API that has the **App ID URI** of `https://contoso.onmicrosoft.com/api`, the scope would be `https://contoso.onmicrosoft.com/api/read`.

Scopes are used by the web API to implement scope-based access control. For example, users of the web API could have both read and write access, or users of the web API might have only read access. To acquire multiple permissions in the same request, you can add multiple entries in the single **scope** parameter of the request, separated by spaces.

The following example shows scopes decoded in a URL:

```
scope=https://contoso.onmicrosoft.com/api/read openid offline_access
```

The following example shows scopes encoded in a URL:

```
scope=https%3A%2F%2Fcontoso.onmicrosoft.com%2Fapi%2Fread%20openid%20offline_access
```

If you request more scopes than what is granted for your client application, the call succeeds if at least one permission is granted. The **scp** claim in the resulting access token is populated with only the permissions that were successfully granted. The OpenID Connect standard specifies several special scope values. The following scopes represent the permission to access the user's profile:

- **openid** - Requests an ID token.
- **offline_access** - Requests a refresh token using [Auth Code flows](#).

If the **response_type** parameter in an `/authorize` request includes `token`, the **scope** parameter must include at least one resource scope other than `openid` and `offline_access` that will be granted. Otherwise, the `/authorize` request fails.

Request a token

To request an access token, you need an authorization code. Below is an example of a request to the `/authorize` endpoint for an authorization code. Custom domains are not supported for use with access tokens. Use your `tenant-name.onmicrosoft.com` domain in the request URL.

In the following example, you replace these values:

- `<tenant-name>` - The name of your Azure AD B2C tenant.
- `<policy-name>` - The name of your custom policy or user flow.
- `<application-ID>` - The application identifier of the web application that you registered to support the user flow.
- `<redirect-uri>` - The **Redirect URI** that you entered when you registered the client application.

```
GET https://<tenant-name>.b2clogin.com/tfp/<tenant-name>.onmicrosoft.com/<policy-name>/oauth2/v2.0/authorize?
client_id=<application-ID>
&nonce=anyRandomValue
&redirect_uri=https://jwt.ms
&scope=https://<tenant-name>.onmicrosoft.com/api/read
&response_type=code
```

The response with the authorization code should be similar to this example:

```
https://jwt.ms/?code=eyJraWQiOiJjcGltY29yZV8wOTI1MjAxNSIsInZlciI6IjEuMC...
```

After successfully receiving the authorization code, you can use it to request an access token:

```
POST <tenant-name>.onmicrosoft.com/oauth2/v2.0/token?p=<policy-name> HTTP/1.1
Host: <tenant-name>.b2clogin.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&client_id=<application-ID>
&scope=https://<tenant-name>.onmicrosoft.com/api/read
&code=eyJraWQiOiJjcGltY29yZV8wOTI1MjAxNSIsInZlciI6IjEuMC...
&redirect_uri=https://jwt.ms
&client_secret=2hMG2-_y12n10vwH...
```

You should see something similar to the following response:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilg1ZVhrN...",
  "token_type": "Bearer",
  "not_before": 1549647431,
  "expires_in": 3600,
  "expires_on": 1549651031,
  "resource": "f2a76e08-93f2-4350-833c-965c02483b11",
  "profile_info": "eyJ2ZXIiOiIxLjAiLCJ0aWQiOiJjNjRhNGY3ZC0zMDkxLTRjNzMtYTcyMi1hM2YwNjk0Z..."
}
```

When using <https://jwt.ms> to examine the access token that was returned, you should see something similar to the following example:

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "X5eXk4xyojNFum1kl2Ytv8dl..."  
}.{  
  "iss": "https://contoso0926tenant.b2clogin.com/c64a4f7d-3091-4c73-a7.../v2.0/",  
  "exp": 1549651031,  
  "nbf": 1549647431,  
  "aud": "f2a76e08-93f2-4350-833c-965...",  
  "oid": "1558f87f-452b-4757-bcd1-883...",  
  "sub": "1558f87f-452b-4757-bcd1-883...",  
  "name": "David",  
  "tfp": "B2C_1_signupsignin1",  
  "nonce": "anyRandomValue",  
  "scp": "read",  
  "azp": "38307aee-303c-4fff-8087-d8d2...",  
  "ver": "1.0",  
  "iat": 1549647431  
}.[Signature]
```

Next steps

- Learn about how to [configure tokens in Azure AD B2C](#)

User flows in Azure Active Directory B2C

2/20/2020 • 3 minutes to read • [Edit Online](#)

The extensible policy framework of Azure Active Directory B2C (Azure AD B2C) is the core strength of the service. Policies fully describe identity experiences such as sign-up, sign-in, or profile editing. To help you set up the most common identity tasks, the Azure AD B2C portal includes predefined, configurable policies called **user flows**.

What are user flows?

A user flow enables you to control behaviors in your applications by configuring the following settings:

- Account types used for sign-in, such as social accounts like a Facebook or local accounts
- Attributes to be collected from the consumer, such as first name, postal code, and shoe size
- Azure Multi-Factor Authentication
- Customization of the user interface
- Information that the application receives as claims in a token

You can create many user flows of different types in your tenant and use them in your applications as needed. User flows can be reused across applications. This flexibility enables you to define and modify identity experiences with minimal or no changes to your code. Your application triggers a user flow by using a standard HTTP authentication request that includes a user flow parameter. A customized [token](#) is received as a response.

The following examples show the "p" query string parameter that specifies the user flow to be used:

```
https://contosob2c.b2clogin.com/contosob2c.onmicrosoft.com/oauth2/v2.0/authorize?  
client_id=2d4d11a2-f814-46a7-890a-274a72a7309e      // Your registered Application ID  
&redirect_uri=https%3A%2F%2Flocalhost%3A44321%2F      // Your registered Reply URL, url encoded  
&response_mode=form_post                            // 'query', 'form_post' or 'fragment'  
&response_type=id_token  
&scope=openid  
&nonce=dummy  
&state=12345                                         // Any value provided by your application  
&p=b2c_1_siup                                         // Your sign-up user flow
```

```
https://contosob2c.b2clogin.com/contosob2c.onmicrosoft.com/oauth2/v2.0/authorize?  
client_id=2d4d11a2-f814-46a7-890a-274a72a7309e      // Your registered Application ID  
&redirect_uri=https%3A%2F%2Flocalhost%3A44321%2F      // Your registered Reply URL, url encoded  
&response_mode=form_post                            // 'query', 'form_post' or 'fragment'  
&response_type=id_token  
&scope=openid  
&nonce=dummy  
&state=12345                                         // Any value provided by your application  
&p=b2c_1_siin                                         // Your sign-in user flow
```

User flow versions

In the Azure portal, new [versions of user flows](#) are being added all the time. When you get started with Azure AD B2C, tested user flows are recommended for you to use. When you create a new user flow, you choose the user flow that you need from the **Recommended** tab.

The following user flows are currently recommended:

- **Sign up and sign in** - Handles both of the sign-up and sign-in experiences with a single configuration. Users are led down the right path depending on the context. It's recommended that you use this user flow over a **sign-up** user flow or a **sign-in** user flow.
- **Profile editing** - Enables users to edit their profile information.
- **Password reset** - Enables you to configure whether and how users can reset their password.

Linking user flows

A **sign-up or sign-in** user flow with local accounts includes a **Forgot password?** link on the first page of the experience. Clicking this link doesn't automatically trigger a password reset user flow.

Instead, the error code `AADB2C90118` is returned to your application. Your application needs to handle this error code by running a specific user flow that resets the password. To see an example, take a look at a [simple ASP.NET sample](#) that demonstrates the linking of user flows.

Email address storage

An email address can be required as part of a user flow. If the user authenticates with a social identity provider, the email address is stored in the **otherMails** property. If a local account is based on a user name, then the email address is stored in a strong authentication detail property. If a local account is based on an email address, then the email address is stored in the **signInNames** property.

The email address isn't guaranteed to be verified in any of these cases. A tenant administrator can disable email verification in the basic policies for local accounts. Even if email address verification is enabled, addresses aren't verified if they come from a social identity provider and they haven't been changed.

Only the **otherMails** and **signInNames** properties are exposed through the Microsoft Graph API. The email address in the strong authentication detail property is not available.

Next steps

To create the recommended user flows, follow the instructions in [Tutorial: Create a user flow](#).

Custom policies in Azure Active Directory B2C

1/28/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Custom policies are configuration files that define the behavior of your Azure Active Directory B2C (Azure AD B2C) tenant. User flows are predefined in the Azure AD B2C portal for the most common identity tasks. Custom policies can be fully edited by an identity developer to complete many different tasks.

Comparing user flows and custom policies

	USER FLOWS	CUSTOM POLICIES
Target users	All application developers with or without identity expertise.	Identity pros, systems integrators, consultants, and in-house identity teams. They are comfortable with OpenID Connect flows and understand identity providers and claims-based authentication.
Configuration method	Azure portal with a user-friendly user-interface (UI).	Directly editing XML files and then uploading to the Azure portal.
UI customization	Full UI customization including HTML, CSS and JavaScript. Multilanguage support with Custom strings.	Same
Attribute customization	Standard and custom attributes.	Same
Token and session management	Custom token and multiple session options.	Same
Identity Providers	Predefined local or social provider and most OIDC identity providers, such as federation with Azure Active Directory tenants.	Standards-based OIDC, OAUTH, and SAML. Authentication is also possible by using integration with REST APIs.

	USER FLOWS	CUSTOM POLICIES
Identity Tasks	<p>Sign-up or sign-in with local or many social accounts.</p> <p>Self-service password reset.</p> <p>Profile edit.</p> <p>Multi-Factor Authentication.</p> <p>Customize tokens and sessions.</p> <p>Access token flows.</p>	<p>Complete the same tasks as user flows using custom identity providers or use custom scopes.</p> <p>Provision a user account in another system at the time of registration.</p> <p>Send a welcome email using your own email service provider.</p> <p>Use a user store outside Azure AD B2C.</p> <p>Validate user provided information with a trusted system by using an API.</p>

Policy files

These three types of policy files are used:

- **Base file** - contains most of the definitions. It is recommended that you make a minimum number of changes to this file to help with troubleshooting, and long-term maintenance of your policies.
- **Extensions file** - holds the unique configuration changes for your tenant.
- **Relying Party (RP) file** - The single task-focused file that is invoked directly by the application or service (also, known as a Relying Party). Each unique task requires its own RP and depending on branding requirements, the number might be "total of applications x total number of use cases."

User flows in Azure AD B2C follow the three-file pattern depicted above, but the developer only sees the RP file, while the Azure portal makes changes in the background to the extensions file.

Custom policy core concepts

The customer identity and access management (CIAM) service in Azure includes:

- A user directory that is accessible by using Microsoft Graph and which holds user data for both local accounts and federated accounts.
- Access to the **Identity Experience Framework** that orchestrates trust between users and entities and passes claims between them to complete an identity or access management task.
- A security token service (STS) that issues ID tokens, refresh tokens, and access tokens (and equivalent SAML assertions) and validates them to protect resources.

Azure AD B2C interacts with identity providers, users, other systems, and with the local user directory in sequence to achieve an identity task. For example, sign in a user, register a new user, or reset a password. The Identity Experience Framework and a policy (also called a user journey or a trust framework policy) establishes multi-party trust and explicitly defines the actors, the actions, the protocols, and the sequence of steps to complete.

The Identity Experience Framework is a fully configurable, policy-driven, cloud-based Azure platform that orchestrates trust between entities in standard protocol formats such as OpenID Connect, OAuth, SAML, and a few non-standard ones, for example REST API-based system-to-system claims exchanges. The framework creates user-friendly, white-labeled experiences that support HTML and CSS.

A custom policy is represented as one or several XML-formatted files that refer to each other in a hierarchical chain. The XML elements define the claims schema, claims transformations, content definitions, claims

providers, technical profiles, and user journey orchestration steps, among other elements. A custom policy is accessible as one or several XML files that are executed by the Identity Experience Framework when invoked by a relying party. Developers configuring custom policies must define the trusted relationships in careful detail to include metadata endpoints, exact claims exchange definitions, and configure secrets, keys, and certificates as needed by each identity provider.

Inheritance model

When an application calls the RP policy file, the Identity Experience Framework in Azure AD B2C adds all of the elements from base file, from the extensions file, and then from the RP policy file to assemble the current policy in effect. Elements of the same type and name in the RP file will override those in the extensions, and extensions overrides base.

Next steps

[Get started with custom policies](#)

Overview of user accounts in Azure Active Directory B2C

2/20/2020 • 4 minutes to read • [Edit Online](#)

In Azure Active Directory B2C (Azure AD B2C), there are several types of accounts that can be created. Azure Active Directory, Active Directory B2B, and Active Directory B2C share in the types of user accounts that can be used.

The following types of accounts are available:

- **Work account** - A work account can access resources in a tenant, and with an administrator role, can manage tenants.
- **Guest account** - A guest account can only be a Microsoft account or an Azure Active Directory user that can be used to access applications or manage tenants.
- **Consumer account** - A consumer account is used by a user of the applications you've registered with Azure AD B2C. Consumer accounts can be created by:
 - The user going through a sign-up user flow in an Azure AD B2C application
 - Using Microsoft Graph API
 - Using the Azure portal

Work account

A work account is created the same way for all tenants based on Azure AD. To create a work account, you can use the information in [Quickstart: Add new users to Azure Active Directory](#). A work account is created using the **New user** choice in the Azure portal.

When you add a new work account, you need to consider the following configuration settings:

- **Name and User name** - The **Name** property contains the given and surname of the user. The **User name** is the identifier that the user enters to sign in. The user name includes the full domain. The domain name portion of the user name must either be the initial default domain name *your-domain.onmicrosoft.com*, or a verified, non-federated [custom domain](#) name such as *contoso.com*.
- **Profile** - The account is set up with a profile of user data. You have the opportunity to enter a first name, last name, job title, and department name. You can edit the profile after the account is created.
- **Groups** - Use a group to perform management tasks such as assigning licenses or permissions to a number of users or devices at once. You can put the new account into an existing [group](#) in your tenant.
- **Directory role** - You need to specify the level of access that the user account has to resources in your tenant. The following permission levels are available:
 - **User** - Users can access assigned resources but cannot manage most tenant resources.
 - **Global administrator** - Global administrators have full control over all tenant resources.
 - **Limited administrator** - Select the administrative role or roles for the user. For more information about the roles that can be selected, see [Assigning administrator roles in Azure Active Directory](#).

Create a work account

You can use the following information to create a new work account:

- [Azure portal](#)

- Microsoft Graph

Update a user profile

You can use the following information to update the profile of a user:

- Azure portal
- Microsoft Graph

Reset a password for a user

You can use the following information to reset the password of a user:

- Azure portal
- Microsoft Graph

Guest user

You can invite external users to your tenant as a guest user. A typical scenario for inviting a guest user to your Azure AD B2C tenant is to share administration responsibilities. For an example of using a guest account, see [Properties of an Azure Active Directory B2B collaboration user](#).

When you invite a guest user to your tenant, you provide the email address of the recipient and a message describing the invitation. The invitation link takes the user to the consent page where the **Get Started** button is selected and the review of permissions is accepted. If an inbox isn't attached to the email address, the user can navigate to the consent page by going to a Microsoft page using the invited credentials. The user is then forced to redeem the invitation the same way as clicking on the link in the email. For example:

`https://myapps.microsoft.com/B2CTENANTNAME .`

You can also use the [Microsoft Graph API](#) to invite a guest user.

Consumer user

The consumer user can sign in to applications secured by Azure AD B2C, but cannot access Azure resources such as the Azure portal. The consumer user can use a local account or federated accounts, such as Facebook or Twitter. A consumer account is created by using a [sign-up or sign-in user flow](#), using the Microsoft Graph API, or by using the Azure portal.

You can specify the data that is collected when a consumer user account is created by using custom user attributes. For more information, see [Define custom attributes in Azure Active Directory B2C](#).

For more information about managing consumer accounts, see [Manage Azure AD B2C user accounts with Microsoft Graph](#).

Migrate consumer user accounts

You might have a need to migrate existing consumer user accounts from any identity provider to Azure AD B2C. For more information, see [Migrate users to Azure AD B2C](#).

Tutorial: Register an application in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

Before your [applications](#) can interact with Azure Active Directory B2C (Azure AD B2C), they must be registered in a tenant that you manage. This tutorial shows you how to register a web application using the Azure portal.

In this article, you learn how to:

- Register a web application
- Create a client secret

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

If you haven't already created your own [Azure AD B2C Tenant](#), create one now. You can use an existing Azure AD B2C tenant.

Register a web application

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *webapp1*.
6. For **Include web app/ web API** and **Allow implicit flow**, select **Yes**.
7. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. For example, you could set it to listen locally at `https://localhost:44316`. If you don't yet know the port number, you can enter a placeholder value and change it later.

For testing purposes like this tutorial you can set it to `https://jwt.ms` which displays the contents of a token for inspection. For this tutorial, set the **Reply URL** to `https://jwt.ms`.

The following restrictions apply to reply URLs:

- The reply URL must begin with the scheme `https`.
- The reply URL is case-sensitive. Its case must match the case of the URL path of your running application. For example, if your application includes as part of its path `.../abc/response-oidc`, do not specify `.../ABC/response-oidc` in the reply URL. Because the web browser treats paths as case-

sensitive, cookies associated with `.../abc/response-oidc` may be excluded if redirected to the case-mismatched `.../ABC/response-oidc` URL.

8. Select **Create** to complete the application registration.

Create a client secret

If your application exchanges an authorization code for an access token, you need to create an application secret.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. In the **Azure AD B2C - Applications** page, select the application you created, for example *webapp1*.
2. Select **Keys** and then select **Generate key**.
3. Select **Save** to view the key. Make note of the **App key** value. You use this value as the application secret in your application's code.

Next steps

In this article, you learned how to:

- Register a web application
- Create a client secret

Next, learn how to create user flows to enable your users to sign up, sign in, and manage their profiles.

[Create user flows in Azure Active Directory B2C >](#)

Register a SAML application in Azure AD B2C

2/28/2020 • 11 minutes to read • [Edit Online](#)

In this article, you learn how to configure Azure Active Directory B2C (Azure AD B2C) to act as a Security Assertion Markup Language (SAML) identity provider (IdP) to your applications.

NOTE

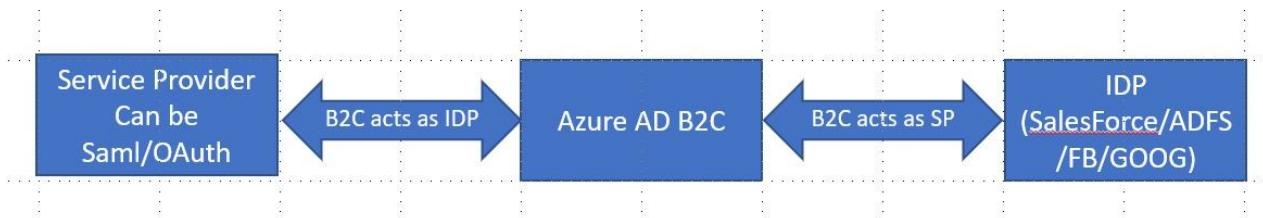
This feature is in public preview.

Scenario overview

Organizations that use Azure AD B2C as their customer identity and access management solution might require interaction with identity providers or applications that are configured to authenticate using the SAML protocol.

Azure AD B2C achieves SAML interoperability in one of two ways:

- By acting as an *identity provider* (IdP) and achieving single-sign-on (SSO) with SAML-based service providers (your applications)
- By acting as a *service provider* (SP) and interacting with SAML-based identity providers like Salesforce and ADFS



Summarizing the two non-exclusive core scenarios with SAML:

SCENARIO	AZURE AD B2C ROLE	HOW-TO
My application expects a SAML assertion to complete an authentication.	Azure AD B2C acts as the identity provider (IdP) Azure AD B2C acts as a SAML IdP to the applications.	This article.
My users need single-sign-on with a SAML-compliant identity provider like ADFS, Salesforce, or Shibboleth.	Azure AD B2C acts as the service provider (SP) Azure AD B2C acts as a service provider when connecting to the SAML identity provider. It's a federation proxy between your application and the SAML identity provider.	<ul style="list-style-type: none">• Set up sign-in with ADFS as a SAML IdP using custom policies• Set up sign-in with a Salesforce SAML provider using custom policies

Prerequisites

- Complete the steps in [Get started with custom policies in Azure AD B2C](#). You need the *SocialAndLocalAccounts* custom policy from the custom policy starter pack discussed in the article.
- Basic understanding of the Security Assertion Markup Language (SAML) protocol.
- A web application configured as a SAML service provider (SP). For this tutorial, you can use a [SAML test](#)

application that we provide.

Components of the solution

There are three main components required for this scenario:

- SAML **service provider** with the ability to send SAML requests, and receive, decode, and respond to SAML assertions from Azure AD B2C. This is also known as the relying party.
- Publicly available SAML **metadata endpoint** for your service provider.
- Azure AD B2C tenant

If you don't yet have a SAML service provider and an associated metadata endpoint, you can use this sample SAML application that we've made available for testing:

[SAML Test Application](#)

1. Set up certificates

To build a trust relationship between your service provider and Azure AD B2C, you need to provide X509 certificates and their private keys.

- **Service provider certificates**

- Certificate with a private key stored in your Web App. This certificate is used by your service provider to sign the SAML request sent to Azure AD B2C. Azure AD B2C reads the public key from the service provider metadata to validate the signature.
- (Optional) Certificate with a private key stored in your Web App. Azure AD B2C reads the public key from the service provider metadata to encrypt the SAML assertion. The service provider then uses the private key to decrypt the assertion.

- **Azure AD B2C certificates**

- Certificate with a private key in Azure AD B2C. This certificate is used by Azure AD B2C to sign the SAML response sent to your service provider. Your service provider reads the Azure AD B2C metadata public key to validate the signature of the SAML response.

You can use a certificate issued by a public certificate authority or, for this tutorial, a self-signed certificate.

1.1 Prepare a self-signed certificate

If you don't already have a certificate, you can use a self-signed certificate for this tutorial. On Windows, you can use PowerShell's [New-SelfSignedCertificate](#) cmdlet to generate a certificate.

1. Execute this PowerShell command to generate a self-signed certificate. Modify the `-Subject` argument as appropriate for your application and Azure AD B2C tenant name. You can also adjust the `-NotAfter` date to specify a different expiration for the certificate.

```
New-SelfSignedCertificate ` 
-KeyExportPolicy Exportable ` 
-Subject "CN=yourappname.yourtenant.onmicrosoft.com" ` 
-KeyAlgorithm RSA ` 
-KeyLength 2048 ` 
-KeyUsage DigitalSignature ` 
-NotAfter (Get-Date).AddMonths(12) ` 
-CertStoreLocation "Cert:\CurrentUser\My"
```

2. Open **Manage user certificates > Current User > Personal > Certificates >**

yourappname.yourtenant.onmicrosoft.com

3. Select the certificate > **Action > All Tasks > Export**

4. Select **Yes** > **Next** > **Yes, export the private key** > **Next**

5. Accept the defaults for **Export File Format**

6. Provide a password for the certificate

1.2 Upload the certificate

Next, upload the SAML assertion and response signing certificate to Azure AD B2C.

1. Sign in to the [Azure portal](#) and browse to your Azure AD B2C tenant.

2. Under **Policies**, select **Identity Experience Framework** and then **Policy keys**.

3. Select **Add**, and then select **Options > Upload**.

4. Enter a **Name**, for example *SamlIdpCert*. The prefix *B2C_1A_* is automatically added to the name of your key.

5. Upload your certificate using the upload file control.

6. Enter the certificate's password.

7. Select **Create**.

8. Verify that the key appears as expected. For example, *B2C_1A_SamlIdpCert*.

2. Prepare your policy

2.1 Create the SAML token issuer

Now, add the capability for your tenant to issue SAML tokens.

Open `SocialAndLocalAccounts\ TrustFrameworkExtensions.xml` in the custom policy starter pack.

Locate the `<ClaimsProviders>` section and add the following XML snippet.

You can change the value of the `IssuerUri` metadata. This is the issuer URI that is returned in the SAML response from Azure AD B2C. Your relying party application should be configured to accept an issuer URI during SAML assertion validation.

```

<ClaimsProvider>
  <DisplayName>Token Issuer</DisplayName>
  <TechnicalProfiles>

    <!-- SAML Token Issuer technical profile -->
    <TechnicalProfile Id="Saml2AssertionIssuer">
      <DisplayName>Token Issuer</DisplayName>
      <Protocol Name="None"/>
      <OutputTokenFormat>SAML2</OutputTokenFormat>
      <Metadata>
        <!-- The issuer contains the policy name; it should be the same name as configured in the relying party application. B2C_1A_signup_signin_SAML is used below. -->
        <!--<Item Key="IssuerUri">https://tenant-name.b2clogin.com/tenant-name.onmicrosoft.com/B2C_1A_signup_signin_SAML</Item>-->
      </Metadata>
      <CryptographicKeys>
        <Key Id="MetadataSigning" StorageReferenceId="B2C_1A_SamlIdpCert"/>
        <Key Id="SamlAssertionSigning" StorageReferenceId="B2C_1A_SamlIdpCert"/>
        <Key Id="SamlMessageSigning" StorageReferenceId="B2C_1A_SamlIdpCert"/>
      </CryptographicKeys>
      <InputClaims/>
      <OutputClaims/>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Saml-sp"/>
    </TechnicalProfile>

    <!-- Session management technical profile for SAML based tokens -->
    <TechnicalProfile Id="SM-Saml-sp">
      <DisplayName>Session Management Provider</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.SamlSSOServiceProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

3. Add the SAML relying party policy

Now that your tenant can issue SAML assertions, you need to create the SAML relying party policy, and modify the user journey so that it issues a SAML assertion instead of a JWT.

3.1 Create sign-up or sign-in policy

1. Create a copy of the *SignUpOrSignin.xml* file in your starter pack working directory and save it with a new name. For example, *SignUpOrSigninSAML.xml*. This is your relying party policy file.
2. Open the *SignUpOrSigninSAML.xml* file in your preferred editor.
3. Change the `PolicyId` and `PublicPolicyUri` of the policy to `B2C_1A_signup_signin_saml` and `http://tenant-name.onmicrosoft.com/B2C_1A_signup_signin_saml` as seen below.

```

<TrustFrameworkPolicy
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
  PolicySchemaVersion="0.3.0.0"
  TenantId="tenant-name.onmicrosoft.com"
  PolicyId="B2C_1A_signup_signin_saml"
  PublicPolicyUri="http://tenant-name.onmicrosoft.com/B2C_1A_signup_signin_saml">

```

4. Add following XML snippet just before the `<RelyingParty>` element. This XML overwrites orchestration step number 7 of the *SignUpOrSignIn* user journey. If you started from a different folder in the starter pack, or customized your user journey by adding or removing orchestration steps, make sure the number (in the

`order` element) is aligned with the one specified in the user journey for the token issuer step (for example, in the other starter pack folders it's step number 4 for `LocalAccounts`, 6 for `SocialAccounts` and 9 for `SocialAndLocalAccountsWithMfa`).

```
<UserJourneys>
  <UserJourney Id="SignUpOrSignIn">
    <OrchestrationSteps>
      <OrchestrationStep Order="7" Type="SendClaims"
CpimIssuerTechnicalProfileReferenceId="Saml2AssertionIssuer"/>
    </OrchestrationSteps>
  </UserJourney>
</UserJourneys>
```

5. Replace the entire `<TechnicalProfile>` element in the `<RelyingParty>` element with the following technical profile XML.

```
<TechnicalProfile Id="PolicyProfile">
  <DisplayName>PolicyProfile</DisplayName>
  <Protocol Name="SAML2"/>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="surname" />
    <OutputClaim ClaimTypeReferenceId="email" DefaultValue="" />
    <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="" />
    <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="objectId"/>
  </OutputClaims>
  <SubjectNamingInfo ClaimType="objectId" ExcludeAsClaim="true"/>
</TechnicalProfile>
```

6. Update `tenant-name` with the name of your Azure AD B2C tenant.

Your final relying party policy file should look like the following:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TrustFrameworkPolicy
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
    PolicySchemaVersion="0.3.0.0"
    TenantId="contoso.onmicrosoft.com"
    PolicyId="B2C_1A_signup_signin_saml"
    PublicPolicyUri="http://contoso.onmicrosoft.com/B2C_1A_signup_signin_saml">

    <BasePolicy>
        <TenantId>contoso.onmicrosoft.com</TenantId>
        <PolicyId>B2C_1A_TrustFrameworkExtensions</PolicyId>
    </BasePolicy>

    <UserJourneys>
        <UserJourney Id="SignUpOrSignIn">
            <OrchestrationSteps>
                <OrchestrationStep Order="7" Type="SendClaims"
CpimIssuerTechnicalProfileReferenceId="Saml2AssertionIssuer"/>
            </OrchestrationSteps>
        </UserJourney>
    </UserJourneys>

    <RelyingParty>
        <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
        <TechnicalProfile Id="PolicyProfile">
            <DisplayName>PolicyProfile</DisplayName>
            <Protocol Name="SAML2"/>
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="displayName" />
                <OutputClaim ClaimTypeReferenceId="givenName" />
                <OutputClaim ClaimTypeReferenceId="surname" />
                <OutputClaim ClaimTypeReferenceId="email" DefaultValue="" />
                <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="" />
                <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="objectId"/>
            </OutputClaims>
            <SubjectNamingInfo ClaimType="objectId" ExcludeAsClaim="true"/>
        </TechnicalProfile>
    </RelyingParty>
</TrustFrameworkPolicy>

```

3.2 Upload and test your policy metadata

Save your changes and upload the new policy file. After you've uploaded both policies (the extension and the relying party files), open a web browser and navigate to the policy metadata.

Azure AD B2C policy IDP metadata is information used in the SAML protocol to expose the configuration of a SAML identity provider. Metadata defines the location of the services, such as sign-in and sign-out, certificates, sign-in method, and more. The Azure AD B2C policy metadata is available at the following URL. Replace `tenant-name` with the name of your Azure AD B2C tenant, and `policy-name` with the name (ID) of the policy:

`https://tenant-name.b2clogin.com/tenant-name.onmicrosoft.com/policy-name/Samlp/metadata`

Your custom policy and Azure AD B2C tenant are now ready. Next, create an application registration in Azure AD B2C.

4. Setup application in the Azure AD B2C Directory

4.1 Register your application in Azure Active Directory

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your

Azure AD B2C tenant.

3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **App registrations (Preview)**, and then select **New registration**.
5. Enter a **Name** for the application. For example, *SAMLApp1*.
6. Under **Supported account types**, select **Accounts in this organizational directory only**.
7. Under **Redirect URI**, select **Web**, and then enter `https://localhost`. You modify this value later in the application registration's manifest.
8. Select **Register**.

4.2 Update the app manifest

For SAML apps, there are several properties you need to configure in the application registration's manifest.

1. In the [Azure portal](#), navigate to the application registration that you created in the previous section.
2. Under **Manage**, select **Manifest** to open the manifest editor. You modify several properties in the following sections.

identifierUris

The `identifierUris` is a string collection containing user-defined URI(s) that uniquely identify a Web app within its Azure AD B2C tenant. Your service provider must set this value in the `Issuer` element of a SAML request.

samlMetadataUrl

This property represents service provider's publicly available metadata URL. The metadata URL can point to a metadata file uploaded to any anonymously accessible endpoint, for example blob storage.

The metadata is information used in the SAML protocol to expose the configuration of a SAML party, such as a service provider. Metadata defines the location of the services like sign-in and sign-out, certificates, sign-in method, and more. Azure AD B2C reads the service provider metadata and acts accordingly. The metadata is not required. You can also specify some of attributes like the reply URI and logout URI directly in the app manifest.

If there are properties specified in *both* the SAML metadata URL and in the application registration's manifest, they are **merged**. The properties specified in the metadata URL are processed first and take precedence.

For this tutorial which uses the SAML test application, use the following value for `samlMetadataUrl`:

```
"samlMetadataUrl": "https://samltestapp2.azurewebsites.net/Metadata",
```

replyUrlsWithType (Optional)

If you do not provide a metadata URI, you can explicitly specify the reply URL. This optional property represents the `AssertionConsumerServiceUrl` (`SingleSignOnService` URL in the service provider metadata) and the `BindingType` is assumed to be `HTTP POST`.

If you choose to configure the reply URL and logout URL in the application manifest without using the service provider metadata, Azure AD B2C will not validate the SAML request signature, nor encrypt the SAML response.

For this tutorial, in which you use the SAML test application, set the `url` property of `replyUrlsWithType` to the value shown in the following JSON snippet.

```
"replyUrlsWithType": [  
  {  
    "url": "https://samltestapp2.azurewebsites.net/SP/AssertionConsumer",  
    "type": "Web"  
  }  
,
```

logoutUrl (Optional)

This optional property represents the `Logout` URL (`SingleLogoutService` URL in the relying party metadata), and the `BindingType` for this is assumed to be `Http-Redirect`.

For this tutorial which uses the SAML test application, leave `logoutUrl` set to
`https://samltestapp2.azurewebsites.net/logout`:

```
"logoutUrl": "https://samltestapp2.azurewebsites.net/logout",
```

5. Update your application code

The last step is to enable Azure AD B2C as a SAML IdP in your SAML relying party application. Each application is different and the steps to do so vary. Consult your app's documentation for details.

Some or all the following are typically required:

- **Metadata:** `https://tenant-name.b2clogin.com/tenant-name.onmicrosoft.com/policy-name/Samlp/metadata`
- **Issuer:** `https://tenant-name.b2clogin.com/tenant-name.onmicrosoft.com/policy-name`
- **Login Url/SAML endpoint/SAML Url:** Check the value in the metadata file
- **Certificate:** This is `B2C_1A_SamlIdpCert`, but without the private key. To get the public key of the certificate:
 1. Go to the metadata URL specified above.
 2. Copy the value in the `<x509Certificate>` element.
 3. Paste it into a text file.
 4. Save the text file as a `.cer` file.

5.1 Test with the SAML Test App (optional)

To complete this tutorial using our [SAML Test Application](#):

- Update the tenant name
- Update policy name, for example `B2C_1A_signup_signin_saml`
- Specify this issuer URL: `https://contoso.onmicrosoft.com/app-name`

Select **Login** and you should be presented with an end user sign-in screen. Upon sign-in, a SAML assertion is issued back to the sample application.

Sample policy

We provide a complete sample policy that you can use for testing with the SAML Test App.

1. Download the [SAML-SP-initiated login sample policy](#)
2. Update `TenantId` to match your tenant name, for example `contoso.b2clogin.com`
3. Keep the policy name of `B2C_1A_SAML2_signup_signin`

Supported and unsupported SAML modalities

The following SAML relying party (RP) scenarios are supported via your own metadata endpoint:

- Multiple logout URLs or POST binding for logout URL in application/service principal object.
- Specify signing key to verify RP requests in application/service principal object.
- Specify token encryption key in application/service principal object.
- Identity provider-initiated logins are not currently supported in the preview release.

Next steps

You can find more information about the [SAML protocol](#) on the [OASIS website](#).

Manage Azure AD B2C with Microsoft Graph

2/20/2020 • 5 minutes to read • [Edit Online](#)

[Microsoft Graph](#) allows you to manage many of the resources within your Azure AD B2C tenant, including customer user accounts and custom policies. By writing scripts or applications that call the [Microsoft Graph API](#), you can automate tenant management tasks like:

- Migrate an existing user store to an Azure AD B2C tenant
- Deploy custom policies with an Azure Pipeline in Azure DevOps, and manage custom policy keys
- Host user registration on your own page, and create user accounts in your Azure AD B2C directory behind the scenes
- Automate application registration
- Obtain audit logs

The following sections help you prepare for using the Microsoft Graph API to automate the management of resources in your Azure AD B2C directory.

Microsoft Graph API interaction modes

There are two modes of communication you can use when working with the Microsoft Graph API to manage resources in your Azure AD B2C tenant:

- **Interactive** - Appropriate for run-once tasks, you use an administrator account in the B2C tenant to perform the management tasks. This mode requires an administrator to sign in using their credentials before calling the Microsoft Graph API.
- **Automated** - For scheduled or continuously run tasks, this method uses a service account that you configure with the permissions required to perform management tasks. You create the "service account" in Azure AD B2C by registering an application that your applications and scripts use for authenticating using its *Application (Client) ID* and the OAuth 2.0 client credentials grant. In this case, the application acts as itself to call the Microsoft Graph API, not the administrator user as in the previously described interactive method.

You enable the **Automated** interaction scenario by creating an application registration shown in the following sections.

Register management application

Before your scripts and applications can interact with the [Microsoft Graph API](#) to manage Azure AD B2C resources, you need to create an application registration in your Azure AD B2C tenant that grants the required API permissions.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.

3. In the Azure portal, search for and select **Azure Active Directory**.
4. Under **Manage**, select **App registrations (Legacy)**.
5. Select **New application registration**.
6. Enter a name for the application. For example, *managementapp1*.
7. For **Application type**, select **Web app / API**.
8. Enter any valid URL in **Sign-on URL**. For example, `https://localhost`. The endpoint doesn't need to be reachable, but must be a valid URL.
9. Select **Create**.
10. Record the **Application ID** that appears on the **Registered app** overview page. You use this value in a later step.

Grant API access

Next, grant the registered application permissions to manipulate tenant resources through calls to the Microsoft Graph API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. On the **Registered app** overview page, select **Settings**.
2. Under **API Access**, select **Required permissions**.
3. Select **Microsoft Graph**.
4. Under **Application Permissions**, select the check box of the permission to grant to your management application. For example:
 - **Read all audit log data**: Select this permission for reading the directory's audit logs.
 - **Read and write directory data**: Select this permission for user migration or user management scenarios.
 - **Read and write your organization's trust framework policies**: Select this permission for continuous integration/continuous delivery (CI/CD) scenarios. For example, custom policy deployment with Azure Pipelines.
5. Select **Save**.
6. Select **Grant permissions**, and then select **Yes**. It might take a few minutes to for the permissions to fully propagate.

Create client secret

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Under **API ACCESS**, select **Keys**.
2. Enter a description for the key in the **Key description** box. For example, *clientsecret1*.
3. Select a validity **Duration** and then select **Save**.
4. Record the key's **VALUE**. You use this value for configuration in a later step.

You now have an application that has permission to *create, read, update, and delete* users in your Azure AD B2C tenant. Continue to the next section to add *password update* permissions.

Enable user delete and password update

The *Read and write directory data* permission does **NOT** include the ability delete users or update user account passwords.

If your application or script needs to delete users or update their passwords, assign the *User administrator* role to your application:

1. Sign in to the [Azure portal](#) and use the **Directory + Subscription** filter to switch to your Azure AD B2C tenant.
2. Search for and select **Azure AD B2C**.
3. Under **Manage**, select **Roles and administrators**.
4. Select the **User administrator** role.
5. Select **Add assignments**.
6. In the **Select** text box, enter the name of the application you registered earlier, for example, *managementapp1*.
Select your application when it appears in the search results.
7. Select **Add**. It might take a few minutes for the permissions to fully propagate.

Next steps

Now that you've registered your management application and have granted it the required permissions, your applications and services (for example, Azure Pipelines) can use its credentials and permissions to interact with the Microsoft Graph API.

- [B2C operations supported by Microsoft Graph](#)
- [Manage Azure AD B2C user accounts with Microsoft Graph](#)
- [Get audit logs with the Azure AD reporting API](#)

Add a web API application to your Azure Active Directory B2C tenant

11/5/2019 • 5 minutes to read • [Edit Online](#)

Register web API resources in your tenant so that they can accept and respond to requests by client applications that present an access token. This article shows you how to register a web API in Azure Active Directory B2C (Azure AD B2C).

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *webapi1*.
6. For **Include web app/ web API** and **Allow implicit flow**, select **Yes**.
7. For **Reply URL**, enter an endpoint where Azure AD B2C should return any tokens that your application requests. In your production application, you might set the reply URL to a value such as `https://localhost:44332`. For testing purposes, set the reply URL to `https://jwt.ms`.
8. For **App ID URI**, enter the identifier used for your web API. The full identifier URI including the domain is generated for you. For example, `https://contosotenant.onmicrosoft.com/api`.
9. Click **Create**.
10. On the properties page, record the application ID that you'll use when you configure the web application.

Configure scopes

Scopes provide a way to govern access to protected resources. Scopes are used by the web API to implement scope-based access control. For example, users of the web API could have both read and write access, or users of the web API might have only read access. In this tutorial, you use scopes to define read and write permissions for the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**.
2. Select the *webapi1* application to open its **Properties** page.
3. Select **Published scopes**. Published scopes can be used to grant a client application certain permissions to the web API.
4. For **SCOPE**, enter `demo.read`, and for **DESCRIPTION**, enter `Read access to the web API`.
5. For **SCOPE**, enter `demo.write`, and for **DESCRIPTION**, enter `Write access to the web API`.
6. Select **Save**.

Grant permissions

To call a protected web API from an application, you need to grant your application permissions to the API. For example, in [Tutorial: Register an application in Azure Active Directory B2C](#), a web application named *webapp1* is registered in Azure AD B2C. You can use this application to call the web API.

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Select **Applications**, and then select the web application that should have access to the API. For example, *webapp1*.
2. Select **API access**, and then select **Add**.
3. In the **Select API** dropdown, select the API to which web application should be granted access. For example, *webapi1*.
4. In the **Select Scopes** dropdown, select the scopes that you defined earlier. For example, *demo.read* and *demo.write*.
5. Select **OK**.

Your application is registered to call the protected web API. A user authenticates with Azure AD B2C to use the application. The application obtains an authorization grant from Azure AD B2C to access the protected web API.

Add a native client application to your Azure Active Directory B2C tenant

11/4/2019 • 2 minutes to read • [Edit Online](#)

Native client resources need to be registered in your tenant before your application can communicate with Azure Active Directory B2C.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *nativeapp1*.
6. For **Native client**, select **Yes**.
7. Enter a **Custom Redirect URI** with a unique scheme. For example, `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`. There are two important considerations when choosing a redirect URI:
 - **Unique:** The scheme of the redirect URI must be unique for every application. In the example `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`, `com.onmicrosoft.contosob2c.exampleapp` is the scheme. This pattern should be followed. If two applications share the same scheme, the user is given a choice to choose an application. If the user chooses incorrectly, the sign-in fails.
 - **Complete:** The redirect URI must have both a scheme and a path. The path must contain at least one forward slash after the domain. For example, `//oauth/` works while `//oauth` fails. Don't include special characters in the URI, for example, underscores.
8. Select **Create**.

Azure AD B2C: Sign-in using an iOS application

1/28/2020 • 6 minutes to read • [Edit Online](#)

The Microsoft identity platform uses open standards such as OAuth2 and OpenID Connect. Using an open standard protocol offers more developer choice when selecting a library to integrate with our services. We've provided this walkthrough and others like it to aid developers with writing applications that connect to the Microsoft Identity platform. Most libraries that implement the [RFC6749 OAuth2 spec](#) are able to connect to the Microsoft Identity platform.

WARNING

Microsoft does not provide fixes for third-party libraries and has not done a review of those libraries. This sample is using a third-party library called AppAuth that has been tested for compatibility in basic scenarios with the Azure AD B2C. Issues and feature requests should be directed to the library's open-source project. For more information, see [this article](#).

If you're new to OAuth2 or OpenID Connect, much of this sample configuration may not make much sense to you. We recommend you look at a brief [overview of the protocol we've documented here](#).

Get an Azure AD B2C directory

Before you can use Azure AD B2C, you must create a directory, or tenant. A directory is a container for all your users, apps, groups, and more. If you don't have one already, [create a B2C directory](#) before you continue.

Create an application

Next, register an application in your Azure AD B2C tenant. This gives Azure AD the information it needs to communicate securely with your app.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *nativeapp1*.
6. For **Native client**, select **Yes**.
7. Enter a **Custom Redirect URI** with a unique scheme. For example,

`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`. There are two important considerations when choosing a redirect URI:

- **Unique:** The scheme of the redirect URI must be unique for every application. In the example `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`, `com.onmicrosoft.contosob2c.exampleapp` is the scheme. This pattern should be followed. If two applications share the same scheme, the user is given a choice to choose an application. If the user chooses incorrectly, the sign-in fails.
- **Complete:** The redirect URI must have both a scheme and a path. The path must contain at least one

forward slash after the domain. For example, `//oauth/` works while `//oauth` fails. Don't include special characters in the URI, for example, underscores.

8. Select **Create**.

Record the **Application (client) ID** for use in a later step.

Also record your custom redirect URI for use in a later step. For example,

`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`.

Create your user flows

In Azure AD B2C, every user experience is defined by a [user flow](#). This application contains one identity experience: a combined sign-in and sign-up. When you create the user flow, be sure to:

- Under **Sign-up attributes**, select the attribute **Display name**. You can select other attributes as well.
- Under **Application claims**, select the claims **Display name** and **User's Object ID**. You can select other claims as well.
- Copy the **Name** of each user flow after you create it. Your user flow name is prefixed with `b2c_1_` when you save the user flow. You need the user flow name later.

After you have created your user flows, you're ready to build your app.

Download the sample code

We have provided a working sample that uses AppAuth with Azure AD B2C [on GitHub](#). You can download the code and run it. To use your own Azure AD B2C tenant, follow the instructions in the [README.md](#).

This sample was created by following the README instructions by the [iOS AppAuth project on GitHub](#). For more details on how the sample and the library work, reference the AppAuth README on GitHub.

Modifying your app to use Azure AD B2C with AppAuth

NOTE

AppAuth supports iOS 7 and above. However, to support social logins on Google, SFSafariViewController is needed which requires iOS 9 or higher.

Configuration

You can configure communication with Azure AD B2C by specifying both the authorization endpoint and token endpoint URIs. To generate these URIs, you need the following information:

- Tenant ID (for example, contoso.onmicrosoft.com)
- User flow name (for example, B2C_1_SignUpIn)

The token endpoint URI can be generated by replacing the Tenant_ID and the Policy_Name in the following URL:

```
static NSString *const tokenEndpoint =
@"https://<Tenant_name>.b2clogin.com/te/<Tenant_ID>/<Policy_Name>/oauth2/v2.0/token";
```

The authorization endpoint URI can be generated by replacing the Tenant_ID and the Policy_Name in the following URL:

```
static NSString *const authorizationEndpoint =
@"https://<Tenant_name>.b2clogin.com/te/<Tenant_ID>/<Policy_Name>/oauth2/v2.0/authorize";
```

Run the following code to create your AuthorizationServiceConfiguration object:

```
OIDServiceConfiguration *configuration =
[[OIDServiceConfiguration alloc] initWithAuthorizationEndpoint:authorizationEndpoint
tokenEndpoint:tokenEndpoint];
// now we are ready to perform the auth request...
```

Authorizing

After configuring or retrieving an authorization service configuration, an authorization request can be constructed. To create the request, you need the following information:

- Client ID (APPLICATION ID) that you recorded earlier. For example, `00000000-0000-0000-0000-000000000000`.
- Custom Redirect URI that you recorded earlier. For example,
`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`.

Both items should have been saved when you were [registering your app](#).

```
OIDAuthorizationRequest *request =
[[OIDAuthorizationRequest alloc] initWithConfiguration:configuration
                                         clientId:kClientId
                                         scopes:@[OIDScopeOpenID, OIDScopeProfile]
                                         redirectURL:[NSURL URLWithString:kRedirectUri]
                                         responseType:OIDResponseTypeCode
                                         additionalParameters:nil];

AppDelegate *appDelegate = (AppDelegate *)[UIApplication sharedApplication].delegate;
appDelegate.currentAuthorizationFlow =
[OIDAuthState authStateByPresentingAuthorizationRequest:request
              presentingViewController:self
                           callback:^(OIDAuthState *_Nullable authState, NSError *_Nullable error) {
        if (authState) {
            NSLog(@"Got authorization tokens. Access token: %@", authState.lastTokenResponse.accessToken);
            [self setAuthState:authState];
        } else {
            NSLog(@"Authorization error: %@", [error localizedDescription]);
            [self setAuthState:nil];
        }
    }];
}
```

To set up your application to handle the redirect to the URI with the custom scheme, you need to update the list of 'URL Schemes' in your Info.plist:

- Open Info.plist.
- Hover over a row like 'Bundle OS Type Code' and click the + symbol.
- Rename the new row 'URL types'.
- Click the arrow to the left of 'URL types' to open the tree.
- Click the arrow to the left of 'Item 0' to open the tree.
- Rename first item underneath Item 0 to 'URL Schemes'.
- Click the arrow to the left of 'URL Schemes' to open the tree.
- In the 'Value' column, there is a blank field to the left of 'Item 0' underneath 'URL Schemes'. Set the value to your application's unique scheme. The value must match the scheme used in redirectURL when creating the OIDAuthorizationRequest object. In the sample, the scheme 'com.onmicrosoft.fabrikamb2c.exampleapp' is used.

Refer to the [AppAuth guide](#) on how to complete the rest of the process. If you need to quickly get started with a working app, check out [the sample](#). Follow the steps in the [README.md](#) to enter your own Azure AD B2C configuration.

We are always open to feedback and suggestions! If you have any difficulties with this article, or have recommendations for improving this content, we would appreciate your feedback at the bottom of the page. For feature requests, add them to [UserVoice](#).

Sign-in using an Android application in Azure Active Directory B2C

1/28/2020 • 6 minutes to read • [Edit Online](#)

The Microsoft identity platform uses open standards such as OAuth2 and OpenID Connect. These standards allow you to leverage any library you wish to integrate with Azure Active Directory B2C. To help you use other libraries, you can use a walkthrough like this one to demonstrate how to configure 3rd party libraries to connect to the Microsoft identity platform. Most libraries that implement the [RFC6749 OAuth2 spec](#) can connect to the Microsoft Identity platform.

WARNING

Microsoft does not provide fixes for 3rd party libraries and has not done a review of those libraries. This sample is using a 3rd party library called AppAuth that has been tested for compatibility in basic scenarios with the Azure AD B2C. Issues and feature requests should be directed to the library's open-source project. Please see [this article](#) for more information.

If you're new to OAuth2 or OpenID Connect much of this sample configuration may not make much sense to you. We recommend you look at a brief [overview of the protocol we've documented here](#).

Get an Azure AD B2C directory

Before you can use Azure AD B2C, you must create a directory, or tenant. A directory is a container for all of your users, apps, groups, and more. If you don't have one already, [create a B2C directory](#) before you continue.

Create an application

Next, register an application in your Azure AD B2C tenant. This gives Azure AD the information it needs to communicate securely with your app.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *nativeapp1*.
6. For **Native client**, select **Yes**.
7. Enter a **Custom Redirect URI** with a unique scheme. For example, `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`. There are two important considerations when choosing a redirect URI:
 - **Unique:** The scheme of the redirect URI must be unique for every application. In the example `com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`, `com.onmicrosoft.contosob2c.exampleapp` is the scheme. This pattern should be followed. If two applications share the same scheme, the user is given a

choice to choose an application. If the user chooses incorrectly, the sign-in fails.

- **Complete:** The redirect URI must have both a scheme and a path. The path must contain at least one forward slash after the domain. For example, `//oauth/` works while `//oauth` fails. Don't include special characters in the URI, for example, underscores.

8. Select **Create**.

Record the **Application (client) ID** for use in a later step.

Also record your custom redirect URI for use in a later step. For example,

`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`.

Create your user flows

In Azure AD B2C, every user experience is defined by a **user flow**, which is a set of policies that control the behavior of Azure AD. This application requires a sign-in and sign-up user flow. When you create the user flow, be sure to:

- Choose the **Display name** as a sign-up attribute in your user flow.
- Choose the **Display name** and **Object ID** application claims in every user flow. You can choose other claims as well.
- Copy the **Name** of each user flow after you create it. It should have the prefix `b2c_1_`. You'll need the user flow name later.

After you have created your user flows, you're ready to build your app.

Download the sample code

We have provided a working sample that uses AppAuth with Azure AD B2C [on GitHub](#). You can download the code and run it. You can quickly get started with your own app using your own Azure AD B2C configuration by following the instructions in the [README.md](#).

The sample is a modification of the sample provided by [AppAuth](#). Please visit their page to learn more about AppAuth and its features.

Modifying your app to use Azure AD B2C with AppAuth

NOTE

AppAuth supports Android API 16 (Jellybean) and above. We recommend using API 23 and above.

Configuration

You can configure communication with Azure AD B2C by either specifying the discovery URI or by specifying both the authorization endpoint and token endpoint URIs. In either case, you will need the following information:

- Tenant ID (e.g. contoso.onmicrosoft.com)
- User flow name (e.g. B2C_1_SignUpIn)

If you choose to automatically discover the authorization and token endpoint URIs, you will need to fetch information from the discovery URI. The discovery URI can be generated by replacing the Tenant_ID and the Policy_Name in the following URL:

```
String mDiscoveryURI = "https://<Tenant_name>.b2clogin.com/<Tenant_ID>/v2.0/.well-known/openid-configuration?p=<Policy_Name>";
```

You can then acquire the authorization and token endpoint URIs and create an `AuthorizationServiceConfiguration` object by running the following:

```
final Uri issuerUri = Uri.parse(mDiscoveryURI);
AuthorizationServiceConfiguration config;

AuthorizationServiceConfiguration.fetchFromIssuer(
    issuerUri,
    new RetrieveConfigurationCallback() {
        @Override public void onFetchConfigurationCompleted(
            @Nullable AuthorizationServiceConfiguration serviceConfiguration,
            @Nullable AuthorizationException ex) {
            if (ex != null) {
                Log.w(TAG, "Failed to retrieve configuration for " + issuerUri, ex);
            } else {
                // service configuration retrieved, proceed to authorization...
            }
        }
    });
});
```

Instead of using discovery to obtain the authorization and token endpoint URIs, you can also specify them explicitly by replacing the `Tenant_ID` and the `Policy_Name` in the URL's below:

```
String mAuthEndpoint = "https://<Tenant_name>.b2clogin.com/<Tenant_ID>/oauth2/v2.0/authorize?p=<Policy_Name>";

String mTokenEndpoint = "https://<Tenant_name>.b2clogin.com/<Tenant_ID>/oauth2/v2.0/token?p=<Policy_Name>";
```

Run the following code to create your `AuthorizationServiceConfiguration` object:

```
AuthorizationServiceConfiguration config =
    new AuthorizationServiceConfiguration(name, mAuthEndpoint, mTokenEndpoint);

// perform the auth request...
```

Authorizing

After configuring or retrieving an authorization service configuration, an authorization request can be constructed. To create the request, you will need the following information:

- Client ID (APPLICATION ID) that you recorded earlier. For example, `00000000-0000-0000-0000-000000000000`.
- Custom Redirect URI that you recorded earlier. For example,
`com.onmicrosoft.contosob2c.exampleapp://oauth/redirect`.

Both items should have been saved when you were [registering your app](#).

```
AuthorizationRequest req = new AuthorizationRequest.Builder(
    config,
    clientId,
    ResponseTypeValues.CODE,
    redirectUri)
    .build();
```

Please refer to the [AppAuth guide](#) on how to complete the rest of the process. If you need to quickly get started with a working app, check out [our sample](#). Follow the steps in the [README.md](#) to enter your own Azure AD B2C configuration.

Tutorial: Create user flows in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

In your applications you may have [user flows](#) that enable users to sign up, sign in, or manage their profile. You can create multiple user flows of different types in your Azure Active Directory B2C (Azure AD B2C) tenant and use them in your applications as needed. User flows can be reused across applications.

In this article, you learn how to:

- Create a sign-up and sign-in user flow
- Create a profile editing user flow
- Create a password reset user flow

This tutorial shows you how to create some recommended user flows by using the Azure portal. If you're looking for information about how to set up a resource owner password credentials (ROPC) flow in your application, see [Configure the resource owner password credentials flow in Azure AD B2C](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

[Register your applications](#) that are part of the user flows you want to create.

Create a sign-up and sign-in user flow

The sign-up and sign-in user flow handles both sign-up and sign-in experiences with a single configuration. Users of your application are led down the right path depending on the context.

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.

The screenshot shows the Microsoft Azure portal home page. In the top right corner, there is a 'Directory + subscription' dropdown menu with a red box around it. Below it, the 'Default subscription filter' is set to 'contoso@contoso.com'. The main interface includes sections for 'Azure services' (Create a resource, Virtual machines, Web Apps, Storage accounts, SQL databases, Azure Database for PostgreSQL, Azure Cosmos DB, Kubernetes), 'Navigate' (Subscriptions, Resource groups, All resources), 'Tools' (Microsoft Learn, Azure Monitor, Security Center), and 'Useful links' (Technical Documentation, Azure Services, Recent Azure Updates, Quickstart Center). The 'Switch directory' section shows a list of directories, with 'Contoso' highlighted by a red box.

3. In the Azure portal, search for and select **Azure AD B2C**.
4. Under **Policies**, select **User flows (policies)**, and then select **New user flow**.

The screenshot shows the 'Azure AD B2C - User flows (policies)' page. The left sidebar has sections for Overview, Manage (Applications, Identity providers, User attributes, Users), and Policies (User flows (policies), Identity Experience Framework). The 'User flows (policies)' option is highlighted with a red box. The main area shows a search bar, a '+ New user flow' button (also highlighted with a red box), and a table with columns NAME, TYPE, and MFA. A message indicates 'No user flows found.'

5. On the **Recommended** tab, select the **Sign up and sign in** user flow.

Home > Azure AD B2C - User flows (policies) > Create a user flow

Create a user flow

Select a user flow type

A user flow is a series of pages for your users to interact with to sign up or sign into their account. Choose one to start with and you can create multiple user flows to define your entire authentication experience for your app. [Learn more about user flow types.](#)

Recommended [Preview](#) [All](#)

Recommended for most applications.

Sign up and sign in Lets a user register for or log into their account	Profile editing Lets the user configure their user attributes	Password reset Allows a user to choose a new password after verifying their email
---	--	--

6. Enter a **Name** for the user flow. For example, *signupsignin1*.

7. For **Identity providers**, select **Email signup**.

Home > Azure AD B2C - User flows (policies) > Create a user flow > Create

Create

Sign up and sign in

← Select a different type of user flow

Get started with your user flow with a few basic selections. Don't worry about getting everything right here, you can modify your user flow after you've created it.

1. Name *

The unique string used to identify this user flow in requests to Azure AD B2C. This cannot be changed after a user flow has been created.

* B2C_1_ 

2. Identity providers *

Identity providers are the different types of accounts your users can use to log into your application. You need to select at least one for a valid user flow and you can add more in the Identity providers section for your directory. [Learn more about identity providers.](#)

Please select at least one identity provider

 Email signup

3. Multifactor authentication

Enabling multifactor authentication (MFA) requires your users to verify their identity with a second factor before allowing them into your application. [Learn more about multifactor authentication.](#)

Multifactor authentication [Enabled](#) [Disabled](#)

8. For **User attributes and claims**, choose the claims and attributes that you want to collect and send from the user during sign-up. For example, select **Show more**, and then choose attributes and claims for **Country/Region**, **Display Name**, and **Postal Code**. Click **OK**.

Home > Azure AD B2C - User flows

Create

Create
Sign up and sign in

← Select a different type of user flow
Get started with your user flow with a flow after you've created it.

1. Name *
The unique string used to identify this created.
* B2C_1_signupsignin1

2. Identity providers *
Identity providers are the different types for a valid user flow and you can add
Please select at least one identity provider
 Email signup

3. Multifactor authentication
Enabling multifactor authentication (MFA) to your application. [Learn more about MFA](#)
Multifactor authentication

4. User attributes and claims
User attributes are values collected on sign up. Claims are values about the user returned to the application in the token. You can create custom attributes for use in your directory.

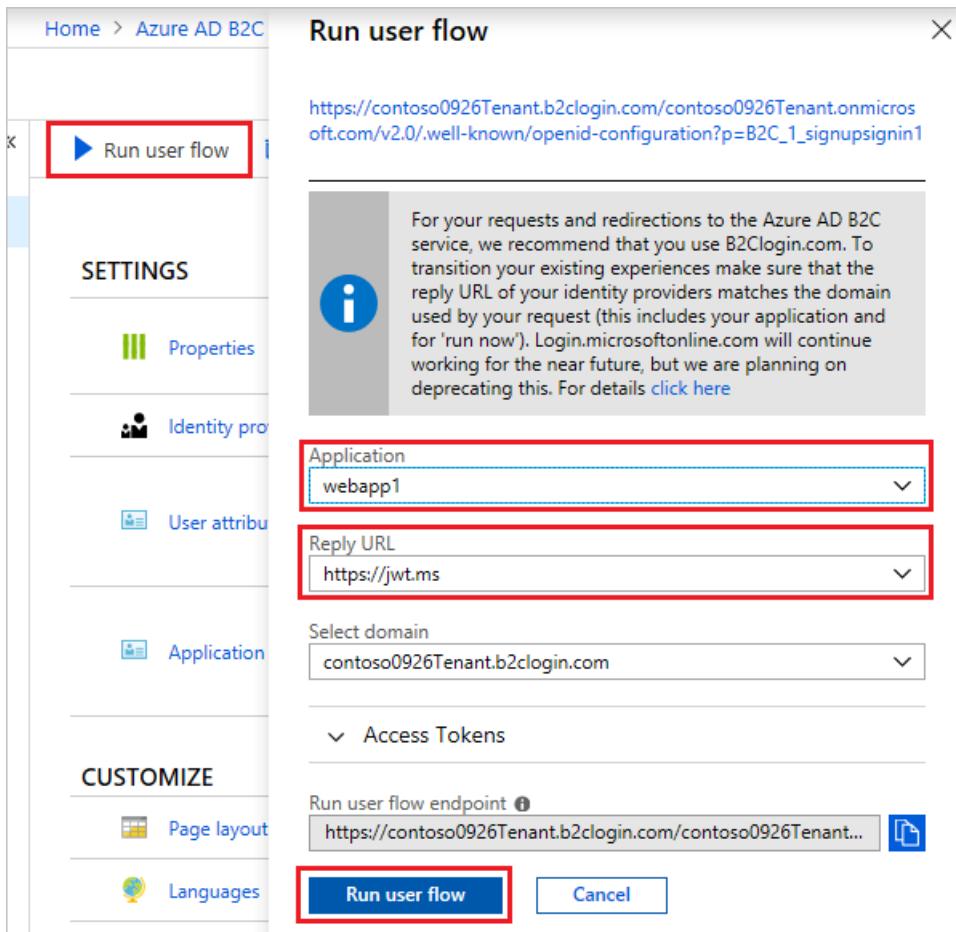
	Collect attribute	Return claim
City ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Country/Region ⓘ	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Display Name ⓘ	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Email Address ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Email Addresses ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Given Name ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Identity Provider ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Identity Provider Access Token ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Job Title ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Postal Code ⓘ	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
State/Province ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Street Address ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
Surname ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
User is new ⓘ	<input type="checkbox"/>	<input type="checkbox"/>
User's Object ID ⓘ	<input type="checkbox"/>	<input type="checkbox"/>

Given Name ⓘ
Surname ⓘ
City ⓘ
Country/Region ⓘ
Email Address ⓘ
[Show more...](#)

9. Click **Create** to add the user flow. A prefix of *B2C_1* is automatically appended to the name.

Test the user flow

- Select the user flow you created to open its overview page, then select **Run user flow**.
- For **Application**, select the web application named *webapp1* that you previously registered. The **Reply URL** should show `https://jwt.ms`.
- Click **Run user flow**, and then select **Sign up now**.



4. Enter a valid email address, click **Send verification code**, enter the verification code that you receive, then select **Verify code**.
5. Enter a new password and confirm the password.
6. Select your country and region, enter the name that you want displayed, enter a postal code, and then click **Create**. The token is returned to `https://jwt.ms` and should be displayed to you.
7. You can now run the user flow again and you should be able to sign in with the account that you created. The returned token includes the claims that you selected of country/region, name, and postal code.

Create a profile editing user flow

If you want to enable users to edit their profile in your application, you use a profile editing user flow.

1. In the menu of the Azure AD B2C tenant overview page, select **User flows (policies)**, and then select **New user flow**.
2. Select the **Profile editing** user flow on the **Recommended** tab.
3. Enter a **Name** for the user flow. For example, *profileediting1*.
4. For **Identity providers**, select **Local Account SignIn**.
5. For **User attributes**, choose the attributes that you want the customer to be able to edit in their profile. For example, select **Show more**, and then choose both attributes and claims for **Display name** and **Job title**. Click **OK**.
6. Click **Create** to add the user flow. A prefix of *B2C_1* is automatically appended to the name.

Test the user flow

1. Select the user flow you created to open its overview page, then select **Run user flow**.
2. For **Application**, select the web application named *webapp1* that you previously registered. The **Reply URL** should show `https://jwt.ms`.

3. Click **Run user flow**, and then sign in with the account that you previously created.
4. You now have the opportunity to change the display name and job title for the user. Click **Continue**. The token is returned to `https://jwt.ms` and should be displayed to you.

Create a password reset user flow

To enable users of your application to reset their password, you use a password reset user flow.

1. In the Azure AD B2C tenant overview menu, select **User flows (policies)**, and then select **New user flow**.
2. Select the **Password reset** user flow on the **Recommended** tab.
3. Enter a **Name** for the user flow. For example, `passwordreset1`.
4. For **Identity providers**, enable **Reset password using email address**.
5. Under Application claims, click **Show more** and choose the claims that you want returned in the authorization tokens sent back to your application. For example, select **User's Object ID**.
6. Click **OK**.
7. Click **Create** to add the user flow. A prefix of `B2C_1` is automatically appended to the name.

Test the user flow

1. Select the user flow you created to open its overview page, then select **Run user flow**.
2. For **Application**, select the web application named `webapp1` that you previously registered. The **Reply URL** should show `https://jwt.ms`.
3. Click **Run user flow**, verify the email address of the account that you previously created, and select **Continue**.
4. You now have the opportunity to change the password for the user. Change the password and select **Continue**. The token is returned to `https://jwt.ms` and should be displayed to you.

Next steps

In this article, you learned how to:

- Create a sign-up and sign-in user flow
- Create a profile editing user flow
- Create a password reset user flow

Next, learn about adding identity providers to your applications to enable user sign-in with providers like Azure AD, Amazon, Facebook, GitHub, LinkedIn, Microsoft, or Twitter.

[Add identity providers to your applications >](#)

Configure the resource owner password credentials flow in Azure AD B2C

2/28/2020 • 5 minutes to read • [Edit Online](#)

The resource owner password credentials (ROPC) flow is an OAuth standard authentication flow where the application, also known as the relying party, exchanges valid credentials such as userid and password for an ID token, access token, and a refresh token.

NOTE

This feature is in public preview.

ROPC flow notes

In Azure Active Directory B2C (Azure AD B2C), the following options are supported:

- **Native Client:** User interaction during authentication happens when code runs on a user-side device. The device can be a mobile application that's running in a native operating system, such as Android and iOS.
- **Public client flow:** Only user credentials, gathered by an application, are sent in the API call. The credentials of the application are not sent.
- **Add new claims:** The ID token contents can be changed to add new claims.

The following flows are not supported:

- **Server-to-server:** The identity protection system needs a reliable IP address gathered from the caller (the native client) as part of the interaction. In a server-side API call, only the server's IP address is used. If a dynamic threshold of failed authentications is exceeded, the identity protection system may identify a repeated IP address as an attacker.
- **Confidential client flow:** The application client ID is validated, but the application secret is not validated.

When using the ROPC flow, consider the following:

- ROPC doesn't work when there is any interruption to the authentication flow that needs user interaction. For example, when a password has expired or needs to be changed, multi-factor authentication is required, or when more information needs to be collected during sign-in (for example, user consent).
- ROPC supports local accounts only. Users can't sign in with federated identity providers like Microsoft, Google+, Twitter, AD-FS, or Facebook.
- Session Management, including keep me signed-in (KMSI), is not applicable.

Create a resource owner user flow

1. Sign in to the Azure portal as the global administrator of your Azure AD B2C tenant.
2. To switch to your Azure AD B2C tenant, select the B2C directory in the upper-right corner of the portal.
3. Click **User flows**, and select **New user flow**.
4. Click the **All** tab and select **Sign in using ROPC**.
5. Provide a name for the user flow, such as *ROPC_Auth*.

6. Under **Application claims**, click **Show more**.
7. Select the application claims that you need for your application, such as Display Name, Email Address, and Identity Provider.
8. Select **OK**, and then select **Create**.
9. Click **Run user flow**.

You'll then see an endpoint such as this example:

```
https://yourtenant.b2clogin.com/yourtenant.onmicrosoft.com/v2.0/.well-known/openid-configuration?
p=B2C_1_ROPC_Auth
```

Register an application

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *ROPC_Auth_app*.
6. For **Native client**, select **Yes**.
7. Leave the other values as they are, and then select **Create**.
8. Record the **APPLICATION ID** for use in a later step.

Test the user flow

Use your favorite API development application to generate an API call, and review the response to debug your user flow. Construct a call like this with the information in the following table as the body of the POST request:

- Replace <*yourtenant.onmicrosoft.com*> with the name of your B2C tenant.
- Replace <*B2C_1A_ROPC_Auth*> with the full name of your resource owner password credentials policy.
- Replace <*bef2222d56-552f-4a5b-b90a-1988a7d634c3*> with the Application ID from your registration.

```
https://yourtenant.b2clogin.com/<yourtenant.onmicrosoft.com>/oauth2/v2.0/token?p=B2C_1_ROPC_Auth
```

KEY	VALUE
username	leadiocl@outlook.com
password	Passxword1
grant_type	password
scope	openid < <i>bef2222d56-552f-4a5b-b90a-1988a7d634c3</i> > offline_access
client_id	< <i>bef2222d56-552f-4a5b-b90a-1988a7d634c3</i> >

KEY	VALUE
response_type	token id_token

Client_id is the value that you previously noted as the application ID. *Offline_access* is optional if you want to receive a refresh token. The username and password that you use must be credentials from an existing user in your Azure AD B2C tenant.

The actual POST request looks like the following:

```
POST /yourtenant.onmicrosoft.com/oauth2/v2.0/token?p=B2C_1_ROPC_Auth HTTP/1.1
Host: yourtenant.b2clogin.com
Content-Type: application/x-www-form-urlencoded

username=leadioic1%40trashmail.ws&password=Passxword1&grant_type=password&scope=openid+bef22d56-552f-4a5b-b90a-1988a7d634ce+offline_access&client_id=bef22d56-552f-4a5b-b90a-1988a7d634ce&response_type=token+id_token
```

A successful response with offline-access looks like the following example:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ik9YQjNhdTNScWhUQWN6R0RWZDM5djNpTmlyTWhqN2wxMjIySnh6TmgwRlkifQ.eyJpc3MiOiJodHRwczovL3R1LmNwaW0ud2luZG93cy5uZXQvZja2YzJmZTgtNzA5Zi00MDMwLTg1ZGMtMzhhNGJmZD1lODJkL3YyLjAvIiwiZXhwIjoxNTExMTMwMDc4LCJuYmYi0jE1MTMxMjY0NzsImF1ZCI6ImJlZjIyZDU2LTU1MmYtNGE1Yi1iOTBhLTE50DhhN2Q2MzRjZSIisIm9pZCI6IjNjM2I5Nj1jLThjNDktNGUxMS1hNGVmLWZkYjJmZkzyZjA0OSiisInN1YiI6Ik5vdCBzdXBwb3J0ZWQgY3VycmVudGx5LiBVc2Ugb2lkIGNsYWltLiIsImF6cCI6ImJlZjIyZDU2LTU1MmYtNGE1Yi1iOTBhLTE50DhhN2Q2MzRjZSIisInZlcii6IjeuMCiisIm1hdCI6MTUxMzeyNjQ3OHO.MSEThYXzCS4SeBw3-3ecnVLUkucFkeh-H-P7SFcJ-MhsBeQEpMF1Rzu_R9kUqv3qEWKAPYCndZ3_P4Dd3a63iG6m9Tn01Vt5SKTEtuhVx3X15LYeA1i3s1t9Y7LIicn59hGKRZ8ddrQzkqj69j723ooY01amrXvF6zNOudh0acsesz7fbzzofyagKPerxaeTH0NgYoInLwXu0eNj_6RtF9gBfgwVidRy90zXUJnqm1GdrS61XUqiIUtv4H04jYxDem7ek6E4jsH809uSXT0id5_4C5bdHrp01N6pXSasmVR9GM1XgfXA_IRLFU4Nd26CzGl1NjbhLnvlizqY4A",
  "token_type": "Bearer",
  "expires_in": "3600",
  "refresh_token": "eyJraWQiOiJacW9pQlp2TW5pYvc2MUY0Tn1fR3REVk1EVFBLoUJLb0FUcWQ1ZWFja1hBIiwidmVyIjoiMS4wIiwiemlwIjoiRGVmbGF0ZSIisInNlcii6IjEuMCJ9.aJ_2UW14dh4saWTQ0jLJ7ByQs5JzIeW_AU90_RVFgrnnYiPhikEc68i1vwWo8B20KTRB_s7oy_Eoh5LACsqU60z0Mjh0-DxgrMb1UOTAQ9dbfAT5WoLZiCJBjIz4YT50UA_RAGjhBUkqGwdWEumDExQnXiJRSeaUBmWCQHPGpuV1_5wSj8aW2zIzYIMbofpjwIAT1bIZwJ7ufnLypRuq_MDbZhJkegDw10K14MHJ1J40Ip8mCoe0eXjdPfefij6WQpUq4z106N07j8kvDoVq9WALJiao7LYk_x9UIT-3d0W0eDBHGSRcNgtMyPyamaN9ltx6djceEsXnN4CFnWG3g.y6KKeA9EcsW9zW-g.TrTSgn4WBt18gezegxihBla9SLTC3YFDROqsL9K4yX4400FK1T1f-219CnpGTEdWxVi7sIMHC18S4oUiXd-rvY2mn_NfDrbbVJfgKp1j7Nnq9FFyeJEFCP_FtUXgsNTG9iwfzWox04B1d845qNRWiS9N8BhAAAIdz5N0ChHu0xsVOC0Y_Ly3DNe-JQyxCc964M6-jp3cgi4UqMxT837L6pLy5IH_iPsSfyHzstsFequIiktntz1MpT1yW-_GDyFK1S-SyV8PPQ7phgFouw2jho1iboHX70R1DGyYvmP1CfQzKE_zWxj3rgaCzvYMWN8fUenoiatzhvWkUm7dhqKGjofPeL8r0Mkh16afLLj0bzhUg3PZFcmR6guLjQdEwQFufWxGjfPvaHycZSKewu6-7dF8Hy_nyMLLdBpUkdrXPob_5gRiaH72KvncSIFvJLqhY3NgX005Fy87PORjggXwYkhWh4FgQZBIYD6h0CSk2nFFjR9uD9EKibBWSBzj814S_Jdw6HESFn91thpvU3hi3qNoi1m41gg1vt5Kh35A5AyDY1J7a9i_1N4B7e_pknX1VX6Z-Z2BYZvwAU7KLKsy5a99p9FX01lg6QweDzhukXrB4wgfKvVRTo.mjk92wMk-zUSrzuuuXPVeg",
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ik9YQjNhdTNScWhUQWN6R0RWZDM5djNpTmlyTWhqN2wxMjIySnh6TmgwRlkifQ.eyJleHAiOjE1MTMxMzAwNzsIm5iZii6MTUxMzEyNjQ30CwidmVyIjoiMS4wIiwiiaXNzIjoiaHR0cHM6Ly90Z55jcGltLndpbmRvd3MubmV0L2YwNmMyZmU4LTcwOWytNDAzMC04NWRjLTM4YTRiZmQ5ZTgyZC92M14wLyIsInN1YiI6Ik5vdCBzdXBwb3J0ZWQgY3VycmVudGx5LiBVc2Ugb2lkIGNsYwltLiIsImF1ZCI6ImJlZjIyZDU2LTU1MmYtNGE1Yi1iOTBhLTE50DhhN2Q2MzRjZSIisImFjciI6ImIyY18xYV9yZXNvdXJjZW93bmVydjiIiLCJpYXQiOjE1MTMxMjY0NzsImF1dGhfdGltZSI6MTUxMzEyNjQ30Cwib2lkIjoiM2MzYjk20WmtOGM00S00ZTExlWE0ZWYtzmr1MmYzOTJmMDQ5IiwiYXRfaGFzaCI6Ik6QUNCTVJtck1lwYm90dkFtNHHmWEEifQ.iAJg13cgySsdH3cmoEPGZB_g-408KwvGr6W5VzRXtkrlLoKB1p14hL6f_0xOrxnQwj2sUgW-wjsCVzMc_dkHSwd9QFZ4EYJEJbi1LMGk2lW-PgjsbwHPDU1mz-SR1PeqqJgv0qrzXo0YHXR-e07M4v4Tko-i_0YcrdJzj4Bkv7ZZilsSj621Nig4HkxtIWi5Ec2gD79bPKzgCtIww1KrnwmrlnCorMFYNj-0T31TDcXAQog63MOacf70uRVUC5k_IdseigeMSscrYrNrH28s3r0JaqDhNUTewuw1jx0X6gdqQWZKOLJ70F_EJMP-BkRTixBGK5eW2YeUEVQxsFlUg"
}
```

Redeem a refresh token

Construct a POST call like the one shown here with the information in the following table as the body of the request:

```
https://yourtenant.b2clogin.com/<yourtenant.onmicrosoft.com>/oauth2/v2.0/token?p=B2C_1_ROPC_Auth
```

KEY	VALUE
grant_type	refresh_token
response_type	id_token
client_id	<bef222d56-552f-4a5b-b90a-1988a7d634c3>
resource	<bef222d56-552f-4a5b-b90a-1988a7d634c3>
refresh_token	eyJraWQiOiJacW9pQlp2TW5pYVc2MUY0TnlfR3...

Client_id and *resource* are the values that you previously noted as the application ID. *Refresh_token* is the token that you received in the authentication call mentioned previously.

A successful response looks like the following example:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilglg1ZVhrNHh5b2p0RnVtMWtsM1l0djhkbE5QNC1jNTdkTzZRR1RWQndhTmsifQ.eyJpc3MiOiJodHRwczovL2xvZ2luLm1pY3Jvc29mdG9ubGluZS5jb20vNTE2ZmMwNjUtZmYzNi00YjkzLWFhNWetZDlZWRhN2NiYWM4L3YyLjAvIiwiZXhwIjoxNTMzNjc2NTkwLCJuYmYiOjE1MzM2NzI50TAisImF1ZC16IjljNTA2MThjLWY5NTEtNDlhNS1iZmU1LWQ3ODA4NTEyMWMzYSIsIm1kC16IkxvY2FsQWNjb3VudCIsInN1YiI6ImjZDgwODBjLTBjNDAtNDNjYS05ZTI3LTUyZTAyNzIyNWYyMSIsIm5hbWUiOjJEYXZpZE11IiwiZW1haWxZIjpbImRhdm1kd20xMDMwQGhvdG1haWwU29tI10sInRmcCI6IkIyQ18x1JPUENfQXV0aCIsImF6cCI6IjljNTA2MThjLWY5NTEtND1hNS1iZmU1LWQ3ODA4NTEyMWMzYSIsInZlcii6IjEuMCIsIm1hdCI6MTUzMzY3Mjk5MH0.RULweBR8--s5cCGG6Xoi8m-AGyCaASx9W5B3tNUQjbVkhnGdo2_0UrnVooZ1PTcrc1b0PQM2kVWi7NpYn57ifnqL_feTJPDbj9FJ8BmyxULdoECWxSM6KhsOPWZOig5y11NwN_IQ2HNF6uaDyYf1ZIM-jHr-uSfUnQxyWRnGdwNKX7TQbfMfk4oFmbPxTE7ioWAmxSnroiB4__P9D0rUM1vf_qfzempf2ErIWSF9rGtCNBG-BvJ1r3ZMCxIhRiIWmN2bVY0i3Nprzj0V8_FM6q8U19bvg9yDeZUcbe_1PMqzP3IrXW9N1XvQHups0j8Keb7SmpgY1GG091X6wBCypw",
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilglg1ZVhrNHh5b2p0RnVtMWtsM1l0djhkbE5QNC1jNTdkTzZRR1RWQndhTmsifQ.eyJleHAiOiJE1MzM2NzY10TAsIm5iziI6MTUzMzY3Mjk5MCwidmVjIjoiMS4wiLiwaXNzIjoiHR0cHM6Ly9sb2dpbi5taWNyb3NvZnRvbmxpbmuuY29tLzUxNmZjMDY1LWZmMzYtNGI5My1hYTvhLWQ2ZWVkyTDjYmFjOC92Mi4wLyIsInN1YiI6ImjZDgwODBjLTBjNDAtNDNjYS05ZTI3LTUyZTAyNzIyNWYyMSIsImF1ZC16IjljNTA2MThjLWY5NTEtNDlhNS1iZmU1LWQ3ODA4NTEyMWMzYSIsIm1hdCI6MTUzMzY3Mjk5MCwiYXV0aF90aW11IjoxNTMzNjcyOTkwLCJpZHAIoijMb2NhEFjY291bnQilCJuYw1lIjoiRGF2aWRNdsIsImVtYwlscyI6WyJkYXZpZHdtMTAzMEBob3RtYwlsLmNvbSJdLCj0ZnAi0iJCMkNfMV9ST1BDX0F1dGgiLCJhdF9oYXNoIjoiYw5hZ3QtX1NveUtBQV9UNFBLaHN4dyJ9.bPzpUFh94XFHC_yR6qH_Unf6_hN-9-BjDX0zrdB1Au0U6-owQ3fWDxNBuBYPEALid3sgm4qhJ6BROFKryD8awFrNyaErmYZwZ6rl1hK4fao3JsbGm3yNGPL0hz0FpC4Y9QhUjNgQ0xvnQLtqbHVNonSvBc7VVPAjBDza44GowmvLORFj1qkTjdrFM75H1LVeQch8cUNf-Ova77JdG5WHgYgqRhAq10hV68YgEpQkARyz77zbAz9zEHZZlgsli8UV6C-CPcmoHbwS-85mLzF9nLxhzjgIXJwckB6I71vTpfpRtaqZib3pMYeHZJaxaNLdvq9Qe4N-danXABg1B2w",
  "token_type": "Bearer",
  "not_before": 1533672990,
  "expires_in": 3600,
  "expires_on": 1533676590,
  "resource": "bef2222d56-552f-4a5b-b90a-1988a7d634c3",
  "id_token_expires_in": 3600,
  "profile_info": "eyJ2ZXIi0iIxLjAilCJ0aWQi0iI1MTZmYzA2NS1mZjM2LTRi0TMtYWE1YS1kNmVlZGE3Y2JhYzgilCJzdWIi0m51bGwsIm5hbWUiOjJEYXZpZE11IiwiCHJ1ZmVycmVkJX3VzZXJuYW11IjpuDWxsLCJpZHAIoijMb2NhEFjY291bnQifQ",
  "refresh_token": "eyJraWQiOjJcg1tY29yZV8wOTI1MjAxNSIsInZlcii6IjEuMCIsInppcCI6IkR1ZmxhdGuilCJzZXi0iIxLjAifQ..8oC4Q6aKdr35yMwm.p43lns-cfWNFbtmrhvtssQXCItb3E9aSlafZJ6nKnnpXGQ-Zap0OyH7hPK7AN_RT7NMsQwNdy0Fyv_hOMrFbMPZNvHsa91RsQIVBZ73-CVyoHNF0grSezjCATg4NVHfricuQVegEmZKF0oNP6TaMC2k1lEi3hrr08VE3ZfQ3j6j91BjaE9ybb02HwookqlzhiazwQyUhujw_R0TyxaQCI_gtLAr5QUxm7h1AfHhxR9uewQK1RbeuMH8ncMLSMASCJyzfeSJTjXmA0F0Vrxozrqz0Jdy0EETPR7oA48MJ916C2sy2ZELkqp0M3xhbhv-Re7nM09b8DeWuCw7VNTcQc9DKnIHDr-H5U2Tc-1MJQadgUNZv7KGSRGTyprWb7wF7FEPnRNID5PCDV_N_yoQpI7VvJO_NotXEgHFo70Hs5Gsgwp15mrDtyMzIMM7onTf101u46em_qltji7xcWNouHq4Ae0lcy9ZyThZgJH7l1vljReTwyXQuUpomXVeUGdc5pAnvgSozxnUbM7A1wfUeJzRT45P7L7683RSqChdNx1Qk0sXUECqxnFxMAz4VuZld2yFe-pzvxFF4_feQjBEmSCAvkpvJUrEticEs4QzByV5UZ2ZCKccijFTg4doACiCo_z13Jtm47mxm-5juhXOQqil69oxztk.KqI-z2L1C771lvwqmeFtdGO",
  "refresh_token_expires_in": 1209600
}
```

NOTE

When creating users via Graph API, the application needs to have "openid", "offline_access", and "profile" permissions from Microsoft Graph.

Implement with your preferred native SDK or use App-Auth

The Azure AD B2C implementation meets OAuth 2.0 standards for public client resource owner password credentials and should be compatible with most client SDKs. We have tested this flow extensively, in production, with AppAuth for iOS and AppAuth for Android. For the latest information, see [Native App SDK for OAuth 2.0](#) and [OpenID Connect implementing modern best practices](#).

Download working samples that have been configured for use with Azure AD B2C from GitHub, [for Android](#) and [for iOS](#).

Set up self-service password reset for your customers

1/28/2020 • 2 minutes to read • [Edit Online](#)

With the self-service password reset feature, your customers who have signed up for local accounts can reset their passwords on their own. This significantly reduces the burden on your support staff, especially if your application has millions of customers using it on a regular basis. Currently, using a verified email address is the only supported recovery method.

NOTE

This article applies to self-service password reset used in the context of the V1 **Sign in** user flow, which uses **Local Account SignIn** as the identity provider. If you need fully customizable password reset user flows invoked from your app, see [this article](#).

By default, your directory doesn't have self-service password reset turned on. Use the following steps to turn it on:

1. Sign in to the [Azure portal](#) as the Subscription Administrator. This is the same work or school account or the same Microsoft account that you used to create your directory.
2. Open **Azure Active Directory** (in the navigation bar on the left side).
3. Scroll down on the options blade and select **Password reset**.
4. Set **Self service password reset enabled** to **All**.
5. Click **Save** at the top of the page. You're done!

To test, use the "Run now" feature on any sign-in user flow that has local accounts as an identity provider. On the local account sign-in page (where you enter an email address and password, or a username and password), click **Can't access your account?** to verify the customer experience.

NOTE

The self-service password reset pages can be customized by using the [company branding feature](#).

Customize the user interface in Azure Active Directory B2C

2/17/2020 • 9 minutes to read • [Edit Online](#)

Branding and customizing the user interface that Azure Active Directory B2C (Azure AD B2C) displays to your customers helps provide a seamless user experience in your application. These experiences include signing up, signing in, profile editing, and password resetting. This article introduces the methods of user interface (UI) customization for both user flows and custom policies.

UI customization in different scenarios

There are several ways to customize the UI of the user experiences your application, each appropriate for different scenarios.

User flows

If you use [user flows](#), you can change the look of your user flow pages by using built-in *page layout templates*, or by using your own HTML and CSS. Both methods are discussed later in this article.

You use the [Azure portal](#) to configure the UI customization for user flows.

TIP

If you want to modify only the banner logo, background image, and background color of your user flow pages, you can try the [Company branding \(preview\)](#) feature described later in this article.

Custom policies

If you're using [custom policies](#) to provide sign-up or sign-in, password reset, or profile-editing in your application, use [policy files to customize the UI](#).

If you need to provide dynamic content based on a customer's decision, use custom policies that can [change page content dynamically](#) depending on a parameter that's sent in a query string. For example, you can change the background image on the Azure AD B2C sign-up or sign-in page based on a parameter that you pass from your web or mobile application.

JavaScript

You can enable client-side JavaScript code in both [user flows](#) and [custom policies](#).

Sign in-only UI customization

If you're providing sign-in only, along with its accompanying password reset page and verification emails, use the same customization steps that are used for an [Azure AD sign-in page](#).

If customers try to edit their profile before signing in, they're redirected to a page that you customize by using the same steps that are used for customizing the Azure AD sign-in page.

Page layout templates

User flows provide several built-in templates you can choose from to give your user experience pages a professional look. These layout templates can also serve as starting point for your own customization.

Under **Customize** in the left menu, select **Page layouts** and then select **Template**.

Next, select a template from the list. Here are examples of the sign-in pages for each template:

OCEAN BLUE	SLATE GRAY	CLASSIC

When you choose a template, the selected layout is applied to all pages in your user flow, and the URI for each page is visible in the **Custom page URI** field.

Custom HTML and CSS

If you wish to design your own policy layout with your customized HTML and CSS, you can do so by switching the "Use custom page content" toggle for each of the Layout names present in your policy. Please follow the below instructions regarding the custom layout configurations:

Azure AD B2C runs code in your customer's browser by using an approach called [Cross-Origin Resource Sharing \(CORS\)](#).

At runtime, content is loaded from a URL that you specify in your user flow or custom policy. Each page in the user experience loads its content from the URL you specify for that page. After content is loaded from your URL, it's merged with an HTML fragment inserted by Azure AD B2C, and then the page is displayed to your customer.

Review the following guidance before using your own HTML and CSS files to customize the UI:

- Azure AD B2C **merges** HTML content into your pages. Don't copy and try to change the default content that Azure AD B2C provides. It's best to build your HTML content from scratch and use the default content as reference.
- **JavaScript** can be included in your custom content for both [user flows](#) and [custom policies](#).
- Supported **browser versions** are:
 - Internet Explorer 11, 10, and Microsoft Edge
 - Limited support for Internet Explorer 9 and 8

- Google Chrome 42.0 and above
- Mozilla Firefox 38.0 and above
- Don't include **form tags** in your HTML. Form tags interfere with the POST operations generated by the HTML injected by Azure AD B2C.

Where do I store UI content?

When using your own HTML and CSS files to customize the UI, you can host your UI content on any publicly available HTTPS endpoint that supports CORS. For example, [Azure Blob storage](#), web servers, CDNs, AWS S3, or file sharing systems.

The important point is that you host the content on a publicly available HTTPS endpoint with CORS enabled. You must use an absolute URL when you specify it in your content.

Get started with custom HTML and CSS

Get started using your own HTML and CSS in your user experience pages by following these guidelines.

- Create well-formed HTML content with an empty `<div id="api"></div>` element located somewhere in the `<body>`. This element marks where the Azure AD B2C content is inserted. The following example shows a minimal page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Add your title here!</title>
    <link rel="stylesheet" href="https://mystore1.blob.core.windows.net/b2c/style.css">
  </head>
  <body>
    <h1>My B2C Application</h1>
    <div id="api"></div>  <!-- Leave this element empty because Azure AD B2C will insert content here.
--&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre>

```

- Use CSS to style the UI elements that Azure AD B2C inserts into your page. The following example shows a simple CSS file that also includes settings for the sign-up injected HTML elements:

```
h1 {
  color: blue;
  text-align: center;
}
.intro h2 {
  text-align: center;
}
.entry {
  width: 400px ;
  margin-left: auto ;
  margin-right: auto ;
}
.divider h2 {
  text-align: center;
}
.create {
  width: 400px ;
  margin-left: auto ;
  margin-right: auto ;
}
```

- Host your content on an HTTPS endpoint (with CORS allowed). Both GET and OPTIONS request methods

must be enabled when configuring CORS.

- Create or edit a user flow or custom policy to use the content that you created.

HTML fragments from Azure AD B2C

The following table lists the HTML fragments that Azure AD B2C merges into the `<div id="api"></div>` element located in your content.

INSERTED PAGE	DESCRIPTION OF HTML
Identity provider selection	Contains a list of buttons for identity providers that the customer can choose from during sign-up or sign-in. These buttons include social identity providers such as Facebook, Google, or local accounts (based on email address or user name).
Local account sign-up	Contains a form for local account sign-up based on an email address or a user name. The form can contain different input controls such as text input box, password entry box, radio button, single-select drop-down boxes, and multi-select check boxes.
Social account sign-up	May appear when signing up using an existing account from a social identity provider such as Facebook or Google. It's used when additional information must be collected from the customer using a sign-up form.
Unified sign-up or sign-in	Handles both sign-up and sign-in of customers who can use social identity providers such as Facebook, Google, or local accounts.
Multi-factor authentication	Customers can verify their phone numbers (using text or voice) during sign-up or sign-in.
Error	Provides error information to the customer.

Company branding (preview)

You can customize your user flow pages with a banner logo, background image, and background color by using Azure Active Directory [Company branding](#).

To customize your user flow pages, you first configure company branding in Azure Active Directory, then you enable it in the page layouts of your user flows in Azure AD B2C.

NOTE

This feature is in public preview.

Configure company branding

Start by setting the banner logo, background image, and background color within **Company branding**.

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Under **Manage**, select **Company branding**.

5. Follow the steps in [Add branding to your organization's Azure Active Directory sign-in page](#)

Keep these things in mind when you configure company branding in Azure AD B2C:

- Company branding in Azure AD B2C is currently limited to **background image**, **banner logo**, and **background color** customization. The other properties in the company branding pane, for example those in **Advanced settings**, are *not supported*.
- In your user flow pages, the background color is shown before the background image is loaded. We recommend you choose a background color that closely matches the colors in your background image for a smoother loading experience.
- The banner logo appears in the verification emails sent to your users when they initiate a sign-up user flow.

Enable branding in user flow pages

Once you've configured company branding, enable it in your user flows.

1. In the left menu of the Azure portal, select **Azure AD B2C**.
2. Under **Policies**, select **User flows (policies)**.
3. Select the user flow for which you'd like to enable company branding. Company branding is **not supported** for the *Sign in v1* and *Profile editing v1* user flow types.
4. Under **Customize**, select **Page layouts**, and then select the layout you'd like to brand. For example, select **Unified sign up or sign in page**.
5. For the **Page Layout Version (Preview)**, choose version **1.2.0** or above.
6. Select **Save**.

If you'd like to brand all pages in the user flow, set the page layout version for each page layout in the user flow.

B2C_1_signupin - Page layouts
Sign up and sign in v2 (Preview)

Search (Ctrl+ /) | Run user flow | Save | Discard | Template

Overview

Settings

Properties

Identity providers

User attributes

Application claims

Customize

Page layouts

Languages

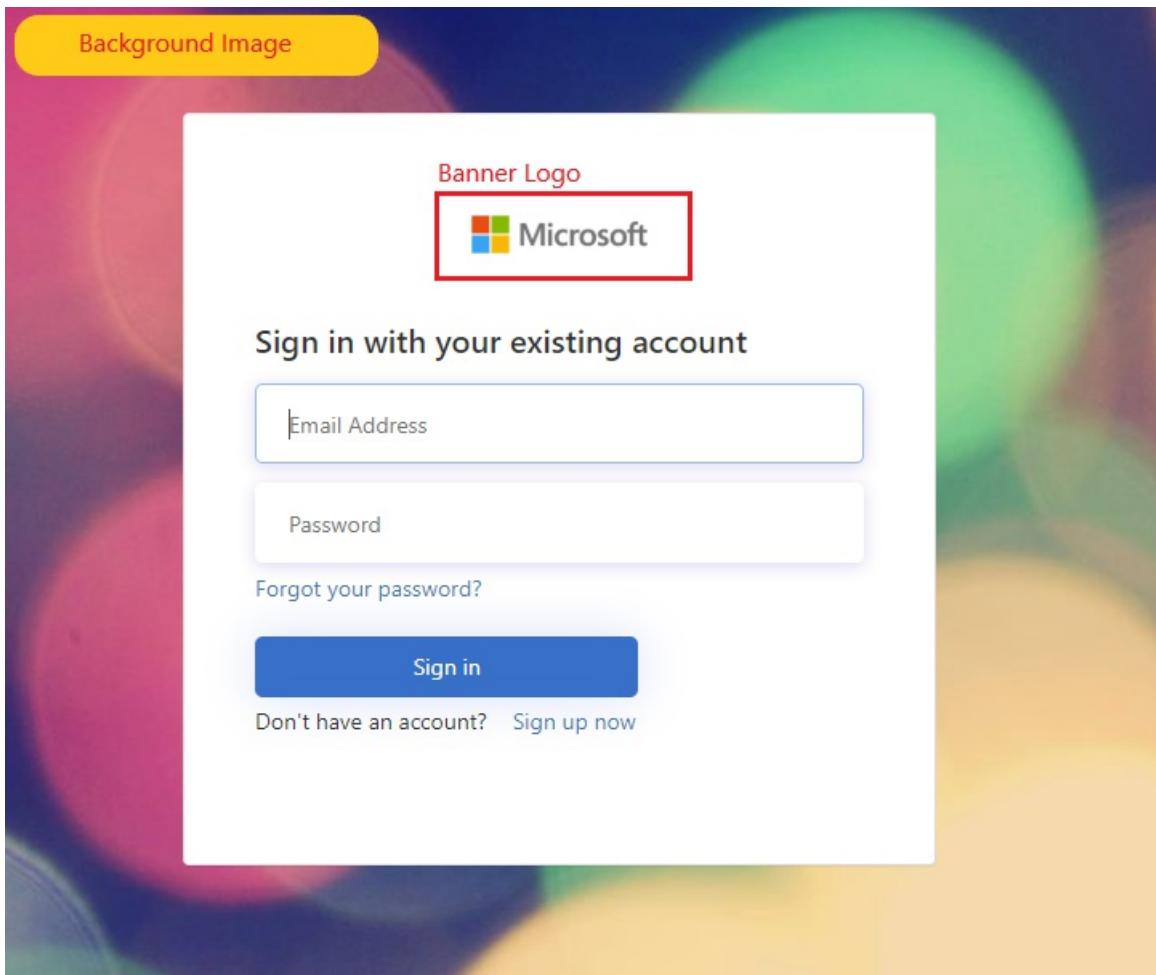
Got a second? We would love your feedback on customizing user flows →

Select a page to customize its appearance. You can provide your own html and css to add your own branding and layout. [Learn more about customizing your page](#).

Layout name	Custom page
Unified sign up or sign in page	No
Local account sign up page	No
Error page	No

Unified sign up or sign in page: 1.2.0 - Current or higher
Use custom page content
Custom page URI * ① 1.1.0
1.0.0
Page Layout Version (Preview) ① 1.2.0 - Current ▾ Learn more about Page Layout versions.

This annotated example shows a custom banner logo and background image on a *Sign up and sign in* user flow page that uses the Ocean Blue template:



Use company branding assets in custom HTML

To use your company branding assets in custom HTML, add the following tags outside the `<div id="api">` tag:

```
<img data-tenant-branding-background="true" />
<img data-tenant-branding-logo="true" alt="Company Logo" />
```

The image source is replaced with that of the background image and banner logo. As described in the [Get started with custom HTML and CSS](#) section, use CSS classes to style and position the assets on the page .

Localize content

You localize your HTML content by enabling [language customization](#) in your Azure AD B2C tenant. Enabling this feature allows Azure AD B2C to forward the OpenID Connect parameter `ui-locales` to your endpoint. Your content server can use this parameter to provide language-specific HTML pages.

Content can be pulled from different places based on the locale that's used. In your CORS-enabled endpoint, you set up a folder structure to host content for specific languages. You'll call the right one if you use the wildcard value `{Culture:RFC5646}` .

For example, your custom page URI might look like:

```
https://contoso.blob.core.windows.net/{Culture:RFC5646}/myHTML/unified.html
```

You can load the page in French by pulling content from:

```
https://contoso.blob.core.windows.net/fr/myHTML/unified.html
```

Examples

You can find several sample template files in the [B2C-AzureBlobStorage-Client](#) repository on GitHub.

The sample HTML and CSS files in the templates are located in the [/sample_templates](#) directory.

Next steps

- If you're using **user flows**, you can start customizing your UI with the tutorial:

[Customize the user interface of your applications in Azure Active Directory B2C.](#)

- If you're using **custom policies**, you can start customizing the UI with the article:

[Customize the user interface of your application using a custom policy in Azure Active Directory B2C.](#)

JavaScript and page layout versions in Azure Active Directory B2C

2/11/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

Azure AD B2C provides a set of packaged content containing HTML, CSS, and JavaScript for the user interface elements in your user flows and custom policies.

To enable JavaScript for your applications:

- Enable it on the user flow by using the Azure portal
- Select a [page layout](#)
- Use [b2clogin.com](#) in your requests

If you intend to enable [JavaScript](#) client-side code, the elements you base your JavaScript on must be immutable. If they're not immutable, any changes could cause unexpected behavior on your user pages. To prevent these issues, enforce the use of a page layout and specify a page layout version to ensure the content definitions you've based your JavaScript on are immutable. Even if you don't intend to enable JavaScript, you can specify a page layout version for your pages.

Enable JavaScript

In the user flow **Properties**, you can enable JavaScript. Enabling JavaScript also enforces the use of a page layout. You can then set the page layout version for the user flow as described in the next section.

The screenshot shows the Azure AD B2C - User flows (policies) blade. A user flow named "B2C_1_signupsignin1" is selected. The "Properties" tab is active, indicated by a red box. In the "Multifactor authentication" section, there is a toggle switch labeled "Enabled" (which is highlighted with a red box). Below this, there is a button labeled "Enable JavaScript enforcing page layout (preview)" with a switch that is set to "On" (also highlighted with a red box). At the top right, there are "Run user flow", "Save", and "Discard" buttons.

Select a page layout version

Whether or not you enable JavaScript in your user flow's properties, you can specify a page layout version for your user flow pages. Open the user flow and select **Page layouts**. Under **LAYOUT NAME**, select a user flow page and choose the **Page Layout Version**.

For information about the different page layout versions, see the [Page layout version change log](#).

Home > Azure AD B2C - User flows (policies) > B2C_1_signupsignin1 - Page layouts

B2C_1_signupsignin1 - Page layouts

Sign up and sign in

Search (Ctrl+ /)

Run user flow | Save | Discard

Got a second? We would love your feedback on customizing user flows →

Select a page to customize its appearance. You can provide your own html and css to add your own branding and layout. [Learn more about customizing your page.](#)

LAYOUT NAME	CUSTOM PAGE
Unified sign up or sign in page	No
Local account sign up page	No
Social account sign up page	No
Error page	No

Customize

- Overview
- Properties
- Identity providers
- User attributes
- Application claims
- Page layouts**
- Languages

Unified sign up or sign in page

Use custom page content

* Custom page URI <https://marsmatest.b2clogin.com/static/tenant/default/unifi...>

Page Layout Version (Preview) [Learn more about Page Layout versions.](#)

1.1.0
1.0.0

NAME	IDENTITY PROVIDER	LABEL
Facebook	Facebook	Facebook

Guidelines for using JavaScript

Follow these guidelines when you customize the interface of your application using JavaScript:

- Don't bind a click event on `<a>` HTML elements.
- Don't take a dependency on Azure AD B2C code or comments.
- Don't change the order or hierarchy of Azure AD B2C HTML elements. Use an Azure AD B2C policy to control the order of the UI elements.
- You can call any RESTful service with these considerations:
 - You may need to set your RESTful service CORS to allow client-side HTTP calls.
 - Make sure your RESTful service is secure and uses only the HTTPS protocol.
 - Don't use JavaScript directly to call Azure AD B2C endpoints.
- You can embed your JavaScript or you can link to external JavaScript files. When using an external JavaScript file, make sure to use the absolute URL and not a relative URL.
- JavaScript frameworks:
 - Azure AD B2C uses a specific version of jQuery. Don't include another version of jQuery. Using more than one version on the same page causes issues.
 - Using RequireJS isn't supported.
 - Most JavaScript frameworks are not supported by Azure AD B2C.
- Azure AD B2C settings can be read by calling `window.SETTINGS`, `window.CONTENT` objects, such as the current UI language. Don't change the value of these objects.
- To customize the Azure AD B2C error message, use localization in a policy.
- If anything can be achieved by using a policy, generally it's the recommended way.

Next steps

You can find examples of JavaScript usage in [JavaScript samples for use in Azure Active Directory B2C](#).

Language customization in Azure Active Directory B2C

1/28/2020 • 7 minutes to read • [Edit Online](#)

Language customization in Azure Active Directory B2C (Azure AD B2C) allows your user flow to accommodate different languages to suit your customer needs. Microsoft provides the translations for [36 languages](#), but you can also provide your own translations for any language. Even if your experience is provided for only a single language, you can customize any text on the pages.

How language customization works

You use language customization to select which languages your user flow is available in. After the feature is enabled, you can provide the query string parameter, `ui_locales`, from your application. When you call into Azure AD B2C, your page is translated to the locale that you have indicated. This type of configuration gives you complete control over the languages in your user flow and ignores the language settings of the customer's browser.

You might not need that level of control over what languages your customer sees. If you don't provide a `ui_locales` parameter, the customer's experience is dictated by their browser's settings. You can still control which languages your user flow is translated to by adding it as a supported language. If a customer's browser is set to show a language that you don't want to support, then the language that you selected as a default in supported cultures is shown instead.

- **ui-locales specified language:** After you enable language customization, your user flow is translated to the language that's specified here.
- **Browser-requested language:** If no `ui_locales` parameter was specified, your user flow is translated to the browser-requested language, *if the language is supported*.
- **Policy default language:** If the browser doesn't specify a language, or it specifies one that is not supported, the user flow is translated to the user flow default language.

NOTE

If you're using custom user attributes, you need to provide your own translations. For more information, see [Customize your strings](#).

Support requested languages for ui_locales

Policies that were created before the general availability of language customization need to enable this feature first. Policies and user flows that were created after have language customization enabled by default.

When you enable language customization on a user flow, you can control the language of the user flow by adding the `ui_locales` parameter.

1. In your Azure AD B2C tenant, select **User flows**.
2. Click the user flow that you want to enable for translations.
3. Select **Languages**.
4. Select **Enable language customization**.

Select which languages in your user flow are enabled

Enable a set of languages for your user flow to be translated to when requested by the browser without the `ui_locales` parameter.

1. Ensure that your user flow has language customization enabled from previous instructions.
2. On the **Languages** page for the user flow, select a language that you want to support.
3. In the properties pane, change **Enabled** to **Yes**.
4. Select **Save** at the top of the properties pane.

NOTE

If a `ui_locales` parameter is not provided, the page is translated to the customer's browser language only if it is enabled.

Customize your strings

Language customization enables you to customize any string in your user flow.

1. Ensure that your user flow has language customization enabled from the previous instructions.
2. On the **Languages** page for the user flow, select the language that you want to customize.
3. Under **Page-level-resources files**, select the page that you want to edit.
4. Select **Download defaults** (or **Download overrides** if you have previously edited this language).

These steps give you a JSON file that you can use to start editing your strings.

Change any string on the page

1. Open the JSON file downloaded from previous instructions in a JSON editor.
2. Find the element that you want to change. You can find `StringId` for the string you're looking for, or look for the `Value` attribute that you want to change.
3. Update the `value` attribute with what you want displayed.
4. For every string that you want to change, change `Override` to `true`.
5. Save the file and upload your changes. (You can find the upload control in the same place as where you downloaded the JSON file.)

IMPORTANT

If you need to override a string, make sure to set the `Override` value to `true`. If the value isn't changed, the entry is ignored.

Change extension attributes

If you want to change the string for a custom user attribute, or you want to add one to the JSON, it's in the following format:

```
{
  "LocalizedStrings": [
    {
      "ElementType": "ClaimType",
      "ElementId": "extension_<ExtensionAttribute>",
      "StringId": "DisplayName",
      "Override": true,
      "Value": "<ExtensionAttributeValue>"
    }
  [...]
}
```

Replace `<ExtensionAttribute>` with the name of your custom user attribute.

Replace `<ExtensionAttributeValue>` with the new string to be displayed.

Provide a list of values by using LocalizedCollections

If you want to provide a set list of values for responses, you need to create a `LocalizedCollections` attribute.

`LocalizedCollections` is an array of `Name` and `Value` pairs. The order for the items will be the order they are displayed. To add `LocalizedCollections`, use the following format:

```
{
  "LocalizedStrings": [...],
  "LocalizedCollections": [
    {
      "ElementType": "ClaimType",
      "ElementId": "<UserAttribute>",
      "TargetCollection": "Restriction",
      "Override": true,
      "Items": [
        {
          "Name": "<Response1>",
          "Value": "<Value1>"
        },
        {
          "Name": "<Response2>",
          "Value": "<Value2>"
        }
      ]
    }
  ]
}
```

- `ElementId` is the user attribute that this `LocalizedCollections` attribute is a response to.
- `Name` is the value that's shown to the user.
- `Value` is what is returned in the claim when this option is selected.

Upload your changes

1. After you complete the changes to your JSON file, go back to your B2C tenant.
2. Select **User flows** and click the user flow that you want to enable for translations.
3. Select **Languages**.
4. Select the language that you want to translate to.
5. Select the page where you want to provide translations.
6. Select the folder icon, and select the JSON file to upload.

The changes are saved to your user flow automatically.

Customize the page UI by using language customization

There are two ways to localize your HTML content. One way is to turn on [language customization](#). Enabling this

feature allows Azure AD B2C to forward the OpenID Connect parameter, `ui-locales`, to your endpoint. Your content server can use this parameter to provide customized HTML pages that are language-specific.

Alternatively, you can pull content from different places based on the locale that's used. In your CORS-enabled endpoint, you can set up a folder structure to host content for specific languages. You'll call the right one if you use the wildcard value `{Culture:RFC5646}`. For example, assume that this is your custom page URI:

```
https://wingtiptoybs2c.blob.core.windows.net/{Culture:RFC5646}/wingtip/unified.html
```

You can load the page in `fr`. When the page pulls HTML and CSS content, it's pulling from:

```
https://wingtiptoybs2c.blob.core.windows.net/fr/wingtip/unified.html
```

Add custom languages

You can also add languages that Microsoft currently does not provide translations for. You'll need to provide the translations for all the strings in the user flow. Language and locale codes are limited to those in the ISO 639-1 standard.

1. In your Azure AD B2C tenant, select **User flows**.
2. Click the user flow where you want to add custom languages, and then click **Languages**.
3. Select **Add custom language** from the top of the page.
4. In the context pane that opens, identify which language you're providing translations for by entering a valid locale code.
5. For each page, you can download a set of overrides for English and work on the translations.
6. After you're done with the JSON files, you can upload them for each page.
7. Select **Enable**, and your user flow can now show this language for your users.
8. Save the language.

IMPORTANT

You need to either enable the custom languages or upload overrides for it before you can save.

Additional information

Page UI customization labels as overrides

When you enable language customization, your previous edits for labels using page UI customization are persisted in a JSON file for English (en). You can continue to change your labels and other strings by uploading language resources in language customization.

Up-to-date translations

Microsoft is committed to providing the most up-to-date translations for your use. Microsoft continuously improves translations and keeps them in compliance for you. Microsoft will identify bugs and changes in global terminology and make updates that will work seamlessly in your user flow.

Support for right-to-left languages

Microsoft currently doesn't provide support for right-to-left languages. You can accomplish this by using custom locales and using CSS to change the way the strings are displayed. If you need this feature, please vote for it on [Azure Feedback](#).

Social identity provider translations

Microsoft provides the `ui_locales` OIDC parameter to social logins. But some social identity providers, including Facebook and Google, don't honor them.

Browser behavior

Chrome and Firefox both request for their set language. If it's a supported language, it's displayed before the default. Microsoft Edge currently does not request a language and goes straight to the default language.

Supported languages

Azure AD B2C includes support for the following languages. User flow languages are provided by Azure AD B2C. The multi-factor authentication (MFA) notification languages are provided by [Azure MFA](#).

LANGUAGE	LANGUAGE CODE	USER FLOWS	MFA NOTIFICATIONS
Arabic	ar	✗	✓
Bulgarian	bg	✗	✓
Bangla	bn	✓	✗
Catalan	ca	✗	✓
Czech	cs	✓	✓
Danish	da	✓	✓
German	de	✓	✓
Greek	el	✓	✓
English	en	✓	✓
Spanish	es	✓	✓
Estonian	et	✗	✓
Basque	eu	✗	✓
Finnish	fi	✓	✓
French	fr	✓	✓
Galician	gl	✗	✓
Gujarati	gu	✓	✗
Hebrew	he	✗	✓
Hindi	hi	✓	✓
Croatian	hr	✓	✓

LANGUAGE	LANGUAGE CODE	USER FLOWS	MFA NOTIFICATIONS
Hungarian	hu	✓	✓
Indonesian	id	✗	✓
Italian	it	✓	✓
Japanese	ja	✓	✓
Kazakh	kk	✗	✓
Kannada	kn	✓	✗
Korean	ko	✓	✓
Lithuanian	lt	✗	✓
Latvian	lv	✗	✓
Malayalam	ml	✓	✗
Marathi	mr	✓	✗
Malay	ms	✓	✓
Norwegian Bokmal	nb	✓	✗
Dutch	nl	✓	✓
Norwegian	no	✗	✓
Punjabi	pa	✓	✗
Polish	pl	✓	✓
Portuguese - Brazil	pt-br	✓	✓
Portuguese - Portugal	pt-pt	✓	✓
Romanian	ro	✓	✓
Russian	ru	✓	✓
Slovak	sk	✓	✓
Slovenian	sl	✗	✓
Serbian - Cyrillic	sr-cryl-cs	✗	✓

LANGUAGE	LANGUAGE CODE	USER FLOWS	MFA NOTIFICATIONS
Serbian - Latin	sr-latn-cs	✗	✓
Swedish	sv	✓	✓
Tamil	ta	✓	✗
Telugu	te	✓	✗
Thai	th	✓	✓
Turkish	tr	✓	✓
Ukrainian	uk	✗	✓
Vietnamese	vi	✗	✓
Chinese - Simplified	zh-hans	✓	✓
Chinese - Traditional	zh-hant	✓	✓

Configure complexity requirements for passwords in Azure Active Directory B2C

2/18/2020 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) supports changing the complexity requirements for passwords supplied by an end user when creating an account. By default, Azure AD B2C uses **Strong** passwords. Azure AD B2C also supports configuration options to control the complexity of passwords that customers can use.

Password rule enforcement

During sign-up or password reset, an end user must supply a password that meets the complexity rules. Password complexity rules are enforced per user flow. It is possible to have one user flow require a four-digit pin during sign-up while another user flow requires an eight character string during sign-up. For example, you may use a user flow with different password complexity for adults than for children.

Password complexity is never enforced during sign-in. Users are never prompted during sign-in to change their password because it doesn't meet the current complexity requirement.

Password complexity can be configured in the following types of user flows:

- Sign-up or Sign-in user flow
- Password Reset user flow

If you are using custom policies, you can ([configure password complexity in a custom policy](#)).

Configure password complexity

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Select **User flows (policies)**.
5. Select a user flow, and click **Properties**.
6. Under **Password complexity**, change the password complexity for this user flow to **Simple**, **Strong**, or **Custom**.

Comparison Chart

COMPLEXITY	DESCRIPTION
Simple	A password that is at least 8 to 64 characters.
Strong	A password that is at least 8 to 64 characters. It requires 3 out of 4 of lowercase, uppercase, numbers, or symbols.
Custom	This option provides the most control over password complexity rules. It allows configuring a custom length. It also allows accepting number-only passwords (pins).

Custom options

Character Set

Allows you to accept digits only (pins) or the full character set.

- **Numbers only** allows digits only (0-9) while entering a password.
- **All** allows any letter, number, or symbol.

Length

Allows you to control the length requirements of the password.

- **Minimum Length** must be at least 4.
- **Maximum Length** must be greater or equal to minimum length and at most can be 64 characters.

Character classes

Allows you to control the different character types used in the password.

- **2 of 4: Lowercase character, Uppercase character, Number (0-9), Symbol** ensures the password contains at least two character types. For example, a number and a lowercase character.
- **3 of 4: Lowercase character, Uppercase character, Number (0-9), Symbol** ensures the password contains at least three character types. For example, a number, a lowercase character and an uppercase character.
- **4 of 4: Lowercase character, Uppercase character, Number (0-9), Symbol** ensures the password contains all four character types.

NOTE

Requiring **4 of 4** can result in end-user frustration. Some studies have shown that this requirement does not improve password entropy. See [NIST Password Guidelines](#)

Disable email verification during customer sign-up in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

By default, Azure Active Directory B2C (Azure AD B2C) verifies your customer's email address for local accounts (accounts for users who sign up with email address or username). Azure AD B2C ensures valid email addresses by requiring customers to verify them during the sign-up process. It also prevents a malicious actors from using automated processes to generate fraudulent accounts in your applications.

Some application developers prefer to skip email verification during the sign-up process and instead have customers verify their email address later. To support this, Azure AD B2C can be configured to disable email verification. Doing so creates a smoother sign-up process and gives developers the flexibility to differentiate customers that have verified their email address from customers that have not.

Follow these steps to disable email verification:

1. Sign in to the [Azure portal](#)
2. Use the **Directory + subscription** filter in the top menu to select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **User flows**.
5. Select the user flow for which you want to disable email verification. For example, *B2C_1_signinsignup*.
6. Select **Page layouts**.
7. Select **Local account sign-up page**.
8. Under **User attributes**, select **Email Address**.
9. In the **REQUIRES VERIFICATION** drop down, select **No**.
10. Select **Save**. Email verification is now disabled for this user flow.

WARNING

Disabling email verification in the sign-up process may lead to spam. If you disable the default Azure AD B2C-provided email verification, we recommend that you implement a replacement verification system.

Enable multi-factor authentication in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) integrates directly with [Azure Multi-Factor Authentication](#) so that you can add a second layer of security to sign-up and sign-in experiences in your applications. You enable multi-factor authentication without writing a single line of code. If you already created sign up and sign-in user flows, you can still enable multi-factor authentication.

This feature helps applications handle scenarios such as the following:

- You don't require multi-factor authentication to access one application, but you do require it to access another. For example, the customer can sign into an auto insurance application with a social or local account, but must verify the phone number before accessing the home insurance application registered in the same directory.
- You don't require multi-factor authentication to access an application in general, but you do require it to access the sensitive portions within it. For example, the customer can sign in to a banking application with a social or local account and check the account balance, but must verify the phone number before attempting a wire transfer.

Set multi-factor authentication

When you create a user flow, you have the option to enable multi-factor authentication.

The screenshot shows the 'Create a user flow' wizard in the Azure AD B2C portal. The steps are as follows:

- 1. Name ***: A unique string for the user flow, currently set to "SignUpIn".
- 2. Identity providers ***: Options include "Email signup" (selected).
- 3. Multifactor authentication**: A note states: "Enabling multifactor authentication (MFA) requires your users to verify their identity with a second factor before allowing them into your application." The "Multifactor authentication" toggle switch is set to "Enabled".
- 4. User attributes and claims**: This step is not yet completed.

Set **Multifactor authentication** to **Enabled**.

You can use **Run user flow** to verify the experience. Confirm the following scenario:

A customer account is created in your tenant before the multi-factor authentication step occurs. During the step, the customer is asked to provide a phone number and verify it. If verification is successful, the phone number is attached to the account for later use. Even if the customer cancels or drops out, the customer can be asked to verify a phone number again during the next sign-in with multi-factor authentication enabled.

Add multi-factor authentication

It's possible to enable multi-factor authentication on a user flow that you previously created.

To enable multi-factor authentication:

1. Open the user flow and then select **Properties**.
2. Next to **Multifactor authentication**, select **Enabled**.
3. Click **Save** at the top of the page.

Set up sign-up and sign-in with an Amazon account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create an Amazon application

To use an Amazon account as an [identity provider](#) in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have an Amazon account you can sign up at <https://www.amazon.com/>.

1. Sign in to the [Amazon Developer Center](#) with your Amazon account credentials.
2. If you have not already done so, click **Sign Up**, follow the developer registration steps, and accept the policy.
3. Select **Register new application**.
4. Enter a **Name**, **Description**, and **Privacy Notice URL**, and then click **Save**. The privacy notice is a page that you manage that provides privacy information to users.
5. In the **Web Settings** section, copy the values of **Client ID**. Select **Show Secret** to get the client secret and then copy it. You need both of them to configure an Amazon account as an identity provider in your tenant. **Client Secret** is an important security credential.
6. In the **Web Settings** section, select **Edit**, and then enter `https://your-tenant-name.b2clogin.com` in **Allowed JavaScript Origins** and `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **Allowed Return URLs**. Replace `your-tenant-name` with the name of your tenant. You need to use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
7. Click **Save**.

Configure an Amazon account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Amazon**.
5. Enter a **Name**. For example, *Amazon*.
6. For the **Client ID**, enter the Client ID of the Amazon application that you created earlier.
7. For the **Client secret**, enter the Client Secret that you recorded.
8. Select **Save**.

Set up sign-in for a specific Azure Active Directory organization in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

To use an Azure Active Directory (Azure AD) as an [identity provider](#) in Azure AD B2C, you need to create an application that represents it. This article shows you how to enable sign-in for users from a specific Azure AD organization using a user flow in Azure AD B2C.

Create an Azure AD app

To enable sign-in for users from a specific Azure AD organization, you need to register an application within the organizational Azure AD tenant, which is not the same as your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
 2. Make sure you're using the directory that contains your Azure AD tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your Azure AD tenant. This is not the same tenant as your Azure AD B2C tenant.
 3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
 4. Select **New registration**.
 5. Enter a name for your application. For example, `Azure AD B2C App`.
 6. Accept the selection of **Accounts in this organizational directory only** for this application.
 7. For the **Redirect URI**, accept the value of **Web**, and enter the following URL in all lowercase letters, where `your-B2C-tenant-name` is replaced with the name of your Azure AD B2C tenant. For example,
`https://fabrikam.b2clogin.com/fabrikam.onmicrosoft.com/oauth2/authresp` :
- `https://your-B2C-tenant-name.b2clogin.com/your-B2C-tenant-name.onmicrosoft.com/oauth2/authresp`
- All URLs should now be using [b2clogin.com](#).
8. Click **Register**. Copy the **Application (client) ID** to be used later.
 9. Select **Certificates & secrets** in the application menu, and then select **New client secret**.
 10. Enter a name for the client secret. For example, `Azure AD B2C App Secret`.
 11. Select the expiration period. For this application, accept the selection of **In 1 year**.
 12. Select **Add** and copy the value of the new client secret that is displayed to be used later.

Configure Azure AD as an identity provider

1. Make sure you're using the directory that contains Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your Azure AD B2C tenant.
2. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.

3. Select **Identity providers**, and then select **New OpenID Connect provider**.
4. Enter a **Name**. For example, enter *Contoso Azure AD*.
5. For **Metadata url**, enter the following URL replacing `your-AD-tenant-domain` with the domain name of your Azure AD tenant:

```
https://login.microsoftonline.com/your-AD-tenant-domain/.well-known/openid-configuration
```

For example, `https://login.microsoftonline.com/contoso.onmicrosoft.com/.well-known/openid-configuration`.

Do not use the Azure AD v2.0 metadata endpoint, for example

`https://login.microsoftonline.com/contoso.onmicrosoft.com/v2.0/.well-known/openid-configuration`. Doing so results in an error similar to

```
AADB2C: A claim with id 'UserId' was not found, which is required by ClaimsTransformation 'CreateAlternativeSecurityId' with id 'CreateAlternativeSecurityId' in policy 'B2C_1_SignUpOrIn' of tenant 'contoso.onmicrosoft.com'
```

when attempting to sign in.

6. For **Client ID**, enter the application ID that you previously recorded.
7. For **Client secret**, enter the client secret that you previously recorded.
8. Leave the default values for **Scope**, **Response type**, and **Response mode**.
9. (Optional) Enter a value for **Domain_hint**. For example, *ContosoAD*. This is the value to use when referring to this identity provider using *domain_hint* in the request.
10. Under **Identity provider claims mapping**, enter the following claims mapping values:
 - **User ID**: *oid*
 - **Display name**: *name*
 - **Given name**: *given_name*
 - **Surname**: *family_name*
 - **Email**: *unique_name*
11. Select **Save**.

Set up sign-up and sign-in with a Microsoft account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create a Microsoft account application

To use a Microsoft account as an [identity provider](#) in Azure Active Directory B2C (Azure AD B2C), you need to create an application in the Azure AD tenant. The Azure AD tenant is not the same as your Azure AD B2C tenant. If you don't already have a Microsoft account, you can get one at <https://www.live.com/>.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your Azure AD tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
4. Select **New registration**.
5. Enter a **Name** for your application. For example, *MSAapp1*.
6. Under **Supported account types**, select **Accounts in any organizational directory and personal Microsoft accounts (e.g. Skype, Xbox, Outlook.com)**. This option targets the widest set of Microsoft identities.

For more information on the different account type selections, see [Quickstart: Register an application with the Microsoft identity platform](#).
7. Under **Redirect URI (optional)**, select **Web** and enter
`https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in the text box.
Replace `your-tenant-name` with your Azure AD B2C tenant name.
8. Select **Register**
9. Record the **Application (client) ID** shown on the application Overview page. You need this when you configure the identity provider in the next section.
10. Select **Certificates & secrets**
11. Click **New client secret**
12. Enter a **Description** for the secret, for example *Application password 1*, and then click **Add**.
13. Record the application password shown in the **Value** column. You need this when you configure the identity provider in the next section.

Configure a Microsoft account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Microsoft Account**.

5. Enter a **Name**. For example, *MSA*.
6. For the **Client ID**, enter the Application (client) ID of the Azure AD application that you created earlier.
7. For the **Client secret**, enter the client secret that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a Facebook account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create a Facebook application

To use a Facebook account as an [identity provider](#) in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a Facebook account, you can sign up at <https://www.facebook.com/>.

1. Sign in to [Facebook for developers](#) with your Facebook account credentials.
2. If you have not already done so, you need to register as a Facebook developer. To do this, select **Get Started** on the upper-right corner of the page, accept Facebook's policies, and complete the registration steps.
3. Select **My Apps** and then **Create App**.
4. Enter a **Display Name** and a valid **Contact Email**.
5. Select **Create App ID**. This may require you to accept Facebook platform policies and complete an online security check.
6. Select **Settings > Basic**.
7. Choose a **Category**, for example `Business and Pages`. This value is required by Facebook, but not used for Azure AD B2C.
8. At the bottom of the page, select **Add Platform**, and then select **Website**.
9. In **Site URL**, enter `https://your-tenant-name.b2clogin.com/` replacing `your-tenant-name` with the name of your tenant. Enter a URL for the **Privacy Policy URL**, for example `http://www.contoso.com`. The policy URL is a page you maintain to provide privacy information for your application.
10. Select **Save Changes**.
11. At the top of the page, copy the value of **App ID**.
12. Select **Show** and copy the value of **App Secret**. You use both of them to configure Facebook as an identity provider in your tenant. **App Secret** is an important security credential.
13. Select the plus sign next to **PRODUCTS**, and then select **Set up** under **Facebook Login**.
14. Under **Facebook Login**, select **Settings**.
15. In **Valid OAuth redirect URIs**, enter
`https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp`. Replace `your-tenant-name` with the name of your tenant. Select **Save Changes** at the bottom of the page.
16. To make your Facebook application available to Azure AD B2C, select the Status selector at the top right of the page and turn it **On** to make the Application public, and then select **Switch Mode**. At this point the Status should change from **Development** to **Live**.

Configure a Facebook account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Facebook**.
5. Enter a **Name**. For example, *Facebook*.
6. For the **Client ID**, enter the App ID of the Facebook application that you created earlier.

7. For the **Client secret**, enter the App Secret that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a GitHub account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

Create a GitHub OAuth application

To use a GitHub account as an **identity provider** in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a GitHub account, you can sign up at <https://www.github.com/>.

1. Sign in to the [GitHub Developer](#) website with your GitHub credentials.
2. Select **OAuth Apps** and then select **New OAuth App**.
3. Enter an **Application name** and your **Homepage URL**.
4. Enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **Authorization callback URL**. Replace `your-tenant-name` with the name of your Azure AD B2C tenant. Use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
5. Click **Register application**.
6. Copy the values of **Client ID** and **Client Secret**. You need both to add the identity provider to your tenant.

Configure a GitHub account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **GitHub (Preview)**.
5. Enter a **Name**. For example, *GitHub*.
6. For the **Client ID**, enter the Client ID of the GitHub application that you created earlier.
7. For the **Client secret**, enter the Client Secret that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a Google account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create a Google application

To use a Google account as an [identity provider](#) in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your Google Developers Console. If you don't already have a Google account you can sign up at <https://accounts.google.com/SignUp>.

1. Sign in to the [Google Developers Console](#) with your Google account credentials.
2. In the upper-left corner of the page, select the project list, and then select **New Project**.
3. Enter a **Project Name**, select **Create**.
4. Make sure you are using the new project by selecting the project drop-down in the top-left of the screen, select your project by name, then select **Open**.
5. Select **OAuth consent screen** in the left menu, select **External**, and then select **Create**. Enter a **Name** for your application. Enter *b2clogin.com* in the **Authorized domains** section and select **Save**.
6. Select **Credentials** in the left menu, and then select **Create credentials > Oauth client ID**.
7. Under **Application type**, select **Web application**.
8. Enter a **Name** for your application, enter `https://your-tenant-name.b2clogin.com` in **Authorized JavaScript origins**, and `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **Authorized redirect URIs**. Replace `your-tenant-name` with the name of your tenant. You need to use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
9. Click **Create**.
10. Copy the values of **Client ID** and **Client secret**. You will need both of them to configure Google as an identity provider in your tenant. **Client secret** is an important security credential.

Configure a Google account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Google**.
5. Enter a **Name**. For example, *Google*.
6. For the **Client ID**, enter the Client ID of the Google application that you created earlier.
7. For the **Client secret**, enter the Client Secret that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a LinkedIn account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create a LinkedIn application

To use a LinkedIn account as an [identity provider](#) in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a LinkedIn account, you can sign up at <https://www.linkedin.com/>.

1. Sign in to the [LinkedIn Developers website](#) with your LinkedIn account credentials.
2. Select **My Apps**, and then click **Create Application**.
3. Enter **Company Name**, **Application Name**, **Application Description**, **Application Logo**, **Application Use**, **Website URL**, **Business Email**, and **Business Phone**.
4. Agree to the [LinkedIn API Terms of Use](#) and click **Submit**.
5. Copy the values of **Client ID** and **Client Secret**. You can find them under **Authentication Keys**. You will need both of them to configure LinkedIn as an identity provider in your tenant. **Client Secret** is an important security credential.
6. Enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **Authorized Redirect URLs**. Replace `your-tenant-name` with the name of your tenant. You need to use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C. Select **Add**, and then click **Update**.

Configure a LinkedIn account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **LinkedIn**.
5. Enter a **Name**. For example, *LinkedIn*.
6. For the **Client ID**, enter the Client ID of the LinkedIn application that you created earlier.
7. For the **Client secret**, enter the Client Secret that you recorded.
8. Select **Save**.

Migration from v1.0 to v2.0

LinkedIn recently [updated their APIs from v1.0 to v2.0](#). As part of the migration, Azure AD B2C is only able to obtain the full name of the LinkedIn user during the sign-up. If an email address is one of the attributes that is collected during sign-up, the user must manually enter the email address and validate it.

Set up sign-up and sign-in with a QQ account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

Create a QQ application

To use a QQ account as an identity provider in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a QQ account, you can sign up at <https://ssl.zc.qq.com/en/index.html?type=1&ptlang=1033>.

Register for the QQ developer program

1. Sign in to the [QQ developer portal](#) with your QQ account credentials.
2. After signing in, go to <https://open.qq.com/reg> to register yourself as a developer.
3. Select **个人** (individual developer).
4. Enter the required information and select **下一步** (next step).
5. Complete the email verification process. You will need to wait a few days to be approved after registering as a developer.

Register a QQ application

1. Go to <https://connect.qq.com/index.html>.
2. Select **应用管理** (app management).
3. Select **创建应用** (create app) and enter the required information.
4. Enter `https://your-tenant-name.b2clogin.com/your-tenant-name}.onmicrosoft.com/oauth2/authresp` in **授权回调域** (callback URL). For example, if your `tenant_name` is contoso, set the URL to be `https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp`.
5. Select **创建应用** (create app).
6. On the confirmation page, select **应用管理** (app management) to return to the app management page.
7. Select **查看** (view) next to the app you created.
8. Select **修改** (edit).
9. Copy the **APP ID** and **APP KEY**. You need both of these values to add the identity provider to your tenant.

Configure QQ as an identity provider

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **QQ (Preview)**.
5. Enter a **Name**. For example, **QQ**.
6. For the **Client ID**, enter the APP ID of the QQ application that you created earlier.
7. For the **Client secret**, enter the APP KEY that you recorded.

8. Select **Save**.

Set up sign-up and sign-in with a Twitter account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Create an application

To use Twitter as an identity provider in Azure AD B2C, you need to create a Twitter application. If you don't already have a Twitter account, you can sign up at <https://twitter.com/signup>.

1. Sign in to the [Twitter Developers](#) website with your Twitter account credentials.
2. Select **Create an app**.
3. Enter an **App name** and an **Application description**.
4. In **Website URL**, enter `https://your-tenant.b2clogin.com`. Replace `your-tenant` with the name of your tenant.
For example, <https://contosob2c.b2clogin.com>.
5. For the **Callback URL**, enter
`https://your-tenant.b2clogin.com/your-tenant.onmicrosoft.com/your-user-flow-Id/oauth1/authresp`. Replace `your-tenant` with the name of your tenant name and `your-user-flow-Id` with the identifier of your user flow.
For example, `b2c_1A_signup_signin_twitter`. You need to use all lowercase letters when entering your tenant name and user flow id even if they are defined with uppercase letters in Azure AD B2C.
6. At the bottom of the page, read and accept the terms, and then select **Create**.
7. On the **App details** page, select **Edit > Edit details**, check the box for **Enable Sign in with Twitter**, and then select **Save**.
8. Select **Keys and tokens** and record the **Consumer API Key** and the **Consumer API secret key** values to be used later.

Configure Twitter as an identity provider in your tenant

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Twitter**.
5. Enter a **Name**. For example, *Twitter*.
6. For the **Client ID**, enter the Consumer API Key of the Twitter application that you created earlier.
7. For the **Client secret**, enter the Consumer API secret key that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a WeChat account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

Create a WeChat application

To use a WeChat account as an identity provider in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a WeChat account, you can get information at <https://kf.qq.com/faq/161220Brem2Q161220uUjERB.html>.

Register a WeChat application

1. Sign in to <https://open.weixin.qq.com/> with your WeChat credentials.
2. Select **管理中心** (management center).
3. Follow the steps to register a new application.
4. Enter `https://your-tenant_name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **授权回调域** (callback URL). For example, if your tenant name is contoso, set the URL to be `https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp`.
5. Copy the **APP ID** and **APP KEY**. You will need these to add the identity provider to your tenant.

Configure WeChat as an identity provider in your tenant

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **WeChat (Preview)**.
5. Enter a **Name**. For example, *WeChat*.
6. For the **Client ID**, enter the APP ID of the WeChat application that you created earlier.
7. For the **Client secret**, enter the APP KEY that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with a Weibo account using Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

Create a Weibo application

To use a Weibo account as an identity provider in Azure Active Directory B2C (Azure AD B2C), you need to create an application in your tenant that represents it. If you don't already have a Weibo account, you can sign up at <https://weibo.com/signup/signup.php?lang=en-us>.

1. Sign in to the [Weibo developer portal](#) with your Weibo account credentials.
2. After signing in, select your display name in the top-right corner.
3. In the dropdown, select **编辑开发者信息** (edit developer information).
4. Enter the required information and select **提交** (submit).
5. Complete the email verification process.
6. Go to the [identity verification page](#).
7. Enter the required information and select **提交** (submit).

Register a Weibo application

1. Go to the [new Weibo app registration page](#).
2. Enter the necessary application information.
3. Select **创建** (create).
4. Copy the values of **App Key** and **App Secret**. You need both of these to add the identity provider to your tenant.
5. Upload the required photos and enter the necessary information.
6. Select **保存以上信息** (save).
7. Select **高级信息** (advanced information).
8. Select **编辑** (edit) next to the field for OAuth2.0 **授权设置** (redirect URL).
9. Enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` for OAuth2.0 **授权设置** (redirect URL). For example, if your tenant name is contoso, set the URL to be `https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp`.
10. Select **提交** (submit).

Configure a Weibo account as an identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, then select **Weibo (Preview)**.
5. Enter a **Name**. For example, *Weibo*.

6. For the **Client ID**, enter the App Key of the Weibo application that you created earlier.
7. For the **Client secret**, enter the App Secret that you recorded.
8. Select **Save**.

Set up sign-up and sign-in with OpenID Connect using Azure Active Directory B2C

1/28/2020 • 3 minutes to read • [Edit Online](#)

OpenID Connect is an authentication protocol built on top of OAuth 2.0 that can be used for secure user sign-in. Most identity providers that use this protocol are supported in Azure AD B2C. This article explains how you can add custom OpenID Connect identity providers into your user flows.

Add the identity provider

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by clicking the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **Identity providers**, and then select **New OpenID Connect provider**.

Configure the identity provider

Every OpenID Connect identity provider describes a metadata document that contains most of the information required to perform sign-in. This includes information such as the URLs to use and the location of the service's public signing keys. The OpenID Connect metadata document is always located at an endpoint that ends in `.well-known\openid-configuration`. For the OpenID Connect identity provider you are looking to add, enter its metadata URL.

Client ID and secret

To allow users to sign in, the identity provider requires developers to register an application in their service. This application has an ID that is referred to as the **client ID** and a **client secret**. Copy these values from the identity provider and enter them into the corresponding fields.

NOTE

The client secret is optional. However, you must enter a client secret if you'd like to use the [authorization code flow](#), which uses the secret to exchange the code for the token.

Scope

Scope defines the information and permissions you are looking to gather from your custom identity provider. OpenID Connect requests must contain the `openid` scope value in order to receive the ID token from the identity provider. Without the ID token, users are not able to sign in to Azure AD B2C using the custom identity provider. Other scopes can be appended separated by space. Refer to the custom identity provider's documentation to see what other scopes may be available.

Response type

The response type describes what kind of information is sent back in the initial call to the `authorization_endpoint` of the custom identity provider. The following response types can be used:

- `code` : As per the [authorization code flow](#), a code will be returned back to Azure AD B2C. Azure AD B2C proceeds to call the `token_endpoint` to exchange the code for the token.
- `id_token` : An ID token is returned back to Azure AD B2C from the custom identity provider.

Response mode

The response mode defines the method that should be used to send the data back from the custom identity provider to Azure AD B2C. The following response modes can be used:

- `form_post` : This response mode is recommended for best security. The response is transmitted via the HTTP `POST` method, with the code or token being encoded in the body using the `application/x-www-form-urlencoded` format.
- `query` : The code or token is returned as a query parameter.

Domain hint

The domain hint can be used to skip directly to the sign in page of the specified identity provider, instead of having the user make a selection among the list of available identity providers. To allow this kind of behavior, enter a value for the domain hint. To jump to the custom identity provider, append the parameter

`domain_hint=<domain hint value>` to the end of your request when calling Azure AD B2C for sign in.

Claims mapping

After the custom identity provider sends an ID token back to Azure AD B2C, Azure AD B2C needs to be able to map the claims from the received token to the claims that Azure AD B2C recognizes and uses. For each of the following mappings, refer to the documentation of the custom identity provider to understand the claims that are returned back in the identity provider's tokens:

- **User ID**: Enter the claim that provides the *unique identifier* for the signed-in user.
- **Display Name**: Enter the claim that provides the *display name* or *full name* for the user.
- **Given Name**: Enter the claim that provides the *first name* of the user.
- **Surname**: Enter the claim that provides the *last name* of the user.
- **Email**: Enter the claim that provides the *email address* of the user.

Configure tokens in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to configure the [lifetime and compatibility of a token](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

Create a user flow to enable users to sign up and sign in to your application.

Configure token lifetime

You can configure the token lifetime on any user flow.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your Azure AD B2C tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **User flows (policies)**.
5. Open the user flow that you previously created.
6. Select **Properties**.
7. Under **Token lifetime**, adjust the following properties to fit the needs of your application:

Token lifetime	
Access & ID token lifetimes (minutes)	60
Refresh token lifetime (days)	14
Refresh token sliding window lifetime	<input checked="" type="radio"/> Bounded <input type="radio"/> No expiry
Lifetime length (days)	90

8. Click **Save**.

Configure token compatibility

1. Select **User flows (policies)**.
2. Open the user flow that you previously created.
3. Select **Properties**.
4. Under **Token compatibility settings**, adjust the following properties to fit the needs of your application:

^ Token compatibility settings

Issuer (iss) claim ⓘ	<input type="text" value="https://<domain>/c64a4f7d-3091-4c73-a722-a3f0694f60b7/v2.0/"/>	▼
Subject (sub) claim ⓘ	<input checked="" type="checkbox"/> ObjectID <input type="checkbox"/> Not supported	
Claim representing user flow ⓘ	<input checked="" type="checkbox"/> tfp <input type="checkbox"/> acr	

5. Click **Save**.

Next steps

Learn more about how to [use access tokens](#).

Configure session behavior in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

This feature gives you fine-grained control, on a [per-user flow basis](#), of:

- Lifetimes of web application sessions managed by Azure AD B2C.
- Single sign-on (SSO) behavior across multiple apps and user flows in your Azure AD B2C tenant.

These settings are not available for password reset user flows.

Azure AD B2C supports the [OpenID Connect authentication protocol](#) for enabling secure sign-in to web applications. You can use the following properties to manage web application sessions:

Session behavior properties

- **Web app session lifetime (minutes)** - The lifetime of Azure AD B2C's session cookie stored on the user's browser upon successful authentication.
 - Default = 1440 minutes.
 - Minimum (inclusive) = 15 minutes.
 - Maximum (inclusive) = 1440 minutes.
- **Web app session timeout** - If this switch is set to **Absolute**, the user is forced to authenticate again after the time period specified by **Web app session lifetime (minutes)** elapses. If this switch is set to **Rolling** (the default setting), the user remains signed in as long as the user is continually active in your web application.
- **Single sign-on configuration** If you have multiple applications and user flows in your B2C tenant, you can manage user interactions across them using the **Single sign-on configuration** property. You can set the property to one of the following settings:
 - **Tenant** - This setting is the default. Using this setting allows multiple applications and user flows in your B2C tenant to share the same user session. For example, once a user signs into an application, the user can also seamlessly sign into another one, Contoso Pharmacy, upon accessing it.
 - **Application** - This setting allows you to maintain a user session exclusively for an application, independent of other applications. For example, if you want the user to sign in to Contoso Pharmacy (with the same credentials), even if the user is already signed into Contoso Shopping, another application on the same B2C tenant.
 - **Policy** - This setting allows you to maintain a user session exclusively for a user flow, independent of the applications using it. For example, if the user has already signed in and completed a multi factor authentication (MFA) step, the user can be given access to higher-security parts of multiple applications as long as the session tied to the user flow doesn't expire.
 - **Disabled** - This setting forces the user to run through the entire user flow on every execution of the policy.

The following use cases are enabled using these properties:

- Meet your industry's security and compliance requirements by setting the appropriate web application session lifetimes.
- Force authentication after a set time period during a user's interaction with a high-security part of your web application.

Configure the properties

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your Azure AD B2C tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **User flows (policies)**.
5. Open the user flow that you previously created.
6. Select **Properties**.
7. Configure **Web app session lifetime (minutes)**, **Web app session timeout**, **Single sign-on configuration**, and **Require ID Token in logout requests** as needed.

Session behavior

Web app session lifetime (minutes) <small>i</small>	1440
Web app session timeout <small>i</small>	<input checked="" type="radio"/> Absolute <input type="radio"/> Rolling
Single sign-on configuration <small>i</small>	<input checked="" type="radio"/> Tenant <input type="radio"/> Application <input type="radio"/> Policy <input type="radio"/> Disabled
Require ID Token in logout requests <small>i</small>	<input type="radio"/> No <input checked="" type="radio"/> Yes

8. Click **Save**.

Enable Age Gating in Azure Active Directory B2C

1/28/2020 • 3 minutes to read • [Edit Online](#)

IMPORTANT

This feature is in public preview. Do not use feature for production applications.

Age gating in Azure Active Directory B2C (Azure AD B2C) enables you to identify minors that want to use your application. You can choose to block the minor from signing into the application. Users can also go back to the application and identify their age group and their parental consent status. Azure AD B2C can block minors without parental consent. Azure AD B2C can also be set up to allow the application to decide what to do with minors.

After you enable age gating in your [user flow](#), users are asked when they were born and what country/region they live in. If a user signs in that hasn't previously entered the information, they'll need to enter it the next time they sign in. The rules are applied every time a user signs in.

Azure AD B2C uses the information that the user enters to identify whether they're a minor. The **ageGroup** field is then updated in their account. The value can be `null`, `Undefined`, `Minor`, `Adult`, and `NotAdult`. The **ageGroup** and **consentProvidedForMinor** fields are then used to calculate the value of **legalAgeGroupClassification**.

Age gating involves two age values: the age that someone is no longer considered a minor, and the age at which a minor must have parental consent. The following table lists the age rules that are used for defining a minor and a minor requiring consent.

COUNTRY/REGION	COUNTRY/REGION NAME	MINOR CONSENT AGE	MINOR AGE
Default	None	None	18
AE	United Arab Emirates	None	21
AT	Austria	14	18
BE	Belgium	14	18
BG	Bulgaria	16	18
BH	Bahrain	None	21
CM	Cameroon	None	21
CY	Cyprus	16	18
CZ	Czech Republic	16	18
DE	Germany	16	18
DK	Denmark	16	18
EE	Estonia	16	18

COUNTRY/REGION	COUNTRY/REGION NAME	MINOR CONSENT AGE	MINOR AGE
EG	Egypt	None	21
ES	Spain	13	18
FR	France	16	18
GB	United Kingdom	13	18
GR	Greece	16	18
HR	Croatia	16	18
HU	Hungary	16	18
IE	Ireland	13	18
IT	Italy	16	18
KR	Korea, Republic of	14	18
LT	Lithuania	16	18
LU	Luxembourg	16	18
LV	Latvia	16	18
MT	Malta	16	18
NA	Namibia	None	21
NL	Netherlands	16	18
PL	Poland	13	18
PT	Portugal	16	18
RO	Romania	16	18
SE	Sweden	13	18
SG	Singapore	None	21
SI	Slovenia	16	18
SK	Slovakia	16	18
TD	Chad	None	21
TH	Thailand	None	20

COUNTRY/REGION	COUNTRY/REGION NAME	MINOR CONSENT AGE	MINOR AGE
TW	Taiwan	None	20
US	United States	13	18

Age gating options

Allowing minors without parental consent

For user flows that allow either sign-up, sign-in, or both, you can choose to allow minors without consent into your application. Minors without parental consent are allowed to sign in or sign up as normal and Azure AD B2C issues an ID token with the **legalAgeGroupClassification** claim. This claim defines the experience that users have, such as collecting parental consent and updating the **consentProvidedForMinor** field.

Blocking minors without parental consent

For user flows that allow either sign-up, sign-in or both, you can choose to block minors without consent from the application. The following options are available for handling blocked users in Azure AD B2C:

- Send a JSON back to the application - this option sends a response back to the application that a minor was blocked.
- Show an error page - the user is shown a page informing them that they can't access the application.

Set up your tenant for age gating

To use age gating in a user flow, you need to configure your tenant to have additional properties.

1. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu. Select the directory that contains your tenant.
2. Select **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
3. Select **Properties** for your tenant in the menu on the left.
4. Under the **Age gating** section, click on **Configure**.
5. Wait for the operation to complete and your tenant will be set up for age gating.

Enable age gating in your user flow

After your tenant is set up to use age gating, you can then use this feature in [user flows](#) where it's enabled. You enable age gating with the following steps:

1. Create a user flow that has age gating enabled.
2. After you create the user flow, select **Properties** in the menu.
3. In the **Age gating** section, select **Enabled**.
4. You then decide how you want to manage users that identify as minors. For **Sign-up or sign-in**, you select **Allow minors to access your application** or **Block minors from accessing your application**. If blocking minors is selected, you select **Send a JSON back to the application** or **Show an error message**.

Define custom attributes in Azure Active Directory B2C

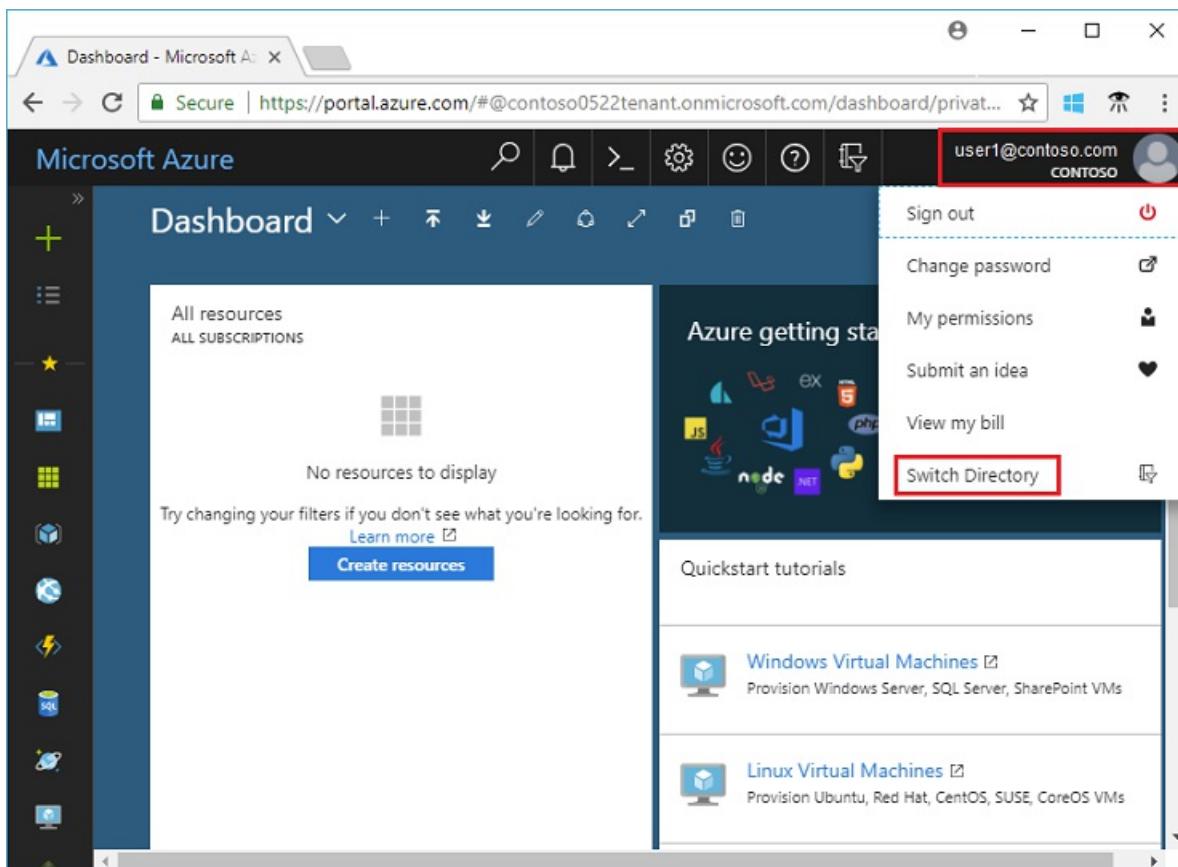
2/20/2020 • 2 minutes to read • [Edit Online](#)

Every customer-facing application has unique requirements for the information that needs to be collected. Your Azure Active Directory B2C (Azure AD B2C) tenant comes with a built-in set of information stored in attributes, such as Given Name, Surname, City, and Postal Code. With Azure AD B2C, you can extend the set of attributes stored on each customer account.

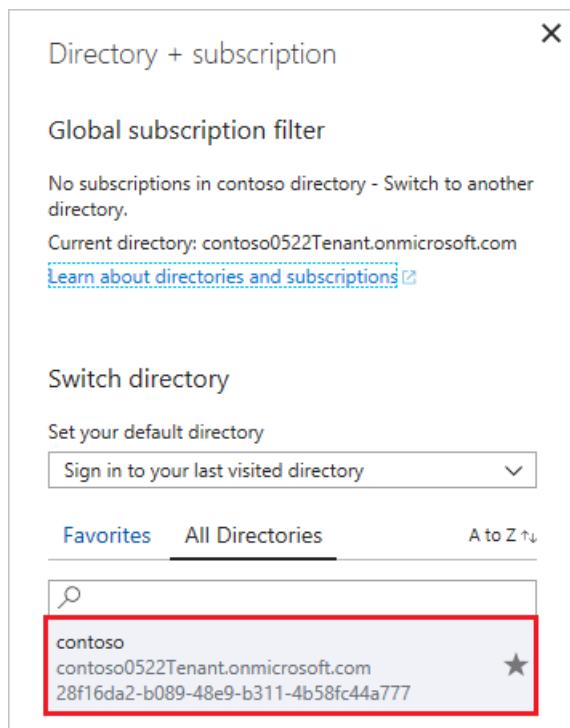
You can create custom attributes in the [Azure portal](#) and use them in your sign-up user flows, sign-up or sign-in user flows, or profile editing user flows. You can also read and write these attributes by using the [Microsoft Graph API](#).

Create a custom attribute

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant by switching to it in the top-right corner of the Azure portal. Select your subscription information, and then select **Switch Directory**.



Choose the directory that contains your tenant.



3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **User attributes**, and then select **Add**.
5. Provide a **Name** for the custom attribute (for example, "ShoeSize")
6. Choose a **Data Type**. Only **String**, **Boolean**, and **Int** are available.
7. Optionally, enter a **Description** for informational purposes.
8. Click **Create**.

The custom attribute is now available in the list of **User attributes** and for use in your user flows. A custom attribute is only created the first time it is used in any user flow, and not when you add it to the list of **User attributes**.

Use a custom attribute in your user flow

1. In your Azure AD B2C tenant, select **User flows**.
2. Select your policy (for example, "B2C_1_SignupSignin") to open it.
3. Select **User attributes** and then select the custom attribute (for example, "ShoeSize"). Click **Save**.
4. Select **Application claims** and then select the custom attribute.
5. Click **Save**.

Once you've created a new user using a user flow which uses the newly created custom attribute, the object can be queried in [Microsoft Graph Explorer](#). Alternatively you can use the [Run user flow](#) feature on the user flow to verify the customer experience. You should now see **ShoeSize** in the list of attributes collected during the sign-up journey, and see it in the token sent back to your application.

Pass an access token through a user flow to your application in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

A [user flow](#) in Azure Active Directory B2C (Azure AD B2C) provides users of your application an opportunity to sign up or sign in with an identity provider. When the journey starts, Azure AD B2C receives an [access token](#) from the identity provider. Azure AD B2C uses that token to retrieve information about the user. You enable a claim in your user flow to pass the token through to the applications that you register in Azure AD B2C.

Azure AD B2C currently only supports passing the access token of [OAuth 2.0](#) identity providers, which include [Facebook](#) and [Google](#). For all other identity providers, the claim is returned blank.

Prerequisites

- Your application must be using a [v2 user flow](#).
- Your user flow is configured with an OAuth 2.0 identity provider.

Enable the claim

1. Sign in to the [Azure portal](#) as the global administrator of your Azure AD B2C tenant.
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, search for and select **Azure AD B2C**.
4. Select **User flows (policies)**, and then select your user flow. For example, **B2C_1_signupsignin1**.
5. Select **Application claims**.
6. Enable the **Identity Provider Access Token** claim.

The screenshot shows the 'Application claims' section of the Azure AD B2C configuration. On the left, there's a sidebar with 'Overview', 'Settings', 'Properties', 'Identity providers', 'User attributes', and 'Application claims' (which is highlighted with a red box). The main area has a note about user attributes and claims. A table lists various claims with their names, data types, and descriptions. The 'Identity Provider Access Token' claim is checked and highlighted with a red box, indicating it's the target for enabling the access token.

NAME	DATA TYPE	DESCRIPTION
<input type="checkbox"/> City	String	The city in which the user is located.
<input type="checkbox"/> Country of Residence	String	The country in which the user legally resides.
<input type="checkbox"/> Country/Region	String	The country/region in which the user is located.
<input type="checkbox"/> Date of Birth	DateTime	The user's date of birth.
<input type="checkbox"/> Display Name	String	Display Name of the User.
<input type="checkbox"/> Email Addresses	StringCollection	Email addresses of the user.
<input type="checkbox"/> Given Name	String	The user's given name (also known as first name).
<input checked="" type="checkbox"/> Identity Provider	String	The social identity provider used by the user to a...
<input checked="" type="checkbox"/> Identity Provider Access Token	String	The access token returned by the identity provid...
<input type="checkbox"/> Job Title	String	The user's job title.

7. Click **Save** to save the user flow.

Test the user flow

When testing your applications in Azure AD B2C, it can be useful to have the Azure AD B2C token returned to <https://jwt.ms> to review the claims in it.

1. On the Overview page of the user flow, select **Run user flow**.
2. For **Application**, select your application that you previously registered. To see the token in the example below, the **Reply URL** should show `https://jwt.ms`.
3. Click **Run user flow**, and then sign in with your account credentials. You should see the access token of the identity provider in the **idp_access_token** claim.

You should see something similar to the following example:

Decoded Token	Claims
{ "typ": "JWT", "alg": "RS256", "kid": "X5eXk4xyojNFum1kl2Ytv8d1NP4-c57d06QGTVBwaNk" }.{ "exp": 1543274600, "nbf": 1543271000, "ver": "1.0", "iss": "https://contoso0926tenant.b2clogin.com/c64a4f7-a3f0694f60b7/v2.0/", "sub": "10bd2040-5faede13b843", "aud": "327fa24a-70c0b5892198", "nonce": "defaultNonce", "iat": 1543271000, "auth_time": 1543271000, "idp_access_token": "EAAM7yCmyOnwBAH3nrI3ZCyIfaIHu6PkE5V9Tb20KjitXX99TNAHczaJjqcDsRNiLZCc10hZCYPd6BU1vQZAwtvCnPX59xeMhZCmfeKS7pxCxR5PZAsekxI6Xa3mdIZCqW617QS6UgZDZD", "idp": "facebook.com", "tfp": "B2C_1_signupsignin3" }.[Signature]	

Next steps

Learn more in the [overview of Azure AD B2C tokens](#).

Get started with custom policies in Azure Active Directory B2C

2/10/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

[Custom policies](#) are configuration files that define the behavior of your Azure Active Directory B2C (Azure AD B2C) tenant. In this article, you create a custom policy that supports local account sign-up or sign-in by using an email address and password. You also prepare your environment for adding identity providers.

Prerequisites

- If you don't have one already, [create an Azure AD B2C tenant](#) that is linked to your Azure subscription.
- [Register your application](#) in the tenant that you created so that it can communicate with Azure AD B2C.
- Complete the steps in [Set up sign-up and sign-in with a Facebook account](#) to configure a Facebook application.

Add signing and encryption keys

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. In the Azure portal, search for and select **Azure AD B2C**.
4. On the overview page, under **Policies**, select **Identity Experience Framework**.

Create the signing key

1. Select **Policy Keys** and then select **Add**.
2. For **Options**, choose **Generate**.
3. In **Name**, enter `TokenSigningKeyContainer`. The prefix `B2C_1A_` might be added automatically.
4. For **Key type**, select **RSA**.
5. For **Key usage**, select **Signature**.
6. Select **Create**.

Create the encryption key

1. Select **Policy Keys** and then select **Add**.
2. For **Options**, choose **Generate**.
3. In **Name**, enter `TokenEncryptionKeyContainer`. The prefix `B2C_1A_` might be added automatically.
4. For **Key type**, select **RSA**.
5. For **Key usage**, select **Encryption**.
6. Select **Create**.

Create the Facebook key

Add your Facebook application's **App Secret** as a policy key. You can use the App Secret of the application you created as part of this article's prerequisites.

1. Select **Policy Keys** and then select **Add**.
2. For **Options**, choose **Manual**.
3. For **Name**, enter `FacebookSecret`. The prefix `B2C_1A_` might be added automatically.
4. In **Secret**, enter your Facebook application's *App Secret* from developers.facebook.com. This value is the secret, not the application ID.
5. For **Key usage**, select **Signature**.
6. Select **Create**.

Register Identity Experience Framework applications

Azure AD B2C requires you to register two applications that it uses to sign up and sign in users with local accounts: *IdentityExperienceFramework*, a web API, and *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.

You need to register these two applications in your Azure AD B2C tenant only once.

Register the *IdentityExperienceFramework* application

To register an application in your Azure AD B2C tenant, you can use the **App registrations (Legacy)** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. In the Azure portal, search for and select **Azure Active Directory**.
 3. In the **Azure Active Directory** overview menu, under **Manage**, select **App registrations (Legacy)**.
 4. Select **New application registration**.
 5. For **Name**, enter `IdentityExperienceFramework`.
 6. For **Application type**, choose **Web app/API**.
 7. For **Sign-on URL**, enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com`, where `your-tenant-name` is your Azure AD B2C tenant domain name. All URLs should now be using b2clogin.com.
 8. Select **Create**. After it's created, copy the application ID and save it to use later.

Register the *ProxyIdentityExperienceFramework* application

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. In **App registrations (Legacy)**, select **New application registration**.
 2. For **Name**, enter `ProxyIdentityExperienceFramework`.
 3. For **Application type**, choose **Native**.
 4. For **Redirect URI**, enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com`, where `your-tenant-name` is your Azure AD B2C tenant.
 5. Select **Create**. After it's created, copy the application ID and save it to use later.

6. Select **Settings**, then select **Required permissions**, and then select **Add**.
7. Choose **Select an API**, search for and select **IdentityExperienceFramework**, and then click **Select**.
8. Select the check box next to **Access IdentityExperienceFramework**, click **Select**, and then click **Done**.
9. Select **Grant permissions**, and then confirm by selecting **Yes**.

Custom policy starter pack

Custom policies are a set of XML files you upload to your Azure AD B2C tenant to define technical profiles and user journeys. We provide starter packs with several pre-built policies to get you going quickly. Each of these starter packs contains the smallest number of technical profiles and user journeys needed to achieve the scenarios described:

- **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 options.

Each starter pack contains:

- **Base file** - Few modifications are required to the base. Example: *TrustFrameworkBase.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*

In this article, you edit the XML custom policy files in the **SocialAndLocalAccounts** starter pack. If you need an XML editor, try [Visual Studio Code](#), a lightweight cross-platform editor.

Get the starter pack

Get the custom policy starter packs from GitHub, then update the XML files in the SocialAndLocalAccounts starter pack with your Azure AD B2C tenant name.

1. [Download the .zip file](#) or clone the repository:

```
git clone https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack
```

2. In all of the files in the **SocialAndLocalAccounts** directory, replace the string `yourtenant` with the name of your Azure AD B2C tenant.

For example, if the name of your B2C tenant is *contosotenant*, all instances of `yourtenant.onmicrosoft.com` become `contosotenant.onmicrosoft.com`.

Add application IDs to the custom policy

Add the application IDs to the extensions file *TrustFrameworkExtensions.xml*.

1. Open `SocialAndLocalAccounts/ TrustFrameworkExtensions.xml` and find the element `<TechnicalProfile Id="login-NonInteractive">`.
2. Replace both instances of `IdentityExperienceFrameworkAppId` with the application ID of the IdentityExperienceFramework application that you created earlier.
3. Replace both instances of `ProxyIdentityExperienceFrameworkAppId` with the application ID of the ProxyIdentityExperienceFramework application that you created earlier.

4. Save the file.

Upload the policies

1. Select the **Identity Experience Framework** menu item in your B2C tenant in the Azure portal.
2. Select **Upload custom policy**.
3. In this order, upload the policy files:
 - a. *TrustFrameworkBase.xml*
 - b. *TrustFrameworkExtensions.xml*
 - c. *SignUpOrSignin.xml*
 - d. *ProfileEdit.xml*
 - e. *PasswordReset.xml*

As you upload the files, Azure adds the prefix `B2C_1A_` to each.

TIP

If your XML editor supports validation, validate the files against the `TrustFrameworkPolicy_0.3.0.0.xsd` XML schema that is located in the root directory of the starter pack. XML schema validation identifies errors before uploading.

Test the custom policy

1. Under **Custom policies**, select **B2C_1A_signup_signin**.
2. For **Select application** on the overview page of the custom policy, select the web application named *webapp1* that you previously registered.
3. Make sure that the **Reply URL** is `https://jwt.ms`.
4. Select **Run now**.
5. Sign up using an email address.
6. Select **Run now** again.
7. Sign in with the same account to confirm that you have the correct configuration.

Add Facebook as an identity provider

1. In the `SocialAndLocalAccounts/ TrustFrameworkExtensions.xml` file, replace the value of `client_id` with the Facebook application ID:

```
<TechnicalProfile Id="Facebook-OAUTH">
  <Metadata>
    <!--Replace the value of client_id in this technical profile with the Facebook app ID-->
    <Item Key="client_id">00000000000000</Item>
```

2. Upload the *TrustFrameworkExtensions.xml* file to your tenant.
3. Under **Custom policies**, select **B2C_1A_signup_signin**.
4. Select **Run now** and select Facebook to sign in with Facebook and test the custom policy.

Next steps

Next, try adding Azure Active Directory (Azure AD) as an identity provider. The base file used in this getting started guide already contains some of the content that you need for adding other identity

providers like Azure AD.

For information about setting up Azure AD as and identity provider, see [Set up sign-up and sign-in with an Azure Active Directory account using Active Directory B2C custom policies](#).

Configure the resource owner password credentials flow in Azure Active Directory B2C using a custom policy

2/28/2020 • 8 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

In Azure Active Directory B2C (Azure AD B2C), the resource owner password credentials (ROPC) flow is an OAuth standard authentication flow. In this flow, an application, also known as the relying party, exchanges valid credentials for tokens. The credentials include a user ID and password. The tokens returned are an ID token, access token, and a refresh token.

ROPC flow notes

In Azure Active Directory B2C (Azure AD B2C), the following options are supported:

- **Native Client:** User interaction during authentication happens when code runs on a user-side device. The device can be a mobile application that's running in a native operating system, such as Android and iOS.
- **Public client flow:** Only user credentials, gathered by an application, are sent in the API call. The credentials of the application are not sent.
- **Add new claims:** The ID token contents can be changed to add new claims.

The following flows are not supported:

- **Server-to-server:** The identity protection system needs a reliable IP address gathered from the caller (the native client) as part of the interaction. In a server-side API call, only the server's IP address is used. If a dynamic threshold of failed authentications is exceeded, the identity protection system may identify a repeated IP address as an attacker.
- **Confidential client flow:** The application client ID is validated, but the application secret is not validated.

When using the ROPC flow, consider the following:

- ROPC doesn't work when there is any interruption to the authentication flow that needs user interaction. For example, when a password has expired or needs to be changed, multi-factor authentication is required, or when more information needs to be collected during sign-in (for example, user consent).
- ROPC supports local accounts only. Users can't sign in with federated identity providers like Microsoft, Google+, Twitter, AD-FS, or Facebook.
- Session Management, including keep me signed-in (KMSI), is not applicable.

Prerequisites

Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).

Register an application

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our

new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *ROPC_Auth_app*.
6. For **Native client**, select **Yes**.
7. Leave the other values as they are, and then select **Create**.
8. Record the **APPLICATION ID** for use in a later step.

Create a resource owner policy

1. Open the *TrustFrameworkExtensions.xml* file.
2. If it doesn't exist already, add a **ClaimsSchema** element and its child elements as the first element under the **BuildingBlocks** element:

```
<ClaimsSchema>
  <ClaimType Id="logonIdentifier">
    <DisplayName>User name or email address that the user can use to sign in</DisplayName>
    <DataType>string</DataType>
  </ClaimType>
  <ClaimType Id="resource">
    <DisplayName>The resource parameter passes to the ROPC endpoint</DisplayName>
    <DataType>string</DataType>
  </ClaimType>
  <ClaimType Id="refreshTokenIssuedOnDateTime">
    <DisplayName>An internal parameter used to determine whether the user should be permitted to
    authenticate again using their existing refresh token.</DisplayName>
    <DataType>string</DataType>
  </ClaimType>
  <ClaimType Id="refreshTokensValidFromDateTime">
    <DisplayName>An internal parameter used to determine whether the user should be permitted to
    authenticate again using their existing refresh token.</DisplayName>
    <DataType>string</DataType>
  </ClaimType>
</ClaimsSchema>
```

3. After **ClaimsSchema**, add a **ClaimsTransformations** element and its child elements to the **BuildingBlocks** element:

```

<ClaimsTransformations>
    <ClaimsTransformation Id="CreateSubjectClaimFromObjectID" TransformationMethod="CreateStringClaim">
        <InputParameters>
            <InputParameter Id="value" DataType="string" Value="Not supported currently. Use oid claim." />
        </InputParameters>
        <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="sub" TransformationClaimType="createdClaim" />
        </OutputClaims>
    </ClaimsTransformation>

    <ClaimsTransformation Id="AssertRefreshTokenIssuedLaterThanValidFromDate"
TransformationMethod="AssertDateTimeIsGreater Than">
        <InputClaims>
            <InputClaim ClaimTypeReferenceId="refreshTokenIssuedOnDateTime"
TransformationClaimType="leftOperand" />
            <InputClaim ClaimTypeReferenceId="refreshTokensValidFromDateTime"
TransformationClaimType="rightOperand" />
        </InputClaims>
        <InputParameters>
            <InputParameter Id="AssertIfEqualTo" DataType="boolean" Value="false" />
            <InputParameter Id="AssertIfRightOperandIsNotPresent" DataType="boolean" Value="true" />
        </InputParameters>
    </ClaimsTransformation>
</ClaimsTransformations>

```

4. Locate the **ClaimsProvider** element that has a **DisplayName** of `Local Account SignIn` and add following technical profile:

```

<TechnicalProfile Id="ResourceOwnerPasswordCredentials-OAUTH2">
    <DisplayName>Local Account SignIn</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <Metadata>
        <Item Key="UserMessageIfClaimsPrincipalDoesNotExist">We can't seem to find your account</Item>
        <Item Key="UserMessageIfInvalidPassword">Your password is incorrect</Item>
        <Item Key="UserMessageIfOldPasswordUsed">Looks like you used an old password</Item>
        <Item Key="DiscoverMetadataByTokenIssuer">true</Item>
        <Item Key="ValidTokenIssuerPrefixes">https://sts.windows.net/</Item>
        <Item Key="METADATA">https://login.microsoftonline.com/{tenant}/.well-known/openid-configuration</Item>
        <Item Key="authorization_endpoint">https://login.microsoftonline.com/{tenant}/oauth2/token</Item>
        <Item Key="response_types">id_token</Item>
        <Item Key="response_mode">query</Item>
        <Item Key="scope">email openid</Item>
    </Metadata>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="logonIdentifier" PartnerClaimType="username" Required="true"
DefaultValue="{OIDC:Username}" />
        <InputClaim ClaimTypeReferenceId="password" Required="true" DefaultValue="{OIDC:Password}" />
        <InputClaim ClaimTypeReferenceId="grant_type" DefaultValue="password" />
        <InputClaim ClaimTypeReferenceId="scope" DefaultValue="openid" />
        <InputClaim ClaimTypeReferenceId="nca" PartnerClaimType="nca" DefaultValue="1" />
        <InputClaim ClaimTypeReferenceId="client_id" DefaultValue="00000000-0000-0000-0000-000000000000" />
        <InputClaim ClaimTypeReferenceId="resource_id" PartnerClaimType="resource" DefaultValue="00000000-0000-0000-0000-000000000000" />
    </InputClaims>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="oid" />
        <OutputClaim ClaimTypeReferenceId="userPrincipalName" PartnerClaimType="upn" />
    </OutputClaims>
    <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromObjectID" />
    </OutputClaimsTransformations>
    <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>

```

Replace the **DefaultValue** of **client_id** with the Application ID of the ProxyIdentityExperienceFramework application that you created in the prerequisite tutorial. Then replace **DefaultValue** of **resource_id** with the Application ID of the IdentityExperienceFramework application that you also created in the prerequisite tutorial.

5. Add following **ClaimsProvider** elements with their technical profiles to the **ClaimsProviders** element:

```
<ClaimsProvider>
  <DisplayName>Azure Active Directory</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="AAD-UserReadUsingObjectId-CheckRefreshTokenDate">
      <Metadata>
        <Item Key="Operation">Read</Item>
        <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="objectId" />
        <OutputClaim ClaimTypeReferenceId="refreshTokensValidFromDateTime" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="AssertRefreshTokenIssuedLaterThanValidFromDate" />
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromObjectID" />
      </OutputClaimsTransformations>
      <IncludeTechnicalProfile ReferenceId="AAD-Common" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

<ClaimsProvider>
  <DisplayName>Session Management</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="SM-RefreshTokenReadAndSetup">
      <DisplayName>Trustframework Policy Engine Refresh Token Setup Technical Profile</DisplayName>
      <Protocol Name="None" />
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="objectId" />
        <OutputClaim ClaimTypeReferenceId="refreshTokenIssuedOnDateTime" />
      </OutputClaims>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

<ClaimsProvider>
  <DisplayName>Token Issuer</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="JwtIssuer">
      <Metadata>
        <!-- Point to the redeem refresh token user journey-->
        <Item Key="RefreshTokenUserJourneyId">ResourceOwnerPasswordCredentials-RedeemRefreshToken</Item>
      </Metadata>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

6. Add a **UserJourneys** element and its child elements to the **TrustFrameworkPolicy** element:

```

<UserJourney Id="ResourceOwnerPasswordCredentials">
    <PreserveOriginalAssertion>false</PreserveOriginalAssertion>
    <OrchestrationSteps>
        <OrchestrationStep Order="1" Type="ClaimsExchange">
            <ClaimsExchanges>
                <ClaimsExchange Id="ResourceOwnerFlow"
                    TechnicalProfileReferenceId="ResourceOwnerPasswordCredentials-OAUTH2" />
            </ClaimsExchanges>
        </OrchestrationStep>
        <OrchestrationStep Order="2" Type="ClaimsExchange">
            <ClaimsExchanges>
                <ClaimsExchange Id="AADUserReadWithObjectId"
                    TechnicalProfileReferenceId="AAD-UserReadUsingObjectId" />
            </ClaimsExchanges>
        </OrchestrationStep>
        <OrchestrationStep Order="3" Type="SendClaims"
            CpiIssuerTechnicalProfileReferenceId="JwtIssuer" />
    </OrchestrationSteps>
</UserJourney>
<UserJourney Id="ResourceOwnerPasswordCredentials-RedeemRefreshToken">
    <PreserveOriginalAssertion>false</PreserveOriginalAssertion>
    <OrchestrationSteps>
        <OrchestrationStep Order="1" Type="ClaimsExchange">
            <ClaimsExchanges>
                <ClaimsExchange Id="RefreshTokenSetupExchange"
                    TechnicalProfileReferenceId="SM-RefreshTokenReadAndSetup" />
            </ClaimsExchanges>
        </OrchestrationStep>
        <OrchestrationStep Order="2" Type="ClaimsExchange">
            <ClaimsExchanges>
                <ClaimsExchange Id="CheckRefreshTokenDateFromAadExchange"
                    TechnicalProfileReferenceId="AAD-UserReadUsingObjectId-CheckRefreshTokenDate" />
            </ClaimsExchanges>
        </OrchestrationStep>
        <OrchestrationStep Order="3" Type="SendClaims"
            CpiIssuerTechnicalProfileReferenceId="JwtIssuer" />
    </OrchestrationSteps>
</UserJourney>

```

7. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
8. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
9. Click **Upload**.

Create a relying party file

Next, update the relying party file that initiates the user journey that you created:

1. Make a copy of *SignUpOrSignin.xml* file in your working directory and rename it to *ROPC_Auth.xml*.
2. Open the new file and change the value of the **PolicyId** attribute for **TrustFrameworkPolicy** to a unique value. The policy ID is the name of your policy. For example, **B2C_1A_ROPC_Auth**.
3. Change the value of the **ReferenceId** attribute in **DefaultUserJourney** to **ResourceOwnerPasswordCredentials**.
4. Change the **OutputClaims** element to only contain the following claims:

```

<OutputClaim ClaimTypeReferenceId="sub" />
<OutputClaim ClaimTypeReferenceId="objectId" />
<OutputClaim ClaimTypeReferenceId="displayName" DefaultValue="" />
<OutputClaim ClaimTypeReferenceId="givenName" DefaultValue="" />
<OutputClaim ClaimTypeReferenceId="surname" DefaultValue="" />

```

5. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
6. Enable **Overwrite the policy if it exists**, and then browse to and select the *ROPC_Auth.xml* file.
7. Click **Upload**.

Test the policy

Use your favorite API development application to generate an API call, and review the response to debug your policy. Construct a call like this example with the following information as the body of the POST request:

https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/v2.0/token?p=B2C_1_ROPC_Auth

- Replace `your-tenant-name` with the name of your Azure AD B2C tenant.
- Replace `B2C_1A_ROPC_Auth` with the full name of your resource owner password credentials policy.

KEY	VALUE
username	<code>user-account</code>
password	<code>password1</code>
grant_type	<code>password</code>
scope	<code>openid</code> <code>application-id</code> <code>offline_access</code>
client_id	<code>application-id</code>
response_type	<code>token id_token</code>

- Replace `user-account` with the name of a user account in your tenant.
- Replace `password1` with the password of the user account.
- Replace `application-id` with the Application ID from the *ROPC_Auth_app* registration.
- *Offline_access* is optional if you want to receive a refresh token.

The actual POST request looks like the following example:

```

POST /yourtenant.onmicrosoft.com/oauth2/v2.0/token?p=B2C_1_ROPC_Auth HTTP/1.1
Host: yourtenant.b2clogin.com
Content-Type: application/x-www-form-urlencoded

username=contosouser.outlook.com.ws&password=Passxword1&grant_type=password&scope=openid+bef22d56-552f-4a5b-b90a-1988a7d634ce+offline_access&client_id=bef22d56-552f-4a5b-b90a-1988a7d634ce&response_type=token+id_token

```

A successful response with offline-access looks like the following example:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ik9YQjNhdTNScWhUQWN6R0RWZDM5djNpTmlyTWhqN2wxMjIySnh6TmgwR1ki...",
  "token_type": "Bearer",
  "expires_in": "3600",
  "refresh_token": "eyJraWQiOiJacW9pQlp2TW5pYVc2MUY0TnlfR3REVk1EVFBLbUJLb0FUcWQ1ZWfja1hBIiwidmVyIjoimS4wIiwiemlwIjoirGVmbGF0ZSiInNlcii6Ij...",
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ik9YQjNhdTNScWhUQWN6R0RWZDM5djNpTmlyTWhqN2wxMjIySnh6TmgwR1ki..."
}
```

Redeem a refresh token

Construct a POST call like the one shown here. Use the information in the following table as the body of the request:

https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/v2.0/token?p=B2C_1_ROPC_Auth

- Replace `your-tenant-name` with the name of your Azure AD B2C tenant.
- Replace `B2C_1A_ROPC_Auth` with the full name of your resource owner password credentials policy.

KEY	VALUE
grant_type	refresh_token
response_type	id_token
client_id	<code>application-id</code>
resource	<code>application-id</code>
refresh_token	<code>refresh-token</code>

- Replace `application-id` with the Application ID from the *ROPC_Auth_app* registration.
- Replace `refresh-token` with the **refresh_token** that was sent back in the previous response.

A successful response looks like the following example:

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilg1ZVhrNHh5b2p0RnVtMWtsM1l0djhkbE5QNC1jNTdkTzZRR1RWQndhT...",
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ilg1ZVhrNHh5b2p0RnVtMWtsM1l0djhkbE5QNC1jNTdkTzZRR1RWQn...",
  "token_type": "Bearer",
  "not_before": 1533672990,
  "expires_in": 3600,
  "expires_on": 1533676590,
  "resource": "bef2222d56-552f-4a5b-b90a-1988a7d634c3",
  "id_token_expires_in": 3600,
  "profile_info": "eyJ2ZXIiOiIxLjAiLCJ0aWQiOiI1MTZmYzA2NS1mZjM2LTRiOTMtYWE1YS1kNmVlZGE3Y2JhYzgiLCJzdWIiOm51bGwsIm5hbWUiOijEYXZpZE11IiwichHJlZmVycmVkcX3VzZXJuYW1IjpudWxsLCJpZHAIoiJMb2NhEFjY291bnQifQ",
  "refresh_token": "eyJraWQiOiJjcGltY29yZV8wOTI1MjAxNSIsInZlcii6IjEuMCIsInppcCI6IkR1ZmxhdGUiLCJzZXIiOiIxLjAi...",
  "refresh_token_expires_in": 1209600
}
```

Use a native SDK or App-Auth

Azure AD B2C meets OAuth 2.0 standards for public client resource owner password credentials and should be compatible with most client SDKs. For the latest information, see [Native App SDK for OAuth 2.0 and OpenID Connect implementing modern best practices](#).

Next steps

- See a full example of this scenario in the [Azure Active Directory B2C custom policy starter pack](#).
- Learn more about the tokens that are used by Azure Active Directory B2C in the [Token reference](#).

Enable Keep me signed in (KMSI) in Azure Active Directory B2C

2/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

You can enable Keep Me Signed In (KMSI) functionality for users of your web and native applications that have local accounts in your Azure Active Directory B2C (Azure AD B2C) directory. This feature grants access to users returning to your application without prompting them to reenter their username and password. This access is revoked when a user signs out.

Users should not enable this option on public computers.

The screenshot shows a sign-in form with the following fields:

- Email Address:** b2cuser@outlook.com
- Password:** (Redacted)
- Forgot your password?** (Link)
- Keep me signed in** (checkbox, highlighted with a red border)
- Sign in** (Large blue button)
- Don't have an account? Sign up now** (Text link)

Prerequisites

- An Azure AD B2C tenant that is configured to allow local account sign-in. KMSI is unsupported for external identity provider accounts.
- Complete the steps in [Get started with custom policies](#).

Configure the page identifier

To enable KMSI, set the content definition `DataUri` element to `page identifier unifiedssp` and `page version 1.1.0` or above.

1. Open the extension file of your policy. For example, `SocialAndLocalAccounts/ TrustFrameworkExtensions.xml`. This extension file is one of the policy files included in the custom policy starter pack, which you should have obtained in the prerequisite, [Get started with custom policies](#).

2. Search for the **BuildingBlocks** element. If the element doesn't exist, add it.
3. Add the **ContentDefinitions** element to the **BuildingBlocks** element of the policy.

Your custom policy should look like the following code snippet:

```
<BuildingBlocks>
  <ContentDefinitions>
    <ContentDefinition Id="api.signuporsignin">
      <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.1.0</DataUri>
    </ContentDefinition>
  </ContentDefinitions>
</BuildingBlocks>
```

4. Save the extensions file.

Configure a relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Open your custom policy file. For example, *SignUpOrSignIn.xml*.
2. If it doesn't already exist, add a `<UserJourneyBehaviors>` child node to the `<RelyingParty>` node. It must be located immediately after `<DefaultUserJourney ReferenceId="User journey Id" />`, for example:
`<DefaultUserJourney ReferenceId="SignUpOrSignIn" />`.
3. Add the following node as a child of the `<UserJourneyBehaviors>` element.

```
<UserJourneyBehaviors>
  <SingleSignOn Scope="Tenant" KeepAliveInDays="30" />
  <SessionExpiryType>Absolute</SessionExpiryType>
  <SessionExpiryInSeconds>1200</SessionExpiryInSeconds>
</UserJourneyBehaviors>
```

- **SessionExpiryType** - Indicates how the session is extended by the time specified in `SessionExpiryInSeconds` and `KeepAliveInDays`. The `Rolling` value (default) indicates that the session is extended every time the user performs authentication. The `Absolute` value indicates that the user is forced to reauthenticate after the time period specified.
 - **SessionExpiryInSeconds** - The lifetime of session cookies when *keep me signed in* is not enabled, or if a user does not select *keep me signed in*. The session expires after `SessionExpiryInSeconds` has passed, or the browser is closed.
 - **KeepAliveInDays** - The lifetime of session cookies when *keep me signed in* is enabled and the user selects *keep me signed in*. The value of `KeepAliveInDays` takes precedence over the `SessionExpiryInSeconds` value, and dictates the session expiry time. If a user closes the browser and reopens it later, they can still silently sign-in as long as it's within the `KeepAliveInDays` time period.

For more information, see [user journey behaviors](#).

We recommend that you set the value of `SessionExpiryInSeconds` to be a short period (1200 seconds), while the value of `KeepAliveInDays` can be set to a relatively long period (30 days), as shown in the following example:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <UserJourneyBehaviors>
    <SingleSignOn Scope="Tenant" KeepAliveInDays="30" />
    <SessionExpiryType>Absolute</SessionExpiryType>
    <SessionExpiryInSeconds>1200</SessionExpiryInSeconds>
  </UserJourneyBehaviors>
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
      <OutputClaim ClaimTypeReferenceId="tenantId" AlwaysUseDefaultValue="true" DefaultValue="{Policy:TenantObjectId}" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>
```

4. Save your changes and then upload the file.
5. To test the custom policy that you uploaded, in the Azure portal, go to the policy page, and then select **Run now**.

You can find the sample policy [here](#).

Configure password change using custom policies in Azure Active Directory B2C

1/28/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In Azure Active Directory B2C (Azure AD B2C), you can enable users who are signed in with a local account to change their password without having to prove their authenticity by email verification. If the session expires by the time the user gets to the password change flow, they're prompted to sign in again. This article shows you how to configure password change in [custom policies](#). It's also possible to configure [self-service password reset](#) for user flows.

Prerequisites

Complete the steps in [Get started with custom policies in Active Directory B2C](#).

Add the elements

1. Open your *TrustframeworkExtensions.xml* file and add the following **ClaimType** element with an identifier of `oldPassword` to the **ClaimsSchema** element:

```
<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="oldPassword">
      <DisplayName>Old Password</DisplayName>
      <DataType>string</DataType>
      <UserHelpText>Enter password</UserHelpText>
      <UserInputType>Password</UserInputType>
    </ClaimType>
  </ClaimsSchema>
</BuildingBlocks>
```

2. A **ClaimsProvider** element contains the technical profile that authenticates the user. Add the following claims providers to the **ClaimsProviders** element:

```
<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>Local Account SignIn</DisplayName>
    <TechnicalProfiles>
      <TechnicalProfile Id="login-NonInteractive-PasswordChange">
        <DisplayName>Local Account SignIn</DisplayName>
        <Protocol Name="OpenIdConnect" />
        <Metadata>
          <Item Key="UserMessageIfClaimsPrincipalDoesNotExist">We can't seem to find your account</Item>
          <Item Key="UserMessageIfInvalidPassword">Your password is incorrect</Item>
          <Item Key="UserMessageIfOldPasswordUsed">Looks like you used an old password</Item>
          <Item Key="ProviderName">https://sts.windows.net/</Item>
          <Item Key="METADATA">https://login.microsoftonline.com/{tenant}/.well-known/openid-configuration</Item>
          <Item
```

```

Key="authorization_endpoint">https://login.microsoftonline.com/{tenant}/oauth2/token</Item>
  <Item Key="response_types">id_token</Item>
  <Item Key="response_mode">query</Item>
  <Item Key="scope">email openid</Item>
  <Item Key="UsePolicyInRedirectUri">false</Item>
  <Item Key="HttpBinding">POST</Item>
  <Item Key="client_id">ProxyIdentityExperienceFrameworkAppId</Item>
  <Item Key="IdTokenAudience">IdentityExperienceFrameworkAppId</Item>
</Metadata>
<InputClaims>
  <InputClaim ClaimTypeReferenceId="signInName" PartnerClaimType="username" Required="true" />
  <InputClaim ClaimTypeReferenceId="oldPassword" PartnerClaimType="password" Required="true" />
  <InputClaim ClaimTypeReferenceId="grant_type" DefaultValue="password" />
  <InputClaim ClaimTypeReferenceId="scope" DefaultValue="openid" />
  <InputClaim ClaimTypeReferenceId="nca" PartnerClaimType="nca" DefaultValue="1" />
  <InputClaim ClaimTypeReferenceId="client_id"
DefaultValue="ProxyIdentityExperienceFrameworkAppID" />
  <InputClaim ClaimTypeReferenceId="resource_id" PartnerClaimType="resource"
DefaultValue="IdentityExperienceFrameworkAppID" />
</InputClaims>
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="oid" />
  <OutputClaim ClaimTypeReferenceId="tenantId" PartnerClaimType="tid" />
  <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
  <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="family_name" />
  <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
  <OutputClaim ClaimTypeReferenceId="userPrincipalName" PartnerClaimType="upn" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource"
DefaultValue="localAccountAuthentication" />
</OutputClaims>
</TechnicalProfile>
</TechnicalProfiles>
</ClaimsProvider>
<ClaimsProvider>
  <DisplayName>Local Account Password Change</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="LocalAccountWritePasswordChangeUsingObjectId">
      <DisplayName>Change password (username)</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider,
Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="ContentDefinitionReferenceId">api.selfasserted</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="issuer_secret" StorageReferenceId="B2C_1A_TokenSigningKeyContainer" />
      </CryptographicKeys>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="objectId" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="oldPassword" Required="true" />
        <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />
        <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
      </OutputClaims>
      <ValidationTechnicalProfiles>
        <ValidationTechnicalProfile ReferenceId="login-NonInteractive-PasswordChange" />
        <ValidationTechnicalProfile ReferenceId="AAD-UserWritePasswordUsingObjectId" />
      </ValidationTechnicalProfiles>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
</ClaimsProviders>

```

Replace `IdentityExperienceFrameworkAppId` with the application ID of the IdentityExperienceFramework application that you created in the prerequisite tutorial. Replace `ProxyIdentityExperienceFrameworkAppId` with the application ID of the ProxyIdentityExperienceFramework application that you also previously created.

3. The **UserJourney** element defines the path that the user takes when interacting with your application. Add the **UserJourneys** element if it doesn't exist with the **UserJourney** identified as `<UserJourneys>` :

```
<UserJourneys>
  <UserJourney Id="PasswordChange">
    <OrchestrationSteps>
      <OrchestrationStep Order="1" Type="ClaimsProviderSelection"
ContentDefinitionReferenceId="api.idpselections">
        <ClaimsProviderSelections>
          <ClaimsProviderSelection TargetClaimsExchangeId="LocalAccountSigninEmailExchange" />
        </ClaimsProviderSelections>
      </OrchestrationStep>
      <OrchestrationStep Order="2" Type="ClaimsExchange">
        <ClaimsExchanges>
          <ClaimsExchange Id="LocalAccountSigninEmailExchange"
TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="3" Type="ClaimsExchange">
        <ClaimsExchanges>
          <ClaimsExchange Id="NewCredentials"
TechnicalProfileReferenceId="LocalAccountWritePasswordChangeUsingObjectId" />
        </ClaimsExchanges>
      </OrchestrationStep>
      <OrchestrationStep Order="4" Type="SendClaims" CpiIssuerTechnicalProfileReferenceId="JwtIssuer"
/>
    </OrchestrationSteps>
    <ClientDefinition ReferenceId="DefaultWeb" />
  </UserJourney>
</UserJourneys>
```

4. Save the *TrustFrameworkExtensions.xml* policy file.
5. Copy the *ProfileEdit.xml* file that you downloaded with the starter pack and name it *ProfileEditPasswordChange.xml*.
6. Open the new file and update the **PolicyId** attribute with a unique value. This value is the name of your policy. For example, *B2C_1A_profile_edit_password_change*.
7. Modify the **ReferenceId** attribute in `<DefaultUserJourney>` to match the ID of the new user journey that you created. For example, *PasswordChange*.
8. Save your changes.

You can find the sample policy [here](#).

Test your policy

When testing your applications in Azure AD B2C, it can be useful to have the Azure AD B2C token returned to <https://jwt.ms> to be able to review the claims in it.

Upload the files

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **Identity Experience Framework**.
5. On the Custom Policies page, click **Upload Policy**.
6. Select **Overwrite the policy if it exists**, and then search for and select the *TrustframeworkExtensions.xml* file.

7. Click **Upload**.
8. Repeat steps 5 through 7 for the relying party file, such as *ProfileEditPasswordChange.xml*.

Run the policy

1. Open the policy that you changed. For example, *B2C_1A_profile_edit_password_change*.
2. For **Application**, select your application that you previously registered. To see the token, the **Reply URL** should show `https://jwt.ms`.
3. Click **Run now**. Sign in with the account that you previously created. You should now have the opportunity to change the password.

Next steps

- Learn about how you can [Configure password complexity using custom policies in Azure Active Directory B2C](#).

Set up phone sign-up and sign-in with custom policies in Azure AD B2C (Preview)

2/26/2020 • 2 minutes to read • [Edit Online](#)

Phone sign-up and sign-in in Azure Active Directory B2C (Azure AD B2C) enables your users to sign up and sign in to your applications by using a one-time password (OTP) sent in a text message to their phone. One-time passwords can help minimize the risk of your users forgetting or having their passwords compromised.

Follow the steps in this article to use the custom policies to enable your customers to sign up and sign in to your applications by using a one-time password sent to their phone.

NOTE

This feature is in public preview.

Pricing

One-time passwords are sent to your users by using SMS text messages, and you may be charged for each message sent. For pricing information, see the **Separate Charges** section of [Azure Active Directory B2C pricing](#).

Prerequisites

You need the following resources in place before setting up OTP.

- [Azure AD B2C tenant](#)
- [Web application registered](#) in your tenant
- [Custom policies](#) uploaded to your tenant

Get the phone sign-up & sign-in starter pack

Start by updating the phone sign-up and sign-in custom policy files to work with your Azure AD B2C tenant.

The following steps assume that you've completed the [prerequisites](#) and have already cloned the [custom policy starter pack](#) repository to your local machine.

1. Find the [phone sign-up and sign-in custom policy files](#) in your local clone of the starter pack repo, or download them directly. The XML policy files are located in the following directory:

```
active-directory-b2c-custom-policy-starterpack/scenarios/ phone-number-passwordless
```

2. In each file, replace the string `yourtenant` with the name of your Azure AD B2C tenant. For example, if the name of your B2C tenant is *contosob2c*, all instances of `yourtenant.onmicrosoft.com` become `contosob2c.onmicrosoft.com`.
3. Complete the steps in the [Add application IDs to the custom policy](#) section of [Get started with custom policies in Azure Active Directory B2C](#). In this case, update `/phone-number-passwordless/Phone_Email_Base.xml` with the **Application (client) IDs** of the two applications you registered when completing the prerequisites, *IdentityExperienceFramework* and *ProxyIdentityExperienceFramework*.

Upload the policy files

1. Sign in to the [Azure portal](#) and navigate to your Azure AD B2C tenant.
2. Under **Policies**, select **Identity Experience Framework**.
3. Select **Upload custom policy**.
4. Upload the policy files in the following order:
 - a. *Phone_Email_Base.xml*
 - b. *SignUpOrSignInWithPhone.xml*
 - c. *SignUpOrSignInWithPhoneOrEmail.xml*
 - d. *ProfileEditPhoneOnly.xml*
 - e. *ProfileEditPhoneEmail.xml*
 - f. *ChangePhoneNumber.xml*
 - g. *PasswordResetEmail.xml*

As you upload each file, Azure adds the prefix `B2C_1A_`.

Test the custom policy

1. Under **Custom policies**, select **B2C_1A_SignUpOrSignInWithPhone**.
2. Under **Select application**, select the *webapp1* application that you registered when completing the prerequisites.
3. For **Select reply url**, choose `https://jwt.ms`.
4. Select **Run now** and sign up using an email address or a phone number.
5. Select **Run now** once again and sign in with the same account to confirm that you have the correct configuration.

Get user account by phone number

A user that signs up with a phone number but does not provide a recovery email address is recorded in your Azure AD B2C directory with their phone number as their sign-in name. If the user then wishes to change their phone number, your help desk or support team must first find their account, and then update their phone number.

You can find a user by their phone number (sign-in name) by using [Microsoft Graph](#):

```
GET https://graph.microsoft.com/v1.0/users?$filter=identities/any(c:c/issuerAssignedId eq '{phone number}' and c/issuer eq '{tenant name}.onmicrosoft.com')
```

For example:

```
GET https://graph.microsoft.com/v1.0/users?$filter=identities/any(c:c/issuerAssignedId eq '450334567890' and c/issuer eq 'contosob2c.onmicrosoft.com')
```

Next steps

You can find the phone sign-up and sign-in custom policy starter pack (and other starter packs) on GitHub:

[Azure-Samples/active-directory-b2c-custom-policy-starterpack/scenarios/phone-number-passwordless](https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack/tree/scenarios/phone-number-passwordless)

The starter pack policy files use multi-factor authentication technical profiles and phone number claims transformations:

- [Define an Azure Multi-Factor Authentication technical profile](#)
- [Define phone number claims transformations](#)

Add claims and customize user input using custom policies in Azure Active Directory B2C

1/28/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In this article, you add a new user provided entry (a claim) to your sign-up user journey in Azure Active Directory B2C (Azure AD B2C). You configure the entry as a dropdown and define whether it's required.

Prerequisites

Complete the steps in the article [Getting Started with Custom Policies](#). Test the sign-up or sign-in user journey to sign up a new local account before proceeding.

Add claims

Gathering initial data from your users is achieved using the sign-up or sign-in user journey. Additional claims can be gathered later by using a profile edit user journey. Anytime Azure AD B2C gathers information directly from the user interactively, the Identity Experience Framework uses its self-asserted provider.

Define the claim

Let's ask the user for their city. Add the following element to the **ClaimsSchema** element in the TrustFrameworkBase policy file:

```
<ClaimType Id="city">
  <DisplayName>city</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Your city</UserHelpText>
  <UserInputType>TextBox</UserInputType>
</ClaimType>
```

The following elements are used to define the claim:

- **DisplayName** - A string that defines the user-facing label.
- **UserHelpText** - Helps the user understand what is required.
- **UserInputType** - Can be a text box, a radio selection, a drop-down list, or a multiple selection.

TextBox

```
<ClaimType Id="city">
  <DisplayName>city where you work</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Your city</UserHelpText>
  <UserInputType>TextBox</UserInputType>
</ClaimType>
```

RadioSingleSelect

```

<ClaimType Id="city">
  <DisplayName>city where you work</DisplayName>
  <DataType>string</DataType>
  <UserInputType>RadioSingleSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Bellevue" Value="bellevue" SelectByDefault="false" />
    <Enumeration Text="Redmond" Value="redmond" SelectByDefault="false" />
    <Enumeration Text="Kirkland" Value="kirkland" SelectByDefault="false" />
  </Restriction>
</ClaimType>

```

DropdownSingleSelect

City where you work

City where you work
Bellevue
Redmond
Kirkland

```

<ClaimType Id="city">
  <DisplayName>city where you work</DisplayName>
  <DataType>string</DataType>
  <UserInputType>DropdownSingleSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Bellevue" Value="bellevue" SelectByDefault="false" />
    <Enumeration Text="Redmond" Value="redmond" SelectByDefault="false" />
    <Enumeration Text="Kirkland" Value="kirkland" SelectByDefault="false" />
  </Restriction>
</ClaimType>

```

CheckboxMultiSelect

Receive updates from which cities?



Bellevue



Redmond



Kirkland

Continue

Cancel

```

<ClaimType Id="city">
  <DisplayName>Receive updates from which cities?</DisplayName>
  <DataType>string</DataType>
  <UserInputType>CheckboxMultiSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Bellevue" Value="bellevue" SelectByDefault="false" />
    <Enumeration Text="Redmond" Value="redmond" SelectByDefault="false" />
    <Enumeration Text="Kirkland" Value="kirkland" SelectByDefault="false" />
  </Restriction>
</ClaimType>

```

Add the claim to the user journey

1. Add the claim as an `<outputClaim ClaimTypeReferenceId="city"/>` to the `LocalAccountSignUpWithLogonEmail` technical profile found in the `TrustFrameworkBase` policy file. This technical profile uses the `SelfAssertedAttributeProvider`.

```

<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
  <DisplayName>Email signup</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider,
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="IpAddressClaimReferenceId">IpAddress</Item>
    <Item Key="ContentDefinitionReferenceId">api.localaccountsSignup</Item>
    <Item Key="language.button_continue">Create</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="issuer_secret" StorageReferenceId="TokenSigningKeyContainer" />
  </CryptographicKeys>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="objectId" />
    <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />
    <OutputClaim ClaimTypeReferenceId="newPassword" Required="true" />
    <OutputClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
    <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
    <OutputClaim ClaimTypeReferenceId="authenticationSource" />
    <OutputClaim ClaimTypeReferenceId="newUser" />
    <!-- Optional claims, to be collected from the user -->
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="surName" />
    <OutputClaim ClaimTypeReferenceId="city"/>
  </OutputClaims>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingLogonEmail" />
  </ValidationTechnicalProfiles>
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>

```

2. Add the claim to the AAD-UserWriteUsingLogonEmail technical profile as a

`<PersistedClaim ClaimTypeReferenceId="city" />` to write the claim to the AAD directory after collecting it from the user. You may skip this step if you prefer not to persist the claim in the directory for future use.

```

<!-- Technical profiles for local accounts -->
<TechnicalProfile Id="AAD-UserWriteUsingLogonEmail">
    <Metadata>
        <Item Key="Operation">Write</Item>
        <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">true</Item>
    </Metadata>
    <IncludeInSso>false</IncludeInSso>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="signInNames.emailAddress" Required="true" />
    </InputClaims>
    <PersistedClaims>
        <!-- Required claims -->
        <PersistedClaim ClaimTypeReferenceId="email" PartnerClaimType="signInNames.emailAddress" />
        <PersistedClaim ClaimTypeReferenceId="newPassword" PartnerClaimType="password" />
        <PersistedClaim ClaimTypeReferenceId="displayName" DefaultValue="unknown" />
        <PersistedClaim ClaimTypeReferenceId="passwordPolicies" DefaultValue="DisablePasswordExpiration" />
        <!-- Optional claims. -->
        <PersistedClaim ClaimTypeReferenceId="givenName" />
        <PersistedClaim ClaimTypeReferenceId="surname" />
        <PersistedClaim ClaimTypeReferenceId="city" />
    </PersistedClaims>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="objectId" />
        <OutputClaim ClaimTypeReferenceId="newUser" PartnerClaimType="newClaimsPrincipalCreated" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="localAccountAuthentication" />
        <OutputClaim ClaimTypeReferenceId="userPrincipalName" />
        <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
    </OutputClaims>
    <IncludeTechnicalProfile ReferenceId="AAD-Common" />
    <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>

```

- Add the `<OutputClaim ClaimTypeReferenceId="city" />` claim to the technical profiles that read from the directory when a user signs in.

```

<TechnicalProfile Id="AAD-UserReadUsingEmailAddress">
    <Metadata>
        <Item Key="Operation">Read</Item>
        <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
        <Item Key="UserMessageIfClaimsPrincipalDoesNotExist">An account could not be found for the provided user ID.</Item>
    </Metadata>
    <IncludeInSso>false</IncludeInSso>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="signInNames" Required="true" />
    </InputClaims>
    <OutputClaims>
        <!-- Required claims -->
        <OutputClaim ClaimTypeReferenceId="objectId" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="localAccountAuthentication" />
        <!-- Optional claims -->
        <OutputClaim ClaimTypeReferenceId="userPrincipalName" />
        <OutputClaim ClaimTypeReferenceId="displayName" />
        <OutputClaim ClaimTypeReferenceId="otherMails" />
        <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
        <OutputClaim ClaimTypeReferenceId="city" />
    </OutputClaims>
    <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>

```

```

<TechnicalProfile Id="AAD-UserReadUsingObjectId">
  <Metadata>
    <Item Key="Operation">Read</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
  </Metadata>
  <IncludeInSso>false</IncludeInSso>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <OutputClaims>
    <!-- Optional claims -->
    <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="otherMails" />
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="city" />
  </OutputClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>

```

- Add the `<OutputClaim ClaimTypeReferenceId="city" />` claim to the SignUpOrSignIn.xml file so that this claim is sent to the application in the token after a successful user journey.

```

<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
      <OutputClaim ClaimTypeReferenceId="city" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>

```

Test the custom policy

- Sign in to the [Azure portal](#).
- Make sure you're using the directory that contains your Azure AD tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your Azure AD tenant.
- Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
- Select **Identity Experience Framework (Preview)**.
- Select **Upload Custom Policy**, and then upload the two policy files that you changed.
- Select the sign-up or sign-in policy that you uploaded, and click the **Run now** button.
- You should be able to sign up using an email address.

The signup screen should look similar to this:

Verification is necessary. Please click Send button.

Email Address
Please enter a valid email address. This information is required.

Send verification code

New Password

Confirm New Password

Given Name

Surname

city where you work

city where you work

- Bellevue
- Redmond
- Kirkland

The token sent back to your application includes the `city` claim.

```
{
  "exp": 1493596822,
  "nbf": 1493593222,
  "ver": "1.0",
  "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",
  "sub": "9c2a3a9e-ac65-4e46-a12d-9557b63033a9",
  "aud": "4e87c1dd-e5f5-4ac8-8368-bc6a98751b8b",
  "acr": "b2c_1a_trustf_signup_signin",
  "nonce": "defaultNonce",
  "iat": 1493593222,
  "auth_time": 1493593222,
  "email": "joe@outlook.com",
  "given_name": "Joe",
  "family_name": "Ras",
  "city": "Bellevue",
  "name": "unknown"
}
```

Optional: Remove email verification

To skip email verification, you can choose to remove `<OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />`. In this case, the email address is required but not verified, unless "Required" = true is removed. Carefully consider if this option is right for your use cases.

Verified email is enabled by default in the `<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">` in the TrustFrameworkBase policy file:

```
<OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="Verified.Email" Required="true" />
```

Next steps

Learn how to [Use custom attributes in a custom profile edit policy](#).

Customize the user interface of your application using a custom policy in Azure Active Directory B2C

2/17/2020 • 10 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

By completing the steps in this article, you create a sign-up and sign-in custom policy with your brand and appearance. With Azure Active Directory B2C (Azure AD B2C), you get nearly full control of the HTML and CSS content that's presented to users. When you use a custom policy, you configure UI customization in XML instead of using controls in the Azure portal.

Prerequisites

Complete the steps in [Get started with custom policies](#). You should have a working custom policy for sign-up and sign-in with local accounts.

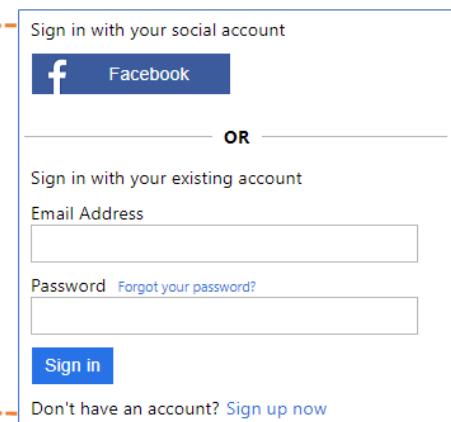
Use custom page content

By using the page UI customization feature, you can customize the look and feel of any custom policy. You can also maintain brand and visual consistency between your application and Azure AD B2C.

How it works

Azure AD B2C runs code in your customer's browser by using [Cross-Origin Resource Sharing \(CORS\)](#). At runtime, content is loaded from a URL you specify in your user flow or custom policy. Each page in the user experience loads its content from the URL you specify for that page. After content is loaded from your URL, it's merged with an HTML fragment inserted by Azure AD B2C, and then the page is displayed to your customer.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Add your title here!</title>
  </head>
  <body>
    <div id="api"></div>
  </body>
</html>
```



Custom HTML page content

Create an HTML page with your own branding to serve your custom page content. This page can be a static `*.html` page, or a dynamic page like .NET, Node.js, or PHP.

Your custom page content can contain any HTML elements, including CSS and JavaScript, but cannot include insecure elements like iframes. The only required element is a div element with `id` set to `api`, such as this one

```
<div id="api"></div>
```

 within your HTML page.

```
<!DOCTYPE html>
<html>
<head>
    <title>My Product Brand Name</title>
</head>
<body>
    <div id="api"></div>
</body>
</html>
```

Customize the default Azure AD B2C pages

Instead of creating your custom page content from scratch, you can customize Azure AD B2C's default page content.

The following table lists the default page content provided by Azure AD B2C. Download the files and use them as a starting point for creating your own custom pages.

DEFAULT PAGE	DESCRIPTION	CONTENT DEFINITION ID (CUSTOM POLICY ONLY)
exception.html	Error page. This page is displayed when an exception or an error is encountered.	<i>api.error</i>
selfasserted.html	Self-Asserted page. Use this file as a custom page content for a social account sign-up page, a local account sign-up page, a local account sign-in page, password reset, and more. The form can contain various input controls, such as: a text input box, a password entry box, a radio button, single-select drop-down boxes, and multi-select check boxes.	<i>api.localaccountssignin,</i> <i>api.localaccountssignup ,</i> <i>api.localaccountpasswordreset,</i> <i>api.selfasserted</i>
multifactor-1.0.0.html	Multi-factor authentication page. On this page, users can verify their phone numbers (by using text or voice) during sign-up or sign-in.	<i>api.phonefactor</i>
updateprofile.html	Profile update page. This page contains a form that users can access to update their profile. This page is similar to the social account sign-up page, except for the password entry fields.	<i>api.selfasserted.profileupdate</i>
unified.html	Unified sign-up or sign-in page. This page handles the user sign-up and sign-in process. Users can use enterprise identity providers, social identity providers such as Facebook or Google+, or local accounts.	<i>api.signuporsignin</i>

Hosting the page content

When using your own HTML and CSS files to customize the UI, host your UI content on any publicly available

HTTPS endpoint that supports CORS. For example, [Azure Blob storage](#), [Azure App Services](#), web servers, CDNs, AWS S3, or file sharing systems.

Guidelines for using custom page content

- Use an absolute URL when you include external resources like media, CSS, and JavaScript files in your HTML file.
- Using [page layout version](#) 1.2.0 and above, you can add the `data-preload="true"` attribute in your HTML tags to control the load order for CSS and JavaScript. With `data-preload=true`, the page is constructed before being shown to the user. This attribute helps prevent the page from "flickering" by preloading the CSS file, without the un-styled HTML being shown to the user. The following HTML code snippet shows the use of the `data-preload` tag.

```
<link href="https://path-to-your-file/sample.css" rel="stylesheet" type="text/css" data-preload="true"/>
```

- We recommend that you start with the default page content and build on top of it.
- You can include JavaScript in your custom content for both [user flows](#) and [custom policies](#).
- Supported browser versions are:
 - Internet Explorer 11, 10, and Microsoft Edge
 - Limited support for Internet Explorer 9 and 8
 - Google Chrome 42.0 and above
 - Mozilla Firefox 38.0 and above
- Due to security restrictions, Azure AD B2C doesn't support `frame`, `iframe`, or `form` HTML elements.

Custom page content walkthrough

Here's an overview of the process:

1. Prepare a location to host your custom page content (a publicly accessible, CORS-enabled HTTPS endpoint).
2. Download and customize a default page content file, for example `unified.html`.
3. Publish your custom page content your publicly available HTTPS endpoint.
4. Set cross-origin resource sharing (CORS) for your web app.
5. Point your policy to your custom policy content URI.

1. Create your HTML content

Create a custom page content with your product's brand name in the title.

1. Copy the following HTML snippet. It is well-formed HTML5 with an empty element called `<div id="api"></div>` located within the `<body>` tags. This element indicates where Azure AD B2C content is to be inserted.

```
<!DOCTYPE html>
<html>
<head>
    <title>My Product Brand Name</title>
</head>
<body>
    <div id="api"></div>
</body>
</html>
```

2. Paste the copied snippet in a text editor, and then save the file as `customize-ui.html`.

NOTE

HTML form elements will be removed due to security restrictions if you use login.microsoftonline.com. If you want to use HTML form elements in your custom HTML content, [use b2clogin.com](#).

2. Create an Azure Blob storage account

In this article, we use Azure Blob storage to host our content. You can choose to host your content on a web server, but you must [enable CORS on your web server](#).

To host your HTML content in Blob storage, perform the following steps:

1. Sign in to the [Azure portal](#).
2. On the **Hub** menu, select **New > Storage > Storage account**.
3. Select a **Subscription** for your storage account.
4. Create a **Resource group** or select an existing one.
5. Enter a unique **Name** for your storage account.
6. Select the **Geographic location** for your storage account.
7. **Deployment model** can remain **Resource Manager**.
8. **Performance** can remain **Standard**.
9. Change **Account Kind** to **Blob storage**.
10. **Replication** can remain **RA-GRS**.
11. **Access tier** can remain **Hot**.
12. Select **Review + create** to create the storage account. After the deployment is completed, the **Storage account** page opens automatically.

2.1 Create a container

To create a public container in Blob storage, perform the following steps:

1. Under **Blob service** in the left-hand menu, select **Blobs**.
2. Select **+Container**.
3. For **Name**, enter *root*. This can be a name of your choosing, for example *wingtiptoys*, but we use *root* in this example for simplicity.
4. For **Public access level**, select **Blob**, then **OK**.
5. Select **root** to open the new container.

2.2 Upload your custom page content files

1. Select **Upload**.
2. Select the folder icon next to **Select a file**.
3. Navigate to and select **customize-ui.html**, which you created earlier in the Page UI customization section.
4. If you want to upload to a subfolder, expand **Advanced** and enter a folder name in **Upload to folder**.
5. Select **Upload**.
6. Select the **customize-ui.html** blob that you uploaded.
7. To the right of the **URL** text box, select the **Copy to clipboard** icon to copy the URL to your clipboard.
8. In web browser, navigate to the 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**.

3. Configure CORS

Configure Blob storage for Cross-Origin Resource Sharing by performing the following steps:

1. In the menu, select **CORS**.

2. For **Allowed origins**, enter `https://your-tenant-name.b2clogin.com`. Replace `your-tenant-name` with the name of your Azure AD B2C tenant. For example, `https://fabrikam.b2clogin.com`. Use all lowercase letters when entering your tenant name.
3. For **Allowed Methods**, select both `GET` and `OPTIONS`.
4. For **Allowed Headers**, enter an asterisk (*).
5. For **Exposed Headers**, enter an asterisk (*).
6. For **Max age**, enter 200.
7. Select **Save**.

3.1 Test CORS

Validate that you're ready by performing the following steps:

1. Navigate to www.test-cors.org and paste the URL in the **Remote URL** box.
2. Select **Send Request**. 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.

4. Modify the extensions file

To configure UI customization, copy the **ContentDefinition** and its child elements from the base file to the extensions file.

1. Open the base file of your policy. For example, `SocialAndLocalAccounts/ TrustFrameworkBase.xml`. This base file is one of the policy files included in the custom policy starter pack, which you should have obtained in the prerequisite, [Get started with custom policies](#).
2. Search for and copy the entire contents of the **ContentDefinitions** element.
3. Open the extension file. For example, `TrustFrameworkExtensions.xml`. Search for the **BuildingBlocks** element. If the element doesn't exist, add it.
4. Paste the entire contents of the **ContentDefinitions** element that you copied as a child of the **BuildingBlocks** element.
5. Search for the **ContentDefinition** element that contains `Id="api.signuporsignin"` in the XML that you copied.
6. Change the value of **LoadUri** to the URL of the HTML file that you uploaded to storage. For example, `https://your-storage-account.blob.core.windows.net/your-container/customize-ui.html`.

Your custom policy should look like the following code snippet:

```
<BuildingBlocks>
  <ContentDefinitions>
    <ContentDefinition Id="api.signuporsignin">
      <LoadUri>https://your-storage-account.blob.core.windows.net/your-container/customize-
ui.html</LoadUri>
      <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
      <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0</DataUri>
      <Metadata>
        <Item Key="DisplayName">Signin and Signup</Item>
      </Metadata>
    </ContentDefinition>
  </ContentDefinitions>
</BuildingBlocks>
```

7. Save the extensions file.

5. Upload and test your updated custom policy

5.1 Upload the custom policy

1. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
2. Search for and select **Azure AD B2C**.
3. Under **Policies**, select **Identity Experience Framework**.
4. Select **Upload custom policy**.
5. Upload the extensions file that you previously changed.

5.2 Test the custom policy by using Run now

1. Select the policy that you uploaded, and then select **Run now**.
2. You should be able to sign up by using an email address.

Sample templates

You can find sample templates for UI customization here:

```
git clone https://github.com/Azure-Samples/Azure-AD-B2C-page-templates
```

This project contains the following templates:

- [Ocean Blue](#)
- [Slate Gray](#)

To use the sample:

1. Clone the repo on your local machine. Choose a template folder `/ocean_blue` or `/slate_gray`.
2. Upload all the files under the template folder and the `/assets` folder, to Blob storage as described in the previous sections.
3. Next, open each `*.html` file in the root of either `/ocean_blue` or `/slate_gray`, replace all instances of relative URLs with the URLs of the css, images, and fonts files you uploaded in step 2. For example:

```
<link href=".//css/assets.css" rel="stylesheet" type="text/css" />
```

To

```
<link href="https://your-storage-account.blob.core.windows.net/your-container/css/assets.css" rel="stylesheet" type="text/css" />
```

4. Save the `*.html` files and upload them to Blob storage.
5. Now modify the policy, pointing to your HTML file, as mentioned previously.
6. If you see missing fonts, images, or CSS, check your references in the extensions policy and the `*.html` files.

Configure dynamic custom page content URI

By using Azure AD B2C custom policies, you can send a parameter in the URL path, or a query string. By passing the parameter to your HTML endpoint, you can dynamically change the page content. For example, you can change the background image on the Azure AD B2C sign-up or sign-in page, based on a parameter that you pass from your web or mobile application. The parameter can be any [claim resolver](#), such as the application ID,

language ID, or custom query string parameter, such as `campaignId`.

Sending query string parameters

To send query string parameters, in the [relying party policy](#), add a `ContentDefinitionParameters` element as shown below.

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <UserJourneyBehaviors>
    <ContentDefinitionParameters>
      <Parameter Name="campaignId">{OAUTH-KV:campaignId}</Parameter>
      <Parameter Name="lang">{Culture:LanguageName}</Parameter>
      <Parameter Name="appId">{OIDC:ClientId}</Parameter>
    </ContentDefinitionParameters>
  </UserJourneyBehaviors>
  ...
</RelyingParty>
```

In your content definition, change the value of `LoadUri` to https://<app_name>.azurewebsites.net/home/unified.

Your custom policy `ContentDefinition` should look like the following code snippet:

```
<ContentDefinition Id="api.signuporsignin">
  <LoadUri>https://<app_name>.azurewebsites.net/home/unified</LoadUri>
  ...
</ContentDefinition>
```

When Azure AD B2C loads the page, it makes a call to your web server endpoint:

```
https://<app_name>.azurewebsites.net/home/unified?campaignId=123&lang=fr&appId=f893d6d3-3b6d-480d-a330-1707bf80ebea
```

Dynamic page content URI

Content can be pulled from different places based on the parameters used. In your CORS-enabled endpoint, set up a folder structure to host content. For example, you can organize the content in following structure. Root *folder/folder per language/your html files*. For example, your custom page URI might look like:

```
<ContentDefinition Id="api.signuporsignin">
  <LoadUri>https://contoso.blob.core.windows.net/{Culture:LanguageName}/myHTML/unified.html</LoadUri>
  ...
</ContentDefinition>
```

Azure AD B2C sends the two letter ISO code for the language, `fr` for French:

```
https://contoso.blob.core.windows.net/fr/myHTML/unified.html
```

Next steps

For more information about UI elements that can be customized, see [reference guide for UI customization for user flows](#).

Custom email verification in Azure Active Directory B2C

2/5/2020 • 9 minutes to read • [Edit Online](#)

Use custom email in Azure Active Directory B2C (Azure AD B2C) to send customized email to users that sign up to use your applications. By using [DisplayControls](#) (currently in preview) and a third-party email provider, you can use your own email template and *From:* address and subject, as well as support localization and custom one-time password (OTP) settings.

Custom email verification requires the use of a third-party email provider like [SendGrid](#) or [SparkPost](#), a custom REST API, or any HTTP-based email provider (including your own). This article describes setting up a solution that uses SendGrid.

NOTE

This feature is in public preview.

Create a SendGrid account

If you don't already have one, start by setting up a SendGrid account (Azure customers can unlock 25,000 free emails each month). For setup instructions, see the [Create a SendGrid Account](#) section of [How to send email using SendGrid with Azure](#).

Be sure to complete the section in which you [create a SendGrid API key](#). Record the API key for use in a later step.

Create Azure AD B2C policy key

Next, store the SendGrid API key in an Azure AD B2C policy key for your policies to reference.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose your Azure AD B2C directory.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.
6. For **Options**, choose `Manual`.
7. Enter a **Name** for the policy key. For example, `SendGridSecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter your client secret that you previously recorded.
9. For **Key usage**, select `Signature`.
10. Select **Create**.

Create SendGrid template

With a SendGrid account created and SendGrid API key stored in a Azure AD B2C policy key, create a SendGrid [dynamic transactional template](#).

1. On the SendGrid site, open the [transactional templates](#) page and select **Create Template**.

2. Enter a unique template name like **Verification email** and then select **Save**.
3. To begin editing your new template, select **Add Version**.
4. Select **Code Editor** and then **Continue**.
5. In the HTML editor, paste following HTML template or use your own. The **{{otp}}** and **{{email}}** parameters will be replaced dynamically with the one-time password value and the user email address.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" lang="en"><head id="Head1">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"><title>Contoso demo account email verification code</title><meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<!-- Template B 0365 -->
<style>
    table td {border-collapse:collapse; margin:0; padding:0;}
</style>
</head>
<body dir="ltr" lang="en">
    <table width="100%" cellpadding="0" cellspacing="0" border="0" dir="ltr" lang="en">
        <tr>
            <td valign="top" width="50%"></td>
            <td valign="top">
                <!-- Email Header -->
                <table width="640" cellpadding="0" cellspacing="0" border="0" dir="ltr" lang="en"
style="border-left:1px solid #e3e3e3; border-right: 1px solid #e3e3e3;">
                    <tr style="background-color: #0072c6;">
                        <td width="1" style="background:#0072c6; border-top:1px solid #e3e3e3;"></td>
                        <td width="24" style="border-top:1px solid #e3e3e3; border-bottom:1px solid
#e3e3e3;">&ampnbsp</td>
                        <td width="310" valign="middle" style="border-top:1px solid #e3e3e3; border-
bottom:1px solid #e3e3e3; padding:12px 0;">
                            <h1 style="line-height:20pt;font-family:Segoe UI Light; font-size:18pt;
color:#ffffff; font-weight:normal;">
                                <span id="HeaderPlaceholder_UserVerificationEmailHeader"><font
color="#FFFFFF">Verify your email address</font></span>
                            </h1>
                        </td>
                        <td width="24" style="border-top: 1px solid #e3e3e3; border-bottom: 1px solid
#e3e3e3;">&ampnbsp</td>
                    </tr>
                </table>
                <!-- Email Content -->
                <table width="640" cellpadding="0" cellspacing="0" border="0" dir="ltr" lang="en">
                    <tr>
                        <td width="1" style="background:#e3e3e3;"></td>
                        <td width="24">&ampnbsp</td>
                        <td id="PageBody" width="640" valign="top" colspan="2" style="border-bottom:1px solid
#e3e3e3; padding:10px 0 20px; border-bottom-style:hidden;">
                            <table cellpadding="0" cellspacing="0" border="0">
                                <tr>
                                    <td width="630" style="font-size:10pt; line-height:13pt; color:#000;">
                                        <table cellpadding="0" cellspacing="0" border="0" width="100%
style="" dir="ltr" lang="en">
                                            <tr>
                                                <td>
                                                    <div style="font-family:'Segoe UI', Tahoma, sans-serif; font-size:14px; color:#333;">
                                                        <span id="BodyPlaceholder_UserVerificationEmailBodySentence1">Thanks for verifying your {{email}}
account!</span>
                                                    </div>
                                                    <br>
                                                    <div style="font-family:'Segoe UI', Tahoma, sans-serif; font-size:14px; color:#333; font-weight:
bold">
                                                        <span id="BodyPlaceholder_UserVerificationEmailBodySentence2">Your code is: {{otp}}</span>
                                                    </div>
                                                </td>
                                            </tr>
                                        </table>
                                    </td>
                                </tr>
                            </table>
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
    </table>
</body>

```

```

</div>
<br>
<br>

<div style="font-family:'Segoe UI', Tahoma, sans-serif;
font-size:14px; color:#333;">
    Sincerely,
</div>
<div style="font-family:'Segoe UI', Tahoma, sans-serif;
font-size:14px; font-style:italic; color:#333;">
    Contoso
</div>
</td>
</tr>
</table>
</td>
</tr>
</table>

</td>

<td width="1">&nbsp;</td>
<td width="1"></td>
<td width="1">&nbsp;</td>
<td width="1" valign="top"></td>
<td width="29">&nbsp;</td>
<td width="1" style="background:#e3e3e3;"></td>
</tr>
<tr>
    <td width="1" style="background:#e3e3e3; border-bottom:1px solid #e3e3e3;"></td>
    <td width="24" style="border-bottom:1px solid #e3e3e3;">&nbsp;</td>
    <td id="PageFooterContainer" width="585" valign="top" colspan="6" style="border-
bottom:1px solid #e3e3e3;padding:0px;">
        </td>
        <td width="29" style="border-bottom:1px solid #e3e3e3;">&nbsp;</td>
        <td width="1" style="background:#e3e3e3; border-bottom:1px solid #e3e3e3;"></td>
    </tr>
    </table>

    </td>
    <td valign="top" width="50%"></td>
</tr>
</table>
</body>
</html>

```

6. Expand **Settings** on the left, and for **Email Subject**, enter `{}{subject}.`
7. Select **Save Template**.
8. Return to the **Transactional Templates** page by selecting the back arrow.
9. Record the **ID** of template you created for use in a later step. For example, `d-989077fbba9746e89f3f6411f596fb96`. You specify this ID when you [add the claims transformation](#).

Add Azure AD B2C claim types

In your policy, add the following claim types to the `<ClaimsSchema>` element within `<BuildingBlocks>`.

These claims types are necessary to generate and verify the email address using a one-time password (OTP) code.

```
<ClaimType Id="Otp">
  <DisplayName>Secondary One-time password</DisplayName>
  <DataType>string</DataType>
</ClaimType>
<ClaimType Id="sendGridReqBody">
  <DisplayName>SendGrid request body</DisplayName>
  <DataType>string</DataType>
</ClaimType>
<ClaimType Id="VerificationCode">
  <DisplayName>Secondary Verification Code</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Enter your email verification code</UserHelpText>
  <UserInputType>TextBox</UserInputType>
</ClaimType>
```

Add the claims transformation

Next, you need a claims transformation to output a JSON string claim that will be the body of the request sent to SendGrid.

The JSON object's structure is defined by the IDs in dot notation of the InputParameters and the TransformationClaimTypes of the InputClaims. Numbers in the dot notation imply arrays. The values come from the InputClaims' values and the InputParameters' "Value" properties. For more information about JSON claims transformations, see [JSON claims transformations](#).

Add the following claims transformation to the `<ClaimsTransformations>` element within `<BuildingBlocks>`. Make the following updates to the claims transformation XML:

- Update the `template_id` InputParameter value with the ID of the SendGrid transactional template you created earlier in [Create SendGrid template](#).
- Update the `from.email` address value. Use a valid email address to help prevent the verification email from being marked as spam.
- Update the value of the `personalizations.0.dynamic_template_data.subject` subject line input parameter with a subject line appropriate for your organization.

```
<ClaimsTransformation Id="GenerateSendGridRequestBody" TransformationMethod="GenerateJson">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="personalizations.0.to.0.email" />
    <InputClaim ClaimTypeReferenceId="otp" TransformationClaimType="personalizations.0.dynamic_template_data.otp" />
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="personalizations.0.dynamic_template_data.email" />
  </InputClaims>
  <InputParameters>
    <!-- Update the template_id value with the ID of your SendGrid template. -->
    <InputParameter Id="template_id" DataType="string" Value="d-989077fbb9746e89f3f6411f596fb96"/>
    <InputParameter Id="from.email" DataType="string" Value="my_email@mydomain.com"/>
    <!-- Update with a subject line appropriate for your organization. -->
    <InputParameter Id="personalizations.0.dynamic_template_data.subject" DataType="string" Value="Contoso account email verification code"/>
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="sendGridReqBody" TransformationClaimType="outputClaim"/>
  </OutputClaims>
</ClaimsTransformation>
```

Add DataUri content definition

Below the claims transformations within `<BuildingBlocks>`, add the following [ContentDefinition](#) to reference the version 2.0.0 data URI:

```
<ContentDefinitions>
<ContentDefinition Id="api.localaccounts signup">
    <DataUri>urn:com:microsoft:aad:b2c:elements:contract:selfasserted:2.0.0</DataUri>
</ContentDefinition>
</ContentDefinitions>
```

Create a DisplayControl

A verification display control is used to verify the email address with a verification code that's sent to the user.

This example display control is configured to:

1. Collect the `email` address claim type from the user.
2. Wait for the user to provide the `verificationCode` claim type with the code sent to the user.
3. Return the `email` back to the self-asserted technical profile that has a reference to this display control.
4. Using the `SendCode` action, generate an OTP code and send an email with the OTP code to the user.



Under content definitions, still within `<BuildingBlocks>`, add the following [DisplayControl](#) of type [VerificationControl](#) to your policy.

```
<DisplayControls>
<DisplayControl Id="emailVerificationControl" UserInterfaceControlType="VerificationControl">
    <DisplayClaims>
        <DisplayClaim ClaimTypeReferenceId="email" Required="true" />
        <DisplayClaim ClaimTypeReferenceId="verificationCode" ControlClaimType="VerificationCode" Required="true" />
    />
    </DisplayClaims>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="email" />
    </OutputClaims>
    <Actions>
        <Action Id="SendCode">
            <ValidationClaimsExchange>
                <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="GenerateOtp" />
                <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="SendGrid" />
            </ValidationClaimsExchange>
        </Action>
        <Action Id="VerifyCode">
            <ValidationClaimsExchange>
                <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="VerifyOtp" />
            </ValidationClaimsExchange>
        </Action>
    </Actions>
</DisplayControl>
</DisplayControls>
```

Add OTP technical profiles

The `GenerateOtp` technical profile generates a code for the email address. The `VerifyOtp` technical profile verifies the code associated with the email address. You can change the configuration of the format and the expiration of the one-time password. For more information about OTP technical profiles, see [Define a one-time password technical profile](#).

Add the following technical profiles to the `<ClaimsProviders>` element.

```
<ClaimsProvider>
  <DisplayName>One time password technical profiles</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="GenerateOtp">
      <DisplayName>Generate one time password</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.OneTimePasswordProtocolProvider,
      Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="Operation">GenerateCode</Item>
        <Item Key="CodeExpirationInSeconds">1200</Item>
        <Item Key="CodeLength">6</Item>
        <Item Key="CharacterSet">0-9</Item>
        <Item Key="ReuseSameCode">true</Item>
        <Item Key="MaxNumAttempts">5</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="identifier" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="otp" PartnerClaimType="otpGenerated" />
      </OutputClaims>
    </TechnicalProfile>

    <TechnicalProfile Id="VerifyOtp">
      <DisplayName>Verify one time password</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.OneTimePasswordProtocolProvider,
      Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="Operation">VerifyCode</Item>
        <Item Key="UserMessage.VerificationHasExpired">You have exceed the maximum time allowed.</Item>
        <Item Key="UserMessage.MaxRetryAttempted">You have exceed the number of retries allowed.</Item>
        <Item Key="UserMessage.InvalidCode">You have entered the wrong code.</Item>
        <Item Key="UserMessage.ServerError">Cannot verify the code, please try again later.</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" PartnerClaimType="identifier" />
        <InputClaim ClaimTypeReferenceId="verificationCode" PartnerClaimType="otpToVerify" />
      </InputClaims>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

Add a REST API technical profile

This REST API technical profile generates the email content (using the SendGrid format). For more information about RESTful technical profiles, see [Define a RESTful technical profile](#).

As with the OTP technical profiles, add the following technical profiles to the `<ClaimsProviders>` element.

```
<ClaimsProvider>
  <DisplayName>RestfulProvider</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="SendGrid">
      <DisplayName>Use SendGrid's email API to send the code the user</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://api.sendgrid.com/v3/mail/send</Item>
        <Item Key="AuthenticationType">Bearer</Item>
        <Item Key="SendClaimsIn">Body</Item>
        <Item Key="ClaimUsedForRequestPayload">sendGridReqBody</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="BearerAuthenticationToken" StorageReferenceId="B2C_1A_SendGridSecret" />
      </CryptographicKeys>
      <InputClaimsTransformations>
        <InputClaimsTransformation ReferenceId="GenerateSendGridRequestBody" />
      </InputClaimsTransformations>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="sendGridReqBody" />
      </InputClaims>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

Make a reference to the DisplayControl

In the final step, add a reference to the DisplayControl you created. Replace your existing

`LocalAccountSignUpWithLogonEmail` self-asserted technical profile with the following if you used an earlier version of Azure AD B2C policy. This technical profile uses `DisplayClaims` with a reference to the DisplayControl.

For more information, see [Self-asserted technical profile](#) and [DisplayControl](#).

```

<ClaimsProvider>
  <DisplayName>Local Account</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
      <DisplayName>Email signup</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="IpAddressClaimReferenceId">IpAddress</Item>
        <Item Key="ContentDefinitionReferenceId">api.localaccounts signup</Item>
        <Item Key="language.button_continue">Create</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" />
      </InputClaims>
      <DisplayClaims>
        <DisplayClaim DisplayControlReferenceId="emailVerificationControl" />
        <DisplayClaim ClaimTypeReferenceId="displayName" Required="true" />
        <DisplayClaim ClaimTypeReferenceId="givenName" Required="true" />
        <DisplayClaim ClaimTypeReferenceId="surName" Required="true" />
        <DisplayClaim ClaimTypeReferenceId="newPassword" Required="true" />
        <DisplayClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
      </DisplayClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="email" Required="true" />
        <OutputClaim ClaimTypeReferenceId="objectId" />
        <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" />
        <OutputClaim ClaimTypeReferenceId="newUser" />
      </OutputClaims>
      <ValidationTechnicalProfiles>
        <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingLogonEmail" />
      </ValidationTechnicalProfiles>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

[Optional] Localize your email

To localize the email, you must send localized strings to SendGrid, or your email provider. For example to localize the email subject, body, your code message, or signature of the email. To do so, you can use the

[GetLocalizedStringsTransformation](#) claims transformation to copy localized strings into claim types. In the

`GenerateSendGridRequestBody` claims transformation, which generates the JSON payload, uses input claims that contain the localized strings.

1. In your policy define the following string claims: subject, message, codeIntro and signature.
2. Define a [GetLocalizedStringsTransformation](#) claims transformation to substitute localized string values into the claims from step 1.
3. Change the `GenerateSendGridRequestBody` claims transformation to use input claims with the following XML snippet.
4. Update your SendGrid template to use dynamic parameters in place of all the strings which will be localized by Azure AD B2C.

```
<ClaimsTransformation Id="GenerateSendGridRequestBody" TransformationMethod="GenerateJson">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="personalizations.0.to.0.email" />
    <InputClaim ClaimTypeReferenceId="subject"
      TransformationClaimType="personalizations.0.dynamic_template_data.subject" />
    <InputClaim ClaimTypeReferenceId="otp"
      TransformationClaimType="personalizations.0.dynamic_template_data.otp" />
    <InputClaim ClaimTypeReferenceId="email"
      TransformationClaimType="personalizations.0.dynamic_template_data.email" />
    <InputClaim ClaimTypeReferenceId="message"
      TransformationClaimType="personalizations.0.dynamic_template_data.message" />
    <InputClaim ClaimTypeReferenceId="codeIntro"
      TransformationClaimType="personalizations.0.dynamic_template_data.codeIntro" />
    <InputClaim ClaimTypeReferenceId="signature"
      TransformationClaimType="personalizations.0.dynamic_template_data.signature" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="template_id" DataType="string" Value="d-1234567890" />
    <InputParameter Id="from.email" DataType="string" Value="my_email@mydomain.com" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="sendGridReqBody" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Next steps

You can find an example of a custom email verification policy on GitHub:

[Custom email verification - DisplayControls](#)

For information about using a custom REST API or any HTTP-based SMTP email provider, see [Define a RESTful technical profile in an Azure AD B2C custom policy](#).

JavaScript samples for use in Azure Active Directory B2C

2/11/2020 • 3 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

You can add your own JavaScript client-side code to your Azure Active Directory B2C (Azure AD B2C) applications.

To enable JavaScript for your applications:

- Add an element to your [custom policy](#)
- Select a [page layout](#)
- Use [b2clogin.com](#) in your requests

This article describes how you can change your custom policy to enable script execution.

NOTE

If you want to enable JavaScript for user flows, see [JavaScript and page layout versions in Azure Active Directory B2C](#).

Prerequisites

Select a page layout

- Select a [page layout](#) for the user interface elements of your application.

If you intend to use JavaScript, you need to [define a page layout version](#) with page `contract` version for *all* of the content definitions in your custom policy.

Add the ScriptExecution element

You enable script execution by adding the **ScriptExecution** element to the **RelyingParty** element.

1. Open your custom policy file. For example, *SignUpOrSignin.xml*.
2. Add the **ScriptExecution** element to the **UserJourneyBehaviors** element of **RelyingParty**:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="B2CSignUpOrSignInWithPassword" />
  <UserJourneyBehaviors>
    <ScriptExecution>Allow</ScriptExecution>
  </UserJourneyBehaviors>
  ...
</RelyingParty>
```

3. Save and upload the file.

Guidelines for using JavaScript

Follow these guidelines when you customize the interface of your application using JavaScript:

- Don't bind a click event on `<a>` HTML elements.
- Don't take a dependency on Azure AD B2C code or comments.
- Don't change the order or hierarchy of Azure AD B2C HTML elements. Use an Azure AD B2C policy to control the order of the UI elements.
- You can call any RESTful service with these considerations:
 - You may need to set your RESTful service CORS to allow client-side HTTP calls.
 - Make sure your RESTful service is secure and uses only the HTTPS protocol.
 - Don't use JavaScript directly to call Azure AD B2C endpoints.
- You can embed your JavaScript or you can link to external JavaScript files. When using an external JavaScript file, make sure to use the absolute URL and not a relative URL.
- JavaScript frameworks:
 - Azure AD B2C uses a specific version of jQuery. Don't include another version of jQuery. Using more than one version on the same page causes issues.
 - Using RequireJS isn't supported.
 - Most JavaScript frameworks are not supported by Azure AD B2C.
- Azure AD B2C settings can be read by calling `window.SETTINGS`, `window.CONTENT` objects, such as the current UI language. Don't change the value of these objects.
- To customize the Azure AD B2C error message, use localization in a policy.
- If anything can be achieved by using a policy, generally it's the recommended way.

JavaScript samples

Show or hide a password

A common way to help your customers with their sign-up success is to allow them to see what they've entered as their password. This option helps users sign up by enabling them to easily see and make corrections to their password if needed. Any field of type password has a checkbox with a **Show password** label. This enables the user to see the password in plain text. Include this code snippet into your sign-up or sign-in template for a self-asserted page:

```

function makePwdToggler(pwd){
    // Create show-password checkbox
    var checkbox = document.createElement('input');
    checkbox.setAttribute('type', 'checkbox');
    var id = pwd.id + 'toggler';
    checkbox.setAttribute('id', id);

    var label = document.createElement('label');
    label.setAttribute('for', id);
    label.appendChild(document.createTextNode('show password'));

    var div = document.createElement('div');
    div.appendChild(checkbox);
    div.appendChild(label);

    // Add show-password checkbox under password input
    pwd.insertAdjacentElement('afterend', div);

    // Add toggle password callback
    function toggle(){
        if(pwd.type === 'password'){
            pwd.type = 'text';
        } else {
            pwd.type = 'password';
        }
    }
    checkbox.onclick = toggle;
    // For non-mouse usage
    checkbox.onkeydown = toggle;
}

function setupPwdTogglers(){
    var pwdInputs = document.querySelectorAll('input[type=password]');
    for (var i = 0; i < pwdInputs.length; i++) {
        makePwdToggler(pwdInputs[i]);
    }
}

setupPwdTogglers();

```

Add terms of use

Include the following code into your page where you want to include a **Terms of Use** checkbox. This checkbox is typically needed in your local account sign-up and social account sign-up pages.

```
function addTermsOfUseLink() {
    // find the terms of use label element
    var termsOfUseLabel = document.querySelector('#api_label[for="termsOfUse"]');
    if (!termsOfUseLabel) {
        return;
    }

    // get the label text
    var termsLabelText = termsOfUseLabel.innerHTML;

    // create a new <a> element with the same inner text
    var termsOfUseUrl = 'https://docs.microsoft.com/legal/termsofuse';
    var termsOfUseLink = document.createElement('a');
    termsOfUseLink.setAttribute('href', termsOfUseUrl);
    termsOfUseLink.setAttribute('target', '_blank');
    termsOfUseLink.appendChild(document.createTextNode(termsLabelText));

    // replace the label text with the new element
    termsOfUseLabel.replaceChild(termsOfUseLink, termsOfUseLabel.firstChild);
}
```

In the code, replace `termsOfUseUrl` with the link to your terms of use agreement. For your directory, create a new user attribute called **termsOfUse** and then include **termsOfUse** as a user attribute.

Next steps

Find more information about how you can customize the user interface of your applications in [Customize the user interface of your application using a custom policy in Azure Active Directory B2C](#).

Configure password complexity using custom policies in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In Azure Active Directory B2C (Azure AD B2C), you can configure the complexity requirements for passwords that are provided by a user when creating an account. By default, Azure AD B2C uses **Strong** passwords. This article shows you how to configure password complexity in [custom policies](#). It's also possible to configure password complexity in [user flows](#).

Prerequisites

Complete the steps in [Get started with custom policies in Active Directory B2C](#).

Add the elements

1. Copy the *SignUpOrSignIn.xml* file that you downloaded with the starter pack and name it *SignUpOrSignInPasswordComplexity.xml*.
2. Open the *SignUpOrSignInPasswordComplexity.xml* file and change the **PolicyId** and the **PublicPolicyUri** to a new policy name. For example, *B2C_1A_signup_signin_password_complexity*.
3. Add the following **ClaimType** elements with identifiers of `newPassword` and `reenterPassword`:

```
<ClaimsSchema>
  <ClaimType Id="newPassword">
    <InputValidationReference Id="PasswordValidation" />
  </ClaimType>
  <ClaimType Id="reenterPassword">
    <InputValidationReference Id="PasswordValidation" />
  </ClaimType>
</ClaimsSchema>
```

4. **Predicates** have method types of `IsLengthRange` or `MatchesRegex`. The `MatchesRegex` type is used to match a regular expression. The `IsLengthRange` type takes a minimum and maximum string length. Add a **Predicates** element to the **BuildingBlocks** element if it doesn't exist with the following **Predicate** elements:

```

<Predicates>
  <Predicate Id="PIN" Method="MatchesRegex" HelpText="The password must be a pin.">
    <Parameters>
      <Parameter Id="RegularExpression">^[0-9]+$</Parameter>
    </Parameters>
  </Predicate>
  <Predicate Id="Length" Method="IsLengthRange" HelpText="The password must be between 8 and 16 characters.">
    <Parameters>
      <Parameter Id="Minimum">8</Parameter>
      <Parameter Id="Maximum">16</Parameter>
    </Parameters>
  </Predicate>
</Predicates>

```

5. Each **InputValidation** element is constructed by using the defined **Predicate** elements. This element allows you to perform boolean aggregations that are similar to **and** and **or**. Add an **InputValidations** element to the **BuildingBlocks** element if it doesn't exist with the following **InputValidation** element:

```

<InputValidations>
  <InputValidation Id="PasswordValidation">
    <PredicateReferences Id="LengthGroup" MatchAtLeast="1">
      <PredicateReference Id="Length" />
    </PredicateReferences>
    <PredicateReferences Id="3of4" MatchAtLeast="3" HelpText="You must have at least 3 of the following character classes:">
      <PredicateReference Id="Lowercase" />
      <PredicateReference Id="Uppercase" />
      <PredicateReference Id="Number" />
      <PredicateReference Id="Symbol" />
    </PredicateReferences>
  </InputValidation>
</InputValidations>

```

6. Make sure that the **PolicyProfile** technical profile contains the following elements:

```

<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn"/>
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect"/>
    <InputClaims>
      <InputClaim ClaimTypeReferenceId="passwordPolicies" DefaultValue="DisablePasswordExpiration, DisableStrongPassword"/>
    </InputClaims>
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName"/>
      <OutputClaim ClaimTypeReferenceId="givenName"/>
      <OutputClaim ClaimTypeReferenceId="surname"/>
      <OutputClaim ClaimTypeReferenceId="email"/>
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub"/>
  </TechnicalProfile>
</RelyingParty>

```

7. Save the policy file.

Test your policy

When testing your applications in Azure AD B2C, it can be useful to have the Azure AD B2C token returned to

<https://jwt.ms> to be able to review the claims in it.

Upload the files

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **Identity Experience Framework**.
5. On the Custom Policies page, click **Upload Policy**.
6. Select **Overwrite the policy if it exists**, and then search for and select the *SingUpOrSignInPasswordComplexity.xml* file.
7. Click **Upload**.

Run the policy

1. Open the policy that you changed. For example, *B2C_1A_signup_signin_password_complexity*.
2. For **Application**, select your application that you previously registered. To see the token, the **Reply URL** should show <https://jwt.ms>.
3. Click **Run now**.
4. Select **Sign up now**, enter an email address, and enter a new password. Guidance is presented on password restrictions. Finish entering the user information, and then click **Create**. You should see the contents of the token that was returned.

Next steps

- Learn how to [Configure password change using custom policies in Azure Active Directory B2C](#).

Set up sign-in with an Amazon account using custom policies in Azure Active Directory B2C

1/28/2020 • 6 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users from an Amazon account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

- Complete the steps in [Get started with custom policies](#).
- If you don't already have an Amazon account, create one at <https://www.amazon.com/>.

Register the application

To enable sign-in for users from an Amazon account, you need to create an Amazon application.

1. Sign in to the [Amazon Developer Center](#) with your Amazon account credentials.
2. If you have not already done so, click **Sign Up**, follow the developer registration steps, and accept the policy.
3. Select **Register new application**.
4. Enter a **Name**, **Description**, and **Privacy Notice URL**, and then click **Save**. The privacy notice is a page that you manage that provides privacy information to users.
5. In the **Web Settings** section, copy the values of **Client ID**. Select **Show Secret** to get the client secret and then copy it. You need both of them to configure an Amazon account as an identity provider in your tenant. **Client Secret** is an important security credential.
6. In the **Web Settings** section, select **Edit**, and then enter `https://your-tenant-name.b2clogin.com` in **Allowed JavaScript Origins** and `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in **Allowed Return URLs**. Replace `your-tenant-name` with the name of your tenant. Use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
7. Click **Save**.

Create a policy key

You need to store the client secret that you previously recorded in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.
6. For **Options**, choose `Manual`.

7. Enter a **Name** for the policy key. For example, `AmazonSecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter your client secret that you previously recorded.
9. For **Key usage**, select `Signature`.
10. Click **Create**.

Add a claims provider

If you want users to sign in by using an Amazon account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define an Amazon account as a claims provider by adding it to the **ClaimsProviders** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml`.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```
<ClaimsProvider>
  <Domain>amazon.com</Domain>
  <DisplayName>Amazon</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Amazon-OAUTH">
      <DisplayName>Amazon</DisplayName>
      <Protocol Name="OAuth2" />
      <Metadata>
        <Item Key="ProviderName">amazon</Item>
        <Item Key="authorization_endpoint">https://www.amazon.com/ap/oa</Item>
        <Item Key="AccessTokenEndpoint">https://api.amazon.com/auth/o2/token</Item>
        <Item Key="ClaimsEndpoint">https://api.amazon.com/user/profile</Item>
        <Item Key="scope">profile</Item>
        <Item Key="HttpBinding">POST</Item>
        <Item Key="UsePolicyInRedirectUri">0</Item>
        <Item Key="client_id">Your Amazon application client ID</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="client_secret" StorageReferenceId="B2C_1A_AmazonSecret" />
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="user_id" />
        <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
        <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="amazon.com" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

4. Set **client_id** to the application ID from the application registration.
5. Save the file.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Azure AD directory. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up/sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the Amazon identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInAmazon`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up/sign-in screen. If you add a **ClaimsProviderSelection** element for an Amazon account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `AmazonExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AmazonExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with an Amazon account to receive a token.

1. Find the **OrchestrationStep** that includes `Order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for the ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="AmazonExchange" TechnicalProfileReferenceId="Amazon-OAuth" />
```

Update the value of **TechnicalProfileReferenceId** to the ID of the technical profile you created earlier. For example, `Amazon-OAuth`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInAmazon.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInAmazon`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_amazon`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (*SignUpSignInAmazon*).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Set up sign-in with an Azure Active Directory account using custom policies in Azure Active Directory B2C

2/11/2020 • 9 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users from an Azure Active Directory (Azure AD) organization by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).

Register an application

To enable sign-in for users from a specific Azure AD organization, you need to register an application within the organizational Azure AD tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your organizational Azure AD tenant (for example, contoso.com). Select the **Directory + subscription filter** in the top menu, and then choose the directory that contains your Azure AD tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
4. Select **New registration**.
5. Enter a **Name** for your application. For example, `Azure AD B2C App`.
6. Accept the default selection of **Accounts in this organizational directory only** for this application.
7. For the **Redirect URI**, accept the value of **Web**, and enter the following URL in all lowercase letters, where `your-B2C-tenant-name` is replaced with the name of your Azure AD B2C tenant.

```
https://your-B2C-tenant-name.b2clogin.com/your-B2C-tenant-name.onmicrosoft.com/oauth2/authresp
```

For example, `https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp`.

8. Select **Register**. Record the **Application (client) ID** for use in a later step.
9. Select **Certificates & secrets**, and then select **New client secret**.
10. Enter a **Description** for the secret, select an expiration, and then select **Add**. Record the **Value** of the secret for use in a later step.

Configuring optional claims

If you want to get the `family_name` and `given_name` claims from Azure AD, you can configure optional claims for your application in the Azure portal UI or application manifest. For more information, see [How to provide optional claims to your Azure AD app](#).

1. Sign in to the [Azure portal](#). Search for and select **Azure Active Directory**.
2. From the **Manage** section, select **App registrations**.
3. Select the application you want to configure optional claims for in the list.
4. From the **Manage** section, select **Token configuration (preview)**.
5. Select **Add optional claim**.
6. Select the token type you want to configure.
7. Select the optional claims to add.
8. Click **Add**.

Create a policy key

You need to store the application key that you created in your Azure AD B2C tenant.

1. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription filter** in the top menu, and then choose the directory that contains your Azure AD B2C tenant.
2. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
3. Under **Policies**, select **Identity Experience Framework**.
4. Select **Policy keys** and then select **Add**.
5. For **Options**, choose `Manual`.
6. Enter a **Name** for the policy key. For example, `ContosoAppSecret`. The prefix `B2C_1A_` is added automatically to the name of your key when it's created, so its reference in the XML in following section is to `B2C_1A_ContosoAppSecret`.
7. In **Secret**, enter your client secret that you recorded earlier.
8. For **Key usage**, select `signature`.
9. Select **Create**.

Add a claims provider

If you want users to sign in by using Azure AD, you need to define Azure AD as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define Azure AD as a claims provider by adding Azure AD to the **ClaimsProvider** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml` file.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```

<ClaimsProvider>
  <Domain>Contoso</Domain>
  <DisplayName>Login using Contoso</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="OIDC-Contoso">
      <DisplayName>Contoso Employee</DisplayName>
      <Description>Login with your Contoso account</Description>
      <Protocol Name="OpenIdConnect"/>
      <Metadata>
        <Item Key="METADATA">https://login.microsoftonline.com/tenant-name.onmicrosoft.com/v2.0/.well-known/openid-configuration</Item>
        <Item Key="client_id">00000000-0000-0000-000000000000</Item>
        <Item Key="response_types">code</Item>
        <Item Key="scope">openid profile</Item>
        <Item Key="response_mode">form_post</Item>
        <Item Key="HttpBinding">POST</Item>
        <Item Key="UsePolicyInRedirectUri">false</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="client_secret" StorageReferenceId="B2C_1A_ContosoAppSecret"/>
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="oid"/>
        <OutputClaim ClaimTypeReferenceId="tenantId" PartnerClaimType="tid"/>
        <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
        <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="family_name" />
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" AlwaysUseDefaultValue="true" />
        <OutputClaim ClaimTypeReferenceId="identityProvider" PartnerClaimType="iss" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin"/>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

4. Under the **ClaimsProvider** element, update the value for **Domain** to a unique value that can be used to distinguish it from other identity providers. For example `Contoso`. You don't put a `.com` at the end of this domain setting.
5. Under the **ClaimsProvider** element, update the value for **DisplayName** to a friendly name for the claims provider. This value is not currently used.

Update the technical profile

To get a token from the Azure AD endpoint, you need to define the protocols that Azure AD B2C should use to communicate with Azure AD. This is done inside the **TechnicalProfile** element of **ClaimsProvider**.

1. Update the ID of the **TechnicalProfile** element. This ID is used to refer to this technical profile from other parts of the policy, for example `OIDC-Contoso`.
2. Update the value for **DisplayName**. This value will be displayed on the sign-in button on your sign-in screen.
3. Update the value for **Description**.
4. Azure AD uses the OpenID Connect protocol, so make sure that the value for **Protocol** is `OpenIdConnect`.
5. Set value of the **METADATA** to

<https://login.microsoftonline.com/tenant-name.onmicrosoft.com/v2.0/.well-known/openid-configuration>, where `tenant-name` is your Azure AD tenant name. For example,

<https://login.microsoftonline.com/contoso.onmicrosoft.com/v2.0/.well-known/openid-configuration>

6. Set **client_id** to the application ID from the application registration.
7. Under **CryptographicKeys**, update the value of **StorageReferenceId** to the name of the policy key that you created earlier. For example, `B2C_1A_ContosoAppSecret`.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Azure AD directory. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not yet available in any of the sign-up/sign-in pages. To make it available, create a duplicate of an existing template user journey, and then modify it so that it also has the Azure AD identity provider:

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInContoso`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up/sign-in page. If you add a **ClaimsProviderSelection** element for Azure AD, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created in *TrustFrameworkExtensions.xml*.
2. Under **ClaimsProviderSelections**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `ContosoExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="ContosoExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with Azure AD to receive a token. Link the button to an action by linking the technical profile for your Azure AD claims provider:

1. Find the **OrchestrationStep** that includes `Order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for **Id** that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="ContosoExchange" TechnicalProfileReferenceId="OIDC-Contoso" />
```

Update the value of **TechnicalProfileReferenceId** to the **Id** of the technical profile you created earlier. For

example, `OIDC-Contoso`.

- Save the `TrustFrameworkExtensions.xml` file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, `testapp 1`.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of `SignUpOrSignIn.xml` in your working directory, and rename it. For example, rename it to `SignUpSignInContoso.xml`.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInContoso`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_contoso`.
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the user journey that you created earlier. For example, `SignUpSignInContoso`.
5. Save your changes and upload the file.
6. Under **Custom policies**, select the new policy in the list.
7. In the **Select application** drop-down, select the Azure AD B2C application that you created earlier. For example, `testapp 1`.
8. Copy the **Run now endpoint** and open it in a private browser window, for example, Incognito Mode in Google Chrome or an InPrivate window in Microsoft Edge. Opening in a private browser window allows you to test the full user journey by not using any currently cached Azure AD credentials.
9. Select the Azure AD sign in button, for example, `Contoso Employee`, and then enter the credentials for a user in your Azure AD organizational tenant. You're asked to authorize the application, and then enter information for your profile.

If the sign in process is successful, your browser is redirected to `https://jwt.ms`, which displays the contents of the token returned by Azure AD B2C.

Next steps

When working with custom policies, you might sometimes need additional information when troubleshooting a policy during its development.

To help diagnose issues, you can temporarily put the policy into "developer mode" and collect logs with Azure Application Insights. Find out how in [Azure Active Directory B2C: Collecting Logs](#).

Set up sign-in for multi-tenant Azure Active Directory using custom policies in Azure Active Directory B2C

2/9/2020 • 10 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users using the multi-tenant endpoint for Azure Active Directory (Azure AD) by using [custom policies](#) in Azure AD B2C. This allows users from multiple Azure AD tenants to sign in using Azure AD B2C, without you having to configure an identity provider for each tenant. However, guest members in any of these tenants **will not** be able to sign in. For that, you need to [individually configure each tenant](#).

Prerequisites

Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).

Register an application

To enable sign-in for users from a specific Azure AD organization, you need to register an application within the organizational Azure AD tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your organizational Azure AD tenant (for example, contoso.com). Select the **Directory + subscription filter** in the top menu, and then choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
4. Select **New registration**.
5. Enter a **Name** for your application. For example, `Azure AD B2C App`.
6. Select **Accounts in any organizational directory** for this application.
7. For the **Redirect URI**, accept the value of **Web**, and enter the following URL in all lowercase letters, where `your-B2C-tenant-name` is replaced with the name of your Azure AD B2C tenant.

```
https://your-B2C-tenant-name.b2clogin.com/your-B2C-tenant-name.onmicrosoft.com/oauth2/authresp
```
- For example, `https://contoso.b2clogin.com/contoso.onmicrosoft.com/oauth2/authresp`.
8. Select **Register**. Record the **Application (client) ID** for use in a later step.
9. Select **Certificates & secrets**, and then select **New client secret**.
10. Enter a **Description** for the secret, select an expiration, and then select **Add**. Record the **Value** of the secret

for use in a later step.

Configuring optional claims

If you want to get the `family_name` and `given_name` claims from Azure AD, you can configure optional claims for your application in the Azure portal UI or application manifest. For more information, see [How to provide optional claims to your Azure AD app](#).

1. Sign in to the [Azure portal](#). Search for and select **Azure Active Directory**.
2. From the **Manage** section, select **App registrations**.
3. Select the application you want to configure optional claims for in the list.
4. From the **Manage** section, select **Token configuration (preview)**.
5. Select **Add optional claim**.
6. Select the token type you want to configure.
7. Select the optional claims to add.
8. Click **Add**.

Create a policy key

You need to store the application key that you created in your Azure AD B2C tenant.

1. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription filter** in the top menu, and then choose the directory that contains your Azure AD B2C tenant.
2. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
3. Under **Policies**, select **Identity Experience Framework**.
4. Select **Policy keys** and then select **Add**.
5. For **Options**, choose `Manual`.
6. Enter a **Name** for the policy key. For example, `AADAppSecret`. The prefix `B2C_1A_` is added automatically to the name of your key when it's created, so its reference in the XML in following section is to `B2C_1A_AADAppSecret`.
7. In **Secret**, enter your client secret that you recorded earlier.
8. For **Key usage**, select `Signature`.
9. Select **Create**.

Add a claims provider

If you want users to sign in by using Azure AD, you need to define Azure AD as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define Azure AD as a claims provider by adding Azure AD to the **ClaimsProvider** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml` file.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```

<ClaimsProvider>
    <Domain>commonaad</Domain>
    <DisplayName>Common AAD</DisplayName>
    <TechnicalProfiles>
        <TechnicalProfile Id="Common-AAD">
            <DisplayName>Multi-Tenant AAD</DisplayName>
            <Description>Login with your Contoso account</Description>
            <Protocol Name="OpenIdConnect"/>
            <Metadata>
                <Item Key="METADATA">https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration</Item>
                <!-- Update the Client ID below to the Application ID -->
                <Item Key="client_id">00000000-0000-0000-0000-000000000000</Item>
                <Item Key="response_types">code</Item>
                <Item Key="scope">openid profile</Item>
                <Item Key="response_mode">form_post</Item>
                <Item Key="HttpBinding">POST</Item>
                <Item Key="UsePolicyInRedirectUri">false</Item>
                <Item Key="DiscoverMetadataByTokenIssuer">true</Item>
                <!-- The key below allows you to specify each of the Azure AD tenants that can be used to sign in. Update the GUIDs below for each tenant. -->
                <Item Key="ValidTokenIssuerPrefixes">https://login.microsoftonline.com/00000000-0000-0000-0000-000000000000,https://login.microsoftonline.com/11111111-1111-1111-1111-111111111111</Item>
                <!-- The commented key below specifies that users from any tenant can sign-in. Uncomment if you would like anyone with an Azure AD account to be able to sign in. -->
                <!-- <Item Key="ValidTokenIssuerPrefixes">https://login.microsoftonline.com/</Item> -->
            </Metadata>
            <CryptographicKeys>
                <Key Id="client_secret" StorageReferenceId="B2C_1A_AADAppSecret"/>
            </CryptographicKeys>
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="oid"/>
                <OutputClaim ClaimTypeReferenceId="tenantId" PartnerClaimType="tid"/>
                <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
                <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="family_name" />
                <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
                <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" AlwaysUseDefaultValue="true" />
                <OutputClaim ClaimTypeReferenceId="identityProvider" PartnerClaimType="iss" />
            </OutputClaims>
            <OutputClaimsTransformations>
                <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
                <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
                <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
                <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
            </OutputClaimsTransformations>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin"/>
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>

```

4. Under the **ClaimsProvider** element, update the value for **Domain** to a unique value that can be used to distinguish it from other identity providers.
5. Under the **TechnicalProfile** element, update the value for **DisplayName**, for example, **Contoso Employee**. This value is displayed on the sign-in button on your sign-in page.
6. Set **client_id** to the application ID of the Azure AD multi-tenant application that you registered earlier.
7. Under **CryptographicKeys**, update the value of **StorageReferenceId** to the name of the policy key that created earlier. For example, **B2C_1A_AADAppSecret**.

Restrict access

NOTE

Using `https://login.microsoftonline.com/` as the value for **ValidTokenIssuerPrefixes** allows all Azure AD users to sign in to your application.

You need to update the list of valid token issuers and restrict access to a specific list of Azure AD tenant users who can sign in.

To obtain the values, look at the OpenID Connect discovery metadata for each of the Azure AD tenants that you would like to have users sign in from. The format of the metadata URL is similar to

`https://login.microsoftonline.com/your-tenant/v2.0/.well-known/openid-configuration`, where `your-tenant` is your Azure AD tenant name. For example:

`https://login.microsoftonline.com/fabrikam.onmicrosoft.com/v2.0/.well-known/openid-configuration`

Perform these steps for each Azure AD tenant that should be used to sign in:

1. Open your browser and go to the OpenID Connect metadata URL for the tenant. Find the **issuer** object and record its value. It should look similar to
`https://login.microsoftonline.com/00000000-0000-0000-0000-000000000000/`.
2. Copy and paste the value into the **ValidTokenIssuerPrefixes** key. Separate multiple issuers with a comma. An example with two issuers appears in the previous `claimsProvider` XML sample.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Azure AD directories. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Select **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up/sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the Azure AD identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInContoso`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up/sign-in screen. If you add a **ClaimsProviderSelection** element for Azure AD, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created in *TrustFrameworkExtensions.xml*.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `AzureADEExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="AzureADExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with Azure AD to receive a token. Link the button to an action by linking the technical profile for your Azure AD claims provider.

1. Find the **OrchestrationStep** that includes `order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for **Id** that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="AzureADExchange" TechnicalProfileReferenceId="Common-AAD" />
```

Update the value of **TechnicalProfileReferenceId** to the **Id** of the technical profile you created earlier. For example, `Common-AAD`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 4. Select **Applications**, and then select **Add**.
 5. Enter a name for the application. For example, *testapp1*.
 6. For **Web App / Web API**, select **Yes**.
 7. For **Reply URL**, enter `https://jwt.ms`
 8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created:

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInContoso.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInContoso`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example, `http://contoso.com/B2C_1A_signup_signin_contoso`.
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the user journey that you created earlier. For example, *SignUpSignInContoso*.

5. Save your changes and upload the file.
6. Under **Custom policies**, select the new policy in the list.
7. In the **Select application** drop-down, select the Azure AD B2C application that you created earlier. For example, *testapp1*.
8. Copy the **Run now endpoint** and open it in a private browser window, for example, Incognito Mode in Google Chrome or an InPrivate window in Microsoft Edge. Opening in a private browser window allows you to test the full user journey by not using any currently cached Azure AD credentials.
9. Select the Azure AD sign in button, for example, *Contoso Employee*, and then enter the credentials for a user in one of your Azure AD organizational tenants. You're asked to authorize the application, and then enter information for your profile.

If the sign in process is successful, your browser is redirected to <https://jwt.ms>, which displays the contents of the token returned by Azure AD B2C.

To test the multi-tenant sign-in capability, perform the last two steps using the credentials for a user that exists another Azure AD tenant.

Next steps

When working with custom policies, you might sometimes need additional information when troubleshooting a policy during its development.

To help diagnose issues, you can temporarily put the policy into "developer mode" and collect logs with Azure Application Insights. Find out how in [Azure Active Directory B2C: Collecting Logs](#).

Set up sign-in with a Google account using custom policies in Azure Active Directory B2C

1/28/2020 • 7 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users with a Google account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

- Complete the steps in the [Get started with custom policies in Active Directory B2C](#).
- If you don't already have a Google account, create one at [Create your Google Account](#).

Register the application

To enable sign-in for users from a Google account, you need to create a Google application project.

1. Sign in to the [Google Developers Console](#) with your account credentials.
2. Enter a **Project Name**, click **Create**, and then make sure you are using the new project.
3. Select **Credentials** in the left menu, and then select **Create credentials > Oauth client ID**.
4. Select **Configure consent screen**.
5. Select or specify a valid **Email address**, provide a **Product name** shown to users, enter `b2clogin.com` in **Authorized domains**, and then click **Save**.
6. Under **Application type**, select **Web application**.
7. Enter a **Name** for your application.
8. In **Authorized JavaScript origins**, enter `https://your-tenant-name.b2clogin.com` and in **Authorized redirect URIs**, enter `https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp`. Replace your-tenant-name with the name of your tenant. You need to use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
9. Click **Create**.
10. Copy the values of **Client ID** and **Client secret**. You will need both of them to configure Google as an identity provider in your tenant. Client secret is an important security credential.

Create a policy key

You need to store the client secret that you previously recorded in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.

6. For **Options**, choose `Manual`.
7. Enter a **Name** for the policy key. For example, `GoogleSecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter your client secret that you previously recorded.
9. For **Key usage**, select `Signature`.
10. Click **Create**.

Add a claims provider

If you want users to sign in by using a Google account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define a Google account as a claims provider by adding it to the **ClaimsProviders** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml`.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```
<ClaimsProvider>
  <Domain>google.com</Domain>
  <DisplayName>Google</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Google-OAUTH">
      <DisplayName>Google</DisplayName>
      <Protocol Name="OAuth2" />
      <Metadata>
        <Item Key="ProviderName">google</Item>
        <Item Key="authorization_endpoint">https://accounts.google.com/o/oauth2/auth</Item>
        <Item Key="AccessTokenEndpoint">https://accounts.google.com/o/oauth2/token</Item>
        <Item Key="ClaimsEndpoint">https://www.googleapis.com/oauth2/v1/userinfo</Item>
        <Item Key="scope">email profile</Item>
        <Item Key="HttpBinding">POST</Item>
        <Item Key="UsePolicyInRedirectUri">0</Item>
        <Item Key="client_id">Your Google application ID</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="client_secret" StorageReferenceId="B2C_1A_GoogleSecret" />
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="id" />
        <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
        <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
        <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name" />
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
        <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="google.com" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

4. Set **client_id** to the application ID from the application registration.
5. Save the file.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Azure AD directory. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up/sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the Azure AD identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInGoogle`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up/sign-in screen. If you add a **ClaimsProviderSelection** element for a Google account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `GoogleExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="GoogleExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with a Google account to receive a token.

1. Find the **OrchestrationStep** that includes `Order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="GoogleExchange" TechnicalProfileReferenceId="Google-OAuth" />
```

Update the value of **TechnicalProfileReferenceId** to the ID of the technical profile you created earlier. For example, `Google-OAuth`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInGoogle.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInGoogle`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_google`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (*SignUpSignGoogle*).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Set up sign-in with a LinkedIn account using custom policies in Azure Active Directory B2C

1/28/2020 • 11 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users from a LinkedIn account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

- Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).
- LinkedIn account - If you don't already have one, [create an account](#).
- LinkedIn Page - You need a [LinkedIn Page](#) to associate with the LinkedIn application you create in the next section.

Create an application

To use LinkedIn as an identity provider in Azure AD B2C, you need to create a LinkedIn application.

Create app

1. Sign in to the [LinkedIn application management](#) website with your LinkedIn account credentials.
2. Select **Create app**.
3. Enter an **App name**.
4. Enter a **Company** name corresponding to a LinkedIn page name. Create a LinkedIn Page if you don't already have one.
5. (Optional) Enter a **Privacy policy URL**. It must be a valid URL, but doesn't need to be a reachable endpoint.
6. Enter a **Business email**.
7. Upload an **App logo** image. The logo image must be square and its dimensions must be at least 100x100 pixels.
8. Leave the default settings in the **Products** section.
9. Review the information presented in **Legal terms**. If you agree to the terms, check the box.
10. Select **Create app**.

Configure auth

1. Select the **Auth** tab.
2. Record the **Client ID**.
3. Reveal and record the **Client Secret**.
4. Under **OAuth 2.0 settings**, add the following **Redirect URL**. Replace `your-tenant` with the name of your tenant. Use **all lowercase letters** for the tenant name even if it's defined with uppercase letters in Azure AD B2C.

<https://your-tenant.b2clogin.com/your-tenant.onmicrosoft.com/oauth2/authresp>

Create a policy key

You need to store the client secret that you previously recorded in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy keys** and then select **Add**.
6. For **Options**, choose `Manual`.
7. Enter a **Name** for the policy key. For example, `LinkedInSecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter the client secret that you previously recorded.
9. For **Key usage**, select `Signature`.
10. Click **Create**.

Add a claims provider

If you want users to sign in using a LinkedIn account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

Define a LinkedIn account as a claims provider by adding it to the **ClaimsProviders** element in the extension file of your policy.

1. Open the *SocialAndLocalAccounts/TrustFrameworkExtensions.xml* file in your editor. This file is in the [Custom policy starter pack](#) you downloaded as part of one of the prerequisites.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```

<ClaimsProvider>
    <Domain>linkedin.com</Domain>
    <DisplayName>LinkedIn</DisplayName>
    <TechnicalProfiles>
        <TechnicalProfile Id="LinkedIn-OAUTH">
            <DisplayName>LinkedIn</DisplayName>
            <Protocol Name="OAuth2" />
            <Metadata>
                <Item Key="ProviderName">linkedin</Item>
                <Item Key="authorization_endpoint">https://www.linkedin.com/oauth/v2/authorization</Item>
                <Item Key="AccessTokenEndpoint">https://www.linkedin.com/oauth/v2/accessToken</Item>
                <Item Key="ClaimsEndpoint">https://api.linkedin.com/v2/me</Item>
                <Item Key="scope">r_emailaddress r_liteprofile</Item>
                <Item Key="HttpBinding">POST</Item>
                <Item Key="external_user_identity_claim_id">id</Item>
                <Item Key="BearerTokenTransmissionMethod">AuthorizationHeader</Item>
                <Item Key="ResolveJsonPathsInJsonTokens">true</Item>
                <Item Key="UsePolicyInRedirectUri">0</Item>
                <Item Key="client_id">Your LinkedIn application client ID</Item>
            </Metadata>
            <CryptographicKeys>
                <Key Id="client_secret" StorageReferenceId="B2C_1A_LinkedInSecret" />
            </CryptographicKeys>
            <InputClaims />
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="id" />
                <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName.localized" />
                <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName.localized" />
                <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="linkedin.com" AlwaysUseDefaultValue="true" />
                <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" AlwaysUseDefaultValue="true" />
            </OutputClaims>
            <OutputClaimsTransformations>
                <OutputClaimsTransformation ReferenceId="ExtractGivenNameFromLinkedInResponse" />
                <OutputClaimsTransformation ReferenceId="ExtractSurNameFromLinkedInResponse" />
                <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
                <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
                <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
                <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
            </OutputClaimsTransformations>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>

```

4. Replace the value of **client_id** with the client ID of the LinkedIn application that you previously recorded.

5. Save the file.

Add the claims transformations

The LinkedIn technical profile requires the **ExtractGivenNameFromLinkedInResponse** and **ExtractSurNameFromLinkedInResponse** claims transformations to be added to the list of ClaimsTransformations. If you don't have a **ClaimsTransformations** element defined in your file, add the parent XML elements as shown below. The claims transformations also need a new claim type defined named **nullStringClaim**.

Add the **BuildingBlocks** element near the top of the *TrustFrameworkExtensions.xml* file. See *TrustFrameworkBase.xml* for an example.

```

<BuildingBlocks>
  <ClaimsSchema>
    <!-- Claim type needed for LinkedIn claims transformations -->
    <ClaimType Id="nullStringClaim">
      <DisplayName>nullClaim</DisplayName>
      <DataType>string</DataType>
      <AdminHelpText>A policy claim to store output values from ClaimsTransformations that aren't useful. This claim should not be used in TechnicalProfiles.</AdminHelpText>
      <UserHelpText>A policy claim to store output values from ClaimsTransformations that aren't useful. This claim should not be used in TechnicalProfiles.</UserHelpText>
    </ClaimType>
  </ClaimsSchema>

  <ClaimsTransformations>
    <!-- Claim transformations needed for LinkedIn technical profile -->
    <ClaimsTransformation Id="ExtractGivenNameFromLinkedInResponse"
TransformationMethod="GetSingleItemFromJson">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="inputJson" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="nullStringClaim" TransformationClaimType="key" />
        <OutputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="value" />
      </OutputClaims>
    </ClaimsTransformation>
    <ClaimsTransformation Id="ExtractSurNameFromLinkedInResponse" TransformationMethod="GetSingleItemFromJson">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="surname" TransformationClaimType="inputJson" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="nullStringClaim" TransformationClaimType="key" />
        <OutputClaim ClaimTypeReferenceId="surname" TransformationClaimType="value" />
      </OutputClaims>
    </ClaimsTransformation>
  </ClaimsTransformations>
</BuildingBlocks>

```

Upload the extension file for verification

You now have a policy configured so that Azure AD B2C knows how to communicate with your LinkedIn account. Try uploading the extension file of your policy to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up or sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the LinkedIn identity provider.

1. Open the *TrustFrameworkBase.xml* file in the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInLinkedIn`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up or sign-in screen.

If you add a **ClaimsProviderSelection** element for a LinkedIn account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelections**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `LinkedInExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="LinkedInExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with a LinkedIn account to receive a token.

1. Find the **OrchestrationStep** that includes `order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for the ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="LinkedInExchange" TechnicalProfileReferenceId="LinkedIn-OAUTH" />
```

Update the value of **TechnicalProfileReferenceId** to the ID of the technical profile you created earlier. For example, `LinkedIn-OAUTH`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInLinkedIn.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique

value. For example, `SignUpSignInLinkedIn`.

3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_linkedin`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (`SignUpSignInLinkedIn`).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Migration from v1.0 to v2.0

LinkedIn recently [updated their APIs from v1.0 to v2.0](#). To migrate your existing configuration to the new configuration, use the information in the following sections to update the elements in the technical profile.

Replace items in the Metadata

In the existing **Metadata** element of the **TechnicalProfile**, update the following **Item** elements from:

```
<Item Key="ClaimsEndpoint">https://api.linkedin.com/v1/people/~:(id,first-name,last-name,email-address,headline)</Item>
<Item Key="scope">r_emailaddress r_basicprofile</Item>
```

To:

```
<Item Key="ClaimsEndpoint">https://api.linkedin.com/v2/me</Item>
<Item Key="scope">r_emailaddress r_liteprofile</Item>
```

Add items to the Metadata

In the **Metadata** of the **TechnicalProfile**, add the following **Item** elements:

```
<Item Key="external_user_identity_claim_id">id</Item>
<Item Key="BearerTokenTransmissionMethod">AuthorizationHeader</Item>
<Item Key="ResolveJsonPathsInJsonTokens">true</Item>
```

Update the OutputClaims

In the existing **OutputClaims** of the **TechnicalProfile**, update the following **OutputClaim** elements from:

```
<OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
<OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
```

To:

```
<OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName.localized" />
<OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName.localized" />
```

Add new OutputClaimsTransformation elements

In the **OutputClaimsTransformations** of the **TechnicalProfile**, add the following **OutputClaimsTransformation** elements:

```
<OutputClaimsTransformation ReferenceId="ExtractGivenNameFromLinkedInResponse" />
<OutputClaimsTransformation ReferenceId="ExtractSurNameFromLinkedInResponse" />
```

Define the new claims transformations and claim type

In the last step, you added new claims transformations that need to be defined. To define the claims transformations, add them to the list of **ClaimsTransformations**. If you don't have a **ClaimsTransformations** element defined in your file, add the parent XML elements as shown below. The claims transformations also need a new claim type defined named **nullStringClaim**.

The **BuildingBlocks** element should be added near the top of the file. See the *TrustframeworkBase.xml* as an example.

```
<BuildingBlocks>
  <ClaimsSchema>
    <!-- Claim type needed for LinkedIn claims transformations -->
    <ClaimType Id="nullStringClaim">
      <DisplayName>nullClaim</DisplayName>
      <DataType>string</DataType>
      <AdminHelpText>A policy claim to store unuseful output values from ClaimsTransformations. This claim should not be used in a TechnicalProfiles.</AdminHelpText>
      <UserHelpText>A policy claim to store unuseful output values from ClaimsTransformations. This claim should not be used in a TechnicalProfiles.</UserHelpText>
    </ClaimType>
  </ClaimsSchema>

  <ClaimsTransformations>
    <!-- Claim transformations needed for LinkedIn technical profile -->
    <ClaimsTransformation Id="ExtractGivenNameFromLinkedInResponse"
TransformationMethod="GetSingleItemFromJson">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="inputJson" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="nullStringClaim" TransformationClaimType="key" />
        <OutputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="value" />
      </OutputClaims>
    </ClaimsTransformation>
    <ClaimsTransformation Id="ExtractSurNameFromLinkedInResponse" TransformationMethod="GetSingleItemFromJson">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="surname" TransformationClaimType="inputJson" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="nullStringClaim" TransformationClaimType="key" />
        <OutputClaim ClaimTypeReferenceId="surname" TransformationClaimType="value" />
      </OutputClaims>
    </ClaimsTransformation>
  </ClaimsTransformations>
</BuildingBlocks>
```

Obtain an email address

As part of the LinkedIn migration from v1.0 to v2.0, an additional call to another API is required to obtain the email address. If you need to obtain the email address during sign-up, do the following:

1. Complete the steps above to allow Azure AD B2C to federate with LinkedIn to let the user sign in. As part of the federation, Azure AD B2C receives the access token for LinkedIn.
2. Save the LinkedIn access token into a claim. [See the instructions here.](#)
3. Add the following claims provider that makes the request to LinkedIn's `/emailAddress` API. In order to authorize this request, you need the LinkedIn access token.

```

<ClaimsProvider>
    <DisplayName>REST APIs</DisplayName>
    <TechnicalProfiles>
        <TechnicalProfile Id="API-LinkedInEmail">
            <DisplayName>Get LinkedIn email</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://api.linkedin.com/v2/emailAddress?q=members&projection=
(elements*(handle~))</Item>
                <Item Key="AuthenticationType">Bearer</Item>
                <Item Key="UseClaimAsBearerToken">identityProviderAccessToken</Item>
                <Item Key="SendClaimsIn">Url</Item>
                <Item Key="ResolveJsonPathsInJsonTokens">true</Item>
            </Metadata>
            <InputClaims>
                <InputClaim ClaimTypeReferenceId="identityProviderAccessToken" />
            </InputClaims>
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="elements[0].handle~.emailAddress"
/>
            </OutputClaims>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>

```

- Add the following orchestration step into your user journey, so that the API claims provider is triggered when a user signs in using LinkedIn. Make sure to update the `order` number appropriately. Add this step immediately after the orchestration step that triggers the LinkedIn technical profile.

```

<!-- Extra step for LinkedIn to get the email -->
<OrchestrationStep Order="3" Type="ClaimsExchange">
    <Preconditions>
        <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
            <Value>identityProvider</Value>
            <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="false">
            <Value>identityProvider</Value>
            <Value>linkedin.com</Value>
            <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
    </Preconditions>
    <ClaimsExchanges>
        <ClaimsExchange Id="GetEmail" TechnicalProfileReferenceId="API-LinkedInEmail" />
    </ClaimsExchanges>
</OrchestrationStep>

```

Obtaining the email address from LinkedIn during sign-up is optional. If you choose not to obtain the email from LinkedIn but require one during sign up, the user is required to manually enter the email address and validate it.

For a full sample of a policy that uses the LinkedIn identity provider, see the [Custom Policy Starter Pack](#).

Set up sign-in with a Microsoft account using custom policies in Azure Active Directory B2C

2/20/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users from a Microsoft account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

- Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).
- If you don't already have a Microsoft account, create one at <https://www.live.com/>.

Register an application

To enable sign-in for users with a Microsoft account, you need to register an application within the Azure AD tenant. The Azure AD tenant is not the same as your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your Azure AD tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **App registrations**.
4. Select **New registration**.
5. Enter a **Name** for your application. For example, *MSAapp1*.
6. Under **Supported account types**, select **Accounts in any organizational directory and personal Microsoft accounts (e.g. Skype, Xbox, Outlook.com)**.
7. Under **Redirect URI (optional)**, select **Web** and enter
`https://your-tenant-name.b2clogin.com/your-tenant-name.onmicrosoft.com/oauth2/authresp` in the text box.
Replace `your-tenant-name` with your Azure AD B2C tenant name.
8. Select **Register**
9. Record the **Application (client) ID** shown on the application Overview page. You need this when you configure the claims provider in a later section.
10. Select **Certificates & secrets**
11. Click **New client secret**
12. Enter a **Description** for the secret, for example *MSA Application Client Secret*, and then click **Add**.
13. Record the application password shown in the **Value** column. You use this value in the next section.

Configuring optional claims

If you want to get the `family_name` and `given_name` claims from Azure AD, you can configure optional claims for your application in the Azure portal UI or application manifest. For more information, see [How to provide optional](#)

claims to your Azure AD app.

1. Sign in to the [Azure portal](#). Search for and select **Azure Active Directory**.
2. From the **Manage** section, select **App registrations**.
3. Select the application you want to configure optional claims for in the list.
4. From the **Manage** section, select **Token configuration (preview)**.
5. Select **Add optional claim**.
6. Select the token type you want to configure.
7. Select the optional claims to add.
8. Click **Add**.

Create a policy key

Now that you've created the application in your Azure AD tenant, you need to store that application's client secret in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.
6. For **Options**, choose `Manual`.
7. Enter a **Name** for the policy key. For example, `MSASecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter the client secret that you recorded in the previous section.
9. For **Key usage**, select `Signature`.
10. Click **Create**.

Add a claims provider

To enable your users to sign in using a Microsoft account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define Azure AD as a claims provider by adding the **ClaimsProvider** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml` policy file.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```

<ClaimsProvider>
    <Domain>live.com</Domain>
    <DisplayName>Microsoft Account</DisplayName>
    <TechnicalProfiles>
        <TechnicalProfile Id="MSA-OIDC">
            <DisplayName>Microsoft Account</DisplayName>
            <Protocol Name="OpenIdConnect" />
            <Metadata>
                <Item Key="ProviderName">https://login.live.com</Item>
                <Item Key="METADATA">https://login.live.com/.well-known/openid-configuration</Item>
                <Item Key="response_types">code</Item>
                <Item Key="response_mode">form_post</Item>
                <Item Key="scope">openid profile email</Item>
                <Item Key="HttpBinding">POST</Item>
                <Item Key="UsePolicyInRedirectUri">0</Item>
                <Item Key="client_id">Your Microsoft application client ID</Item>
            </Metadata>
            <CryptographicKeys>
                <Key Id="client_secret" StorageReferenceId="B2C_1A_MSASecret" />
            </CryptographicKeys>
            <OutputClaims>
                <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="oid" />
                <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name" />
                <OutputClaim ClaimTypeReferenceId="surName" PartnerClaimType="family_name" />
                <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
                <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication"
/>
                <OutputClaim ClaimTypeReferenceId="identityProvider" PartnerClaimType="iss" />
                <OutputClaim ClaimTypeReferenceId="email" />
            </OutputClaims>
            <OutputClaimsTransformations>
                <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
                <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
                <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
                <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
            </OutputClaimsTransformations>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>

```

- Replace the value of **client_id** with the Azure AD application's *Application (client) ID* that you recorded earlier.

- Save the file.

You've now configured your policy so that Azure AD B2C knows how to communicate with your Microsoft account application in Azure AD.

Upload the extension file for verification

Before continuing, upload the modified policy to confirm that it doesn't have any issues so far.

- Navigate to your Azure AD B2C tenant in the Azure portal and select **Identity Experience Framework**.
- On the **Custom policies** page, select **Upload custom policy**.
- Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
- Click **Upload**.

If no errors are displayed in the portal, continue to the next section.

Register the claims provider

At this point, you've set up the identity provider, but it's not yet available in any of the sign-up or sign-in screens. To

make it available, create a duplicate of an existing template user journey, then modify it so that it also has the Microsoft account identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInMSA`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up or sign-in screen. If you add a **ClaimsProviderSelection** element for a Microsoft account, a new button is displayed when a user lands on the page.

1. In the *TrustFrameworkExtensions.xml* file, find the **OrchestrationStep** element that includes `order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `MicrosoftAccountExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="MicrosoftAccountExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with a Microsoft account to receive a token.

1. Find the **OrchestrationStep** that includes `order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for the ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="MicrosoftAccountExchange" TechnicalProfileReferenceId="MSA-OIDC" />
```

Update the value of **TechnicalProfileReferenceId** to match that of the `Id` value in the **TechnicalProfile** element of the claims provider you added earlier. For example, `MSA-OIDC`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.

4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInMSA.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInMSA`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_msa`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the user journey that you created earlier (*SignUpSignInMSA*).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created in the previous section (or by completing the prerequisites, for example *webapp1* or *testapp1*) is selected in the **Select application** field, and then test it by clicking **Run now**.
7. Select the **Microsoft Account** button and sign in.

If the sign-in operation is successful, you're redirected to `jwt.ms` which displays the Decoded Token, similar to:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "<key-ID>"
}.{
  "exp": 1562365200,
  "nbf": 1562361600,
  "ver": "1.0",
  "iss": "https://your-b2c-tenant.b2clogin.com/10000000-0000-0000-0000-000000000000/v2.0/",
  "sub": "20000000-0000-0000-0000-000000000000",
  "aud": "30000000-0000-0000-0000-000000000000",
  "acr": "b2c_1a_signupsigninmsa",
  "nonce": "defaultNonce",
  "iat": 1562361600,
  "auth_time": 1562361600,
  "idp": "live.com",
  "name": "Azure User",
  "email": "azureuser@contoso.com",
  "tid": "6fc3b573-7b38-4c0c-b627-2e8684f6c575"
}.[Signature]
```

Set up sign-in with a Twitter account by using custom policies in Azure Active Directory B2C

1/28/2020 • 6 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users of a Twitter account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Prerequisites

- Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).
- If you don't already have a Twitter account, create one at [Twitter sign-up page](#).

Create an application

To use Twitter as an identity provider in Azure AD B2C, you need to create a Twitter application.

1. Sign in to the [Twitter Developers](#) website with your Twitter account credentials.
2. Select **Create an app**.
3. Enter an **App name** and an **Application description**.
4. In **Website URL**, enter `https://your-tenant.b2clogin.com`. Replace `your-tenant` with the name of your tenant.
For example, <https://contosob2c.b2clogin.com>.
5. For the **Callback URL**, enter
`https://your-tenant.b2clogin.com/your-tenant.onmicrosoft.com/your-policy-Id/oauth1/authresp`. Replace `your-tenant` with the name of your tenant name and `your-policy-Id` with the identifier of your policy. For example, `b2c_1A_signup_signin_twitter`. You need to use all lowercase letters when entering your tenant name even if the tenant is defined with uppercase letters in Azure AD B2C.
6. At the bottom of the page, read and accept the terms, and then select **Create**.
7. On the **App details** page, select **Edit > Edit details**, check the box for **Enable Sign in with Twitter**, and then select **Save**.
8. Select **Keys and tokens** and record the **Consumer API Key** and the **Consumer API secret key** values to be used later.

Create a policy key

You need to store the secret key that you previously recorded in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.

6. For **Options**, choose `Manual`.
7. Enter a **Name** for the policy key. For example, `TwitterSecret`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. In **Secret**, enter your client secret that you previously recorded.
9. For **Key usage**, select `Encryption`.
10. Click **Create**.

Add a claims provider

If you want users to sign in using a Twitter account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define a Twitter account as a claims provider by adding it to the **ClaimsProviders** element in the extension file of your policy.

1. Open the `TrustFrameworkExtensions.xml`.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```
<ClaimsProvider>
  <Domain>twitter.com</Domain>
  <DisplayName>Twitter</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Twitter-OAUTH1">
      <DisplayName>Twitter</DisplayName>
      <Protocol Name="OAuth1" />
      <Metadata>
        <Item Key="ProviderName">Twitter</Item>
        <Item Key="authorization_endpoint">https://api.twitter.com/oauth/authenticate</Item>
        <Item Key="access_token_endpoint">https://api.twitter.com/oauth/access_token</Item>
        <Item Key="request_token_endpoint">https://api.twitter.com/oauth/request_token</Item>
        <Item Key="ClaimsEndpoint">https://api.twitter.com/1.1/account/verify_credentials.json?
include_email=true</Item>
        <Item Key="ClaimsResponseFormat">json</Item>
        <Item Key="client_id">Your Twitter application consumer key</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="client_secret" StorageReferenceId="B2C_1A_TwitterSecret" />
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="user_id" />
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="screen_name" />
        <OutputClaim ClaimTypeReferenceId="email" />
        <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="twitter.com" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName" />
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName" />
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId" />
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId" />
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-SocialLogin" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

4. Replace the value of **client_id** with the consumer key that you previously recorded.

5. Save the file.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Twitter account. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up or sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the Twitter identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInTwitter`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up or sign-in screen. If you add a **ClaimsProviderSelection** element for a Twitter account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `TwitterExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="TwitterExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with a Twitter account to receive a token.

1. Find the **OrchestrationStep** that includes `order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for the ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="TwitterExchange" TechnicalProfileReferenceId="Twitter-OAUTH1" />
```

Update the value of **TechnicalProfileReferenceId** to the ID of the technical profile you created earlier. For example, `Twitter-OAUTH1`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This

section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter `https://jwt.ms`
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInTwitter.xml*.
2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInTwitter`.
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_twitter`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (*SignUpSignTwitter*).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Add ADFS as a SAML identity provider using custom policies in Azure Active Directory B2C

2/28/2020 • 9 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for an ADFS user account by using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C). You enable sign-in by adding a [SAML technical profile](#) to a custom policy.

Prerequisites

- Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).
- Make sure that you have access to a certificate .pfx file with a private key. You can generate your own signed certificate and upload it to Azure AD B2C. Azure AD B2C uses this certificate to sign the SAML request sent to your SAML identity provider.
- In order for Azure to accept the .pfx file password, the password must be encrypted with the TripleDES-SHA1 option in Windows Certificate Store Export utility as opposed to AES256-SHA256.

Create a policy key

You need to store your certificate in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant. Select the **Directory + subscription** filter in the top menu and choose the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.
6. For **Options**, choose [Upload](#).
7. Enter a **Name** for the policy key. For example, `SamlCert`. The prefix `B2C_1A_` is added automatically to the name of your key.
8. Browse to and select your certificate .pfx file with the private key.
9. Click **Create**.

Add a claims provider

If you want users to sign in using an ADFS account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define an ADFS account as a claims provider by adding it to the **ClaimsProviders** element in the extension file of your policy. For more information, see [define a SAML technical profile](#).

1. Open the `TrustFrameworkExtensions.xml`.

2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.

3. Add a new **ClaimsProvider** as follows:

```
<ClaimsProvider>
  <Domain>contoso.com</Domain>
  <DisplayName>Contoso ADFS</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Contoso-SAML2">
      <DisplayName>Contoso ADFS</DisplayName>
      <Description>Login with your ADFS account</Description>
      <Protocol Name="SAML2"/>
      <Metadata>
        <Item Key="WantsEncryptedAssertions">false</Item>
        <Item Key="PartnerEntity">https://your-ADFS-domain/federationmetadata/2007-06/federationmetadata.xml</Item>
        <Item Key="XmlSignatureAlgorithm">Sha256</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="SamlAssertionSigning" StorageReferenceId="B2C_1A_ADFSSamlCert"/>
        <Key Id="SamlMessageSigning" StorageReferenceId="B2C_1A_ADFSSamlCert"/>
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="userPrincipalName" />
        <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name"/>
        <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name"/>
        <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email"/>
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name"/>
        <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="contoso.com" />
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication"/>
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Saml-idp"/>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

4. Replace `your-ADFS-domain` with the name of your ADFS domain and replace the value of the **identityProvider** output claim with your DNS (Arbitrary value that indicates your domain).

5. Locate the `<claimsProviders>` section and add the following XML snippet. If your policy already contains the `SM-Saml-idp` technical profile, skip to the next step. For more information, see [single sign-on session management](#).

```

<ClaimsProvider>
  <DisplayName>Session Management</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="SM-Saml-idp">
      <DisplayName>Session Management Provider</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.SamlSSOSessionProvider, Web.TPEngine,
      Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="IncludeSessionIndex">false</Item>
        <Item Key="RegisterServiceProviders">false</Item>
      </Metadata>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

6. Save the file.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with ADFS account. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

NOTE

The Visual Studio code B2C extension uses "socialIdpUserId." A social policy is also required for ADFS.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up or sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the ADFS identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInADFS`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up or sign-in screen. If you add a **ClaimsProviderSelection** element for an ADFS account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you created.
2. Under **ClaimsProviderSelections**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `ContosoExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="ContosoExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with an ADFS account to receive a token.

1. Find the **OrchestrationStep** that includes `Order="2"` in the user journey.
2. Add the following **ClaimsExchange** element making sure that you use the same value for the ID that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="ContosoExchange" TechnicalProfileReferenceId="Contoso-SAML2" />
```

Update the value of **TechnicalProfileReferenceId** to the ID of the technical profile you created earlier. For example, `Contoso-SAML2`.

3. Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Configure an ADFS relying party trust

To use ADFS as an identity provider in Azure AD B2C, you need to create an ADFS Relying Party Trust with the Azure AD B2C SAML metadata. The following example shows a URL address to the SAML metadata of an Azure AD B2C technical profile:

```
https://your-tenant-name.b2clogin.com/your-tenant-name/your-policy/samlp/metadata?idptp=your-technical-profile
```

Replace the following values:

- **your-tenant** with your tenant name, such as `your-tenant.onmicrosoft.com`.
- **your-policy** with your policy name. For example, `B2C_1A_signup_signin_adfs`.
- **your-technical-profile** with the name of your SAML identity provider technical profile. For example, `Contoso-SAML2`.

Open a browser and navigate to the URL. Make sure you type the correct URL and that you have access to the XML metadata file. To add a new relying party trust by using the ADFS Management snap-in and manually configure the settings, perform the following procedure on a federation server. Membership in **Administrators** or equivalent on the local computer is the minimum required to complete this procedure.

1. In Server Manager, select **Tools**, and then select **ADFS Management**.
2. Select **Add Relying Party Trust**.
3. On the **Welcome** page, choose **Claims aware**, and then click **Start**.
4. On the **Select Data Source** page, select **Import data about the relying party publish online or on a local network**, provide your Azure AD B2C metadata URL, and then click **Next**.
5. On the **Specify Display Name** page, enter a **Display name**, under **Notes**, enter a description for this relying party trust, and then click **Next**.
6. On the **Choose Access Control Policy** page, select a policy, and then click **Next**.
7. On the **Ready to Add Trust** page, review the settings, and then click **Next** to save your relying party trust information.
8. On the **Finish** page, click **Close**, this action automatically displays the **Edit Claim Rules** dialog box.
9. Select **Add Rule**.

10. In **Claim rule template**, select **Send LDAP attributes as claims**.
11. Provide a **Claim rule name**. For the **Attribute store**, select **Select Active Directory**, add the following claims, then click **Finish** and **OK**.

LDAP ATTRIBUTE	OUTGOING CLAIM TYPE
User-Principal-Name	userPrincipalName
Surname	family_name
Given-Name	given_name
E-Mail-Address	email
Display-Name	name

Note that these names will not display in the outgoing claim type dropdown. You need to manually type them in. (The dropdown is actually editable).

12. Based on your certificate type, you may need to set the HASH algorithm. On the relying party trust (B2C Demo) properties window, select the **Advanced** tab and change the **Secure hash algorithm** to **SHA-256**, and click **Ok**.
13. In Server Manager, select **Tools**, and then select **ADFS Management**.
14. Select the relying party trust you created, select **Update from Federation Metadata**, and then click **Update**.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Select **Applications**, and then select **Add**.
5. Enter a name for the application. For example, *testapp1*.
6. For **Web App / Web API**, select **Yes**.
7. For **Reply URL**, enter <https://jwt.ms>
8. Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you created.

1. Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInADFS.xml*.

2. Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInADFS` .
3. Update the value of **PublicPolicyUri** with the URI for the policy. For example,
`http://contoso.com/B2C_1A_signup_signin_adfs`
4. Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (SignUpSignInADFS).
5. Save your changes, upload the file, and then select the new policy in the list.
6. Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Set up sign-in with a Salesforce SAML provider by using custom policies in Azure Active Directory B2C

2/28/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article shows you how to enable sign-in for users from a Salesforce organization using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C). You enable sign-in by adding a [SAML technical profile](#) to a custom policy.

Prerequisites

- Complete the steps in [Get started with custom policies in Azure Active Directory B2C](#).
- If you haven't already done so, sign up for a [free Developer Edition account](#). This article uses the [Salesforce Lightning Experience](#).
- [Set up a My Domain](#) for your Salesforce organization.

Set up Salesforce as an identity provider

1. [Sign in to Salesforce](#).
2. On the left menu, under **Settings**, expand **Identity**, and then select **Identity Provider**.
3. Select **Enable Identity Provider**.
4. Under **Select the certificate**, select the certificate you want Salesforce to use to communicate with Azure AD B2C. You can use the default certificate.
5. Click **Save**.

Create a connected app in Salesforce

1. On the **Identity Provider** page, select **Service Providers are now created via Connected Apps. Click here**.
2. Under **Basic Information**, enter the required values for your connected app.
3. Under **Web App Settings**, check the **Enable SAML** box.
4. In the **Entity ID** field, enter the following URL. Make sure that you replace the value for `your-tenant` with the name of your Azure AD B2C tenant.

```
https://your-tenant.b2clogin.com/your-tenant.onmicrosoft.com/B2C_1A_TrustFrameworkBase
```

5. In the **ACS URL** field, enter the following URL. Make sure that you replace the value for `your-tenant` with the name of your Azure AD B2C tenant.

```
https://your-tenant.b2clogin.com/your-tenant.onmicrosoft.com/B2C_1A_TrustFrameworkBase/samlp/sso/assertionconsumer
```

6. Scroll to the bottom of the list, and then click **Save**.

Get the metadata URL

1. On the overview page of your connected app, click **Manage**.
2. Copy the value for **Metadata Discovery Endpoint**, and then save it. You'll use it later in this article.

Set up Salesforce users to federate

1. On the **Manage** page of your connected app, click **Manage Profiles**.
2. Select the profiles (or groups of users) that you want to federate with Azure AD B2C. As a system administrator, select the **System Administrator** check box, so that you can federate by using your Salesforce account.

Generate a signing certificate

Requests sent to Salesforce need to be signed by Azure AD B2C. To generate a signing certificate, open Azure PowerShell, and then run the following commands.

NOTE

Make sure that you update the tenant name and password in the top two lines.

```
$tenantName = "<YOUR TENANT NAME>.onmicrosoft.com"
$pwdText = "<YOUR PASSWORD HERE>

$Cert = New-SelfSignedCertificate -CertStoreLocation Cert:\CurrentUser\My -DnsName "SamlIdp.$tenantName" -
Subject "B2C SAML Signing Cert" -HashAlgorithm SHA256 -KeySpec Signature -KeyLength 2048

$pwd = ConvertTo-SecureString -String $pwdText -Force -AsPlainText

Export-PfxCertificate -Cert $Cert -FilePath .\B2CSigningCert.pfx -Password $pwd
```

Create a policy key

You need to store the certificate that you created in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. On the Overview page, select **Identity Experience Framework**.
5. Select **Policy Keys** and then select **Add**.
6. For **Options**, choose [Upload](#).
7. Enter a **Name** for the policy. For example, SAMLSigningCert. The prefix `B2C_1A_` is automatically added to the name of your key.
8. Browse to and select the B2CSigningCert.pfx certificate that you created.
9. Enter the **Password** for the certificate.
10. Click **Create**.

Add a claims provider

If you want users to sign in using a Salesforce account, you need to define the account as a claims provider that Azure AD B2C can communicate with through an endpoint. The endpoint provides a set of claims that are used by Azure AD B2C to verify that a specific user has authenticated.

You can define a Salesforce account as a claims provider by adding it to the **ClaimsProviders** element in the

extension file of your policy. For more information, see [define a SAML technical profile](#).

1. Open the *TrustFrameworkExtensions.xml*.
2. Find the **ClaimsProviders** element. If it does not exist, add it under the root element.
3. Add a new **ClaimsProvider** as follows:

```
<ClaimsProvider>
  <Domain>salesforce</Domain>
  <DisplayName>Salesforce</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="salesforce">
      <DisplayName>Salesforce</DisplayName>
      <Description>Login with your Salesforce account</Description>
      <Protocol Name="SAML2"/>
      <Metadata>
        <Item Key="WantsEncryptedAssertions">false</Item>
        <Item Key="WantsSignedAssertions">false</Item>
        <Item Key="PartnerEntity">https://contoso-dev-ed.my.salesforce.com/.well-known/samlidp.xml</Item>
      </Metadata>
      <CryptographicKeys>
        <Key Id="SamlAssertionSigning" StorageReferenceId="B2C_1A_SAMLSigningCert"/>
        <Key Id="SamlMessageSigning" StorageReferenceId="B2C_1A_SAMLSigningCert"/>
      </CryptographicKeys>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="userId"/>
        <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="given_name"/>
        <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="family_name"/>
        <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email"/>
        <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="username"/>
        <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication"/>
        <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="SAMLIdp" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="CreateRandomUPNUserName"/>
        <OutputClaimsTransformation ReferenceId="CreateUserPrincipalName"/>
        <OutputClaimsTransformation ReferenceId="CreateAlternativeSecurityId"/>
        <OutputClaimsTransformation ReferenceId="CreateSubjectClaimFromAlternativeSecurityId"/>
      </OutputClaimsTransformations>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Saml-idp"/>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

4. Update the value of **PartnerEntity** with the Salesforce metadata URL you copied earlier.
5. Update the value of both instances of **StorageReferenceId** to the name of the key of your signing certificate. For example, B2C_1A_SAMLSigningCert.
6. Locate the `<ClaimsProviders>` section and add the following XML snippet. If your policy already contains the `SM-Saml-idp` technical profile, skip to the next step. For more information, see [single sign-on session management](#).

```

<ClaimsProvider>
  <DisplayName>Session Management</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="SM-Saml-idp">
      <DisplayName>Session Management Provider</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.SamlSSOSessionProvider, Web.TPEngine,
      Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="IncludeSessionIndex">false</Item>
        <Item Key="RegisterServiceProviders">false</Item>
      </Metadata>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

7. Save the file.

Upload the extension file for verification

By now, you have configured your policy so that Azure AD B2C knows how to communicate with your Salesforce account. Try uploading the extension file of your policy just to confirm that it doesn't have any issues so far.

1. On the **Custom Policies** page in your Azure AD B2C tenant, select **Upload Policy**.
2. Enable **Overwrite the policy if it exists**, and then browse to and select the *TrustFrameworkExtensions.xml* file.
3. Click **Upload**.

Register the claims provider

At this point, the identity provider has been set up, but it's not available in any of the sign-up or sign-in screens. To make it available, you create a duplicate of an existing template user journey, and then modify it so that it also has the Salesforce identity provider.

1. Open the *TrustFrameworkBase.xml* file from the starter pack.
2. Find and copy the entire contents of the **UserJourney** element that includes `Id="SignUpOrSignIn"`.
3. Open the *TrustFrameworkExtensions.xml* and find the **UserJourneys** element. If the element doesn't exist, add one.
4. Paste the entire content of the **UserJourney** element that you copied as a child of the **UserJourneys** element.
5. Rename the ID of the user journey. For example, `SignUpSignInSalesforce`.

Display the button

The **ClaimsProviderSelection** element is analogous to an identity provider button on a sign-up or sign-in screen. If you add a **ClaimsProviderSelection** element for a LinkedIn account, a new button shows up when a user lands on the page.

1. Find the **OrchestrationStep** element that includes `Order="1"` in the user journey that you just created.
2. Under **ClaimsProviderSelects**, add the following element. Set the value of **TargetClaimsExchangeId** to an appropriate value, for example `SalesforceExchange`:

```
<ClaimsProviderSelection TargetClaimsExchangeId="SalesforceExchange" />
```

Link the button to an action

Now that you have a button in place, you need to link it to an action. The action, in this case, is for Azure AD B2C to communicate with a Salesforce account to receive a token.

- Find the **OrchestrationStep** that includes `Order="2"` in the user journey.
- Add the following **ClaimsExchange** element making sure that you use the same value for **Id** that you used for **TargetClaimsExchangeId**:

```
<ClaimsExchange Id="SalesforceExchange" TechnicalProfileReferenceId="salesforce" />
```

Update the value of **TechnicalProfileReferenceId** to the **Id** of the technical profile you created earlier. For example, `LinkedIn-OAUTH`.

- Save the *TrustFrameworkExtensions.xml* file and upload it again for verification.

Create an Azure AD B2C application

Communication with Azure AD B2C occurs through an application that you register in your B2C tenant. This section lists optional steps you can complete to create a test application if you haven't already done so.

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

- Sign in to the [Azure portal](#).
- Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
- In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
- Select **Applications**, and then select **Add**.
- Enter a name for the application. For example, *testapp 1*.
- For **Web App / Web API**, select **Yes**.
- For **Reply URL**, enter `https://jwt.ms`
- Select **Create**.

Update and test the relying party file

Update the relying party (RP) file that initiates the user journey that you just created:

- Make a copy of *SignUpOrSignIn.xml* in your working directory, and rename it. For example, rename it to *SignUpSignInSalesforce.xml*.
- Open the new file and update the value of the **PolicyId** attribute for **TrustFrameworkPolicy** with a unique value. For example, `SignUpSignInSalesforce`.
- Update the value of **PublicPolicyUri** with the URI for the policy. For example, `http://contoso.com/B2C_1A_signup_signin_salesforce`
- Update the value of the **ReferenceId** attribute in **DefaultUserJourney** to match the ID of the new user journey that you created (*SignUpSignInSalesforce*).
- Save your changes, upload the file, and then select the new policy in the list.
- Make sure that Azure AD B2C application that you created is selected in the **Select application** field, and then test it by clicking **Run now**.

Manage SSO and token customization using custom policies in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

This article provides information about how you can manage your token, session, and single sign-on (SSO) configurations using [custom policies](#) in Azure Active Directory B2C (Azure AD B2C).

Token lifetimes and claims configuration

To change the settings on your token lifetimes, you add a [ClaimsProviders](#) element in the relying party file of the policy you want to impact. The **ClaimsProviders** element is a child of the [TrustFrameworkPolicy](#) element.

Insert the ClaimsProviders element between the BasePolicy element and the RelyingParty element of the relying party file.

Inside, you'll need to put the information that affects your token lifetimes. The XML looks like this example:

```
<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>Token Issuer</DisplayName>
    <TechnicalProfiles>
      <TechnicalProfile Id="JwtIssuer">
        <Metadata>
          <Item Key="token_lifetime_secs">3600</Item>
          <Item Key="id_token_lifetime_secs">3600</Item>
          <Item Key="refresh_token_lifetime_secs">1209600</Item>
          <Item Key="rolling_refresh_token_lifetime_secs">7776000</Item>
          <Item Key="IssuanceClaimPattern">AuthorityAndTenantGuid</Item>
          <Item Key="AuthenticationContextReferenceClaimPattern">None</Item>
        </Metadata>
      </TechnicalProfile>
    </TechnicalProfiles>
  </ClaimsProvider>
</ClaimsProviders>
```

The following values are set in the previous example:

- **Access token lifetimes** - The access token lifetime value is set with **token_lifetime_secs** metadata item. The default value is 3600 seconds (60 minutes).
- **ID token lifetime** - The ID token lifetime value is set with the **id_token_lifetime_secs** metadata item. The default value is 3600 seconds (60 minutes).
- **Refresh token lifetime** - The refresh token lifetime value is set with the **refresh_token_lifetime_secs** metadata item. The default value is 1209600 seconds (14 days).
- **Refresh token sliding window lifetime** - If you would like to set a sliding window lifetime to your refresh token, set the value of **rolling_refresh_token_lifetime_secs** metadata item. The default value is 7776000 (90 days). If you don't want to enforce a sliding window lifetime, replace the item with
`<Item Key="allow_infinite_rolling_refresh_token">True</Item>`
- **Issuer (iss) claim** - The Issuer (iss) claim is set with the **IssuanceClaimPattern** metadata item. The applicable values are `AuthorityAndTenantGuid` and `AuthorityWithTfp`.
- **Setting claim representing policy ID** - The options for setting this value are `TFP` (trust framework

policy) and **ACR** (authentication context reference). **TPP** is the recommended value. Set **AuthenticationContextReferenceClaimPattern** with the value of **None**.

In the **ClaimsSchema** element, add this element:

```
<ClaimType Id="trustFrameworkPolicy">
  <DisplayName>Trust framework policy name</DisplayName>
  <DataType>string</DataType>
</ClaimType>
```

In your **OutputClaims** element, add this element:

```
<OutputClaim ClaimTypeReferenceId="trustFrameworkPolicy" Required="true" DefaultValue="{policy}" />
```

For ACR, remove the **AuthenticationContextReferenceClaimPattern** item.

- **Subject (sub) claim** - This option defaults to ObjectID, if you would like to switch this setting to **Not Supported**, replace this line:

```
<OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub" />
```

with this line:

```
<OutputClaim ClaimTypeReferenceId="sub" />
```

Session behavior and SSO

To change your session behavior and SSO configurations, you add a **UserJourneyBehaviors** element inside of the [RelyingParty](#) element. The **UserJourneyBehaviors** element must immediately follow the **DefaultUserJourney**. The inside of your **UserJourneyBehaviors** element should look like this example:

```
<UserJourneyBehaviors>
  <SingleSignOn Scope="Application" />
  <SessionExpiryType>Absolute</SessionExpiryType>
  <SessionExpiryInSeconds>86400</SessionExpiryInSeconds>
</UserJourneyBehaviors>
```

The following values are configured in the previous example:

- **Single sign on (SSO)** - Single sign-on is configured with the **SingleSignOn**. The applicable values are **Tenant**, **Application**, **Policy**, and **Suppressed**.
- **Web app session time-out** - The web app session timeout is set with the **SessionExpiryType** element. The applicable values are **Absolute** and **Rolling**.
- **Web app session lifetime** - The web app session lifetime is set with the **SessionExpiryInSeconds** element. The default value is 86400 seconds (1440 minutes).

Pass an access token through a custom policy to your application in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

A [custom policy](#) in Azure Active Directory B2C (Azure AD B2C) provides users of your application an opportunity to sign up or sign in with an identity provider. When this happens, Azure AD B2C receives an [access token](#) from the identity provider. Azure AD B2C uses that token to retrieve information about the user. You add a claim type and output claim to your custom policy to pass the token through to the applications that you register in Azure AD B2C.

Azure AD B2C supports passing the access token of [OAuth 2.0](#) and [OpenID Connect](#) identity providers. For all other identity providers, the claim is returned blank.

Prerequisites

- Your custom policy is configured with an OAuth 2.0 or OpenID Connect identity provider.

Add the claim elements

1. Open your *TrustframeworkExtensions.xml* file and add the following **ClaimType** element with an identifier of `identityProviderAccessToken` to the **ClaimsSchema** element:

```
<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="identityProviderAccessToken">
      <DisplayName>Identity Provider Access Token</DisplayName>
      <DataType>string</DataType>
      <AdminHelpText>Stores the access token of the identity provider.</AdminHelpText>
    </ClaimType>
    ...
  </ClaimsSchema>
</BuildingBlocks>
```

2. Add the **OutputClaim** element to the **TechnicalProfile** element for each OAuth 2.0 identity provider that you would like the access token for. The following example shows the element added to the Facebook technical profile:

```
<ClaimsProvider>
  <DisplayName>Facebook</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Facebook-OAUTH">
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="identityProviderAccessToken" PartnerClaimType="
{oauth2:access_token}" />
      </OutputClaims>
      ...
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

3. Save the *TrustframeworkExtensions.xml* file.
4. Open your relying party policy file, such as *SignUpOrSignIn.xml*, and add the **OutputClaim** element to the

TechnicalProfile:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="identityProviderAccessToken"
PartnerClaimType="idp_access_token"/>
    </OutputClaims>
    ...
  </TechnicalProfile>
</RelyingParty>
```

5. Save the policy file.

Test your policy

When testing your applications in Azure AD B2C, it can be useful to have the Azure AD B2C token returned to <https://jwt.ms> to be able to review the claims in it.

Upload the files

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure AD B2C tenant by clicking the **Directory + subscription** filter in the top menu and choosing the directory that contains your tenant.
3. Choose **All services** in the top-left corner of the Azure portal, and then search for and select **Azure AD B2C**.
4. Select **Identity Experience Framework**.
5. On the Custom Policies page, click **Upload Policy**.
6. Select **Overwrite the policy if it exists**, and then search for and select the *TrustframeworkExtensions.xml* file.
7. Select **Upload**.
8. Repeat steps 5 through 7 for the relying party file, such as *SignUpOrSignIn.xml*.

Run the policy

1. Open the policy that you changed. For example, *B2C_1A_signup_signin*.
2. For **Application**, select your application that you previously registered. To see the token in the example below, the **Reply URL** should show <https://jwt.ms>.
3. Select **Run now**.

You should see something similar to the following example:

Decoded Token	Claims
{ "typ": "JWT", "alg": "RS256", "kid": "kz1CxXo0qkPuHP5_-sFljrFyO11JZGB0EJ387LXd0pE" }.{ "exp": 1543350813, "nbf": 1543347213, "ver": "1.0", "iss": "https://contoso0926tenant.b2clogin.com/c64a4f7d-3091-0694f60b7/v2.0/", "sub": "10bd2040-5faede13b843", "aud": "327fa24a-70c0b5892198", "acr": "b2c_1a_signup_signin", "nonce": "defaultNonce", "iat": 1543347213, "auth_time": 1543347213, "idp_access_token": "EAAM7yCmyOnwBADXc4fBLn8xujlta1wbhASDaXTzOPZARqsHxsuLyXyqMHZAJiUWgO98yZCdXmZAxD4wx1VL6R3v1meYxg HYVzgNyVegvLPA3xH8mKju92SdRnGcz1kUZArZCwZDZD", "given_name": "David", "family_name": "Miller", "name": "David Miller", "email": "david.miller@example.com", "idp": "facebook.com" }.[Signature]	

Next steps

Learn more about tokens in the [Azure Active Directory B2C token reference](#).

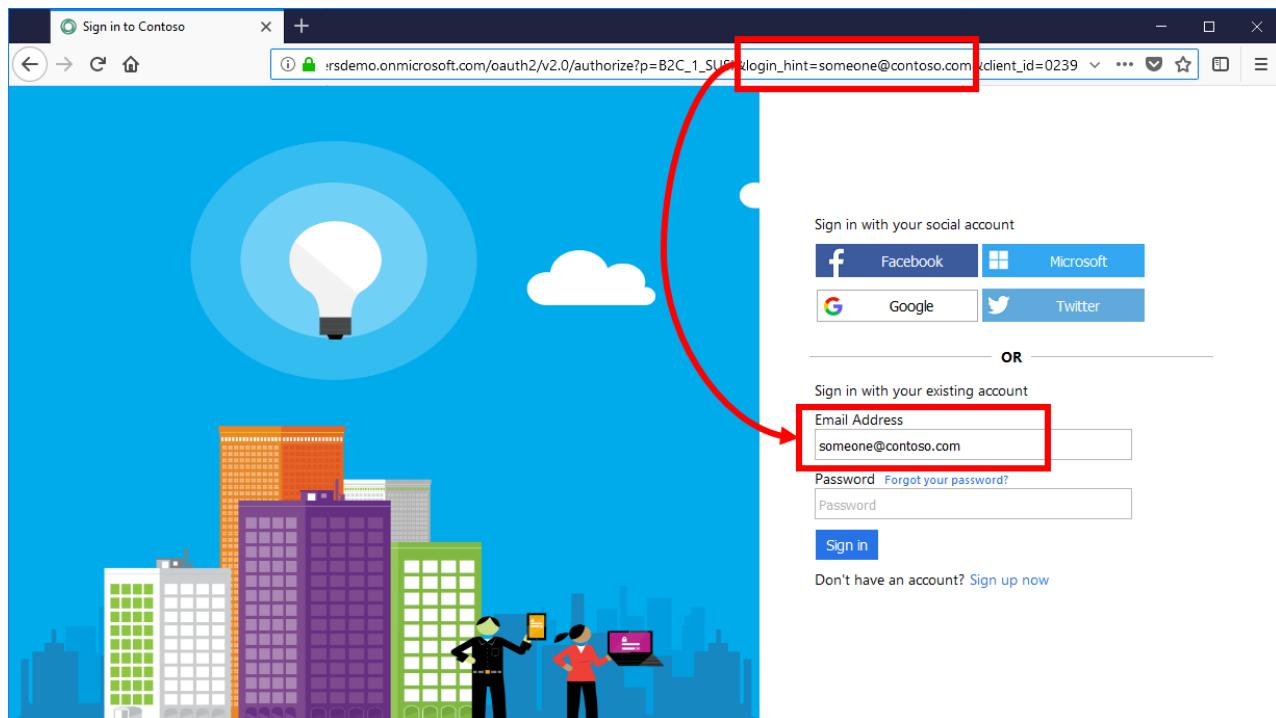
Set up direct sign-in using Azure Active Directory B2C

7/12/2019 • 2 minutes to read • [Edit Online](#)

When setting up sign-in for your application using Azure Active Directory (AD) B2C, you can prepopulate the sign-in name or direct sign-in to a specific social identity provider, such as Facebook, LinkedIn, or a Microsoft account.

Prepopulate the sign-in name

During a sign-in user journey, a relying party application may target a specific user or domain name. When targeting a user, an application can specify, in the authorization request, the `login_hint` query parameter with the user sign-in name. Azure AD B2C automatically populates the sign-in name, while the user only needs to provide the password.



The user is able to change the value in the sign-in textbox.

If you are using a custom policy, override the `SelfAsserted-LocalAccountSignin-Email` technical profile. In the `<InputClaims>` section, set the `DefaultValue` of the `signInName` claim to `{OIDC:LoginHint}`. The `{OIDC:LoginHint}` variable contains the value of the `login_hint` parameter. Azure AD B2C reads the value of the `signInName` claim and pre-populates the `signInName` textbox.

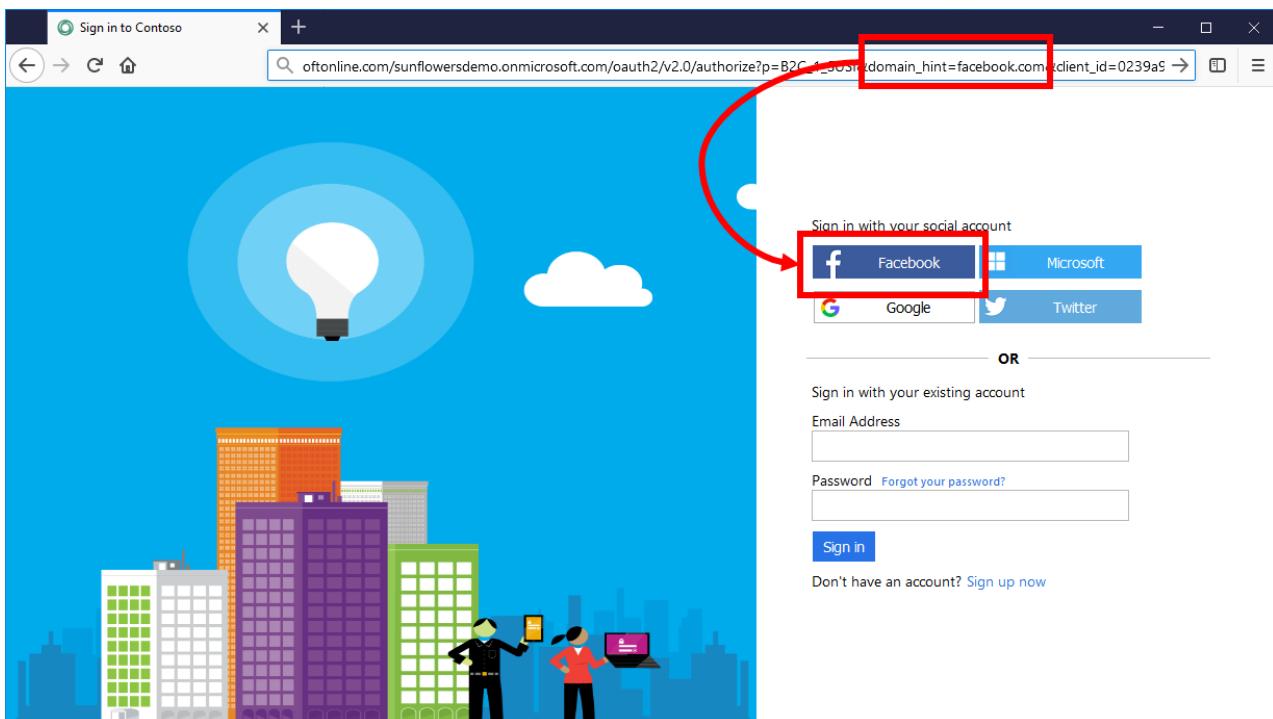
```

<ClaimsProvider>
  <DisplayName>Local Account</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email">
      <InputClaims>
        <!-- Add the login hint value to the sign-in names claim type -->
        <InputClaim ClaimTypeReferenceId="signInName" DefaultValue="{OIDC:LoginHint}" />
      </InputClaims>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

Redirect sign-in to a social provider

If you configured the sign-in journey for your application to include social accounts, such as Facebook, LinkedIn, or Google, you can specify the `domain_hint` parameter. This query parameter provides a hint to Azure AD B2C about the social identity provider that should be used for sign-in. For example, if the application specifies `domain_hint=facebook.com`, sign-in goes directly to the Facebook sign-in page.



If you are using a custom policy, you can configure the domain name using the `<Domain>domain_name</Domain>` XML element of any `<ClaimsProvider>`.

```

<ClaimsProvider>
  <!-- Add the domain hint value to the claims provider -->
  <Domain>facebook.com</Domain>
  <DisplayName>Facebook</DisplayName>
  <TechnicalProfiles>
    ...

```

Walkthrough: Integrate REST API claims exchanges in your Azure AD B2C user journey as validation on user input

1/28/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The Identity Experience Framework (IEF) that underlies Azure Active Directory B2C (Azure AD B2C) enables the identity developer to integrate an interaction with a RESTful API in a user journey.

At the end of this walkthrough, you will be able to create an Azure AD B2C user journey that interacts with RESTful services.

The IEF sends data in claims and receives data back in claims. The interaction with the API:

- Can be designed as a REST API claims exchange or as a validation profile, which happens inside an orchestration step.
- Typically validates input from the user. If the value from the user is rejected, the user can try again to enter a valid value with the opportunity to return an error message.

You can also design the interaction as an orchestration step. For more information, see [Walkthrough: Integrate REST API claims exchanges in your Azure AD B2C user journey as an orchestration step](#).

For the validation profile example, we will use the profile edit user journey in the starter pack file ProfileEdit.xml.

We can verify that the name provided by the user in the profile edit is not part of an exclusion list.

Prerequisites

- An Azure AD B2C tenant configured to complete a local account sign-up/sign-in, as described in [Getting started](#).
- A REST API endpoint to interact with. For this walkthrough, we've set up a demo site called [WingTipGames](#) with a REST API service.

Step 1: Prepare the REST API function

NOTE

Setup of REST API functions is outside the scope of this article. [Azure Functions](#) provides an excellent toolkit to create RESTful services in the cloud.

We have created an Azure function that receives a claim that it expects as `playerTag`. The function validates whether this claim exists. You can access the complete Azure function code in [GitHub](#).

```

if (requestContentAs JObject.playerTag == null)
{
    return request.CreateResponse(HttpStatusCode.BadRequest);
}

var playerTag = ((string) requestContentAs JObject.playerTag).ToLower();

if (playerTag == "mcvinny" || playerTag == "msgates123" || playerTag == "revcottonmarcus")
{
    return request.CreateResponse<ResponseContent>(
        HttpStatusCode.Conflict,
        new ResponseContent
        {
            version = "1.0.0",
            status = (int) HttpStatusCode.Conflict,
            userMessage = $"The player tag '{requestContentAs JObject.playerTag}' is already used."
        },
        new JsonMediaTypeFormatter(),
        "application/json");
}

return request.CreateResponse(HttpStatusCode.OK);

```

The IEF expects the `userMessage` claim that the Azure function returns. This claim will be presented as a string to the user if the validation fails, such as when a 409 conflict status is returned in the preceding example.

Step 2: Configure the RESTful API claims exchange as a technical profile in your TrustFrameworkExtensions.xml file

A technical profile is the full configuration of the exchange desired with the RESTful service. Open the `TrustFrameworkExtensions.xml` file and add the following XML snippet inside the `<ClaimsProviders>` element.

NOTE

In the following XML, RESTful provider `Version=1.0.0.0` is described as the protocol. Consider it as the function that will interact with the external service.

```

<ClaimsProvider>
    <DisplayName>REST APIs</DisplayName>
    <TechnicalProfiles>
        <TechnicalProfile Id="AzureFunctions-CheckPlayerTagWebHook">
            <DisplayName>Check Player Tag Web Hook Azure Function</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://wingtipb2cfunc.azurewebsites.net/api/CheckPlayerTagWebHook?
code=L/05YRSpojU0nECzM4Tp3LjBiA2ZGh3kTwwp10VV7m0Selnv1RVLCg==</Item>
                <Item Key="SendClaimsIn">Body</Item>
                <!-- Set AuthenticationType to Basic or ClientCertificate in production environments -->
                <Item Key="AuthenticationType">None</Item>
                <!-- REMOVE the following line in production environments -->
                <Item Key="AllowInsecureAuthInProduction">true</Item>
            </Metadata>
            <InputClaims>
                <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="playerTag" />
            </InputClaims>
            <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
        </TechnicalProfile>
        <TechnicalProfile Id="SelfAsserted-ProfileUpdate">
            <ValidationTechnicalProfiles>
                <ValidationTechnicalProfile ReferenceId="AzureFunctions-CheckPlayerTagWebHook" />
            </ValidationTechnicalProfiles>
        </TechnicalProfile>
    </TechnicalProfiles>
</ClaimsProvider>

```

The `InputClaims` element defines the claims that will be sent from the IEF to the REST service. In this example, the contents of the claim `givenName` will be sent to the REST service as `playerTag`. In this example, the IEF does not expect claims back. Instead, it waits for a response from the REST service and acts based on the status codes that it receives.

The comments above `AuthenticationType` and `AllowInsecureAuthInProduction` specify changes you should make when you move to a production environment. To learn how to secure your RESTful APIs for production, see [Secure RESTful APIs with basic auth](#) and [Secure RESTful APIs with certificate auth](#).

Step 3: Include the RESTful service claims exchange in self-asserted technical profile where you want to validate the user input

The most common use of the validation step is in the interaction with a user. All interactions where the user is expected to provide input are *self-asserted technical profiles*. For this example, we will add the validation to the Self-Asserted-ProfileUpdate technical profile. This is the technical profile that the relying party (RP) policy file `Profile Edit` uses.

To add the claims exchange to the self-asserted technical profile:

1. Open the TrustFrameworkBase.xml file and search for `<TechnicalProfile Id="SelfAsserted-ProfileUpdate">`.
2. Review the configuration of this technical profile. Observe how the exchange with the user is defined as claims that will be asked of the user (input claims) and claims that will be expected back from the self-asserted provider (output claims).
3. Search for `TechnicalProfileReferenceId="SelfAsserted-ProfileUpdate"`, and notice that this profile is invoked as orchestration step 5 of `<UserJourney Id="ProfileEdit">`.

Step 4: Upload and test the profile edit RP policy file

1. Upload the new version of the TrustFrameworkExtensions.xml file.

2. Use **Run now** to test the profile edit RP policy file.
3. Test the validation by providing one of the existing names (for example, mcvinny) in the **Given Name** field. If everything is set up correctly, you should receive a message that notifies the user that the player tag is already used.

Next steps

[Modify the profile edit and user registration to gather additional information from your users](#)

[Walkthrough: Integrate REST API claims exchanges in your Azure AD B2C user journey as an orchestration step](#)

[Reference: RESTful technical profile](#)

To learn how to secure your APIs, see the following articles:

- [Secure your RESTful API with basic authentication \(username and password\)](#)
- [Secure your RESTful API with client certificates](#)

Add REST API claims exchanges to custom policies in Azure Active Directory B2C

1/28/2020 • 5 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

You can add interaction with a RESTful API to your [custom policies](#) in Azure Active Directory B2C (Azure AD B2C). This article shows you how to create an Azure AD B2C user journey that interacts with RESTful services.

The interaction includes a claims exchange of information between the REST API claims and Azure AD B2C. Claims exchanges have the following characteristics:

- Can be designed as an orchestration step.
- Can trigger an external action. For instance, it can log an event in an external database.
- Can be used to fetch a value and then store it in the user database.
- Can change the flow of execution.

The scenario that is represented in this article includes the following actions:

1. Look up the user in an external system.
2. Get the city where that user is registered.
3. Return that attribute to the application as a claim.

Prerequisites

- Complete the steps in [Get started with custom policies](#).
- A REST API endpoint to interact with. This article uses a simple Azure function as an example. To create the Azure function, see [Create your first function in the Azure portal](#).

Prepare the API

In this section, you prepare the Azure function to receive a value for `email`, and then return the value for `city` that can be used by Azure AD B2C as a claim.

Change the `run.csx` file for the Azure function that you created to use the following code:

```

#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpContext req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");
    string email = req.Query["email"];
    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    email = email ?? data?.email;

    return email != null
        ? (ActionResult)new OkObjectResult(
            new ResponseContent
            {
                version = "1.0.0",
                status = (int) HttpStatusCode.OK,
                city = "Redmond"
            })
        : new BadRequestObjectResult("Please pass an email on the query string or in the request body");
}

public class ResponseContent
{
    public string version { get; set; }
    public int status { get; set; }
    public string city {get; set; }
}

```

Configure the claims exchange

A technical profile provides the configuration for the claim exchange.

Open the *TrustFrameworkExtensions.xml* file and add the following **ClaimsProvider** XML element inside the **ClaimsProviders** element.

```

<ClaimsProvider>
  <DisplayName>REST APIs</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="AzureFunctions-WebHook">
      <DisplayName>Azure Function Web Hook</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://myfunction.azurewebsites.net/api/HttpTrigger1?
        code=bAZ4lLy//ZHxmnC8rI7AgjQsrMKmVXBpP0vd9sm0zdXDDUIaLljA==</Item>
        <Item Key="SendClaimsIn">Body</Item>
        <!-- Set AuthenticationType to Basic or ClientCertificate in production environments -->
        <Item Key="AuthenticationType">None</Item>
        <!-- REMOVE the following line in production environments -->
        <Item Key="AllowInsecureAuthInProduction">true</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="email" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="city" PartnerClaimType="city" />
      </OutputClaims>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

The **InputClaims** element defines the claims that are sent to the REST service. In this example, the value of the claim `givenName` is sent to the REST service as the claim `email`. The **OutputClaims** element defines the claims that are expected from the REST service.

The comments above `AuthenticationType` and `AllowInsecureAuthInProduction` specify changes you should make when you move to a production environment. To learn how to secure your RESTful APIs for production, see [Secure RESTful APIs with basic auth](#) and [Secure RESTful APIs with certificate auth](#).

Add the claim definition

Add a definition for `city` inside the **BuildingBlocks** element. You can find this element at the beginning of the TrustFrameworkExtensions.xml file.

```

<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="city">
      <DisplayName>City</DisplayName>
      <DataType>string</DataType>
      <UserHelpText>Your city</UserHelpText>
      <UserInputType>TextBox</UserInputType>
    </ClaimType>
  </ClaimsSchema>
</BuildingBlocks>

```

Add an orchestration step

There are many use cases where the REST API call can be used as an orchestration step. As an orchestration step, it can be used as an update to an external system after a user has successfully completed a task like first-time registration, or as a profile update to keep information synchronized. In this case, it's used to augment the information provided to the application after the profile edit.

Add a step to the profile edit user journey. After the user is authenticated (orchestration steps 1-4 in the following

XML), and the user has provided the updated profile information (step 5). Copy the profile edit user journey XML code from the *TrustFrameworkBase.xml* file to your *TrustFrameworkExtensions.xml* file inside the **UserJourneys** element. Then make the modification as step 6.

```
<OrchestrationStep Order="6" Type="ClaimsExchange">
  <ClaimsExchanges>
    <ClaimsExchange Id="GetLoyaltyData" TechnicalProfileReferenceId="AzureFunctions-WebHook" />
  </ClaimsExchanges>
</OrchestrationStep>
```

The final XML for the user journey should look like this example:

```

<UserJourney Id="ProfileEdit">
  <OrchestrationSteps>
    <OrchestrationStep Order="1" Type="ClaimsProviderSelection"
ContentDefinitionReferenceId="api.idpselections">
      <ClaimsProviderSelections>
        <ClaimsProviderSelection TargetClaimsExchangeId="FacebookExchange" />
        <ClaimsProviderSelection TargetClaimsExchangeId="LocalAccountSigninEmailExchange" />
      </ClaimsProviderSelections>
    </OrchestrationStep>
    <OrchestrationStep Order="2" Type="ClaimsExchange">
      <ClaimsExchanges>
        <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
        <ClaimsExchange Id="LocalAccountSigninEmailExchange" TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <OrchestrationStep Order="3" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>authenticationSource</Value>
          <Value>localAccountAuthentication</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="AADUserRead" TechnicalProfileReferenceId="AAD-UserReadUsingAlternativeSecurityId" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <OrchestrationStep Order="4" Type="ClaimsExchange">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>authenticationSource</Value>
          <Value>socialIdpAuthentication</Value>
          <Action>SkipThisOrchestrationStep</Action>
        </Precondition>
      </Preconditions>
      <ClaimsExchanges>
        <ClaimsExchange Id="AADUserReadWithObjectId" TechnicalProfileReferenceId="AAD-UserReadUsingObjectId" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <OrchestrationStep Order="5" Type="ClaimsExchange">
      <ClaimsExchanges>
        <ClaimsExchange Id="B2CUserProfileUpdateExchange" TechnicalProfileReferenceId="SelfAsserted-ProfileUpdate" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <!-- Add a step 6 to the user journey before the JWT token is created-->
    <OrchestrationStep Order="6" Type="ClaimsExchange">
      <ClaimsExchanges>
        <ClaimsExchange Id="GetLoyaltyData" TechnicalProfileReferenceId="AzureFunctions-WebHook" />
      </ClaimsExchanges>
    </OrchestrationStep>
    <OrchestrationStep Order="7" Type="SendClaims" CpiIssuerTechnicalProfileReferenceId="JwtIssuer" />
  </OrchestrationSteps>
  <ClientDefinition ReferenceId="DefaultWeb" />
</UserJourney>

```

Add the claim

Edit the *ProfileEdit.xml* file and add `<OutputClaim ClaimTypeReferenceId="city" />` to the **OutputClaims** element.

After you add the new claim, the technical profile looks like this example:

```

<TechnicalProfile Id="PolicyProfile">
  <DisplayName>PolicyProfile</DisplayName>
  <Protocol Name="OpenIdConnect" />
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
    <OutputClaim ClaimTypeReferenceId="tenantId" AlwaysUseDefaultValue="true" DefaultValue="
{Policy:TenantObjectId}" />
    <OutputClaim ClaimTypeReferenceId="city" />
  </OutputClaims>
  <SubjectNamingInfo ClaimType="sub" />
</TechnicalProfile>

```

Upload your changes and test

1. (Optional) Save the existing version (by downloading) of the files before you proceed.
2. Upload the *TrustFrameworkExtensions.xml* and *ProfileEdit.xml* and select to overwrite the existing file.
3. Select **B2C_1A_ProfileEdit**.
4. For **Select application** on the overview page of the custom policy, select the web application named *webapp1* that you previously registered. Make sure that the **Reply URL** is <https://jwt.ms>.
5. Select **Run Now**. Sign in with your account credentials, and click **Continue**.

If everything is set up correctly, the token includes the new claim `city`, with the value `Redmond`.

```
{
  "exp": 1493053292,
  "nbf": 1493049692,
  "ver": "1.0",
  "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",
  "sub": "a58e7c6c-7535-4074-93da-b0023fbaf3ac",
  "aud": "4e87c1dd-e5f5-4ac8-8368-bc6a98751b8b",
  "acr": "b2c_1a_profileedit",
  "nonce": "defaultNonce",
  "iat": 1493049692,
  "auth_time": 1493049692,
  "city": "Redmond"
}
```

Next steps

You can also design the interaction as a validation profile. For more information, see [Walkthrough: Integrate REST API claims exchanges in your Azure AD B2C user journey as validation on user input](#).

[Modify the profile edit to gather additional information from your users](#)

[Reference: RESTful technical profile](#)

To learn out how to secure your APIs, see the following articles:

- [Secure your RESTful API with basic authentication \(username and password\)](#)
- [Secure your RESTful API with client certificates](#)

Integrate REST API claims exchanges in your Azure AD B2C user journey as validation of user input

1/28/2020 • 10 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

With the Identity Experience Framework, which underlies Azure Active Directory B2C (Azure AD B2C), you can integrate with a RESTful API in a user journey. In this walkthrough, you'll learn how Azure AD B2C interacts with .NET Framework RESTful services (web API).

Introduction

By using Azure AD B2C, you can add your own business logic to a user journey by calling your own RESTful service. The Identity Experience Framework sends data to the RESTful service in an *Input claims* collection and receives data back from RESTful in an *Output claims* collection. With RESTful service integration, you can:

- **Validate user input data:** This action prevents malformed data from persisting into Azure AD. If the value from the user is not valid, your RESTful service returns an error message that instructs the user to provide an entry. For example, you can verify that the email address provided by the user exists in your customer's database.
- **Overwrite input claims:** For example, if a user enters the first name in all lowercase or all uppercase letters, you can format the name with only the first letter capitalized.
- **Enrich user data by further integrating with corporate line-of-business applications:** Your RESTful service can receive the user's email address, query the customer's database, and return the user's loyalty number to Azure AD B2C. The return claims can be stored in the user's Azure AD account, evaluated in the next *Orchestration Steps*, or included in the access token.
- **Run custom business logic:** You can send push notifications, update corporate databases, run a user migration process, manage permissions, audit databases, and perform other actions.

You can design the integration with the RESTful services in the following ways:

- **Validation technical profile:** The call to the RESTful service happens within the validation technical profile of the specified technical profile. The validation technical profile validates the user-provided data before the user journey moves forward. With the validation technical profile, you can:
 - Send input claims.
 - Validate the input claims and throw custom error messages.
 - Send back output claims.
- **Claims exchange:** This design is similar to the validation technical profile, but it happens within an orchestration step. This definition is limited to:
 - Send input claims.
 - Send back output claims.

RESTful walkthrough

In this walkthrough, you develop a .NET Framework web API that validates the user input and provides a user loyalty number. For example, your application can grant access to *platinum benefits* based on the loyalty number.

Overview:

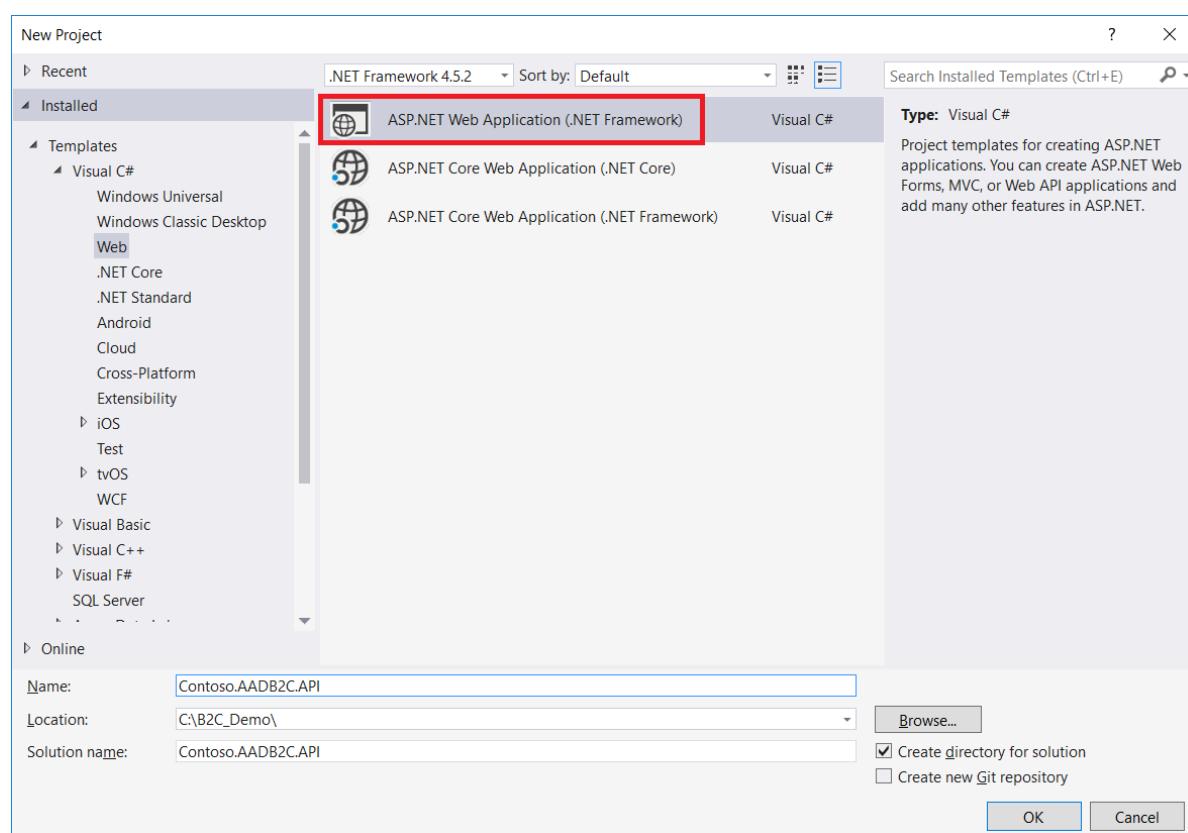
- Develop the RESTful service (.NET Framework web API)
- Use the RESTful service in the user journey
- Send input claims and read them in your code
- Validate the user's first name
- Send back a loyalty number
- Add the loyalty number to a JSON Web Token (JWT)

Prerequisites

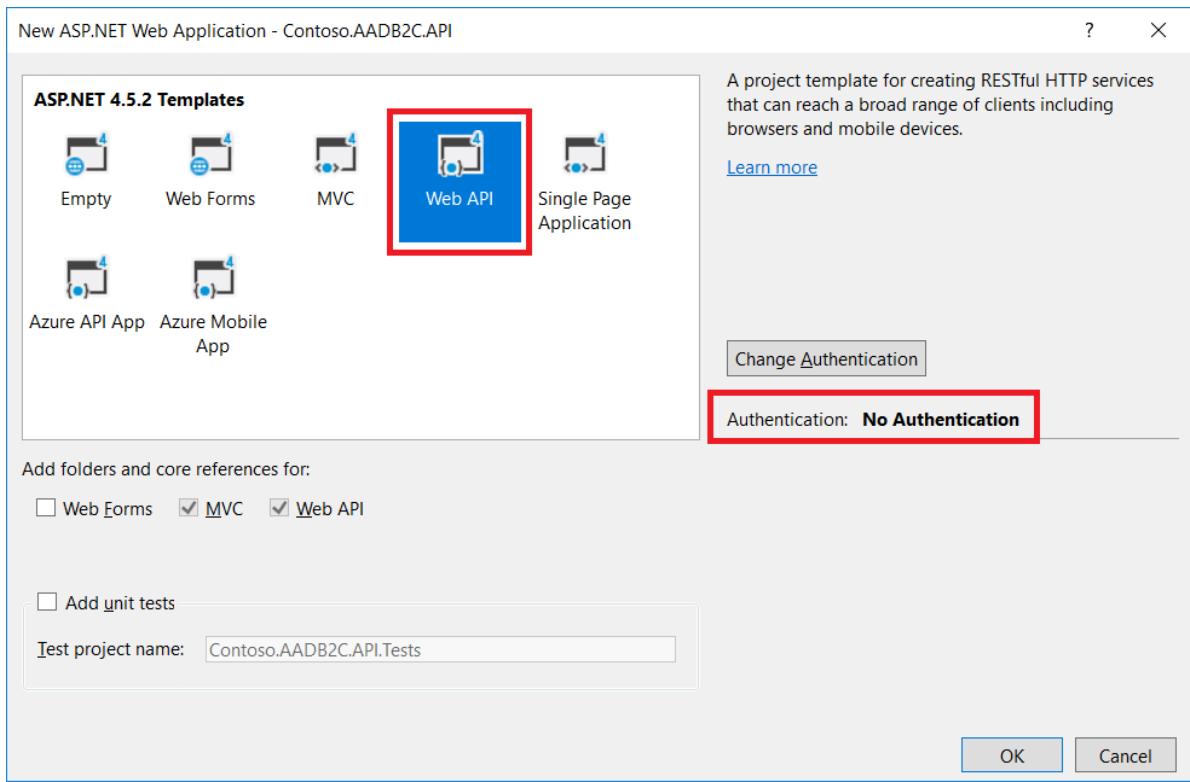
Complete the steps in the [Getting started with custom policies](#) article.

Step 1: Create an ASP.NET web API

1. In Visual Studio, create a project by selecting **File > New > Project**.
2. In the **New Project** window, select **Visual C# > Web > ASP.NET Web Application (.NET Framework)**.
3. In the **Name** box, type a name for the application (for example, *Contoso.AADB2C.API*), and then select **OK**.



4. In the **New ASP.NET Web Application** window, select a **Web API** or **Azure API app** template.



5. Make sure that authentication is set to **No Authentication**.

6. Select **OK** to create the project.

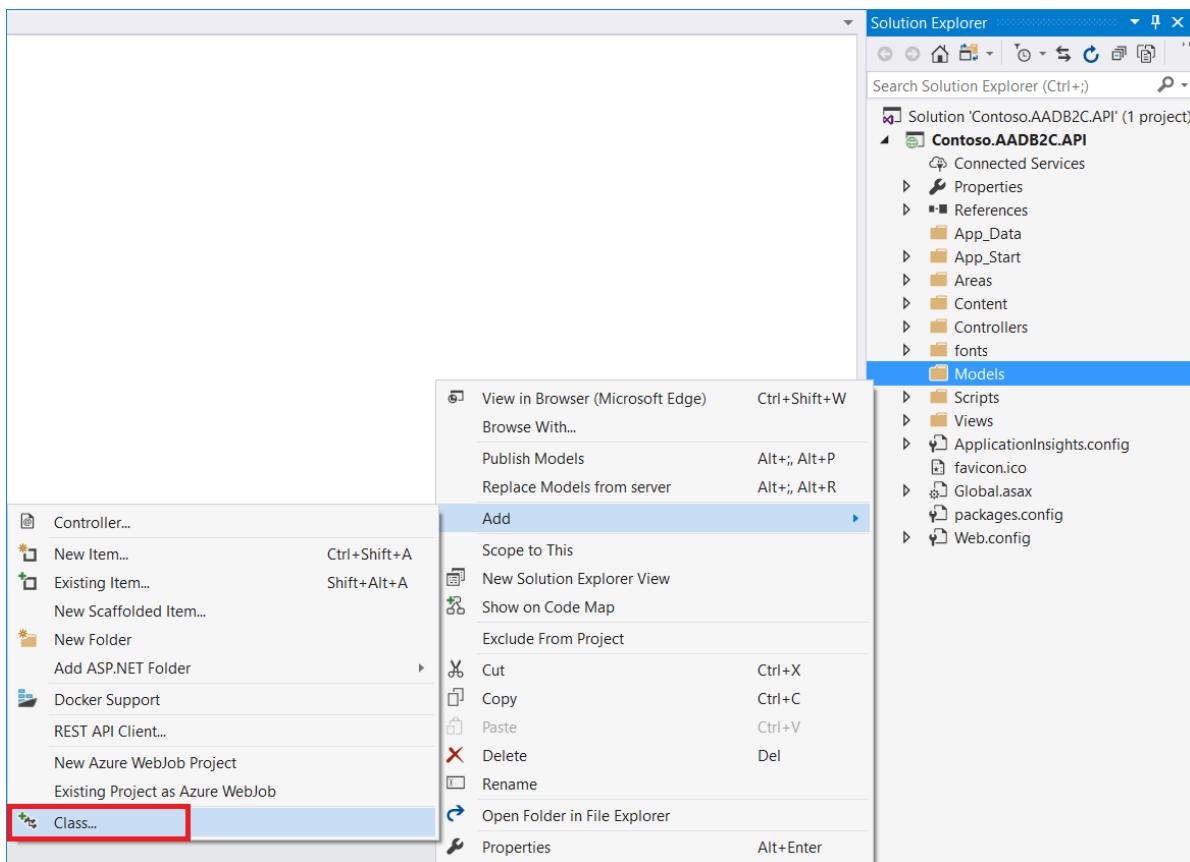
Step 2: Prepare the REST API endpoint

Step 2.1: Add data models

The models represent the input claims and output claims data in your RESTful service. Your code reads the input data by deserializing the input claims model from a JSON string to a C# object (your model). The ASP.NET web API automatically deserializes the output claims model back to JSON and then writes the serialized data to the body of the HTTP response message.

Create a model that represents input claims by doing the following:

1. If Solution Explorer is not already open, select **View > Solution Explorer**.
2. In Solution Explorer, right-click the **Models** folder, select **Add**, and then select **Class**.



3. Name the class `InputClaimsModel`, and then add the following properties to the `InputClaimsModel` class:

```
namespace Contoso.AADB2C.API.Models
{
    public class InputClaimsModel
    {
        public string email { get; set; }
        public string firstName { get; set; }
        public string lastName { get; set; }
    }
}
```

4. Create a new model, `OutputClaimsModel`, and then add the following properties to the `OutputClaimsModel` class:

```
namespace Contoso.AADB2C.API.Models
{
    public class OutputClaimsModel
    {
        public string loyaltyNumber { get; set; }
    }
}
```

5. Create one more model, `B2CResponseContent`, which you use to throw input validation error messages. Add the following properties to the `B2CResponseContent` class, provide the missing references, and then save the file:

```

namespace Contoso.AADB2C.API.Models
{
    public class B2CResponseContent
    {
        public string version { get; set; }
        public int status { get; set; }
        public string userMessage { get; set; }

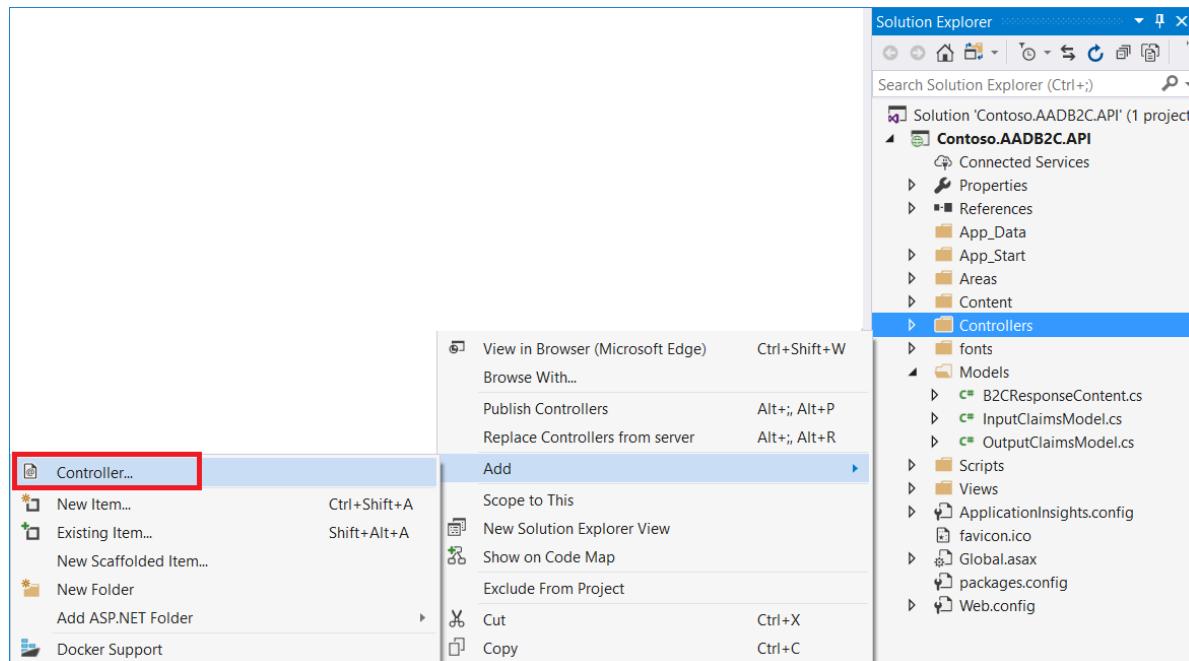
        public B2CResponseContent(string message, HttpStatusCode status)
        {
            this.userMessage = message;
            this.status = (int)status;
            this.version = Assembly.GetExecutingAssembly().GetName().Version.ToString();
        }
    }
}

```

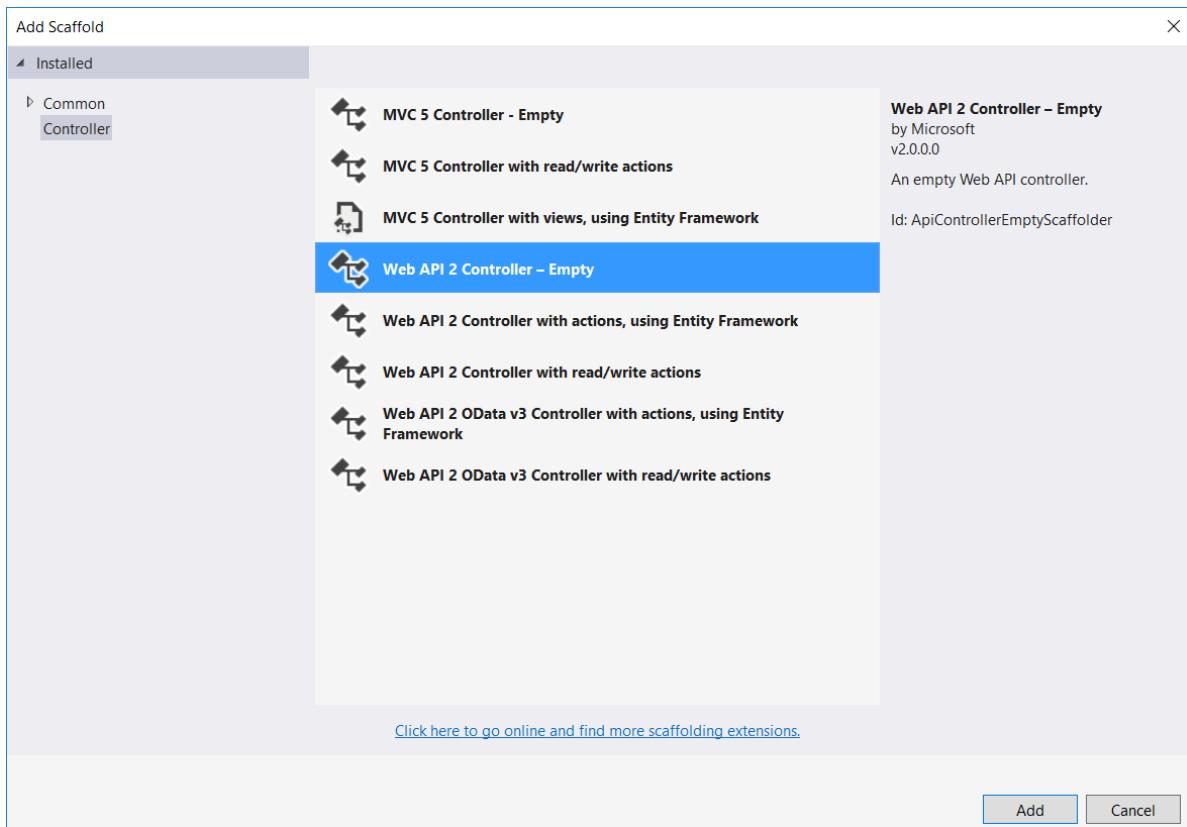
Step 2.2: Add a controller

In the web API, a **controller** is an object that handles HTTP requests. The controller returns output claims or, if the first name is not valid, throws a Conflict HTTP error message.

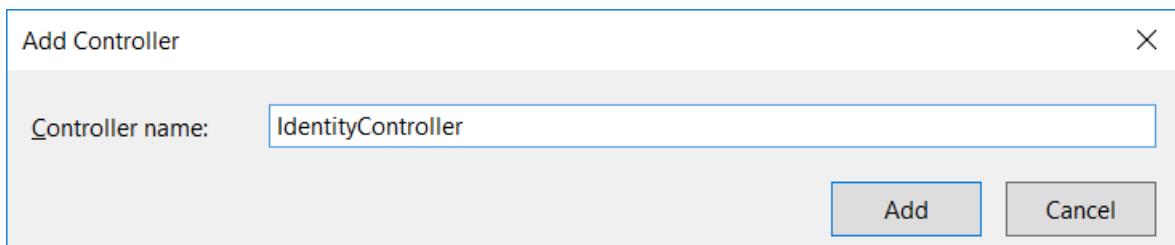
1. In Solution Explorer, right-click the **Controllers** folder, select **Add**, and then select **Controller**.



2. In the **Add Scaffold** window, select **Web API Controller - Empty**, and then select **Add**.



3. In the **Add Controller** window, name the controller **IdentityController**, and then select **Add**.



The scaffolding creates a file named *IdentityController.cs* in the *Controllers* folder.

4. If the *IdentityController.cs* file is not open already, double-click it, and then replace the code in the file with the following code:

```

using Contoso.AADB2C.API.Models;
using Newtonsoft.Json;
using System;
using System.NET;
using System.Web.Http;

namespace Contoso.AADB2C.API.Controllers
{
    public class IdentityController: ApiController
    {
        [HttpPost]
        public IHttpActionResult SignUp()
        {
            // If no data came in, then return
            if (this.Request.Content == null) throw new Exception();

            // Read the input claims from the request body
            string input = Request.Content.ReadAsStringAsync().Result;

            // Check the input content value
            if (string.IsNullOrEmpty(input))
            {
                return Content(HttpStatusCode.Conflict, new B2CResponseContent("Request content is empty", HttpStatusCode.Conflict));
            }

            // Convert the input string into an InputClaimsModel object
            InputClaimsModel inputClaims = JsonConvert.DeserializeObject<InputClaimsModel>(input,
typeof(InputClaimsModel)) as InputClaimsModel;

            if (inputClaims == null)
            {
                return Content(HttpStatusCode.Conflict, new B2CResponseContent("Can not deserialize input claims", HttpStatusCode.Conflict));
            }

            // Run an input validation
            if (inputClaims.firstName.ToLower() == "test")
            {
                return Content(HttpStatusCode.Conflict, new B2CResponseContent("Test name is not valid, please provide a valid name", HttpStatusCode.Conflict));
            }

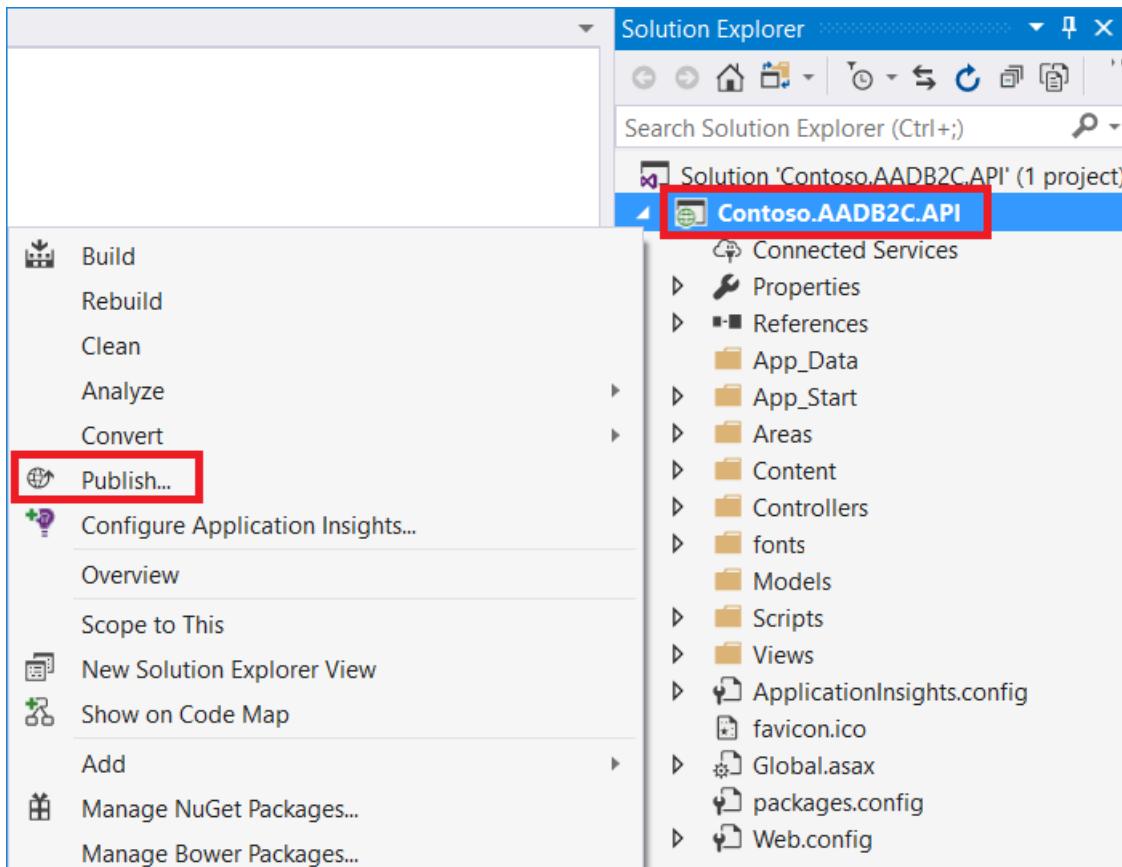
            // Create an output claims object and set the loyalty number with a random value
            OutputClaimsModel outputClaims = new OutputClaimsModel();
            outputClaims.loyaltyNumber = new Random().Next(100, 1000).ToString();

            // Return the output claim(s)
            return Ok(outputClaims);
        }
    }
}

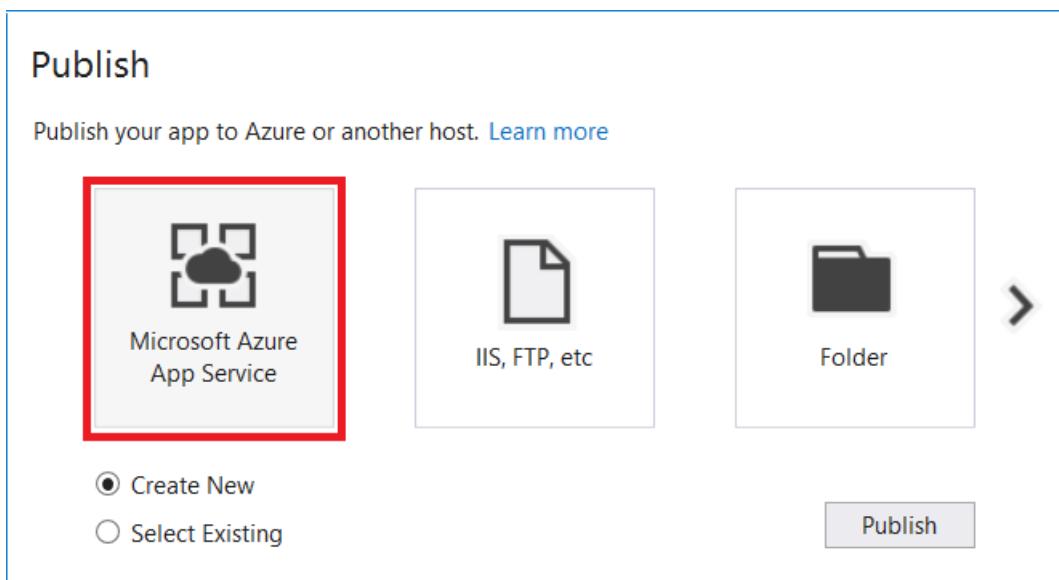
```

Step 3: Publish the project to Azure

1. In Solution Explorer, right-click the **Contoso.AADB2C.API** project, and then select **Publish**.



2. In the **Publish** window, select **Microsoft Azure App Service**, and then select **Publish**.



The **Create App Service** window opens. In it, you create all the necessary Azure resources to run the ASP.NET web app in Azure.

TIP

For more information about how to publish, see [Create an ASP.NET web app in Azure](#).

3. In the **Web App Name** box, type a unique app name (valid characters are a-z, 0-9, and hyphens (-)). The URL of the web app is `http://<app_name>.azurewebsites.NET`, where `app_name` is the name of your web app. You can accept the automatically generated name, which is unique.

Create App Service

Host your web and mobile applications, REST APIs, and more in Azure

Hosting Web App Name [Change Type ▾](#)

Services

Subscription

Resource Group [New...](#)

App Service Plan [New...](#)

Clicking the Create button will create the following Azure resources

[Explore additional Azure services](#)

App Service - ContosoAADB2CAPI

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.

[Learn More](#)

[Export...](#) [Create](#) [Cancel](#)

4. To start creating Azure resources, select **Create**. After the ASP.NET web app has been created, the wizard publishes it to Azure and then starts the app in the default browser.
5. Copy the web app's URL.

Step 4: Add the new `loyaltyNumber` claim to the schema of your `TrustFrameworkExtensions.xml` file

The `loyaltyNumber` claim is not yet defined in our schema. Add a definition within the `<BuildingBlocks>` element, which you can find at the beginning of the `TrustFrameworkExtensions.xml` file.

```

<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="loyaltyNumber">
      <DisplayName>loyaltyNumber</DisplayName>
      <DataType>string</DataType>
      <UserHelpText>Customer loyalty number</UserHelpText>
    </ClaimType>
  </ClaimsSchema>
</BuildingBlocks>

```

Step 5: Add a claims provider

Every claims provider must have one or more technical profiles, which determine the endpoints and protocols needed to communicate with the claims provider.

A claims provider can have multiple technical profiles for various reasons. For example, multiple technical profiles might be defined because the claims provider supports multiple protocols, endpoints can have varying capabilities, or releases can contain claims that have a variety of assurance levels. It might be acceptable to release sensitive claims in one user journey but not in another.

The following XML snippet contains a claims provider node with two technical profiles:

- **TechnicalProfile Id="REST-API-SignUp"**: Defines your RESTful service.

- **Proprietary** is described as the protocol for a RESTful-based provider.
- **InputClaims** defines the claims that will be sent from Azure AD B2C to the REST service.

In this example, the content of the claim **givenName** sends to the REST service as **firstName**, the content of the claim **surname** sends to the REST service as **lastName**, and **email** sends as is. The **OutputClaims** element defines the claims that are retrieved from RESTful service back to Azure AD B2C.

- **TechnicalProfile Id="LocalAccountSignUpWithLogonEmail"**: Adds a validation technical profile to an existing technical profile (defined in base policy). During the sign-up journey, the validation technical profile invokes the preceding technical profile. If the RESTful service returns an HTTP error 409 (a conflict error), the error message is displayed to the user.

Locate the **<ClaimsProviders>** node, and then add the following XML snippet under the **<ClaimsProviders>** node:

```
<ClaimsProvider>
  <DisplayName>REST APIs</DisplayName>
  <TechnicalProfiles>

    <!-- Custom Restful service -->
    <TechnicalProfile Id="REST-API-SignUp">
      <DisplayName>Validate user's input data and return loyaltyNumber claim</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://your-app-name.azurewebsites.NET/api/identity/signup</Item>
        <Item Key="SendClaimsIn">Body</Item>
        <!-- Set AuthenticationType to Basic or ClientCertificate in production environments -->
        <Item Key="AuthenticationType">None</Item>
        <!-- REMOVE the following line in production environments -->
        <Item Key="AllowInsecureAuthInProduction">true</Item>
      </Metadata>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="email" />
        <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
        <InputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
      </InputClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="loyaltyNumber" PartnerClaimType="loyaltyNumber" />
      </OutputClaims>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
    </TechnicalProfile>

    <!-- Change LocalAccountSignUpWithLogonEmail technical profile to support your validation technical
    profile -->
    <TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="loyaltyNumber" PartnerClaimType="loyaltyNumber" />
      </OutputClaims>
      <ValidationTechnicalProfiles>
        <ValidationTechnicalProfile ReferenceId="REST-API-SignUp" />
      </ValidationTechnicalProfiles>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

The comments above **AuthenticationType** and **AllowInsecureAuthInProduction** specify changes you should make when you move to a production environment. To learn how to secure your RESTful APIs for production, see

Secure RESTful APIs with basic auth and [Secure RESTful APIs with certificate auth](#).

Step 6: Add the `loyaltyNumber` claim to your relying party policy file so the claim is sent to your application

Edit your `SignUpOrSignIn.xml` relying party (RP) file, and modify the `TechnicalProfile Id="PolicyProfile"` element to add the following: `<OutputClaim ClaimTypeReferenceId="loyaltyNumber" />`.

After you add the new claim, the relying party code looks like this:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
      <OutputClaim ClaimTypeReferenceId="loyaltyNumber" DefaultValue="" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>
</TrustFrameworkPolicy>
```

Step 7: Upload the policy to your tenant

1. In the [Azure portal](#), Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
2. In the Azure portal, search for and select **Azure AD B2C**.
3. Select **Identity Experience Framework**.
4. Open **All Policies**.
5. Select **Upload Policy**.
6. Select the **Overwrite the policy if it exists** check box.
7. Upload the `TrustFrameworkExtensions.xml` file, and ensure that it passes validation.
8. Repeat the preceding step with the `SignUpOrSignIn.xml` file.

Step 8: Test the custom policy by using Run Now

1. Select **Azure AD B2C Settings**, and then go to **Identity Experience Framework**.

NOTE

Run now requires at least one application to be preregistered on the tenant. To learn how to register applications, see the Azure AD B2C [Get started](#) article or the [Application registration](#) article.

2. Open **B2C_1A_signup_signin**, the relying party (RP) custom policy that you uploaded, and then select **Run now**.

B2C_1A_signup_signin

Edit Delete Download

https://login.microsoftonline.com/butterfliesdemo.onmicrosoft.com/v2.0/.well-known/openid-configuration?p=B2C_1A_signup_signin

RUN POLICY SETTINGS

Select application
B2C demo

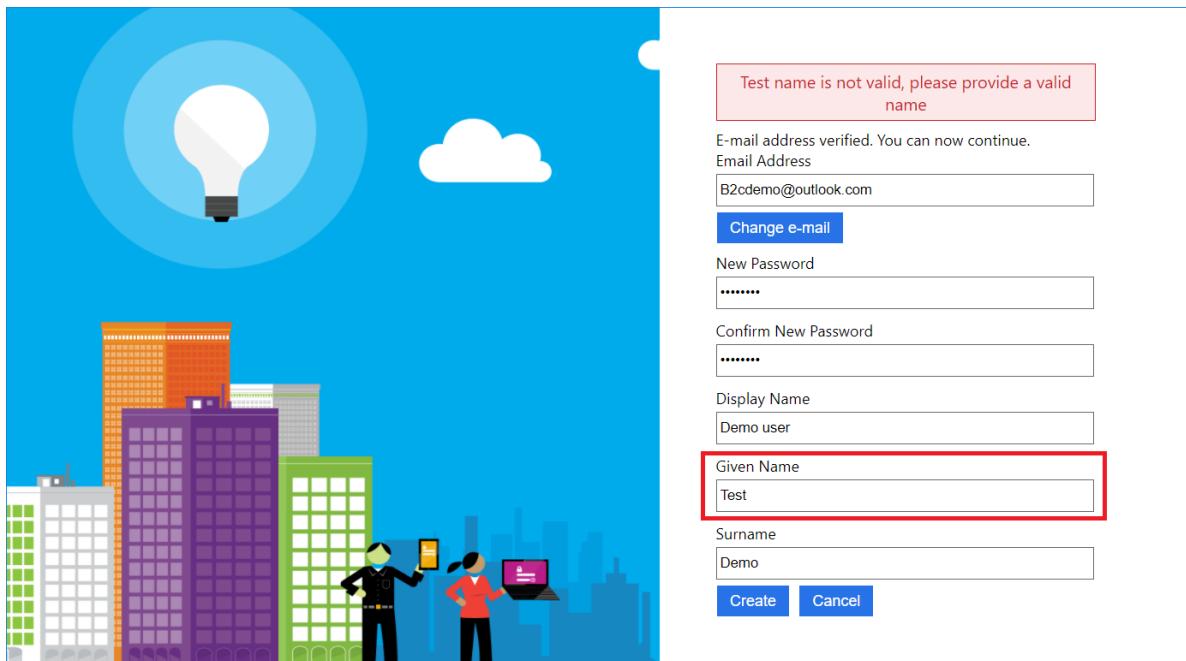
Select reply url
https://jwt.ms

ACCESS TOKENS

Run now endpoint <https://login.microsoftonline.com/butterfliesdemo.onmicrosoft.com/oauth2/v2.0/authorize...> 

Run now

3. Test the process by typing **Test** in the **Given Name** box. Azure AD B2C displays an error message at the top of the window.



4. In the **Given Name** box, type a name (other than "Test"). Azure AD B2C signs up the user and then sends a loyaltyNumber to your application. Note the number in this JWT.

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "X5eXk4xyojNFum1kl2Ytv8d1NP4-c57d06QGTVBwaNk"  
}.{  
  "exp": 1507125903,  
  "nbf": 1507122303,  
  "ver": "1.0",  
  "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",  
  "aud": "e1d2612f-c2bc-4599-8e7b-d874eaca1ee1",  
  "acr": "b2c_1a_signup_signin",  
  "nonce": "defaultNonce",  
  "iat": 1507122303,  
  "auth_time": 1507122303,  
  "loyaltyNumber": "290",  
  "given_name": "Emily",  
  "emails": ["B2cdemo@outlook.com"]  
}
```

(Optional) Download the complete policy files and code

- After you complete the [Get started with custom policies](#) walkthrough, we recommend that you build your scenario by using your own custom policy files. For your reference, we have provided [Sample policy files](#).
- You can download the complete code from [Sample Visual Studio solution for reference](#).

Next steps

Your next task is to secure your RESTful API using basic or client certificate authentication. To learn how to secure your APIs, see the following articles:

- [Secure your RESTful API with basic authentication \(username and password\)](#)
- [Secure your RESTful API with client certificates](#)

For information about all the elements available in a RESTful technical profile, see [Reference: RESTful technical profile](#).

Secure your RESTful services by using HTTP basic authentication

1/28/2020 • 6 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In a [related Azure AD B2C article](#), you create a RESTful service (web API) that integrates with Azure Active Directory B2C (Azure AD B2C) user journeys without authentication.

In this article, you add HTTP basic authentication to your RESTful service so that only verified users, including B2C, can access your API. With HTTP basic authentication, you set the user credentials (app ID and app secret) in your custom policy.

For more information, see [Basic authentication in ASP.NET web API](#).

Prerequisites

Complete the steps in the [Integrate REST API claims exchanges in your Azure AD B2C user journey](#) article.

Step 1: Add authentication support

Step 1.1: Add application settings to your project's web.config file

1. Open the Visual Studio project that you created earlier.
2. Add the following application settings to the web.config file under the `appSettings` element:

```
<add key="WebApp:ClientId" value="B2CServiceUserAccount" />
<add key="WebApp:ClientSecret" value="your secret" />
```

3. Create a password, and then set the `WebApp:ClientSecret` value.

To generate a complex password, run the following PowerShell code. You can use any arbitrary value.

```
$bytes = New-Object Byte[] 32
$rand = [System.Security.Cryptography.RandomNumberGenerator]::Create()
$rand.GetBytes($bytes)
$rand.Dispose()
[System.Convert]::ToBase64String($bytes)
```

Step 1.2: Install OWIN libraries

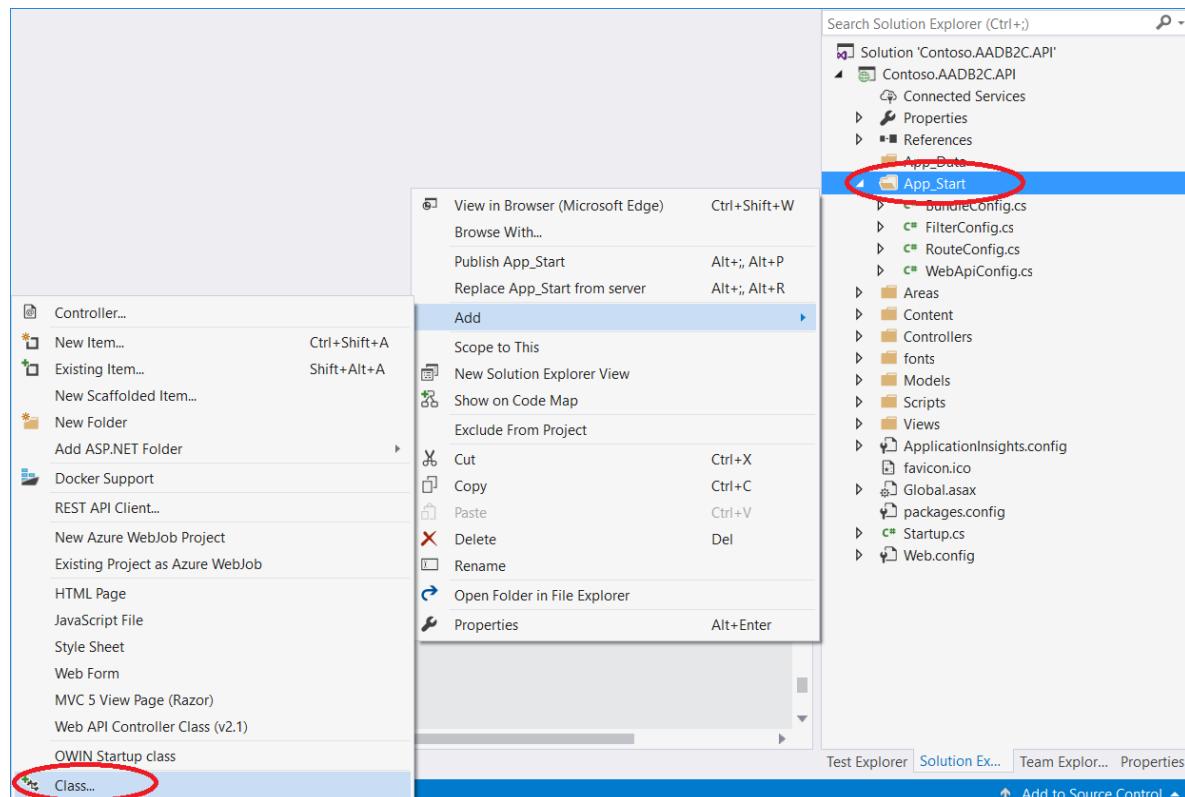
To begin, add the OWIN middleware NuGet packages to the project by using the Visual Studio Package Manager Console:

```
PM> Install-Package Microsoft.Owin
PM> Install-Package Owin
PM> Install-Package Microsoft.Owin.Host.SystemWeb
```

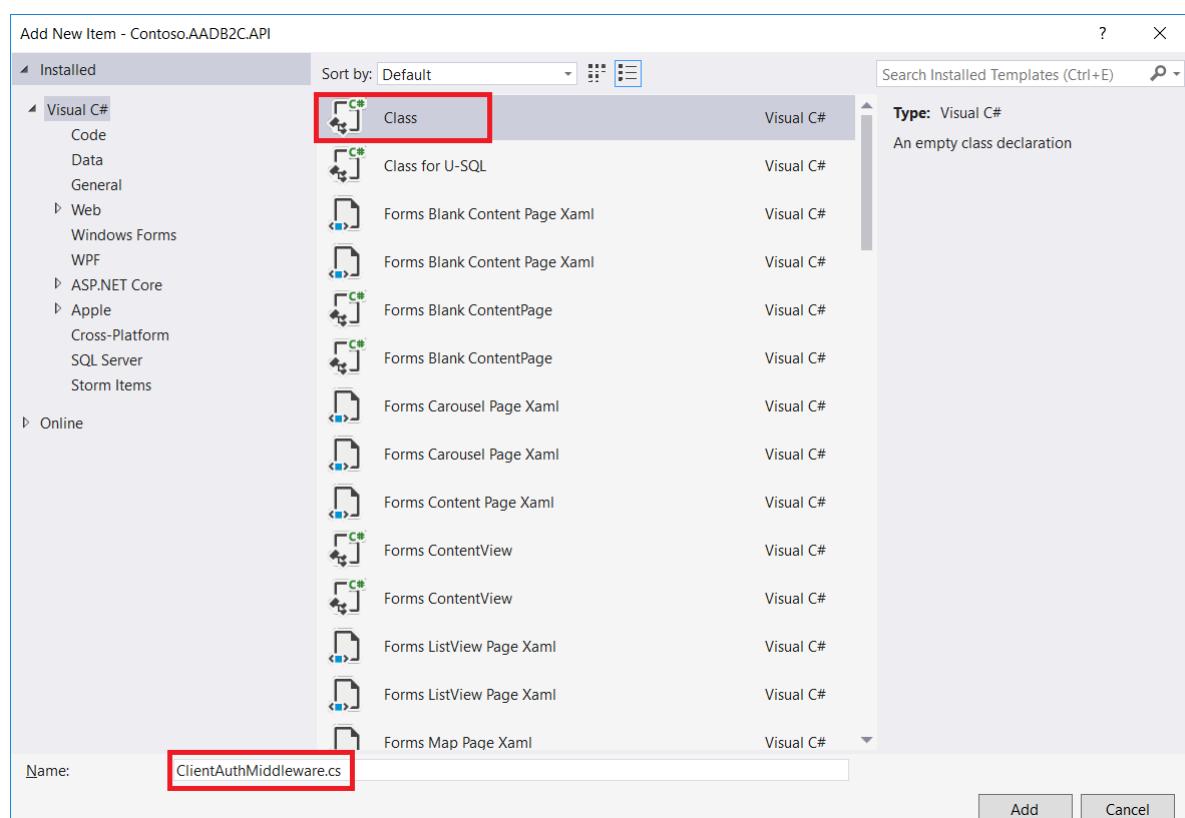
Step 1.3: Add an authentication middleware class

Add the `ClientAuthMiddleware.cs` class under the `App_Start` folder. To do so:

1. Right-click the `App_Start` folder, select **Add**, and then select **Class**.



2. In the **Name** box, type `ClientAuthMiddleware.cs`.



3. Open the *App_Start\ClientAuthMiddleware.cs* file, and replace the file content with following code:

```
using Microsoft.Owin;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Security.Principal;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace Contoso.AADB2C.API
{
    /// <summary>
    /// Class to create a custom owin middleware to check for client authentication
    /// </summary>
    public class ClientAuthMiddleware
    {
        private static readonly string ClientID = ConfigurationManager.AppSettings["WebApp:ClientId"];
        private static readonly string ClientSecret =
            ConfigurationManager.AppSettings["WebApp:ClientSecret"];

        /// <summary>
        /// Gets or sets the next owin middleware
        /// </summary>
        private Func<IDictionary<string, object>, Task> Next { get; set; }

        /// <summary>
        /// Initializes a new instance of the <see cref="ClientAuthMiddleware"/> class.
        /// </summary>
        /// <param name="next"></param>
        public ClientAuthMiddleware(Func<IDictionary<string, object>, Task> next)
        {
            this.Next = next;
        }

        /// <summary>
        /// Invoke client authentication middleware during each request.
        /// </summary>
        /// <param name="environment">Owin environment</param>
        /// <returns></returns>
        public Task Invoke(IDictionary<string, object> environment)
        {
            // Get wrapper class for the environment
            var context = new OwinContext(environment);

            // Check whether the authorization header is available. This contains the credentials.
            var authzValue = context.Request.Headers.Get("Authorization");
            if (string.IsNullOrEmpty(authzValue) || !authzValue.StartsWith("Basic ",
StringComparison.OrdinalIgnoreCase))
            {
                // Process next middleware
                return Next(environment);
            }

            // Get credentials
            var creds = authzValue.Substring("Basic ".Length).Trim();
            string clientId;
            string clientSecret;

            if (RetrieveCreds(creds, out clientId, out clientSecret))
            {
                // Set transaction authenticated as client
                context.Request.User = new GenericPrincipal(new GenericIdentity(clientId, "client"),
new string[] { "client" });
            }
        }
    }
}
```

```

        return Next(environment);
    }

    /// <summary>
    /// Retrieve credentials from header
    /// </summary>
    /// <param name="credentials">Authorization header</param>
    /// <param name="clientId">Client identifier</param>
    /// <param name="clientSecret">Client secret</param>
    /// <returns>True if valid credentials were presented</returns>
    private bool RetrieveCreds(string credentials, out string clientId, out string clientSecret)
    {
        string pair;
        clientId = clientSecret = string.Empty;

        try
        {
            pair = Encoding.UTF8.GetString(Convert.FromBase64String(credentials));
        }
        catch (FormatException)
        {
            return false;
        }
        catch (ArgumentException)
        {
            return false;
        }

        var ix = pair.IndexOf(':');
        if (ix == -1)
        {
            return false;
        }

        clientId = pair.Substring(0, ix);
        clientSecret = pair.Substring(ix + 1);

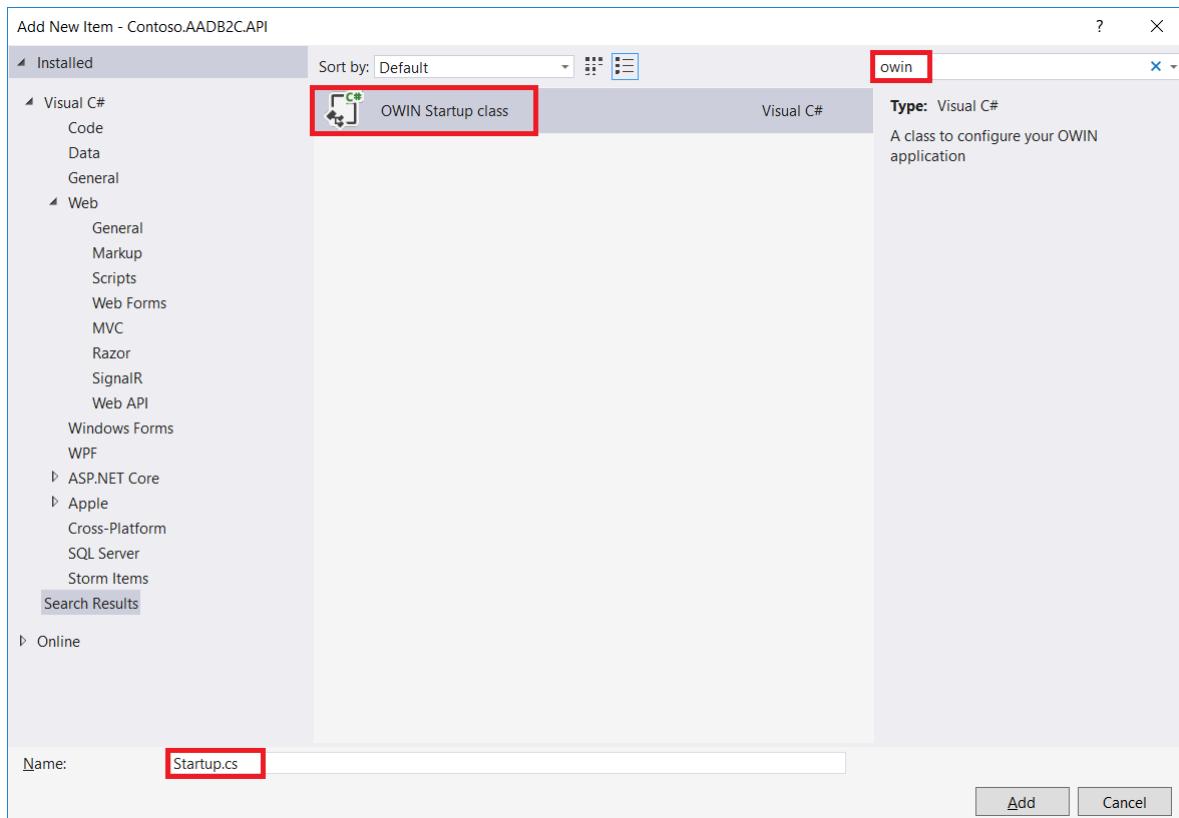
        // Return whether credentials are valid
        return (string.Compare(clientId, ClientAuthMiddleware.ClientID) == 0 &&
            string.Compare(clientSecret, ClientAuthMiddleware.ClientSecret) == 0);
    }
}
}

```

Step 1.4: Add an OWIN startup class

Add an OWIN startup class named `Startup.cs` to the API. To do so:

1. Right-click the project, select **Add > New Item**, and then search for **OWIN**.



2. Open the *Startup.cs* file, and replace the file content with following code:

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartup(typeof(Contoso.AADB2C.API.Startup))]
namespace Contoso.AADB2C.API
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.Use<ClientAuthMiddleware>();
        }
    }
}
```

Step 1.5: Protect the Identity API class

Open Controllers\IdentityController.cs, and add the `[Authorize]` tag to the controller class. This tag restricts access to the controller to users who meet the authorization requirement.

```
namespace ContosoAADB2C.API.Controllers
{
    [Authorize]
    0 references
    public class IdentityController : ApiController
    {
        [HttpPost]
        0 references | 0 requests | 0 exceptions
        public IHttpActionResult SignUp()
        {
            if (this.Request.Content == null) throw new Exception();

            // Read the input claims from the request body
            string input = Request.Content.ReadAsStringAsync().Result;
        }
    }
}
```

Step 2: Publish to Azure

To publish your project, in Solution Explorer, right-click the **ContosoAADB2C.API** project, and then select **Publish**.

Step 3: Add the RESTful services app ID and app secret to Azure AD B2C

After your RESTful service is protected by the client ID (username) and secret, you must store the credentials in your Azure AD B2C tenant. Your custom policy provides the credentials when it invokes your RESTful services.

Step 3.1: Add a RESTful services client ID

1. In your Azure AD B2C tenant, select **B2C Settings > Identity Experience Framework**.
2. Select **Policy Keys** to view the keys that are available in your tenant.
3. Select **Add**.
4. For **Options**, select **Manual**.
5. For **Name**, type **B2cRestClientId**. The prefix *B2C_1A_* might be added automatically.
6. In the **Secret** box, enter the app ID that you defined earlier.
7. For **Key usage**, select **Signature**.
8. Select **Create**.
9. Confirm that you've created the **B2C_1A_B2cRestClientId** key.

Step 3.2: Add a RESTful services client secret

1. In your Azure AD B2C tenant, select **B2C Settings > Identity Experience Framework**.
2. Select **Policy Keys** to view the keys available in your tenant.
3. Select **Add**.
4. For **Options**, select **Manual**.
5. For **Name**, type **B2cRestClientSecret**. The prefix *B2C_1A_* might be added automatically.
6. In the **Secret** box, enter the app secret that you defined earlier.

7. For **Key usage**, select **Signature**.

8. Select **Create**.

9. Confirm that you've created the `B2C_1A_B2cRestClientSecret` key.

Step 4: Change the technical profile to support basic authentication in your extension policy

1. In your working directory, open the extension policy file (`TrustFrameworkExtensions.xml`).

2. Search for the `<TechnicalProfile>` node that includes `Id="REST-API-SignUp"`.

3. Locate the `<Metadata>` element.

4. Change the *AuthenticationType* to *Basic*, as follows:

```
<Item Key="AuthenticationType">Basic</Item>
```

5. Immediately after the closing `<Metadata>` element, add the following XML snippet:

```
<CryptographicKeys>
  <Key Id="BasicAuthenticationUsername" StorageReferenceId="B2C_1A_B2cRestClientId" />
  <Key Id="BasicAuthenticationPassword" StorageReferenceId="B2C_1A_B2cRestClientSecret" />
</CryptographicKeys>
```

After you add the snippet, your technical profile should look like the following XML code:

```
<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>REST APIs</DisplayName>
    <TechnicalProfiles>
      <TechnicalProfile Id="REST-API-SignUp">
        <DisplayName>Validate user's input data and return loyaltyNumber claim</DisplayName>
        <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
        <Metadata>
          <Item Key="ServiceUrl">http://yourdomain.azurewebsites.net/api/identity/signup</Item>
          <Item Key="SendClaimsIn">Body</Item>
          <Item Key="AuthenticationType">Basic</Item>
        </Metadata>
        <CryptographicKeys>
          <Key Id="BasicAuthenticationUsername" StorageReferenceId="B2C_1A_B2cRestClientId" />
          <Key Id="BasicAuthenticationPassword" StorageReferenceId="B2C_1A_B2cRestClientSecret" />
        </CryptographicKeys>
        <InputClaims>
          <InputClaim ClaimTypeReferenceId="email" />
          <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
          <InputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
        </InputClaims>
      </TechnicalProfile>
    </TechnicalProfiles>
  </ClaimsProvider>
</ClaimsProviders>
```

Step 5: Upload the policy to your tenant

1. In the [Azure portal](#), select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.

2. In the Azure portal, search for and select **Azure AD B2C**.

3. Select **Identity Experience Framework**.

4. Open **All Policies**.

5. Select **Upload Policy**.

6. Select the **Overwrite the policy if it exists** check box.
7. Upload the *TrustFrameworkExtensions.xml* file, and then ensure that it passes validation.

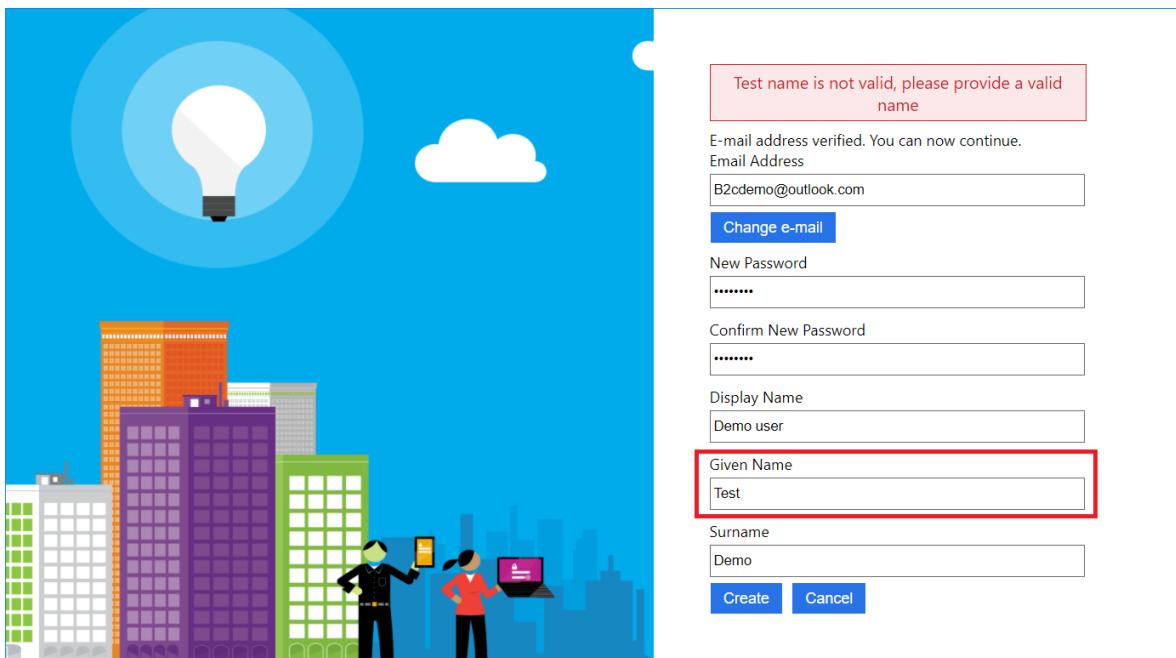
Step 6: Test the custom policy by using Run Now

1. Open **Azure AD B2C Settings**, and then select **Identity Experience Framework**.

NOTE

Run Now requires at least one application to be preregistered on the tenant. To learn how to register applications, see the Azure AD B2C [Get started](#) article or the [Application registration](#) article.

2. Open **B2C_1A_signup_signin**, the relying party (RP) custom policy that you uploaded, and then select **Run now**.
3. Test the process by typing **Test** in the **Given Name** box. Azure AD B2C displays an error message at the top of the window.



4. In the **Given Name** box, type a name (other than "Test"). Azure AD B2C signs up the user and then sends a loyalty number to your application. Note the number in this example:

```
{  
    "typ": "JWT",  
    "alg": "RS256",  
    "kid": "X5eXk4xyojNFum1kl2Ytv8d1NP4-c57d06QGTVBwaNk"  
}.{  
    "exp": 1507125903,  
    "nbf": 1507122303,  
    "ver": "1.0",  
    "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",  
    "aud": "e1d2612f-c2bc-4599-8e7b-d874eaca1ee1",  
    "acr": "b2c_1a_signup_signin",  
    "nonce": "defaultNonce",  
    "iat": 1507122303,  
    "auth_time": 1507122303,  
    "loyaltyNumber": "290",  
    "given_name": "Emily",  
    "emails": ["B2cdemo@outlook.com"]  
}
```

(Optional) Download the complete policy files and code

- After you complete the [Get started with custom policies](#) walkthrough, we recommend that you build your scenario by using your own custom policy files. For your reference, we have provided [Sample policy files](#).
- You can download the complete code from [Sample Visual Studio solution for reference](#).

Next steps

- [Use client certificates to secure your RESTful API](#)

Secure your RESTful service by using client certificates

1/28/2020 • 7 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In a related article, you [create a RESTful service](#) that interacts with Azure Active Directory B2C (Azure AD B2C).

In this article, you learn how to restrict access to your Azure web app (RESTful API) by using a client certificate. This mechanism is called TLS mutual authentication, or *client certificate authentication*. Only services that have proper certificates, such as Azure AD B2C, can access your service.

NOTE

You can also secure your RESTful service by using [HTTP basic authentication](#). However, HTTP basic authentication is considered less secure over a client certificate. Our recommendation is to secure the RESTful service by using client certificate authentication as described in this article.

This article details how to:

- Set up your web app to use client certificate authentication.
- Upload the certificate to Azure AD B2C policy keys.
- Configure your custom policy to use the client certificate.

Prerequisites

- Complete the steps in the [Integrate REST API claims exchanges](#) article.
- Get a valid certificate (a .pfx file with a private key).

Step 1: Configure a web app for client certificate authentication

To set up **Azure App Service** to require client certificates, set the web app `clientCertEnabled` site setting to `true`. To make this change, in the Azure portal, open your web app page. In the left navigation, under **Settings** select **SSL Settings**. In the **Client Certificates** section, turn on the **Incoming client certificate** option.

NOTE

Make sure that your Azure App Service plan is Standard or greater. For more information, see [Azure App Service plans in-depth overview](#).

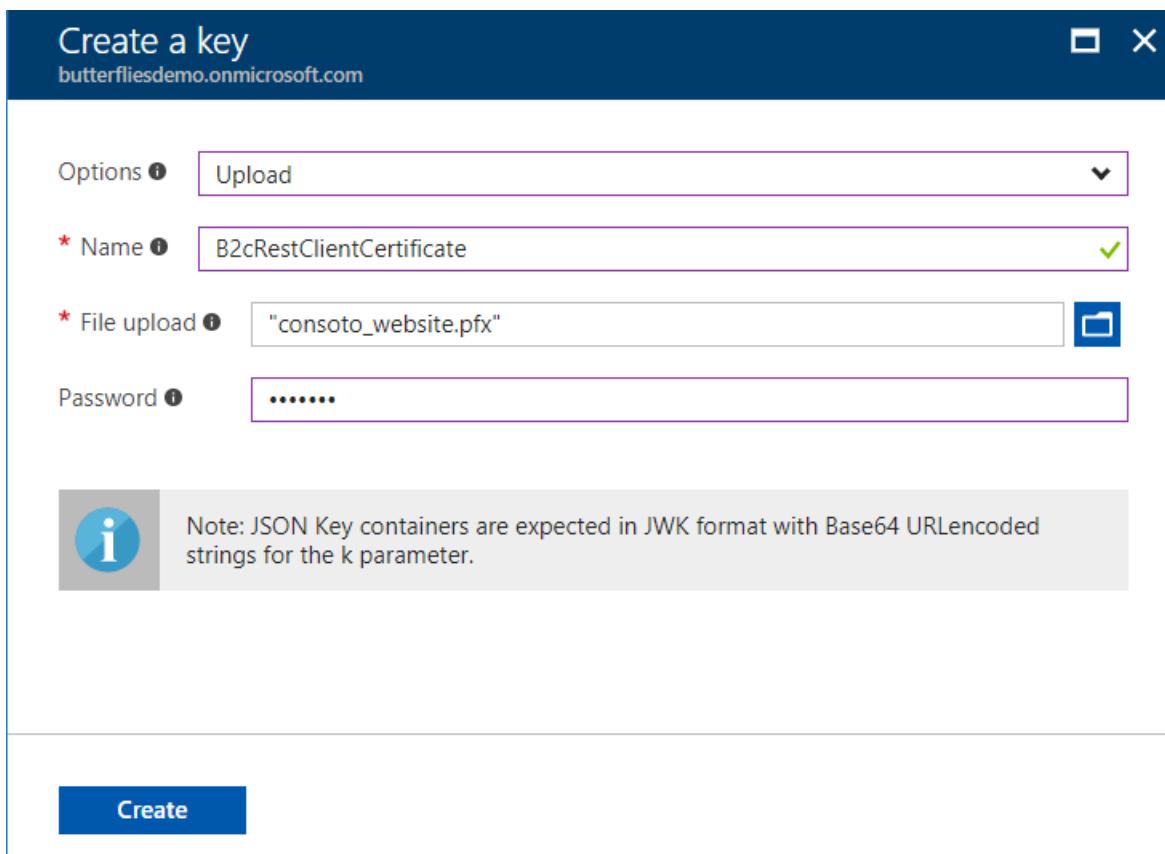
NOTE

For more information about setting the **clientCertEnabled** property, see [Configure TLS mutual authentication for web apps](#).

Step 2: Upload your certificate to Azure AD B2C policy keys

After you set `clientCertEnabled` to *true*, the communication with your RESTful API requires a client certificate. To obtain, upload, and store the client certificate in your Azure AD B2C tenant, do the following:

1. In your Azure AD B2C tenant, select **B2C Settings > Identity Experience Framework**.
2. To view the keys that are available in your tenant, select **Policy Keys**.
3. Select **Add**. The **Create a key** window opens.
4. In the **Options** box, select **Upload**.
5. In the **Name** box, type **B2cRestClientCertificate**. The prefix *B2C_1A_* is added automatically.
6. In the **File upload** box, select your certificate's .pfx file with a private key.
7. In the **Password** box, type the certificate's password.



8. Select **Create**.
9. To view the keys that are available in your tenant and confirm that you've created the `B2C_1A_B2cRestClientCertificate` key, select **Policy Keys**.

Step 3: Change the technical profile

To support client certificate authentication in your custom policy, change the technical profile by doing the following:

1. In your working directory, open the *TrustFrameworkExtensions.xml* extension policy file.
2. Search for the `<TechnicalProfile>` node that includes `Id="REST-API-SignUp"`.
3. Locate the `<Metadata>` element.
4. Change the `AuthenticationType` to `ClientCertificate`, as follows:

```
<Item Key="AuthenticationType">ClientCertificate</Item>
```

5. Immediately after the closing `<Metadata>` element, add the following XML snippet:

```
<CryptographicKeys>
  <Key Id="ClientCertificate" StorageReferenceId="B2C_1A_B2cRestClientCertificate" />
</CryptographicKeys>
```

After you add the snippet, your technical profile should look like the following XML code:

```
<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>REST APIs</DisplayName>
    <TechnicalProfiles>
      <TechnicalProfile Id="REST-API-SignUp">
        <DisplayName>Validate user's input data and return loyaltyNumber claim</DisplayName>
        <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
        <Metadata>
          <Item Key="ServiceUrl">https://yourdomain.azurewebsites.net/api/identity/signup</Item>
          <Item Key="SendClaimsIn">Body</Item>
          <Item Key="AuthenticationType">ClientCertificate</Item>
        </Metadata>
        <CryptographicKeys>
          <Key Id="ClientCertificate" StorageReferenceId="B2C_1A_B2CCClientCertificate" />
        </CryptographicKeys>
        <InputClaims>
          <InputClaim ClaimTypeReferenceId="email" />
          <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
          <InputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
        </InputClaims>
      </TechnicalProfile>
    </TechnicalProfiles>
  </ClaimsProvider>
</ClaimsProviders>
```

Step 4: Upload the policy to your tenant

1. In the [Azure portal](#), select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
2. In the Azure portal, search for and select **Azure AD B2C**.
3. Select **Identity Experience Framework**.
4. Select **All Policies**.
5. Select **Upload Policy**.
6. Select the **Overwrite the policy if it exists** check box.
7. Upload the *TrustFrameworkExtensions.xml* file, and then ensure that it passes validation.

Step 5: Test the custom policy by using Run Now

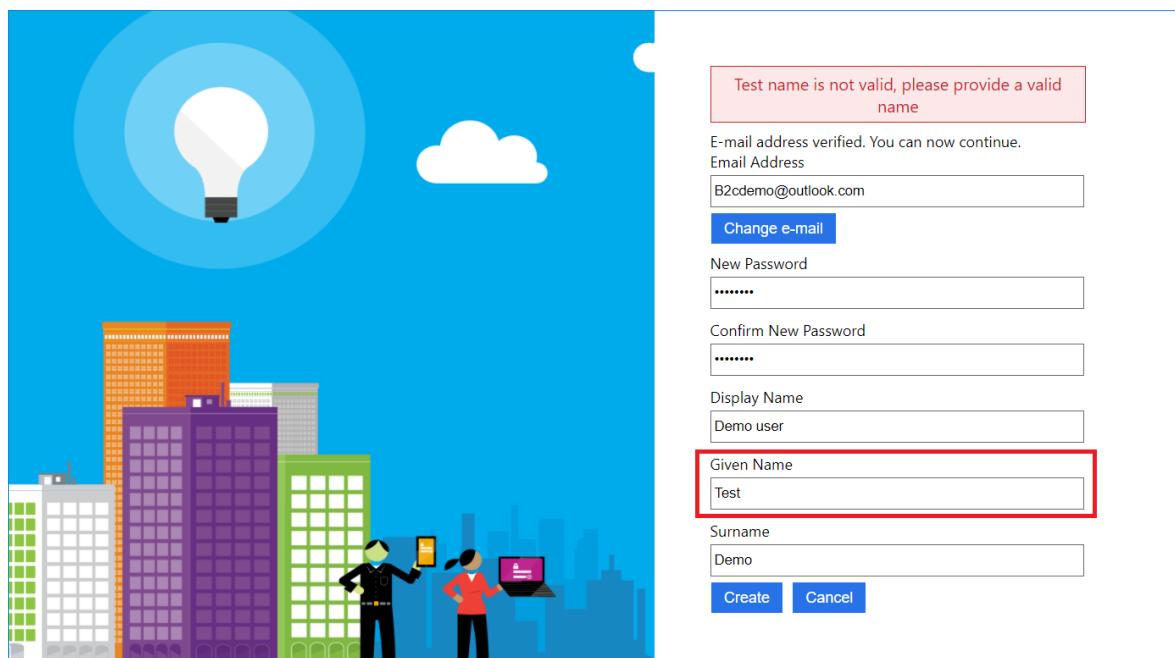
1. Open **Azure AD B2C Settings**, and then select **Identity Experience Framework**.

NOTE

Run Now requires at least one application to be preregistered on the tenant. To learn how to register applications, see the Azure AD B2C [Get started](#) article or the [Application registration](#) article.

2. Open **B2C_1A_signup_signin**, the relying party (RP) custom policy that you uploaded, and then select **Run now**.
3. Test the process by typing **Test** in the **Given Name** box. Azure AD B2C displays an error message at the

top of the window.



4. In the **Given Name** box, type a name (other than "Test"). Azure AD B2C signs up the user and then sends a loyalty number to your application. Note the number in this JWT example:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "X5eXk4xyojNFum1k12Ytv8dlNP4-c57d06QGTVBwaNk"
}.{
  "exp": 1507125903,
  "nbf": 1507122303,
  "ver": "1.0",
  "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",
  "aud": "e1d2612f-c2bc-4599-8e7b-d874eaca1ee1",
  "acr": "b2c_1a_signup_signin",
  "nonce": "defaultNonce",
  "iat": 1507122303,
  "auth_time": 1507122303,
  "loyaltyNumber": "290",
  "given_name": "Emily",
  "emails": ["B2cdemo@outlook.com"]
}
```

NOTE

If you receive the error message, *The name is not valid, please provide a valid name*, it means that Azure AD B2C successfully called your RESTful service while it presented the client certificate. The next step is to validate the certificate.

Step 6: Add certificate validation

The client certificate that Azure AD B2C sends to your RESTful service does not undergo validation by the Azure App Service platform, except to check whether the certificate exists. Validating the certificate is the responsibility of the web app.

In this section, you add sample ASP.NET code that validates the certificate properties for authentication purposes.

NOTE

For more information about configuring Azure App Service for client certificate authentication, see [Configure TLS mutual authentication for web apps](#).

6.1 Add application settings to your project's web.config file

In the Visual Studio project that you created earlier, add the following application settings to the `web.config` file after the `appSettings` element:

```
<add key="ClientCertificate:Subject" value="CN=Subject name" />
<add key="ClientCertificate:Issuer" value="CN=Issuer name" />
<add key="ClientCertificate:Thumbprint" value="Certificate thumbprint" />
```

Replace the certificate's **Subject name**, **Issuer name**, and **Certificate thumbprint** values with your certificate values.

6.2 Add the IsValidClientCertificate function

Open the `Controllers\IdentityController.cs` file, and then add to the `Identity` controller class the following function:

```
private bool IsValidClientCertificate()
{
    string ClientCertificateSubject = ConfigurationManager.AppSettings["ClientCertificate:Subject"];
    string ClientCertificateIssuer = ConfigurationManager.AppSettings["ClientCertificate:Issuer"];
    string ClientCertificateThumbprint = ConfigurationManager.AppSettings["ClientCertificate:Thumbprint"];

    X509Certificate2 clientCertInRequest = RequestContext.ClientCertificate;
    if (clientCertInRequest == null)
    {
        Trace.WriteLine("Certificate is NULL");
        return false;
    }

    // Basic verification
    if (clientCertInRequest.Verify() == false)
    {
        Trace.TraceError("Basic verification failed");
        return false;
    }

    // 1. Check the time validity of the certificate
    if (DateTime.Compare(DateTime.Now, clientCertInRequest.NotBefore) < 0 ||
        DateTime.Compare(DateTime.Now, clientCertInRequest.NotAfter) > 0)
    {
        Trace.TraceError($"NotBefore '{clientCertInRequest.NotBefore}' or NotAfter '{clientCertInRequest.NotAfter}' not valid");
        return false;
    }

    // 2. Check the subject name of the certificate
    bool foundSubject = false;
    string[] certSubjectData = clientCertInRequest.Subject.Split(new char[] { ',' },
        StringSplitOptions.RemoveEmptyEntries);
    foreach (string s in certSubjectData)
    {
        if (String.Compare(s.Trim(), ClientCertificateSubject) == 0)
        {
            foundSubject = true;
            break;
        }
    }
}
```

```

        if (!foundSubject)
        {
            Trace.TraceError($"Subject name '{clientCertInRequest.Subject}' is not valid");
            return false;
        }

        // 3. Check the issuer name of the certificate
        bool foundIssuerCN = false;
        string[] certIssuerData = clientCertInRequest.Issuer.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
        foreach (string s in certIssuerData)
        {
            if (String.Compare(s.Trim(), ClientCertificateIssuer) == 0)
            {
                foundIssuerCN = true;
                break;
            }
        }

        if (!foundIssuerCN)
        {
            Trace.TraceError($"Issuer '{clientCertInRequest.Issuer}' is not valid");
            return false;
        }

        // 4. Check the thumbprint of the certificate
        if (String.Compare(clientCertInRequest.Thumbprint.Trim().ToUpper(), ClientCertificateThumbprint) != 0)
        {
            Trace.TraceError($"Thumbprint '{clientCertInRequest.Thumbprint.Trim().ToUpper()}' is not valid");
            return false;
        }

        // 5. If you also want to test whether the certificate chains to a trusted root authority, you can
        //uncomment the following code:
        //
        //X509Chain certChain = new X509Chain();
        //certChain.Build(certificate);
        //bool isValidCertChain = true;
        //foreach (X509ChainElement chElement in certChain.ChainElements)
        //{
        //    if (!chElement.Certificate.Verify())
        //    {
        //        isValidCertChain = false;
        //        break;
        //    }
        //}
        //if (!isValidCertChain) return false;
        return true;
    }
}

```

In the preceding sample code, we accept the certificate as valid only if all the following conditions are met:

- The certificate is not expired and is active for the current time on server.
- The `Subject` name of the certificate is equal to the `ClientCertificate:Subject` application setting value.
- The `Issuer` name of the certificate is equal to the `ClientCertificate:Issuer` application setting value.
- The `thumbprint` of the certificate is equal to the `ClientCertificate:Thumbprint` application setting value.

IMPORTANT

Depending on the sensitivity of your service, you might need to add more validations. For example, you might need to test whether the certificate chains to a trusted root authority, issuer organization name validation, and so on.

6.3 Call the `IsValidClientCertificate` function

Open the `Controllers\IdentityController.cs` file and then, at the beginning of the `SignUp()` function, add the following code snippet:

```
if (IsValidClientCertificate() == false)
{
    return Content(HttpStatusCode.Conflict, new B2CResponseContent("Your client certificate is not valid",
    HttpStatusCode.Conflict));
}
```

After you add the snippet, your `Identity` controller should look like the following code:

```
namespace Contoso.AADB2C.APIClientCert
{
    0 references
    public class IdentityController : ApiController
    {
        [HttpPost]
        0 references | 0 requests | 0 exceptions
        public IHttpActionResult SignUp()
        {
            if (IsValidClientCertificate() == false)
            {
                return Content(HttpStatusCode.Conflict, new B2CResponseContent("Your client certificate is not valid",
                HttpStatusCode.Conflict));
            }

            // If no data came in, then return
            if (this.Request.Content == null) throw new Exception();
        }
    }
}
```

Step 7: Publish your project to Azure and test it

1. In **Solution Explorer**, right-click the **Contoso.AADB2C.API** project, and then select **Publish**.
2. Repeat "Step 6," and re-test your custom policy with the certificate validation. Try to run the policy, and make sure that everything works after you add the validation.
3. In your `web.config` file, change the value of `ClientCertificate:Subject` to **invalid**. Run the policy again and you should see an error message.
4. Change the value back to **valid**, and make sure that the policy can call your REST API.

If you need to troubleshoot this step, see [Collecting logs by using Application Insights](#).

(Optional) Download the complete policy files and code

- After you complete the [Get started with custom policies](#) walkthrough, we recommend that you build your scenario by using your own custom policy files. For your reference, we have provided [Sample policy files](#).
- You can download the complete code from [Sample Visual Studio solution for reference](#).

Azure Active Directory B2C: Use custom attributes in a custom profile edit policy

2/20/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In this article, you create a custom attribute in your Azure Active Directory B2C (Azure AD B2C) directory. You'll use this new attribute as a custom claim in the profile edit user journey.

Prerequisites

Follow the steps in the article [Azure Active Directory B2C: Get started with custom policies](#).

Use custom attributes to collect information about your customers in Azure AD B2C by using custom policies

Your Azure AD B2C directory comes with a built-in set of attributes. Examples are **Given Name**, **Surname**, **City**, **Postal Code**, and **userPrincipalName**. You often need to create your own attributes like these examples:

- A customer-facing application needs to persist for an attribute like **LoyaltyNumber**.
- An identity provider has a unique user identifier like **uniqueUserGUID** that must be saved.
- A custom user journey needs to persist for a state of a user like **migrationStatus**.

Azure AD B2C extends the set of attributes stored on each user account. You can also read and write these attributes by using the [Microsoft Graph API](#).

Extension properties extend the schema of the user objects in the directory. The terms *extension property*, *custom attribute*, and *custom claim* refer to the same thing in the context of this article. The name varies depending on the context, such as application, object, or policy.

Extension properties can only be registered on an application object even though they might contain data for a user. The property is attached to the application. The application object must have write access to register an extension property. A hundred extension properties, across all types and all applications, can be written to any single object. Extension properties are added to the target directory type and become immediately accessible in the Azure AD B2C directory tenant. If the application is deleted, those extension properties along with any data contained in them for all users are also removed. If an extension property is deleted by the application, it's removed on the target directory objects, and the values are deleted.

Extension properties exist only in the context of a registered application in the tenant. The object ID of that application must be included in the **TechnicalProfile** that uses it.

NOTE

The Azure AD B2C directory typically includes a web app named `b2c-extensions-app`. This application is primarily used by the B2C built-in policies for the custom claims created via the Azure portal. We recommend that only advanced users register extensions for B2C custom policies by using this application. Instructions are included in the **Next steps** section in this article.

Create a new application to store the extension properties

1. Open a browsing session and navigate to the [Azure portal](#). Sign in with the administrative credentials of the B2C directory you want to configure.
2. Select **Azure Active Directory** on the left navigation menu. You might need to find it by selecting **More services**.
3. Select **App registrations**. Select **New application registration**.
4. Provide the following entries:
 - A name for the web application: **WebApp-GraphAPI-DirectoryExtensions**.
 - The application type: **Web app/API**.
 - The sign-on URL: **https://<tenantName>.onmicrosoft.com/WebApp-GraphAPI-DirectoryExtensions**.
5. Select **Create**.
6. Select the newly created web application.
7. Select **Settings > Required permissions**.
8. Select the API **Windows Azure Active Directory**.
9. Enter a checkmark in Application Permissions: **Read and write directory data**. Then select **Save**.
10. Choose **Grant permissions** and confirm **Yes**.
11. Copy the following identifiers to your clipboard and save them:
 - **Application ID**. Example: `103ee0e6-f92d-4183-b576-8c3739027780`.
 - **Object ID**. Example: `80d8296a-da0a-49ee-b6ab-fd232aa45201`.

Modify your custom policy to add the **ApplicationObjectId**

When you followed the steps in [Azure Active Directory B2C: Get started with custom policies](#), you downloaded and modified [sample files](#) named **TrustFrameworkBase.xml**, **TrustFrameworkExtensions.xml**, **SignUpOrSignin.xml**, **ProfileEdit.xml**, and **PasswordReset.xml**. In this step, you make more modifications to those files.

- Open the **TrustFrameworkBase.xml** file and add the `Metadata` section as shown in the following example. Insert the object ID that you previously recorded for the `ApplicationObjectId` value and the application ID that you recorded for the `clientId` value:

```

<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>Azure Active Directory</DisplayName>
    <TechnicalProfile Id="AAD-Common">
      <DisplayName>Azure Active Directory</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureActiveDirectoryProvider,
      Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

      <!-- Provide objectId and appId before using extension properties. -->
      <Metadata>
        <Item Key="ApplicationObjectId">insert objectId here</Item>
        <Item Key="ClientId">insert appId here</Item>
      </Metadata>
      <!-- End of changes -->

      <CryptographicKeys>
        <Key Id="issuer_secret" StorageReferenceId="TokenSigningKeyContainer" />
      </CryptographicKeys>
      <IncludeInSso>false</IncludeInSso>
      <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
    </TechnicalProfile>
  </ClaimsProvider>
</ClaimsProviders>

```

NOTE

When the **TechnicalProfile** writes for the first time to the newly created extension property, you might experience a one-time error. The extension property is created the first time it's used.

Use the new extension property or custom attribute in a user journey

1. Open the **ProfileEdit.xml** file.
2. Add a custom claim `loyaltyId`. By including the custom claim in the `<RelyingParty>` element, it's included in the token for the application.

```

<RelyingParty>
  <DefaultUserJourney ReferenceId="ProfileEdit" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="city" />

      <!-- Provide the custom claim identifier -->
      <OutputClaim ClaimTypeReferenceId="extension_loyaltyId" />
      <!-- End of changes -->
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>

```

3. Open the **TrustFrameworkExtensions.xml** file and add the `<claimsSchema>` element and its child elements to the `BuildingBlocks` element:

```

<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="extension_loyaltyId">
      <DisplayName>Loyalty Identification Tag</DisplayName>
      <DataType>string</DataType>
      <UserHelpText>Your loyalty number from your membership card</UserHelpText>
      <UserInputType>TextBox</UserInputType>
    </ClaimType>
  </ClaimsSchema>
</BuildingBlocks>

```

4. Add the same `ClaimType` definition to **TrustFrameworkBase.xml**. It's not necessary to add a `ClaimType` definition in both the base and the extensions files. However, the next steps add the `extension_loyaltyId` to **TechnicalProfiles** in the base file. So the policy validator rejects the upload of the base file without it. It might be useful to trace the execution of the user journey named **ProfileEdit** in the **TrustFrameworkBase.xml** file. Search for the user journey with the same name in your editor. Observe that Orchestration step 5 invokes the **TechnicalProfileReferenceID="SelfAsserted-ProfileUpdate**. Search and inspect this **TechnicalProfile** to familiarize yourself with the flow.

5. Open the **TrustFrameworkBase.xml** file and add `loyaltyId` as an input and output claim in the **TechnicalProfile SelfAsserted-ProfileUpdate**:

```

<TechnicalProfile Id="SelfAsserted-ProfileUpdate">
  <DisplayName>User ID signup</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider,
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="ContentDefinitionReferenceId">api.selfasserted.profileupdate</Item>
  </Metadata>
  <IncludeInSso>false</IncludeInSso>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="alternativeSecurityId" />
    <InputClaim ClaimTypeReferenceId="userPrincipalName" />
    <InputClaim ClaimTypeReferenceId="givenName" />
    <InputClaim ClaimTypeReferenceId="surname" />

    <!-- Add the loyalty identifier -->
    <InputClaim ClaimTypeReferenceId="extension_loyaltyId"/>
    <!-- End of changes -->
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="surname" />

    <!-- Add the loyalty identifier -->
    <OutputClaim ClaimTypeReferenceId="extension_loyaltyId"/>
    <!-- End of changes -->
  </OutputClaims>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="AAD-UserWriteProfileUsingObjectId" />
  </ValidationTechnicalProfiles>
</TechnicalProfile>

```

6. In the **TrustFrameworkBase.xml** file, add the `loyaltyId` claim to **TechnicalProfile AAD-UserWriteProfileUsingObjectId**. This addition persists the value of the claim in the extension property for the current user in the directory:

```

<TechnicalProfile Id="AAD-UserWriteProfileUsingObjectId">
  <Metadata>
    <Item Key="Operation">Write</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">false</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
  </Metadata>
  <IncludeInSso>false</IncludeInSso>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <PersistedClaims>
    <PersistedClaim ClaimTypeReferenceId="objectId" />
    <PersistedClaim ClaimTypeReferenceId="givenName" />
    <PersistedClaim ClaimTypeReferenceId="surname" />

    <!-- Add the loyalty identifier -->
    <PersistedClaim ClaimTypeReferenceId="extension_loyaltyId" />
    <!-- End of changes -->
  </PersistedClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>

```

7. In the **TrustFrameworkBase.xml** file, add the `loyaltyId` claim to **TechnicalProfile AAD-UserReadUsingObjectId** to read the value of the extension attribute every time a user signs in. So far, the **TechnicalProfiles** have been changed in the flow of local accounts only. If you want the new attribute in the flow of a social or federated account, a different set of **TechnicalProfiles** needs to be changed. See the **Next steps** section.

```

<TechnicalProfile Id="AAD-UserReadUsingObjectId">
  <Metadata>
    <Item Key="Operation">Read</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
  </Metadata>
  <IncludeInSso>false</IncludeInSso>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="otherMails" />
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="surname" />

    <!-- Add the loyalty identifier -->
    <OutputClaim ClaimTypeReferenceId="extension_loyaltyId" />
    <!-- End of changes -->
  </OutputClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>

```

Test the custom policy

1. Open the Azure AD B2C blade and navigate to **Identity Experience Framework > Custom policies**.
2. Select the custom policy that you uploaded. Select **Run now**.
3. Sign up by using an email address.

The ID token sent back to your application includes the new extension property as a custom claim preceded by **extension_loyaltyId**. See the following example:

```
{
  "exp": 1493585187,
  "nbf": 1493581587,
  "ver": "1.0",
  "iss": "https://contoso.b2clogin.com/f06c2fe8-709f-4030-85dc-38a4bfd9e82d/v2.0/",
  "sub": "a58e7c6c-7535-4074-93da-b0023fbaf3ac",
  "aud": "4e87c1dd-e5f5-4ac8-8368-bc6a98751b8b",
  "acr": "b2c_1a_trustframeworkprofileedit",
  "nonce": "defaultNonce",
  "iat": 1493581587,
  "auth_time": 1493581587,
  "extension_loyaltyId": "abc",
  "city": "Redmond"
}
```

Next steps

- Add the new claim to the flows to sign in to social accounts by changing the following **TechnicalProfiles**. Social and federated accounts use these two **TechnicalProfiles** to sign in. They write and read user data by using the **alternativeSecurityId** as the locator of the user object.

```
<TechnicalProfile Id="AAD-UserWriteUsingAlternativeSecurityId">

<TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId">
```

- Use the same extension attributes between built-in and custom policies. When you add extension, or custom, attributes via the portal experience, those attributes are registered by using the **b2c-extensions-app** that exists in every B2C tenant. Take the following steps to use extension attributes in your custom policy:
 - Within your B2C tenant in portal.azure.com, navigate to **Azure Active Directory** and select **App registrations**.
 - Find your **b2c-extensions-app** and select it.
 - Under **Essentials**, enter the **Application ID** and the **Object ID**.
 - Include them in your **AAD-Common** TechnicalProfile metadata:

```
<ClaimsProviders>
  <ClaimsProvider>
    <DisplayName>Azure Active Directory</DisplayName>
    <TechnicalProfile Id="AAD-Common">
      <DisplayName>Azure Active Directory</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureActiveDirectoryProvider,
      Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <!-- Provide objectId and appId before using extension properties. -->
      <Metadata>
        <Item Key="ApplicationObjectId">insert objectId here</Item> <!-- This is the "Object ID"
from the "b2c-extensions-app"-->
        <Item Key="ClientId">insert appId here</Item> <!--This is the "Application ID" from the
"b2c-extensions-app"-->
      </Metadata>
    
```

- Stay consistent with the portal experience. Create these attributes by using the portal UI before you use them in your custom policies. When you create an attribute **ActivationStatus** in the portal, you must refer to it as follows:

```
extension_ActivationStatus in the custom policy.
extension_<app-guid>_ActivationStatus via Graph API.
```

Reference

For more information on extension properties, see the article [Add custom data to resources using extensions](#).

NOTE

- A **TechnicalProfile** is an element type, or function, that defines an endpoint's name, metadata, and protocol. The **TechnicalProfile** details the exchange of claims that the Identity Experience Framework performs. When this function is called in an orchestration step or from another **TechnicalProfile**, the **InputClaims** and **OutputClaims** are provided as parameters by the caller.
- Extension attributes in the Graph API are named by using the convention `extension_ApplicationObjectId_attributename`.
- Custom policies refer to extension attributes as **extension_attributename**. This reference omits the **ApplicationObjectId** in XML.
- You have to specify the attribute ID in the following format **extension_attributename** wherever it is being referenced.

Collect Azure Active Directory B2C logs with Application Insights

1/28/2020 • 3 minutes to read • [Edit Online](#)

This article provides steps for collecting logs from Active Directory B2C (Azure AD B2C) so that you can diagnose problems with your custom policies. Application Insights provides a way to diagnose exceptions and visualize application performance issues. Azure AD B2C includes a feature for sending data to Application Insights.

The detailed activity logs described here should be enabled **ONLY** during the development of your custom policies.

WARNING

Do not enable development mode in production. Logs collect all claims sent to and from identity providers. You as the developer assume responsibility for any personal data collected in your Application Insights logs. These detailed logs are collected only when the policy is placed in **DEVELOPER MODE**.

Set up Application Insights

If you don't already have one, create an instance of Application Insights in your subscription.

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure subscription (not your Azure AD B2C directory).
3. Select **Create a resource** in the left-hand navigation menu.
4. Search for and select **Application Insights**, then select **Create**.
5. Complete the form, select **Review + create**, and then select **Create**.
6. Once the deployment has been completed, select **Go to resource**.
7. Under **Configure** in Application Insights menu, select **Properties**.
8. Record the **INSTRUMENTATION KEY** for use in a later step.

Configure the custom policy

1. Open the relying party (RP) file, for example *SignUpOrSignin.xml*.
2. Add the following attributes to the `<TrustFrameworkPolicy>` element:

```
DeploymentMode="Development"  
UserJourneyRecorderEndpoint="urn:journeyrecorder:applicationinsights"
```

3. If it doesn't already exist, add a `<UserJourneyBehaviors>` child node to the `<RelyingParty>` node. It must be located immediately after
`<DefaultUserJourney ReferenceId="UserJourney_Id" from your extensions policy, or equivalent (for example:SignUpOrSignInWithAAD" />`
4. Add the following node as a child of the `<UserJourneyBehaviors>` element. Make sure to replace `{Your Application Insights Key}` with the Application Insights **Instrumentation Key** that you recorded earlier.

```
<JourneyInsights TelemetryEngine="ApplicationInsights" InstrumentationKey="{Your Application Insights Key}" DeveloperMode="true" ClientEnabled="false" ServerEnabled="true" TelemetryVersion="1.0.0" />
```

- `DeveloperMode="true"` tells ApplicationInsights to expedite the telemetry through the processing pipeline. Good for development, but constrained at high volumes.
- `ClientEnabled="true"` sends the ApplicationInsights client-side script for tracking page view and client-side errors. You can view these in the **browserTimings** table in the Application Insights portal. By setting `clientEnabled= "true"`, you add Application Insights to your page script and you get timings of page loads and AJAX calls, counts, details of browser exceptions and AJAX failures, and user and session counts. This field is **optional**, and is set to `false` by default.
- `ServerEnabled="true"` sends the existing UserJourneyRecorder JSON as a custom event to Application Insights.

For example:

```
<TrustFrameworkPolicy>
  ...
  TenantId="fabrikamb2c.onmicrosoft.com"
  PolicyId="SignUpOrSignInWithAAD"
  DeploymentMode="Development"
  UserJourneyRecorderEndpoint="urn:journeyrecorder:applicationinsights"
>
  ...
<RelyingParty>
  <DefaultUserJourney ReferenceId="UserJourney ID from your extensions policy, or equivalent (for example: SignUpOrSignInWithAzureAD)" />
  <UserJourneyBehaviors>
    <JourneyInsights TelemetryEngine="ApplicationInsights" InstrumentationKey="{Your Application Insights Key}" DeveloperMode="true" ClientEnabled="false" ServerEnabled="true" TelemetryVersion="1.0.0" />
  </UserJourneyBehaviors>
  ...
</RelyingParty>
</TrustFrameworkPolicy>
```

5. Upload the policy.

See the logs in Application Insights

There is a short delay, typically less than five minutes, before you can see new logs in Application Insights.

1. Open the Application Insights resource that you created in the [Azure portal](#).
2. In the **Overview** menu, select **Analytics**.
3. Open a new tab in Application Insights.

Here is a list of queries you can use to see the logs:

QUERY	DESCRIPTION
<code>traces</code>	See all of the logs generated by Azure AD B2C
<code>traces where timestamp > ago(1d)</code>	See all of the logs generated by Azure AD B2C for the last day

The entries may be long. Export to CSV for a closer look.

For more information about querying, see [Overview of log queries in Azure Monitor](#).

Next steps

The community has developed a user journey viewer to help identity developers. It reads from your Application Insights instance and provides a well-structured view of the user journey events. You obtain the source code and deploy it in your own solution.

The user journey player is not supported by Microsoft, and is made available strictly as-is.

You can find the version of the viewer that reads events from Application Insights on GitHub, here:

[Azure-Samples/active-directory-b2c-advanced-policies](#)

Troubleshoot Azure AD B2C custom policies and Identity Experience Framework

1/28/2020 • 5 minutes to read • [Edit Online](#)

If you use Azure Active Directory B2C (Azure AD B2C) custom policies, you might experience challenges setting up the Identity Experience Framework in its policy language XML format. Learning to write custom policies can be like learning a new language. In this article, we describe some tools and tips that can help you discover and resolve issues.

This article focuses on troubleshooting your Azure AD B2C custom policy configuration. It doesn't address the relying party application or its identity library.

XML editing

The most common error in setting up custom policies is improperly formatted XML. A good XML editor is nearly essential. It displays XML natively, color-codes content, pre-fills common terms, keeps XML elements indexed, and can validate against an XML schema.

Two of our favorite editors are [Visual Studio Code](#) and [Notepad++](#).

XML schema validation identifies errors before you upload your XML file. In the root folder of the [starter pack](#), get the XML schema definition file *TrustFrameworkPolicy_0.3.0.0.xsd*. To find out how to use the XSD schema file for validation in your editor, look for *XML tools* and *XML validation* or similar in the editor's documentation.

You might find a review of XML rules helpful. Azure AD B2C rejects any XML formatting errors that it detects. Occasionally, incorrectly formatted XML might cause error messages that are misleading.

Upload policies and policy validation

Validation of the XML policy file is performed automatically on upload. Most errors cause the upload to fail. Validation includes the policy file that you are uploading. It also includes the chain of files the upload file refers to (the relying party policy file, the extensions file, and the base file).

Common validation errors include the following:

Error snippet:

```
...makes a reference to ClaimType with id "displayName" but neither the policy nor any of its base policies contain such an element
```

- The ClaimType value might be misspelled, or does not exist in the schema.
- ClaimType values must be defined in at least one of the files in the policy. For example:

```
<ClaimType Id="issuerUserId">
```
- If ClaimType is defined in the extensions file, but it's also used in a TechnicalProfile value in the base file, uploading the base file results in an error.

Error snippet: ...makes a reference to a ClaimsTransformation with id...

- The causes for this error can be the same as for the ClaimType error.

Error snippet:

Reason: User is currently logged as a user of 'yourtenant.onmicrosoft.com' tenant. In order to manage 'yourtenant.onmicrosoft.com', please login as a user of 'yourtenant.onmicrosoft.com' tenant

- Check that the TenantId value in the `<TrustFrameworkPolicy>` and `<BasePolicy>` elements match your target Azure AD B2C tenant.

Troubleshoot the runtime

- Use **Run now** and <https://jwt.ms> to test your policies independently of your web or mobile application. This website acts like a relying party application. It displays the contents of the JSON web token (JWT) that is generated by your Azure AD B2C policy.

To create a test application that can redirect to <https://jwt.ms> for token inspection:

To register an application in your Azure AD B2C tenant, you can use the current **Applications** experience, or our new unified **App registrations (Preview)** experience. [Learn more about the new experience.](#)

- [Applications](#)
- [App registrations \(Preview\)](#)

- Sign in to the [Azure portal](#).
 - Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 - In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 - Select **Applications**, and then select **Add**.
 - Enter a name for the application. For example, *testapp1*.
 - For **Web App / Web API**, select **Yes**.
 - For **Reply URL**, enter <https://jwt.ms>
 - Select **Create**.
- To trace the exchange of messages between your client browser and Azure AD B2C, use [Fiddler](#). It can help you get an indication of where your user journey is failing in your orchestration steps.
 - In **Development mode**, use [Application Insights](#) to trace the activity of your Identity Experience Framework user journey. In **Development mode**, you can observe the exchange of claims between the Identity Experience Framework and the various claims providers that are defined by technical profiles, such as identity providers, API-based services, the Azure AD B2C user directory, and other services, like Azure Multi-Factor Authentication.

Recommended practices

Keep multiple versions of your scenarios. Group them in a project with your application. The base, extensions, and relying party files are directly dependent on each other. Save them as a group. As new features are added to your policies, keep separate working versions. Stage working versions in your own file system with the application code they interact with. Your applications might invoke many different relying party policies in a tenant. They might become dependent on the claims that they expect from your Azure AD B2C policies.

Develop and test technical profiles with known user journeys. Use tested starter pack policies to set up your technical profiles. Test them separately before you incorporate them into your own user journeys.

Develop and test user journeys with tested technical profiles. Change the orchestration steps of a user journey incrementally. Progressively build your intended scenarios.

Next steps

Available on GitHub, download the [active-directory-b2c-custom-policy-starterpack](#).zip archive. You can also clone the repository:

```
git clone https://github.com/Azure-Samples/active-directory-b2c-custom-policy-starterpack
```

Track user behavior in Azure Active Directory B2C using Application Insights

2/11/2020 • 5 minutes to read • [Edit Online](#)

NOTE

This feature is in public preview.

When you use Azure Active Directory B2C (Azure AD B2C) together with Azure Application Insights, you can get detailed and customized event logs for your user journeys. In this article, you learn how to:

- Gain insights on user behavior.
- Troubleshoot your own policies in development or in production.
- Measure performance.
- Create notifications from Application Insights.

How it works

The Identity Experience Framework in Azure AD B2C includes the provider

`Handler="Web.TPEngine.Providers.AzureApplicationInsightsProvider, Web.TPEngine, Version=1.0.0.0"`. It sends event data directly to Application Insights by using the instrumentation key provided to Azure AD B2C.

A technical profile uses this provider to define an event from Azure AD B2C. The profile specifies the name of the event, the claims that are recorded, and the instrumentation key. To post an event, the technical profile is then added as an `orchestration step` in a custom user journey.

Application Insights can unify the events by using a correlation ID to record a user session. Application Insights makes the event and session available within seconds and presents many visualization, export, and analytical tools.

Prerequisites

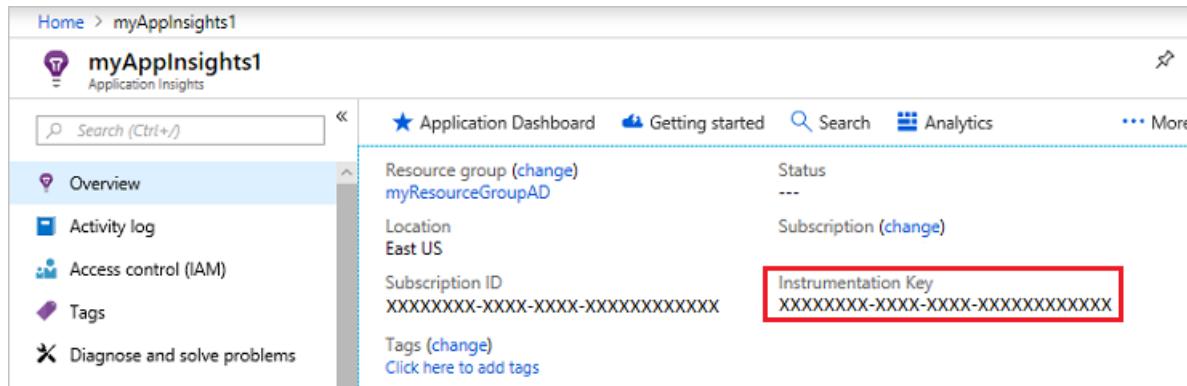
Complete the steps in [Get started with custom policies](#). This article assumes that you're using the custom policy starter pack. But the starter pack isn't required.

Create an Application Insights resource

When you're using Application Insights with Azure AD B2C, all you need to do is create a resource and get the instrumentation key.

1. Sign in to the [Azure portal](#).
2. Make sure you're using the directory that contains your Azure subscription by selecting the **Directory + subscription** filter in the top menu and choosing the directory that contains your subscription. This tenant is not your Azure AD B2C tenant.
3. Choose **Create a resource** in the top-left corner of the Azure portal, and then search for and select **Application Insights**.
4. Click **Create**.
5. Enter a **Name** for the resource.
6. For **Application Type**, select **ASP.NET web application**.

7. For **Resource Group**, select an existing group or enter a name for a new group.
8. Click **Create**.
9. After you create the Application Insights resource, open it, expand **Essentials**, and copy the instrumentation key.



The screenshot shows the 'myApplnights1' Application Insights resource in the Azure portal. The left sidebar has 'Overview' selected. The main content area displays the following details:

- Resource group: myResourceGroupAD
- Status: ---
- Location: East US
- Subscription: (change)
- Subscription ID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
- Instrumentation Key: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX (This field is highlighted with a red box.)
- Tags: (change) Click here to add tags

Add new ClaimType definitions

Open the *TrustFrameworkExtensions.xml* file from the starter pack and add the following elements to the **BuildingBlocks** element:

```

<ClaimsSchema>
  <ClaimType Id="EventType">
    <DisplayName>EventType</DisplayName>
    <DataType>string</DataType>
    <AdminHelpText />
    <UserHelpText />
  </ClaimType>
  <ClaimType Id="PolicyId">
    <DisplayName>PolicyId</DisplayName>
    <DataType>string</DataType>
    <AdminHelpText />
    <UserHelpText />
  </ClaimType>
  <ClaimType Id="Culture">
    <DisplayName>Culture</DisplayName>
    <DataType>string</DataType>
    <AdminHelpText />
    <UserHelpText />
  </ClaimType>
  <ClaimType Id="CorrelationId">
    <DisplayName>CorrelationId</DisplayName>
    <DataType>string</DataType>
    <AdminHelpText />
    <UserHelpText />
  </ClaimType>
  <!--Additional claims used for passing claims to Application Insights Provider -->
  <ClaimType Id="federatedUser">
    <DisplayName>federatedUser</DisplayName>
    <DataType>boolean</DataType>
    <UserHelpText />
  </ClaimType>
  <ClaimType Id="parsedDomain">
    <DisplayName>Parsed Domain</DisplayName>
    <DataType>string</DataType>
    <UserHelpText>The domain portion of the email address.</UserHelpText>
  </ClaimType>
  <ClaimType Id="userInLocalDirectory">
    <DisplayName>userInLocalDirectory</DisplayName>
    <DataType>boolean</DataType>
    <UserHelpText />
  </ClaimType>
</ClaimsSchema>

```

Add new technical profiles

Technical profiles can be considered functions in the Identity Experience Framework of Azure AD B2C. This table defines the technical profiles that are used to open a session and post events.

TECHNICAL PROFILE	TASK
AzureInsights-Common	Creates a common set of parameters to be included in all AzureInsights technical profiles.
AzureInsights-SignInRequest	Creates a SignIn event with a set of claims when a sign-in request has been received.
AzureInsights-UserSignup	Creates a UserSignup event when the user triggers the sign-up option in a sign-up/sign-in journey.
AzureInsights-SignInComplete	Records the successful completion of an authentication when a token has been sent to the relying party application.

Add the profiles to the *TrustFrameworkExtensions.xml* file from the starter pack. Add these elements to the **ClaimsProviders** element:

```
<ClaimsProvider>
  <DisplayName>Application Insights</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="AzureInsights-SignInRequest">
      <InputClaims>
        <!-- An input claim with a PartnerClaimType="eventName" is required. This is used by the AzureApplicationInsightsProvider to create an event with the specified value. -->
        <InputClaim ClaimTypeReferenceId="EventType" PartnerClaimType="eventName" DefaultValue="SignInRequest" />
      </InputClaims>
      <IncludeTechnicalProfile ReferenceId="AzureInsights-Common" />
    </TechnicalProfile>
    <TechnicalProfile Id="AzureInsights-SignInComplete">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="EventType" PartnerClaimType="eventName" DefaultValue="SignInComplete" />
        <InputClaim ClaimTypeReferenceId="federatedUser" PartnerClaimType="{property:FederatedUser}" DefaultValue="false" />
        <InputClaim ClaimTypeReferenceId="parsedDomain" PartnerClaimType="{property:FederationPartner}" DefaultValue="Not Applicable" />
      </InputClaims>
      <IncludeTechnicalProfile ReferenceId="AzureInsights-Common" />
    </TechnicalProfile>
    <TechnicalProfile Id="AzureInsights-UserSignup">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="EventType" PartnerClaimType="eventName" DefaultValue="UserSignup" />
      </InputClaims>
      <IncludeTechnicalProfile ReferenceId="AzureInsights-Common" />
    </TechnicalProfile>
    <TechnicalProfile Id="AzureInsights-Common">
      <DisplayName>Alternate Email</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.Insights.AzureApplicationInsightsProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <!-- The ApplicationInsights instrumentation key which will be used for logging the events -->
        <Item Key="InstrumentationKey">xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</Item>
        <!-- A Boolean that indicates whether developer mode is enabled. This controls how events are buffered. In a development environment with minimal event volume, enabling developer mode results in events being sent immediately to ApplicationInsights. -->
        <Item Key="DeveloperMode">false</Item>
        <!-- A Boolean that indicates whether telemetry should be enabled or not. -->
        <Item Key="DisableTelemetry ">false</Item>
      </Metadata>
      <InputClaims>
        <!-- Properties of an event are added through the syntax {property:NAME}, where NAME is property being added to the event. DefaultValue can be either a static value or a value that's resolved by one of the supported DefaultClaimResolvers. -->
        <InputClaim ClaimTypeReferenceId="PolicyId" PartnerClaimType="{property:Policy}" DefaultValue="{Policy:PolicyId}" />
        <InputClaim ClaimTypeReferenceId="CorrelationId" PartnerClaimType="{property:JourneyId}" DefaultValue="{Context:CorrelationId}" />
        <InputClaim ClaimTypeReferenceId="Culture" PartnerClaimType="{property:Culture}" DefaultValue="{Culture:RFC5646}" />
      </InputClaims>
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

IMPORTANT

Change the instrumentation key in the `AzureInsights-Common` technical profile to the GUID that your Application Insights resource provides.

Add the technical profiles as orchestration steps

Call `Azure-Insights-SignInRequest` as orchestration step 2 to track that a sign-in/sign-up request has been received:

```
<!-- Track that we have received a sign in request -->
<OrchestrationStep Order="1" Type="ClaimsExchange">
  <ClaimsExchanges>
    <ClaimsExchange Id="TrackSignInRequest" TechnicalProfileReferenceId="AzureInsights-SignInRequest" />
  </ClaimsExchanges>
</OrchestrationStep>
```

Immediately *before* the `SendClaims` orchestration step, add a new step that calls `Azure-Insights-UserSignup`. It's triggered when the user selects the sign-up button in a sign-up/sign-in journey.

```
<!-- Handles the user clicking the sign up link in the local account sign in page -->
<OrchestrationStep Order="8" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
      <Value>newUser</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
    <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
      <Value>newUser</Value>
      <Value>false</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="TrackUserSignUp" TechnicalProfileReferenceId="AzureInsights-UserSignup" />
  </ClaimsExchanges>
</OrchestrationStep>
```

Immediately *after* the `SendClaims` orchestration step, call `Azure-Insights-SignInComplete`. This step shows a successfully completed journey.

```
<!-- Track that we have successfully sent a token -->
<OrchestrationStep Order="10" Type="ClaimsExchange">
  <ClaimsExchanges>
    <ClaimsExchange Id="TrackSignInComplete" TechnicalProfileReferenceId="AzureInsights-SignInComplete" />
  </ClaimsExchanges>
</OrchestrationStep>
```

IMPORTANT

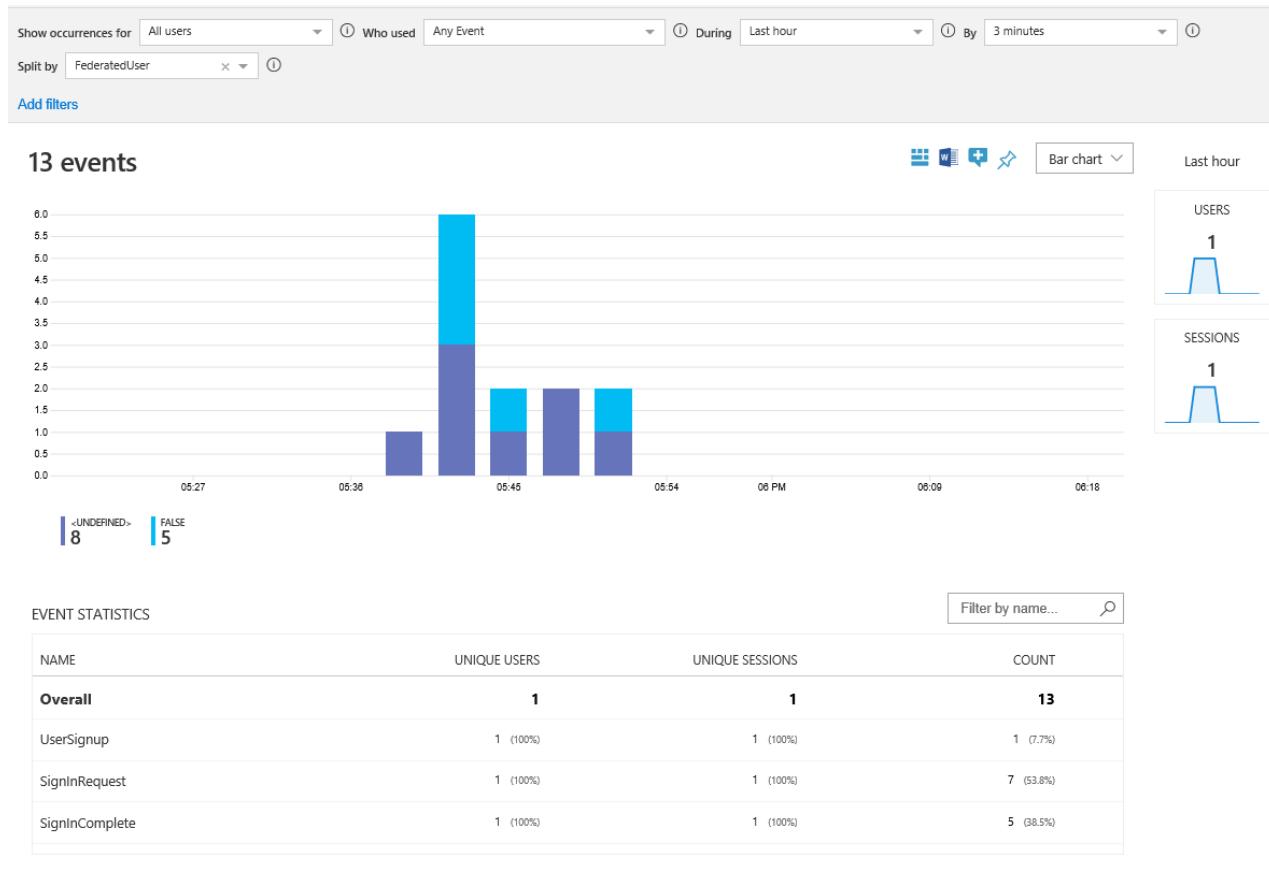
After you add the new orchestration steps, renumber the steps sequentially without skipping any integers from 1 to N.

Upload your file, run the policy, and view events

Save and upload the `TrustFrameworkExtensions.xml` file. Then, call the relying party policy from your application

or use **Run Now** in the Azure portal. In seconds, your events are available in Application Insights.

1. Open the **Application Insights** resource in your Azure Active Directory tenant.
2. Select **Usage > Events**.
3. Set **During** to **Last hour** and **By** to **3 minutes**. You might need to select **Refresh** to view results.



Next steps

Add claim types and events to your user journey to fit your needs. You can use [claim resolvers](#) or any string claim type, add the claims by adding an **Input Claim** element to the Application Insights event or to the AzureInsights-Common technical profile.

- **ClaimTypeReferenceId** is the reference to a claim type.
- **PartnerClaimType** is the name of the property that appears in Azure Insights. Use the syntax of `{property:NAME}`, where `NAME` is property being added to the event.
- **DefaultValue** use any string value or the claim resolver.

```
<InputClaim ClaimTypeReferenceId="app_session" PartnerClaimType="{property:app_session}" DefaultValue="{OAUTH-KV:app_session}" />
<InputClaim ClaimTypeReferenceId="loyalty_number" PartnerClaimType="{property:loyalty_number}" DefaultValue="{OAUTH-KV:loyalty_number}" />
<InputClaim ClaimTypeReferenceId="language" PartnerClaimType="{property:language}" DefaultValue="{Culture:RFC5646}" />
```

Define Trust Frameworks with Azure AD B2C Identity Experience Framework

1/28/2020 • 8 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) custom policies that use the Identity Experience Framework provide your organization with a centralized service. This service reduces the complexity of identity federation in a large community of interest. The complexity is reduced to a single trust relationship and a single metadata exchange.

Azure AD B2C custom policies use the Identity Experience Framework to enable you to answer the following questions:

- What are the legal, security, privacy, and data protection policies that must be adhered to?
- Who are the contacts and what are the processes for becoming an accredited participant?
- Who are the accredited identity information providers (also known as "claims providers") and what do they offer?
- Who are the accredited relying parties (and optionally, what do they require)?
- What are the technical "on the wire" interoperability requirements for participants?
- What are the operational "runtime" rules that must be enforced for exchanging digital identity information?

To answer all these questions, Azure AD B2C custom policies that use the Identity Experience Framework use the Trust Framework (TF) construct. Let's consider this construct and what it provides.

Understand the Trust Framework and federation management foundation

The Trust Framework is a written specification of the identity, security, privacy, and data protection policies to which participants in a community of interest must conform.

Federated identity provides a basis for achieving end-user identity assurance at Internet scale. By delegating identity management to third parties, a single digital identity for an end user can be reused with multiple relying parties.

Identity assurance requires that identity providers (IdPs) and attribute providers (AtPs) adhere to specific security, privacy, and operational policies and practices. If they can't perform direct inspections, relying parties (RPs) must develop trust relationships with the IdPs and AtPs they choose to work with.

As the number of consumers and providers of digital identity information grows, it's difficult to continue pairwise management of these trust relationships, or even the pairwise exchange of the technical metadata that's required for network connectivity. Federation hubs have achieved only limited success at solving these problems.

What a Trust Framework specification defines

TFs are the linchpins of the Open Identity Exchange (OIX) Trust Framework model, where each community of interest is governed by a particular TF specification. Such a TF specification defines:

- **The security and privacy metrics for the community of interest with the definition of:**
 - The levels of assurance (LOA) that are offered/required by participants; for example, an ordered set of confidence ratings for the authenticity of digital identity information.
 - The levels of protection (LOP) that are offered/required by participants; for example, an ordered set of confidence ratings for the protection of digital identity information that's handled by participants in the

community of interest.

- **The description of the digital identity information that's offered/required by participants.**
- **The technical policies for production and consumption of digital identity information, and thus for measuring LOA and LOP. These written policies typically include the following categories of policies:**
 - Identity proofing policies, for example: *How strongly is a person's identity information vetted?*
 - Security policies, for example: *How strongly are information integrity and confidentiality protected?*
 - Privacy policies, for example: *What control does a user have over personal identifiable information (PII)?*
 - Survivability policies, for example: *If a provider ceases operations, how does continuity and protection of PII function?*
- **The technical profiles for production and consumption of digital identity information. These profiles include:**
 - Scope interfaces for which digital identity information is available at a specified LOA.
 - Technical requirements for on-the-wire interoperability.
- **The descriptions of the various roles that participants in the community can perform and the qualifications that are required to fulfill these roles.**

Thus a TF specification governs how identity information is exchanged between the participants of the community of interest: relying parties, identity and attribute providers, and attribute verifiers.

A TF specification is one or multiple documents that serve as a reference for the governance of the community of interest that regulates the assertion and consumption of digital identity information within the community. It's a documented set of policies and procedures designed to establish trust in the digital identities that are used for online transactions between members of a community of interest.

In other words, a TF specification defines the rules for creating a viable federated identity ecosystem for a community.

Currently there's widespread agreement on the benefit of such an approach. There's no doubt that trust framework specifications facilitate the development of digital identity ecosystems with verifiable security, assurance and privacy characteristics, meaning that they can be reused across multiple communities of interest.

For that reason, Azure AD B2C custom policies that use the Identity Experience Framework uses the specification as the basis of its data representation for a TF to facilitate interoperability.

Azure AD B2C Custom policies that leverage the Identity Experience Framework represent a TF specification as a mixture of human and machine-readable data. Some sections of this model (typically sections that are more oriented toward governance) are represented as references to published security and privacy policy documentation along with the related procedures (if any). Other sections describe in detail the configuration metadata and runtime rules that facilitate operational automation.

Understand Trust Framework policies

In terms of implementation, the TF specification consists of a set of policies that allow complete control over identity behaviors and experiences. Azure AD B2C custom policies that leverage the Identity Experience Framework enable you to author and create your own TF through such declarative policies that can define and configure:

- The document reference or references that define the federated identity ecosystem of the community that relates to the TF. They are links to the TF documentation. The (predefined) operational "runtime" rules, or the user journeys that automate and/or control the exchange and usage of the claims. These user journeys are associated with a LOA (and a LOP). A policy can therefore have user journeys with varying LOAs (and LOPs).

- The identity and attribute providers, or the claims providers, in the community of interest and the technical profiles they support along with the (out-of-band) LOA/LOP accreditation that relates to them.
- The integration with attribute verifiers or claims providers.
- The relying parties in the community (by inference).
- The metadata for establishing network communications between participants. This metadata, along with the technical profiles, are used during a transaction to plumb "on the wire" interoperability between the relying party and other community participants.
- The protocol conversion if any (for example, SAML 2.0, OAuth2, WS-Federation, and OpenID Connect).
- The authentication requirements.
- The multifactor orchestration if any.
- A shared schema for all the claims that are available and mappings to participants of a community of interest.
- All the claims transformations, along with the possible data minimization in this context, to sustain the exchange and usage of the claims.
- The binding and encryption.
- The claims storage.

Understand claims

NOTE

We collectively refer to all the possible types of identity information that might be exchanged as "claims": claims about an end user's authentication credential, identity vetting, communication device, physical location, personally identifying attributes, and so on.

We use the term "claims"--rather than "attributes"--because in online transactions, these data artifacts are not facts that can be directly verified by the relying party. Rather they're assertions, or claims, about facts for which the relying party must develop sufficient confidence to grant the end user's requested transaction.

We also use the term "claims" because Azure AD B2C custom policies that use the Identity Experience Framework are designed to simplify the exchange of all types of digital identity information in a consistent manner regardless of whether the underlying protocol is defined for user authentication or attribute retrieval. Likewise, we use the term "claims providers" to collectively refer to identity providers, attribute providers, and attribute verifiers when we do not want to distinguish between their specific functions.

Thus they govern how identity information is exchanged between a relying party, identity and attribute providers, and attribute verifiers. They control which identity and attribute providers are required for a relying party's authentication. They should be considered as a domain-specific language (DSL), that is, a computer language that's specialized for a particular application domain with inheritance, *if* statements, polymorphism.

These policies constitute the machine-readable portion of the TF construct in Azure AD B2C Custom policies leveraging the Identity Experience Framework. They include all the operational details, including claims providers' metadata and technical profiles, claims schema definitions, claims transformation functions, and user journeys that are filled in to facilitate operational orchestration and automation.

They are assumed to be *living documents* because there is a good chance that their contents will change over time concerning the active participants declared in the policies. There is also the potential that the terms and conditions for being a participant might change.

Federation setup and maintenance are vastly simplified by shielding relying parties from ongoing trust and

connectivity reconfigurations as different claims providers/verifiers join or leave (the community represented by) the set of policies.

Interoperability is another significant challenge. Additional claims providers/verifiers must be integrated, because relying parties are unlikely to support all the necessary protocols. Azure AD B2C custom policies solve this problem by supporting industry-standard protocols and by applying specific user journeys to transpose requests when relying parties and attribute providers do not support the same protocol.

User journeys include protocol profiles and metadata that are used to plumb "on the wire" interoperability between the relying party and other participants. There are also operational runtime rules that are applied to identity information exchange request/response messages for enforcing compliance with published policies as part of the TF specification. The idea of user journeys is key to the customization of the customer experience. It also sheds light on how the system works at the protocol level.

On that basis, relying party applications and portals can, depending on their context, invoke Azure AD B2C custom policies that leverage the Identity Experience Framework passing the name of a specific policy and get precisely the behavior and information exchange they want without any muss, fuss, or risk.

TrustFrameworkPolicy

2/3/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A custom policy is represented as one or more XML-formatted files, which refer to each other in a hierarchical chain. The XML elements define elements of the policy, such as the claims schema, claims transformations, content definitions, claims providers, technical profiles, user journey, and orchestration steps. Each policy file is defined within the top-level **TrustFrameworkPolicy** element of a policy file.

```
<TrustFrameworkPolicy
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="https://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
  PolicySchemaVersion="0.3.0.0"
  TenantId="mytenant.onmicrosoft.com"
  PolicyId="B2C_1A_TrustFrameworkBase"
  PublicPolicyUri="http://mytenant.onmicrosoft.com/B2C_1A_TrustFrameworkBase">
  ...
</TrustFrameworkPolicy>
```

The **TrustFrameworkPolicy** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
PolicySchemaVersion	Yes	The schema version that is to be used to execute the policy. The value must be 0.3.0.0
TenantObjectId	No	The unique object identifier of the Azure Active Directory B2C (Azure AD B2C) tenant.
TenantId	Yes	The unique identifier of the tenant to which this policy belongs.
PolicyId	Yes	The unique identifier for the policy. This identifier must be prefixed by B2C_1A_
PublicPolicyUri	Yes	The URI for the policy, which is combination of the tenant ID and the policy ID.
DeploymentMode	No	Possible values: Production , or Development . Production is the default. Use this property to debug your policy. For more information, see Collecting Logs .

ATTRIBUTE	REQUIRED	DESCRIPTION
UserJourneyRecorderEndpoint	No	The endpoint that is used when DeploymentMode is set to Development . The value must be urn:journeyrecorder:applicationinsights . For more information, see Collecting Logs .

The following example shows how to specify the **TrustFrameworkPolicy** element:

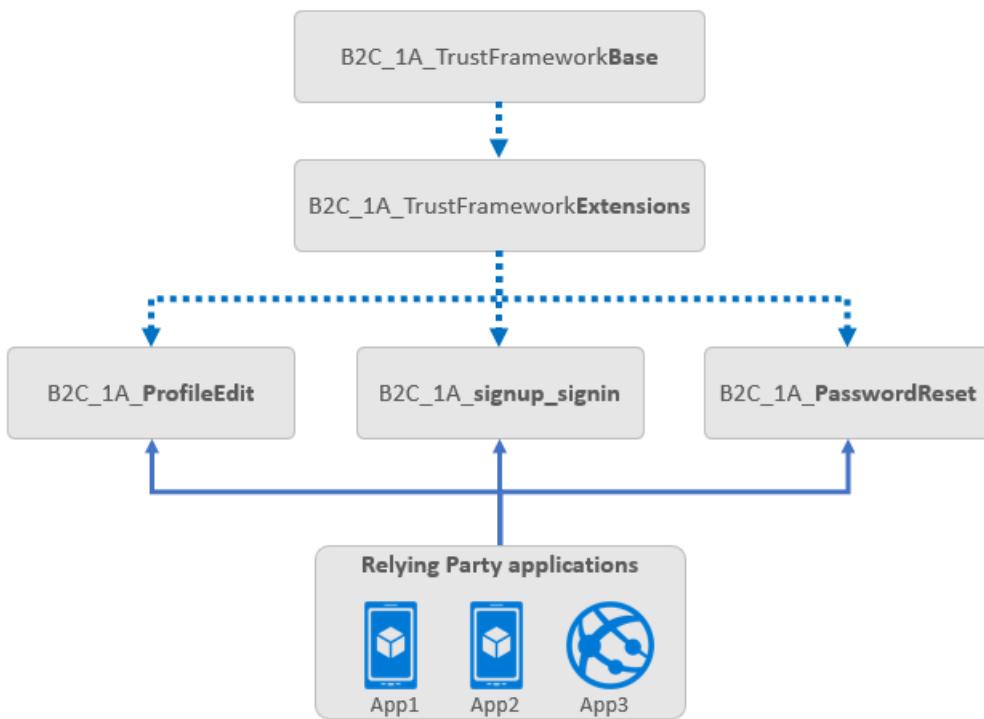
```
<TrustFrameworkPolicy
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="https://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
  PolicySchemaVersion="0.3.0.0"
  TenantId="mytenant.onmicrosoft.com"
  PolicyId="B2C_1A_TrustFrameworkBase"
  PublicPolicyUri="http://mytenant.onmicrosoft.com/B2C_1A_TrustFrameworkBase">
```

Inheritance model

These types of policy files are typically used in a user journey:

- A **Base** file that contains most of the definitions. To help with troubleshooting and long-term maintenance of your policies, it is recommended that you make a minimum number of changes to this file.
- An **Extensions** file that holds the unique configuration changes for your tenant. This policy file is derived from the Base file. Use this file to add new functionality or override existing functionality. For example, use this file to federate with new identity providers.
- A **Relying Party (RP)** file that is the single task-focused file that is invoked directly by the relying party application, such as your web, mobile, or desktop applications. Each unique task such as sign-up or sign-in, password reset, or profile edit, requires its own RP policy file. This policy file is derived from the Extensions file.

A relying party application calls the RP policy file to execute a specific task. For example, to initiate the sign-in flow. The Identity Experience Framework in Azure AD B2C adds all of the elements first from the Base file, and then from the Extensions file, and finally from the RP policy file to assemble the current policy in effect. Elements of the same type and name in the RP file override those elements in the Extensions, and Extensions overrides Base. The following diagram shows the relationship between the policy files and the relying party applications.



The inheritance model is as follows:

- The parent policy and child policy are of the same schema.
- The child policy at any level can inherit from the parent policy and extend it by adding new elements.
- There is no limit on the number of levels.

For more information, see [Get started with custom policies](#).

Base policy

To inherit a policy from another policy, a **BasePolicy** element must be declared under the **TrustFrameworkPolicy** element of the policy file. The **BasePolicy** element is a reference to the base policy from which this policy is derived.

The **BasePolicy** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
TenantId	1:1	The identifier of your Azure AD B2C tenant.
PolicyId	1:1	The identifier of the parent policy.

The following example shows how to specify a base policy. This **B2C_1A_TrustFrameworkExtensions** policy is derived from the **B2C_1A_TrustFrameworkBase** policy.

```

<TrustFrameworkPolicy
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="https://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
    PolicySchemaVersion="0.3.0.0"
    TenantId="mytenant.onmicrosoft.com"
    PolicyId="B2C_1A_TrustFrameworkExtensions"
    PublicPolicyUri="http://mytenant.onmicrosoft.com/B2C_1A_TrustFrameworkExtensions">

    <BasePolicy>
        <TenantId>yourtenant.onmicrosoft.com</TenantId>
        <PolicyId>B2C_1A_TrustFrameworkBase</PolicyId>
    </BasePolicy>
    ...
</TrustFrameworkPolicy>

```

Policy execution

A relying party application, such as a web, mobile, or desktop application, calls the [relying party \(RP\) policy](#). The RP policy file executes a specific task, such as signing in, resetting a password, or editing a profile. The RP policy configures the list of claims the relying party application receives as part of the token that is issued. Multiple applications can use the same policy. All applications receive the same token with claims, and the user goes through the same user journey. A single application can use multiple policies.

Inside the RP policy file, you specify the **DefaultUserJourney** element, which points to the [UserJourney](#). The user journey usually is defined in the Base or Extensions policy.

B2C_1A_signup_signin policy:

```

<RelyingParty>
    <DefaultUserJourney ReferenceId="SignUpOrSignIn">
        ...

```

B2C_1A_TrustFrameWorkBase or B2C_1A_TrustFrameworkExtensionPolicy:

```

<UserJourneys>
    <UserJourney Id="SignUpOrSignIn">
        ...

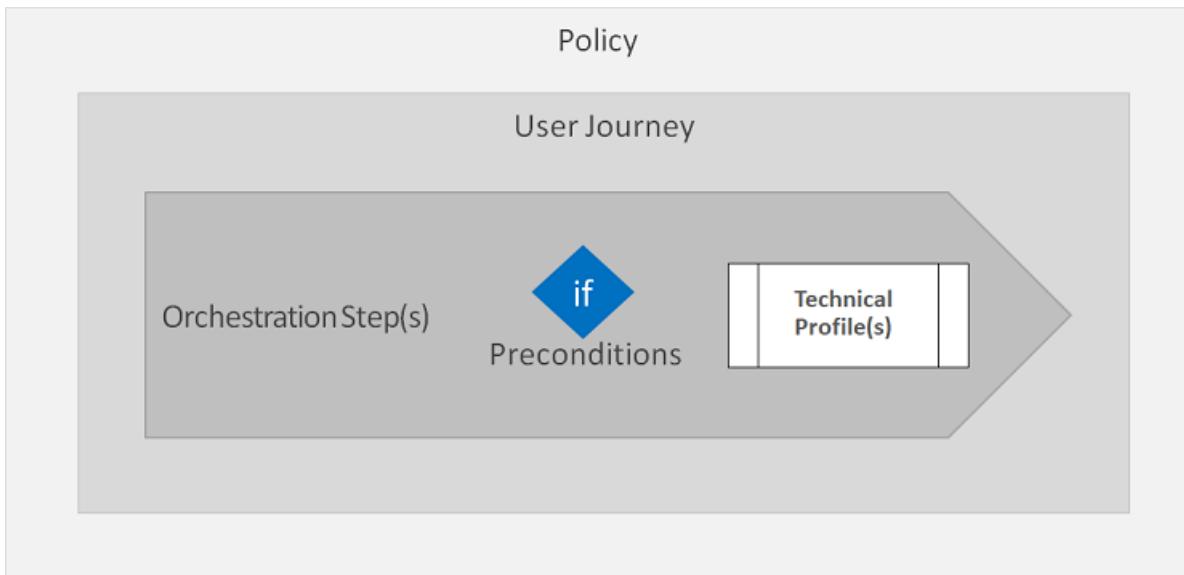
```

A user journey defines the business logic of what a user goes through. Each user journey is a set of orchestration steps that performs a series of actions, in sequence in terms of authentication and information collection.

The **SocialAndLocalAccounts** policy file in the [starter pack](#) contains the SignUpOrSignIn, ProfileEdit, PasswordReset user journeys. You can add more user journeys for other scenarios, such as changing an email address or linking and unlinking a social account.

The orchestration steps may call a [Technical Profile](#). A technical profile provides a framework with a built-in mechanism to communicate with different types of parties. For example, a technical profile can perform these actions among others:

- Render a user experience.
- Allow users to sign in with social or an enterprise account, such as Facebook, Microsoft account, Google, Salesforce or any other identity provider.
- Set up phone verification for MFA.
- Read and write data to and from an Azure AD B2C identity store.
- Call a custom Restful API service.



The **TrustFrameworkPolicy** element contains the following elements:

- BasePolicy as specified above
- [BuildingBlocks](#)
- [ClaimsProviders](#)
- [UserJourneys](#)
- [RelyingParty](#)

BuildingBlocks

12/12/2019 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **BuildingBlocks** element is added inside the [TrustFrameworkPolicy](#) element.

```
<TrustFrameworkPolicy
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="https://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
  PolicySchemaVersion="0.3.0.0"
  TenantId="mytenant.onmicrosoft.com"
  PolicyId="B2C_1A_TrustFrameworkBase"
  PublicPolicyUri="http://mytenant.onmicrosoft.com/B2C_1A_TrustFrameworkBase">

  <BuildingBlocks>
    <ClaimsSchema>
      ...
    </ClaimsSchema>
    <Predicates>
      ...
    </Predicates>
    <PredicateValidations>
      ...
    </PredicateValidations>
    <ClaimsTransformations>
      ...
    </ClaimsTransformations>
    <ContentDefinitions>
      ...
    </ContentDefinitions>
    <Localization>
      ...
    </Localization>
    <DisplayControls>
      ...
    </DisplayControls>
  </BuildingBlocks>
```

The **BuildingBlocks** element contains the following elements that must be specified in the order defined:

- [ClaimsSchema](#) - 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.
- [Predicates and PredicateValidationsInput](#) - Enables you to perform a validation process to ensure that only properly formed data is entered into a claim.
- [ClaimsTransformations](#) - Contains a list of claims transformations that can be used in your policy. A claims transformation converts one claim into another. In the claims transformation, you specify a transform

method, such as:

- Changing the case of a string claim to the one specified. For example, changing a string from lowercase to uppercase.
 - Comparing two claims and returning a claim with true indicating that the claims match, otherwise false.
 - Creating a string claim from the provided parameter in the policy.
 - Creating a random string using the random number generator.
 - Formatting a claim according to the provided format string. This transformation uses the C# `String.Format` method.
- **InputValidation** - This element allows you to perform boolean aggregations that are similar to *and* and *or*.
 - **ContentDefinitions** - Contains URLs for HTML5 templates to use in your user journey. In a custom policy, a content definition defines the HTML5 page URI that's used for a specified step in the user journey. For example, the sign-in or sign-up, password reset, or error pages. You can modify the look and feel by overriding the LoadUri for the HTML5 file. Or you can create new content definitions according to your needs. This element may contain a localized resources reference using a localization ID.
 - **Localization** - Allows you to support multiple languages. The localization support in policies allows you set up the list of supported languages in a policy and pick a default language. Language-specific strings and collections are also supported.
 - **DisplayControls** - Defines the controls to be displayed on a page. Display controls have special functionality and interact with back-end validation technical profiles. Display controls are currently in **preview**.

ClaimsSchema

2/24/2020 • 9 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **ClaimsSchema** element defines the claim types that can be referenced as part of the policy. Claims schema is the place where you declare your claims. A claim can be first name, last name, display name, phone number and more. ClaimsSchema element contains list of **ClaimType** elements. The **ClaimType** element contains the **Id** attribute, which is the claim name.

```
<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="Id">
      <DisplayName>Surname</DisplayName>
      <DataType>string</DataType>
      <DefaultPartnerClaimTypes>
        <Protocol Name="OAuth2" PartnerClaimType="family_name" />
        <Protocol Name="OpenIdConnect" PartnerClaimType="family_name" />
        <Protocol Name="SAML2" PartnerClaimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" />
      </DefaultPartnerClaimTypes>
      <UserHelpText>Your surname (also known as family name or last name).</UserHelpText>
      <UserInputType>TextBox</UserInputType>
```

ClaimType

The **ClaimType** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the claim type. Other elements can use this identifier in the policy.

The **ClaimType** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
DisplayName	1:1	The title that's displayed to users on various screens. The value can be localized .
DataType	1:1	The type of the claim.

ELEMENT	OCCURRENCES	DESCRIPTION
DefaultPartnerClaimTypes	0:1	The partner default claim types to use for a specified protocol. The value can be overwritten in the PartnerClaimType specified in the InputClaim or OutputClaim elements. Use this element to specify the default name for a protocol.
Mask	0:1	An optional string of masking characters that can be applied when displaying the claim. For example, the phone number 324-232-4343 can be masked as XXX-XXX-4343.
UserHelpText	0:1	A description of the claim type that can be helpful for users to understand its purpose. The value can be localized .
UserInputType	0:1	The type of input control that should be available to the user when manually entering the claim data for the claim type. See the user input types defined later in this page.
Restriction	0:1	The value restrictions for this claim, such as a regular expression (Regex) or a list of acceptable values. The value can be localized .
PredicateValidationReference	0:1	A reference to a PredicateValidationsInput element. The PredicateValidationReference elements enable you to perform a validation process to ensure that only properly formed data is entered. For more information, see Predicates .

Data Type

The **Data Type** element supports the following values:

TYPE	DESCRIPTION
boolean	Represents a Boolean (<code>true</code> or <code>false</code>) value.
date	Represents an instant in time, typically expressed as a date of a day. The value of the date follows ISO 8601 convention.
dateTime	Represents an instant in time, typically expressed as a date and time of day. The value of the date follows ISO 8601 convention.

TYPE	DESCRIPTION
duration	Represents a time interval in years, months, days, hours, minutes, and seconds. The format of is <code>PnYnMnDTnHnMnS</code> , where <code>P</code> indicates positive, or <code>N</code> for negative value. <code>nY</code> is the number of years followed by a literal <code>Y</code> . <code>nMo</code> is the number of months followed by a literal <code>Mo</code> . <code>nD</code> is the number of days followed by a literal <code>D</code> . Examples: <code>P21Y</code> represents 21 years. <code>P1Y2Mo</code> represents one year, and two months. <code>P1Y2Mo5D</code> represents one year, two months, and five days. <code>P1Y2M5DT8H5M620S</code> represents one year, two months, five days, eight hours, five minutes, and twenty seconds.
phoneNumber	Represents a phone number.
int	Represents number between -2,147,483,648 and 2,147,483,647
long	Represents number between -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
string	Represents text as a sequence of UTF-16 code units.
stringCollection	Represents a collection of <code>string</code> .
userIdentity	Represents a user identity.
userIdentityCollection	Represents a collection of <code>userIdentity</code> .

DefaultPartnerClaimTypes

The **DefaultPartnerClaimTypes** may contain the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
Protocol	1:n	List of protocols with their default partner claim type name.

The **Protocol** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Name	Yes	The name of a valid protocol supported by Azure AD B2C. Possible values are: OAuth1, OAuth2, SAML2, OpenIdConnect.
PartnerClaimType	Yes	The claim type name to be used.

In the following example, when the Identity Experience Framework interacts with a SAML2 identity provider or relying party application, the **surname** claim is mapped to

`http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname`, with OpenIdConnect and OAuth2, the claim is mapped to `family_name`.

```

<ClaimType Id="surname">
  <DisplayName>Surname</DisplayName>
  <DataType>string</DataType>
  <DefaultPartnerClaimTypes>
    <Protocol Name="OAuth2" PartnerClaimType="family_name" />
    <Protocol Name="OpenIdConnect" PartnerClaimType="family_name" />
    <Protocol Name="SAML2" PartnerClaimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname" />
  </DefaultPartnerClaimTypes>
</ClaimType>

```

As a result, the JWT token issued by Azure AD B2C, emits the `family_name` instead of ClaimType name **surname**.

```
{
  "sub": "6fbbd70d-262b-4b50-804c-257ae1706ef2",
  "auth_time": 1535013501,
  "given_name": "David",
  "family_name": "Williams",
  "name": "David Williams",
}
```

Mask

The **Mask** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
<code>Type</code>	Yes	The type of the claim mask. Possible values: <code>Simple</code> or <code>Regex</code> . The <code>Simple</code> value indicates that a simple text mask is applied to the leading portion of a string claim. The <code>Regex</code> value indicates that a regular expression is applied to the string claim as whole. If the <code>Regex</code> value is specified, an optional attribute must also be defined with the regular expression to use.
<code>Regex</code>	No	If <code>Type</code> is set to <code>Regex</code> , specify the regular expression to use.

The following example configures a **PhoneNumber** claim with the `Simple` mask:

```

<ClaimType Id="PhoneNumber">
  <DisplayName>Phone Number</DisplayName>
  <DataType>string</DataType>
  <Mask Type="Simple">XXX-XXX-</Mask>
  <UserHelpText>Your telephone number.</UserHelpText>
</ClaimType>

```

The Identity Experience Framework renders the phone number while hiding the first six digits:

We have the following number on record for you. We can send a code via SMS or phone to authenticate you.

Phone Number - XXX-XXX-456120

[Send Code](#) [Call Me](#) [Cancel](#)

The following example configures a **AlternateEmail** claim with the `Regex` mask:

```
<ClaimType Id="AlternateEmail">
  <DisplayName>Please verify the secondary email linked to your account</DisplayName>
  <DataType>string</DataType>
  <Mask Type="Regex" Regex="(?&lt;=.) .(?=.*)@)"></Mask>
  <UserInputType>Readonly</UserInputType>
</ClaimType>
```

The Identity Experience Framework renders only the first letter of the email address and the email domain name:

Please verify the secondary email linked to your account

s*****@contoso.com

Continue Cancel

Restriction

The **Restriction** element may contain the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
MergeBehavior	No	The method used to merge enumeration values with a ClaimType in a parent policy with the same identifier. Use this attribute when you overwrite a claim specified in the base policy. Possible values: <code>Append</code> , <code>Prepend</code> , or <code>ReplaceAll</code> . The <code>Append</code> value is a collection of data that should be appended to the end of the collection specified in the parent policy. The <code>Prepend</code> value is a collection of data that should be added before the collection specified in the parent policy. The <code>ReplaceAll</code> value is a collection of data specified in the parent policy that should be ignored.

The **Restriction** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Enumeration	1:n	The available options in the user interface for the user to select for a claim, such as a value in a dropdown.
Pattern	1:1	The regular expression to use.

Enumeration

The **Enumeration** element defines available options for the user to select for a claim in the user interface, such as a value in a `CheckboxMultiSelect`, `DropdownSingleSelect`, or `RadioSingleSelect`. Alternatively, you can define and localize available options with `LocalizedCollections` element. To look up an item from a claim **Enumeration** collection, use `GetMappedValueFromLocalizedCollection` claims transformation.

The **Enumeration** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Text	Yes	The display string that is shown to the user in the user interface for this option.
Value	Yes	The claim value that is associated with selecting this option.
SelectByDefault	No	Indicates whether or not this option should be selected by default in the UI. Possible values: True or False.

The following example configures a **city** dropdown list claim with a default value set to `New York`:

```
<ClaimType Id="city">
  <DisplayName>city where you work</DisplayName>
  <DataType>string</DataType>
  <UserInputType>DropdownSingleSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Bellevue" Value="bellevue" SelectByDefault="false" />
    <Enumeration Text="Redmond" Value="redmond" SelectByDefault="false" />
    <Enumeration Text="New York" Value="new-york" SelectByDefault="true" />
  </Restriction>
</ClaimType>
```

Dropdown city list with a default value set to New York:

A screenshot of a dropdown menu. The title is "City where you work". The options are: New York (selected), Bellevue, Redmond, and New York (highlighted with a blue background).

Pattern

The **Pattern** element can contain the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
RegularExpression	Yes	The regular expression that claims of this type must match in order to be valid.
HelpText	No	An error message for users if the regular expression check fails.

The following example configures an **email** claim with regular expression input validation and help text:

```

<ClaimType Id="email">
  <DisplayName>Email Address</DisplayName>
  <DataType>string</DataType>
  <DefaultPartnerClaimTypes>
    <Protocol Name="OpenIdConnect" PartnerClaimType="email" />
  </DefaultPartnerClaimTypes>
  <UserHelpText>Email address that can be used to contact you.</UserHelpText>
  <UserInputType>TextBox</UserInputType>
  <Restriction>
    <Pattern RegularExpression="^a-zA-Z0-9.!#$%&'^_`{}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$"
    HelpText="Please enter a valid email address." />
  </Restriction>
</ClaimType>

```

The Identity Experience Framework renders the email address claim with email format input validation:

Verification is necessary. Please click Send button.

Email Address

Please enter a valid email address.

Send verification code

UserInputType

Azure AD B2C supports a variety of user input types, such as a textbox, password, and dropdown list that can be used when manually entering claim data for the claim type. You must specify the **UserInputType** when you collect information from the user by using a [self-asserted technical profile](#) and [display controls](#).

The **UserInputType** element available user input types:

USERINPUTTYPE	SUPPORTED CLAIMTYPE	DESCRIPTION
CheckboxMultiSelect	string	Multi select drop-down box. The claim value is represented in a comma delimiter string of the selected values.
DateTimeDropdown	date , dateTime	Drop-downs to select a day, month, and year.
DropdownSingleSelect	string	Single select drop-down box. The claim value is the selected value.
EmailBox	string	Email input field.
Paragraph	boolean , date , dateTime , duration , int , long , string	A field that shows text only in a paragraph tag.
Password	string	Password text box.
RadioSingleSelect	string	Collection of radio buttons. The claim value is the selected value.
Readonly	boolean , date , dateTime , duration , int , long , string	Read-only text box.
TextBox	boolean , int , string	Single-line text box.

TextBox

The **TextBox** user input type is used to provide a single-line text box.

Display Name
Emily Smith

```
<ClaimType Id="displayName">
  <DisplayName>Display Name</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Your display name.</UserHelpText>
  <UserInputType>TextBox</UserInputType>
</ClaimType>
```

EmailBox

The **EmailBox** user input type is used to provide a basic email input field.

Email Address
someone@contoso.com

```
<ClaimType Id="email">
  <DisplayName>Email Address</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Email address that can be used to contact you.</UserHelpText>
  <UserInputType>EmailBox</UserInputType>
  <Restriction>
    <Pattern RegularExpression="^([a-zA-Z0-9!#$%&;'+'`{}~-]+)(?:\.( [a-zA-Z0-9!#$%&;'+'`{}~-]+)+)*(@([a-zA-Z0-9](:[a-zA-Z0-9-]*[a-zA-Z0-9])?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-Z0-9])? $" HelpText="Please enter a valid email address." />
  </Restriction>
</ClaimType>
```

Password

The **Password** user input type is used to record a password entered by the user.

New Password
.....
Confirm New Password
Confirm New Password

```
<ClaimType Id="password">
  <DisplayName>Password</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Enter password</UserHelpText>
  <UserInputType>Password</UserInputType>
</ClaimType>
```

DateTimeDropdown

The **DateTimeDropdown** user input type is used to provide a set of drop-downs to select a day, month, and year. You can use Predicates and PredicateValidations elements to control the minimum and maximum date values. For more information, see the **Configure a date range** section of [Predicates and PredicateValidations](#).

Date Of Birth

Day	Month	Year
-----	-------	------

```
<ClaimType Id="dateOfBirth">
  <DisplayName>Date Of Birth</DisplayName>
  <DataType>date</DataType>
  <UserHelpText>The date on which you were born.</UserHelpText>
  <UserInputType>DateTimeDropdown</UserInputType>
</ClaimType>
```

RadioSingleSelect

The **RadioSingleSelect** user input type is used to provide a collection of radio buttons that allows the user to select one option.

Preferred color
<input type="radio"/>
Blue
<input type="radio"/>
Green
<input checked="" type="radio"/>
Orange

```
<ClaimType Id="color">
  <DisplayName>Preferred color</DisplayName>
  <DataType>string</DataType>
  <UserInputType>RadioSingleSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Blue" Value="Blue" SelectByDefault="false" />
    <Enumeration Text="Green " Value="Green" SelectByDefault="false" />
    <Enumeration Text="Orange" Value="Orange" SelectByDefault="true" />
  </Restriction>
</ClaimType>
```

DropdownSingleSelect

The **DropdownSingleSelect** user input type is used to provide a drop-down box that allows the user to select one option.

City where you work

New York
Bellevue
Redmond
New York

```
<ClaimType Id="city">
  <DisplayName>City where you work</DisplayName>
  <DataType>string</DataType>
  <UserInputType>DropdownSingleSelect</UserInputType>
  <Restriction>
    <Enumeration Text="Bellevue" Value="bellevue" SelectByDefault="false" />
    <Enumeration Text="Redmond" Value="redmond" SelectByDefault="false" />
    <Enumeration Text="New York" Value="new-york" SelectByDefault="true" />
  </Restriction>
</ClaimType>
```

CheckboxMultiSelect

The **CheckboxMultiSelect** user input type is used to provide a collection of checkboxes that allows the user to select multiple options.

Languages you speak

English

France

Spanish

```
<ClaimType Id="languages">
  <DisplayName>Languages you speak</DisplayName>
  <DataType>string</DataType>
  <UserInputType>CheckboxMultiSelect</UserInputType>
  <Restriction>
    <Enumeration Text="English" Value="English" SelectByDefault="true" />
    <Enumeration Text="France " Value="France" SelectByDefault="false" />
    <Enumeration Text="Spanish" Value="Spanish" SelectByDefault="false" />
  </Restriction>
</ClaimType>
```

Readonly

The **Readonly** user input type is used to provide a readonly field to display the claim and value.

Membership number

46726293

```
<ClaimType Id="membershipNumber">
  <DisplayName>Membership number</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Your membership number (read only)</UserHelpText>
  <UserInputType>Readonly</UserInputType>
</ClaimType>
```

Paragraph

The **Paragraph** user input type is used to provide a field that shows text only in a paragraph tag. For example, `<p>text</p>`. A **Paragraph** user input type `outputClaim` of self-asserted technical profile, must set the `Required` attribute `false` (default).

Error message:
This action can only be performed by gold members

```
<ClaimType Id="responseMsg">
  <DisplayName>Error message: </DisplayName>
  <DataType>string</DataType>
  <AdminHelpText>A claim responsible for holding response messages to send to the relying party</AdminHelpText>
  <UserHelpText>A claim responsible for holding response messages to send to the relying party</UserHelpText>
  <UserInputType>Paragraph</UserInputType>
  <Restriction>
    <Enumeration Text="B2C_V1_90001" Value="You cannot sign in because you are a minor" />
    <Enumeration Text="B2C_V1_90002" Value="This action can only be performed by gold members" />
    <Enumeration Text="B2C_V1_90003" Value="You have not been enabled for this operation" />
  </Restriction>
</ClaimType>
```

ClaimsTransformations

1/28/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **ClaimsTransformations** element contains a list of claims transformation functions that can be used in user journeys as part of a [custom policy](#). 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.

To include the list of claims transformation functions that can be used in the user journeys, a ClaimsTransformations XML element must be declared under the BuildingBlocks section of the policy.

```
<ClaimsTransformations>
  <ClaimsTransformation Id=<identifier>" TransformationMethod=<method>">
    ...
  </ClaimsTransformation>
</ClaimsTransformations>
```

The **ClaimsTransformation** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that is used to uniquely identify the claim transformation. The identifier is referenced from other XML elements in the policy.
TransformationMethod	Yes	The transform method to use in the claims transformation. Each claim transformation has its own values. See the claims transformation reference for a complete list of the available values.

ClaimsTransformation

The **ClaimsTransformation** element contains the following elements:

```
<ClaimsTransformation Id=<identifier>" TransformationMethod=<method>">
  <InputClaims>
    ...
  </InputClaims>
  <InputParameters>
    ...
  </InputParameters>
  <OutputClaims>
    ...
  </OutputClaims>
</ClaimsTransformation>
```

ELEMENT	OCCURRENCES	DESCRIPTION
InputClaims	0:1	A list of InputClaim elements that specify claim types that are taken as input to the claims transformation. Each of these elements contains a reference to a ClaimType already defined in the ClaimsSchema section in the policy.
InputParameters	0:1	A list of InputParameter elements that are provided as input to the claims transformation.
OutputClaims	0:1	A list of OutputClaim elements that specify claim types that are produced after the ClaimsTransformation has been invoked. Each of these elements contains reference to a ClaimType already defined in the ClaimsSchema section.

InputClaims

The **InputClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
InputClaim	1:n	An expected input claim type.

InputClaim

The **InputClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceId	Yes	A reference to a ClaimType already defined in the ClaimsSchema section in the policy.
TransformationClaimType	Yes	An identifier to reference a transformation claim type. Each claim transformation has its own values. See the claims transformation reference for a complete list of the available values.

InputParameters

The **InputParameters** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
InputParameter	1:n	An expected input parameter.

InputParameter

ATTRIBUTE	REQUIRED	DESCRIPTION

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that is a reference to a parameter of the claims transformation method. Each claims transformation method has its own values. See the claims transformation table for a complete list of the available values.
DataType	Yes	The type of data of the parameter, such as String, Boolean, Int, or DateTime as per the DataType enumeration in the custom policy XML schema. This type is used to perform arithmetic operations correctly. Each claim transformation has its own values. See the claims transformation reference for a complete list of the available values.
Value	Yes	A value that is passed verbatim to the transformation. Some of the values are arbitrary, some of them you select from the claims transformation method.

OutputClaims

The **OutputClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
OutputClaim	0:n	An expected output claim type.

OutputClaim

The **OutputClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceld	Yes	A reference to a ClaimType already defined in the ClaimsSchema section in the policy.
TransformationClaimType	Yes	An identifier to reference a transformation claim type. Each claim transformation has its own values. See the claims transformation reference for a complete list of the available values.

If input claim and the output claim are the same type (string, or boolean), you can use the same input claim as the output claim. In this case, the claims transformation changes the input claim with the output value.

Example

For example, you may store the last version of your terms of services that the user accepted. When you update the terms of services, you can ask the user to accept the new version. In the following example, the **HasTOSVersionChanged** claims transformation compares the value of the **TOSVersion** claim with the value of the **LastTOSAcceptedVersion** claim and then returns the boolean **TOSVersionChanged** claim.

```

<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="TOSVersionChanged">
      <DisplayName>Indicates if the TOS version accepted by the end user is equal to the current version</DisplayName>
      <DataType>boolean</DataType>
    </ClaimType>
    <ClaimType Id="TOSVersion">
      <DisplayName>TOS version</DisplayName>
      <DataType>string</DataType>
    </ClaimType>
    <ClaimType Id="LastTOSAcceptedVersion">
      <DisplayName>TOS version accepted by the end user</DisplayName>
      <DataType>string</DataType>
    </ClaimType>
  </ClaimsSchema>

  <ClaimsTransformations>
    <ClaimsTransformation Id="HasTOSVersionChanged" TransformationMethod="CompareClaims">
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="TOSVersion" TransformationClaimType="inputClaim1" />
        <InputClaim ClaimTypeReferenceId="LastTOSAcceptedVersion" TransformationClaimType="inputClaim2" />
      </InputClaims>
      <InputParameters>
        <InputParameter Id="operator" DataType="string" Value="NOT EQUAL" />
      </InputParameters>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="TOSVersionChanged" TransformationClaimType="outputClaim" />
      </OutputClaims>
    </ClaimsTransformation>
  </ClaimsTransformations>
</BuildingBlocks>

```

Claims transformations reference

For examples of claims transformations, see the following reference pages:

- [Boolean](#)
- [Date](#)
- [Integer](#)
- [JSON](#)
- [General](#)
- [Social account](#)
- [String](#)
- [StringCollection](#)

Boolean claims transformations

2/26/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using the boolean claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

AndClaims

Performs an And operation of two boolean inputClaims and sets the outputClaim with result of the operation.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	boolean	The first ClaimType to evaluate.
InputClaim	inputClaim2	boolean	The second ClaimType to evaluate.
OutputClaim	outputClaim	boolean	The ClaimTypes that will be produced after this claims transformation has been invoked (true or false).

The following claims transformation demonstrates how to And two boolean ClaimTypes: `isEmailNotExist`, and `isSocialAccount`. The output claim `presentEmailSelfAsserted` is set to `true` if the value of both input claims are `true`. In an orchestration step, you can use a precondition to preset a self-asserted page, only if a social account email is empty.

```
<ClaimsTransformation Id="CheckWhetherEmailBePresented" TransformationMethod="AndClaims">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="isEmailNotExist" TransformationClaimType="inputClaim1" />
    <InputClaim ClaimTypeReferenceId="isSocialAccount" TransformationClaimType="inputClaim2" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="presentEmailSelfAsserted" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim1:** true
 - **inputClaim2:** false
- Output claims:
 - **outputClaim:** false

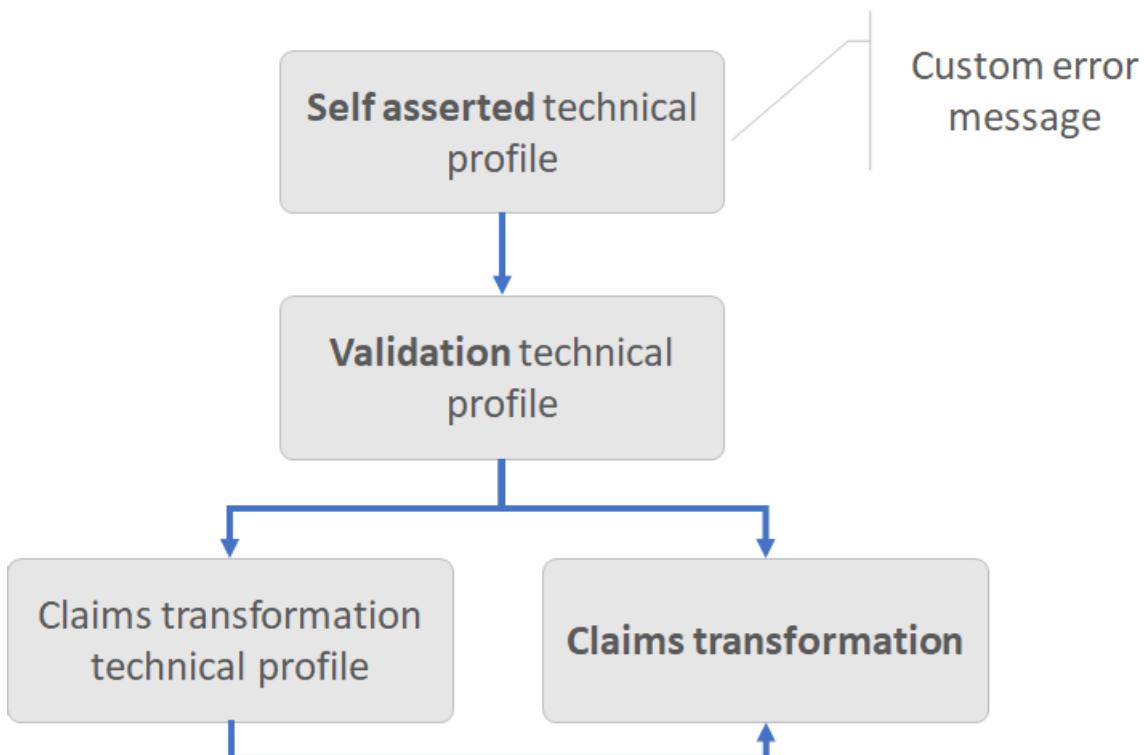
AssertBooleanClaimIsEqualToValue

Checks that boolean values of two claims are equal, and throws an exception if they are not.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
inputClaim	inputClaim	boolean	The ClaimType to be asserted.
InputParameter	valueToCompareTo	boolean	The value to compare (true or false).

The **AssertBooleanClaimIsEqualToValue** claims transformation is always executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#). The

UserMessageIfClaimsTransformationBooleanValuesNotEqual self-asserted technical profile metadata controls the error message that the technical profile presents to the user.



The following claims transformation demonstrates how to check the value of a boolean ClaimType with a `true` value. If the value of the `accountEnabled` ClaimType is false, an error message is thrown.

```
<ClaimsTransformation Id="AssertAccountEnabledIsTrue" TransformationMethod="AssertBooleanClaimIsEqualToValue">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="accountEnabled" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="valueToCompareTo" DataType="boolean" Value="true" />
  </InputParameters>
</ClaimsTransformation>
```

The `login-NonInteractive` validation technical profile calls the `AssertAccountEnabledIsTrue` claims transformation.

```

<TechnicalProfile Id="login-NonInteractive">
  ...
  <OutputClaimsTransformations>
    <OutputClaimsTransformation ReferenceId="AssertAccountEnabledIsTrue" />
  </OutputClaimsTransformations>
</TechnicalProfile>

```

The self-asserted technical profile calls the validation **login-NonInteractive** technical profile.

```

<TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email">
  <Metadata>
    <Item Key="UserMessageIfClaimsTransformationBooleanValueIsNotEqual">Custom error message if account is
disabled.</Item>
  </Metadata>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="login-NonInteractive" />
  </ValidationTechnicalProfiles>
</TechnicalProfile>

```

Example

- Input claims:
 - **inputClaim**: false
 - **valueToCompareTo**: true
- Result: Error thrown

CompareBooleanClaimToValue

Checks that boolean value of a claims is equal to `true` or `false`, and return the result of the compression.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	boolean	The ClaimType to be asserted.
InputParameter	valueToCompareTo	boolean	The value to compare (true or false).
OutputClaim	compareResult	boolean	The ClaimType that is produced after this ClaimsTransformation has been invoked.

The following claims transformation demonstrates how to check the value of a boolean ClaimType with a `true` value. If the value of the `IsAgeOver21Years` ClaimType is equal to `true`, the claims transformation returns `true`, otherwise `false`.

```

<ClaimsTransformation Id="AssertAccountEnabled" TransformationMethod="CompareBooleanClaimToValue">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="IsAgeOver21Years" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="valueToCompareTo" DataType="boolean" Value="true" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="accountEnabled" TransformationClaimType="compareResult"/>
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **inputClaim**: false
- Input parameters:
 - **valueToCompareTo**: true
- Output claims:
 - **compareResult**: false

NotClaims

Performs a Not operation of the boolean inputClaim and sets the outputClaim with result of the operation.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	boolean	The claim to be operated.
OutputClaim	outputClaim	boolean	The ClaimTypes that are produced after this ClaimsTransformation has been invoked (true or false).

Use this claim transformation to perform logical negation on a claim.

```

<ClaimsTransformation Id="CheckWhetherEmailBePresented" TransformationMethod="NotClaims">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="userExists" TransformationClaimType="inputClaim" />
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="userExists" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **inputClaim**: false
- Output claims:
 - **outputClaim**: true

OrClaims

Computes an Or of two boolean inputClaims and sets the outputClaim with result of the operation.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	boolean	The first ClaimType to evaluate.
InputClaim	inputClaim2	boolean	The second ClaimType to evaluate.
OutputClaim	outputClaim	boolean	The ClaimTypes that will be produced after this ClaimsTransformation has been invoked (true or false).

The following claims transformation demonstrates how to **or** two boolean ClaimTypes. In the orchestration step, you can use a precondition to preset a self-asserted page, if the value of one of the claims is **true**.

```
<ClaimsTransformation Id="CheckWhetherEmailBePresented" TransformationMethod="OrClaims">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="isLastTOSAcceptedNotExists" TransformationClaimType="inputClaim1" />
    <InputClaim ClaimTypeReferenceId="isLastTOSAcceptedGreaterThanOrNow" TransformationClaimType="inputClaim2" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="presentTOSSelfAsserted" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim1**: true
 - **inputClaim2**: false
- Output claims:
 - **outputClaim**: true

Date claims transformations

2/4/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

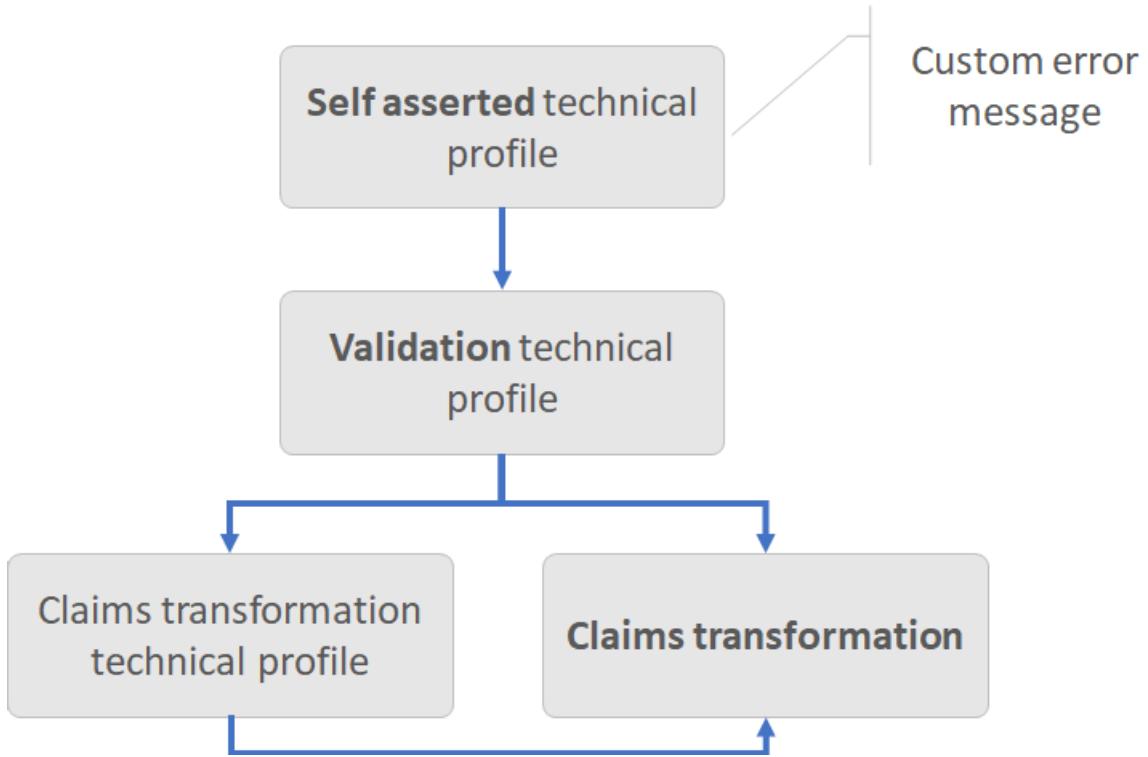
This article provides examples for using the date claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

AssertDateTimeIsGreaterThan

Checks that one date and time claim (string data type) is later than a second date and time claim (string data type), and throws an exception.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	leftOperand	string	First claim's type, which should be later than the second claim.
InputClaim	rightOperand	string	Second claim's type, which should be earlier than the first claim.
InputParameter	AssertIfEqualTo	boolean	Specifies whether this assertion should pass if the left operand is equal to the right operand.
InputParameter	AssertIfRightOperandIsNotPresent	boolean	Specifies whether this assertion should pass if the right operand is missing.
InputParameter	TreatAsEqualIfWithinMilliseconds	int	Specifies the number of milliseconds to allow between the two date times to consider the times equal (for example, to account for clock skew).

The **AssertDateTimeIsGreaterThan** claims transformation is always executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#). The **DateTimeGreaterThan** self-asserted technical profile metadata controls the error message that the technical profile presents to the user.



The following example compares the `currentDateTime` claim with the `approvedDateTime` claim. An error is thrown if `currentDateTime` is later than `approvedDateTime`. The transformation treats values as equal if they are within 5 minutes (30000 milliseconds) difference.

```

<ClaimsTransformation Id="AssertApprovedDateTimeLaterThanCurrentDateTime"
TransformationMethod="AssertDateTimeIsGreaterThan">
<InputClaims>
<InputClaim ClaimTypeReferenceId="approvedDateTime" TransformationClaimType="leftOperand" />
<InputClaim ClaimTypeReferenceId="currentDateTime" TransformationClaimType="rightOperand" />
</InputClaims>
<InputParameters>
<InputParameter Id="AssertIfEqualTo" DataType="boolean" Value="false" />
<InputParameter Id="AssertIfRightOperandIsNotPresent" DataType="boolean" Value="true" />
<InputParameter Id="TreatAsEqualIfWithinMilliseconds" DataType="int" Value="30000" />
</InputParameters>
</ClaimsTransformation>

```

The `login-NonInteractive` validation technical profile calls the `AssertApprovedDateTimeLaterThanCurrentDateTime` claims transformation.

```

<TechnicalProfile Id="login-NonInteractive">
...
<OutputClaimsTransformations>
<OutputClaimsTransformation ReferenceId="AssertApprovedDateTimeLaterThanCurrentDateTime" />
</OutputClaimsTransformations>
</TechnicalProfile>

```

The self-asserted technical profile calls the validation **login-NonInteractive** technical profile.

```

<TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email">
  <Metadata>
    <Item Key="DateTimeGreaterThan">Custom error message if the provided left operand is greater than the right operand.</Item>
  </Metadata>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="login-NonInteractive" />
  </ValidationTechnicalProfiles>
</TechnicalProfile>

```

Example

- Input claims:
 - leftOperand:** 2018-10-01T15:00:00.0000000Z
 - rightOperand:** 2018-10-01T14:00:00.0000000Z
- Result: Error thrown

ConvertDateToDateTimeClaim

Converts a **Date** ClaimType to a **DateTime** ClaimType. The claims transformation converts the time format and adds 12:00:00 AM to the date.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	date	The ClaimType to be converted.
OutputClaim	outputClaim	dateTime	The ClaimType that is produced after this ClaimsTransformation has been invoked.

The following example demonstrates the conversion of the claim `dateOfBirth` (date data type) to another claim `dateOfBirthWithTime` (dateTime data type).

```

<ClaimsTransformation Id="ConvertToDate" TransformationMethod="ConvertDateToDateTimeClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="dateOfBirth" TransformationClaimType="inputClaim" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="dateOfBirthWithTime" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - inputClaim:** 2019-06-01
- Output claims:
 - outputClaim:** 1559347200 (June 1, 2019 12:00:00 AM)

ConvertDateTimeToDateClaim

Converts a **DateTime** ClaimType to a **Date** ClaimType. The claims transformation removes the time format from the date.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	dateTime	The ClaimType to be converted.
OutputClaim	outputClaim	date	The ClaimType that is produced after this ClaimsTransformation has been invoked.

The following example demonstrates the conversion of the claim `systemDateTime` (dateTime data type) to another claim `systemDate` (date data type).

```
<ClaimsTransformation Id="ConvertToDate" TransformationMethod="ConvertDateTimeToDateClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="systemDateTime" TransformationClaimType="inputClaim" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="systemDate" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** 1559347200 (June 1, 2019 12:00:00 AM)
- Output claims:
 - **outputClaim:** 2019-06-01

GetCurrentDateTime

Get the current UTC date and time and add the value to a ClaimType.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	currentDateTime	dateTime	The ClaimType that is produced after this ClaimsTransformation has been invoked.

```
<ClaimsTransformation Id="GetSystemDateTime" TransformationMethod="GetCurrentDateTime">
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="systemDateTime" TransformationClaimType="currentDateTime" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Output claims:
 - **currentDateTime:** 1534418820 (August 16, 2018 11:27:00 AM)

DateTimeComparison

Determine whether one dateTime is later, earlier, or equal to another. The result is a new boolean ClaimType boolean with a value of `true` or `false`.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	firstDateTime	dateTime	The first dateTime to compare whether it is earlier or later than the second dateTime. Null value throws an exception.
InputClaim	secondDateTime	dateTime	The second dateTime to compare whether it is earlier or later than the first dateTime. Null value is treated as the current dateTime.
InputParameter	operator	string	One of following values: same, later than, or earlier than.
InputParameter	timeSpanInSeconds	int	Add the timespan to the first datetime.
OutputClaim	result	boolean	The ClaimType that is produced after this ClaimsTransformation has been invoked.

Use this claims transformation to determine if two ClaimTypes are equal, later, or earlier than each other. For example, you may store the last time a user accepted your terms of services (TOS). After 3 months, you can ask the user to access the TOS again. To run the claim transformation, you first need to get the current dateTime and also the last time user accepts the TOS.

```

<ClaimsTransformation Id="CompareLastTOSAcceptedWithCurrentDateTime"
TransformationMethod="DateTimeComparison">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="currentDateTime" TransformationClaimType="firstDateTime" />
    <InputClaim ClaimTypeReferenceId="extension_LastTOSAccepted" TransformationClaimType="secondDateTime" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="operator" DataType="string" Value="later than" />
    <InputParameter Id="timeSpanInSeconds" DataType="int" Value="7776000" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isLastTOSAcceptedGreaterThanNow" TransformationClaimType="result" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **firstDateTime:** 2018-01-01T00:00:00.100000Z
 - **secondDateTime:** 2018-04-01T00:00:00.100000Z
- Input parameters:
 - **operator:** later than
 - **timeSpanInSeconds:** 7776000 (90 days)
- Output claims:
 - **result:** true

General claims transformations

2/4/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using general claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

CopyClaim

Copy value of a claim to another. Both claims must be from the same type.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string, int	The claim type which is to be copied.
OutputClaim	outputClaim	string, int	The ClaimType that is produced after this ClaimsTransformation has been invoked.

Use this claims transformation to copy a value from a string or numeric claim, to another claim. The following example copies the externalEmail claim value to email claim.

```
<ClaimsTransformation Id="CopyEmailAddress" TransformationMethod="CopyClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="externalEmail" TransformationClaimType="inputClaim"/>
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" TransformationClaimType="outputClaim"/>
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** bob@contoso.com
- Output claims:
 - **outputClaim:** bob@contoso.com

DoesClaimExist

Checks if the **inputClaim** exists or not and sets **outputClaim** to true or false accordingly.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	Any	The input claim whose existence needs to be verified.
OutputClaim	outputClaim	boolean	The ClaimType that is produced after this ClaimsTransformation has been invoked.

Use this claims transformation to check if a claim exists or contains any value. The return value is a boolean that indicates whether the claim exists. Following example checks if the email address exists.

```
<ClaimsTransformation Id="CheckIfEmailPresent" TransformationMethod="DoesClaimExist">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="inputClaim" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isEmailPresent" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** someone@contoso.com
- Output claims:
 - **outputClaim:** true

Hash

Hash the provided plain text using the salt and a secret. The hashing algorithm used is SHA-256.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	plaintext	string	The input claim to be encrypted
InputClaim	salt	string	The salt parameter. You can create a random value, using CreateRandomString claims transformation.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	randomizerSecret	string	Points to an existing Azure AD B2C policy key . To create a new policy key: In your Azure AD B2C tenant, under Manage , select Identity Experience Framework . Select Policy keys to view the keys that are available in your tenant. Select Add . For Options , select Manual . Provide a name (the prefix <i>B2C_1A_</i> might be added automatically.). In the Secret text box, enter any secret you want to use, such as 1234567890. For Key usage , select Signature . Select Create .
OutputClaim	hash	string	The ClaimType that is produced after this claims transformation has been invoked. The claim configured in the plaintext inputClaim.

```

<ClaimsTransformation Id="HashPasswordWithEmail" TransformationMethod="Hash">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="password" TransformationClaimType="plaintext" />
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="salt" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="randomizerSecret" DataType="string" Value="B2C_1A_AccountTransformSecret" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="hashedPassword" TransformationClaimType="hash" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **plaintext**: MyPass@word1
 - **salt**: 487624568
 - **randomizerSecret**: B2C_1A_AccountTransformSecret
- Output claims:
 - **outputClaim**: CdMNb/KTEfsWzh9MR1kQGRZCKjuxGMWhA5YQNihzV6U=

Integer claims transformations

12/9/2019 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using the integer claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

ConvertNumberToStringClaim

Converts a long data type into a string data type.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	long	The ClaimType to convert to a string.
OutputClaim	outputClaim	string	The ClaimType that is produced after this ClaimsTransformation has been invoked.

In this example, the `numericUserId` claim with a value type of long is converted to a `userId` claim with a value type of string.

```
<ClaimsTransformation Id="CreateUserId" TransformationMethod="ConvertNumberToStringClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="numericUserId" TransformationClaimType="inputClaim" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="UserId" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** 12334 (long)
- Output claims:
 - **outputClaim:** "12334" (string)

JSON claims transformations

12/12/2019 • 5 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using the JSON claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

GenerateJson

Use either claim values or constants to generate a JSON string. The path string following dot notation is used to indicate where to insert the data into a JSON string. After splitting by dots, any integers are interpreted as the index of a JSON array and non-integers are interpreted as the index of a JSON object.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	Any string following dot notation	string	The JsonPath of the JSON where the claim value will be inserted into.
InputParameter	Any string following dot notation	string	The JsonPath of the JSON where the constant string value will be inserted into.
OutputClaim	outputClaim	string	The generated JSON string.

The following example generates a JSON string based on the claim value of "email" and "otp" as well as constant strings.

```
<ClaimsTransformation Id="GenerateRequestBody" TransformationMethod="GenerateJson">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="personalizations.0.to.0.email" />
    <InputClaim ClaimTypeReferenceId="otp"
TransformationClaimType="personalizations.0.dynamic_template_data.otp" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="template_id" DataType="string" Value="d-4c56ffb40fa648b1aa6822283df94f60"/>
    <InputParameter Id="from.email" DataType="string" Value="service@contoso.com"/>
    <InputParameter Id="personalizations.0.subject" DataType="string" Value="Contoso account email
verification code"/>
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="requestBody" TransformationClaimType="outputClaim"/>
  </OutputClaims>
</ClaimsTransformation>
```

Example

The following claims transformation outputs a JSON string claim that will be the body of the request sent to SendGrid (a third-party email provider). The JSON object's structure is defined by the IDs in dot notation of the

InputParameters and the TransformationClaimTypes of the InputClaims. Numbers in the dot notation imply arrays. The values come from the InputClaims' values and the InputParameters' "Value" properties.

- Input claims :
 - **email**, transformation claim type **personalizations.0.to.0.email**: "someone@example.com"
 - **otp**, transformation claim type **personalizations.0.dynamic_template_data.otp** "346349"
- Input parameter:
 - **template_id**: "d-4c56ffb40fa648b1aa6822283df94f60"
 - **from.email**: "service@contoso.com"
 - **personalizations.0.subject** "Contoso account email verification code"
- Output claim:
 - **requestBody**: JSON value

```
{  
  "personalizations": [  
    {  
      "to": [  
        {  
          "email": "someone@example.com"  
        }  
      ],  
      "dynamic_template_data": {  
        "otp": "346349",  
        "verify-email" : "someone@example.com"  
      },  
      "subject": "Contoso account email verification code"  
    }  
  ],  
  "template_id": "d-989077fbba9746e89f3f6411f596fb96",  
  "from": {  
    "email": "service@contoso.com"  
  }  
}
```

GetClaimFromJson

Get a specified element from a JSON data.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputJson	string	The ClaimTypes that are used by the claims transformation to get the item.
InputParameter	claimToExtract	string	the name of the JSON element to be extracted.
OutputClaim	extractedClaim	string	The ClaimType that is produced after this claims transformation has been invoked, the element value specified in the <i>claimToExtract</i> input parameter.

In the following example, the claims transformation extracted the `emailAddress` element from the JSON data:

```
{"emailAddress": "someone@example.com", "displayName": "Someone"}
```

```
<ClaimsTransformation Id="GetEmailClaimFromJson" TransformationMethod="GetClaimFromJson">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="customUserData" TransformationClaimType="inputJson" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="claimToExtract" DataType="string" Value="emailAddress" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="extractedEmail" TransformationClaimType="extractedClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputJson**: {"emailAddress": "someone@example.com", "displayName": "Someone"}
- Input parameter:
 - **claimToExtract**: emailAddress
- Output claims:
 - **extractedClaim**: someone@example.com

GetClaimsFromJsonArray

Get a list of specified elements from Json data.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	jsonSourceClaim	string	The ClaimTypes that are used by the claims transformation to get the claims.
InputParameter	errorOnMissingClaims	boolean	Specifies whether to throw an error if one of the claims is missing.
InputParameter	includeEmptyClaims	string	Specify whether to include empty claims.
InputParameter	jsonSourceKeyName	string	Element key name
InputParameter	jsonSourceValueName	string	Element value name
OutputClaim	Collection	string, int, boolean, and datetime	List of claims to extract. The name of the claim should be equal to the one specified in <i>jsonSourceClaim</i> input claim.

In the following example, the claims transformation extracts the following claims: email (string), displayName (string), membershipNum (int), active (boolean) and birthdate (datetime) from the JSON data.

```
[{"key": "email", "value": "someone@example.com"}, {"key": "displayName", "value": "Someone"}, {"key": "membershipNum", "value": 6353399}, {"key": "active", "value": true}, {"key": "birthdate", "value": "1980-09-23T00:00:00Z"}]
```

```

<ClaimsTransformation Id="GetClaimsFromJson" TransformationMethod="GetClaimsFromArray">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="jsonSourceClaim" TransformationClaimType="jsonSource" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="errorOnMissingClaims" DataType="boolean" Value="false" />
    <InputParameter Id="includeEmptyClaims" DataType="boolean" Value="false" />
    <InputParameter Id="jsonSourceKeyName" DataType="string" Value="key" />
    <InputParameter Id="jsonSourceValueName" DataType="string" Value="value" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" />
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="membershipNum" />
    <OutputClaim ClaimTypeReferenceId="active" />
    <OutputClaim ClaimTypeReferenceId="birthdate" />
  </OutputClaims>
</ClaimsTransformation>

```

- Input claims:

- **jsonSourceClaim:** [{"key": "email", "value": "someone@example.com"}, {"key": "displayName", "value": "Someone"}, {"key": "membershipNum", "value": 6353399}, {"key": "active", "value": true}, {"key": "birthdate", "value": "1980-09-23T00:00:00Z"}]

- Input parameters:

- **errorOnMissingClaims:** false
- **includeEmptyClaims:** false
- **jsonSourceKeyName:** key
- **jsonSourceValueName:** value

- Output claims:

- **email:** "someone@example.com"
- **displayName:** "Someone"
- **membershipNum:** 6353399
- **active:** true
- **birthdate:** 1980-09-23T00:00:00Z

GetNumericClaimFromJson

Gets a specified numeric (long) element from a JSON data.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputJson	string	The ClaimTypes that are used by the claims transformation to get the claim.
InputParameter	claimToExtract	string	The name of the JSON element to extract.
OutputClaim	extractedClaim	long	The ClaimType that is produced after this ClaimsTransformation has been invoked, the element's value specified in the <i>claimToExtract</i> input parameters.

In the following example, the claims transformation extracts the `id` element from the JSON data.

```
{  
    "emailAddress": "someone@example.com",  
    "displayName": "Someone",  
    "id" : 6353399  
}
```

```
<ClaimsTransformation Id="GetIdFromResponse" TransformationMethod="GetNumericClaimFromJson">  
    <InputClaims>  
        <InputClaim ClaimTypeReferenceId="exampleInputClaim" TransformationClaimType="inputJson" />  
    </InputClaims>  
    <InputParameters>  
        <InputParameter Id="claimToExtract" DataType="string" Value="id" />  
    </InputParameters>  
    <OutputClaims>  
        <OutputClaim ClaimTypeReferenceId="membershipId" TransformationClaimType="extractedClaim" />  
    </OutputClaims>  
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputJson**: {"emailAddress": "someone@example.com", "displayName": "Someone", "id": 6353399}
- Input parameters
 - **claimToExtract**: id
- Output claims:
 - **extractedClaim**: 6353399

GetSingleValueFromJsonArray

Gets the first element from a JSON data array.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputJsonClaim	string	The ClaimTypes that are used by the claims transformation to get the item from the JSON array.
OutputClaim	extractedClaim	string	The ClaimType that is produced after this ClaimsTransformation has been invoked, the first element in the JSON array.

In the following example, the claims transformation extracts the first element (email address) from the JSON array `["someone@example.com", "Someone", 6353399"]`.

```
<ClaimsTransformation Id="GetEmailFromJson" TransformationMethod="GetSingleValueFromJsonArray">  
    <InputClaims>  
        <InputClaim ClaimTypeReferenceId="userData" TransformationClaimType="inputJsonClaim" />  
    </InputClaims>  
    <OutputClaims>  
        <OutputClaim ClaimTypeReferenceId="email" TransformationClaimType="extractedClaim" />  
    </OutputClaims>  
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputJsonClaim**: ["someone@example.com", "Someone", 6353399]
- Output claims:
 - **extractedClaim**: someone@example.com

XmlStringToJsonString

Converts XML data to JSON format.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	xml	string	The ClaimTypes that are used by the claims transformation to convert the data from XML to JSON format.
OutputClaim	json	string	The ClaimType that is produced after this ClaimsTransformation has been invoked, the data in JSON format.

```
<ClaimsTransformation Id="ConvertXmlToJson" TransformationMethod="XmlStringToJsonString">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="intpuXML" TransformationClaimType="xml" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="outputJson" TransformationClaimType="json" />
  </OutputClaims>
</ClaimsTransformation>
```

In the following example, the claims transformation converts the following XML data to JSON format.

Example

Input claim:

```
<user>
  <name>Someone</name>
  <email>someone@example.com</email>
</user>
```

Output claim:

```
{
  "user": {
    "name": "Someone",
    "email": "someone@example.com"
  }
}
```

Define phone number claims transformations in Azure AD B2C

2/26/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides reference and examples for using the phone number claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information about claims transformations in general, see [ClaimsTransformations](#).

NOTE

This feature is in public preview.

ConvertPhoneNumberClaimToString

Converts a `phoneNumber` data type into a `string` data type.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	phoneNumber	phoneNumber	The ClaimType to convert to a string.
OutputClaim	phoneNumberString	string	The ClaimType that is produced after this claims transformation has been invoked.

In this example, the `cellPhoneNumber` claim with a value type of `phoneNumber` is converted to a `cellPhone` claim with a value type of `string`.

```
<ClaimsTransformation Id="PhoneNumberToString" TransformationMethod="ConvertPhoneNumberClaimToString">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="cellPhoneNumber" TransformationClaimType="phoneNumber" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="cellPhone" TransformationClaimType="phoneNumberString" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

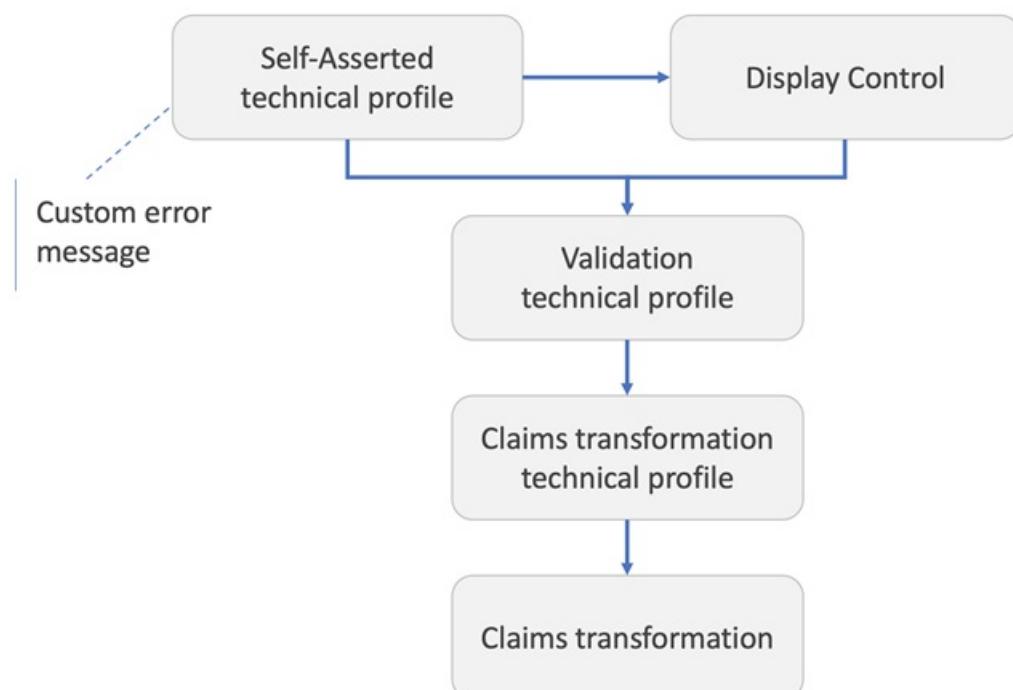
- Input claims:
 - **phoneNumber:** +11234567890 (phoneNumber)
- Output claims:
 - **phoneNumberString:** +11234567890 (string)

ConvertStringToPhoneNumberClaim

This claim transformation validates the format of the phone number. If it is in a valid format, change it to a standard format used by Azure AD B2C. If the provided phone number is not in a valid format, an error message is returned.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	phoneNumberString	string	The string claim for the phone number. The phone number has to be in international format, complete with a leading "+" and country code. If input claim <code>country</code> is provided, the phone number is in local format (without the country code).
InputClaim	country	string	[Optional] The string claim for the country code of the phone number in ISO3166 format (the two-letter ISO-3166 country code).
OutputClaim	outputClaim	phoneNumber	The result of this claims transformation.

The **ConvertStringToPhoneNumberClaim** claims transformation is always executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#) or [display control](#). The **UserMessageIfClaimsTransformationInvalidPhoneNumber** self-asserted technical profile metadata controls the error message that is presented to the user.



You can use this claims transformation to ensure that the provided string claim is a valid phone number. If not, an error message is thrown. The following example checks that the **phoneString** ClaimType is indeed a valid phone number, and then returns the phone number in the standard Azure AD B2C format. Otherwise, an error message is thrown.

```

<ClaimsTransformation Id="ConvertStringToPhoneNumber" TransformationMethod="ConvertStringToPhoneNumberClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="phoneString" TransformationClaimType="phoneNumberString" />
    <InputClaim ClaimTypeReferenceId="countryCode" TransformationClaimType="country" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="phoneNumber" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

The self-asserted technical profile that calls the validation technical profile that contains this claims transformation can define the error message.

```

<TechnicalProfile Id="SelfAsserted-LocalAccountSignup-Phone">
  <Metadata>
    <Item Key="UserMessageIfClaimsTransformationInvalidPhoneNumber">Custom error message if the phone number is not valid.</Item>
  </Metadata>
  ...
</TechnicalProfile>

```

Example 1

- Input claims:
 - **phoneNumberString**: 033 456-7890
 - **country**: DK
- Output claims:
 - **outputClaim**: +450334567890

Example 2

- Input claims:
 - **phoneNumberString**: +1 (123) 456-7890
- Output claims:
 - **outputClaim**: +11234567890

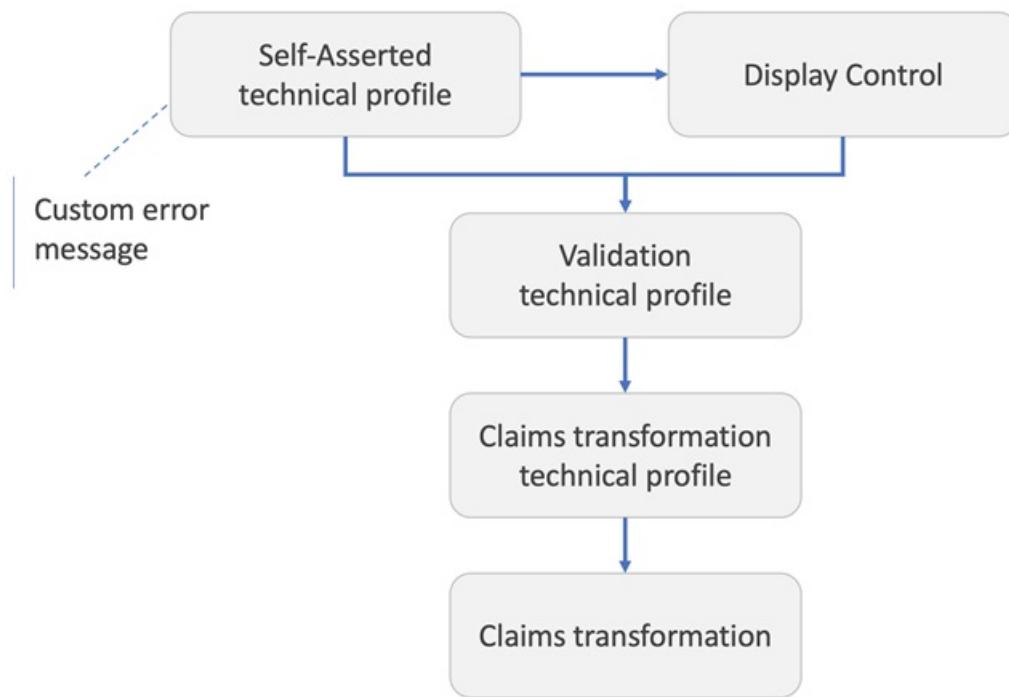
GetNationalNumberAndCountryCodeFromPhoneNumberString

This extracts the country code and the national number from the input claim, and optionally throws an exception if the supplied phone number is not valid.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	phoneNumber	string	The string claim of the phone number. The phone number has to be in international format, complete with a leading "+" and country code.
InputParameter	throwExceptionOnFailure	boolean	[Optional] A parameter indicating whether an exception is thrown when the phone number is not valid. Default value is false.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	countryCodeType	string	[Optional] A parameter indicating the type of country code in the output claim. Available values are CallingCode (the international calling code for a country, for example +1) or ISO3166 (the two-letter ISO-3166 country code).
OutputClaim	nationalNumber	string	The string claim for the national number of the phone number.
OutputClaim	countryCode	string	The string claim for the country code of the phone number.

If the **GetNationalNumberAndCountryCodeFromPhoneNumberString** claims transformation is executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#) or a [display control action](#), then the **UserMessageIfPhoneNumberParseFailure** self-asserted technical profile metadata controls the error message that is presented to the user.



You can use this claims transformation to split a full phone number into the country code and the national number. If the phone number provided is not valid, you can choose to throw an error message.

The following example tries to split the phone number into national number and country code. If the phone number is valid, the phone number will be overridden by the national number. If the phone number is not valid, an exception will not be thrown and the phone number still has its original value.

```

<ClaimsTransformation Id="GetNationalNumberAndCountryCodeFromPhoneNumberString"
TransformationMethod="GetNationalNumberAndCountryCodeFromPhoneNumberString">
<InputClaims>
<InputClaim ClaimTypeReferenceId="phoneNumber" TransformationClaimType="phoneNumber" />
</InputClaims>
<InputParameters>
<InputParameter Id="throwExceptionOnFailure" DataType="boolean" Value="false" />
<InputParameter Id="countryCodeType" DataType="string" Value="ISO3166" />
</InputParameters>
<OutputClaims>
<OutputClaim ClaimTypeReferenceId="nationalNumber" TransformationClaimType="nationalNumber" />
<OutputClaim ClaimTypeReferenceId="countryCode" TransformationClaimType="countryCode" />
</OutputClaims>
</ClaimsTransformation>

```

The self-asserted technical profile that calls the validation technical profile that contains this claims transformation can define the error message.

```

<TechnicalProfile Id="SelfAsserted-LocalAccountSignup-Phone">
<Metadata>
<Item Key="UserMessageIfPhoneNumberParseFailure">Custom error message if the phone number is not valid.
</Item>
</Metadata>
...
</TechnicalProfile>

```

Example 1

- Input claims:
 - **phoneNumber:** +49 (123) 456-7890
- Input parameters:
 - **throwExceptionOnFailure:** false
 - **countryCodeType:** ISO3166
- Output claims:
 - **nationalNumber:** 1234567890
 - **countryCode:** DE

Example 2

- Input claims:
 - **phoneNumber:** +49 (123) 456-7890
- Input parameters
 - **throwExceptionOnFailure:** false
 - **countryCodeType:** CallingCode
- Output claims:
 - **nationalNumber:** 1234567890
 - **countryCode:** +49

Social accounts claims transformations

2/20/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

In Azure Active Directory B2C (Azure AD B2C), social account identities are stored in a `userIdentities` attribute of a **alternativeSecurityIdCollection** claim type. Each item in the **alternativeSecurityIdCollection** specifies the issuer (identity provider name, such as facebook.com) and the `issuerUserId`, which is a unique user identifier for the issuer.

```
"userIdentities": [{
  "issuer": "google.com",
  "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw"
},
{
  "issuer": "facebook.com",
  "issuerUserId": "MTIzNDU="
}]
```

This article provides examples for using the social account claims transformations of the Identity Experience Framework schema in Azure AD B2C. For more information, see [ClaimsTransformations](#).

CreateAlternativeSecurityId

Creates a JSON representation of the user's alternativeSecurityId property that can be used in the calls to Azure Active Directory. For more information, see the [AlternativeSecurityId](#) schema.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	key	string	The ClaimType that specifies the unique user identifier used by the social identity provider.
InputClaim	identityProvider	string	The ClaimType that specifies the social account identity provider name, such as facebook.com.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	alternativeSecurityId	string	The ClaimType that is produced after the ClaimsTransformation has been invoked. Contains information about the identity of a social account user. The issuer is the value of the identityProvider claim. The issuerUserId is the value of the key claim in base64 format.

Use this claims transformation to generate a **alternativeSecurityId** ClaimType. It's used by all social identity provider technical profiles, such as **Facebook-OAUTH**. The following claims transformation receives the user social account ID and the identity provider name. The output of this technical profile is a JSON string format that can be used in Azure AD directory services.

```
<ClaimsTransformation Id="CreateAlternativeSecurityId" TransformationMethod="CreateAlternativeSecurityId">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="issuerUserId" TransformationClaimType="key" />
    <InputClaim ClaimTypeReferenceId="identityProvider" TransformationClaimType="identityProvider" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="alternativeSecurityId" TransformationClaimType="alternativeSecurityId" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **key:** 12334
 - **identityProvider:** Facebook.com
- Output claims:
 - **alternativeSecurityId:** { "issuer": "facebook.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw"}

AddItemToAlternativeSecurityIdCollection

Adds an **AlternativeSecurityId** to an **alternativeSecurityIdCollection** claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	item	string	The ClaimType to be added to the output claim.
InputClaim	collection	alternativeSecurityIdCollection	The ClaimTypes that are used by the claims transformation if available in the policy. If provided, the claims transformation adds the item at the end of the collection.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	collection	alternativeSecurityIdCollection	The ClaimTypes that are produced after this ClaimsTransformation has been invoked. The new collection that contains both the items from input collection and item.

The following example links a new social identity with an existing account. To link a new social identity:

1. In the **AAD-UserReadUsingAlternativeSecurityId** and **AAD-UserReadUsingObjectId** technical profiles, output the user's **alternativeSecurityIds** claim.
2. Ask the user to sign in with one of the identity providers that are not associated with this user.
3. Using the **CreateAlternativeSecurityId** claims transformation, create a new **alternativeSecurityId** claim type with a name of `AlternativeSecurityId2`
4. Call the **AddItemToAlternativeSecurityIdCollection** claims transformation to add the **AlternativeSecurityId2** claim to the existing **AlternativeSecurityIds** claim.
5. Persist the **alternativeSecurityIds** claim to the user account

```
<ClaimsTransformation Id="AddAnotherAlternativeSecurityId"
TransformationMethod="AddItemToAlternativeSecurityIdCollection">
<InputClaims>
    <InputClaim ClaimTypeReferenceId="AlternativeSecurityId2" TransformationClaimType="item" />
    <InputClaim ClaimTypeReferenceId="AlternativeSecurityIds" TransformationClaimType="collection" />
</InputClaims>
<OutputClaims>
    <OutputClaim ClaimTypeReferenceId="AlternativeSecurityIds" TransformationClaimType="collection" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **item**: { "issuer": "facebook.com", "issuerUserId": "MTIzNDU=" }
 - **collection**: [{ "issuer": "live.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw" }]
- Output claims:
 - **collection**: [{ "issuer": "live.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw" }, { "issuer": "facebook.com", "issuerUserId": "MTIzNDU=" }]

GetIdentityProvidersFromAlternativeSecurityIdCollectionTransformation

Returns list of issuers from the **alternativeSecurityIdCollection** claim into a new **stringCollection** claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	alternativeSecurityIdCollection	alternativeSecurityIdCollection	The ClaimType to be used to get the list of identity providers (issuer).

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	identityProvidersCollection	stringCollection	The ClaimTypes that are produced after this ClaimsTransformation has been invoked. List of identity providers associate with the alternativeSecurityIdCollection input claim

The following claims transformation reads the user **alternativeSecurityIds** claim and extracts the list of identity provider names associated with that account. Use output **identityProvidersCollection** to show the user the list of identity providers associated with the account. Or, on the identity provider selection page, filter the list of identity providers based on output **identityProvidersCollection** claim. So, user can select to link new social identity that is not already associated with the account.

```
<ClaimsTransformation Id="ExtractIdentityProviders"
TransformationMethod="GetIdentityProvidersFromAlternativeSecurityIdCollectionTransformation">
<InputClaims>
<InputClaim ClaimTypeReferenceId="alternativeSecurityIds"
TransformationClaimType="alternativeSecurityIdCollection" />
</InputClaims>
<OutputClaims>
<OutputClaim ClaimTypeReferenceId="identityProviders"
TransformationClaimType="identityProvidersCollection" />
</OutputClaims>
</ClaimsTransformation>
```

- Input claims:
 - **alternativeSecurityIdCollection:** [{ "issuer": "google.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw" }, { "issuer": "facebook.com", "issuerUserId": "MTIzNDU=" }]
- Output claims:
 - **identityProvidersCollection:** ["facebook.com", "google.com"]

RemoveAlternativeSecurityIdByIdentityProvider

Removes an **AlternativeSecurityId** from an **alternativeSecurityIdCollection** claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	identityProvider	string	The ClaimType that contains the identity provider name to be removed from the collection.
InputClaim	collection	alternativeSecurityIdCollection	The ClaimTypes that are used by the claims transformation. The claims transformation removes the identityProvider from the collection.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	collection	alternativeSecurityIdCollection	The ClaimTypes that are produced after this ClaimsTransformation has been invoked. The new collection, after the identityProvider removed from the collection.

The following example unlinks one of the social identity with an existing account. To unlink a social identity:

1. In the **AAD-UserReadUsingAlternativeSecurityId** and **AAD-UserReadUsingObjectId** technical profiles, output the user's **alternativeSecurityIds** claim.
2. Ask the user to select which social account to remove from the list identity providers that are associated with this user.
3. Call a claims transformation technical profile that calls the **RemoveAlternativeSecurityIdByIdentityProvider** claims transformation, that removed the selected social identity, using identity provider name.
4. Persist the **alternativeSecurityIds** claim to the user account.

```
<ClaimsTransformation Id="RemoveAlternativeSecurityIdByIdentityProvider"
TransformationMethod="RemoveAlternativeSecurityIdByIdentityProvider">
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="secondIdentityProvider" TransformationClaimType="identityProvider"
/>
        <InputClaim ClaimTypeReferenceId="AlternativeSecurityIds" TransformationClaimType="collection" />
    </InputClaims>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="AlternativeSecurityIds" TransformationClaimType="collection" />
    </OutputClaims>
</ClaimsTransformation>
</ClaimsTransformations>
```

Example

- Input claims:
 - **identityProvider**: facebook.com
 - **collection**: [{ "issuer": "live.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw" }, { "issuer": "facebook.com", "issuerUserId": "MTIzNDU=" }]
- Output claims:
 - **collection**: [{ "issuer": "live.com", "issuerUserId": "MTA4MTQ2MDgyOTI3MDUyNTYzMjcw" }]

StringCollection claims transformations

2/27/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using the string collection claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [ClaimsTransformations](#).

AddItemToStringCollection

Adds a string claim to a new unique values stringCollection claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	item	string	The ClaimType to be added to the output claim.
InputClaim	collection	stringCollection	[Optional] If specified, the claims transformation copies the items from this collection, and adds the item to the end of the output collection claim.
OutputClaim	collection	stringCollection	The ClaimType that is produced after this claims transformation has been invoked, with the value specified in the input claim.

Use this claims transformation to add a string to a new or existing stringCollection. It's commonly used in a **AAD-UserWriteUsingAlternativeSecurityId** technical profile. Before a new social account is created, **CreateOtherMailsFromEmail** claims transformation reads the ClaimType and adds the value to the **otherMails** ClaimType.

The following claims transformation adds the **email** ClaimType to **otherMails** ClaimType.

```
<ClaimsTransformation Id="CreateOtherMailsFromEmail" TransformationMethod="AddItemToStringCollection">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="item" />
    <InputClaim ClaimTypeReferenceId="otherMails" TransformationClaimType="collection" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="otherMails" TransformationClaimType="collection" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **collection**: ["someone@outlook.com"]
 - **item**: "admin@contoso.com"
- Output claims:
 - **collection**: ["someone@outlook.com", "admin@contoso.com"]

AddParameterToStringCollection

Adds a string parameter to a new unique values stringCollection claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	collection	stringCollection	[Optional] If specified, the claims transformation copies the items from this collection, and adds the item to the end of the output collection claim.
InputParameter	item	string	The value to be added to the output claim.
OutputClaim	collection	stringCollection	The ClaimType that is produced after this claims transformation has been invoked, with the value specified in the input parameter.

Use this claims transformation to add a string value to a new or existing stringCollection. The following example adds a constant email address (admin@contoso.com) to the **otherMails** claim.

```
<ClaimsTransformation Id="SetCompanyEmail" TransformationMethod="AddParameterToStringCollection">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="otherMails" TransformationClaimType="collection" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="item" DataType="string" Value="admin@contoso.com" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="otherMails" TransformationClaimType="collection" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **collection**: ["someone@outlook.com"]
- Input parameters
 - **item**: "admin@contoso.com"
- Output claims:
 - **collection**: ["someone@outlook.com", "admin@contoso.com"]

GetSingleItemFromStringCollection

Gets the first item from the provided string collection.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	collection	stringCollection	The ClaimTypes that are used by the claims transformation to get the item.
OutputClaim	extractedItem	string	The ClaimTypes that are produced after this ClaimsTransformation has been invoked. The first item in the collection.

The following example reads the **otherMails** claim and return the first item into the **email** claim.

```
<ClaimsTransformation Id="CreateEmailFromOtherMails" TransformationMethod="GetSingleItemFromStringCollection">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="otherMails" TransformationClaimType="collection" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" TransformationClaimType="extractedItem" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **collection**: ["someone@outlook.com", "someone@contoso.com"]
- Output claims:
 - **extractedItem**: "someone@outlook.com"

StringCollectionContains

Checks if a StringCollection claim type contains an element

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	stringCollection	The claim type which is to be searched.
InputParameter	item	string	The value to search.
InputParameter	ignoreCase	string	Specifies whether this comparison should ignore the case of the strings being compared.
OutputClaim	outputClaim	boolean	The ClaimType that is produced after this ClaimsTransformation has been invoked. A boolean indicator if the collection contains such a string

Following example checks whether the **roles** stringCollection claim type contains the value of **admin**.

```
<ClaimsTransformation Id="IsAdmin" TransformationMethod="StringCollectionContains">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="roles" TransformationClaimType="inputClaim"/>
  </InputClaims>
  <InputParameters>
    <InputParameter Id="item" DataType="string" Value="Admin"/>
    <InputParameter Id="ignoreCase" DataType="string" Value="true"/>
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isAdmin" TransformationClaimType="outputClaim"/>
  </OutputClaims>
</ClaimsTransformation>
```

- Input claims:
 - **inputClaim**: ["reader", "author", "admin"]
- Input parameters:
 - **item**: "Admin"
 - **ignoreCase**: "true"
- Output claims:
 - **outputClaim**: "true"

String claims transformations

2/24/2020 • 19 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

This article provides examples for using the string claims transformations of the Identity Experience Framework schema in Azure Active Directory B2C (Azure AD B2C). For more information, see [Claims Transformations](#).

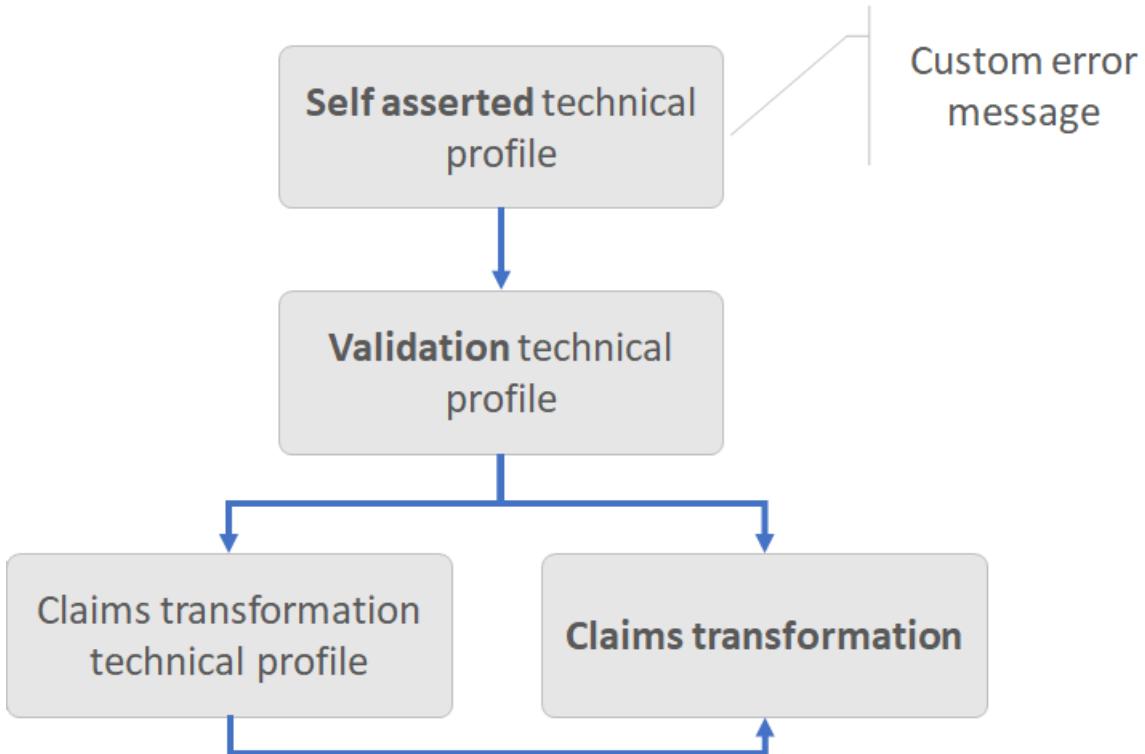
AssertStringClaimsAreEqual

Compare two claims, and throw an exception if they are not equal according to the specified comparison inputClaim1, inputClaim2 and stringComparison.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	string	First claim's type, which is to be compared.
InputClaim	inputClaim2	string	Second claim's type, which is to be compared.
InputParameter	stringComparison	string	string comparison, one of the values: Ordinal, OrdinalIgnoreCase.

The **AssertStringClaimsAreEqual** claims transformation is always executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#), or a [DisplayConrtol](#). The

`UserMessageIfClaimsTransformationStringsAreNotEqual` metadata of a self-asserted technical profile controls the error message that is presented to the user.



You can use this claims transformation to make sure, two ClaimTypes have the same value. If not, an error message is thrown. The following example checks that the **strongAuthenticationEmailAddress** ClaimType is equal to **email** ClaimType. Otherwise an error message is thrown.

```

<ClaimsTransformation Id="AssertEmailAndStrongAuthenticationEmailAddressAreEqual"
TransformationMethod="AssertStringClaimsAreEqual">
<InputClaims>
<InputClaim ClaimTypeReferenceId="strongAuthenticationEmailAddress" TransformationClaimType="inputClaim1"/>
<InputClaim ClaimTypeReferenceId="email" TransformationClaimType="inputClaim2"/>
</InputClaims>
<InputParameters>
<InputParameter Id="stringComparison" DataType="string" Value="ordinalIgnoreCase" />
</InputParameters>
</ClaimsTransformation>

```

The **login-NonInteractive** validation technical profile calls the **AssertEmailAndStrongAuthenticationEmailAddressAreEqual** claims transformation.

```

<TechnicalProfile Id="login-NonInteractive">
...
<OutputClaimsTransformations>
<OutputClaimsTransformation ReferenceId="AssertEmailAndStrongAuthenticationEmailAddressAreEqual" />
</OutputClaimsTransformations>
</TechnicalProfile>

```

The self-asserted technical profile calls the validation **login-NonInteractive** technical profile.

```

<TechnicalProfile Id="SelfAsserted-LocalAccountSignin-Email">
  <Metadata>
    <Item Key="UserMessageIfClaimsTransformationStringsAreNotEqual">Custom error message the email addresses you provided are not the same.</Item>
  </Metadata>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="login-NonInteractive" />
  </ValidationTechnicalProfiles>
</TechnicalProfile>

```

Example

- Input claims:
 - **inputClaim1**: someone@contoso.com
 - **inputClaim2**: someone@outlook.com
- Input parameters:
 - **stringComparison**: ordinalIgnoreCase
- Result: Error thrown

ChangeCase

Changes the case of the provided claim to lower or upper case depending on the operator.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	string	The ClaimType to be changed.
InputParameter	toCase	string	One of the following values: LOWER or UPPER .
OutputClaim	outputClaim	string	The ClaimType that is produced after this claims transformation has been invoked.

Use this claim transformation to change any string ClaimType to lower or upper case.

```

<ClaimsTransformation Id="ChangeToLower" TransformationMethod="ChangeCase">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="inputClaim1" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="toCase" DataType="string" Value="LOWER" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **email**: SomeOne@contoso.com
- Input parameters:
 - **toCase**: LOWER
- Output claims:

- **email:** someone@contoso.com

CreateStringClaim

Creates a string claim from the provided input parameter in the transformation.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	value	string	The string to be set. This input parameter supports string claims transformation expressions .
OutputClaim	createdClaim	string	The ClaimType that is produced after this claims transformation has been invoked, with the value specified in the input parameter.

Use this claims transformation to set a string ClaimType value.

```
<ClaimsTransformation Id="CreateTermsOfService" TransformationMethod="CreateStringClaim">
  <InputParameters>
    <InputParameter Id="value" DataType="string" Value="Contoso terms of service..." />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="TOS" TransformationClaimType="createdClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input parameter:
 - **value:** Contoso terms of service...
- Output claims:
 - **createdClaim:** The TOS ClaimType contains the "Contoso terms of service..." value.

CompareClaims

Determine whether one string claim is equal to another. The result is a new boolean ClaimType with a value of

`true` or `false`.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	string	First claim type, which is to be compared.
InputClaim	inputClaim2	string	Second claim type, which is to be compared.
InputParameter	operator	string	Possible values: <code>EQUAL</code> or <code>NOT EQUAL</code> .

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	ignoreCase	boolean	Specifies whether this comparison should ignore the case of the strings being compared.
OutputClaim	outputClaim	boolean	The ClaimType that is produced after this claims transformation has been invoked.

Use this claims transformation to check if a claim is equal to another claim. For example, the following claims transformation checks if the value of the **email** claim is equal to the **Verified.Email** claim.

```
<ClaimsTransformation Id="CheckEmail" TransformationMethod="CompareClaims">
<InputClaims>
    <InputClaim ClaimTypeReferenceId="Email" TransformationClaimType="inputClaim1" />
    <InputClaim ClaimTypeReferenceId="Verified.Email" TransformationClaimType="inputClaim2" />
</InputClaims>
<InputParameters>
    <InputParameter Id="operator" DataType="string" Value="NOT EQUAL" />
    <InputParameter Id="ignoreCase" DataType="string" Value="true" />
</InputParameters>
<OutputClaims>
    <OutputClaim ClaimTypeReferenceId="SameEmailAddress" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim1**: someone@contoso.com
 - **inputClaim2**: someone@outlook.com
- Input parameters:
 - **operator**: NOT EQUAL
 - **ignoreCase**: true
- Output claims:
 - **outputClaim**: true

CompareClaimToValue

Determines whether a claim value is equal to the input parameter value.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim1	string	The claim's type, which is to be compared.
InputParameter	operator	string	Possible values: EQUAL or NOT EQUAL .
InputParameter	compareTo	string	string comparison, one of the values: Ordinal , OrdinalIgnoreCase .

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	ignoreCase	boolean	Specifies whether this comparison should ignore the case of the strings being compared.
OutputClaim	outputClaim	boolean	The ClaimType that is produced after this claims transformation has been invoked.

You can use this claims transformation to check if a claim is equal to a value you specified. For example, the following claims transformation checks if the value of the **termsOfUseConsentVersion** claim is equal to **v1**.

```
<ClaimsTransformation Id="IsTermsOfUseConsentRequiredForVersion" TransformationMethod="CompareClaimToValue">
<InputClaims>
    <InputClaim ClaimTypeReferenceId="termsOfUseConsentVersion" TransformationClaimType="inputClaim1" />
</InputClaims>
<InputParameters>
    <InputParameter Id="compareTo" DataType="string" Value="V1" />
    <InputParameter Id="operator" DataType="string" Value="not equal" />
    <InputParameter Id="ignoreCase" DataType="string" Value="true" />
</InputParameters>
<OutputClaims>
    <OutputClaim ClaimTypeReferenceId="termsOfUseConsentRequired" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim1**: v1
- Input parameters:
 - **compareTo**: V1
 - **operator**: EQUAL
 - **ignoreCase**: true
- Output claims:
 - **outputClaim**: true

CreateRandomString

Creates a random string using the random number generator. If the random number generator is of type **integer**, optionally a seed parameter and a maximum number may be provided. An optional string format parameter allows the output to be formatted using it, and an optional base64 parameter specifies whether the output is base64 encoded randomGeneratorType [guid, integer] outputClaim (String).

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	randomGeneratorType	string	Specifies the random value to be generated, GUID (global unique ID) or INTEGER (a number).
InputParameter	stringFormat	string	[Optional] Format the random value.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	base64	boolean	[Optional] Convert the random value to base64. If string format is applied, the value after string format is encoded to base64.
InputParameter	maximumNumber	int	[Optional] For INTEGER randomGeneratorType only. Specify the maximum number.
InputParameter	seed	int	[Optional] For INTEGER randomGeneratorType only. Specify the seed for the random value. Note: same seed yields same sequence of random numbers.
OutputClaim	outputClaim	string	The ClaimTypes that will be produced after this claims transformation has been invoked. The random value.

Following example generates a global unique ID. This claims transformation is used to create the random UPN (user principle name).

```
<ClaimsTransformation Id="CreateRandomUPNUserName" TransformationMethod="CreateRandomString">
<InputParameters>
    <InputParameter Id="randomGeneratorType" DataType="string" Value="GUID" />
</InputParameters>
<OutputClaims>
    <OutputClaim ClaimTypeReferenceId="upnUserName" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input parameters:
 - **randomGeneratorType:** GUID
- Output claims:
 - **outputClaim:** bc8bedd2-aaa3-411e-bdee-2f1810b73dfc

Following example generates an integer random value between 0 and 1000. The value is formatted to OTP_{random value}.

```
<ClaimsTransformation Id="SetRandomNumber" TransformationMethod="CreateRandomString">
<InputParameters>
    <InputParameter Id="randomGeneratorType" DataType="string" Value="INTEGER" />
    <InputParameter Id="maximumNumber" DataType="int" Value="1000" />
    <InputParameter Id="stringFormat" DataType="string" Value="OTP_{0}" />
    <InputParameter Id="base64" DataType="boolean" Value="false" />
</InputParameters>
<OutputClaims>
    <OutputClaim ClaimTypeReferenceId="randomNumber" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input parameters:
 - **randomGeneratorType**: INTEGER
 - **maximumNumber**: 1000
 - **stringFormat**: OTP_{0}
 - **base64**: false
- Output claims:
 - **outputClaim**: OTP_853

FormatStringClaim

Format a claim according to the provided format string. This transformation uses the C# `String.Format` method.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The ClaimType that acts as string format {0} parameter.
InputParameter	stringFormat	string	The string format, including the {0} parameter. This input parameter supports string claims transformation expressions .
OutputClaim	outputClaim	string	The ClaimType that is produced after this claims transformation has been invoked.

Use this claims transformation to format any string with one parameter {0}. The following example creates a **userPrincipalName**. All social identity provider technical profiles, such as `Facebook-OAUTH` calls the **CreateUserPrincipalName** to generate a **userPrincipalName**.

```
<ClaimsTransformation Id="CreateUserPrincipalName" TransformationMethod="FormatStringClaim">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="upnUserName" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="stringFormat" DataType="string" Value="cpim_{0}@{RelyingPartyTenantId}" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="userPrincipalName" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim**: 5164db16-3eee-4629-bfda-dcc3326790e9
- Input parameters:
 - **stringFormat**: cpim_{0}@{RelyingPartyTenantId}
- Output claims:
 - **outputClaim**: cpim_5164db16-3eee-4629-bfda-dcc3326790e9@b2cdemo.onmicrosoft.com

FormatStringMultipleClaims

Format two claims according to the provided format string. This transformation uses the C# `String.Format` method.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The ClaimType that acts as string format {0} parameter.
InputClaim	inputClaim	string	The ClaimType that acts as string format {1} parameter.
InputParameter	stringFormat	string	The string format, including the {0} and {1} parameters. This input parameter supports string claims transformation expressions .
OutputClaim	outputClaim	string	The ClaimType that is produced after this claims transformation has been invoked.

Use this claims transformation to format any string with two parameters, {0} and {1}. The following example creates a **displayName** with the specified format:

```
<ClaimsTransformation Id="CreateDisplayNameFromFirstNameAndLastName"
TransformationMethod="FormatStringMultipleClaims">
<InputClaims>
<InputClaim ClaimTypeReferenceId="givenName" TransformationClaimType="inputClaim1" />
<InputClaim ClaimTypeReferenceId="surName" TransformationClaimType="inputClaim2" />
</InputClaims>
<InputParameters>
<InputParameter Id="stringFormat" DataType="string" Value="{0} {1}" />
</InputParameters>
<OutputClaims>
<OutputClaim ClaimTypeReferenceId="displayName" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim1**: Joe
 - **inputClaim2**: Fernando
- Input parameters:
 - **stringFormat**: {0} {1}
- Output claims:
 - **outputClaim**: Joe Fernando

GetLocalizedStringsTransformation

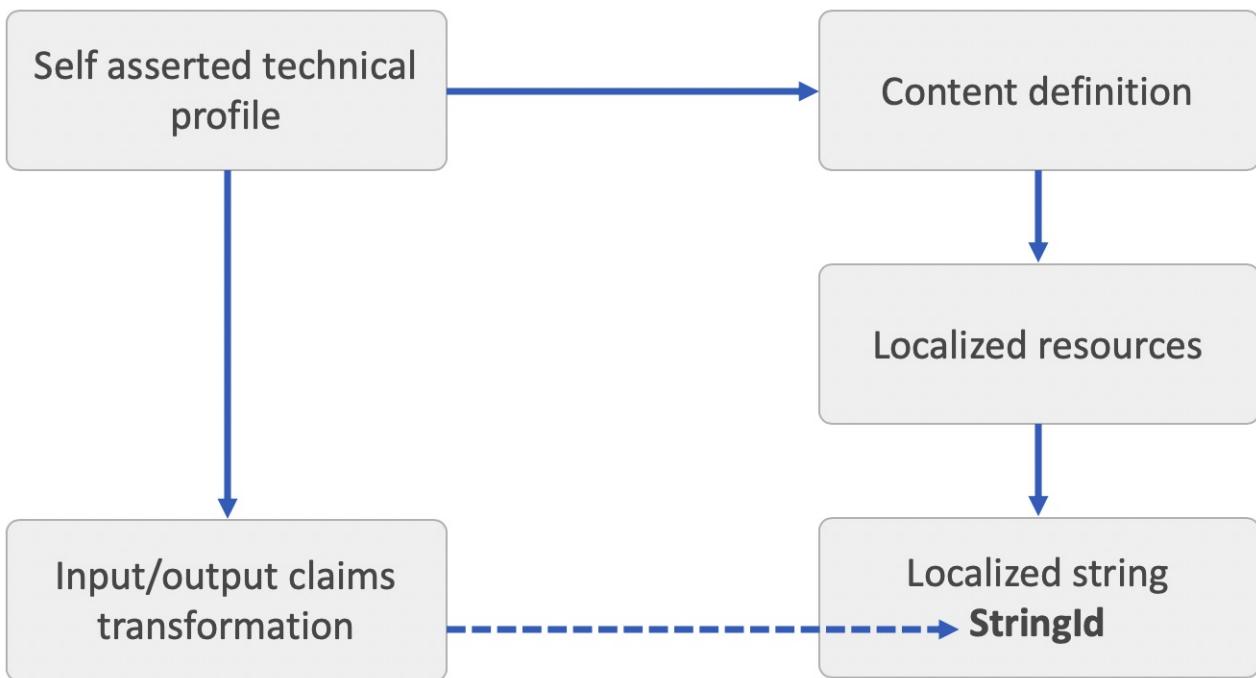
Copies localized strings into claims.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	The name of the localized string	string	List of claim types that is produced after this claims transformation has been invoked.

To use the GetLocalizedStringsTransformation claims transformation:

1. Define a [localization string](#) and associate it with a [self-asserted-technical-profile](#).
2. The `ElementType` of the `LocalizedString` element must set to `GetLocalizedStringsTransformationClaimType`.
3. The `StringId` is a unique identifier that you define, and use it later in your claims transformation.
4. In the claims transformation, specify the list of claims to be set with the localized string. The `ClaimTypeReferenceId` is a reference to a ClaimType already defined in the ClaimsSchema section in the policy. The `TransformationClaimType` is the name of the localized string as defined in the `StringId` of the `LocalizedString` element.
5. In a [self-asserted technical profile](#), or a [display control](#) input or output claims transformation, make a reference to your claims transformation.



The following example looks up the email subject, body, your code message, and the signature of the email, from localized strings. These claims later used by custom email verification template.

Define localized strings for English (default) and Spanish.

```

<Localization Enabled="true">
  <SupportedLanguages DefaultLanguage="en" MergeBehavior="Append">
    <SupportedLanguage>en</SupportedLanguage>
    <SupportedLanguage>es</SupportedLanguage>
  </SupportedLanguages>

  <LocalizedResources Id="api.localaccounts.signup.en">
    <LocalizedStrings>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_subject">Contoso account email verification code</LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_message">Thanks for verifying your account!</LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType" StringId="email_code">Your
code is</LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_signature">Sincerely</LocalizedString>
    </LocalizedStrings>
  </LocalizedResources>
  <LocalizedResources Id="api.localaccounts.signup.es">
    <LocalizedStrings>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_subject">Código de verificación del correo electrónico de la cuenta de
Contoso</LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_message">Gracias por comprobar la cuenta de </LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType" StringId="email_code">Su
código es</LocalizedString>
      <LocalizedString ElementType="GetLocalizedStringsTransformationClaimType"
StringId="email_signature">Atentamente</LocalizedString>
    </LocalizedStrings>
  </LocalizedResources>
</Localization>

```

The claims transformation sets the value of the claim type *subject* with the value of the `StringId` *email_subject*.

```

<ClaimsTransformation Id="GetLocalizedStringsForEmail"
TransformationMethod="GetLocalizedStringsTransformation">
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="subject" TransformationClaimType="email_subject" />
    <OutputClaim ClaimTypeReferenceId="message" TransformationClaimType="email_message" />
    <OutputClaim ClaimTypeReferenceId="codeIntro" TransformationClaimType="email_code" />
    <OutputClaim ClaimTypeReferenceId="signature" TransformationClaimType="email_signature" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Output claims:
 - **subject:** Contoso account email verification code
 - **message:** Thanks for verifying your account!
 - **codeIntro:** Your code is
 - **signature:** Sincerely

GetMappedValueFromLocalizedCollection

Looking up an item from a claim **Restriction** collection.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	mapFromClaim	string	The claim that contains the text to be looked up in the restrictionValueClaim claims with the Restriction collection.
OutputClaim	restrictionValueClaim	string	The claim that contains the Restriction collection. After the claims transformation has been invoked, the value of this claim contains the value of the selected item.

The following example looks up the error message description based on the error key. The **responseMsg** claim contains a collection of error messages to present to the end user or to be sent to the relying party.

```
<ClaimType Id="responseMsg">
  <DisplayName>Error message: </DisplayName>
  <DataType>string</DataType>
  <UserInputType>Paragraph</UserInputType>
  <Restriction>
    <Enumeration Text="B2C_V1_90001" Value="You cannot sign in because you are a minor" />
    <Enumeration Text="B2C_V1_90002" Value="This action can only be performed by gold members" />
    <Enumeration Text="B2C_V1_90003" Value="You have not been enabled for this operation" />
  </Restriction>
</ClaimType>
```

The claims transformation looks up the text of the item and returns its value. If the restriction is localized using **<LocalizedCollection>**, the claims transformation returns the localized value.

```
<ClaimsTransformation Id="GetResponseMsgMappedToResponseCode"
  TransformationMethod="GetMappedValueFromLocalizedCollection">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="responseCode" TransformationClaimType="mapFromClaim" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="responseMsg" TransformationClaimType="restrictionValueClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **mapFromClaim:** B2C_V1_90001
- Output claims:
 - **restrictionValueClaim:** You cannot sign in because you are a minor.

LookupValue

Look up a claim value from a list of values based on the value of another claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputParameterId	string	The claim that contains the lookup value

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter		string	Collection of inputParameters.
InputParameter	errorOnFailedLookup	boolean	Controlling whether an error is returned when no matching lookup.
OutputClaim	inputParameterId	string	The ClaimTypes that will be produced after this claims transformation has been invoked. The value of the matching Id .

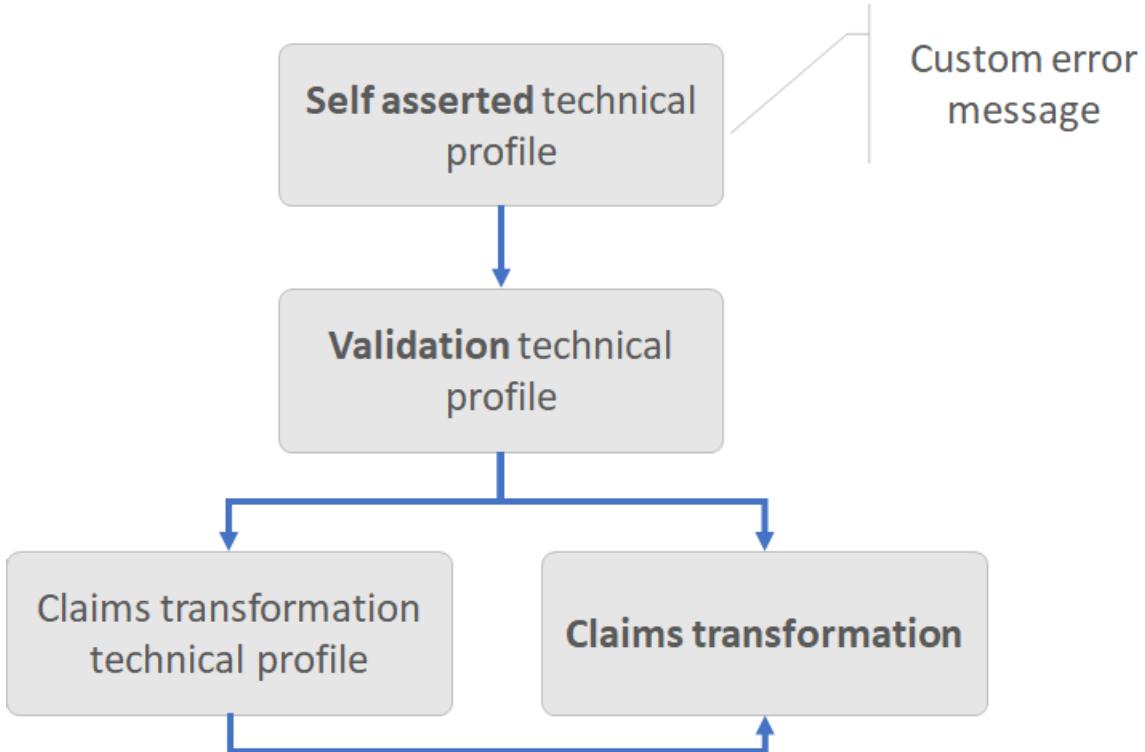
The following example looks up the domain name in one of the inputParameters collections. The claims transformation looks up the domain name in the identifier and returns its value (an application ID).

```
<ClaimsTransformation Id="DomainToClientId" TransformationMethod="LookupValue">
<InputClaims>
  <InputClaim ClaimTypeReferenceId="domainName" TransformationClaimType="inputParameterId" />
</InputClaims>
<InputParameters>
  <InputParameter Id="contoso.com" DataType="string" Value="13c15f79-8fb1-4e29-a6c9-be0d36ff19f1" />
  <InputParameter Id="microsoft.com" DataType="string" Value="0213308f-17cb-4398-b97e-01da7bd4804e" />
  <InputParameter Id="test.com" DataType="string" Value="c7026f88-4299-4cdb-965d-3f166464b8a9" />
  <InputParameter Id="errorOnFailedLookup" DataType="boolean" Value="false" />
</InputParameters>
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="domainAppId" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputParameterId:** test.com
- Input parameters:
 - **contoso.com:** 13c15f79-8fb1-4e29-a6c9-be0d36ff19f1
 - **microsoft.com:** 0213308f-17cb-4398-b97e-01da7bd4804e
 - **test.com:** c7026f88-4299-4cdb-965d-3f166464b8a9
 - **errorOnFailedLookup:** false
- Output claims:
 - **outputClaim:** c7026f88-4299-4cdb-965d-3f166464b8a9

When **errorOnFailedLookup** input parameter is set to **true**, the **LookupValue** claims transformation is always executed from a [validation technical profile](#) that is called by a [self-asserted technical profile](#), or a [DisplayControl](#). The **LookupNotFound** metadata of a self-asserted technical profile controls the error message that is presented to the user.



The following example looks up the domain name in one of the inputParameters collections. The claims transformation looks up the domain name in the identifier and returns its value (an application ID), or raises an error message.

```

<ClaimsTransformation Id="DomainToClientId" TransformationMethod="LookupValue">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="domainName" TransformationClaimType="inputParameterId" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="contoso.com" DataType="string" Value="13c15f79-8fb1-4e29-a6c9-be0d36ff19f1" />
    <InputParameter Id="microsoft.com" DataType="string" Value="0213308f-17cb-4398-b97e-01da7bd4804e" />
    <InputParameter Id="test.com" DataType="string" Value="c7026f88-4299-4cdb-965d-3f166464b8a9" />
    <InputParameter Id="errorOnFailedLookup" DataType="boolean" Value="true" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="domainAppId" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **inputParameterId:** live.com
- Input parameters:
 - **contoso.com:** 13c15f79-8fb1-4e29-a6c9-be0d36ff19f1
 - **microsoft.com:** 0213308f-17cb-4398-b97e-01da7bd4804e
 - **test.com:** c7026f88-4299-4cdb-965d-3f166464b8a9
 - **errorOnFailedLookup:** true
- Error:
 - No match found for the input claim value in the list of input parameter ids and errorOnFailedLookup is true.

NullClaim

Clean the value of a given claim.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	claim_to_null	string	The claim's value is set to NULL.

Use this claim transformation to remove unnecessary data from the claims property bag so the session cookie will be smaller. The following example removes the value of the `TermsOfService` claim type.

```
<ClaimsTransformation Id="SetTOSToNull" TransformationMethod="NullClaim">
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="TermsOfService" TransformationClaimType="claim_to_null" />
  </OutputClaims>
</ClaimsTransformation>
```

- Input claims:
 - **outputClaim:** Welcome to Contoso App. If you continue to browse and use this website, you are agreeing to comply with and be bound by the following terms and conditions...
- Output claims:
 - **outputClaim:** NULL

ParseDomain

Gets the domain portion of an email address.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	emailAddress	string	The ClaimType that contains the email address.
OutputClaim	domain	string	The ClaimType that is produced after this claims transformation has been invoked - the domain.

Use this claims transformation to parse the domain name after the @ symbol of the user. The following claims transformation demonstrates how to parse the domain name from an `email` claim.

```
<ClaimsTransformation Id="SetDomainName" TransformationMethod="ParseDomain">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="emailAddress" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="domainName" TransformationClaimType="domain" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **emailAddress:** joe@outlook.com
- Output claims:
 - **domain:** outlook.com

SetClaimsIfRegexMatch

Checks that a string claim `claimToMatch` and `matchTo` input parameter are equal, and sets the output claims with the value present in `outputClaimIfMatched` input parameter, along with compare result output claim, which is to be set as `true` or `false` based on the result of comparison.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
inputClaim	claimToMatch	string	The claim type, which is to be compared.
InputParameter	matchTo	string	The regular expression to match.
InputParameter	outputClaimIfMatched	string	The value to be set if strings are equal.
OutputClaim	outputClaim	string	If regular expression is match, this output claim contains the value of <code>outputClaimIfMatched</code> input parameter. Or null, if no match.
OutputClaim	regexCompareResultClaim	boolean	The regular expression match result output claim type, which is to be set as <code>true</code> or <code>false</code> based on the result of matching.

For example, checks whether the provided phone number is valid, based on phone number regular expression pattern.

```
<ClaimsTransformation Id="SetIsPhoneRegex" TransformationMethod="setClaimsIfRegexMatch">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="phone" TransformationClaimType="claimToMatch" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="matchTo" DataType="string" Value="^[0-9]{4,16}$" />
    <InputParameter Id="outputClaimIfMatched" DataType="string" Value="isPhone" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="validationResult" TransformationClaimType="outputClaim" />
    <OutputClaim ClaimTypeReferenceId="isPhoneBoolean" TransformationClaimType="regexCompareResultClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **claimToMatch:** "64854114520"
- Input parameters:
 - **matchTo:** "^[0-9]{4,16}\$"
 - **outputClaimIfMatched:** "isPhone"
- Output claims:
 - **outputClaim:** "isPhone"
 - **regexCompareResultClaim:** true

SetClaimsIfStringsAreEqual

Checks that a string claim and `matchTo` input parameter are equal, and sets the output claims with the value present in `stringMatchMsg` and `stringMatchMsgCode` input parameters, along with compare result output claim, which is to be set as `true` or `false` based on the result of comparison.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The claim type, which is to be compared.
InputParameter	matchTo	string	The string to be compared with <code>inputClaim</code> .
InputParameter	stringComparison	string	Possible values: <code>Ordinal</code> or <code>OrdinalIgnoreCase</code> .
InputParameter	stringMatchMsg	string	First value to be set if strings are equal.
InputParameter	stringMatchMsgCode	string	Second value to be set if strings are equal.
OutputClaim	outputClaim1	string	If strings are equals, this output claim contains the value of <code>stringMatchMsg</code> input parameter.
OutputClaim	outputClaim2	string	If strings are equals, this output claim contains the value of <code>stringMatchMsgCode</code> input parameter.
OutputClaim	stringCompareResultClaim	boolean	The compare result output claim type, which is to be set as <code>true</code> or <code>false</code> based on the result of comparison.

You can use this claims transformation to check if a claim is equal to value you specified. For example, the following claims transformation checks if the value of the **termsOfUseConsentVersion** claim is equal to `v1`. If yes, change the value to `v2`.

```

<ClaimsTransformation Id="CheckTheTOS" TransformationMethod="SetClaimsIfStringsAreEqual">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="termsOfUseConsentVersion" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="matchTo" DataType="string" Value="v1" />
    <InputParameter Id="stringComparison" DataType="string" Value="ordinalIgnoreCase" />
    <InputParameter Id="stringMatchMsg" DataType="string" Value="B2C_V1_90005" />
    <InputParameter Id="stringMatchMsgCode" DataType="string" Value="The TOS is upgraded to v2" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="termsOfUseConsentVersion" TransformationClaimType="outputClaim1" />
    <OutputClaim ClaimTypeReferenceId="termsOfUseConsentVersionUpgradeCode"
      TransformationClaimType="outputClaim2" />
    <OutputClaim ClaimTypeReferenceId="termsOfUseConsentVersionUpgradeResult"
      TransformationClaimType="stringCompareResultClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **inputClaim**: v1
- Input parameters:
 - **matchTo**: V1
 - **stringComparison**: ordinalIgnoreCase
 - **stringMatchMsg**: B2C_V1_90005
 - **stringMatchMsgCode**: The TOS is upgraded to v2
- Output claims:
 - **outputClaim1**: B2C_V1_90005
 - **outputClaim2**: The TOS is upgraded to v2
 - **stringCompareResultClaim**: true

SetClaimsIfStringsMatch

Checks that a string claim and `matchTo` input parameter are equal, and sets the output claims with the value present in `outputClaimIfMatched` input parameter, along with compare result output claim, which is to be set as `true` or `false` based on the result of comparison.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	claimToMatch	string	The claim type, which is to be compared.
InputParameter	matchTo	string	The string to be compared with inputClaim.
InputParameter	stringComparison	string	Possible values: <code>Ordinal</code> or <code>OrdinalIgnoreCase</code> .
InputParameter	outputClaimIfMatched	string	The value to be set if strings are equal.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	outputClaim	string	If strings are equals, this output claim contains the value of <code>outputClaimIfMatched</code> input parameter. Or null, if the strings aren't match.
OutputClaim	stringCompareResultClaim	boolean	The compare result output claim type, which is to be set as <code>true</code> or <code>false</code> based on the result of comparison.

For example, the following claims transformation checks if the value of **ageGroup** claim is equal to `Minor`. If yes, return the value to `B2C_V1_90001`.

```
<ClaimsTransformation Id="SetIsMinor" TransformationMethod="SetClaimsIfStringsMatch">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="ageGroup" TransformationClaimType="claimToMatch" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="matchTo" DataType="string" Value="Minor" />
    <InputParameter Id="stringComparison" DataType="string" Value="ordinalIgnoreCase" />
    <InputParameter Id="outputClaimIfMatched" DataType="string" Value="B2C_V1_90001" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isMinor" TransformationClaimType="outputClaim" />
    <OutputClaim ClaimTypeReferenceId="isMinorResponseCode" TransformationClaimType="stringCompareResultClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **claimToMatch**: Minor
- Input parameters:
 - **matchTo**: Minor
 - **stringComparison**: ordinalIgnoreCase
 - **outputClaimIfMatched**: B2C_V1_90001
- Output claims:
 - **isMinorResponseCode**: B2C_V1_90001
 - **isMinor**: true

StringContains

Determine whether a specified substring occurs within the input claim. The result is a new boolean ClaimType with a value of `true` or `false`. `true` if the value parameter occurs within this string, otherwise, `false`.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The claim type, which is to be searched.
InputParameter	contains	string	The value to search.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	ignoreCase	string	Specifies whether this comparison should ignore the case of the string being compared.
OutputClaim	outputClaim	string	The ClaimType that is produced after this ClaimsTransformation has been invoked. A boolean indicator if the substring occurs within the input claim.

Use this claims transformation to check if a string claim type contains a substring. Following example, checks whether the `roles` string claim type contains the value of **admin**.

```
<ClaimsTransformation Id="CheckIsAdmin" TransformationMethod="StringContains">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="roles" TransformationClaimType="inputClaim"/>
  </InputClaims>
  <InputParameters>
    <InputParameter Id="contains" DataType="string" Value="admin"/>
    <InputParameter Id="ignoreCase" DataType="string" Value="true"/>
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isAdmin" TransformationClaimType="outputClaim"/>
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim**: "Admin, Approver, Editor"
- Input parameters:
 - **contains**: "admin,"
 - **ignoreCase**: true
- Output claims:
 - **outputClaim**: true

StringSubstring

Extracts parts of a string claim type, beginning at the character at the specified position, and returns the specified number of characters.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The claim type, which contains the string.
InputParameter	startIndex	int	The zero-based starting character position of a substring in this instance.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputParameter	length	int	The number of characters in the substring.
OutputClaim	outputClaim	boolean	A string that is equivalent to the substring of length that begins at startIndex in this instance, or Empty if startIndex is equal to the length of this instance and length is zero.

For example, get the phone number country prefix.

```
<ClaimsTransformation Id="GetPhonePrefix" TransformationMethod="StringSubstring">
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="phoneNumber" TransformationClaimType="inputClaim" />
    </InputClaims>
    <InputParameters>
        <InputParameter Id="startIndex" DataType="int" Value="0" />
        <InputParameter Id="length" DataType="int" Value="2" />
    </InputParameters>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="phonePrefix" TransformationClaimType="outputClaim" />
    </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** "+1644114520"
- Input parameters:
 - **startIndex:** 0
 - **length:** 2
- Output claims:
 - **outputClaim:** "+1"

StringReplace

Searches a claim type string for a specified value, and returns a new claim type string in which all occurrences of a specified string in the current string are replaced with another specified string.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	The claim type, which contains the string.
InputParameter	oldValue	string	The string to be searched.
InputParameter	newValue	string	The string to replace all occurrences of <code>oldValue</code>

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
OutputClaim	outputClaim	boolean	A string that is equivalent to the current string except that all instances of oldValue are replaced with newValue. If oldValue is not found in the current instance, the method returns the current instance unchanged.

For example, normalize a phone number, by removing the `-` characters

```
<ClaimsTransformation Id="NormalizePhoneNumber" TransformationMethod="StringReplace">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="phoneNumber" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="oldValue" DataType="string" Value="-" />
    <InputParameter Id="newValue" DataType="string" Value="" />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="phoneNumber" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** "+164-411-452-054"
- Input parameters:
 - **oldValue:** "-"
 - **length:** ""
- Output claims:
 - **outputClaim:** "+164411452054"

StringJoin

Concatenates the elements of a specified string collection claim type, using the specified separator between each element or member.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	stringCollection	A collection that contains the strings to concatenate.
InputParameter	delimiter	string	The string to use as a separator, such as comma <code>,</code> .
OutputClaim	outputClaim	string	A string that consists of the members of the <code>inputClaim</code> string collection, delimited by the <code>delimiter</code> input parameter.

The following example takes a string collection of user roles, and converts it to a comma delimiter string. You can use this method to store a string collection in Azure AD user account. Later, when you read the account from the directory, use the `StringSplit` to convert the comma delimiter string back to string collection.

```
<ClaimsTransformation Id="ConvertRolesStringCollectionToCommaDelimiterString"
TransformationMethod="StringJoin">
<InputClaims>
<InputClaim ClaimTypeReferenceId="roles" TransformationClaimType="inputClaim" />
</InputClaims>
<InputParameters>
<InputParameter DataType="string" Id="delimiter" Value="," />
</InputParameters>
<OutputClaims>
<OutputClaim ClaimTypeReferenceId="rolesCommaDelimiterConverted" TransformationClaimType="outputClaim" />
</OutputClaims>
</ClaimsTransformation>
```

Example

- Input claims:
 - **inputClaim:** ["Admin", "Author", "Reader"]
- Input parameters:
 - **delimiter:** ","
- Output claims:
 - **outputClaim:** "Admin,Author,Reader"

StringSplit

Returns a string array that contains the substrings in this instance that are delimited by elements of a specified string.

ITEM	TRANSFORMATIONCLAIMTYPE	DATA TYPE	NOTES
InputClaim	inputClaim	string	A string claim type that contains the sub strings to split.
InputParameter	delimiter	string	The string to use as a separator, such as comma [,].
OutputClaim	outputClaim	stringCollection	A string collection whose elements contain the substrings in this string that are delimited by the <code>delimiter</code> input parameter.

The following example takes a comma delimiter string of user roles, and converts it to a string collection.

```

<ClaimsTransformation Id="ConvertRolesToStringCollection" TransformationMethod="StringSplit">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="rolesCommaDelimiter" TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter DataType="string" Id="delimiter" Value="," />
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="roles" TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>

```

Example

- Input claims:
 - **inputClaim**: "Admin,Author,Reader"
- Input parameters:
 - **delimiter**: ","
- Output claims:
 - **outputClaim**: ["Admin", "Author", "Reader"]

String claim transformations expressions

Claim transformations expressions in Azure AD B2C custom policies provide context information about the tenant ID and technical profile ID.

EXPRESSION	DESCRIPTION	EXAMPLE
{TechnicalProfileId}	The technical profileId name.	Facebook-OAUTH
{RelyingPartyTenantId}	The tenant ID of the relying party policy.	your-tenant.onmicrosoft.com
{TrustFrameworkTenantId}	The tenant ID of the trust framework.	your-tenant.onmicrosoft.com

Predicates and PredicateValidations

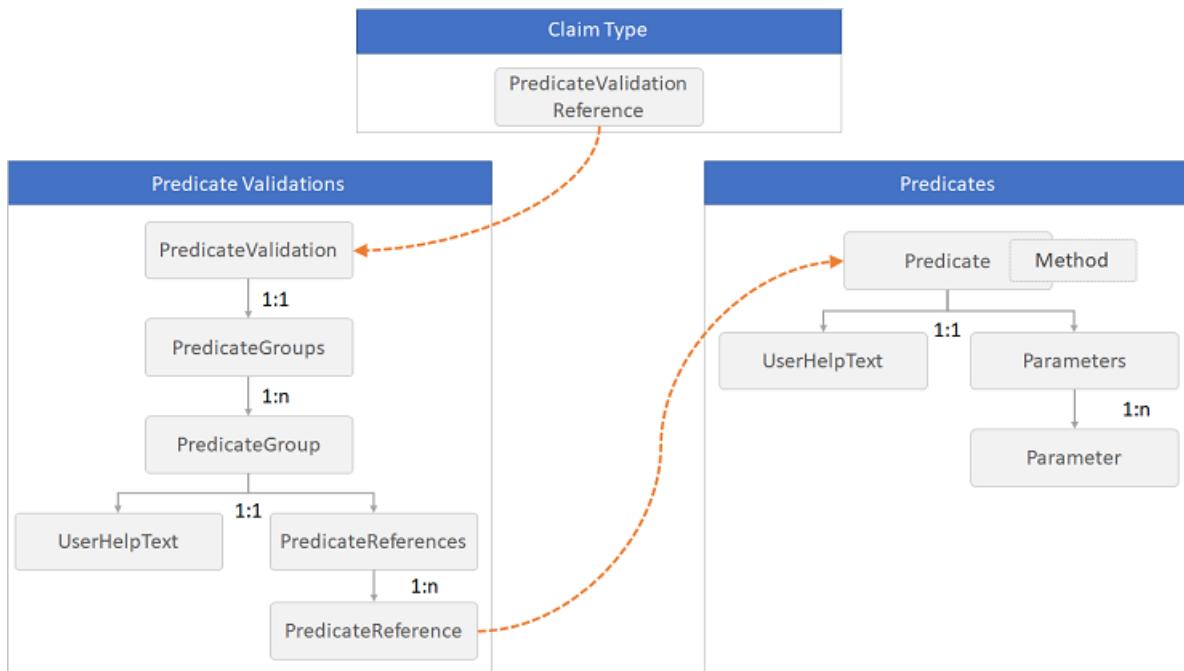
2/24/2020 • 7 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **Predicates** and **PredicateValidations** elements enable you to perform a validation process to ensure that only properly formed data is entered into your Azure Active Directory B2C (Azure AD B2C) tenant.

The following diagram shows the relationship between the elements:



Predicates

The **Predicate** element defines a basic validation to check the value of a claim type and returns `true` or `false`. The validation is done by using a specified **Method** element and a set of **Parameter** elements relevant to the method. For example, a predicate can check whether the length of a string claim value is within the range of minimum and maximum parameters specified, or whether a string claim value contains a character set. The **UserHelpText** element provides an error message for users if the check fails. The value of **UserHelpText** element can be localized using [language customization](#).

The **Predicates** element must appear directly following the **ClaimsSchema** element within the **BuildingBlocks** element.

The **Predicates** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
Predicate	1:n	A list of predicates.

The **Predicate** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the predicate. Other elements can use this identifier in the policy.
Method	Yes	The method type to use for validation. Possible values: IsLengthRange , MatchesRegex , IncludesCharacters , or IsDateRange . The IsLengthRange value checks whether the length of a string claim value is within the range of minimum and maximum parameters specified. The MatchesRegex value checks whether a string claim value matches a regular expression. The IncludesCharacters value checks whether a string claim value contains a character set. The IsDateRange value checks whether a date claim value is between a range of minimum and maximum parameters specified.
HelpText	No	An error message for users if the check fails. This string can be localized using the language customization

The **Predicate** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
UserHelpText	0:1	(Deprecated) An error message for users if the check fails.
Parameters	1:1	The parameters for the method type of the string validation.

The **Parameters** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Parameter	1:n	The parameters for the method type of the string validation.

The **Parameter** element contains the following attributes:

ELEMENT	OCCURRENCES	DESCRIPTION
Id	1:1	The identifier of the parameter.

The following example shows a **IsLengthRange** method with the parameters **Minimum** and **Maximum** that specify the length range of the string:

```

<Predicate Id="IsLengthBetween8And64" Method="IsLengthRange" HelpText="The password must be between 8 and 64 characters.">
  <Parameters>
    <Parameter Id="Minimum">8</Parameter>
    <Parameter Id="Maximum">64</Parameter>
  </Parameters>
</Predicate>

```

The following example shows a `MatchesRegex` method with the parameter `RegularExpression` that specifies a regular expression:

```

<Predicate Id="PIN" Method="MatchesRegex" HelpText="The password must be numbers only.">
  <Parameters>
    <Parameter Id="RegularExpression">^[0-9]+\$</Parameter>
  </Parameters>
</Predicate>

```

The following example shows a `IncludesCharacters` method with the parameter `CharacterSet` that specifies the set of characters:

```

<Predicate Id="Lowercase" Method="IncludesCharacters" HelpText="a lowercase letter">
  <Parameters>
    <Parameter Id="CharacterSet">a-z</Parameter>
  </Parameters>
</Predicate>

```

The following example shows a `IsDateRange` method with the parameters `Minimum` and `Maximum` that specify the date range with a format of `yyyy-MM-dd` and `Today`.

```

<Predicate Id="DateRange" Method="IsDateRange" HelpText="The date must be between 1970-01-01 and today.">
  <Parameters>
    <Parameter Id="Minimum">1970-01-01</Parameter>
    <Parameter Id="Maximum">Today</Parameter>
  </Parameters>
</Predicate>

```

PredicateValidations

While the predicates define the validation to check against a claim type, the **PredicateValidations** group a set of predicates to form a user input validation that can be applied to a claim type. Each **PredicateValidation** element contains a set of **PredicateGroup** elements that contain a set of **PredicateReference** elements that points to a **Predicate**. To pass the validation, the value of the claim should pass all of the tests of any predicate under all of the **PredicateGroup** with their set of **PredicateReference** elements.

The **PredicateValidations** element must appear directly following the **Predicates** element within the [BuildingBlocks](#) element.

```

<PredicateValidations>
  <PredicateValidation Id="">
    <PredicateGroups>
      <PredicateGroup Id="">
        <UserHelpText></UserHelpText>
        <PredicateReferences MatchAtLeast="">
          <PredicateReference Id="" />
          ...
        </PredicateReferences>
      </PredicateGroup>
      ...
    </PredicateGroups>
  </PredicateValidation>
  ...
</PredicateValidations>

```

The **PredicateValidations** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
PredicateValidation	1:n	A list of predicate validation.

The **PredicateValidation** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the predicate validation. The ClaimType element can use this identifier in the policy.

The **PredicateValidation** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
PredicateGroups	1:n	A list of predicate groups.

The **PredicateGroups** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
PredicateGroup	1:n	A list of predicates.

The **PredicateGroup** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the predicate group.

The **PredicateGroup** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION

ELEMENT	OCCURRENCES	DESCRIPTION
UserHelpText	0:1	A description of the predicate that can be helpful for users to know what value they should type.
PredicateReferences	1:n	A list of predicate references.

The **PredicateReferences** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
MatchAtLeast	No	Specifies that the value must match at least that many predicate definitions for the input to be accepted. If not specified, the value must match all predicate definitions.

The **PredicateReferences** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
PredicateReference	1:n	A reference to a predicate.

The **PredicateReference** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the predicate validation.

Configure password complexity

With **Predicates** and **PredicateValidationsInput** you can control the complexity requirements for passwords provided by a user when creating an account. By default, Azure AD B2C uses strong passwords. Azure AD B2C also supports configuration options to control the complexity of passwords that customers can use. You can define password complexity by using these predicate elements:

- **IsLengthBetween8And64** using the `IsLengthRange` method, validates that the password must be between 8 and 64 characters.
- **Lowercase** using the `IncludesCharacters` method, validates that the password contains a lowercase letter.
- **Uppercase** using the `IncludesCharacters` method, validates that the password contains an uppercase letter.
- **Number** using the `IncludesCharacters` method, validates that the password contains a digit.
- **Symbol** using the `IncludesCharacters` method, validates that the password contains one of several symbol characters.
- **PIN** using the `MatchesRegex` method, validates that the password contains numbers only.
- **AllowedAADCharacters** using the `MatchesRegex` method, validates that the password only invalid character was provided.
- **DisallowedWhitespace** using the `MatchesRegex` method, validates that the password doesn't begin or end with a whitespace character.

```

<Predicates>
  <Predicate Id="IsLengthBetween8And64" Method="IsLengthRange" HelpText="The password must be between 8 and 64 characters.">
    <Parameters>
      <Parameter Id="Minimum">8</Parameter>
      <Parameter Id="Maximum">64</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="Lowercase" Method="IncludesCharacters" HelpText="a lowercase letter">
    <Parameters>
      <Parameter Id="CharacterSet">a-z</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="Uppercase" Method="IncludesCharacters" HelpText="an uppercase letter">
    <Parameters>
      <Parameter Id="CharacterSet">A-Z</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="Number" Method="IncludesCharacters" HelpText="a digit">
    <Parameters>
      <Parameter Id="CharacterSet">0-9</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="Symbol" Method="IncludesCharacters" HelpText="a symbol">
    <Parameters>
      <Parameter Id="CharacterSet">@#$%^&;*-_+=[]{}|\\:'.?/`~()!;</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="PIN" Method="MatchesRegex" HelpText="The password must be numbers only.">
    <Parameters>
      <Parameter Id="RegularExpression">^[0-9]+$</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="AllowedAADCharacters" Method="MatchesRegex" HelpText="An invalid character was provided.">
    <Parameters>
      <Parameter Id="RegularExpression">(^([0-9A-Za-z\d@#$%^&;*-_+=[]{}|\\:'.?/`~()! ]|(\.(?!@)))+$)|(^$)</Parameter>
    </Parameters>
  </Predicate>

  <Predicate Id="DisallowedWhitespace" Method="MatchesRegex" HelpText="The password must not begin or end with a whitespace character.">
    <Parameters>
      <Parameter Id="RegularExpression">(^\$.*\$\$)|(^\$+\$)|(^$)</Parameter>
    </Parameters>
  </Predicate>

```

After you define the basic validations, you can combine them together and create a set of password policies that you can use in your policy:

- **SimplePassword** validates the DisallowedWhitespace, AllowedAADCharacters, and IsLengthBetween8And64
- **StrongPassword** validates the DisallowedWhitespace, AllowedAADCharacters, IsLengthBetween8And64. The last group `CharacterClasses` runs an additional set of predicates with `MatchAtLeast` set to 3. The user password must be between 8 and 16 characters, and three of the following characters: Lowercase, Uppercase, Number, or Symbol.
- **CustomPassword** validates only DisallowedWhitespace, AllowedAADCharacters. So, user can provide any password with any length, as long as the characters are valid.

```

<PredicateValidations>
  <PredicateValidation Id="SimplePassword">
    <PredicateGroups>
      <PredicateGroup Id="DisallowWhitespaceGroup">
        <PredicateReferences>
          <PredicateReference Id="DisallowWhitespace" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="AllowedAADCharactersGroup">
        <PredicateReferences>
          <PredicateReference Id="AllowedAADCharacters" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="LengthGroup">
        <PredicateReferences>
          <PredicateReference Id="IsLengthBetween8And64" />
        </PredicateReferences>
      </PredicateGroup>
    </PredicateGroups>
  </PredicateValidation>

  <PredicateValidation Id="StrongPassword">
    <PredicateGroups>
      <PredicateGroup Id="DisallowWhitespaceGroup">
        <PredicateReferences>
          <PredicateReference Id="DisallowWhitespace" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="AllowedAADCharactersGroup">
        <PredicateReferences>
          <PredicateReference Id="AllowedAADCharacters" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="LengthGroup">
        <PredicateReferences>
          <PredicateReference Id="IsLengthBetween8And64" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="CharacterClasses">
        <UserHelpText>The password must have at least 3 of the following:</UserHelpText>
        <PredicateReferences MatchAtLeast="3">
          <PredicateReference Id="Lowercase" />
          <PredicateReference Id="Uppercase" />
          <PredicateReference Id="Number" />
          <PredicateReference Id="Symbol" />
        </PredicateReferences>
      </PredicateGroup>
    </PredicateGroups>
  </PredicateValidation>

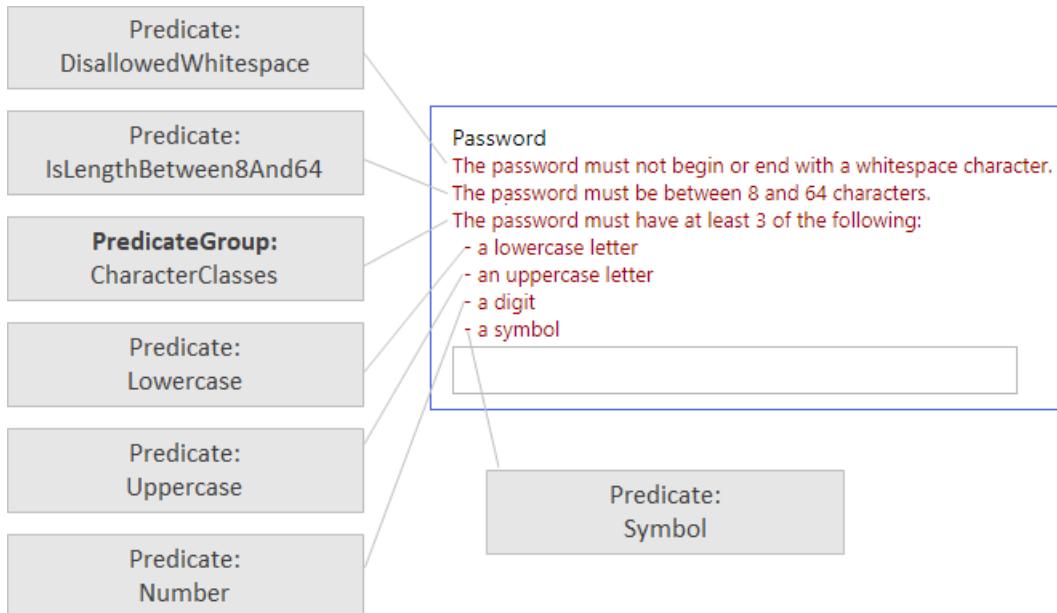
  <PredicateValidation Id="CustomPassword">
    <PredicateGroups>
      <PredicateGroup Id="DisallowWhitespaceGroup">
        <PredicateReferences>
          <PredicateReference Id="DisallowWhitespace" />
        </PredicateReferences>
      </PredicateGroup>
      <PredicateGroup Id="AllowedAADCharactersGroup">
        <PredicateReferences>
          <PredicateReference Id="AllowedAADCharacters" />
        </PredicateReferences>
      </PredicateGroup>
    </PredicateGroups>
  </PredicateValidation>
</PredicateValidations>

```

In your claim type, add the **PredicateValidationReference** element and specify the identifier as one of the predicate validations, such as SimplePassword, StrongPassword, or CustomPassword.

```
<ClaimType Id="password">
  <DisplayName>Password</DisplayName>
  <DataType>string</DataType>
  <AdminHelpText>Enter password</AdminHelpText>
  <UserHelpText>Enter password</UserHelpText>
  <UserInputType>Password</UserInputType>
  <PredicateValidationReference Id="StrongPassword" />
</ClaimType>
```

The following shows how the elements are organized when Azure AD B2C displays the error message:



Configure a date range

With the **Predicates** and **PredicateValidations** elements you can control the minimum and maximum date values of the **UserInputType** by using a **DateTimeDropdown**. To do this, create a **Predicate** with the **IsDateRange** method and provide the minimum and maximum parameters.

```
<Predicates>
  <Predicate Id="DateRange" Method="IsDateRange" HelpText="The date must be between 01-01-1980 and today.">
    <Parameters>
      <Parameter Id="Minimum">1980-01-01</Parameter>
      <Parameter Id="Maximum">Today</Parameter>
    </Parameters>
  </Predicate>
</Predicates>
```

Add a **PredicateValidation** with a reference to the **DateRange** predicate.

```
<PredicateValidations>
  <PredicateValidation Id="CustomDateRange">
    <PredicateGroups>
      <PredicateGroup Id="DateRangeGroup">
        <PredicateReferences>
          <PredicateReference Id="DateRange" />
        </PredicateReferences>
      </PredicateGroup>
    </PredicateGroups>
  </PredicateValidation>
</PredicateValidations>
```

In your claim type, add **PredicateValidationReference** element and specify the identifier as `CustomDateRange`.

```
<ClaimType Id="dateOfBirth">
  <DisplayName>Date of Birth</DisplayName>
  <DataType>date</DataType>
  <AdminHelpText>The user's date of birth.</AdminHelpText>
  <UserHelpText>Your date of birth.</UserHelpText>
  <UserInputType>DateTimeDropdown</UserInputType>
  <PredicateValidationReference Id="CustomDateRange" />
</ClaimType>
```

ContentDefinitions

2/17/2020 • 7 minutes to read [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

You can customize the look and feel of any [self-asserted technical profile](#). Azure Active Directory B2C (Azure AD B2C) runs code in your customer's browser and uses a modern approach called Cross-Origin Resource Sharing (CORS).

To customize the user interface, you specify a URL in the **ContentDefinition** element with customized HTML content. In the self-asserted technical profile or **OrchestrationStep**, you point to that content definition identifier. The content definition may contain a **LocalizedResourcesReferences** element that specifies a list of localized resources to load. Azure AD B2C merges user interface elements with the HTML content that's loaded from your URL and then displays the page to the user.

The **ContentDefinitions** element 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 by overriding the LoadUri for the HTML5 file. You can create new content definitions according to your needs. This element may contain a localized resources reference, to the localization identifier specified in the [Localization](#) element.

The following example shows the content definition identifier and the definition of localized resources:

```
<ContentDefinition Id="api.localaccounts signup">
  <LoadUri>/tenant/default/selfAsserted.cshtml</LoadUri>
  <RecoveryUri>/common/default_page_error.html</RecoveryUri>
  <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0</DataUri>
  <Metadata>
    <Item Key="DisplayName">Local account sign up page</Item>
  </Metadata>
  <LocalizedResourcesReferences MergeBehavior="Prepend">
    <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.localaccounts.signup.en" />
    <LocalizedResourcesReference Language="es" LocalizedResourcesReferenceId="api.localaccounts.signup.es" />
    ...
  </LocalizedResourcesReferences>
</ContentDefinition>
```

The metadata of the **LocalAccountSignUpWithLogonEmail** self-asserted technical profile contains the content definition identifier **ContentDefinitionReferenceId** set to `api.localaccounts.signup`

```
<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
  <DisplayName>Email signup</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="ContentDefinitionReferenceId">api.localaccounts.signup</Item>
    ...
  </Metadata>
  ...
</TechnicalProfile>
```

ContentDefinition

The **ContentDefinition** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier for a content definition. The value is one specified in the Content definition IDs section later in this page.

The **ContentDefinition** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
LoadUri	1:1	A string that contains the URL of the HTML5 page for the content definition.
RecoveryUri	1:1	A string that contains the URL of the HTML page for displaying an error relating to the content definition.
DataUri	1:1	A string that contains the relative URL of an HTML file that provides the user experience to invoke for the step.
Metadata	0:1	A collection of key/value pairs that contains the metadata utilized by the content definition.
LocalizedResourcesReferences	0:1	A collection of localized resources references. Use this element to customize the localization of a user interface and claims attribute.

DataUri

The **DataUri** element is used to specify the page identifier. Azure AD B2C uses the page identifier to load and initiate UI elements and client side JavaScript. The format of the value is `urn:com:microsoft:aad:b2c:elements:page-name:version`.

The following table lists the page identifiers you can use.

PAGE IDENTIFIER	DESCRIPTION
<code>globalexception</code>	Displays an error page when an exception or an error is encountered.
<code>providerselection</code> , <code>idpselection</code>	Lists the identity providers that users can choose from during sign-in.
<code>unifiedssp</code>	Displays a form for signing in with a local account that's based on an email address or a user name. This value also provides the "keep me sign-in functionality" and "Forgot your password?" link.
<code>unifiedssd</code>	Displays a form for signing in with a local account that's based on an email address or a user name.
<code>multipfactor</code>	Verifies phone numbers by using text or voice during sign-up or sign-in.
<code>selfasserted</code>	Displays a form to collect data from a user. For example, enables users to create or update their profile.

Select a page layout

You can enable [JavaScript client-side code](#) by inserting `contract` between `elements` and the page type. For example, `urn:com:microsoft:aad:b2c:elements:contract:page-name:version`.

NOTE

This feature is in public preview.

The **version** part of the **DataUri** specifies the package of content containing HTML, CSS, and JavaScript for the user interface elements in your policy. If you intend to enable JavaScript client-side code, the elements you base your JavaScript on must be immutable. If they're not immutable, any changes could cause unexpected behavior on your user pages. To prevent these issues, enforce the use of a page layout and specify a page layout version. Doing so ensures that all content definitions you've based your JavaScript on are immutable. Even if you don't intend to enable JavaScript, you still need to specify the page layout version for your pages.

The following example shows the **DataUri** of **selfasserted** version **1.2.0**:

```
<ContentDefinition Id="api.localaccountpasswordreset">
<LoadUri>/tenant/templates/AzureBlue/selfAsserted.cshtml</LoadUri>
<RecoveryUri>/common/default_page_error.html</RecoveryUri>
<DataUri>urn:com:microsoft:aad:b2c:elements:contract:selfasserted:1.2.0</DataUri>
<Metadata>
    <Item Key="DisplayName">Local account change password page</Item>
</Metadata>
</ContentDefinition>
```

Migrating to page layout

The format of the value must contain the word **contract** : *urn:com:microsoft:aad:b2c:elements:**contract**:page-name:version*. To specify a page layout in your custom policies that use an old **DataUri** value, use following table to migrate to the new format.

OLD DATAURI VALUE	NEW DATAURI VALUE
urn:com:microsoft:aad:b2c:elements:globalexception:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:globalexception:1.2.0
urn:com:microsoft:aad:b2c:elements:globalexception:1.1.0	urn:com:microsoft:aad:b2c:elements:contract:globalexception:1.2.0
urn:com:microsoft:aad:b2c:elements:idpselection:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:providerselection:1.2.0
urn:com:microsoft:aad:b2c:elements:multifactor:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:multifactor:1.2.0
urn:com:microsoft:aad:b2c:elements:multifactor:1.1.0	urn:com:microsoft:aad:b2c:elements:contract:multifactor:1.2.0
urn:com:microsoft:aad:b2c:elements:selfasserted:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:selfasserted:1.2.0
urn:com:microsoft:aad:b2c:elements:selfasserted:1.1.0	urn:com:microsoft:aad:b2c:elements:contract:selfasserted:1.2.0
urn:com:microsoft:aad:b2c:elements:unifiedssd:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:unifiedssd:1.2.0
urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0	urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:1.2.0
urn:com:microsoft:aad:b2c:elements:unifiedssp:1.1.0	urn:com:microsoft:aad:b2c:elements:contract:unifiedssp:1.2.0

Metadata

A **Metadata** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Item	0:n	The metadata that relates to the content definition.

The **Item** element of the **Metadata** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Key	Yes	The metadata key.

Metadata keys

Content definition supports following metadata items:

KEY	REQUIRED	DESCRIPTION
DisplayName	No	A string that contains the name of the content definition.

LocalizedResourcesReferences

The **LocalizedResourcesReferences** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
LocalizedResourcesReference	1:n	A list of localized resource references for the content definition.

The **LocalizedResourcesReference** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Language	Yes	A string that contains a supported language for the policy per RFC 5646 - Tags for Identifying Languages.
LocalizedResourcesReferenceld	Yes	The identifier of the LocalizedResources element.

The following example shows a sign-up or sign-in content definition with a reference to localization for English, French and Spanish:

```
<ContentDefinition Id="api.signuporsignin">
  <LoadUri>~/tenant/default/unified.cshtml</LoadUri>
  <RecoveryUri>~/common/default_page_error.html</RecoveryUri>
  <DataUri>urn:com:microsoft:aad:b2c:elements:unifiedssp:1.0.0</DataUri>
  <Metadata>
    <Item Key="DisplayName">Signin and Signup</Item>
  </Metadata>
  <LocalizedResourcesReferences MergeBehavior="Prepend">
    <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.signuporsignin.en" />
    <LocalizedResourcesReference Language="fr" LocalizedResourcesReferenceId="api.signuporsignin.rf" />
    <LocalizedResourcesReference Language="es" LocalizedResourcesReferenceId="api.signuporsignin.es" />
  </LocalizedResourcesReferences>
</ContentDefinition>
```

To learn how to add localization support to your content definitions, see [Localization](#).

Content definition IDs

The ID attribute of the **ContentDefinition** element specifies the type of page that relates to the content definition. The element defines the context that a custom HTML5/CSS template is going to apply. The following table describes the set of content definition IDs that is recognized by the Identity Experience Framework, and the page types that relate to them. You can create your own content definitions with an arbitrary ID.

ID	DEFAULT TEMPLATE	DESCRIPTION
api.error	exception.cshtml	Error page - Displays an error page when an exception or an error is encountered.
api.idpselections	idpSelector.cshtml	Identity provider selection page - Lists identity providers that users can choose from during sign-in. The options are usually enterprise identity providers, social identity providers such as Facebook and Google+, or local accounts.
api.idpselections.signup	idpSelector.cshtml	Identity provider selection for sign-up - Lists identity providers that users can choose from during sign-up. The options are usually enterprise identity providers, social identity providers such as Facebook and Google+, or local accounts.
api.localaccountpasswordreset	selfasserted.cshtml	Forgot password page - Displays a form that users must complete to initiate a password reset.
api.localaccountsigin	selfasserted.cshtml	Local account sign-in page - Displays a form for signing in with a local account that's based on an email address or a user name. The form can contain a text input box and password entry box.
api.localaccountsigup	selfasserted.cshtml	Local account sign-up page - Displays a form for signing up for a local account that's based on an email address or a user name. The form can contain various input controls, such as: a text input box, a password entry box, a radio button, single-select drop-down boxes, and multi-select check boxes.
api.phonefactor	multifactor-1.0.0.cshtml	Multi-factor authentication page - Verifies phone numbers, by using text or voice, during sign-up or sign-in.
api.selfasserted	selfasserted.cshtml	Social account sign-up page - Displays a form that users must complete when they sign up by using an existing account from a social identity provider. This page is similar to the preceding social account sign up page, except for the password entry fields.
api.selfasserted.profileupdate	updateprofile.cshtml	Profile update page - Displays a form that users can access to update their profile. This page is similar to the social account sign up page, except for the password entry fields.

ID	DEFAULT TEMPLATE	DESCRIPTION
api.signuporsignin	unified.cshtml	Unified sign-up or sign-in page - Handles the user sign-up and sign-in process. Users can use enterprise identity providers, social identity providers such as Facebook or Google+, or local accounts.

Next steps

For an example of customizing the user interface by using content definitions, see:

[Customize the user interface of your application using a custom policy](#)

Localization

8/26/2019 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **Localization** element allows you to support multiple locales or languages in the policy for the user journeys. The localization support in policies allows you to:

- Set up the explicit list of the supported languages in a policy and pick a default language.
- Provide language-specific strings and collections.

```
<Localization Enabled="true">
  <SupportedLanguages DefaultLanguage="en" MergeBehavior="ReplaceAll">
    <SupportedLanguage>en</SupportedLanguage>
    <SupportedLanguage>es</SupportedLanguage>
  </SupportedLanguages>
  <LocalizedResources Id="api.localaccounts signup.en">
    <LocalizedResources Id="api.localaccounts signup.es">
      ...
    </LocalizedResources>
  </LocalizedResources>
</Localization>
```

The **Localization** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Enabled	No	Possible values: <code>true</code> or <code>false</code> .

The **Localization** element contains following XML elements

ELEMENT	OCCURRENCES	DESCRIPTION
SupportedLanguages	1:n	List of supported languages.
LocalizedResources	0:n	List of localized resources.

SupportedLanguages

The **SupportedLanguages** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
DefaultLanguage	Yes	The language to be used as the default for localized resources.

ATTRIBUTE	REQUIRED	DESCRIPTION
MergeBehavior	No	An enumeration values of values that are merged together with any ClaimType present in a parent policy with the same identifier. Use this attribute when you overwrite a claim specified in base policy. Possible values: <code>Append</code> , <code>Prepend</code> , or <code>ReplaceAll</code> . The <code>Append</code> value specifies that the collection of data present should be appended to the end of the collection specified in the parent policy. The <code>Prepend</code> value specifies that the collection of data present should be added before the collection specified in the parent policy. The <code>ReplaceAll</code> value specifies that the collection of data defined in the parent policy should be ignored, using instead the data defined in the current policy.

SupportedLanguages

The **SupportedLanguages** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
SupportedLanguage	1:n	Displays content that conforms to a language tag per RFC 5646 - Tags for Identifying Languages.

LocalizedResources

The **LocalizedResources** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that is used to uniquely identify localized resources.

The **LocalizedResources** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
LocalizedCollections	0:n	Defines entire collections in various cultures. A collection can have different number of items and different strings for various cultures. Examples of collections include the enumerations that appear in claim types. For example, a country/region list is shown to the user in a drop-down list.
LocalizedStrings	0:n	Defines all of the strings, except those strings that appear in collections, in various cultures.

LocalizedCollections

The **LocalizedCollections** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
LocalizedCollection	1:n	List of supported languages.

LocalizedCollection

The **LocalizedCollection** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ElementType	Yes	References a ClaimType element or a user interface element in the policy file.
ElementId	Yes	A string that contains a reference to a claim type already defined in the ClaimsSchema section that is used if ElementType is set to a ClaimType.
TargetCollection	Yes	The target collection.

The **LocalizedCollection** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Item	0:n	Defines an available option for the user to select for a claim in the user interface, such as a value in a dropdown.

The **Item** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Text	Yes	The user-friendly display string that should be shown to the user in the user interface for this option.
Value	Yes	The string claim value associated with selecting this option.
SelectByDefault	No	Indicates whether or not this option should be selected by default in the UI. Possible values: True or False.

The following example shows the use of the **LocalizedCollections** element. It contains two **LocalizedCollection** elements, one for English and another one for Spanish. Both set the **Restriction** collection of the claim `Gender` with a list of items for English and Spanish.

```

<LocalizedResources Id="api.selfasserted.en">
  <LocalizedCollections>
    <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
      <Item Text="Female" Value="F" />
      <Item Text="Male" Value="M" />
    </LocalizedCollection>
  </LocalizedCollections>

  <LocalizedResources Id="api.selfasserted.es">
    <LocalizedCollections>
      <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
        <Item Text="Femenino" Value="F" />
        <Item Text="Masculino" Value="M" />
      </LocalizedCollection>
    </LocalizedCollections>
  </LocalizedResources>

```

LocalizedStrings

The **LocalizedStrings** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
LocalizedString	1:n	A localized string.

The **LocalizedString** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ElementType	Yes	A reference to a claim type element or a user interface element in the policy. Possible values: <code>ClaimType</code> , <code>UxElement</code> , <code>ErrorMessage</code> , <code>Predicate</code> , or . The <code>ClaimType</code> value is used to localize one of the claim attributes, as specified in the StringId. The <code>UxElement</code> value is used to localize one of the user interface elements as specified in the StringId. The <code>ErrorMessage</code> value is used to localize one of the system error messages as specified in the StringId. The <code>Predicate</code> value is used to localize one of the <code>Predicate</code> error messages, as specified in the StringId. The <code>InputValidation</code> value is used to localize one of the <code>PredicateValidation</code> group error messages as specified in the StringId.
ElementId	Yes	If <code>ElementType</code> is set to <code>ClaimType</code> , <code>Predicate</code> , or <code>InputValidation</code> , this element contains a reference to a claim type already defined in the ClaimsSchema section.

ATTRIBUTE	REQUIRED	DESCRIPTION
StringId	Yes	If ElementType is set to <code>ClaimType</code> , this element contains a reference to an attribute of a claim type. Possible values: <code>DisplayName</code> , <code>AdminHelpText</code> , or <code>PatternHelpText</code> . The <code>DisplayName</code> value is used to set the claim display name. The <code>AdminHelpText</code> value is used to set the help text name of the claim user. The <code>PatternHelpText</code> value is used to set the claim pattern help text. If ElementType is set to <code>UxElement</code> , this element contains a reference to an attribute of a user interface element. If ElementType is set to <code>ErrorMessage</code> , this element specifies the identifier of an error message. See Localization string IDs for a complete list of the <code>UxElement</code> identifiers.

The following example shows a localized sign-up page. The first three **LocalizedString** values set the claim attribute. The third changes the value of the continue button. The last one changes the error message.

```
<LocalizedResources Id="api.selfasserted.en">
  <LocalizedStrings>
    <LocalizedString ElementType="ClaimType" ElementId="email"
      StringId="DisplayName">Email</LocalizedString>
    <LocalizedString ElementType="ClaimType" ElementId="email" StringId="UserHelpText">Please enter your
      email</LocalizedString>
    <LocalizedString ElementType="ClaimType" ElementId="email" StringId="PatternHelpText">Please enter a
      valid email address</LocalizedString>
    <LocalizedString ElementType="UxElement" StringId="button_continue">Create new account</LocalizedString>
    <LocalizedString ElementType="ErrorMessage" StringId="UserMessageIfClaimsPrincipalAlreadyExists">The
      account you are trying to create already exists, please sign-in.</LocalizedString>
  </LocalizedStrings>
</LocalizedResources>
```

The following example shows a localized the **UserHelpText** of **Predicate** with Id `IsLengthBetween8And64`. And a localized **UserHelpText** of **PredicateGroup** with Id `CharacterClasses` of **PredicateValidation** with Id `StrongPassword`.

```

<PredicateValidation Id="StrongPassword">
  <PredicateGroups>
    ...
    <PredicateGroup Id="CharacterClasses">
      ...
    </PredicateGroup>
  </PredicateGroups>
</PredicateValidation>

...
<Predicate Id="IsLengthBetween8And64" Method="IsLengthRange">
  ...
</Predicate>
...

<LocalizedString ElementType="InputValidation" ElementId="StrongPassword" StringId="CharacterClasses">The
password must have at least 3 of the following:</LocalizedString>

<LocalizedString ElementType="Predicate" ElementId="IsLengthBetween8And64" StringId="HelpText">The password
must be between 8 and 64 characters.</LocalizedString>

```

Set up localization

This article shows you how to support multiple locales or languages in the policy for user journeys. Localization requires three steps: set-up the explicit list of the supported languages, provide language-specific strings and collections, and edit the ContentDefinition for the page.

Set up the explicit list of supported languages

Under the **BuildingBlocks** element, add the **Localization** element with the list of supported languages. The following example shows how to define the localization support for both English (default) and Spanish:

```

<Localization Enabled="true">
  <SupportedLanguages DefaultLanguage="en" MergeBehavior="ReplaceAll">
    <SupportedLanguage>en</SupportedLanguage>
    <SupportedLanguage>es</SupportedLanguage>
  </SupportedLanguages>
</Localization>

```

Provide language-specific strings and collections

Add **LocalizedResources** elements inside the **Localization** element after the close of the **SupportedLanguages** element. You add **LocalizedResources** elements for each page (content definition) and any language you want to support. To customize the unified sign-up or sign-in page, sign-up and multi-factor authentication (MFA) pages for English, Spanish, and France, you add the following **LocalizedResources** elements.

- Unified sign-up or sign-in page, English <LocalizedResources Id="api.signuporsignin.en">
- Unified sign-up or sign-in page, Spanish <LocalizedResources Id="api.signuporsignin.es">
- Unified sign-up or sign-in page, France <LocalizedResources Id="api.signuporsignin.fr">
- Sign-Up, English <LocalizedResources Id="api.localaccounts signup.en">
- Sign-Up, Spanish <LocalizedResources Id="api.localaccounts signup.es">
- Sign-Up, France <LocalizedResources Id="api.localaccounts signup.fr">
- MFA, English <LocalizedResources Id="api.phonefactor.en">
- MFA, Spanish <LocalizedResources Id="api.phonefactor.es">
- MFA, France <LocalizedResources Id="api.phonefactor.fr">

Each **LocalizedResources** element contains all of the required **LocalizedStrings** elements with multiple **LocalizedString** elements and **LocalizedCollections** elements with multiple **LocalizedCollection** elements.

The following example adds the sign-up page English localization:

Note: This example makes a reference to `Gender` and `City` claim types. To use this example, make sure you define those claims. For more information, see [ClaimsSchema](#).

```
<LocalizedResources Id="api.localaccounts.signup.en">

    <LocalizedCollections>
        <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
            <Item Text="Female" Value="F" />
            <Item Text="Male" Value="M" />
        </LocalizedCollection>
        <LocalizedCollection ElementType="ClaimType" ElementId="City" TargetCollection="Restriction">
            <Item Text="New York" Value="NY" />
            <Item Text="Paris" Value="Paris" />
            <Item Text="London" Value="London" />
        </LocalizedCollection>
    </LocalizedCollections>

    <LocalizedStrings>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="DisplayName">Email</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="UserHelpText">Please enter your email</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="PatternHelpText">Please enter a valid email address</LocalizedString>
        <LocalizedString ElementType="UxElement" StringId="button_continue">Create new account</LocalizedString>
        <LocalizedString ElementType="ErrorMessage" StringId="UserMessageIfClaimsPrincipalAlreadyExists">The account you are trying to create already exists, please sign-in.</LocalizedString>
    </LocalizedStrings>
</LocalizedResources>
```

The sign-up page localization for Spanish.

```
<LocalizedResources Id="api.localaccounts.signup.es">

    <LocalizedCollections>
        <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
            <Item Text="Femenino" Value="F" />
            <Item Text="Masculino" Value="M" />
        </LocalizedCollection>
        <LocalizedCollection ElementType="ClaimType" ElementId="City" TargetCollection="Restriction">
            <Item Text="Nueva York" Value="NY" />
            <Item Text="París" Value="Paris" />
            <Item Text="Londres" Value="London" />
        </LocalizedCollection>
    </LocalizedCollections>

    <LocalizedStrings>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="DisplayName">Dirección de correo electrónico</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="UserHelpText">Dirección de correo electrónico que puede usarse para ponerse en contacto con usted.</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="PatternHelpText">Introduzca una dirección de correo electrónico.</LocalizedString>
        <LocalizedString ElementType="UxElement" StringId="button_continue">Crear</LocalizedString>
        <LocalizedString ElementType="ErrorMessage" StringId="UserMessageIfClaimsPrincipalAlreadyExists">Ya existe un usuario con el id. especificado. Elija otro diferente.</LocalizedString>
    </LocalizedStrings>
</LocalizedResources>
```

Edit the ContentDefinition for the page

For each page that you want to localize, specify the language codes to look for in the **ContentDefinition**.

In the following example, English (en) and Spanish (es) custom strings are added to the sign-up page. The **LocalizedResourcesReferenceId** for each **LocalizedResourcesReference** is the same as their locale, but you could use any string as the identifier. For each language and page combination, you point to the corresponding **LocalizedResources** you previously created.

```
<ContentDefinition Id="api.localaccounts signup">
...
<LocalizedResourcesReferences MergeBehavior="Prepend">
    <LocalizedResourcesReference Language="en" LocalizedResourcesReferenceId="api.localaccounts signup.en" />
    <LocalizedResourcesReference Language="es" LocalizedResourcesReferenceId="api.localaccounts signup.es" />
</LocalizedResourcesReferences>
</ContentDefinition>
```

The following example shows the final XML:

```
<BuildingBlocks>
<ContentDefinitions>
    <ContentDefinition Id="api.localaccounts signup">
        <!-- Other content definitions elements... -->
        <LocalizedResourcesReferences MergeBehavior="Prepend">
            <LocalizedResourcesReference Language="en"
LocalizedResourcesReferenceId="api.localaccounts signup.en" />
            <LocalizedResourcesReference Language="es"
LocalizedResourcesReferenceId="api.localaccounts signup.es" />
        </LocalizedResourcesReferences>
    </ContentDefinition>
    <!-- More content definitions... -->
</ContentDefinitions>

<Localization Enabled="true">

    <SupportedLanguages DefaultLanguage="en" MergeBehavior="ReplaceAll">
        <SupportedLanguage>en</SupportedLanguage>
        <SupportedLanguage>es</SupportedLanguage>
        <!-- More supported language... -->
    </SupportedLanguages>

    <LocalizedResources Id="api.localaccounts signup.en">
        <LocalizedCollections>
            <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
                <Item Text="Female" Value="F" />
                <Item Text="Male" Value="M" />
                <!-- More items... -->
            </LocalizedCollection>
            <LocalizedCollection ElementType="ClaimType" ElementId="City" TargetCollection="Restriction">
                <Item Text="New York" Value="NY" />
                <Item Text="Paris" Value="Paris" />
                <Item Text="London" Value="London" />
            </LocalizedCollection>
            <!-- More localized collections... -->
        </LocalizedCollections>
        <LocalizedStrings>
            <LocalizedString ElementType="ClaimType" ElementId="email"
StringId="DisplayName">Email</LocalizedString>
            <LocalizedString ElementType="ClaimType" ElementId="email" StringId="UserHelpText">Please enter your
email</LocalizedString>
            <LocalizedString ElementType="ClaimType" ElementId="email" StringId="PatternHelpText">Please enter a
valid email address</LocalizedString>
            <LocalizedString ElementType="UxElement" StringId="button_continue">Create new
account</LocalizedString>
            <LocalizedString ElementType="ErrorMessage" StringId="UserMessageIfClaimsPrincipalAlreadyExists">The
account you are trying to create already exists. Please sign in.</LocalizedString>
        </LocalizedStrings>
    </LocalizedResources>
</Localization>
```

```
account you are trying to create already exists, please sign-in.</LOCALIZEDSTRING>
    <!-- More localized strings... -->
</LocalizedStrings>
</LocalizedResources>

<LocalizedResources Id="api.localaccounts.signup.es">
    <LocalizedCollections>
        <LocalizedCollection ElementType="ClaimType" ElementId="Gender" TargetCollection="Restriction">
            <Item Text="Femenino" Value="F" />
            <Item Text="Masculino" Value="M" />
        </LocalizedCollection>
        <LocalizedCollection ElementType="ClaimType" ElementId="City" TargetCollection="Restriction">
            <Item Text="Nueva York" Value="NY" />
            <Item Text="París" Value="Paris" />
            <Item Text="Londres" Value="London" />
        </LocalizedCollection>
    </LocalizedCollections>
    <LocalizedStrings>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="DisplayName">Dirección de
correo electrónico</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="UserHelpText">Dirección de
correo electrónico que puede usarse para ponerse en contacto con usted.</LocalizedString>
        <LocalizedString ElementType="ClaimType" ElementId="email" StringId="PatternHelpText">Introduzca una
dirección de correo electrónico.</LocalizedString>
        <LocalizedString ElementType="UxElement" StringId="button_continue">Crear</LocalizedString>
        <LocalizedString ElementType="ErrorMessage" StringId="UserMessageIfClaimsPrincipalAlreadyExists">Ya
existe un usuario con el id. especificado. Elija otro diferente.</LocalizedString>
    </LocalizedStrings>
</LocalizedResources>
<!-- More localized resources... -->
</Localization>
</BuildingBlocks>
```

Localization string IDs

2/4/2020 • 6 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **Localization** element enables you to support multiple locales or languages in the policy for the user journeys. This article provides the list of localization IDs that you can use in your policy. To get familiar with UI localization, see [Localization](#).

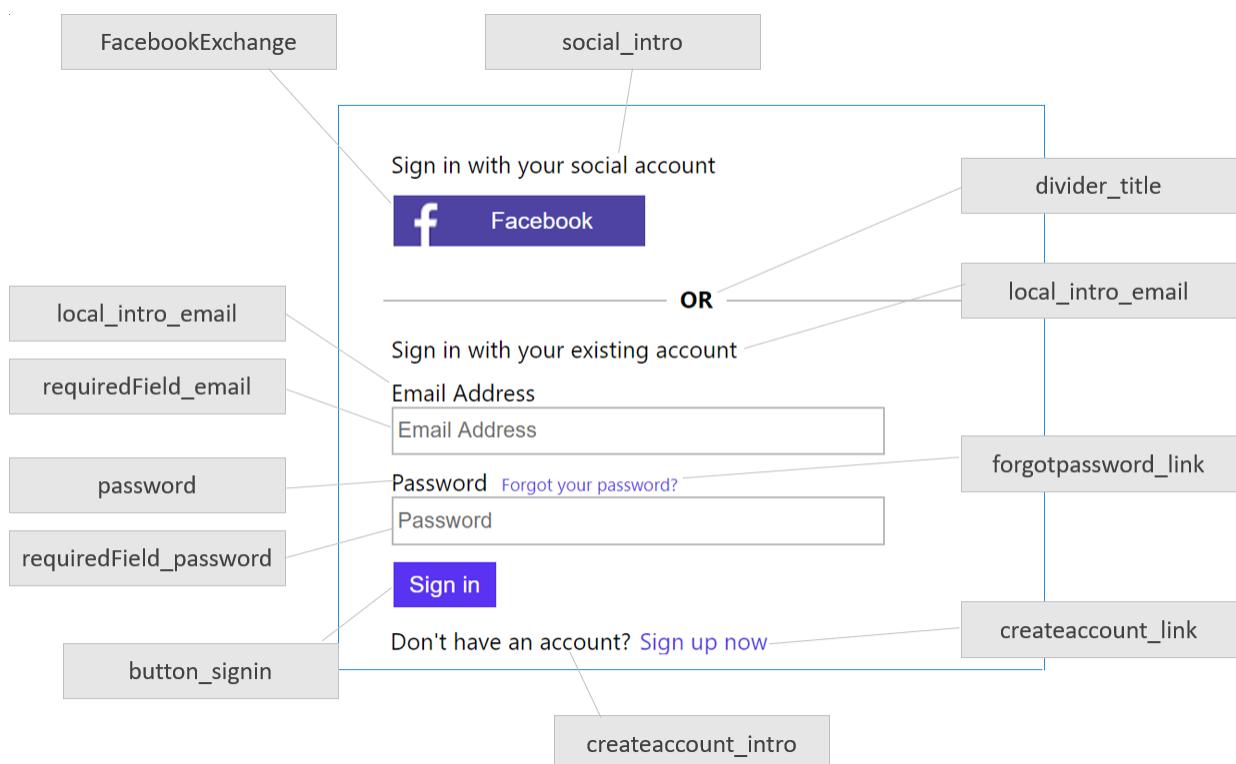
Sign-up or sign-in page elements

The following IDs are used for a content definition with an ID of `api.signuporsignin`.

ID	DEFAULT VALUE
local_intro_email	Sign in with your existing account
logonIdentifier_email	Email Address
requiredField_email	Please enter your email
invalid_email	Please enter a valid email address
email_pattern	<code>^[a-zA-Z0-9.!#\$%&'^+/?^_`{}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)\$</code>
local_intro_username	Sign in with your user name
logonIdentifier_username	Username
requiredField_username	Please enter your user name
password	Password
requiredField_password	Please enter your password
invalid_password	The password you entered is not in the expected format.
forgotpassword_link	Forgot your password?
createaccount_intro	Don't have an account?
createaccount_link	Sign up now
divider_title	OR

ID	DEFAULT VALUE
cancel_message	The user has forgotten their password
button_signin	Sign in
social_intro	Sign in with your social account
remember_me	Keep me signed in
unknown_error	We are having trouble signing you in. Please try again later.

The following example shows the use of some of the user interface elements in the sign-up or sign-in page:



The ID of the identity providers is configured in the user journey **ClaimsExchange** element. To localize the title of the identity provider, the **ElementType** is set to `ClaimsProvider`, while the **StringId** is set to the ID of the `ClaimsExchange`.

```

<OrchestrationStep Order="2" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
      <Value>objectId</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
    <ClaimsExchange Id="MicrosoftExchange" TechnicalProfileReferenceId="MSA-OIDC" />
    <ClaimsExchange Id="GoogleExchange" TechnicalProfileReferenceId="Google-OAUTH" />
    <ClaimsExchange Id="SignUpWithLogonEmailExchange" TechnicalProfileReferenceId="LocalAccount" />
  </ClaimsExchanges>
</OrchestrationStep>

```

The following example localizes the Facebook identity provider to Arabic:

```
<LocalizedString ElementType="ClaimsProvider" StringId="FacebookExchange">فیس بوک</LocalizedString>
```

Sign-up or sign-in error messages

ID	DEFAULT VALUE
UserMessageIfInvalidPassword	Your password is incorrect.
UserMessageIfClaimsPrincipalDoesNotExist	We can't seem to find your account.
UserMessageIfOldPasswordUsed	Looks like you used an old password.
DefaultMessage	Invalid username or password.
UserMessageIfUserAccountDisabled	Your account has been locked. Contact your support person to unlock it, then try again.
UserMessageIfUserAccountLocked	Your account is temporarily locked to prevent unauthorized use. Try again later.
AADRequestsThrottled	There are too many requests at this moment. Please wait for some time and try again.

Sign-up and self asserted pages user interface elements

The following are the IDs for a content definition with an ID of `api.localaccounts.signup` or any content definition that starts with `api.selfasserted`, such as `api.selfasserted.profileupdate` and `api.localaccountpasswordreset`.

ID	DEFAULT VALUE
ver_sent	Verification code has been sent to:
ver_but_default	Default
cancel_message	The user has canceled entering self-asserted information
preloader_alt	Please wait
ver_but_send	Send verification code
alert_yes	Yes
error_fieldIncorrect	One or more fields are filled out incorrectly. Please check your entries and try again.
year	Year
verifying_blurb	Please wait while we process your information.
button_cancel	Cancel

ID	DEFAULT VALUE
ver_fail_no_retry	You've made too many incorrect attempts. Please try again later.
month	Month
ver_success_msg	E-mail address verified. You can now continue.
months	January, February, March, April, May, June, July, August, September, October, November, December
ver_fail_server	We are having trouble verifying your email address. Please enter a valid email address and try again.
error_requiredFieldMissing	A required field is missing. Please fill out all required fields and try again.
initial_intro	Please provide the following details.
ver_but_resend	Send new code
button_continue	Create
error_passwordEntryMismatch	The password entry fields do not match. Please enter the same password in both fields and try again.
ver_incorrect_format	Incorrect format.
ver_but_edit	Change e-mail
ver_but_verify	Verify code
alert_no	No
ver_info_msg	Verification code has been sent to your inbox. Please copy it to the input box below.
day	Day
ver_fail_throttled	There have been too many requests to verify this email address. Please wait a while, then try again.
helplink_text	What is this?
ver_fail_retry	That code is incorrect. Please try again.
alert_title	Cancel Entering Your Details
required_field	This information is required.
alert_message	Are you sure that you want to cancel entering your details?

ID	DEFAULT VALUE
ver_intro_msg	Verification is necessary. Please click Send button.
ver_input	Verification code

Sign-up and self asserted pages error messages

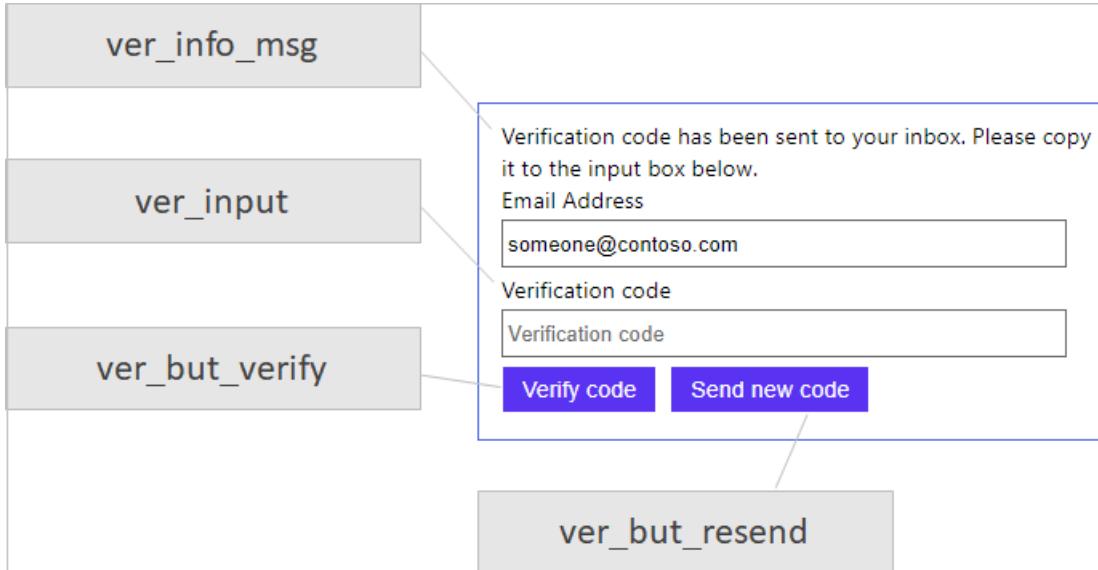
ID	DEFAULT VALUE
UserMessageIfClaimsPrincipalAlreadyExists	A user with the specified ID already exists. Please choose a different one.
UserMessageIfClaimNotVerified	Claim not verified: {0}
UserMessageIfIncorrectPattern	Incorrect pattern for: {0}
UserMessageIfMissingRequiredElement	Missing required element: {0}
UserMessageIfValidationError	Error in validation by: {0}
UserMessageIfInvalidInput	{0} has invalid input.
ServiceThrottled	There are too many requests at this moment. Please wait for some time and try again.

The following example shows the use of some of the user interface elements in the sign-up page:

ver_intro_msg	Verification is necessary. Please click Send button.
ver_but_send	Email Address Send verification code
day	Given Name Surname Date Of Birth Day ▾ Month ▾ Year ▾
button_continue	Create Cancel
button_cancel	month months year

The following example shows the use of some of the user interface elements in the sign-up page, after user clicks

on send verification code button:



Phone factor authentication page user interface elements

The Following are the IDs for a content definition with an ID of `api.phonefactor`.

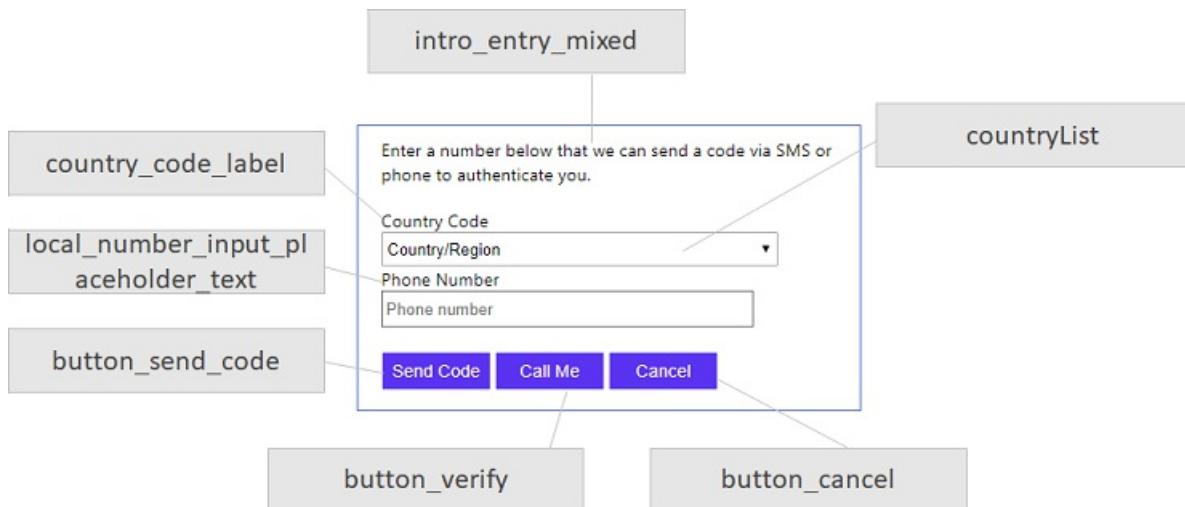
ID	DEFAULT VALUE
button_verify	Call Me
country_code_label	Country Code
cancel_message	The user has canceled multi-factor authentication
text_button_send_second_code	send a new code
code_pattern	\d{6}
intro_mixed	We have the following number on record for you. We can send a code via SMS or phone to authenticate you.
intro_mixed_p	We have the following numbers on record for you. Choose a number that we can phone or send a code via SMS to authenticate you.
button_verify_code	Verify Code
requiredField_code	Please enter the verification code you received
invalid_code	Please enter the 6 digit code you received
button_cancel	Cancel
local_number_input_placeholder_text	Phone number
button_retry	Retry
alternative_text	I don't have my phone

ID	DEFAULT VALUE
intro_phone_p	We have the following numbers on record for you. Choose a number that we can phone to authenticate you.
intro_phone	We have the following number on record for you. We will phone to authenticate you.
enter_code_text_intro	Enter your verification code below, or
intro_entry_phone	Enter a number below that we can phone to authenticate you.
intro_entry_sms	Enter a number below that we can send a code via SMS to authenticate you.
button_send_code	Send Code
invalid_number	Please enter a valid phone number
intro_sms	We have the following number on record for you. We will send a code via SMS to authenticate you.
intro_entry_mixed	Enter a number below that we can send a code via SMS or phone to authenticate you.
number_pattern	^\+(?:[0-9][\x20-]?){6,14}[0-9]\$
intro_sms_p	We have the following numbers on record for you. Choose a number that we can send a code via SMS to authenticate you.
requiredField_countryCode	Please select your country code
requiredField_number	Please enter your phone number
country_code_input_placeholder_text	Country or region
number_label	Phone Number
error_tryagain	The phone number you provided is busy or unavailable. Please check the number and try again.
error_incorrect_code	The verification code you have entered does not match our records. Please try again, or request a new code.
countryList	{"DEFAULT":"Country/Region","AF":"Afghanistan","AX":"Åland Islands","AL":"Albania","DZ":"Algeria","AS":"American Samoa","AD":"Andorra","AO":"Angola","AI":"Anguilla","AQ":"Antarctica","AG":"Antigua and Barbuda","AR":"Argentina","AM":"Armenia","AW":"Aruba","AU":"Australia","AT":"Austria","AZ":"Azerbaijan","BS":"Bahamas","BH":"Bahrain","BD":"Bangladesh","BB":"Barbados","BY":"Belarus","BE":"Belgium","BZ":"Belize","BJ":"Benin","BM":"Bermuda","BT":"Bhutan","BO":"Bolivia","BQ":"Bonaire","BA":"Bosnia and Herzegovina","BW":"Botswana","BV":"Bouvet Island","BR":"Brazil","IO":"British Indian Ocean Territory","VG":"British Virgin Islands","BN":"Brunei","BG":"Bulgaria","BF":"Burkina

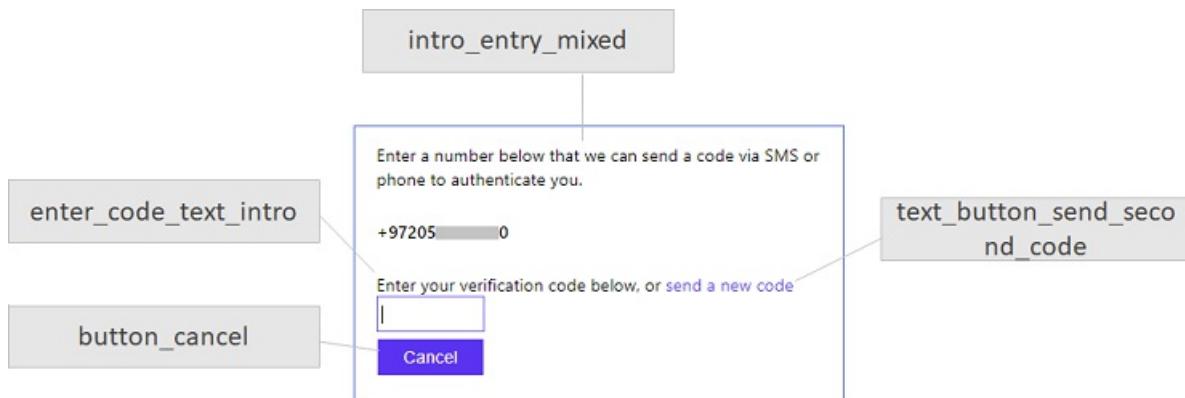
ID	DEFAULT VALUE
Easo","BI":"Burundi","CV":"Cabo Verde","KH":"Cambodia","CM":"Cameroon","CA":"Canada","KY":"Cayman Islands","CF":"Central African Republic","TD":"Chad","CL":"Chile","CN":"China","CX":"Christmas Island","CC":"Cocos (Keeling) Islands","CO":"Colombia","KM":"Comoros","CG":"Congo","CD":"Congo (DRC)","CK":"Cook Islands","CR":"Costa Rica","CI":"Côte d'Ivoire","HR":"Croatia","CU":"Cuba","CW":"Curaçao","CY":"Cyprus","CZ":"Czech Republic","DK":"Denmark","DJ":"Djibouti","DM":"Dominica","DO":"Dominican Republic","EC":"Ecuador","EG":"Egypt","SV":"El Salvador","GQ":"Equatorial Guinea","ER":"Eritrea","EE":"Estonia","ET":"Ethiopia","FK":"Falkland Islands","FO":"Faroe Islands","FJ":"Fiji","FI":"Finland","FR":"France","GF":"French Guiana","PF":"French Polynesia","TF":"French Southern Territories","GA":"Gabon","GM":"Gambia","GE":"Georgia","DE":"Germany","GH":"Ghana","GI":"Gibraltar","GR":"Greece","GL":"Greenland","GD":"Grenada","GP":"Guadeloupe","GU":"Guam","GT":"Guatemala","GG":"Guernsey","GN":"Guinea","GW":"Guinea-Bissau","GY":"Guyana","HT":"Haiti","HM":"Heard Island and McDonald Islands","HN":"Honduras","HK":"Hong Kong SAR","HU":"Hungary","IS":"Iceland","IN":"India","ID":"Indonesia","IR":"Iran","IQ":"Iraq","IE":"Ireland","IM":"Isle of Man","IL":"Israel","IT":"Italy","JM":"Jamaica","JP":"Japan","JE":"Jersey","JO":"Jordan","KZ":"Kazakhstan","KE":"Kenya","KI":"Kiribati","KR":"Korea","KW":"Kuwait","KG":"Kyrgyzstan","LA":"Laos","LV":"Latvia","LB":"Lebanon","LS":"Lesotho","LR":"Liberia","LY":"Libya","LI":"Liechtenstein","LT":"Lithuania","LU":"Luxembourg","MO":"Macao SAR","MK":"North Macedonia","MG":"Madagascar","MW":"Malawi","MY":"Malaysia","MV":"Maldives","ML":"Mali","MT":"Malta","MH":"Marshall Islands","MQ":"Martinique","MR":"Mauritania","MU":"Mauritius","YT":"Mayotte","MX":"Mexico","FM":"Micronesia","MD":"Moldova","MC":"Monaco","MN":"Mongolia","ME":"Montenegro","MS":"Montserrat","MA":"Morocco","MZ":"Mozambique","MM":"Myanmar","NA":"Namibia","NR":"Nauru","NP":"Nepal","NL":"Netherlands","NC":"New Caledonia","NZ":"New Zealand","NI":"Nicaragua","NE":"Niger","NG":"Nigeria","NU":"Niue","NF":"Norfolk Island","KP":"North Korea","MP":"Northern Mariana Islands","NO":"Norway","OM":"Oman","PK":"Pakistan","PW":"Palau","PS":"Palestinian Authority","PA":"Panama","PG":"Papua New Guinea","PY":"Paraguay","PE":"Peru","PH":"Philippines","PN":"Pitcairn Islands","PL":"Poland","PT":"Portugal","PR":"Puerto Rico","QA":"Qatar","RE":"Réunion","RO":"Romania","RU":"Russia","RW":"Rwanda","BL":"Saint Barthélemy","KN":"Saint Kitts and Nevis","LC":"Saint Lucia","MF":"Saint Martin","PM":"Saint Pierre and Miquelon","VC":"Saint Vincent and the Grenadines","WS":"Samoa","SM":"San Marino","ST":"São Tomé and Príncipe","SA":"Saudi Arabia","SN":"Senegal","RS":"Serbia","SC":"Seychelles","SL":"Sierra Leone","SG":"Singapore","SX":"Sint Maarten","SK":"Slovakia","SI":"Slovenia","SB":"Solomon Islands","SO":"Somalia","ZA":"South Africa","GS":"South Georgia and South Sandwich Islands","SS":"South Sudan","ES":"Spain","LK":"Sri Lanka","SH":"St Helena, Ascension, Tristan da Cunha","SD":"Sudan","SR":"Suriname","SJ":"Svalbard","SZ":"Swaziland","SE":"Sweden","CH":"Switzerland","SY":"Syria","TW":"Taiwan","TJ":"Tajikistan","TZ":"Tanzania","TH":"Thailand","TL":"Timor-Leste","TG":"Togo","TK":"Tokelau","TO":"Tonga","TT":"Trinidad and Tobago","TN":"Tunisia","TR":"Turkey","TM":"Turkmenistan","TC":	

ID	DEFAULT VALUE
	"Turks and Caicos Islands","TV":"Tuvalu","UM":"U.S. Outlying Islands","VG":"U.S. Virgin Islands","UG":"Uganda","UA":"Ukraine","AE":"United Arab Emirates","GB":"United Kingdom","US":"United States","UY":"Uruguay","UZ":"Uzbekistan","VU":"Vanuatu","VA":"Vatican City","VE":"Venezuela","VN":"Vietnam","WF":"Wallis and Futuna","YE":"Yemen","ZM":"Zambia","ZW":"Zimbabwe"}
error_448	The phone number you provided is unreachable.
error_449	User has exceeded the number of retry attempts.
verification_code_input_placeholder_text	Verification code

The following example shows the use of some of the user interface elements in the MFA enrollment page:



The following example shows the use of some of the user interface elements in the MFA validation page:



Verification display control user interface elements

The following are the IDs for a [Verification display control](#)

ID	DEFAULT VALUE
verification_control_but_change_claims	Change
verification_control_fail_send_code	Failed to send the code, please try again later.
verification_control_fail_verify_code	Failed to verify the code, please try again later.

ID	DEFAULT VALUE
verification_control_but_send_code	Send Code
verification_control_but_send_new_code	Send New Code
verification_control_but_verify_code	Verify Code

One time password error messages

The following are the IDs for a [one time password technical profile](#) error messages

ID	DEFAULT VALUE
UserMessageIfMaxRetryAttempted	One time password provided verification has exceeded maximum number of attempts
UserMessageIfSessionDoesNotExist	One time password verification session has expired
UserMessageIfSessionConflict	One time password verification session has conflict
UserMessageIfInvalidCode	One time password provided for verification is incorrect

Display controls

2/11/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A **display control** is a user interface element that has special functionality and interacts with the Azure Active Directory B2C (Azure AD B2C) back-end service. It allows the user to perform actions on the page that invoke a [validation technical profile](#) at the back end. Display controls are displayed on the page and are referenced by a [self-asserted technical profile](#).

The following image illustrates a self-assigned sign-up page with two display controls that validate a primary and secondary email address.

The screenshot shows a sign-up form with several input fields. The first two sections, 'Email Address' and 'Secondary Email Address', are highlighted with red boxes. Each section contains an input field and a blue 'Send Code' button below it. The remaining fields are standard text inputs: 'Display Name', 'Given Name', 'Surname', 'New Password', and 'Confirm New Password'. At the bottom are 'Create' and 'Cancel' buttons. The entire form is contained within a light gray box.

NOTE

This feature is in public preview.

Prerequisites

In the [Metadata](#) section of a [self-assigned technical profile](#), the referenced [ContentDefinition](#) needs to have [Datauri](#) set to page contract version 2.0.0 or higher. For example:

```

<ContentDefinition Id="api.selfasserted">
  <LoadUri>/~/tenant/default/selfAsserted.cshtml</LoadUri>
  <RecoveryUri>/~/common/default_page_error.html</RecoveryUri>
  <DataUri>urn:com:microsoft:aad:b2c:elements:selfasserted:2.0.0</DataUri>
  ...

```

Defining display controls

The **DisplayControl** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier that's used for the display control. It can be referenced .
UserInterfaceControlType	Yes	The type of the display control. Currently supported is VerificationControl

The **DisplayControl** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
InputClaims	0:1	InputClaims are used to prepopulate the value of the claims to be collected from the user.
DisplayClaims	0:1	DisplayClaims are used to represent claims to be collected from the user.
OutputClaims	0:1	OutputClaims are used to represent claims to be saved temporarily for this DisplayControl .
Actions	0:1	Actions are used to list the validation technical profiles to invoke for user actions happening at the front-end.

Input claims

In a display control, you can use **InputClaims** elements to prepopulate the value of claims to collect from the user on the page. Any **InputClaimsTransformations** can be defined in the self-asserted technical profile which references this display control.

The following example prepopulates the email address to be verified with the address already present.

```

<DisplayControl Id="emailControl" UserInterfaceControlType="VerificationControl">
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="emailAddress" />
  </InputClaims>
  ...

```

Display claims

Each type of display control requires a different set of display claims, [output claims](#), and [actions](#) to be performed.

Similar to the **display claims** defined in a [self-asserted technical profile](#), the display claims represent the claims to be collected from the user within the display control. The **ClaimType** element referenced needs to specify the **UserInputType** element for a user input type supported by Azure AD B2C, such as `TextBox` or `DropdownSingleSelect`. If a display claim value is required by an **Action**, set the **Required** attribute to `true` to force the user to provide a value for that specific display claim.

Certain display claims are required for certain types of display control. For example, **VerificationCode** is required for the display control of type **VerificationControl**. Use the attribute **ControlClaimType** to specify which DisplayClaim is designated for that required claim. For example:

```
<DisplayClaim ClaimTypeReferenceId="otpCode" ControlClaimType="VerificationCode" Required="true" />
```

Output claims

The **output claims** of a display control are not sent to the next orchestration step. They are saved temporarily only for the current display control session. These temporary claims can be shared between the different actions of the same display control.

To bubble up the output claims to the next orchestration step, use the **OutputClaims** of the actual self-asserted technical profile which references this display control.

Display control Actions

The **Actions** of a display control are procedures that occur in the Azure AD B2C back end when a user performs a certain action on the client side (the browser). For example, the validations to perform when the user selects a button on the page.

An action defines a list of [validation technical profiles](#). They are used for validating some or all of the display claims of the display control. The validation technical profile validates the user input and may return an error to the user. You can use **ContinueOnError**, **ContinueOnSuccess**, and **Preconditions** in the display control Action similar to the way they're used in [validation technical profiles](#) in a self asserted technical profile.

The following example sends a code either in email or SMS based on the user's selection of the **mfaType** claim.

```
<Action Id="SendCode">
  <ValidationClaimsExchange>
    <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="AzureMfa-SendSms">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>mfaType</Value>
          <Value>email</Value>
          <Action>SkipThisValidationTechnicalProfile</Action>
        </Precondition>
      </Preconditions>
    </ValidationClaimsExchangeTechnicalProfile>
    <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="AadSspr-SendEmail">
      <Preconditions>
        <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
          <Value>mfaType</Value>
          <Value>phone</Value>
          <Action>SkipThisValidationTechnicalProfile</Action>
        </Precondition>
      </Preconditions>
    </ValidationClaimsExchangeTechnicalProfile>
  </ValidationClaimsExchange>
</Action>
```

Referencing display controls

Display controls are referenced in the [display claims](#) of the [self-asserted technical profile](#).

For example:

```
<TechnicalProfile Id="SelfAsserted-ProfileUpdate">
  ...
  <DisplayClaims>
    <DisplayClaim DisplayControlReferenceId="emailVerificationControl" />
    <DisplayClaim DisplayControlReferenceId="PhoneVerificationControl" />
    <DisplayClaim ClaimTypeReferenceId="displayName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="givenName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="surName" Required="true" />
```

Verification display control

12/12/2019 • 2 minutes to read • [Edit Online](#)

Use a verification [display control](#) to verify a claim, for example an email address or phone number, with a verification code sent to the user.

VerificationControl actions

The verification display control consists of two steps (actions):

1. Request a destination from the user, such as an email address or phone number, to which the verification code should be sent. When the user selects the **Send Code** button, the **SendCode Action** of the verification display control is executed. The **SendCode Action** generates a code, constructs the content to be sent, and sends it to the user. The value of the address can be pre-populated and serve as a second-factor authentication.

Email Address
someone@contoso.com
Send Code

2. After the code has been sent, the user reads the message, enters the verification code into the control provided by the display control, and selects **Verify Code**. By selecting **Verify Code**, the **VerifyCode Action** is executed to verify the code associated with the address. If the user selects **Send New Code**, the first action is executed again.

Email Address
someone@contoso.com
Secondary Verification Code
Verify Code Send New Code

NOTE

This feature is in public preview.

VerificationControl required elements

The **VerificationControl** must contain following elements:

- The type of the `DisplayControl` is `VerificationControl`.
- `DisplayClaims`
 - **Send to** - One or more claims specifying where to send the verification code to. For example, *email* or *country code* and *phone number*.
 - **Verification code** - The verification code claim that user provides after the code has been sent. This claim must be set as required, and the `ControlClaimType` must be set to `VerificationCode`.
- Output claim (optional) to be returned to the self-asserted page after the user completes verification process. For example, *email* or *country code* and *phone number*. The self-asserted technical profile uses the claims to persist the data or bubble up the output claims to the next orchestration step.

- Two **Action**s with following names:
 - **SendCode** - Sends a code to the user. This action usually contains two validation technical profile, to generate a code and to send it.
 - **VerifyCode** - Verifies the code. This action usually contains a single validation technical profile.

In the example below, an **email** textbox is displayed on the page. When the user enters their email address and selects **SendCode**, the **SendCode** action is triggered in the Azure AD B2C back end.

Then, the user enters the **verificationCode** and selects **VerifyCode** to trigger the **VerifyCode** action in the back end. If all validations pass, the **VerificationControl** is considered complete and the user can continue to the next step.

```
<DisplayControl Id="emailVerificationControl" UserInterfaceControlType="VerificationControl">
  <DisplayClaims>
    <DisplayClaim ClaimTypeReferenceId="email" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="verificationCode" ControlClaimType="VerificationCode" Required="true" />
  />
  </DisplayClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" />
  </OutputClaims>
  <Actions>
    <Action Id="SendCode">
      <ValidationClaimsExchange>
        <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="GenerateOtp" />
        <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="SendGrid" />
      </ValidationClaimsExchange>
    </Action>
    <Action Id="VerifyCode">
      <ValidationClaimsExchange>
        <ValidationClaimsExchangeTechnicalProfile TechnicalProfileReferenceId="VerifyOtp" />
      </ValidationClaimsExchange>
    </Action>
  </Actions>
</DisplayControl>
```

ClaimsProviders

2/3/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A claims provider contains a set of [technical profiles](#). Every claims provider must have one or more technical profiles that determine the endpoints and the protocols needed to communicate with the claims provider. A claims provider can have multiple technical profiles. For example, 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.

```
<ClaimsProviders>
  <ClaimsProvider>
    <Domain>Domain name</Domain>
    <DisplayName>Display name</DisplayName>
    <TechnicalProfiles>
      </TechnicalProfile>
      ...
      </TechnicalProfile>
      ...
    </TechnicalProfiles>
  </ClaimsProvider>
  ...
</ClaimsProviders>
```

The **ClaimsProviders** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
ClaimsProvider	1:n	An accredited claims provider that can be leveraged in various user journeys.

ClaimsProvider

The **ClaimsProvider** element contains the following child elements:

ELEMENT	OCCURRENCES	DESCRIPTION

ELEMENT	OCCURRENCES	DESCRIPTION
Domain	0:1	A string that contains the domain name for the claim provider. For example, if your claims provider includes the Facebook technical profile, the domain name is Facebook.com. This domain name is used for all technical profiles defined in the claims provider unless overridden by the technical profile. The domain name can also be referenced in a domain_hint . For more information, see the Redirect sign-in to a social provider section of Set up direct sign-in using Azure Active Directory B2C .
DisplayName	1:1	A string that contains the name of the claims provider.
TechnicalProfiles	0:1	A set of technical profiles supported by the claim provider

ClaimsProvider organizes how your technical profiles relate to the claims provider. The following example shows the Azure Active Directory claims provider with the Azure Active Directory technical profiles:

```

<ClaimsProvider>
  <DisplayName>Azure Active Directory</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="AAD-Common">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserWriteUsingAlternativeSecurityId">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId-NoError">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserReadUsingEmailAddress">
      ...
    </TechnicalProfile>
    ...
    <TechnicalProfile Id="AAD-UserWritePasswordUsingObjectId">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserWriteProfileUsingObjectId">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserReadUsingObjectId">
      ...
    </TechnicalProfile>
    <TechnicalProfile Id="AAD-UserWritePhoneNumberUsingObjectId">
      ...
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

The following example shows the Facebook claims provider with the **Facebook-OAUTH** technical profile.

```
<ClaimsProvider>
  <Domain>facebook.com</Domain>
  <DisplayName>Facebook</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Facebook-OAUTH">
      <DisplayName>Facebook</DisplayName>
      <Protocol Name="OAuth2" />
      ...
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>
```

TechnicalProfiles

2/17/2020 • 10 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A **TechnicalProfiles** element contains a set of technical profiles supported by the claim provider. Every claims provider must have one or more technical profiles that determine the endpoints and the protocols needed to communicate with the claims provider. A claims provider can have multiple technical profiles.

```

<ClaimsProvider>
  <DisplayName>Display name</DisplayName>
  <TechnicalProfiles>
    <TechnicalProfile Id="Technical profile identifier">
      <DisplayName>Display name of technical profile</DisplayName>
      <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
      Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
      <Metadata>
        <Item Key="ServiceUrl">URL of service</Item>
        <Item Key="AuthenticationType">None</Item>
        <Item Key="SendClaimsIn">Body</Item>
      </Metadata>
      <InputTokenFormat>JWT</InputTokenFormat>
      <OutputTokenFormat>JWT</OutputTokenFormat>
      <CryptographicKeys>
        <Key ID="Key identifier" StorageReferenceId="Storage key container identifier"/>
        ...
      </CryptographicKeys>
      <InputClaimsTransformations>
        <InputClaimsTransformation ReferenceId="Claims transformation identifier" />
        ...
      <InputClaimsTransformations>
      <InputClaims>
        <InputClaim ClaimTypeReferenceId="givenName" DefaultValue="givenName" PartnerClaimType="firstName" />
        ...
      </InputClaims>
      <PersistedClaims>
        <PersistedClaim ClaimTypeReferenceId="givenName" DefaultValue="givenName" PartnerClaimType="firstName"
      />
        ...
      </PersistedClaims>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="loyaltyNumber" DefaultValue="loyaltyNumber"
      PartnerClaimType="loyaltyNumber" />
      </OutputClaims>
      <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="Claims transformation identifier" />
        ...
      <OutputClaimsTransformations>
      <ValidationTechnicalProfiles>
        <ValidationTechnicalProfile ReferenceId="Technical profile identifier" />
        ...
      </ValidationTechnicalProfiles>
      <SubjectNamingInfo ClaimType="Claim type identifier" />
      <IncludeTechnicalProfile ReferenceId="Technical profile identifier" />
      <UseTechnicalProfileForSessionManagement ReferenceId="Technical profile identifier" />
    </TechnicalProfile>
  </TechnicalProfiles>
</ClaimsProvider>

```

The **TechnicalProfile** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	A unique identifier of the technical profile. The technical profile can be referenced using this identifier from other elements in the policy file. For example, OrchestrationSteps and ValidationTechnicalProfile .

The **TechnicalProfile** contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Domain	0:1	The domain name for the technical profile. For example, if your technical profile specifies the Facebook identity provider, the domain name is Facebook.com.
DisplayName	1:1	The name of the technical profile that can be displayed to users.
Description	0:1	The description of the technical profile that can be displayed to users.
Protocol	0:1	The protocol used for the communication with the other party.
Metadata	0:1	A collection of key/value pairs that are utilized by the protocol for communicating with the endpoint in the course of a transaction.
InputTokenFormat	0:1	The format of the input token. Possible values: <code>JSON</code> , <code>JWT</code> , <code>SAML11</code> , or <code>SAML2</code> . The <code>JWT</code> value represents a JSON Web Token as per IETF specification. The <code>SAML11</code> value represents a SAML 1.1 security token as per OASIS specification. The <code>SAML2</code> value represents a SAML 2.0 security token as per OASIS specification.
OutputTokenFormat	0:1	The format of the output token. Possible values: <code>JSON</code> , <code>JWT</code> , <code>SAML11</code> , or <code>SAML2</code> .
CryptographicKeys	0:1	A list of cryptographic keys that are used in the technical profile.
InputClaimsTransformations	0:1	A list of previously defined references to claims transformations that should be executed before any claims are sent to the claims provider or the relying party.
InputClaims	0:1	A list of the previously defined references to claim types that are taken as input in the technical profile.
PersistedClaims	0:1	A list of the previously defined references to claim types that are persisted by the claims provider that relates to the technical profile.

ELEMENT	OCCURRENCES	DESCRIPTION
DisplayClaims	0:1	A list of the previously defined references to claim types that are presented by the claims provider that relates to the self-asserted technical profile . The DisplayClaims feature is currently in preview .
OutputClaims	0:1	A list of the previously defined references to claim types that are taken as output in the technical profile.
OutputClaimsTransformations	0:1	A list of previously defined references to claims transformations that should be executed after the claims are received from the claims provider.
ValidationTechnicalProfiles	0:n	A list of references to other technical profiles that the technical profile uses for validation purposes. For more information, see validation technical profile
SubjectNamingInfo	0:1	Controls the production of the subject name in tokens where the subject name is specified separately from claims. For example, OAuth or SAML.
IncludeInSso	0:1	Whether usage of this technical profile should apply single sign-on (SSO) behavior for the session, or instead require explicit interaction. This element is valid only in SelfAsserted profiles used within a Validation technical profile. Possible values: <code>true</code> (default), or <code>false</code> .
IncludeClaimsFromTechnicalProfile	0:1	An identifier of a technical profile from which you want all of the input and output claims to be added to this technical profile. The referenced technical profile must be defined in the same policy file.
IncludeTechnicalProfile	0:1	An identifier of a technical profile from which you want all data to be added to this technical profile. The referenced technical profile must exist in the same policy file.
UseTechnicalProfileForSessionManagement	0:1	A different technical profile to be used for session management.
EnabledForUserJourneys	0:1	Controls if the technical profile is executed in a user journey.

Protocol

The **Protocol** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Name	Yes	The name of a valid protocol supported by Azure AD B2C that is used as part of the technical profile. Possible values: <code>OAuth1</code> , <code>OAuth2</code> , <code>SAML2</code> , <code>OpenIdConnect</code> , <code>Proprietary</code> , or <code>None</code> .
Handler	No	When the protocol name is set to <code>Proprietary</code> , specify the fully-qualified name of the assembly that is used by Azure AD B2C to determine the protocol handler.

Metadata

A **Metadata** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Item	0:n	The metadata that relates to the technical profile. Each type of technical profile has a different set of metadata items. See the technical profile types section, for more information.

Item

The **Item** element of the **Metadata** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Key	Yes	The metadata key. See each technical profile type, for the list of metadata items.

CryptographicKeys

The **CryptographicKeys** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
Key	1:n	A cryptographic key used in this technical profile.

Key

The **Key** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	No	A unique identifier of a particular key pair referenced from other elements in the policy file.
StorageReferenceld	Yes	An identifier of a storage key container referenced from other elements in the policy file.

InputClaimsTransformations

The **InputClaimsTransformations** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
InputClaimsTransformation	1:n	The identifier of a claims transformation that should be executed before any claims are sent to the claims provider or the relying party. A claims transformation can be used to modify existing ClaimsSchema claims or generate new ones.

InputClaimsTransformation

The **InputClaimsTransformation** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a claims transformation already defined in the policy file or parent policy file.

InputClaims

The **InputClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
InputClaim	1:n	An expected input claim type.

InputClaim

The **InputClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceld	Yes	The identifier of a claim type already defined in the ClaimsSchema section in the policy file or parent policy file.
DefaultValue	No	A default value to use to create a claim if the claim indicated by ClaimTypeReferenceld does not exist so that the resulting claim can be used as an InputClaim by the technical profile.

ATTRIBUTE	REQUIRED	DESCRIPTION
PartnerClaimType	No	The identifier of the claim type of the external partner that the specified policy claim type maps to. If the PartnerClaimType attribute is not specified, then the specified policy claim type is mapped to the partner claim type of the same name. Use this property when your claim type name is different from the other party. For example, the first claim name is 'givenName', while the partner uses a claim named 'first_name'.

DisplayClaims

The **DisplayClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
DisplayClaim	1:n	An expected input claim type.

The DislayClaims feature is currently in [preview](#).

DisplayClaim

The **DisplayClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceld	No	The identifier of a claim type already defined in the ClaimsSchema section in the policy file or parent policy file.
DisplayControlReferenceld	No	The identifier of a display control already defined in the ClaimsSchema section in the policy file or parent policy file.
Required	No	Indicates whether the display claim is required.

The **DisplayClaim** requires that you specify either a `ClaimTypeReferenceId` or `DisplayControlReferenceId`.

PersistedClaims

The **PersistedClaims** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
PersistedClaim	1:n	The claim type to persist.

PersistedClaim

The **PersistedClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceld	Yes	The identifier of a claim type already defined in the ClaimsSchema section in the policy file or parent policy file.
DefaultValue	No	A default value to use to create a claim if the claim indicated by ClaimTypeReferenceld does not exist so that the resulting claim can be used as an InputClaim by the technical profile.
PartnerClaimType	No	The identifier of the claim type of the external partner that the specified policy claim type maps to. If the PartnerClaimType attribute is not specified, then the specified policy claim type is mapped to the partner claim type of the same name. Use this property when your claim type name is different from the other party. For example, the first claim name is 'givenName', while the partner uses a claim named 'first_name'.

OutputClaims

The **OutputClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
OutputClaim	1:n	An expected output claim type.

OutputClaim

The **OutputClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceld	Yes	The identifier of a claim type already defined in the ClaimsSchema section in the policy file or parent policy file.
DefaultValue	No	A default value to use to create a claim if the claim indicated by ClaimTypeReferenceld does not exist so that the resulting claim can be used as an InputClaim by the technical profile.
AlwaysUseDefaultValue	No	Force the use of the default value.

ATTRIBUTE	REQUIRED	DESCRIPTION
PartnerClaimType	No	The identifier of the claim type of the external partner that the specified policy claim type maps to. If the PartnerClaimType attribute is not specified, then the specified policy claim type is mapped to the partner claim type of the same name. Use this property when your claim type name is different from the other party. For example, the first claim name is 'givenName', while the partner uses a claim named 'first_name'.

OutputClaimsTransformations

The **OutputClaimsTransformations** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
OutputClaimsTransformation	1:n	The identifiers of claims transformations that should be executed before any claims are sent to the claims provider or the relying party. A claims transformation can be used to modify existing ClaimsSchema claims or generate new ones.

OutputClaimsTransformation

The **OutputClaimsTransformation** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a claims transformation already defined in the policy file or parent policy file.

ValidationTechnicalProfiles

The **ValidationTechnicalProfiles** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
ValidationTechnicalProfile	1:n	The identifiers of technical profiles that are used validate some or all of the output claims of the referencing technical profile. All of the input claims of the referenced technical profile must appear in the output claims of the referencing technical profile.

ValidationTechnicalProfile

The **ValidationTechnicalProfile** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a technical profile already defined in the policy file or parent policy file.

SubjectNamingInfo

The **SubjectNamingInfo** contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimType	Yes	An identifier of a claim type already defined in the ClaimsSchema section in the policy file.

IncludeTechnicalProfile

The **IncludeTechnicalProfile** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a technical profile already defined in the policy file or parent policy file.

UseTechnicalProfileForSessionManagement

The **UseTechnicalProfileForSessionManagement** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a technical profile already defined in the policy file or parent policy file.

EnabledForUserJourneys

The **ClaimsProviderSelections** in a user journey defines the list of claims provider selection options and their order. With the **EnabledForUserJourneys** element you filter, which claims provider is available to the user. The **EnabledForUserJourneys** element contains one of the following values:

- **Always**, execute the technical profile.
- **Never**, skip the technical profile.
- **OnClaimsExistence** execute only when a certain claim, specified in the technical profile exists.
- **OnItemExistenceInStringCollectionClaim**, execute only when an item exists in a string collection claim.
- **OnItemAbsenceInStringCollectionClaim** execute only when an item does not exist in a string collection claim.

Using **OnClaimsExistence**, **OnItemExistenceInStringCollectionClaim** or

OnItemAbsenceInStringCollectionClaim, requires you to provide the following metadata:

ClaimTypeOnWhichToEnable specifies the claim's type that is to be evaluated,

ClaimValueOnWhichToEnable specifies the value that is to be compared.

The following technical profile is executed only if the **identityProviders** string collection contains the value of

`facebook.com` :

```
<TechnicalProfile Id="UnLink-Facebook-OAUTH">
  <DisplayName>Unlink Facebook</DisplayName>
  ...
  <Metadata>
    <Item Key="ClaimTypeOnWhichToEnable">identityProviders</Item>
    <Item Key="ClaimValueOnWhichToEnable">facebook.com</Item>
  </Metadata>
  ...
  <EnabledForUserJourneys>OnItemExistenceInStringCollectionClaim</EnabledForUserJourneys>
</TechnicalProfile>
```

About technical profiles in Azure Active Directory B2C custom policies

2/20/2020 • 5 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A technical profile provides a framework with a built-in mechanism to communicate with different type of parties using a custom policy in Azure Active Directory B2C (Azure AD B2C). Technical profiles are used to communicate with your Azure AD B2C tenant, to create a user, or read a user profile. A technical profile can be self-asserted to enable interaction with the user. For example, collect the user's credential to sign in and then render the sign-up page or password reset page.

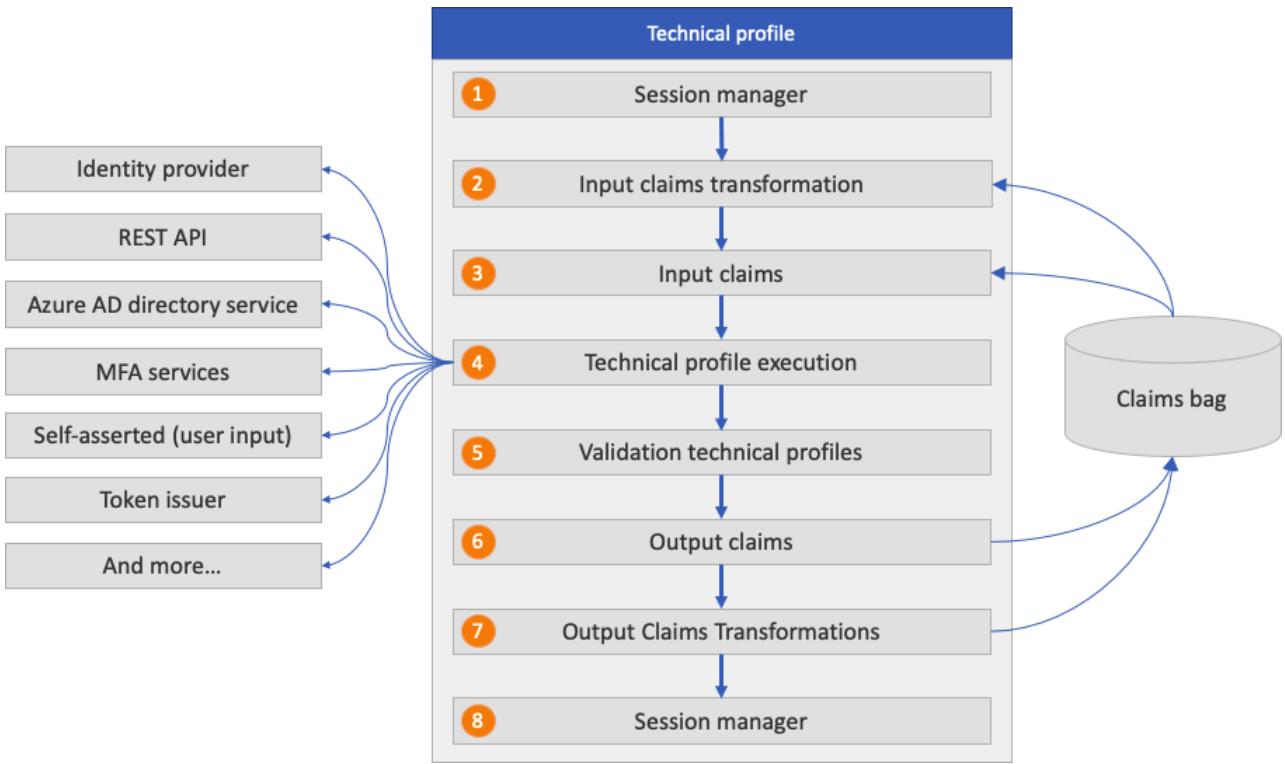
Type of technical profiles

A technical profile enables these types of scenarios:

- [Azure Active Directory](#) - Provides support for the Azure Active Directory B2C user management.
- [JWT token issuer](#) - Emits a JWT token that is returned back to the relying party application.
- **Phone factor provider** - Multi-factor authentication.
- [OAuth1](#) - Federation with any OAuth 1.0 protocol identity provider.
- [OAuth2](#) - Federation with any OAuth 2.0 protocol identity provider.
- [OpenID Connect](#) - Federation with any OpenID Connect protocol identity provider.
- [Claims transformation](#) - Call output claims transformations to manipulate claims values, validate claims, or set default values for a set of output claims.
- [RESTful provider](#) - Call to REST API services, such as validate user input, enrich user data, or integrate with line-of-business applications.
- [SAML2](#) - Federation with any SAML protocol identity provider.
- [Self-Asserted](#) - Interact with the user. For example, collect the user's credential to sign in, render the sign-up page, or password reset.
- [Session management](#) - Handle different types of sessions.
- [Application Insights](#)
- [One time password](#) - Provides support for managing the generation and verification of a one-time password.

Technical profile flow

All types of technical profiles share the same concept. You send input claims, run claims transformation, and communicate with the configured party, such as an identity provider, REST API, or Azure AD directory services. After the process is completed, the technical profile returns the output claims and may run output claims transformation. The following diagram shows how the transformations and mappings referenced in the technical profile are processed. Regardless of the party the technical profile interacts with, after any claims transformation is executed, the output claims from the technical profile are immediately stored in the claims bag.



- Single sign-on (SSO) session management** - Restores technical profile's session state, using [SSO session management](#).
- Input claims transformation** - Input claims of every input [claims transformation](#) are picked up from the claims bag. The output claims of an input claims transformation can be input claims of a subsequent input claims transformation.
- Input claims** - Claims are picked up from the claims bag and are used for the technical profile. For example, a [self-asserted technical profile](#) uses the input claims to prepopulate the output claims that the user provides. A REST API technical profile uses the input claims to send input parameters to the REST API endpoint. Azure Active Directory uses input claim as a unique identifier to read, update, or delete an account.
- Technical profile execution** - The technical profile exchanges the claims with the configured party. For example:
 - Redirect the user to the identity provider to complete the sign-in. After successful sign-in, the user returns back and the technical profile execution continues.
 - Call a REST API while sending parameters as InputClaims and getting information back as OutputClaims.
 - Create or update the user account.
 - Sends and verifies the MFA text message.
- Validation technical profiles** - A [self-asserted technical profile](#) can call [validation technical profiles](#). The validation technical profile validates the data profiled by the user and returns an error message or Ok, with or without output claims. For example, before Azure AD B2C creates a new account, it checks whether the user already exists in the directory services. You can call a REST API technical profile to add your own business logic. The scope of the output claims of a validation technical profile is limited to the technical profile that invokes the validation technical profile, and other validation technical profiles under same technical profile. If you want to use the output claims in the next orchestration step, you need to add the output claims to the technical profile that invokes the validation technical profile.
- Output claims** - Claims are returned back to the claims bag. You can use those claims in the next orchestrations step, or output claims transformations.
- Output claims transformations** - Input claims of every output [claims transformation](#) are picked up from the claims bag. The output claims of the technical profile from the previous steps can be input claims of an output

claims transformation. After execution, the output claims are put back in the claims bag. The output claims of an output claims transformation can also be input claims of a subsequent output claims transformation.

8. **Single sign-on (SSO) session management** - Persists technical profile's data to the session, using [SSO session management](#).

Technical profile inclusion

A technical profile can include another technical profile to change settings or add new functionality. The `IncludeTechnicalProfile` element is a reference to the base technical profile from which a technical profile is derived. There is no limit on the number of levels.

For example, the **AAD-UserReadUsingAlternativeSecurityId-NoError** technical profile includes the **AAD-UserReadUsingAlternativeSecurityId**. This technical profile sets the `RaiseErrorIfClaimsPrincipalDoesNotExist` metadata item to `true`, and raises an error if a social account does not exist in the directory. **AAD-UserReadUsingAlternativeSecurityId-NoError** overrides this behavior, and disables that error message.

```
<TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId-NoError">
  <Metadata>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">false</Item>
  </Metadata>
  <IncludeTechnicalProfile ReferenceId="AAD-UserReadUsingAlternativeSecurityId" />
</TechnicalProfile>
```

AAD-UserReadUsingAlternativeSecurityId includes the `AAD-Common` technical profile.

```
<TechnicalProfile Id="AAD-UserReadUsingAlternativeSecurityId">
  <Metadata>
    <Item Key="Operation">Read</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
    <Item Key="UserMessageIfClaimsPrincipalDoesNotExist">User does not exist. Please sign up before you can
sign in.</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="AlternativeSecurityId" PartnerClaimType="alternativeSecurityId"
Required="true" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="objectId" />
    <OutputClaim ClaimTypeReferenceId="userPrincipalName" />
    <OutputClaim ClaimTypeReferenceId="displayName" />
    <OutputClaim ClaimTypeReferenceId="otherMails" />
    <OutputClaim ClaimTypeReferenceId="givenName" />
    <OutputClaim ClaimTypeReferenceId="surname" />
  </OutputClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

Both **AAD-UserReadUsingAlternativeSecurityId-NoError** and **AAD-UserReadUsingAlternativeSecurityId** don't specify the required **Protocol** element, because it's specified in the **AAD-Common** technical profile.

```
<TechnicalProfile Id="AAD-Common">
  <DisplayName>Azure Active Directory</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureActiveDirectoryProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  ...
</TechnicalProfile>
```

Define an Azure MFA technical profile in an Azure AD B2C custom policy

12/17/2019 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for verifying a phone number by using Azure Multi-Factor Authentication (MFA). Use this technical profile to generate and send a code to a phone number, and then verify the code.

The Azure MFA technical profile may also return an error message. You can design the integration with Azure MFA by using a **Validation technical profile**. A validation technical profile calls the Azure MFA service. The validation technical profile validates the user-provided data before the user journey continues. With the validation technical profile, an error message is displayed on a self-asserted page.

NOTE

This feature is in public preview.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly that is used by Azure AD B2C:

```
Web.TPEngine.Providers.AzureMfaProtocolProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=null
```

The following example shows an Azure MFA technical profile:

```
<TechnicalProfile Id="AzureMfa-SendSms">  
  <DisplayName>Send Sms</DisplayName>  
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureMfaProtocolProvider, Web.TPEngine,  
  Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  ...
```

Send SMS

The first mode of this technical profile is to generate a code and send it. The following options can be configured for this mode.

Input claims

The **InputClaims** element contains a list of claims to send to Azure MFA. You can also map the name of your claim to the name defined in the MFA technical profile.

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
userPrincipalName	Yes	The identifier for the user who owns the phone number.
phoneNumber	Yes	The phone number to send an SMS code to.
companyName	No	The company name in the SMS. If not provided, the name of your application is used.
locale	No	The locale of the SMS. If not provided, the browser locale of the user is used.

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones before sending to the Azure MFA service.

Output claims

The Azure MFA protocol provider does not return any **OutputClaims**, thus there is no need to specify output claims. You can, however, include claims that aren't returned by the Azure MFA identity provider as long as you set the `DefaultValue` attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
Operation	Yes	Must be OneWaySMS .
UserMessageIfInvalidFormat	No	Custom error message if the phone number provided is not a valid phone number
UserMessageIfCouldntSendSms	No	Custom error message if the phone number provided does not accept SMS
UserMessageIfServerError	No	Custom error message if the server has encountered an internal error

Return an error message

As described in [Metadata](#), you can customize the error message shown to the user for different error cases. You can further localize those messages by prefixing the locale. For example:

```
<Item Key="en.UserMessageIfInvalidFormat">Invalid phone number.</Item>
```

Example: send an SMS

The following example shows an Azure MFA technical profile that is used to send a code via SMS.

```

<TechnicalProfile Id="AzureMfa-SendSms">
  <DisplayName>Send Sms</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureMfaProtocolProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="Operation">OneWaySMS</Item>
  </Metadata>
  <InputClaimsTransformations>
    <InputClaimsTransformation ReferenceId="CombinePhoneAndCountryCode" />
    <InputClaimsTransformation ReferenceId="ConvertStringToPhoneNumber" />
  </InputClaimsTransformations>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="userPrincipalName" />
    <InputClaim ClaimTypeReferenceId="fullPhoneNumber" PartnerClaimType="phoneNumber" />
  </InputClaims>
</TechnicalProfile>

```

Verify code

The second mode of this technical profile is to verify a code. The following options can be configured for this mode.

Input claims

The **InputClaims** element contains a list of claims to send to Azure MFA. You can also map the name of your claim to the name defined in the MFA technical profile.

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
phoneNumber	Yes	Same phone number as previously used to send a code. It is also used to locate a phone verification session.
verificationCode	Yes	The verification code provided by the user to be verified

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones before calling the Azure MFA service.

Output claims

The Azure MFA protocol provider does not return any **OutputClaims**, thus there is no need to specify output claims. You can, however, include claims that aren't returned by the Azure MFA identity provider as long as you set the `DefaultValue` attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
Operation	Yes	Must be Verify
UserMessageIfInvalidFormat	No	Custom error message if the phone number provided is not a valid phone number

ATTRIBUTE	REQUIRED	DESCRIPTION
UserMessageIfWrongCodeEntered	No	Custom error message if the code entered for verification is wrong
UserMessageIfMaxAllowedCodeRetryReached	No	Custom error message if the user has attempted a verification code too many times
UserMessageIfThrottled	No	Custom error message if the user is throttled
UserMessageIfServerError	No	Custom error message if the server has encountered an internal error

Return an error message

As described in [Metadata](#), you can customize the error message shown to the user for different error cases. You can further localize those messages by prefixing the locale. For example:

```
<Item Key="en.UserMessageIfWrongCodeEntered">Wrong code has been entered.</Item>
```

Example: verify a code

The following example shows an Azure MFA technical profile used to verify the code.

```
<TechnicalProfile Id="AzureMfa-VerifySms">
    <DisplayName>Verify Sms</DisplayName>
    <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureMfaProtocolProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
    <Metadata>
        <Item Key="Operation">Verify</Item>
    </Metadata>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="phoneNumber" PartnerClaimType="phoneNumber" />
        <InputClaim ClaimTypeReferenceId="verificationCode" />
    </InputClaims>
</TechnicalProfile>
```

About claim resolvers in Azure Active Directory B2C custom policies

2/17/2020 • 5 minutes to read • [Edit Online](#)

Claim resolvers in Azure Active Directory B2C (Azure AD B2C) [custom policies](#) provide context information about an authorization request, such as the policy name, request correlation ID, user interface language, and more.

To use a claim resolver in an input or output claim, you define a string **ClaimType**, under the [ClaimsSchema](#) element, and then you set the **DefaultValue** to the claim resolver in the input or output claim element. Azure AD B2C reads the value of the claim resolver and uses the value in the technical profile.

In the following example, a claim type named `correlationId` is defined with a **DataType** of `string`.

```
<ClaimType Id="correlationId">
  <DisplayName>correlationId</DisplayName>
  <DataType>string</DataType>
  <UserHelpText>Request correlation Id</UserHelpText>
</ClaimType>
```

In the technical profile, map the claim resolver to the claim type. Azure AD B2C populates the value of the claim resolver `{Context:CorrelationId}` into the claim `correlationId` and sends the claim to the technical profile.

```
<InputClaim ClaimTypeReferenceId="correlationId" DefaultValue="{Context:CorrelationId}" />
```

Claim resolver types

The following sections list available claim resolvers.

Culture

CLAIM	DESCRIPTION	EXAMPLE
{Culture:LanguageName}	The two letter ISO code for the language.	en
{Culture:LCID}	The LCID of language code.	1033
{Culture:RegionName}	The two letter ISO code for the region.	US
{Culture:RFC5646}	The RFC5646 language code.	en-US

Policy

CLAIM	DESCRIPTION	EXAMPLE
{Policy:PolicyId}	The relying party policy name.	B2C_1A_signup_signin

CLAIM	DESCRIPTION	EXAMPLE
{Policy:RelyingPartyTenantId}	The tenant ID of the relying party policy.	your-tenant.onmicrosoft.com
{Policy:TenantObjectId}	The tenant object ID of the relying party policy.	00000000-0000-0000-0000-000000000000
{Policy:TrustFrameworkTenantId}	The tenant ID of the trust framework.	your-tenant.onmicrosoft.com

OpenID Connect

CLAIM	DESCRIPTION	EXAMPLE
{OIDC:AuthenticationContextReferences}	The <code>acr_values</code> query string parameter.	N/A
{OIDC:ClientId}	The <code>client_id</code> query string parameter.	00000000-0000-0000-0000-000000000000
{OIDC:DomainHint}	The <code>domain_hint</code> query string parameter.	facebook.com
{OIDC:LoginHint}	The <code>login_hint</code> query string parameter.	someone@contoso.com
{OIDC:MaxAge}	The <code>max_age</code> .	N/A
{OIDC:Nonce}	The <code>nonce</code> query string parameter.	defaultNonce
{OIDC:Prompt}	The <code>prompt</code> query string parameter.	login
{OIDC:Resource}	The <code>resource</code> query string parameter.	N/A
{OIDC:scope}	The <code>scope</code> query string parameter.	openid

Context

CLAIM	DESCRIPTION	EXAMPLE
{Context:BuildNumber}	The Identity Experience Framework version (build number).	1.0.507.0
{Context:CorrelationId}	The correlation ID.	00000000-0000-0000-0000-000000000000
{Context:DateTimeInUtc}	The date time in UTC.	10/10/2018 12:00:00 PM
{Context:DeploymentMode}	The policy deployment mode.	Production
{Context:IPAddress}	The user IP address.	11.111.111.11

Non-protocol parameters

Any parameter name included as part of an OIDC or OAuth2 request can be mapped to a claim in the user

journey. For example, the request from the application might include a query string parameter with a name of `app_session`, `loyalty_number`, or any custom query string.

CLAIM	DESCRIPTION	EXAMPLE
{OAUTH-KV:campaignId}	A query string parameter.	hawaii
{OAUTH-KV:app_session}	A query string parameter.	A3C5R
{OAUTH-KV:loyalty_number}	A query string parameter.	1234
{OAUTH-KV:any custom query string}	A query string parameter.	N/A

OAuth2

CLAIM	DESCRIPTION	EXAMPLE
{oauth2:access_token}	The access token.	N/A

SAML

CLAIM	DESCRIPTION	EXAMPLE
{SAML:AuthnContextClassReferences}	The <code>AuthnContextClassRef</code> element value, from the SAML request.	urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
{SAML:NameIdPolicyFormat}	The <code>Format</code> attribute, from the <code>NameIDPolicy</code> element of the SAML request.	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
{SAML:Issuer}	The SAML <code>Issuer</code> element value of the SAML request.	https://contoso.com
{SAML:AllowCreate}	The <code>AllowCreate</code> attribute value, from the <code>NameIDPolicy</code> element of the SAML request.	True
{SAML:ForceAuthn}	The <code>ForceAuthN</code> attribute value, from the <code>AuthnRequest</code> element of the SAML request.	True
{SAML:ProviderName}	The <code>ProviderName</code> attribute value, from the <code>AuthnRequest</code> element of the SAML request.	Contoso.com

Using claim resolvers

You can use claims resolvers with the following elements:

ITEM	ELEMENT	SETTINGS
Application Insights technical profile	<code>InputClaim</code>	
Azure Active Directory technical profile	<code>InputClaim</code> , <code>OutputClaim</code>	1, 2

ITEM	ELEMENT	SETTINGS
OAuth2 technical profile	<code>InputClaim</code> , <code>OutputClaim</code>	1, 2
OpenID Connect technical profile	<code>InputClaim</code> , <code>OutputClaim</code>	1, 2
Claims transformation technical profile	<code>InputClaim</code> , <code>OutputClaim</code>	1, 2
RESTful provider technical profile	<code>InputClaim</code>	1, 2
SAML2 technical profile	<code>OutputClaim</code>	1, 2
Self-Asserted technical profile	<code>InputClaim</code> , <code>OutputClaim</code>	1, 2
ContentDefinition	<code>LoadUri</code>	
ContentDefinitionParameters	<code>Parameter</code>	
RelyingParty technical profile	<code>OutputClaim</code>	2

Settings:

1. The `IncludeClaimResolvingInClaimsHandling` metadata must be set to `true`.
2. The input or output claims attribute `AlwaysUseDefaultValue` must be set to `true`.

Claim resolvers samples

RESTful technical profile

In a RESTful technical profile, you may want to send the user language, policy name, scope, and client ID. Based on these claims the REST API can run custom business logic, and if necessary raise a localized error message.

The following example shows a RESTful technical profile with this scenario:

```
<TechnicalProfile Id="REST">
  <DisplayName>Validate user input data and return loyaltyNumber claim</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://your-app.azurewebsites.net/api/identity</Item>
    <Item Key="AuthenticationType">None</Item>
    <Item Key="SendClaimsIn">Body</Item>
    <Item Key="IncludeClaimResolvingInClaimsHandling">true</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="userLanguage" DefaultValue="{Culture:LCID}"
AlwaysUseDefaultValue="true" />
    <InputClaim ClaimTypeReferenceId="policyName" DefaultValue="{Policy:PolicyId}"
AlwaysUseDefaultValue="true" />
    <InputClaim ClaimTypeReferenceId="scope" DefaultValue="{OIDC:scope}" AlwaysUseDefaultValue="true" />
    <InputClaim ClaimTypeReferenceId="clientId" DefaultValue="{OIDC:ClientId}" AlwaysUseDefaultValue="true"
/>
  </InputClaims>
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

Direct sign-in

Using claim resolvers, you can prepopulate the sign-in name or direct sign-in to a specific social identity provider, such as Facebook, LinkedIn, or a Microsoft account. For more information, see [Set up direct sign-in using Azure Active Directory B2C](#).

Dynamic UI customization

Azure AD B2C enables you to pass query string parameters to your HTML content definition endpoints to dynamically render the page content. For example, this allows the ability to modify the background image on the Azure AD B2C sign-up or sign-in page based on a custom parameter that you pass from your web or mobile application. For more information, see [Dynamically configure the UI by using custom policies in Azure Active Directory B2C](#). You can also localize your HTML page based on a language parameter, or you can change the content based on the client ID.

The following example passes in the query string parameter named **campaignId** with a value of `hawaii`, a **language** code of `en-US`, and **app** representing the client ID:

```
<UserJourneyBehaviors>
  <ContentDefinitionParameters>
    <Parameter Name="campaignId">{OAUTH-KV:campaignId}</Parameter>
    <Parameter Name="language">{Culture:RFC5646}</Parameter>
    <Parameter Name="app">{OIDC:ClientId}</Parameter>
  </ContentDefinitionParameters>
</UserJourneyBehaviors>
```

As a result, Azure AD B2C sends the above parameters to the HTML content page:

```
/selfAsserted.aspx?campaignId=hawaii&language=en-US&app=0239a9cc-309c-4d41-87f1-31288feb2e82
```

Content definition

In a [ContentDefinition](#) `LoadUri`, you can send claim resolvers to pull content from different places, based on the parameters used.

```
<ContentDefinition Id="api.signuporsignin">
  <LoadUri>https://contoso.blob.core.windows.net/{Culture:LanguageName}/myHTML/unified.html</LoadUri>
  ...
</ContentDefinition>
```

Application Insights technical profile

With Azure Application Insights and claim resolvers you can gain insights on user behavior. In the Application Insights technical profile, you send input claims that are persisted to Azure Application Insights. For more information, see [Track user behavior in Azure AD B2C journeys by using Application Insights](#). The following example sends the policy ID, correlation ID, language, and the client ID to Azure Application Insights.

```

<TechnicalProfile Id="AzureInsights-Common">
  <DisplayName>Alternate Email</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.Insights.AzureApplicationInsightsProvider,
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  ...
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="PolicyId" PartnerClaimType="{property:Policy}" DefaultValue="
{Policy:PolicyId}" />
    <InputClaim ClaimTypeReferenceId="CorrelationId" PartnerClaimType="{property:CorrelationId}" DefaultValue="
{Context:CorrelationId}" />
    <InputClaim ClaimTypeReferenceId="language" PartnerClaimType="{property:language}" DefaultValue="
{Culture:RFC5646}" />
    <InputClaim ClaimTypeReferenceId="AppId" PartnerClaimType="{property:App}" DefaultValue="{OIDC:ClientId}" />
  </InputClaims>
</TechnicalProfile>

```

Relying party policy

In a [Relying party](#) policy technical profile, you may want to send the tenant ID, or correlation ID to the relying party application within the JWT.

```

<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
      <OutputClaim ClaimTypeReferenceId="tenantId" AlwaysUseDefaultValue="true" DefaultValue="
{Policy:TenantObjectId}" />
      <OutputClaim ClaimTypeReferenceId="correlationId" AlwaysUseDefaultValue="true" DefaultValue="
{Context:CorrelationId}" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>

```

Define an Azure Active Directory technical profile in an Azure Active Directory B2C custom policy

2/13/2020 • 6 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for the Azure Active Directory user management. This article describes the specifics of a technical profile for interacting with a claims provider that supports this standardized protocol.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly

```
Web.TPEngine.Providers.AzureActiveDirectoryProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=null
```

All Azure AD technical profiles include the **AAD-Common** technical profile. The following technical profiles don't specify the protocol because the protocol is configured in the **AAD-Common** technical profile:

- **AAD-UserReadUsingAlternativeSecurityId** and **AAD-UserReadUsingAlternativeSecurityId-NoError** - Look up a social account in the directory.
- **AAD-UserWriteUsingAlternativeSecurityId** - Create a new social account.
- **AAD-UserReadUsingEmailAddress** - Look up a local account in the directory.
- **AAD-UserWriteUsingLogonEmail** - Create a new local account.
- **AAD-UserWritePasswordUsingObjectId** - Update a password of a local account.
- **AAD-UserWriteProfileUsingObjectId** - Update a user profile of a local or social account.
- **AAD-UserReadUsingObjectId** - Read a user profile of a local or social account.
- **AAD-UserWritePhoneNumberUsingObjectId** - Write the MFA phone number of a local or social account

The following example shows the **AAD-Common** technical profile:

```
<TechnicalProfile Id="AAD-Common">  
  <DisplayName>Azure Active Directory</DisplayName>  
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.AzureActiveDirectoryProvider, Web.TPEngine,  
  Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  
  <CryptographicKeys>  
    <Key Id="issuer_secret" StorageReferenceId="B2C_1A_TokenSigningKeyContainer" />  
  </CryptographicKeys>  
  
  <!-- We need this here to suppress the SelfAsserted provider from invoking SSO on validation profiles. -->  
  <IncludeInSso>false</IncludeInSso>  
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />  
</TechnicalProfile>
```

Input claims

The following technical profiles include **InputClaims** for social and local accounts:

- The social account technical profiles **AAD-UserReadUsingAlternativeSecurityId** and **AAD-UserWriteUsingAlternativeSecurityId** includes the **AlternativeSecurityId** claim. This claim contains the social account user identifier.
- The local account technical profiles **AAD-UserReadUsingEmailAddress** and **AAD-UserWriteUsingLogonEmail** includes the **email** claim. This claim contains the sign-in name of the local account.
- The unified (local and social) technical profiles **AAD-UserReadUsingObjectId**, **AAD-UserWritePasswordUsingObjectId**, **AAD-UserWriteProfileUsingObjectId**, and **AAD-UserWritePhoneNumberUsingObjectId** includes the **objectId** claim. The unique identifier of an account.

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones.

Output claims

The **OutputClaims** element contains a list of claims returned by the Azure AD technical profile. You may need to map the name of the claim defined in your policy to the name defined in Azure Active Directory. You can also include claims that aren't returned by the Azure Active Directory, as long as you set the **DefaultValue** attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

For example, the **AAD-UserWriteUsingLogonEmail** technical profile creates a local account and returns the following claims:

- **objectId**, which is identifier of the new account
- **newUser**, which indicates whether the user is new
- **authenticationSource**, which sets authentication to **localAccountAuthentication**
- **userPrincipalName**, which is the user principal name of the new account
- **signInNames.emailAddress**, which is the account sign-in name, similar to the **email** input claim

```
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="objectId" />
  <OutputClaim ClaimTypeReferenceId="newUser" PartnerClaimType="newClaimsPrincipalCreated" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="localAccountAuthentication" />
  <OutputClaim ClaimTypeReferenceId="userPrincipalName" />
  <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
</OutputClaims>
```

PersistedClaims

The **PersistedClaims** element contains all of the values that should be persisted by Azure AD with possible mapping information between a claim type already defined in the ClaimsSchema section in the policy and the Azure AD attribute name.

The **AAD-UserWriteUsingLogonEmail** technical profile, which creates new local account, persists following claims:

```

<PersistedClaims>
    <!-- Required claims -->
    <PersistedClaim ClaimTypeReferenceId="email" PartnerClaimType="signInNames.emailAddress" />
    <PersistedClaim ClaimTypeReferenceId="newPassword" PartnerClaimType="password"/>
    <PersistedClaim ClaimTypeReferenceId="displayName" DefaultValue="unknown" />
    <PersistedClaim ClaimTypeReferenceId="passwordPolicies" DefaultValue="DisablePasswordExpiration" />

    <!-- Optional claims. -->
    <PersistedClaim ClaimTypeReferenceId="givenName" />
    <PersistedClaim ClaimTypeReferenceId="surname" />
</PersistedClaims>

```

The name of the claim is the name of the Azure AD attribute unless the **PartnerClaimType** attribute is specified, which contains the Azure AD attribute name.

Requirements of an operation

- There must be exactly one **InputClaim** element in the claims bag for all Azure AD technical profiles.
- If the operation is **Write** or **DeleteClaims**, then it must also appear in a **PersistedClaims** element.
- The value of the **userPrincipalName** claim must be in the format of `user@tenant.onmicrosoft.com`.
- The **displayName** claim is required and cannot be an empty string.

Azure AD technical provider operations

Read

The **Read** operation reads data about a single user account. To read user data, you need to provide a key as an input claim, such as **objectId**, **userPrincipalName**, **signInNames** (any type, user name and email-based account) or **alternativeSecurityId**.

The following technical profile reads data about a user account using the user's objectId:

```

<TechnicalProfile Id="AAD-UserReadUsingObjectId">
    <Metadata>
        <Item Key="Operation">Read</Item>
        <Item Key="RaiseErrorIfClaimsPrincipalDoesNotExist">true</Item>
    </Metadata>
    <IncludeInSso>false</IncludeInSso>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
    </InputClaims>
    <OutputClaims>

        <!-- Required claims -->
        <OutputClaim ClaimTypeReferenceId="strongAuthenticationPhoneNumber" />

        <!-- Optional claims -->
        <OutputClaim ClaimTypeReferenceId="signInNames.emailAddress" />
        <OutputClaim ClaimTypeReferenceId="displayName" />
        <OutputClaim ClaimTypeReferenceId="otherMails" />
        <OutputClaim ClaimTypeReferenceId="givenName" />
        <OutputClaim ClaimTypeReferenceId="surname" />
    </OutputClaims>
    <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>

```

Write

The **Write** operation creates or updates a single user account. To write a user account, you need to provide a key as an input claim, such as **objectId**, **userPrincipalName**, **signInNames.emailAddress**, or

alternativeSecurityId.

The following technical profile creates new social account:

```
<TechnicalProfile Id="AAD-UserWriteUsingAlternativeSecurityId">
  <Metadata>
    <Item Key="Operation">Write</Item>
    <Item Key="RaiseErrorIfClaimsPrincipalAlreadyExists">true</Item>
    <Item Key="UserMessageIfClaimsPrincipalAlreadyExists">You are already registered, please press the back button and sign in instead.</Item>
  </Metadata>
  <IncludeInSso>false</IncludeInSso>
  <InputClaimsTransformations>
    <InputClaimsTransformation ReferenceId="CreateOtherMailsFromEmail" />
  </InputClaimsTransformations>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="AlternativeSecurityId" PartnerClaimType="alternativeSecurityId" Required="true" />
  </InputClaims>
  <PersistedClaims>
    <!-- Required claims -->
    <PersistedClaim ClaimTypeReferenceId="alternativeSecurityId" />
    <PersistedClaim ClaimTypeReferenceId="userPrincipalName" />
    <PersistedClaim ClaimTypeReferenceId="mailNickName" DefaultValue="unknown" />
    <PersistedClaim ClaimTypeReferenceId="displayName" DefaultValue="unknown" />

    <!-- Optional claims -->
    <PersistedClaim ClaimTypeReferenceId="otherMails" />
    <PersistedClaim ClaimTypeReferenceId="givenName" />
    <PersistedClaim ClaimTypeReferenceId="surname" />
  </PersistedClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="objectId" />
    <OutputClaim ClaimTypeReferenceId="newUser" PartnerClaimType="newClaimsPrincipalCreated" />
    <OutputClaim ClaimTypeReferenceId="otherMails" />
  </OutputClaims>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>
```

DeleteClaims

The **DeleteClaims** operation clears the information from a provided list of claims. To delete information from claims, you need to provide a key as an input claim, such as **objectId**, **userPrincipalName**, **signInNames.emailAddress** or **alternativeSecurityId**.

The following technical profile deletes claims:

```
<TechnicalProfile Id="AAD-DeleteClaimsUsingObjectId">
  <Metadata>
    <Item Key="Operation">DeleteClaims</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <PersistedClaims>
    <PersistedClaim ClaimTypeReferenceId="objectId" />
    <PersistedClaim ClaimTypeReferenceId="Verified.strongAuthenticationPhoneNumber" PartnerClaimType="strongAuthenticationPhoneNumber" />
  </PersistedClaims>
  <OutputClaims />
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

DeleteClaimsPrincipal

The **DeleteClaimsPrincipal** operation deletes a single user account from the directory. To delete a user account, you need to provide a key as an input claim, such as **objectId**, **userPrincipalName**, **signInNames.emailAddress** or **alternativeSecurityId**.

The following technical profile deletes a user account from the directory using the user principal name:

```
<TechnicalProfile Id="AAD-DeleteUserUsingObjectId">
  <Metadata>
    <Item Key="Operation">DeleteClaimsPrincipal</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="objectId" Required="true" />
  </InputClaims>
  <OutputClaims/>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

The following technical profile deletes a social user account using **alternativeSecurityId**:

```
<TechnicalProfile Id="AAD-DeleteUserUsingAlternativeSecurityId">
  <Metadata>
    <Item Key="Operation">DeleteClaimsPrincipal</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="alternativeSecurityId" Required="true" />
  </InputClaims>
  <OutputClaims/>
  <IncludeTechnicalProfile ReferenceId="AAD-Common" />
</TechnicalProfile>
```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
Operation	Yes	The operation to be performed. Possible values: <code>Read</code> , <code>Write</code> , <code>DeleteClaims</code> , or <code>DeleteClaimsPrincipal</code> .
RaiseErrorIfClaimsPrincipalDoesNotExist	No	Raise an error if the user object does not exist in the directory. Possible values: <code>true</code> or <code>false</code> .
UserMessageIfClaimsPrincipalDoesNotExist	No	If an error is to be raised (see the <code>RaiseErrorIfClaimsPrincipalDoesNotExist</code> attribute description), specify the message to show to the user if user object does not exist. The value can be localized.
RaiseErrorIfClaimsPrincipalAlreadyExists	No	Raise an error if the user object already exists. Possible values: <code>true</code> or <code>false</code> .

ATTRIBUTE	REQUIRED	DESCRIPTION
UserMessageIfClaimsPrincipalAlreadyExists	No	If an error is to be raised (see RaiseErrorIfClaimsPrincipalAlreadyExists attribute description), specify the message to show to the user if user object already exists. The value can be localized .
ApplicationObjectId	No	The application object identifier for extension attributes. Value: ObjectId of an application. For more information, see Use custom attributes in a custom profile edit policy .
ClientId	No	The client identifier for accessing the tenant as a third party. For more information, see Use custom attributes in a custom profile edit policy
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .

Define a claims transformation technical profile in an Azure Active Directory B2C custom policy

2/13/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A claims transformation technical profile enables you to call output claims transformations to manipulate claims values, validate claims, or set default values for a set of output claims.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly that is used by Azure AD B2C:

```
Web.TPEngine.Providers.ClaimsTransformationProtocolProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=null
```

The following example shows a claims transformation technical profile:

```
<TechnicalProfile Id="Facebook-OAUTH-UnLink">  
  <DisplayName>Unlink Facebook</DisplayName>  
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.ClaimsTransformationProtocolProvider,  
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  ...
```

Output claims

The **OutputClaims** element is mandatory. You should provide at least one output claim returned by the technical profile. The following example shows how to set default values in the output claims:

```
<OutputClaims>  
  <OutputClaim ClaimTypeReferenceId="ageGroup" DefaultValue="Undefined" />  
  <OutputClaim ClaimTypeReferenceId="ageGroupValueChanged" DefaultValue="false" />  
</OutputClaims>
```

Output claims transformations

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify claims or generate new ones. The following technical profile calls the **RemoveAlternativeSecurityIdByIdentityProvider** claims transformation. This claims transformation removes a social identity from the collection of **AlternativeSecurityIds**. The output claims of this technical profile are **identityProvider2**, which is set to `facebook.com`, and **AlternativeSecurityIds**, which contains the list of social identities associated with this user after facebook.com identity is removed.

```

<ClaimsTransformations>
    <ClaimsTransformation Id="RemoveAlternativeSecurityIdByIdentityProvider"
TransformationMethod="RemoveAlternativeSecurityIdByIdentityProvider">
        <InputClaims>
            <InputClaim ClaimTypeReferenceId="IdentityProvider2"
TransformationClaimType="identityProvider" />
            <InputClaim ClaimTypeReferenceId="AlternativeSecurityIds"
TransformationClaimType="collection" />
        </InputClaims>
        <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="AlternativeSecurityIds"
TransformationClaimType="collection" />
        </OutputClaims>
    </ClaimsTransformation>
</ClaimsTransformations>
...
<TechnicalProfile Id="Facebook-OAUTH-UnLink">
    <DisplayName>Unlink Facebook</DisplayName>
    <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.ClaimsTransformationProtocolProvider,
Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="identityProvider2" DefaultValue="facebook.com"
AlwaysUseDefaultValue="true" />
    </OutputClaims>
    <OutputClaimsTransformations>
        <OutputClaimsTransformation ReferenceId="RemoveAlternativeSecurityIdByIdentityProvider" />
    </OutputClaimsTransformations>
    <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>

```

The claims transformation technical profile enables you to execute a claims transformation from any user journey's orchestration step. In the following example, the orchestration step calls one of the unlink technical profiles, such as **UnLink-Facebook-OAUTH**. This technical profile calls the claims transformation technical profile **RemoveAlternativeSecurityIdByIdentityProvider**, which generates a new **AlternativeSecurityIds2** claim that contains the list of user social identities, while removing the Facebook identity from the collections.

```

<UserJourney Id="AccountUnLink">
    <OrchestrationSteps>
        ...
        <OrchestrationStep Order="8" Type="ClaimsExchange">
            <ClaimsExchanges>
                <ClaimsExchange Id="UnLinkFacebookExchange" TechnicalProfileReferenceId="UnLink-Facebook-OAUTH" />
                <ClaimsExchange Id="UnLinkMicrosoftExchange" TechnicalProfileReferenceId="UnLink-Microsoft-OAUTH" />
                <ClaimsExchange Id="UnLinkGitHubExchange" TechnicalProfileReferenceId="UnLink-GitHub-OAUTH" />
            </ClaimsExchanges>
        </OrchestrationStep>
        ...
    </OrchestrationSteps>
</UserJourney>

```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
-----------	----------	-------------

ATTRIBUTE	REQUIRED	DESCRIPTION
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .

Use a validation technical profile

A claims transformation technical profile can be used to validate information. In the following example, the [self asserted technical profile](#) named **LocalAccountSignUpWithLogonEmail** asks the user to enter the email twice, then calls the [validation technical profile](#) named **Validate-Email** to validate the emails. The **Validate-Email** technical profile calls the claims transformation **AssertEmailAreEqual** to compare the two claims **email** and **emailRepeat**, and throw an exception if they are not equal according to the specified comparison.

```
<ClaimsTransformations>
  <ClaimsTransformation Id="AssertEmailAreEqual" TransformationMethod="AssertStringClaimsAreEqual">
    <InputClaims>
      <InputClaim ClaimTypeReferenceId="email" TransformationClaimType="inputClaim1" />
      <InputClaim ClaimTypeReferenceId="emailRepeat" TransformationClaimType="inputClaim2" />
    </InputClaims>
    <InputParameters>
      <InputParameter Id="stringComparison" DataType="string" Value="ordinalIgnoreCase" />
    </InputParameters>
  </ClaimsTransformation>
</ClaimsTransformations>
```

The claims transformation technical profile calls the **AssertEmailAreEqual** claims transformation, which asserts that emails provided by the user are same.

```
<TechnicalProfile Id="Validate-Email">
  <DisplayName>Unlink Facebook</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.ClaimsTransformationProtocolProvider,
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="emailRepeat" />
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" />
  </OutputClaims>
  <OutputClaimsTransformations>
    <OutputClaimsTransformation ReferenceId="AssertEmailAreEqual" />
  </OutputClaimsTransformations>
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-Noop" />
</TechnicalProfile>
```

A self-asserted technical profile can call the validation technical profile and show the error message as specified in the **UserMessageIfClaimsTransformationStringsAreNotEqual** metadata.

```
<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
  <DisplayName>User ID signup</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    ...
    <Item Key="UserMessageIfClaimsTransformationStringsAreNotEqual">The email addresses you provided are not
the same</Item>
  </Metadata>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" />
    <OutputClaim ClaimTypeReferenceId="emailRepeat" />
    ...
  </OutputClaims>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="Validate-Email" />
  </ValidationTechnicalProfiles>
</TechnicalProfile>
```

Define a technical profile for a JWT token issuer in an Azure Active Directory B2C custom policy

1/28/2020 • 3 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) emits several types of security tokens as it processes each authentication flow. A technical profile for a JWT token issuer emits a JWT token that is returned back to the relying party application. Usually this technical profile is the last orchestration step in the user journey.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `None`. Set the **OutputTokenFormat** element to `JWT`.

The following example shows a technical profile for `JwtIssuer`:

```
<TechnicalProfile Id="JwtIssuer">
  <DisplayName>JWT Issuer</DisplayName>
  <Protocol Name="None" />
  <OutputTokenFormat>JWT</OutputTokenFormat>
  ...
</TechnicalProfile>
```

Input, output, and persist claims

The **InputClaims**, **OutputClaims**, and **PersistClaims** elements are empty or absent. The **InputClaimsTransformations** and **OutputClaimsTransformations** elements are also absent.

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
issuer_refresh_token_user_identity_claim_type	Yes	The claim that should be used as the user identity claim within the OAuth2 authorization codes and refresh tokens. By default, you should set it to <code>objectId</code> , unless you specify a different SubjectNamingInfo claim type.
SendTokenResponseBodyWithJsonNumbers	No	Always set to <code>true</code> . For legacy format where numeric values are given as strings instead of JSON numbers, set to <code>false</code> . This attribute is needed for clients that have taken a dependency on an earlier implementation that returned such properties as strings.

ATTRIBUTE	REQUIRED	DESCRIPTION
token_lifetime_secs	No	Access token lifetimes. The lifetime of the OAuth 2.0 bearer token used to gain access to a protected resource. The default is 3,600 seconds (1 hour). The minimum (inclusive) is 300 seconds (5 minutes). The maximum (inclusive) is 86,400 seconds (24 hours).
id_token_lifetime_secs	No	ID token lifetimes. The default is 3,600 seconds (1 hour). The minimum (inclusive) is 300 seconds (5 minutes). The maximum (inclusive) is 86,400 seconds (24 hours).
refresh_token_lifetime_secs	No	Refresh token lifetimes. The maximum time period before which a refresh token can be used to acquire a new access token, if your application had been granted the offline_access scope. The default is 120,9600 seconds (14 days). The minimum (inclusive) is 86,400 seconds (24 hours). The maximum (inclusive) is 7,776,000 seconds (90 days).
rolling_refresh_token_lifetime_secs	No	Refresh token sliding window lifetime. After this time period elapses the user is forced to reauthenticate, irrespective of the validity period of the most recent refresh token acquired by the application. If you don't want to enforce a sliding window lifetime, set the value of allow_infinite_rolling_refresh_token to <code>true</code> . The default is 7,776,000 seconds (90 days). The minimum (inclusive) is 86,400 seconds (24 hours). The maximum (inclusive) is 31,536,000 seconds (365 days).
allow_infinite_rolling_refresh_token	No	If set to <code>true</code> , the refresh token sliding window lifetime never expires.
IssuanceClaimPattern	No	Controls the Issuer (iss) claim. One of the values: <ul style="list-style-type: none"> • AuthorityAndTenantGuid - The iss claim includes your domain name, such as <code>login.microsoftonline</code> or <code>tenant-name.b2clogin.com</code>, and your tenant identifier https://login.microsoftonline.com/00000000-0000-0000-0000-000000000000/v2.0/ • AuthorityWithTfp - The iss claim includes your domain name, such as <code>login.microsoftonline</code> or <code>tenant-name.b2clogin.com</code>, your tenant identifier and your relying party policy name. https://login.microsoftonline.com/tfp/00000000-0000-0000-0000-000000000000/b2c_1a_tp_sign-up-or-sign-in/v2.0/ Default value: AuthorityAndTenantGuid

ATTRIBUTE	REQUIRED	DESCRIPTION
AuthenticationContextReferenceClaimPattern	No	<p>Controls the <code>acr</code> claim value.</p> <ul style="list-style-type: none"> None - Azure AD B2C doesn't issue the acr claim PolicyId - the <code>acr</code> claim contains the policy name <p>The options for setting this value are TFP (trust framework policy) and ACR (authentication context reference). It is recommended setting this value to TFP, to set the value, ensure the <code><Item></code> with the</p> <pre>Key="AuthenticationContextReferenceClaimPattern" exists and the value is <code>None</code>. In your relying party policy, add <code><OutputClaims></code> item, add this element</pre> <pre><OutputClaim ClaimTypeReferenceId="trustFrameworkPolicy" Required="true" DefaultValue="{policy}" /></pre> <p>. Also make sure your policy contains the claim type</p> <pre><ClaimType Id="trustFrameworkPolicy"> <DisplayName>trustFrameworkPolicy</DisplayName> <DataType>string</DataType> </ClaimType></pre>

Cryptographic keys

The CryptographicKeys element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
issuer_secret	Yes	<p>The X509 certificate (RSA key set) to use to sign the JWT token. This is the <code>B2C_1A_TokenSigningKeyContainer</code> key you configured in Get started with custom policies.</p>
issuer_refresh_token_key	Yes	<p>The X509 certificate (RSA key set) to use to encrypt the refresh token. You configured the <code>B2C_1A_TokenEncryptionKeyContainer</code> key in Get started with custom policies</p>

Define an OAuth1 technical profile in an Azure Active Directory B2C custom policy

1/28/2020 • 2 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for the [OAuth 1.0 protocol](#) identity provider. This article describes the specifics of a technical profile for interacting with a claims provider that supports this standardized protocol. With an OAuth1 technical profile, you can federate with an OAuth1 based identity provider, such as Twitter. Federating with the identity provider allows users to sign in with their existing social or enterprise identities.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `OAuth1`. For example, the protocol for the **Twitter-OAUTH1** technical profile is `OAuth1`.

```
<TechnicalProfile Id="Twitter-OAUTH1">
  <DisplayName>Twitter</DisplayName>
  <Protocol Name="OAuth1" />
  ...

```

Input claims

The **InputClaims** and **InputClaimsTransformations** elements are empty or absent.

Output claims

The **OutputClaims** element contains a list of claims returned by the OAuth1 identity provider. You may need to map the name of the claim defined in your policy to the name defined in the identity provider. You can also include claims that aren't returned by the identity provider as long as you set the **DefaultValue** attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

The following example shows the claims returned by the Twitter identity provider:

- The **user_id** claim that is mapped to the **issuerUserId** claim.
- The **screen_name** claim that is mapped to the **displayName** claim.
- The **email** claim without name mapping.

The technical profile also returns claims that aren't returned by the identity provider:

- The **identityProvider** claim that contains the name of the identity provider.
- The **authenticationSource** claim with a default value of `socialIdpAuthentication`.

```

<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="user_id" />
  <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="screen_name" />
  <OutputClaim ClaimTypeReferenceId="email" />
  <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="twitter.com" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
</OutputClaims>

```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
client_id	Yes	The application identifier of the identity provider.
ProviderName	No	The name of the identity provider.
request_token_endpoint	Yes	The URL of the request token endpoint as per RFC 5849.
authorization_endpoint	Yes	The URL of the authorization endpoint as per RFC 5849.
access_token_endpoint	Yes	The URL of the token endpoint as per RFC 5849.
ClaimsEndpoint	No	The URL of the user information endpoint.
ClaimsResponseFormat	No	The claims response format.

Cryptographic keys

The **CryptographicKeys** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
client_secret	Yes	The client secret of the identity provider application.

Redirect URI

When you configure the redirect URL of your identity provider, enter

`https://login.microsoftonline.com/te/tenant/policyId/oauth2/authresp`. Make sure to replace **tenant** with your tenant name (for example, contosob2c.onmicrosoft.com) and **policyId** with the identifier of your policy (for example, b2c_1a_policy). The redirect URI needs to be in all lowercase. Add a redirect URL for all policies that use the identity provider login.

If you are using the **b2clogin.com** domain instead of **login.microsoftonline.com** Make sure to use b2clogin.com instead of login.microsoftonline.com.

Examples:

- Add Twitter as an OAuth1 identity provider by using custom policies

Define an OAuth2 technical profile in an Azure Active Directory B2C custom policy

2/24/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for the OAuth2 protocol identity provider. OAuth2 is the primary protocol for authorization and delegated authentication. For more information, see the [RFC 6749 The OAuth 2.0 Authorization Framework](#). With an OAuth2 technical profile, you can federate with an OAuth2 based identity provider, such as Facebook. Federating with an identity provider allows users to sign in with their existing social or enterprise identities.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `OAuth2`. For example, the protocol for the **Facebook-OAUTH** technical profile is `OAuth2`:

```
<TechnicalProfile Id="Facebook-OAUTH">
  <DisplayName>Facebook</DisplayName>
  <Protocol Name="OAuth2" />
  ...

```

Input claims

The **InputClaims** and **InputClaimsTransformations** elements are not required. But you may want to send additional parameters to your identity provider. The following example adds the **domain_hint** query string parameter with the value of `contoso.com` to the authorization request.

```
<InputClaims>
  <InputClaim ClaimTypeReferenceId="domain_hint" DefaultValue="contoso.com" />
</InputClaims>
```

Output claims

The **OutputClaims** element contains a list of claims returned by the OAuth2 identity provider. You may need to map the name of the claim defined in your policy to the name defined in the identity provider. You can also include claims that aren't returned by the identity provider as long as you set the `DefaultValue` attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

The following example shows the claims returned by the Facebook identity provider:

- The **first_name** claim is mapped to the **givenName** claim.
- The **last_name** claim is mapped to the **surname** claim.

- The **displayName** claim without name-mapping.
- The **email** claim without name mapping.

The technical profile also returns claims that aren't returned by the identity provider:

- The **identityProvider** claim that contains the name of the identity provider.
- The **authenticationSource** claim with a default value of **socialIdpAuthentication**.

```
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="id" />
  <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="first_name" />
  <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="last_name" />
  <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
  <OutputClaim ClaimTypeReferenceId="email" PartnerClaimType="email" />
  <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="facebook.com" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
</OutputClaims>
```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
client_id	Yes	The application identifier of the identity provider.
IdTokenAudience	No	The audience of the id_token. If specified, Azure AD B2C checks whether the token is in a claim returned by the identity provider and is equal to the one specified.
authorization_endpoint	Yes	The URL of the authorization endpoint as per RFC 6749.
AccessTokenEndpoint	Yes	The URL of the token endpoint as per RFC 6749.
ClaimsEndpoint	Yes	The URL of the user information endpoint as per RFC 6749.
AccessTokenResponseFormat	No	The format of the access token endpoint call. For example, Facebook requires an HTTP GET method, but the access token response is in JSON format.
AdditionalRequestQueryParameters	No	Additional request query parameters. For example, you may want to send additional parameters to your identity provider. You can include multiple parameters using comma delimiter.

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimsEndpointAccessTokenName	No	The name of the access token query string parameter. Some identity providers' claims endpoints support GET HTTP request. In this case, the bearer token is sent by using a query string parameter instead of the authorization header.
ClaimsEndpointFormatName	No	The name of the format query string parameter. For example, you can set the name as <code>format</code> in this LinkedIn claims endpoint <code>https://api.linkedin.com/v1/people/~?format=json</code>
ClaimsEndpointFormat	No	The value of the format query string parameter. For example, you can set the value as <code>json</code> in this LinkedIn claims endpoint <code>https://api.linkedin.com/v1/people/~?format=json</code>
ProviderName	No	The name of the identity provider.
response_mode	No	The method that the identity provider uses to send the result back to Azure AD B2C. Possible values: <code>query</code> , <code>form_post</code> (default), or <code>fragment</code> .
scope	No	The scope of the request that is defined according to the OAuth2 identity provider specification. Such as <code>openid</code> , <code>profile</code> , and <code>email</code> .
HttpBinding	No	The expected HTTP binding to the access token and claims token endpoints. Possible values: <code>GET</code> or <code>POST</code> .
ResponseErrorCodeParamName	No	The name of the parameter that contains the error message returned over HTTP 200 (Ok).
ExtraParamsInAccessTokenEndpointResponse	No	Contains the extra parameters that can be returned in the response from AccessTokenEndpoint by some identity providers. For example, the response from AccessTokenEndpoint contains an extra parameter such as <code>openid</code> , which is a mandatory parameter besides the <code>access_token</code> in a ClaimsEndpoint request query string. Multiple parameter names should be escaped and separated by the comma ',' delimiter.

ATTRIBUTE	REQUIRED	DESCRIPTION
ExtraParamsInClaimsEndpointRequest	No	Contains the extra parameters that can be returned in the ClaimsEndpoint request by some identity providers. Multiple parameter names should be escaped and separated by the comma ',' delimiter.
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .
ResolveJsonPathsInJsonTokens	No	Indicates whether the technical profile resolves JSON paths. Possible values: <code>true</code> , or <code>false</code> (default). Use this metadata to read data from a nested JSON element. In an OutputClaim , set the PartnerClaimType to the JSON path element you want to output. For example: <code>firstName.localized</code> , or <code>data.0.to.0.email</code> .

Cryptographic keys

The **CryptographicKeys** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
client_secret	Yes	The client secret of the identity provider application. The cryptographic key is required only if the response_types metadata is set to <code>code</code> . In this case, Azure AD B2C makes another call to exchange the authorization code for an access token. If the metadata is set to <code>id_token</code> , you can omit the cryptographic key.

Redirect URI

When you configure the redirect URL of your identity provider, enter

`https://login.microsoftonline.com/te/tenant/policyId/oauth2/authresp`. Make sure to replace **tenant** with your tenant's name (for example, contosob2c.onmicrosoft.com) and **policyId** with the identifier of your policy (for example, b2c_1a_policy). The redirect URI needs to be in all lowercase.

If you are using the **b2clogin.com** domain instead of **login.microsoftonline.com** Make sure to use b2clogin.com instead of login.microsoftonline.com.

Examples:

- [Add Google+ as an OAuth2 identity provider using custom policies](#)

Define a one-time password technical profile in an Azure AD B2C custom policy

2/10/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for managing the generation and verification of a one-time password. Use a technical profile to generate a code, and then verify that code later.

The one-time password technical profile can also return an error message during code verification. Design the integration with the one-time password by using a **Validation technical profile**. A validation technical profile calls the one-time password technical profile to verify a code. The validation technical profile validates the user-provided data before the user journey continues. With the validation technical profile, an error message is displayed on a self-asserted page.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly that is used by Azure AD B2C:

```
Web.TPEngine.Providers.OneTimePasswordProtocolProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=null
```

The following example shows a one-time password technical profile:

```
<TechnicalProfile Id="VerifyCode">  
  <DisplayName>Validate user input verification code</DisplayName>  
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.OneTimePasswordProtocolProvider, Web.TPEngine,  
  Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  ...
```

Generate code

The first mode of this technical profile is to generate a code. Below are the options that can be configured for this mode.

Input claims

The **InputClaims** element contains a list of claims required to send to the one-time password protocol provider. You can also map the name of your claim to the name defined below.

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
------------------	----------	-------------

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
identifier	Yes	The identifier to identify the user who needs to verify the code later. It is commonly used as the identifier of the destination where the code is delivered to, for example email address or phone number.

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones before sending to the one-time password protocol provider.

Output claims

The **OutputClaims** element contains a list of claims generated by the one-time password protocol provider. You can also map the name of your claim to the name defined below.

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
otpGenerated	Yes	The generated code whose session is managed by Azure AD B2C.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

Metadata

The following settings can be used to configure code generation and maintenance:

ATTRIBUTE	REQUIRED	DESCRIPTION
CodeExpirationInSeconds	No	Time in seconds until code expiration. Minimum: <code>60</code> ; Maximum: <code>1200</code> ; Default: <code>600</code> .
CodeLength	No	Length of the code. The default value is <code>6</code> .
CharacterSet	No	The character set for the code, formatted for use in a regular expression. For example, <code>a-zA-Z0-9</code> . The default value is <code>0-9</code> . The character set must include a minimum of 10 different characters in the set specified.
NumRetryAttempts	No	The number of verification attempts before the code is considered invalid. The default value is <code>5</code> .
Operation	Yes	The operation to be performed. Possible values: <code>GenerateCode</code> , or <code>VerifyCode</code> .

ATTRIBUTE	REQUIRED	DESCRIPTION
ReuseSameCode	No	Whether a duplicate code should be given rather than generating a new code when given code has not expired and is still valid. The default value is <code>false</code> .

Returning error message

There is no error message returned for code generation mode.

Example

The following example `TechnicalProfile` is used for generating a code:

```
<TechnicalProfile Id="GenerateCode">
    <DisplayName>Generate Code</DisplayName>
    <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.OneTimePasswordProtocolProvider,
    Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
    <Metadata>
        <Item Key="Operation">GenerateCode</Item>
        <Item Key="CodeExpirationInSeconds">600</Item>
        <Item Key="CodeLength">6</Item>
        <Item Key="CharacterSet">0-9</Item>
        <Item Key="NumRetryAttempts">5</Item>
        <Item Key="ReuseSameCode">false</Item>
    </Metadata>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="identifier" PartnerClaimType="identifier" />
    </InputClaims>
    <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="otpGenerated" PartnerClaimType="otpGenerated" />
    </OutputClaims>
</TechnicalProfile>
```

Verify code

The second mode of this technical profile is to verify a code. Below are the options that can be configured for this mode.

Input claims

The **InputClaims** element contains a list of claims required to send to the one-time password protocol provider. You can also map the name of your claim to the name defined below.

CLAIMREFERENCEID	REQUIRED	DESCRIPTION
identifier	Yes	The identifier to identify the user who has previously generated a code. It is commonly used as the identifier of the destination where the code is delivered to, for example email address or phone number.
otpToVerify	Yes	The verification code provided by the user.

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones before sending to the one-time password protocol provider.

Output claims

There are no output claims provided during code verification of this protocol provider.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

Metadata

The following settings can be used to configure the error message displayed upon code verification failure:

ATTRIBUTE	REQUIRED	DESCRIPTION
UserMessageIfSessionDoesNotExist	No	The message to display to the user if the code verification session has expired. It is either the code has expired or the code has never been generated for a given identifier.
UserMessageIfMaxRetryAttempted	No	The message to display to the user if they've exceeded the maximum allowed verification attempts.
UserMessageIfInvalidCode	No	The message to display to the user if they've provided an invalid code.

Returning error message

As described in [Metadata](#), you can customize error message shown to the user for different error cases. You can further localize those messages by prefixing the locale, for example:

```
<Item Key="en.UserMessageIfInvalidCode">Wrong code has been entered.</Item>
```

Example

The following example `TechnicalProfile` is used for verifying a code:

```
<TechnicalProfile Id="VerifyCode">
    <DisplayName>Verify Code</DisplayName>
    <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.OneTimePasswordProtocolProvider,
    Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
    <Metadata>
        <Item Key="Operation">VerifyCode</Item>
        <Item Key="UserMessageIfInvalidCode">Wrong code has been entered.</Item>
        <Item Key="UserMessageIfSessionDoesNotExist">Code has expired.</Item>
        <Item Key="UserMessageIfMaxRetryAttempted">You've tried too many times.</Item>
    </Metadata>
    <InputClaims>
        <InputClaim ClaimTypeReferenceId="identifier" PartnerClaimType="identifier" />
        <InputClaim ClaimTypeReferenceId="otpGenerated" PartnerClaimType="otpToVerify" />
    </InputClaims>
</TechnicalProfile>
```

Next steps

See the following article for example of using one-time password technical profile with custom email verification:

- [Custom email verification in Azure Active Directory B2C](#)

Define an OpenID Connect technical profile in an Azure Active Directory B2C custom policy

2/13/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for the [OpenID Connect](#) protocol identity provider. OpenID Connect 1.0 defines an identity layer on top of OAuth 2.0 and represents the state of the art in modern authentication protocols. With an OpenID Connect technical profile, you can federate with an OpenID Connect based identity provider, such as Azure AD. Federating with an identity provider allows users to sign in with their existing social or enterprise identities.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `OpenIdConnect`. For example, the protocol for the **MSA-OIDC** technical profile is `OpenIdConnect`:

```
<TechnicalProfile Id="MSA-OIDC">
  <DisplayName>Microsoft Account</DisplayName>
  <Protocol Name="OpenIdConnect" />
  ...

```

Input claims

The **InputClaims** and **InputClaimsTransformations** elements are not required. But you may want to send additional parameters to your identity provider. The following example adds the **domain_hint** query string parameter with the value of `contoso.com` to the authorization request.

```
<InputClaims>
  <InputClaim ClaimTypeReferenceId="domain_hint" DefaultValue="contoso.com" />
</InputClaims>
```

Output claims

The **OutputClaims** element contains a list of claims returned by the OpenID Connect identity provider. You may need to map the name of the claim defined in your policy to the name defined in the identity provider. You can also include claims that aren't returned by the identity provider, as long as you set the `DefaultValue` attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

The following example shows the claims returned by the Microsoft Account identity provider:

- The **sub** claim that is mapped to the **issuerUserId** claim.
- The **name** claim that is mapped to the **displayName** claim.

- The **email** without name mapping.

The technical profile also returns claims that aren't returned by the identity provider:

- The **identityProvider** claim that contains the name of the identity provider.
- The **authenticationSource** claim with a default value of **socialIdpAuthentication**.

```
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="live.com" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
  <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="sub" />
  <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
  <OutputClaim ClaimTypeReferenceId="email" />
</OutputClaims>
```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
client_id	Yes	The application identifier of the identity provider.
IdTokenAudience	No	The audience of the id_token. If specified, Azure AD B2C checks whether the token is in a claim returned by the identity provider and is equal to the one specified.
METADATA	Yes	A URL that points to a JSON configuration document formatted according to the OpenID Connect Discovery specification, which is also known as a well-known openid configuration endpoint.
ProviderName	No	The name of the identity provider.
response_types	No	The response type according to the OpenID Connect Core 1.0 specification. Possible values: <code>id_token</code> , <code>code</code> , or <code>token</code> .
response_mode	No	The method that the identity provider uses to send the result back to Azure AD B2C. Possible values: <code>query</code> , <code>form_post</code> (default), or <code>fragment</code> .
scope	No	The scope of the request that is defined according to the OpenID Connect Core 1.0 specification. Such as <code>openid</code> , <code>profile</code> , and <code>email</code> .

ATTRIBUTE	REQUIRED	DESCRIPTION
HttpBinding	No	The expected HTTP binding to the access token and claims token endpoints. Possible values: <code>GET</code> or <code>POST</code> .
ValidTokenIssuerPrefixes	No	A key that can be used to sign in to each of the tenants when using a multi-tenant identity provider such as Azure Active Directory.
UsePolicyInRedirectUri	No	Indicates whether to use a policy when constructing the redirect URI. When you configure your application in the identity provider, you need to specify the redirect URI. The redirect URI points to Azure AD B2C, <code>https://{{your-tenant-name}}.b2clogin.com/{{your-tenant-name}}.onmicrosoft.com/oauth2/authresp</code> . If you specify <code>false</code> , you need to add a redirect URI for each policy you use. For example: <code>https://{{your-tenant-name}}.b2clogin.com/{{your-tenant-name}}.onmicrosoft.com/{{policy-name}}/oauth2/authresp</code> .
MarkAsFailureOnStatusCode5xx	No	Indicates whether a request to an external service should be marked as a failure if the Http status code is in the 5xx range. The default is <code>false</code> .
DiscoverMetadataByTokenIssuer	No	Indicates whether the OIDC metadata should be discovered by using the issuer in the JWT token.
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .

Cryptographic keys

The **CryptographicKeys** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
-----------	----------	-------------

ATTRIBUTE	REQUIRED	DESCRIPTION
client_secret	Yes	The client secret of the identity provider application. The cryptographic key is required only if the response_types metadata is set to <code>code</code> . In this case, Azure AD B2C makes another call to exchange the authorization code for an access token. If the metadata is set to <code>id_token</code> you can omit the cryptographic key.

Redirect Uri

When you configure the redirect URI of your identity provider, enter

`https://{your-tenant-name}.b2clogin.com/{your-tenant-name}.onmicrosoft.com/oauth2/authresp`. Make sure to replace `{your-tenant-name}` with your tenant's name. The redirect URI needs to be in all lowercase.

Examples:

- [Add Microsoft Account \(MSA\) as an identity provider using custom policies](#)
- [Sign in by using Azure AD accounts](#)
- [Allow users to sign in to a multi-tenant Azure AD identity provider using custom policies](#)

Define a RESTful technical profile in an Azure Active Directory B2C custom policy

2/24/2020 • 9 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for your own RESTful service. Azure AD B2C sends data to the RESTful service in an input claims collection and receives data back in an output claims collection. With RESTful service integration, you can:

- **Validate user input data** - Prevents malformed data from persisting into Azure AD B2C. If the value from the user is not valid, your RESTful service returns an error message that instructs the user to provide an entry. For example, you can verify that the email address provided by the user exists in your customer's database.
- **Overwrite input claims** - Enables you to reformat values in input claims. For example, if a user enters the first name in all lowercase or all uppercase letters, you can format the name with only the first letter capitalized.
- **Enrich user data** - Enables you to further integrate with corporate line-of-business applications. For example, your RESTful service can receive the user's email address, query the customer's database, and return the user's loyalty number to Azure AD B2C. The return claims can be stored, evaluated in the next Orchestration Steps, or included in the access token.
- **Run custom business logic** - Enables you to send push notifications, update corporate databases, run a user migration process, manage permissions, audit databases, and perform other actions.

Your policy may send input claims to your REST API. The REST API may also return output claims that you can use later in your policy, or it can throw an error message. You can design the integration with the RESTful services in the following ways:

- **Validation technical profile** - A validation technical profile calls the RESTful service. The validation technical profile validates the user-provided data before the user journey continues. With the validation technical profile, an error message is displayed on a self-asserted page and returned in output claims.
- **Claims exchange** - A call is made to the RESTful service through an orchestration step. In this scenario, there is no user-interface to render the error message. If your REST API returns an error, the user is redirected back to the relying party application with the error message.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly that is used by Azure AD B2C:

```
Web.TPEngine.Providers.RestfulProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
```

The following example shows a RESTful technical profile:

```
<TechnicalProfile Id="REST-UserMembershipValidator">
  <DisplayName>Validate user input data and return loyaltyNumber claim</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.RestfulProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  ...

```

Input claims

The **InputClaims** element contains a list of claims to send to the REST API. You can also map the name of your claim to the name defined in the REST API. Following example shows the mapping between your policy and the REST API. The **givenName** claim is sent to the REST API as **firstName**, while **surname** is sent as **lastName**. The **email** claim is set as is.

```
<InputClaims>
  <InputClaim ClaimTypeReferenceId="email" />
  <InputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="firstName" />
  <InputClaim ClaimTypeReferenceId="surname" PartnerClaimType="lastName" />
</InputClaims>
```

The **InputClaimsTransformations** element may contain a collection of **InputClaimsTransformation** elements that are used to modify the input claims or generate new ones before sending to the REST API.

Send a JSON payload

The REST API technical profile allows you to send a complex JSON payload to an endpoint.

To send a complex JSON payload:

1. Build your JSON payload with the [GenerateJson](#) claims transformation.
2. In the REST API technical profile:
 - a. Add an input claims transformation with a reference to the [GenerateJson](#) claims transformation.
 - b. Set the [SendClaimsIn](#) metadata option to [body](#)
 - c. Set the [ClaimUsedForRequestPayload](#) metadata option to the name of the claim containing the JSON payload.
 - d. In the input claim, add a reference to the input claim containing the JSON payload.

The following example [TechnicalProfile](#) sends a verification email by using a third-party email service (in this case, SendGrid).

```

<TechnicalProfile Id="SendGrid">
  <DisplayName>Use SendGrid's email API to send the code the user</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://api.sendgrid.com/v3/mail/send</Item>
    <Item Key="AuthenticationType">Bearer</Item>
    <Item Key="SendClaimsIn">Body</Item>
    <Item Key="ClaimUsedForRequestPayload">sendGridReqBody</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="BearerAuthenticationToken" StorageReferenceId="B2C_1A_SendGridApiKey" />
  </CryptographicKeys>
  <InputClaimsTransformations>
    <InputClaimsTransformation ReferenceId="GenerateSendGridRequestBody" />
  </InputClaimsTransformations>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="sendGridReqBody" />
  </InputClaims>
</TechnicalProfile>

```

Output claims

The **OutputClaims** element contains a list of claims returned by the REST API. You may need to map the name of the claim defined in your policy to the name defined in the REST API. You can also include claims that aren't returned by the REST API identity provider, as long as you set the `DefaultValue` attribute.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

The following example shows the claim returned by the REST API:

- The **MembershipId** claim that is mapped to the **loyaltyNumber** claim name.

The technical profile also returns claims, that aren't returned by the identity provider:

- The **loyaltyNumberIsNew** claim that has a default value set to `true`.

```

<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="loyaltyNumber" PartnerClaimType="MembershipId" />
  <OutputClaim ClaimTypeReferenceId="loyaltyNumberIsNew" DefaultValue="true" />
</OutputClaims>

```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
ServiceUrl	Yes	The URL of the REST API endpoint.

ATTRIBUTE	REQUIRED	DESCRIPTION
AuthenticationType	Yes	The type of authentication being performed by the RESTful claims provider. Possible values: <code>None</code> , <code>Basic</code> , <code>Bearer</code> , or <code>ClientCertificate</code> . The <code>None</code> value indicates that the REST API is not anonymous. The <code>Basic</code> value indicates that the REST API is secured with HTTP basic authentication. Only verified users, including Azure AD B2C, can access your API. The <code>ClientCertificate</code> (recommended) value indicates that the REST API restricts access by using client certificate authentication. Only services that have the appropriate certificates, for example Azure AD B2C, can access your API. The <code>Bearer</code> value indicates that the REST API restricts access using client OAuth2 Bearer token.
SendClaimsIn	No	Specifies how the input claims are sent to the RESTful claims provider. Possible values: <code>Body</code> (default), <code>Form</code> , <code>Header</code> , or <code>QueryString</code> . The <code>Body</code> value is the input claim that is sent in the request body in JSON format. The <code>Form</code> value is the input claim that is sent in the request body in an ampersand '&' separated key value format. The <code>Header</code> value is the input claim that is sent in the request header. The <code>QueryString</code> value is the input claim that is sent in the request query string. The HTTP verbs invoked by each are as follows: <ul style="list-style-type: none"> • <code>Body</code> : POST • <code>Form</code> : POST • <code>Header</code> : GET • <code>QueryString</code> : GET
ClaimsFormat	No	Specifies the format for the output claims. Possible values: <code>Body</code> (default), <code>Form</code> , <code>Header</code> , or <code>QueryString</code> . The <code>Body</code> value is the output claim that is sent in the request body in JSON format. The <code>Form</code> value is the output claim that is sent in the request body in an ampersand '&' separated key value format. The <code>Header</code> value is the output claim that is sent in the request header. The <code>QueryString</code> value is the output claim that is sent in the request query string.
ClaimUsedForRequestPayload	No	Name of a string claim that contains the payload to be sent to the REST API.

ATTRIBUTE	REQUIRED	DESCRIPTION
DebugMode	No	Runs the technical profile in debug mode. Possible values: <code>true</code> , or <code>false</code> (default). In debug mode, the REST API can return more information. See the Returning error message section.
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .
ResolveJsonPathsInJsonTokens	No	Indicates whether the technical profile resolves JSON paths. Possible values: <code>true</code> , or <code>false</code> (default). Use this metadata to read data from a nested JSON element. In an OutputClaim , set the <code>PartnerClaimType</code> to the JSON path element you want to output. For example: <code>firstName.localized</code> , or <code>data.0.to.0.email</code> .

Cryptographic keys

If the type of authentication is set to `None`, the **CryptographicKeys** element is not used.

```
<TechnicalProfile Id="REST-API-SignUp">
  <DisplayName>Validate user's input data and return loyaltyNumber claim</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://your-app-name.azurewebsites.NET/api/identity/signup</Item>
    <Item Key="AuthenticationType">None</Item>
    <Item Key="SendClaimsIn">Body</Item>
  </Metadata>
</TechnicalProfile>
```

If the type of authentication is set to `Basic`, the **CryptographicKeys** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
BasicAuthenticationUsername	Yes	The username that is used to authenticate.
BasicAuthenticationPassword	Yes	The password that is used to authenticate.

The following example shows a technical profile with basic authentication:

```

<TechnicalProfile Id="REST-API-SignUp">
  <DisplayName>Validate user's input data and return loyaltyNumber claim</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://your-app-name.azurewebsites.NET/api/identity/signup</Item>
    <Item Key="AuthenticationType">Basic</Item>
    <Item Key="SendClaimsIn">Body</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="BasicAuthenticationUsername" StorageReferenceId="B2C_1A_B2cRestClientId" />
    <Key Id="BasicAuthenticationPassword" StorageReferenceId="B2C_1A_B2cRestClientSecret" />
  </CryptographicKeys>
</TechnicalProfile>

```

If the type of authentication is set to **ClientCertificate**, the **CryptographicKeys** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClientCertificate	Yes	The X509 certificate (RSA key set) to use to authenticate.

```

<TechnicalProfile Id="REST-API-SignUp">
  <DisplayName>Validate user's input data and return loyaltyNumber claim</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://your-app-name.azurewebsites.NET/api/identity/signup</Item>
    <Item Key="AuthenticationType">ClientCertificate</Item>
    <Item Key="SendClaimsIn">Body</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="ClientCertificate" StorageReferenceId="B2C_1A_B2cRestClientCertificate" />
  </CryptographicKeys>
</TechnicalProfile>

```

If the type of authentication is set to **Bearer**, the **CryptographicKeys** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
BearerAuthenticationToken	No	The OAuth 2.0 Bearer Token.

```

<TechnicalProfile Id="REST-API-SignUp">
  <DisplayName>Validate user's input data and return loyaltyNumber claim</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://your-app-name.azurewebsites.NET/api/identity/signup</Item>
    <Item Key="AuthenticationType">Bearer</Item>
    <Item Key="SendClaimsIn">Body</Item>
  </Metadata>
  <CryptographicKeys>
    <Key Id="BearerAuthenticationToken" StorageReferenceId="B2C_1A_B2cRestClientAccessToken" />
  </CryptographicKeys>
</TechnicalProfile>

```

Returning error message

Your REST API may need to return an error message, such as 'The user was not found in the CRM system'. If an error occurs, the REST API should return an HTTP 409 error message (Conflict response status code) with following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
version	Yes	1.0.0
status	Yes	409
code	No	An error code from the RESTful endpoint provider, which is displayed when <code>DebugMode</code> is enabled.
requestId	No	A request identifier from the RESTful endpoint provider, which is displayed when <code>DebugMode</code> is enabled.
userMessage	Yes	An error message that is shown to the user.
developerMessage	No	The verbose description of the problem and how to fix it, which is displayed when <code>DebugMode</code> is enabled.
moreInfo	No	A URI that points to additional information, which is displayed when <code>DebugMode</code> is enabled.

The following example shows a REST API that returns an error message formatted in JSON:

```
{
  "version": "1.0.0",
  "status": 409,
  "code": "API12345",
  "requestId": "50f0bd91-2ff4-4b8f-828f-00f170519ddb",
  "userMessage": "Message for the user",
  "developerMessage": "Verbose description of problem and how to fix it.",
  "moreInfo": "https://restapi/error/API12345/moreinfo"
}
```

The following example shows a C# class that returns an error message:

```
public class ResponseContent
{
    public string version { get; set; }
    public int status { get; set; }
    public string code { get; set; }
    public string userMessage { get; set; }
    public string developerMessage { get; set; }
    public string requestId { get; set; }
    public string moreInfo { get; set; }
}
```

Next steps

See the following articles for examples of using a RESTful technical profile:

- Integrate REST API claims exchanges in your Azure AD B2C user journey as validation of user input
- Secure your RESTful services by using HTTP basic authentication
- Secure your RESTful service by using client certificates
- Walkthrough: Integrate REST API claims exchanges in your Azure AD B2C user journey as validation on user input

Define a SAML technical profile in an Azure Active Directory B2C custom policy

2/13/2020 • 10 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Azure Active Directory B2C (Azure AD B2C) provides support for the SAML 2.0 identity provider. This article describes the specifics of a technical profile for interacting with a claims provider that supports this standardized protocol. With a SAML technical profile you can federate with a SAML-based identity provider, such as [ADFS](#) and [Salesforce](#). This federation allows your users to sign in with their existing social or enterprise identities.

Metadata exchange

Metadata is information used in the SAML protocol to expose the configuration of a SAML party, such as a service provider or identity provider. Metadata defines the location of the services, such as sign-in and sign-out, certificates, sign-in method, and more. The identity provider uses the metadata to know how to communicate with Azure AD B2C. The metadata is configured in XML format and may be signed with a digital signature so that the other party can validate the integrity of the metadata. When Azure AD B2C federates with a SAML identity provider, it acts as a service provider initiating a SAML request and waiting for a SAML response. And, in some cases, accepts unsolicited SAML authentication, which is also known as identity provider initiated.

The metadata can be configured in both parties as "Static Metadata" or "Dynamic Metadata". In static mode, you copy the entire metadata from one party and set it in the other party. In dynamic mode, you set the URL to the metadata while the other party reads the configuration dynamically. The principles are the same, you set the metadata of the Azure AD B2C technical profile in your identity provider and set the metadata of the identity provider in Azure AD B2C.

Each SAML identity provider has different steps to expose and set the service provider, in this case Azure AD B2C, and set the Azure AD B2C metadata in the identity provider. Look at your identity provider's documentation for guidance on how to do so.

The following example shows a URL address to the SAML metadata of an Azure AD B2C technical profile:

```
https://your-tenant-name.b2clogin.com/your-tenant-name/your-policy/samlp/metadata?idptp=your-technical-profile
```

Replace the following values:

- **your-tenant-name** with your tenant name, such as fabrikam.b2clogin.com.
- **your-policy** with your policy name. Use the policy where you configure the SAML provider technical profile, or a policy that inherits from that policy.
- **your-technical-profile** with your SAML identity provider technical profile name.

Digital signing certificates exchange

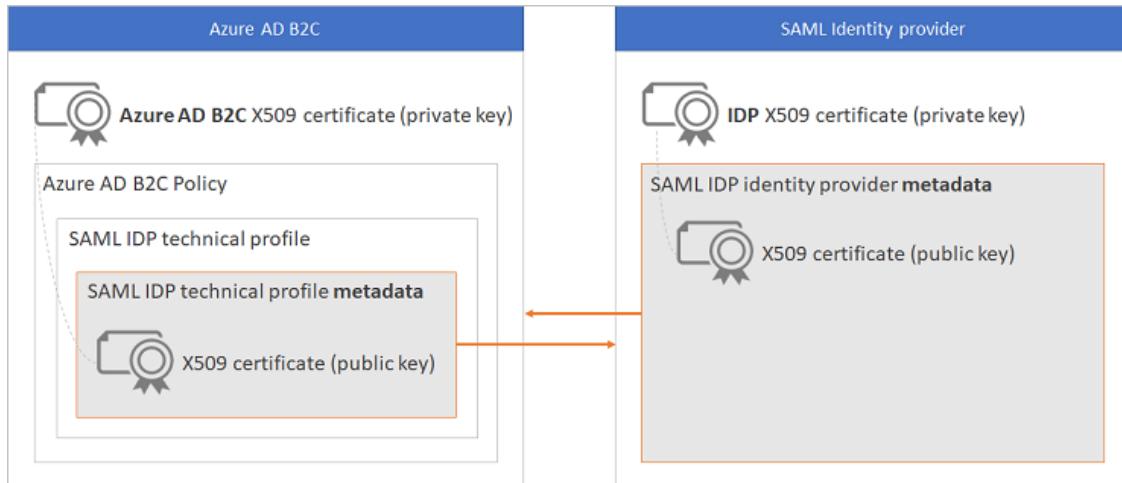
To build a trust between Azure AD B2C and your SAML identity provider, you need to provide a valid X509 certificate with the private key. You upload the certificate with the private key (.pfx file) to the Azure AD B2C policy key store. Azure AD B2C digitally signs the SAML sign-in request using the certificate that you provide.

The certificate is used in the following ways:

- Azure AD B2C generates and signs a SAML request, using the Azure AD B2C private key of the certificate. The SAML request is sent to the identity provider, which validates the request using Azure AD B2C public key of the certificate. The Azure AD B2C public certificate is accessible through technical profile metadata. Alternatively, you can manually upload the .cer file to your SAML identity provider.
- The identity provider signs the data sent to Azure AD B2C using the identity provider's private key of the certificate. Azure AD B2C validates the data using the identity provider's public certificate. Each identity provider has different steps for setup, look at your identity provider's documentation for guidance on how to do so. In Azure AD B2C, your policy needs access to the certificate public key using the identity provider's metadata.

A self-signed certificate is acceptable for most scenarios. For production environments, it is recommended to use an X509 certificate that is issued by a certificate authority. Also, as described later in this document, for a non-production environment you can disable the SAML signing on both sides.

The following diagram shows the metadata and certificate exchange:



Digital encryption

To encrypt the SAML response assertion, the identity provider always uses a public key of an encryption certificate in an Azure AD B2C technical profile. When Azure AD B2C needs to decrypt the data, it uses the private portion of the encryption certificate.

To encrypt the SAML response assertion:

1. Upload a valid X509 certificate with the private key (.pfx file) to the Azure AD B2C policy key store.
2. Add a **CryptographicKey** element with an identifier of `Sam1AssertionDecryption` to the technical profile **CryptographicKeys** collection. Set the **StorageReferenceId** to the name of the policy key you created in step 1.
3. Set the technical profile metadata **WantsEncryptedAssertions** to `true`.
4. Update the identity provider with the new Azure AD B2C technical profile metadata. You should see the **KeyDescriptor** with the **use** property set to `encryption` containing the public key of your certificate.

The following example shows the Azure AD B2C technical profile encryption section of the metadata:

```
<KeyDescriptor use="encryption">
  <KeyInfo xmlns="https://www.w3.org/2000/09/xmldsig#">
    <X509Data>
      <X509Certificate>valid certificate</X509Certificate>
    </X509Data>
  </KeyInfo>
</KeyDescriptor>
```

Protocol

The **Name** attribute of the Protocol element needs to be set to `SAML2`.

Output claims

The **OutputClaims** element contains a list of claims returned by the SAML identity provider under the **AttributeStatement** section. You may need to map the name of the claim defined in your policy to the name defined in the identity provider. You can also include claims that aren't returned by the identity provider as long as you set the **DefaultValue** attribute.

To read the SAML assertion **NamedId** in **Subject** as a normalized claim, set the claim **PartnerClaimType** to **assertionSubjectName**. Make sure the **NamedId** is the first value in assertion XML. When you define more than one assertion, Azure AD B2C picks the subject value from the last assertion.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

The following example shows the claims returned by the Facebook identity provider:

- The **issuerUserId** claim is mapped to the **assertionSubjectName** claim.
- The **first_name** claim is mapped to the **givenName** claim.
- The **last_name** claim is mapped to the **surname** claim.
- The **displayName** claim without name mapping.
- The **email** claim without name mapping.

The technical profile also returns claims that aren't returned by the identity provider:

- The **identityProvider** claim that contains the name of the identity provider.
- The **authenticationSource** claim with a default value of **socialIdpAuthentication**.

```
<OutputClaims>
  <OutputClaim ClaimTypeReferenceId="issuerUserId" PartnerClaimType="assertionSubjectName" />
  <OutputClaim ClaimTypeReferenceId="givenName" PartnerClaimType="first_name" />
  <OutputClaim ClaimTypeReferenceId="surname" PartnerClaimType="last_name" />
  <OutputClaim ClaimTypeReferenceId="displayName" PartnerClaimType="name" />
  <OutputClaim ClaimTypeReferenceId="email" />
  <OutputClaim ClaimTypeReferenceId="identityProvider" DefaultValue="contoso.com" />
  <OutputClaim ClaimTypeReferenceId="authenticationSource" DefaultValue="socialIdpAuthentication" />
</OutputClaims>
```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
PartnerEntity	Yes	URL of the metadata of the SAML identity provider. Copy the identity provider metadata and add it inside the CDATA element <code><![CDATA[Your IDP metadata]]></code>

ATTRIBUTE	REQUIRED	DESCRIPTION
WantsSignedRequests	No	<p>Indicates whether the technical profile requires all of the outgoing authentication requests to be signed. Possible values: <code>true</code> or <code>false</code>. The default value is <code>true</code>. When the value is set to <code>true</code>, the SamlMessageSigning cryptographic key needs to be specified and all of the outgoing authentication requests are signed. If the value is set to <code>false</code>, the SigAlg and Signature parameters (query string or post parameter) are omitted from the request. This metadata also controls the metadata AuthnRequestsSigned attribute, which is output in the metadata of the Azure AD B2C technical profile that is shared with the identity provider. Azure AD B2C doesn't sign the request if the value of WantsSignedRequests in the technical profile metadata is set to <code>false</code> and the identity provider metadata WantAuthnRequestsSigned is set to <code>false</code> or not specified.</p>
XmlSignatureAlgorithm	No	<p>The method that Azure AD B2C uses to sign the SAML request. This metadata controls the value of the SigAlg parameter (query string or post parameter) in the SAML request. Possible values: <code>Sha256</code>, <code>Sha384</code>, <code>Sha512</code>, or <code>Sha1</code>. Make sure you configure the signature algorithm on both sides with same value. Use only the algorithm that your certificate supports.</p>
WantsSignedAssertions	No	<p>Indicates whether the technical profile requires all incoming assertions to be signed. Possible values: <code>true</code> or <code>false</code>. The default value is <code>true</code>. If the value is set to <code>true</code>, all assertions section <code>saml:Assertion</code> sent by the identity provider to Azure AD B2C must be signed. If the value is set to <code>false</code>, the identity provider shouldn't sign the assertions, but even if it does, Azure AD B2C won't validate the signature. This metadata also controls the metadata flag WantsAssertionsSigned, which is output in the metadata of the Azure AD B2C technical profile that is shared with the identity provider. If you disable the assertions validation, you also may want to disable the response signature validation (for more information, see ResponsesSigned).</p>

ATTRIBUTE	REQUIRED	DESCRIPTION
ResponsesSigned	No	Possible values: <code>true</code> or <code>false</code> . The default value is <code>true</code> . If the value is set to <code>false</code> , the identity provider shouldn't sign the SAML response, but even if it does, Azure AD B2C won't validate the signature. If the value is set to <code>true</code> , the SAML response sent by the identity provider to Azure AD B2C is signed and must be validated. If you disable the SAML response validation, you also may want to disable the assertion signature validation (for more information, see WantsSignedAssertions).
WantsEncryptedAssertions	No	Indicates whether the technical profile requires all incoming assertions to be encrypted. Possible values: <code>true</code> or <code>false</code> . The default value is <code>false</code> . If the value is set to <code>true</code> , assertions sent by the identity provider to Azure AD B2C must be signed and the SamlAssertionDecryption cryptographic key needs to be specified. If the value is set to <code>true</code> , the metadata of the Azure AD B2C technical profile includes the encryption section. The identity provider reads the metadata and encrypts the SAML response assertion with the public key that is provided in the metadata of the Azure AD B2C technical profile. If you enable the assertions encryption, you also may need to disable the response signature validation (for more information, see ResponsesSigned).
IdpInitiatedProfileEnabled	No	Indicates whether a single sign-on session profile is enabled that was initiated by a SAML identity provider profile. Possible values: <code>true</code> or <code>false</code> . The default is <code>false</code> . In the flow initiated by the identity provider, the user is authenticated externally and an unsolicited response is sent to Azure AD B2C, which then consumes the token, executes orchestration steps, and then sends a response to the relying party application.
NameIdPolicyFormat	No	Specifies constraints on the name identifier to be used to represent the requested subject. If omitted, any type of identifier supported by the identity provider for the requested subject can be used. For example, <code>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</code> . NameIdPolicyFormat can be used with NameIdPolicyAllowCreate . Look at your identity provider's documentation for guidance about which name ID policies are supported.

ATTRIBUTE	REQUIRED	DESCRIPTION
NameIdPolicyAllowCreate	No	When using NameIdPolicyFormat , you can also specify the <code>AllowCreate</code> property of NameIDPolicy . The value of this metadata is <code>true</code> or <code>false</code> to indicate whether the identity provider is allowed to create a new account during the sign-in flow. Look at your identity provider's documentation for guidance on how to do so.
AuthenticationRequestExtensions	No	Optional protocol message extension elements that are agreed on between Azure AD BC and the identity provider. The extension is presented in XML format. You add the XML data inside the CDATA element <code><![CDATA[Your IDP metadata]]></code> . Check your identity provider's documentation to see if the extensions element is supported.
IncludeAuthnContextClassReferences	No	Specifies one or more URI references identifying authentication context classes. For example, to allow a user to sign in with username and password only, set the value to <code>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</code> . To allow sign-in through username and password over a protected session (SSL/TLS), specify <code>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</code> . Look at your identity provider's documentation for guidance about the AuthnContextClassRef URIs that are supported. Specify multiple URLs as a comma-delimited list.
IncludeKeyInfo	No	Indicates whether the SAML authentication request contains the public key of the certificate when the binding is set to <code>HTTP-POST</code> . Possible values: <code>true</code> or <code>false</code> .
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .

Cryptographic keys

The **CryptographicKeys** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
SamlMessageSigning	Yes	The X509 certificate (RSA key set) to use to sign SAML messages. Azure AD B2C uses this key to sign the requests and send them to the identity provider.

ATTRIBUTE	REQUIRED	DESCRIPTION
SamlAssertionDecryption	Yes	The X509 certificate (RSA key set) to use to decrypt SAML messages. This certificate should be provided by the identity provider. Azure AD B2C uses this certificate to decrypt the data sent by the identity provider.
MetadataSigning	No	The X509 certificate (RSA key set) to use to sign SAML metadata. Azure AD B2C uses this key to sign the metadata.

Next steps

See the following articles for examples of working with SAML identity providers in Azure AD B2C:

- [Add ADFS as a SAML identity provider using custom policies](#)
- [Sign in by using Salesforce accounts via SAML](#)

Define a self-asserted technical profile in an Azure Active Directory B2C custom policy

2/17/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

All interactions in Azure Active Directory B2C (Azure AD B2C) where the user is expected to provide input are self-asserted technical profiles. For example, a sign-up page, sign-in page, or password reset page.

Protocol

The **Name** attribute of the **Protocol** element needs to be set to `Proprietary`. The **handler** attribute must contain the fully qualified name of the protocol handler assembly that is used by Azure AD B2C, for self-asserted:

```
Web.TPEngine.Providers.SelfAssertedAttributeProvider, Web.TPEngine, Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=null
```

The following example shows a self-asserted technical profile for email sign-up:

```
<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">  
  <DisplayName>Email signup</DisplayName>  
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider,  
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
```

Input claims

In a self-asserted technical profile, you can use the **InputClaims** and **InputClaimsTransformations** elements to prepopulate the value of the claims that appear on the self-asserted page (display claims). For example, in the edit profile policy, the user journey first reads the user profile from the Azure AD B2C directory service, then the self-asserted technical profile sets the input claims with the user data stored in the user profile. These claims are collected from the user profile and then presented to the user who can then edit the existing data.

```
<TechnicalProfile Id="SelfAsserted-ProfileUpdate">  
  ...  
  <InputClaims>  
    <InputClaim ClaimTypeReferenceId="alternativeSecurityId" />  
    <InputClaim ClaimTypeReferenceId="userPrincipalName" />  
    <InputClaim ClaimTypeReferenceId="givenName" />  
    <InputClaim ClaimTypeReferenceId="surname" />  
  </InputClaims>
```

Display claims

The display claims feature is currently in [preview](#).

The **DisplayClaims** element contains a list of claims to be presented on the screen for collecting data from

the user. To prepopulate the values of display claims, use the input claims that were previously described. The element may also contain a default value.

The order of the claims in **DisplayClaims** specifies the order in which Azure AD B2C renders the claims on the screen. To force the user to provide a value for a specific claim, set the **Required** attribute of the **DisplayClaim** element to `true`.

The **ClaimType** element in the **DisplayClaims** collection needs to set the **UserInputType** element to any user input type supported by Azure AD B2C. For example, `TextBox` or `DropdownSingleSelect`.

Add a reference to a DisplayControl

In the display claims collection, you can include a reference to a **DisplayControl** that you've created. A display control is a user interface element that has special functionality and interacts with the Azure AD B2C back-end service. It allows the user to perform actions on the page that invoke a validation technical profile at the back end. For example, verifying an email address, phone number, or customer loyalty number.

The following example `TechnicalProfile` illustrates the use of display claims with display controls.

- The first display claim makes a reference to the `emailVerificationControl` display control which collects and verifies the email address.
- The fifth display claim makes a reference to the `phoneVerificationControl` display control which collects and verifies a phone number.
- The other display claims are ClaimTypes to be collected from the user.

```
<TechnicalProfile Id="Id">
  <DisplayClaims>
    <DisplayClaim DisplayControlReferenceId="emailVerificationControl" />
    <DisplayClaim ClaimTypeReferenceId="displayName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="givenName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="surName" Required="true" />
    <DisplayClaim DisplayControlReferenceId="phoneVerificationControl" />
    <DisplayClaim ClaimTypeReferenceId="newPassword" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
  </DisplayClaims>
</TechnicalProfile>
```

As mentioned, a display claim with a reference to a display control may run its own validation, for example verifying the email address. In addition, the self-asserted page supports using a validation technical profile to validate the entire page, including any user input (claim types or display controls), before moving on to the next orchestration step.

Combine usage of display claims and output claims carefully

If you specify one or more **DisplayClaim** elements in a self-asserted technical profile, you must use a **DisplayClaim** for every claim that you want to display on-screen and collect from the user. No output claims are displayed by a self-asserted technical profile that contains at least one display claim.

Consider the following example in which an `age` claim is defined as an **output** claim in a base policy. Before adding any display claims to the self-asserted technical profile, the `age` claim is displayed on the screen for data collection from the user:

```
<TechnicalProfile Id="id">
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="age" />
  </OutputClaims>
</TechnicalProfile>
```

If a leaf policy that inherits that base subsequently specifies `officeNumber` as a **display** claim:

```
<TechnicalProfile Id="id">
  <DisplayClaims>
    <DisplayClaim ClaimTypeReferenceId="officeNumber" />
  </DisplayClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="officeNumber" />
  </OutputClaims>
</TechnicalProfile>
```

The `age` claim in the base policy is no longer presented on the screen to the user - it's effectively "hidden." To display the `age` claim and collect the age value from the user, you must add an `age` **DisplayClaim**.

Output claims

The **OutputClaims** element contains a list of claims to be returned to the next orchestration step. The **DefaultValue** attribute takes effect only if the claim has never been set. If it was set in a previous orchestration step, the default value does not take effect even if the user leaves the value empty. To force the use of a default value, set the **AlwaysUseDefaultValue** attribute to `true`.

NOTE

In previous versions of the Identity Experience Framework (IEF), output claims were used to collect data from the user. To collect data from the user, use a **DisplayClaims** collection instead.

The **OutputClaimsTransformations** element may contain a collection of **OutputClaimsTransformation** elements that are used to modify the output claims or generate new ones.

When you should use output claims

In a self-asserted technical profile, the output claims collection returns the claims to the next orchestration step.

You should use output claims when:

- **Claims are output by output claims transformation.**
- **Setting a default value in an output claim** without collecting data from the user or returning the data from the validation technical profile. The `LocalAccountSignUpWithLogonEmail` self-asserted technical profile sets the **executed-SelfAsserted-Input** claim to `true`.
- **A validation technical profile returns the output claims** - Your technical profile may call a validation technical profile that returns some claims. You may want to bubble up the claims and return them to the next orchestration steps in the user journey. For example, when signing in with a local account, the self-asserted technical profile named `SelfAsserted-LocalAccountSignin-Email` calls the validation technical profile named `login-NonInteractive`. This technical profile validates the user credentials and also returns the user profile. Such as 'userPrincipalName', 'displayName', 'givenName' and 'surName'.
- **A display control returns the output claims** - Your technical profile may have a reference to a **display control**. The display control returns some claims, such as the verified email address. You may want to bubble up the claims and return them to the next orchestration steps in the user journey. The display control feature is currently in **preview**.

The following example demonstrates the use of a self-asserted technical profile that uses both display claims and output claims.

```

<TechnicalProfile Id="LocalAccountSignUpWithLogonEmail">
  <DisplayName>Email signup</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.Providers.SelfAssertedAttributeProvider,
  Web.TPEngine, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="IpAddressClaimReferenceId">IpAddress</Item>
    <Item Key="ContentDefinitionReferenceId">api.localaccountsSignup</Item>
    <Item Key="language.button_continue">Create</Item>
  </Metadata>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="email" />
  </InputClaims>
  <DisplayClaims>
    <DisplayClaim DisplayControlReferenceId="emailVerificationControl" />
    <DisplayClaim DisplayControlReferenceId="SecondaryEmailVerificationControl" />
    <DisplayClaim ClaimTypeReferenceId="displayName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="givenName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="surName" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="newPassword" Required="true" />
    <DisplayClaim ClaimTypeReferenceId="reenterPassword" Required="true" />
  </DisplayClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email" Required="true" />
    <OutputClaim ClaimTypeReferenceId="objectId" />
    <OutputClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" DefaultValue="true" />
    <OutputClaim ClaimTypeReferenceId="authenticationSource" />
    <OutputClaim ClaimTypeReferenceId="newUser" />
  </OutputClaims>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="AAD-UserWriteUsingLogonEmail" />
  </ValidationTechnicalProfiles>
  <UseTechnicalProfileForSessionManagement ReferenceId="SM-AAD" />
</TechnicalProfile>

```

Persist claims

If the **PersistedClaims** element is absent, the self-assigned technical profile doesn't persist the data to Azure AD B2C. Instead, a call is made to a validation technical profile that's responsible for persisting the data. For example, the sign-up policy uses the `LocalAccountSignUpWithLogonEmail` self-assigned technical profile to collect the new user profile. The `LocalAccountSignUpWithLogonEmail` technical profile calls the validation technical profile to create the account in Azure AD B2C.

Validation technical profiles

A validation technical profile is used for validating some or all of the output claims of the referencing technical profile. The input claims of the validation technical profile must appear in the output claims of the self-assigned technical profile. The validation technical profile validates the user input and can return an error to the user.

The validation technical profile can be any technical profile in the policy, such as [Azure Active Directory](#) or a [REST API](#) technical profiles. In the previous example, the `LocalAccountSignUpWithLogonEmail` technical profile validates that the `signinName` does not exist in the directory. If not, the validation technical profile creates a local account and returns the `objectId`, `authenticationSource`, `newUser`. The `SelfAsserted-LocalAccountSignin-Email` technical profile calls the `login-NonInteractive` validation technical profile to validate the user credentials.

You can also call a REST API technical profile with your business logic, overwrite input claims, or enrich user data by further integrating with corporate line-of-business application. For more information, see [Validation technical profile](#)

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
setting.operatingMode ¹	No	For a sign-in page, this property controls the behavior of the username field, such as input validation and error messages. Expected values: <code>Username</code> or <code>Email</code> .
AllowGenerationOfClaimsWithNullValues	No	Allow to generate a claim with null value. For example, in a case user doesn't select a checkbox.
ContentDefinitionReferenceId	Yes	The identifier of the content definition associated with this technical profile.
EnforceEmailVerification	No	For sign-up or profile edit, enforces email verification. Possible values: <code>true</code> (default), or <code>false</code> .
setting.retryLimit	No	Controls the number of times a user can try to provide the data that is checked against a validation technical profile . For example, a user tries to sign-up with an account that already exists and keeps trying until the limit reached.
SignUpTarget ¹	No	The signup target exchange identifier. When the user clicks the sign-up button, Azure AD B2C executes the specified exchange identifier.
setting.showCancelButton	No	Displays the cancel button. Possible values: <code>true</code> (default), or <code>false</code>
setting.showContinueButton	No	Displays the continue button. Possible values: <code>true</code> (default), or <code>false</code>
setting.showSignupLink ²	No	Displays the sign-up button. Possible values: <code>true</code> (default), or <code>false</code>
setting.forgotPasswordLinkLocation ²	No	Displays the forgot password link. Possible values: <code>AfterInput</code> (default) the link is displayed at the bottom of the page, or <code>None</code> removes the forgot password link.
IncludeClaimResolvingInClaimsHandling	No	For input and output claims, specifies whether claims resolution is included in the technical profile. Possible values: <code>true</code> , or <code>false</code> (default). If you want to use a claims resolver in the technical profile, set this to <code>true</code> .

Notes:

1. Available for content definition [DataUri](#) type of `unifiedssp`, or `unifiedssd`.
2. Available for content definition [DataUri](#) type of `unifiedssp`, or `unifiedssd`. [Page layout version](#) 1.1.0 and above.

Cryptographic keys

The **CryptographicKeys** element is not used.

Single sign-on session management in Azure Active Directory B2C

2/28/2020 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Single sign-on (SSO) session management in Azure Active Directory B2C (Azure AD B2C) enables an administrator to control interaction with a user after the user has already authenticated. For example, the administrator can control whether the selection of identity providers is displayed, or whether local account details need to be entered again. This article describes how to configure the SSO settings for Azure AD B2C.

SSO session management has two parts. The first deals with the user's interactions directly with Azure AD B2C and the other deals with the user's interactions with external parties such as Facebook. Azure AD B2C does not override or bypass SSO sessions that might be held by external parties. Rather the route through Azure AD B2C to get to the external party is "remembered", avoiding the need to reprompt the user to select their social or enterprise identity provider. The ultimate SSO decision remains with the external party.

SSO session management uses the same semantics as any other technical profile in custom policies. When an orchestration step is executed, the technical profile associated with the step is queried for a

`<UseTechnicalProfileForSessionManagement ReferenceId="{ID}" />` reference. If one exists, the referenced SSO session provider is then checked to see if the user is a session participant. If so, the SSO session provider is used to repopulate the session. Similarly, when the execution of an orchestration step is complete, the provider is used to store information in the session if an SSO session provider has been specified.

Azure AD B2C has defined a number of SSO session providers that can be used:

- NoopSSOSessionProvider
- DefaultSSOSessionProvider
- ExternalLoginSSOSessionProvider
- SamISSOSessionProvider

SSO management classes are specified using the

`<UseTechnicalProfileForSessionManagement ReferenceId="{ID}" />` element of a technical profile.

Input claims

The `InputClaims` element is empty or absent.

Persisted claims

Claims that need to be returned to the application or used by preconditions in subsequent steps, should be stored in the session or augmented by a read from the user's profile in the directory. Using persisted claims ensures that your authentication journeys won't fail on missing claims. To add claims in the session, use the `<PersistedClaims>` element of the technical profile. When the provider is used to repopulate the session, the persisted claims are added to the claims bag.

Output claims

The `<OutputClaims>` is used for retrieving claims from the session.

Session providers

NoopSSOSessionProvider

As the name dictates, this provider does nothing. This provider can be used for suppressing SSO behavior for a specific technical profile. The following `SM-Noop` technical profile is included in the [custom policy starter pack](#).

```
<TechnicalProfile Id="SM-Noop">
  <DisplayName>Noop Session Management Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.NoopSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
</TechnicalProfile>
```

DefaultSSOSessionProvider

This provider can be used for storing claims in a session. This provider is typically referenced in a technical profile used for managing local accounts. The following `SM-AAD` technical profile is included in the [custom policy starter pack](#).

```
<TechnicalProfile Id="SM-AAD">
  <DisplayName>Session Mananagement Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.DefaultSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <PersistedClaims>
    <PersistedClaim ClaimTypeReferenceId="objectId" />
    <PersistedClaim ClaimTypeReferenceId="signInName" />
    <PersistedClaim ClaimTypeReferenceId="authenticationSource" />
    <PersistedClaim ClaimTypeReferenceId="identityProvider" />
    <PersistedClaim ClaimTypeReferenceId="newUser" />
    <PersistedClaim ClaimTypeReferenceId="executed-SelfAsserted-Input" />
  </PersistedClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="objectIdFromSession" DefaultValue="true"/>
  </OutputClaims>
</TechnicalProfile>
```

The following `SM-MFA` technical profile is included in the [custom policy starter pack](#)

`SocialAndLocalAccountsWithMfa`. This technical profile manages the multi-factor authentication session.

```
<TechnicalProfile Id="SM-MFA">
  <DisplayName>Session Mananagement Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.DefaultSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <PersistedClaims>
    <PersistedClaim ClaimTypeReferenceId="Verified.strongAuthenticationPhoneNumber" />
  </PersistedClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="isActiveMFASession" DefaultValue="true"/>
  </OutputClaims>
</TechnicalProfile>
```

ExternalLoginSSOSessionProvider

This provider is used to suppress the "choose identity provider" screen. It is typically referenced in a technical profile configured for an external identity provider, such as Facebook. The following `SM-SocialLogin` technical profile is included in the [custom policy starter pack](#).

```

<TechnicalProfile Id="SM-SocialLogin">
  <DisplayName>Session Management Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.ExternalLoginSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="AlwaysFetchClaimsFromProvider">true</Item>
  </Metadata>
  <PersistedClaims>
    <PersistedClaim ClaimTypeReferenceId="AlternativeSecurityId" />
  </PersistedClaims>
</TechnicalProfile>

```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
AlwaysFetchClaimsFromProvider	No	Not currently used, can be ignored.

SamlSSOSessionProvider

This provider is used for managing the Azure AD B2C SAML sessions between a relying party application or a federated SAML identity provider. When using the SSO provider for storing a SAML identity provider session, the `IncludeSessionIndex` and `RegisterServiceProviders` must be set to `false`. The following `SM-Saml-idp` technical profile is used by the [SAML technical profile](#).

```

<TechnicalProfile Id="SM-Saml-idp">
  <DisplayName>Session Management Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.SamlSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />
  <Metadata>
    <Item Key="IncludeSessionIndex">false</Item>
    <Item Key="RegisterServiceProviders">false</Item>
  </Metadata>
</TechnicalProfile>

```

When using the provider for storing the B2C SAML session, the `IncludeSessionIndex` and `RegisterServiceProviders` must set to `true`. SAML session logout requires the `SessionIndex` and `NameID` to complete.

The following `SM-Saml-sp` technical profile is used by [SAML issuer technical profile](#)

```

<TechnicalProfile Id="SM-Saml-sp">
  <DisplayName>Session Management Provider</DisplayName>
  <Protocol Name="Proprietary" Handler="Web.TPEngine.SSO.SamlSSOSessionProvider, Web.TPEngine,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
</TechnicalProfile>

```

Metadata

ATTRIBUTE	REQUIRED	DESCRIPTION
IncludeSessionIndex	No	Indicates to the provider that the session index should be stored. Possible values: <code>true</code> (default), or <code>false</code> .

ATTRIBUTE	REQUIRED	DESCRIPTION
RegisterServiceProviders	No	Indicates that the provider should register all SAML service providers that have been issued an assertion. Possible values: <code>true</code> (default), or <code>false</code> .

Define a validation technical profile in an Azure Active Directory B2C custom policy

12/9/2019 • 4 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

A validation technical profile is an ordinary technical profile from any protocol, such as [Azure Active Directory](#) or a [REST API](#). The validation technical profile returns output claims or returns an HTTP 409 error message (Conflict response status code), with the following data:

```
{  
    "version": "1.0.0",  
    "status": 409,  
    "userMessage": "Your error message"  
}
```

Claims that are returned from a validation technical profile are added back to the claims bag. You can use those claims in the next validation technical profiles.

Validation technical profiles are executed in the sequence that they appear in the **ValidationTechnicalProfiles** element. You can configure in a validation technical profile whether the execution of any subsequent validation technical profiles should continue if the validation technical profile raises an error or is successful.

A validation technical profile can be conditionally executed based on preconditions defined in the **ValidationTechnicalProfile** element. For example, you can check whether a specific claims exists, or if a claim is equal or not to the specified value.

A self-asserted technical profile may define a validation technical profile to be used for validating some or all of its output claims. All of the input claims of the referenced technical profile must appear in the output claims of the referencing validation technical profile.

NOTE

Only self-asserted technical profiles can use validation technical profiles. If you need to validate the output claims from non-self-asserted technical profiles, consider using an additional orchestration step in your user journey to accommodate the technical profile in charge of the validation.

ValidationTechnicalProfiles

The **ValidationTechnicalProfiles** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
---------	-------------	-------------

ELEMENT	OCCURRENCES	DESCRIPTION
ValidationTechnicalProfile	1:n	A technical profile to be used for validating some or all of the output claims of the referencing technical profile.

The **ValidationTechnicalProfile** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Referenceld	Yes	An identifier of a technical profile already defined in the policy or parent policy.
ContinueOnError	No	Indicating whether validation of any subsequent validation technical profiles should continue if this validation technical profile raises an error. Possible values: <code>true</code> or <code>false</code> (default, processing of further validation profiles will stop and an error returned).
ContinueOnSuccess	No	Indicating whether validation of any subsequent validation profiles should continue if this validation technical profile succeeds. Possible values: <code>true</code> or <code>false</code> . The default is <code>true</code> , meaning that the processing of further validation profiles will continue.

The **ValidationTechnicalProfile** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
Preconditions	0:1	A list of preconditions that must be satisfied for the validation technical profile to execute.

The **Precondition** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
<code>Type</code>	Yes	The type of check or query to perform for the precondition. Either <code>ClaimsExist</code> is specified to ensure that actions should be performed if the specified claims exist in the user's current claim set, or <code>ClaimEquals</code> is specified that the actions should be performed if the specified claim exists and its value is equal to the specified value.
<code>ExecuteActionsIf</code>	Yes	Indicates whether the actions in the precondition should be performed if the test is true or false.

The **Precondition** element contains following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Value	1:n	The data that is used by the check. If the type of this check is <code>ClaimsExist</code> , this field specifies a <code>ClaimTypeReferenceId</code> to query for. If the type of check is <code>ClaimEquals</code> , this field specifies a <code>ClaimTypeReferenceId</code> to query for. While another value element contains the value to be checked.
Action	1:1	The action that should be taken if the precondition check within an orchestration step is true. The value of the Action is set to <code>SkipThisValidationTechnicalProfile</code> . Specifies that the associated validation technical profile should not be executed.

Example

Following example uses these validation technical profiles:

1. The first validation technical profile checks user credentials and doesn't continue if an error occurs, such as invalid username or bad password.
2. The next validation technical profile, doesn't execute if the `userType` claim does not exist, or if the value of the `userType` is `Partner`. The validation technical profile tries to read the user profile from the internal customer database and continue if an error occurs, such as REST API service not available, or any internal error.
3. The last validation technical profile, doesn't execute if the `userType` claim has not existed, or if the value of the `userType` is `Customer`. The validation technical profile tries to read the user profile from the internal partner database and continues if an error occurs, such as REST API service not available, or any internal error.

```
<ValidationTechnicalProfiles>
  <ValidationTechnicalProfile ReferenceId="login-NonInteractive" ContinueOnError="false" />
  <ValidationTechnicalProfile ReferenceId="REST-ReadProfileFromCustomertsDatabase" ContinueOnError="true" >
    <Preconditions>
      <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
        <Value>userType</Value>
        <Action>SkipThisValidationTechnicalProfile</Action>
      </Precondition>
      <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
        <Value>userType</Value>
        <Value>Partner</Value>
        <Action>SkipThisValidationTechnicalProfile</Action>
      </Precondition>
    </Preconditions>
  </ValidationTechnicalProfile>
  <ValidationTechnicalProfile ReferenceId="REST-ReadProfileFromPartnersDatabase" ContinueOnError="true" >
    <Preconditions>
      <Precondition Type="ClaimsExist" ExecuteActionsIf="false">
        <Value>userType</Value>
        <Action>SkipThisValidationTechnicalProfile</Action>
      </Precondition>
      <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
        <Value>userType</Value>
        <Value>Customer</Value>
        <Action>SkipThisValidationTechnicalProfile</Action>
      </Precondition>
    </Preconditions>
  </ValidationTechnicalProfile>
</ValidationTechnicalProfiles>
```

UserJourneys

2/4/2020 • 7 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

User journeys specify explicit paths through which a policy allows a relying party application to obtain the desired claims for a user. The user is taken through these paths to retrieve the claims that are to be presented to the relying party. In other words, user journeys define the business logic of what an end user goes through as the Azure AD B2C Identity Experience Framework processes the request.

These user journeys can be considered as templates available to satisfy the core need of the various relying parties of the community of interest. User journeys facilitate the definition of the relying party part of a policy. A policy can define multiple user journeys. Each user journey is a sequence of orchestration steps.

To define the user journeys supported by the policy, a **UserJourneys** element is added under the top-level element of the policy file.

The **UserJourneys** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
UserJourney	1:n	A user journey that defines all of the constructs necessary for a complete user flow.

The **UserJourney** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier of a user journey that can be used to reference it from other elements in the policy. The DefaultUserJourney element of the relying party policy points to this attribute.

The **UserJourney** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
OrchestrationSteps	1:n	An orchestration sequence that must be followed through for a successful transaction. Every user journey consists of an ordered list of orchestration steps that are executed in sequence. If any step fails, the transaction fails.

OrchestrationSteps

A user journey is represented as an orchestration sequence that must be followed through for a successful transaction. If any step fails, the transaction fails. These orchestration steps reference both the building blocks and the claims providers allowed in the policy file. Any orchestration step that is responsible to show or render a user experience also has a reference to the corresponding content definition identifier.

Orchestration steps can be conditionally executed based on preconditions defined in the orchestration step element. For example, you can check to perform an orchestration step only if a specific claims exists, or if a claim is equal or not to the specified value.

To specify the ordered list of orchestration steps, an **OrchestrationSteps** element is added as part of the policy. This element is required.

The **OrchestrationSteps** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
OrchestrationStep	1:n	An ordered orchestration step.

The **OrchestrationStep** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Order	Yes	The order of the orchestration steps.
Type	Yes	<p>The type of the orchestration step. Possible values:</p> <ul style="list-style-type: none"> • ClaimsProviderSelection - Indicates that the orchestration step presents various claims providers to the user to select one. • CombinedSignInAndSignUp - Indicates that the orchestration step presents a combined social provider sign-in and local account sign-up page. • ClaimsExchange - Indicates that the orchestration step exchanges claims with a claims provider. • SendClaims - Indicates that the orchestration step sends the claims to the relying party with a token issued by a claims issuer.
ContentDefinitionReferenceId	No	<p>The identifier of the content definition associated with this orchestration step. Usually the content definition reference identifier is defined in the self-asserted technical profile. But, there are some cases when Azure AD B2C needs to display something without a technical profile. There are two examples - if the type of the orchestration step is one of following:</p> <p><code>ClaimsProviderSelection</code> or <code>CombinedSignInAndSignUp</code>, Azure AD B2C needs to display the identity provider selection without having a technical profile.</p>
CpiIssuerTechnicalProfileReferenceId	No	<p>The type of the orchestration step is <code>SendClaims</code>. This property defines the technical profile identifier of the claims provider that issues the token for the relying party. If absent, no relying party token is created.</p>

The **OrchestrationStep** element can contain the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Preconditions	0:n	A list of preconditions that must be satisfied for the orchestration step to execute.
ClaimsProviderSelections	0:n	A list of claims provider selections for the orchestration step.
ClaimsExchanges	0:n	A list of claims exchanges for the orchestration step.

Preconditions

The **Preconditions** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
Precondition	1:n	Depending on the technical profile being used, either redirects the client according to the claims provider selection or makes a server call to exchange claims.

Precondition

The **Precondition** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Type	Yes	The type of check or query to perform for this precondition. The value can be ClaimsExist , which specifies that the actions should be performed if the specified claims exist in the user's current claim set, or ClaimEquals , which specifies that the actions should be performed if the specified claim exists and its value is equal to the specified value.
ExecuteActionsIf	Yes	Use a true or false test to decide if the actions in the precondition should be performed.

The **Precondition** elements contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
Value	1:n	A ClaimTypeReferenceld to be queried for. Another value element contains the value to be checked.
Action	1:1	The action that should be performed if the precondition check within an orchestration step is true. If the value of the Action is set to SkipThisOrchestrationStep , the associated OrchestrationStep should not be executed.

Preconditions examples

The following preconditions checks whether the user's objectId exists. In the user journey, the user has selected to sign in using local account. If the objectId exists, skip this orchestration step.

```
<OrchestrationStep Order="2" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
      <Value>objectId</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
    <ClaimsExchange Id="SignUpWithLogonEmailExchange" TechnicalProfileReferenceId="LocalAccountSignUpWithLogonEmail" />
  </ClaimsExchanges>
</OrchestrationStep>
```

The following preconditions checks whether the user signed in with a social account. An attempt is made to find the user account in the directory. If the user signs in or signs up with a local account, skip this orchestration step.

```
<OrchestrationStep Order="3" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimEquals" ExecuteActionsIf="true">
      <Value>authenticationSource</Value>
      <Value>localAccountAuthentication</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="AADUserReadUsingAlternativeSecurityId" TechnicalProfileReferenceId="AAD-UserReadUsingAlternativeSecurityId-NoError" />
  </ClaimsExchanges>
</OrchestrationStep>
```

Preconditions can check multiple preconditions. The following example checks whether 'objectId' or 'email' exists. If the first condition is true, the journey skips to the next orchestration step.

```

<OrchestrationStep Order="4" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
      <Value>objectId</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
      <Value>email</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="SelfAsserted-SocialEmail" TechnicalProfileReferenceId="SelfAsserted-SocialEmail" />
  </ClaimsExchanges>
</OrchestrationStep>

```

ClaimsProviderSelection

An orchestration step of type `ClaimsProviderSelection` or `CombinedSignInAndSignUp` may contain a list of claims providers that a user can sign in with. The order of the elements inside the `ClaimsProviderSelections` elements controls the order of the identity providers presented to the user.

The **ClaimsProviderSelections** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
ClaimsProviderSelection	1:n	Provides the list of claims providers that can be selected.

The **ClaimsProviderSelections** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
DisplayOption	No	Controls the behavior of a case where a single claims provider selection is available. Possible values: <code>DoNotShowSingleProvider</code> (default), the user is redirected immediately to the federated identity provider. Or <code>ShowSingleProvider</code> Azure AD B2C presents the sign-in page with the single identity provider selection. To use this attribute, the content definition version must be <code>urn:com:microsoft:aad:b2c:elements:contract:provider</code> and above.

The **ClaimsProviderSelection** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
TargetClaimsExchangeId	No	The identifier of the claims exchange, which is executed in the next orchestration step of the claims provider selection. This attribute or the ValidationClaimsExchangeId attribute must be specified, but not both.
ValidationClaimsExchangeId	No	The identifier of the claims exchange, which is executed in the current orchestration step to validate the claims provider selection. This attribute or the TargetClaimsExchangeId attribute must be specified, but not both.

ClaimsProviderSelection example

In the following orchestration step, the user can choose to sign in with Facebook, LinkedIn, Twitter, Google, or a local account. If the user selects one of the social identity providers, the second orchestration step executes with the selected claim exchange specified in the `TargetClaimsExchangeId` attribute. The second orchestration step redirects the user to the social identity provider to complete the sign-in process. If the user chooses to sign in with the local account, Azure AD B2C stays on the same orchestration step (the same sign-up page or sign-in page) and skips the second orchestration step.

```

<OrchestrationStep Order="1" Type="CombinedSignInAndSignUp" ContentDefinitionReferenceId="api.signuporsignin">
  <ClaimsProviderSelections>
    <ClaimsProviderSelection TargetClaimsExchangeId="FacebookExchange" />
    <ClaimsProviderSelection TargetClaimsExchangeId="LinkedInExchange" />
    <ClaimsProviderSelection TargetClaimsExchangeId="TwitterExchange" />
    <ClaimsProviderSelection TargetClaimsExchangeId="GoogleExchange" />
    <ClaimsProviderSelection ValidationClaimsExchangeId="LocalAccountSigninEmailExchange" />
  </ClaimsProviderSelections>
  <ClaimsExchanges>
    <ClaimsExchange Id="LocalAccountSigninEmailExchange"
      TechnicalProfileReferenceId="SelfAsserted-LocalAccountSignin-Email" />
  </ClaimsExchanges>
</OrchestrationStep>

<OrchestrationStep Order="2" Type="ClaimsExchange">
  <Preconditions>
    <Precondition Type="ClaimsExist" ExecuteActionsIf="true">
      <Value>objectId</Value>
      <Action>SkipThisOrchestrationStep</Action>
    </Precondition>
  </Preconditions>
  <ClaimsExchanges>
    <ClaimsExchange Id="FacebookExchange" TechnicalProfileReferenceId="Facebook-OAUTH" />
    <ClaimsExchange Id="SignUpWithLogonEmailExchange" TechnicalProfileReferenceId="LocalAccountSignUpWithLogonEmail" />
    <ClaimsExchange Id="GoogleExchange" TechnicalProfileReferenceId="Google-OAUTH" />
    <ClaimsExchange Id="LinkedInExchange" TechnicalProfileReferenceId="LinkedIn-OAUTH" />
    <ClaimsExchange Id="TwitterExchange" TechnicalProfileReferenceId="Twitter-OAUTH1" />
  </ClaimsExchanges>
</OrchestrationStep>

```

ClaimsExchanges

The **ClaimsExchanges** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
ClaimsExchange	1:n	Depending on the technical profile being used, either redirects the client according to the ClaimsProviderSelection that was selected, or makes a server call to exchange claims.

The **ClaimsExchange** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	An identifier of the claims exchange step. The identifier is used to reference the claims exchange from a claims provider selection step in the policy.
TechnicalProfileReferenceId	Yes	The identifier of the technical profile that is to be executed.

RelyingParty

2/24/2020 • 8 minutes to read • [Edit Online](#)

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

The **RelyingParty** element specifies the user journey to enforce for the current request to Azure Active Directory B2C (Azure AD B2C). It also specifies the list of claims that the relying party (RP) application needs as part of the issued token. An RP application, such as a web, mobile, or desktop application, calls the RP policy file. The RP policy file executes a specific task, such as signing in, resetting a password, or editing a profile. Multiple applications can use the same RP policy and a single application can use multiple policies. All RP applications receive the same token with claims, and the user goes through the same user journey.

The following example shows a **RelyingParty** element in the *B2C_1A_signup_signin* policy file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TrustFrameworkPolicy
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="https://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
  PolicySchemaVersion="0.3.0.0"
  TenantId="your-tenant.onmicrosoft.com"
  PolicyId="B2C_1A_signup_signin"
  PublicPolicyUri="http://your-tenant.onmicrosoft.com/B2C_1A_signup_signin">

  <BasePolicy>
    <TenantId>your-tenant.onmicrosoft.com</TenantId>
    <PolicyId>B2C_1A_TrustFrameworkExtensions</PolicyId>
  </BasePolicy>

  <RelyingParty>
    <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
    <UserJourneyBehaviors>
      <SingleSignOn Scope="TrustFramework" KeepAliveInDays="7"/>
      <SessionExpiryType>Rolling</SessionExpiryType>
      <SessionExpiryInSeconds>300</SessionExpiryInSeconds>
      <JourneyInsights TelemetryEngine="ApplicationInsights" InstrumentationKey="your-application-insights-key" DeveloperMode="true" ClientEnabled="false" ServerEnabled="true" TelemetryVersion="1.0.0" />
      <ContentDefinitionParameters>
        <Parameter Name="campaignId">{OAUTH-KV:campaignId}</Parameter>
      </ContentDefinitionParameters>
    </UserJourneyBehaviors>
    <TechnicalProfile Id="PolicyProfile">
      <DisplayName>PolicyProfile</DisplayName>
      <Description>The policy profile</Description>
      <Protocol Name="OpenIdConnect" />
      <Metadata>collection of key/value pairs of data</Metadata>
      <OutputClaims>
        <OutputClaim ClaimTypeReferenceId="displayName" />
        <OutputClaim ClaimTypeReferenceId="givenName" />
        <OutputClaim ClaimTypeReferenceId="surname" />
        <OutputClaim ClaimTypeReferenceId="email" />
        <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
        <OutputClaim ClaimTypeReferenceId="identityProvider" />
        <OutputClaim ClaimTypeReferenceId="loyaltyNumber" />
      </OutputClaims>
      <SubjectNamingInfo ClaimType="sub" />
    </TechnicalProfile>
  </RelyingParty>
  ...

```

The optional **RelyingParty** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
DefaultUserJourney	1:1	The default user journey for the RP application.
UserJourneyBehaviors	0:1	The scope of the user journey behaviors.
TechnicalProfile	1:1	A technical profile that's supported by the RP application. The technical profile provides a contract for the RP application to contact Azure AD B2C.

DefaultUserJourney

The `<DefaultUserJourney>` element specifies a reference to the identifier of the user journey that is usually defined in the Base or Extensions policy. The following examples show the sign-up or sign-in user journey specified in the **RelyingParty** element:

B2C_1A_signup_signin policy:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn">
    ...
  </DefaultUserJourney>
</RelyingParty>
```

B2C_1A_TrustFrameWorkBase or *B2C_1A_TrustFrameworkExtensionPolicy*:

```
<UserJourneys>
  <UserJourney Id="SignUpOrSignIn">
    ...
  </UserJourney>
</UserJourneys>
```

The **DefaultUserJourney** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
ReferenceId	Yes	An identifier of the user journey in the policy. For more information, see user journeys

UserJourneyBehaviors

The **UserJourneyBehaviors** element contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
SingleSignOn	0:1	The scope of the single sign-on (SSO) session behavior of a user journey.
SessionExpiryType	0:1	The authentication behavior of the session. Possible values: <code>Rolling</code> or <code>Absolute</code> . The <code>Rolling</code> value (default) indicates that the user remains signed in as long as the user is continually active in the application. The <code>Absolute</code> value indicates that the user is forced to reauthenticate after the time period specified by application session lifetime.
SessionExpiryInSeconds	0:1	The lifetime of Azure AD B2C's session cookie specified as an integer stored on the user's browser upon successful authentication.

ELEMENT	OCCURRENCES	DESCRIPTION
JourneyInsights	0:1	The Azure Application Insights instrumentation key to be used.
ContentDefinitionParameters	0:1	The list of key value pairs to be appended to the content definition load URI.
ScriptExecution	0:1	The supported JavaScript execution modes. Possible values: <code>Allow</code> or <code>Disallow</code> (default).

SingleSignOn

The **SingleSignOn** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Scope	Yes	The scope of the single sign-on behavior. Possible values: <code>Suppressed</code> , <code>Tenant</code> , <code>Application</code> , or <code>Policy</code> . The <code>Suppressed</code> value indicates that the behavior is suppressed. For example, in the case of a single sign-on session, no session is maintained for the user and the user is always prompted for an identity provider selection. The <code>TrustFramework</code> value indicates that the behavior is applied for all policies in the trust framework. For example, a user navigating through two policy journeys for a trust framework is not prompted for an identity provider selection. The <code>Tenant</code> value indicates that the behavior is applied to all policies in the tenant. For example, a user navigating through two policy journeys for a tenant is not prompted for an identity provider selection. The <code>Application</code> value indicates that the behavior is applied to all policies for the application making the request. For example, a user navigating through two policy journeys for an application is not prompted for an identity provider selection. The <code>Policy</code> value indicates that the behavior only applies to a policy. For example, a user navigating through two policy journeys for a trust framework is prompted for an identity provider selection when switching between policies.
KeepAliveInDays	Yes	Controls how long the user remains signed in. Setting the value to 0 turns off KMSI functionality. For more information, see Keep me signed in .
EnforceIdTokenHintOnLogout	No	Force to pass a previously issued ID token to the logout endpoint as a hint about the end user's current authenticated session with the client. Possible values: <code>false</code> (default), or <code>true</code> . For more information, see Web sign-in with OpenID Connect .

JourneyInsights

The **JourneyInsights** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
TelemetryEngine	Yes	The value must be <code>ApplicationInsights</code> .
InstrumentationKey	Yes	The string that contains the instrumentation key for the application insights element.
DeveloperMode	Yes	Possible values: <code>true</code> or <code>false</code> . If <code>true</code> , Application Insights expedites the telemetry through the processing pipeline. This setting is good for development, but constrained at high volumes. The detailed activity logs are designed only to aid in development of custom policies. Do not use development mode in production. Logs collect all claims sent to and from the identity providers during development. If used in production, the developer assumes responsibility for PII (Privately Identifiable Information) collected in the App Insights log that they own. These detailed logs are only collected when this value is set to <code>true</code> .
ClientEnabled	Yes	Possible values: <code>true</code> or <code>false</code> . If <code>true</code> , sends the Application Insights client-side script for tracking page view and client-side errors.
ServerEnabled	Yes	Possible values: <code>true</code> or <code>false</code> . If <code>true</code> , sends the existing UserJourneyRecorder JSON as a custom event to Application Insights.
TelemetryVersion	Yes	The value must be <code>1.0.0</code> .

For more information, see [Collecting Logs](#)

ContentDefinitionParameters

By using custom policies in Azure AD B2C, you can send a parameter in a query string. By passing the parameter to your HTML endpoint, you can dynamically change the page content. For example, you can change the background image on the Azure AD B2C sign-up or sign-in page, based on a parameter that you pass from your web or mobile application. Azure AD B2C passes the query string parameters to your dynamic HTML file, such as aspx file.

The following example passes a parameter named `campaignId` with a value of `hawaii` in the query string:

```
https://login.microsoft.com/contoso.onmicrosoft.com/oauth2/v2.0/authorize?pB2C_1A_signup_signin&client_id=a415078a-0402-4ce3-a9c6-ec1947fcfb3f&nonce=defaultNonce&redirect_uri=http%3A%2F%2Fjwt.io%2F&scope=openid&response_type=id_token&prompt=login&campaignId=hawaii
```

The **ContentDefinitionParameters** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
ContentDefinitionParameter	0:n	A string that contains the key value pair that's appended to the query string of a content definition load URI.

The **ContentDefinitionParameter** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Name	Yes	The name of the key value pair.

For more information, see [Configure the UI with dynamic content by using custom policies](#)

TechnicalProfile

The **TechnicalProfile** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Id	Yes	The value must be <code>PolicyProfile</code> .

The **TechnicalProfile** contains the following elements:

ELEMENT	OCCURRENCES	DESCRIPTION
DisplayName	1:1	The string that contains the name of the technical profile.
Description	0:1	The string that contains the description of the technical profile.
Protocol	1:1	The protocol used for the federation.
Metadata	0:1	The collection of <i>Item</i> of key/value pairs utilized by the protocol for communicating with the endpoint in the course of a transaction to configure interaction between the relying party and other community participants.
OutputClaims	1:1	A list of claim types that are taken as output in the technical profile. Each of these elements contains reference to a ClaimType already defined in the ClaimsSchema section or in a policy from which this policy file inherits.
SubjectNamingInfo	1:1	The subject name used in tokens.

The **Protocol** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
Name	Yes	The name of a valid protocol supported by Azure AD B2C that is used as part of the technical profile. Possible values: <code>OpenIdConnect</code> or <code>SAML2</code> . The <code>OpenIdConnect</code> value represents the OpenID Connect 1.0 protocol standard as per OpenID foundation specification. The <code>SAML2</code> represents the SAML 2.0 protocol standard as per OASIS specification. Do not use a SAML token in production.

OutputClaims

The **OutputClaims** element contains the following element:

ELEMENT	OCCURRENCES	DESCRIPTION
OutputClaim	0:n	The name of an expected claim type in the supported list for the policy to which the relying party subscribes. This claim serves as an output for the technical profile.

The **OutputClaim** element contains the following attributes:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimTypeReferenceId	Yes	A reference to a ClaimType already defined in the ClaimsSchema section in the policy file.
DefaultValue	No	A default value that can be used if the claim value is empty.
PartnerClaimType	No	Sends the claim in a different name as configured in the ClaimType definition.

SubjectNamingInfo

With the **SubjectNameingInfo** element, you control the value of the token subject:

- **JWT token** - the `sub` claim. This is a principal about which the token asserts information, such as the user of an application. This value is immutable and cannot be reassigned or reused. It can be used to perform safe authorization checks, such as when the token is used to access a resource. By default, the subject claim is populated with the object ID of the user in the directory. For more information, see [Token, session and single sign-on configuration](#).
- **SAML token** - the `<Subject><NameID>` element which identifies the subject element.

The **SubjectNameingInfo** element contains the following attribute:

ATTRIBUTE	REQUIRED	DESCRIPTION
ClaimType	Yes	A reference to an output claim's PartnerClaimType . The output claims must be defined in the relying party policy OutputClaims collection.

The following example shows how to define an OpenID Connect relying party. The subject name info is configured as the `objectId`:

```

<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub"/>
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>

```

The JWT token includes the `sub` claim with the user `objectId`:

```
{
  ...
  "sub": "6fbbd70d-262b-4b50-804c-257ae1706ef2",
  ...
}
```

Set redirect URLs to b2clogin.com for Azure Active Directory B2C

2/17/2020 • 3 minutes to read • [Edit Online](#)

When you set up an identity provider for sign-up and sign-in in your Azure Active Directory B2C (Azure AD B2C) application, you need to specify a redirect URL. You should no longer reference *login.microsoftonline.com* in your applications and APIs. Instead, use *b2clogin.com* for all new applications, and migrate existing applications from *login.microsoftonline.com* to *b2clogin.com*.

Deprecation of login.microsoftonline.com

On 04 December 2019, we announced the scheduled retirement of *login.microsoftonline.com* support in Azure AD B2C on **04 December 2020**:

[Azure Active Directory B2C is deprecating login.microsoftonline.com](#)

The deprecation of *login.microsoftonline.com* goes into effect for all Azure AD B2C tenants on 04 December 2020, providing existing tenants one (1) year to migrate to *b2clogin.com*. New tenants created after 04 December 2019 will not accept requests from *login.microsoftonline.com*. All functionality remains the same on the *b2clogin.com* endpoint.

The deprecation of *login.microsoftonline.com* does not impact Azure Active Directory tenants. Only Azure Active Directory B2C tenants are affected by this change.

Benefits of b2clogin.com

When you use *b2clogin.com* as your redirect URL:

- Space consumed in the cookie header by Microsoft services is reduced.
- Your redirect URLs no longer need to include a reference to Microsoft.
- JavaScript client-side code is supported (currently in [preview](#)) in customized pages. Due to security restrictions, JavaScript code and HTML form elements are removed from custom pages if you use *login.microsoftonline.com*.

Overview of required changes

There are several modifications you might need to make to migrate your applications to *b2clogin.com*:

- Change the redirect URL in your identity provider's applications to reference *b2clogin.com*.
- Update your Azure AD B2C applications to use *b2clogin.com* in their user flow and token endpoint references.
- Update any **Allowed Origins** that you've defined in the CORS settings for [user interface customization](#).

Change identity provider redirect URLs

On each identity provider's website in which you've created an application, change all trusted URLs to redirect to `your-tenant-name.b2clogin.com` instead of *login.microsoftonline.com*.

There are two formats you can use for your *b2clogin.com* redirect URLs. The first provides the benefit of not having "Microsoft" appear anywhere in the URL by using the Tenant ID (a GUID) in place of your tenant domain name:

```
https://{{your-tenant-name}}.b2clogin.com/{{your-tenant-id}}/oauth2/authresp
```

The second option uses your tenant domain name in the form of `your-tenant-name.onmicrosoft.com`. For example:

```
https://{{your-tenant-name}}.b2clogin.com/{{your-tenant-name}}.onmicrosoft.com/oauth2/authresp
```

For both formats:

- Replace `{your-tenant-name}` with the name of your Azure AD B2C tenant.
- Remove `/te` if it exists in the URL.

Update your applications and APIs

The code in your Azure AD B2C-enabled applications and APIs may refer to `login.microsoftonline.com` in several places. For example, your code might have references to user flows and token endpoints. Update the following to instead reference `your-tenant-name.b2clogin.com`:

- Authorization endpoint
- Token endpoint
- Token issuer

For example, the authority endpoint for Contoso's sign-up/sign-in policy would now be:

```
https://contosob2c.b2clogin.com/00000000-0000-0000-0000-000000000000/B2C_1_signupsignin1
```

For information about migrating OWIN-based web applications to b2clogin.com, see [Migrate an OWIN-based web API to b2clogin.com](#).

For migrating Azure API Management APIs protected by Azure AD B2C, see the [Migrate to b2clogin.com](#) section of [Secure an Azure API Management API with Azure AD B2C](#).

Microsoft Authentication Library (MSAL)

ValidateAuthority property

If you're using [MSAL.NET v2](#) or earlier, set the **ValidateAuthority** property to `false` on client instantiation to allow redirects to `b2clogin.com`. This setting is not required for MSAL.NET v3 and above.

```
ConfidentialClientApplication client = new ConfidentialClientApplication(...); // Can also be  
PublicClientApplication  
client.ValidateAuthority = false; // MSAL.NET v2 and earlier **ONLY**
```

If you're using [MSAL for JavaScript](#):

```
this.clientApplication = new UserAgentApplication(  
    env.auth.clientId,  
    env.auth.loginAuthority,  
    this.authCallback.bind(this),  
    {  
        validateAuthority: false  
    }  
);
```

Next steps

For information about migrating OWIN-based web applications to b2clogin.com, see [Migrate an OWIN-based web API to b2clogin.com](#).

For migrating Azure API Management APIs protected by Azure AD B2C, see the [Migrate to b2clogin.com](#) section of [Secure an Azure API Management API with Azure AD B2C](#).

Migrate an OWIN-based web API to b2clogin.com

1/28/2020 • 4 minutes to read • [Edit Online](#)

This article describes a technique for enabling support for multiple token issuers in web APIs that implement the [Open Web Interface for .NET \(OWIN\)](#). Supporting multiple token endpoints is useful when you're migrating Azure Active Directory B2C (Azure AD B2C) APIs and their applications from login.microsoftonline.com to b2clogin.com.

By adding support in your API for accepting tokens issued by both b2clogin.com and login.microsoftonline.com, you can migrate your web applications in a staged manner before removing support for login.microsoftonline.com-issued tokens from the API.

The following sections present an example of how to enable multiple issuers in a web API that uses the [Microsoft OWIN](#) middleware components (Katana). Although the code examples are specific to the Microsoft OWIN middleware, the general technique should be applicable to other OWIN libraries.

NOTE

This article is intended for Azure AD B2C customers with currently deployed APIs and applications that reference login.microsoftonline.com and who want to migrate to the recommended b2clogin.com endpoint. If you're setting up a new application, use b2clogin.com as directed.

Prerequisites

You need the following Azure AD B2C resources in place before continuing with the steps in this article:

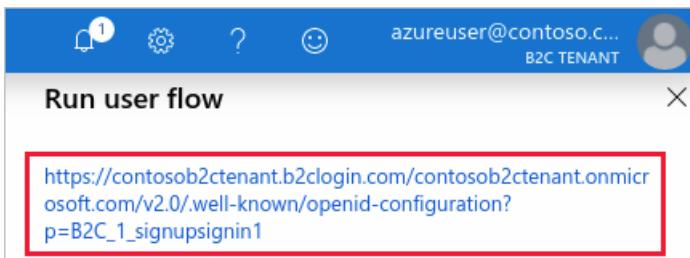
- [User flows](#) or [custom policies](#) created in your tenant

Get token issuer endpoints

You first need to get the token issuer endpoint URIs for each issuer you want to support in your API. To get the b2clogin.com and login.microsoftonline.com endpoints supported by your Azure AD B2C tenant, use the following procedure in the Azure portal.

Start by selecting one of your existing user flows:

1. Navigate to your Azure AD B2C tenant in the [Azure portal](#)
2. Under **Policies**, select **User flows (policies)**
3. Select an existing policy, for example *B2C_1_signupsignin1*, then select **Run user flow**
4. Under the **Run user flow** heading near the top of the page, select the hyperlink to navigate to the OpenID Connect discovery endpoint for that user flow.



5. In the page that opens in your browser, record the `issuer` value, for example:

```
https://your-b2c-tenant.b2clogin.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/
```

6. Use the **Select domain** drop-down to select the other domain, then perform the previous two steps once again and record its `issuer` value.

You should now have two URIs recorded that are similar to:

```
https://login.microsoftonline.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/  
https://your-b2c-tenant.b2clogin.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/
```

Custom policies

If you have custom policies instead of user flows, you can use a similar process to get the issuer URIs.

1. Navigate to your Azure AD B2C tenant
2. Select **Identity Experience Framework**
3. Select one of your relying party policies, for example, *B2C_1A_signup_signin*
4. Use the **Select domain** drop-down to select a domain, for example *yourtenant.b2clogin.com*
5. Select the hyperlink displayed under **OpenID Connect discovery endpoint**
6. Record the `issuer` value
7. Perform the steps 4-6 for the other domain, for example *login.microsoftonline.com*

Get the sample code

Now that you have both token endpoint URIs, you need to update your code to specify that both endpoints are valid issuers. To walk through an example, download or clone the sample application, then update the sample to support both endpoints as valid issuers.

Download the archive: [active-directory-b2c-dotnet-webapp-and-webapi-master.zip](#)

```
git clone https://github.com/Azure-Samples/active-directory-b2c-dotnet-webapp-and-webapi.git
```

Enable multiple issuers in web API

In this section, you update the code to specify that both token issuer endpoints are valid.

1. Open the **B2C-WebAPI-DotNet.sln** solution in Visual Studio
2. In the **TaskService** project, open the *TaskService\App_Start\Startup.Auth.cs* file in your editor
3. Add the following `using` directive to the top of the file:

```
using System.Collections.Generic;
```

4. Add the `ValidIssuers` property to the `TokenValidationParameters` definition and specify both URIs you recorded in the previous section:

```

TokenValidationParameters tvps = new TokenValidationParameters
{
    // Accept only those tokens where the audience of the token is equal to the client ID of this app
    ValidAudience = ClientId,
    AuthenticationType = Startup.DefaultPolicy,
    ValidIssuers = new List<string> {
        "https://login.microsoftonline.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/",
        "https://{your-b2c-tenant}.b2clogin.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/"
    }
};

```

`TokenValidationParameters` is provided by MSAL.NET and is consumed by the OWIN middleware in the next section of code in *Startup.Auth.cs*. With multiple valid issuers specified, the OWIN application pipeline is made aware that both token endpoints are valid issuers.

```

app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions
{
    // This SecurityTokenProvider fetches the Azure AD B2C metadata & from the OpenID Connect metadata
    // endpoint
    AccessTokenFormat = new JwtFormat(tvps, new tCachingSecurityTokenProvider(String.Format(AadInstance,
        ultPolicy)))
});

```

As mentioned previously, other OWIN libraries typically provide a similar facility for supporting multiple issuers. Although providing examples for every library is outside the scope of this article, you can use a similar technique for most libraries.

Switch endpoints in web app

With both URIs now supported by your web API, you now need update your web application so that it retrieves tokens from the b2clogin.com endpoint.

For example, you can configure the sample web application to use the new endpoint by modifying the `ida:AadInstance` value in the *TaskWebApp\Web.config* file of the **TaskWebApp** project.

Change the `ida:AadInstance` value in the *Web.config* of TaskWebApp so that it references `{your-b2c-tenant-name}.b2clogin.com` instead of `login.microsoftonline.com`.

Before:

```

<!-- Old value -->
<add key="ida:AadInstance" value="https://login.microsoftonline.com/tfp/{0}/{1}" />

```

After (replace `{your-b2c-tenant}` with the name of your B2C tenant):

```

<!-- New value -->
<add key="ida:AadInstance" value="https://{your-b2c-tenant}.b2clogin.com/tfp/{0}/{1}" />

```

When the endpoint strings are constructed during execution of the web app, the b2clogin.com-based endpoints are used when it requests tokens.

Next steps

This article presented a method of configuring a web API implementing the Microsoft OWIN middleware (Katana) to accept tokens from multiple issuer endpoints. As you might notice, there are several other strings in the

Web.Config files of both the TaskService and TaskWebApp projects that would need to be changed if you want to build and run these projects against your own tenant. You're welcome to modify the projects appropriately if you want to see them in action, however, a full walk-through of doing so is outside the scope of this article.

For more information about the different types of security tokens emitted by Azure AD B2C, see [Overview of tokens in Azure Active Directory B2C](#).

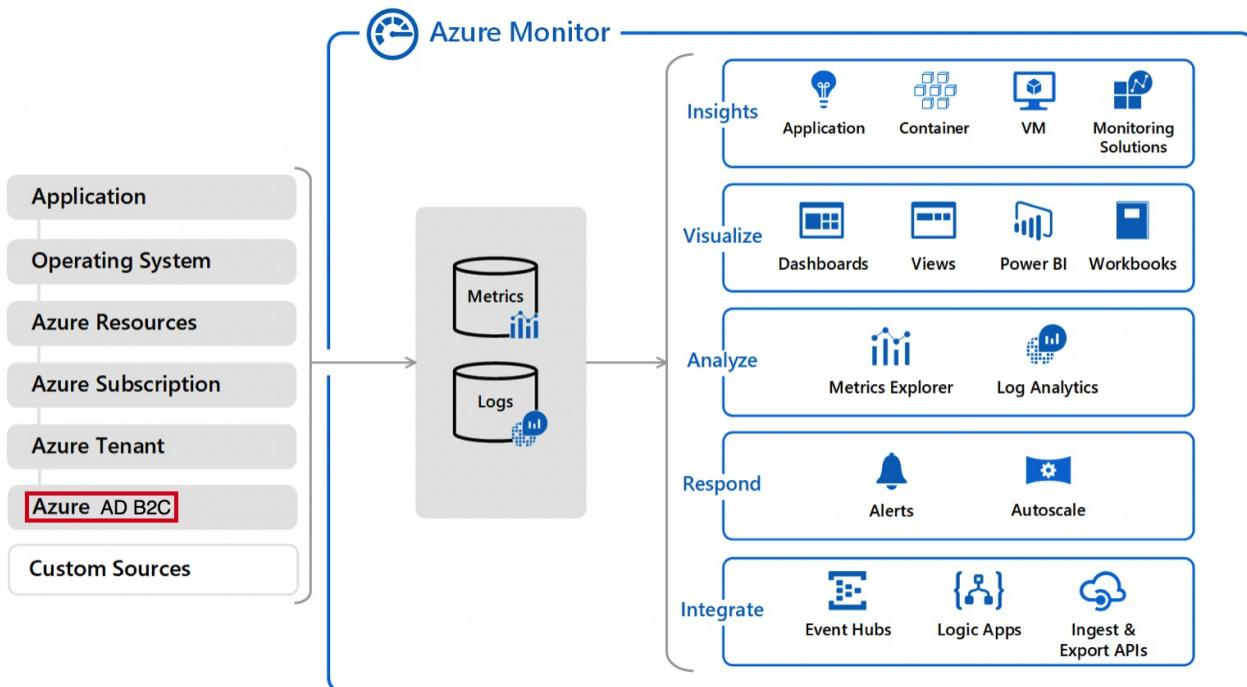
Monitor Azure AD B2C with Azure Monitor

2/10/2020 • 7 minutes to read • [Edit Online](#)

Use Azure Monitor to route Azure Active Directory B2C (Azure AD B2C) sign-in and [auditing](#) logs to different monitoring solutions. You can retain the logs for long-term use or integrate with third-party security information and event management (SIEM) tools to gain insights into your environment.

You can route log events to:

- An Azure [storage account](#).
- An Azure [event hub](#) (and integrate with your Splunk and Sumo Logic instances).
- An [Log Analytics workspace](#) (to analyze data, create dashboards, and alert on specific events).



Prerequisites

To complete the steps in this article, you deploy an Azure Resource Manager template by using the Azure PowerShell module.

- [Azure PowerShell module](#) version 6.13.1 or higher

You can also use the [Azure Cloud Shell](#), which includes the latest version of the Azure PowerShell module.

Delegated resource management

Azure AD B2C leverages [Azure Active Directory monitoring](#). To enable *Diagnostic settings* in Azure Active Directory within your Azure AD B2C tenant, you use [delegated resource management](#).

You authorize a user or group in your Azure AD B2C directory (the **Service Provider**) to configure the Azure Monitor instance within the tenant that contains your Azure subscription (the **Customer**). To create the authorization, you deploy an [Azure Resource Manager](#) template to your Azure AD tenant containing the subscription. The following sections walk you through the process.

Create or choose resource group

This is the resource group containing the destination Azure storage account, event hub, or Log Analytics workspace to receive data from Azure Monitor. You specify the resource group name when you deploy the Azure Resource Manager template.

[Create a resource group](#) or choose an existing one in the Azure Active Directory (Azure AD) tenant that contains your Azure subscription, *not* the directory that contains your Azure AD B2C tenant.

This example uses a resource group named *azure-ad-b2c-monitor* in the *Central US* region.

Delegate resource management

Next, gather the following information:

Directory ID of your Azure AD B2C directory (also known as the tenant ID).

1. Sign in to the [Azure portal](#) as a user with the *User administrator* role (or higher).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. Select **Azure Active Directory**, select **Properties**.
4. Record the **Directory ID**.

Object ID of the Azure AD B2C group or user you want to give *Contributor* permission to the resource group you created earlier in the directory containing your subscription.

To make management easier, we recommend using Azure AD user *groups* for each role, allowing you to add or remove individual users to the group rather than assigning permissions directly to that user. In this walkthrough, you add a user.

1. With **Azure Active Directory** still selected in the Azure portal, select **Users**, and then select a user.
2. Record the user's **Object ID**.

Create an Azure Resource Manager template

To onboard your Azure AD tenant (the **Customer**), create an [Azure Resource Manager template](#) for your offer with the following information. The `mspOfferName` and `mspOfferDescription` values are visible when you view offer details in the [Service providers page](#) of the Azure portal.

FIELD	DEFINITION
<code>mspOfferName</code>	A name describing this definition. For example, <i>Azure AD B2C Managed Services</i> . This value is displayed to the customer as the title of the offer.
<code>mspOfferDescription</code>	A brief description of your offer. For example, <i>Enables Azure Monitor in Azure AD B2C</i> .
<code>rgName</code>	The name of the resource group you create earlier in your Azure AD tenant. For example, <i>azure-ad-b2c-monitor</i> .
<code>managedByTenantId</code>	The Directory ID of your Azure AD B2C tenant (also known as the tenant ID).
<code>authorizations.value.principalId</code>	The Object ID of the B2C group or user that will have access to resources in this Azure subscription. For this walkthrough, specify the user's Object ID that you recorded earlier.

Download the Azure Resource Manager template and parameter files:

- [rgDelegatedResourceManagement.json](#)
- [rgDelegatedResourceManagement.parameters.json](#)

Next, update the parameters file with the values you recorded earlier. The following JSON snippet shows an example of an Azure Resource Manager template parameters file. For `authorizations.value.roleDefinitionId`, use the [built-in role](#) value for the *Contributor role*, `b24988ac-6180-42a0-ab88-20f7382dd24c`.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "mspOfferName": {  
            "value": "Azure AD B2C Managed Services"  
        },  
        "mspOfferDescription": {  
            "value": "Enables Azure Monitor in Azure AD B2C"  
        },  
        "rgName": {  
            "value": "azure-ad-b2c-monitor"  
        },  
        "managedByTenantId": {  
            "value": "<Replace with DIRECTORY ID of Azure AD B2C tenant (tenant ID)>"  
        },  
        "authorizations": {  
            "value": [  
                {  
                    "principalId": "<Replace with user's OBJECT ID>",  
                    "principalIdDisplayName": "Azure AD B2C tenant administrator",  
                    "roleDefinitionId": "b24988ac-6180-42a0-ab88-20f7382dd24c"  
                }  
            ]  
        }  
    }  
}
```

Deploy the Azure Resource Manager templates

Once you've updated your parameters file, deploy the Azure Resource Manager template into the Azure tenant as a subscription-level deployment. Because this is a subscription-level deployment, it cannot be initiated in the Azure portal. You can deploy by using the Azure PowerShell module or the Azure CLI. The Azure PowerShell method is shown below.

Sign in to the directory containing your subscription by using [Connect-AzAccount](#). Use the `-tenant` flag to force authentication to the correct directory.

```
Connect-AzAccount -tenant contoso.onmicrosoft.com
```

Use the [Get-AzSubscription](#) cmdlet to list the subscriptions that the current account can access under the Azure AD tenant. Record the ID of the subscription you want to project into your Azure AD B2C tenant.

```
Get-AzSubscription
```

Next, switch to the subscription you want to project into the Azure AD B2C tenant:

```
Select-AzSubscription <subscription ID>
```

Finally, deploy the Azure Resource Manager template and parameter files you downloaded and updated earlier. Replace the `Location`, `TemplateFile`, and `TemplateParameterFile` values accordingly.

```
New-AzDeployment -Name "AzureADB2C" `  
    -Location "centralus" `  
    -TemplateFile "C:\Users\azureuser\Documents\rgDelegatedResourceManagement.json" `  
    -TemplateParameterFile  
"C:\Users\azureuser\Documents\rgDelegatedResourceManagement.parameters.json" `  
    -Verbose
```

Successful deployment of the template produces output similar to the following (output truncated for brevity):

```
PS /usr/csuser/clouddrive> New-AzDeployment -Name "AzureADB2C" `  
>>         -Location "centralus" `  
>>         -TemplateFile "rgDelegatedResourceManagement.json" `  
>>         -TemplateParameterFile "rgDelegatedResourceManagement.parameters.json" `  
>>         -Verbose  
WARNING: Breaking changes in the cmdlet 'New-AzDeployment' :  
WARNING: - The cmdlet 'New-AzSubscriptionDeployment' is replacing this cmdlet.  
  
WARNING: NOTE : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning,  
and other information on breaking changes in Azure PowerShell.  
VERBOSE: 7:25:14 PM - Template is valid.  
VERBOSE: 7:25:15 PM - Create template deployment 'AzureADB2C'  
VERBOSE: 7:25:15 PM - Checking deployment status in 5 seconds  
VERBOSE: 7:25:42 PM - Resource Microsoft.ManagedServices/registrationDefinitions '44444444-4444-4444-4444-  
444444444444' provisioning status is succeeded  
VERBOSE: 7:25:48 PM - Checking deployment status in 5 seconds  
VERBOSE: 7:25:53 PM - Resource Microsoft.Resources/deployments 'rgAssignment' provisioning status is running  
VERBOSE: 7:25:53 PM - Checking deployment status in 5 seconds  
VERBOSE: 7:25:59 PM - Resource Microsoft.ManagedServices/registrationAssignments '11111111-1111-1111-1111-  
111111111111' provisioning status is running  
VERBOSE: 7:26:17 PM - Checking deployment status in 5 seconds  
VERBOSE: 7:26:23 PM - Resource Microsoft.ManagedServices/registrationAssignments '11111111-1111-1111-1111-  
111111111111' provisioning status is succeeded  
VERBOSE: 7:26:23 PM - Checking deployment status in 5 seconds  
VERBOSE: 7:26:29 PM - Resource Microsoft.Resources/deployments 'rgAssignment' provisioning status is succeeded  
  
DeploymentName      : AzureADB2C  
Location           : centralus  
ProvisioningState  : Succeeded  
Timestamp          : 1/31/20 7:26:24 PM  
Mode               : Incremental  
TemplateLink       :  
Parameters          :  
                    Name          Type          Value  
                    ======  ======  ======  
                    mspOfferName   String        Azure AD B2C Managed Services  
                    mspOfferDescription String        Enables Azure Monitor in Azure AD  
B2C  
...
```

After you deploy the template, it can take a few minutes for the resource projection to complete. You may need to wait a few minutes (typically no more than five) before moving on to the next section to select the subscription.

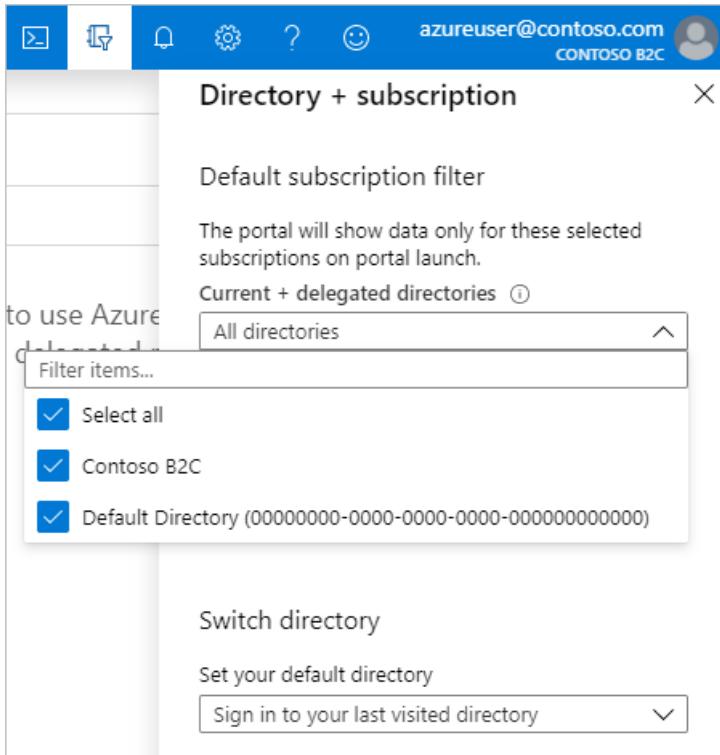
Select your subscription

Once you've deployed the template and have waited a few minutes for the resource projection to complete, associate your subscription to your Azure AD B2C directory with the following steps.

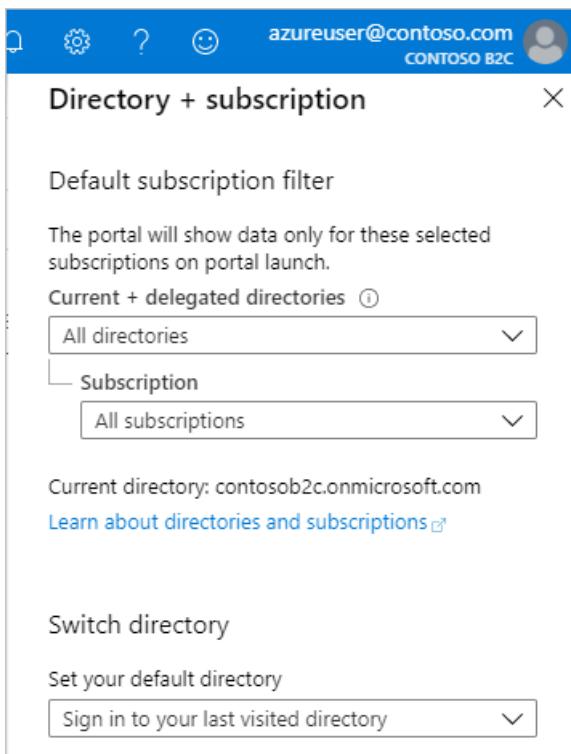
1. **Sign out** of the Azure portal if you're currently signed in. This and the following step are done to refresh

your credentials in the portal session.

2. Sign in to the [Azure portal](#) with your Azure AD B2C administrative account.
3. Select the **Directory + Subscription** icon in the portal toolbar.
4. Select the directory that contains your subscription.



5. Verify that you've selected the correct directory and subscription. In this example, all directories and subscriptions are selected.



Configure diagnostic settings

Diagnostic settings define where logs and metrics for a resource should be sent. Possible destinations are:

- Azure storage account
- Event hubs solutions.
- Log Analytics workspace

If you haven't already, create an instance of your chosen destination type in the resource group you specified in the [Azure Resource Manager template](#).

Create diagnostic settings

You're ready to [Create diagnostic settings](#) in the Azure portal.

To configure monitoring settings for Azure AD B2C activity logs:

1. Sign in to the [Azure portal](#).
2. Select the **Directory + Subscription** icon in the portal toolbar, and then select the directory that contains your Azure AD B2C tenant.
3. Select **Azure Active Directory**
4. Under **Monitoring**, select **Diagnostic settings**.
5. If there are existing settings on the resource, you will see a list of settings already configured. Either select **Add diagnostic setting** to add a new setting, or **Edit** setting to edit an existing one. Each setting can have no more than one of each of the destination types..

6. Give your setting a name if it doesn't already have one.
7. Check the box for each destination to send the logs. Select **Configure** to specify their settings as described in the following table.

SETTING	DESCRIPTION
Archive to a storage account	Name of storage account.
Stream to an event hub	The namespace where the event hub is created (if this is your first time streaming logs) or streamed to (if there are already resources that are streaming that log category to this namespace).
Send to Log Analytics	Name of workspace.

8. Select **AuditLogs** and **SignInLogs**.

9. Select **Save**.

Next steps

For more information about adding and configuring diagnostic settings in Azure Monitor, see [Tutorial: Collect and analyze resource logs from an Azure resource](#).

For information about streaming Azure AD logs to an event hub, see [Tutorial: Stream Azure Active Directory logs to an Azure event hub](#).

Manage Azure AD B2C user accounts with Microsoft Graph

2/24/2020 • 6 minutes to read • [Edit Online](#)

Microsoft Graph allows you to manage user accounts in your Azure AD B2C directory by providing create, read, update, and delete methods in the Microsoft Graph API. You can migrate an existing user store to an Azure AD B2C tenant and perform other user account management operations by calling the Microsoft Graph API.

In the sections that follow, the key aspects of Azure AD B2C user management with the Microsoft Graph API are presented. The Microsoft Graph API operations, types, and properties presented here are a subset of that which appears in the Microsoft Graph API reference documentation.

Register a management application

Before any user management application or script you write can interact with the resources in your Azure AD B2C tenant, you need an application registration that grants the permissions to do so.

Follow the steps in this how-to article to create an application registration that your management application can use:

[Manage Azure AD B2C with Microsoft Graph](#)

User management Microsoft Graph operations

The following user management operations are available in the [Microsoft Graph API](#):

- [Get a list of users](#)
- [Create a user](#)
- [Get a user](#)
- [Update a user](#)
- [Delete a user](#)

User properties

Display name property

The `displayName` is the name to display in Azure portal user management for the user, and in the access token Azure AD B2C returns to the application. This property is required.

Identities property

A customer account, which could be a consumer, partner, or citizen, can be associated with these identity types:

- **Local** identity - The username and password are stored locally in the Azure AD B2C directory. We often refer to these identities as "local accounts."
- **Federated** identity - Also known as a *social* or *enterprise* accounts, the identity of the user is managed by a federated identity provider like Facebook, Microsoft, ADFS, or Salesforce.

A user with a customer account can sign in with multiple identities. For example, username, email, employee ID, government ID, and others. A single account can have multiple identities, both local and social, with the same password.

In the Microsoft Graph API, both local and federated identities are stored in the user `identities` attribute, which is of type `objectIdentity`. The `identities` collection represents a set of identities used to sign in to a user account. This collection enables the user to sign in to the user account with any of its associated identities.

PROPERTY	TYPE	DESCRIPTION
signInType	string	<p>Specifies the user sign-in types in your directory. For local account:</p> <ul style="list-style-type: none"> <code>emailAddress</code> , <code>emailAddress1</code> , <code>emailAddress2</code> , <code>emailAddress3</code> , <code>userName</code> , or any other type you like. <p>Social account must be set to <code>federated</code>.</p>
issuer	string	<p>Specifies the issuer of the identity. For local accounts (where <code>signInType</code> is not <code>federated</code>), this property is the local B2C tenant default domain name, for example <code>contoso.onmicrosoft.com</code>. For social identity (where <code>signInType</code> is <code>federated</code>) the value is the name of the issuer, for example <code>facebook.com</code></p>
issuerAssignedId	string	<p>Specifies the unique identifier assigned to the user by the issuer. The combination of <code>issuer</code> and <code>issuerAssignedId</code> must be unique within your tenant. For local account, when <code>signInType</code> is set to <code>emailAddress</code> or <code>userName</code>, it represents the sign-in name for the user.</p> <p>When <code>signInType</code> is set to:</p> <ul style="list-style-type: none"> • <code>emailAddress</code> (or starts with <code>emailAddress</code> like <code>emailAddress1</code>) <code>issuerAssignedId</code> must be a valid email address • <code>userName</code> (or any other value), <code>issuerAssignedId</code> must be a valid local part of an email address • <code>federated</code>, <code>issuerAssignedId</code> represents the federated account unique identifier

For federated identities, depending on the identity provider, the `issuerAssignedId` is a unique value for a given user per application or development account. Configure the Azure AD B2C policy with the same application ID that was previously assigned by the social provider or another application within the same development account.

Password profile property

For a local identity, the `passwordProfile` property is required, and contains the user's password. The `forceChangePasswordNextSignIn` property must set to `false`.

For a federated (social) identity, the `passwordProfile` property is not required.

```
"passwordProfile" : {  
    "password": "password-value",  
    "forceChangePasswordNextSignIn": false  
}
```

Password policy property

The Azure AD B2C password policy (for local accounts) is based on the Azure Active Directory [strong password strength](#) policy. The Azure AD B2C sign-up or sign-in and password reset policies require this strong password strength, and don't expire passwords.

In user migration scenarios, if the accounts you want to migrate have weaker password strength than the [strong password strength](#) enforced by Azure AD B2C, you can disable the strong password requirement. To change the default password policy, set the `passwordPolicies` property to `DisableStrongPassword`. For example, you can modify the create user request as follows:

```
"passwordPolicies": "DisablePasswordExpiration, DisableStrongPassword"
```

Extension properties

Every customer-facing application has unique requirements for the information to be collected. Your Azure AD B2C tenant comes with a built-in set of information stored in properties, such as Given Name, Surname, City, and Postal Code. With Azure AD B2C, you can extend the set of properties stored in each customer account. For more information on defining custom attributes, see [custom attributes \(user flows\)](#) and [custom attributes \(custom policies\)](#).

Microsoft Graph API supports creating and updating a user with extension attributes. Extension attributes in the Graph API are named by using the convention `extension_ApplicationObjectID_attributename`. For example:

```
"extension_831374b3bd5041bfaa54263ec9e050fc_loyaltyNumber": "212342"
```

Code sample

This code sample is a .NET Core console application that uses the [Microsoft Graph SDK](#) to interact with Microsoft Graph API. Its code demonstrates how to call the API to programmatically manage users in an Azure AD B2C tenant. You can [download the sample archive \(*.zip\)](#), [browse the repository](#) on GitHub, or clone the repository:

```
git clone https://github.com/Azure-Samples/ms-identity-dotnetcore-b2c-account-management.git
```

After you've obtained the code sample, configure it for your environment and then build the project:

1. Open the project in [Visual Studio](#) or [Visual Studio Code](#).
2. Open `src/appsettings.json`.
3. In the `appSettings` section, replace `your-b2c-tenant` with the name of your tenant, and `Application (client) ID` and `client secret` with the values for your management application registration (see the [Register a management application](#) section of this article).
4. Open a console window within your local clone of the repo, switch into the `src` directory, then build the project:

```
cd src  
dotnet build
```

5. Run the application with the `dotnet` command:

```
dotnet bin/Debug/netcoreapp3.0/b2c-ms-graph.dll
```

The application displays a list of commands you can execute. For example, get all users, get a single user, delete a user, update a user's password, and bulk import.

Code discussion

The sample code uses the [Microsoft Graph SDK](#), which is designed to simplify building high-quality, efficient, and resilient applications that access Microsoft Graph. So, you don't need to make a direct all the Microsoft Graph API.

Any request to the Microsoft Graph API requires an access token for authentication. The solution makes use of the [Microsoft.Graph.Auth](#) NuGet package that provides an authentication scenario-based wrapper of the Microsoft Authentication Library (MSAL) for use with the Microsoft Graph SDK.

The `RunAsync` method in the *Program.cs* file:

1. Reads application settings from the *appsettings.json* file
2. Initializes the auth provider using [OAuth 2.0 client credentials grant](#) flow. With the client credentials grant flow, the app is able to get an access token to call the Microsoft Graph API.
3. Sets up the Microsoft Graph service client with the auth provider:

```
// Read application settings from appsettings.json (tenant ID, app ID, client secret, etc.)
AppSettings config = AppSettingsFile.ReadFromFile();

// Initialize the client credential auth provider
IConfidentialClientApplication confidentialClientApplication = ConfidentialClientApplicationBuilder
    .Create(config.AppId)
    .WithTenantId(config.TenantId)
    .WithClientSecret(config.ClientSecret)
    .Build();
ClientCredentialProvider authProvider = new ClientCredentialProvider(confidentialClientApplication);

// Set up the Microsoft Graph service client with client credentials
GraphServiceClient graphClient = new GraphServiceClient(authProvider);
```

The initialized *GraphServiceClient* is then used in *UserService.cs* to perform the user management operations. For example, getting a list of the user accounts in the tenant:

```
public static async Task ListUsers(GraphServiceClient graphClient)
{
    Console.WriteLine("Getting list of users...");

    // Get all users (one page)
    var result = await graphClient.Users
        .Request()
        .Select(e => new
    {
        e.DisplayName,
        e.Id,
        e.Identities
    })
    .GetAsync();

    foreach (var user in result.CurrentPage)
    {
        Console.WriteLine(JsonConvert.SerializeObject(user));
    }
}
```

[Make API calls using the Microsoft Graph SDKs](#) includes information on how to read and write information from Microsoft Graph, use `$select` to control the properties returned, provide custom query parameters, and use the `$filter` and `$orderBy` query parameters.

Next steps

For a full index of the Microsoft Graph API operations supported for Azure AD B2C resources, see [Microsoft Graph operations available for Azure AD B2C](#).

Deploy custom policies with Azure Pipelines

2/20/2020 • 6 minutes to read • [Edit Online](#)

By using a continuous integration and delivery (CI/CD) pipeline that you set up in [Azure Pipelines](#), you can include your Azure AD B2C custom policies in your software delivery and code control automation. As you deploy to different Azure AD B2C environments, for example dev, test, and production, we recommend that you remove manual processes and perform automated testing by using Azure Pipelines.

There are three primary steps required for enabling Azure Pipelines to manage custom policies within Azure AD B2C:

1. Create a web application registration in your Azure AD B2C tenant
2. Configure an Azure Repo
3. Configure an Azure Pipeline

IMPORTANT

Managing Azure AD B2C custom policies with an Azure Pipeline currently uses **preview** operations available on the Microsoft Graph API `/beta` endpoint. Use of these APIs in production applications is not supported. For more information, see the [Microsoft Graph REST API beta endpoint reference](#).

Prerequisites

- [Azure AD B2C tenant](#), and credentials for a user in the directory with the [B2C IEF Policy Administrator](#) role
- [Custom policies](#) uploaded to your tenant
- [Management app](#) registered in your tenant with the Microsoft Graph API permission `Policy.ReadWrite.TrustFramework`
- [Azure Pipeline](#), and access to an [Azure DevOps Services project](#)

Client credentials grant flow

The scenario described here makes use of service-to-service calls between Azure Pipelines and Azure AD B2C by using the OAuth 2.0 [client credentials grant flow](#). This grant flow permits a web service like Azure Pipelines (the confidential client) to use its own credentials instead of impersonating a user to authenticate when calling another web service (the Microsoft Graph API, in this case). Azure Pipelines obtains a token non-interactively, then makes requests to the Microsoft Graph API.

Register an application for management tasks

As mentioned in [Prerequisites](#), you need an application registration that your PowerShell scripts--executed by Azure Pipelines--can use for accessing the resources in your tenant.

If you already have an application registration that you use for automation tasks, ensure it's been granted the **Microsoft Graph > Policy > Policy.ReadWrite.TrustFramework** permission within the **API Permissions** of the app registration.

For instructions on registering a management application, see [Manage Azure AD B2C with Microsoft Graph](#).

Configure an Azure Repo

With a management application registered, you're ready to configure a repository for your policy files.

1. Sign in to your Azure DevOps Services organization.
2. [Create a new project](#) or select an existing project.
3. In your project, navigate to **Repos** and select the **Files** page. Select an existing repository or create one for this exercise.
4. Create a folder named *B2CAssets*. Name the required placeholder file *README.md* and **Commit** the file. You can remove this file later, if you like.
5. Add your Azure AD B2C policy files to the *B2CAssets* folder. This includes the *TrustFrameworkBase.xml*, *TrustFrameworkExtensions.xml*, *SignUpOrSignin.xml*, *ProfileEdit.xml*, *PasswordReset.xml*, and any other policies you've created. Record the filename of each Azure AD B2C policy file for use in a later step (they're used as PowerShell script arguments).
6. Create a folder named *Scripts* in the root directory of the repository, name the placeholder file *DeployToB2c.ps1*. Don't commit the file at this point, you'll do so in a later step.
7. Paste the following PowerShell script into *DeployToB2c.ps1*, then **Commit** the file. The script acquires a token from Azure AD and calls the Microsoft Graph API to upload the policies within the *B2CAssets* folder to your Azure AD B2C tenant.

```

[CmdletBinding()]
Param(
    [Parameter(Mandatory = $true)][string]$ClientID,
    [Parameter(Mandatory = $true)][string]$ClientSecret,
    [Parameter(Mandatory = $true)][string]$TenantId,
    [Parameter(Mandatory = $true)][string]$PolicyId,
    [Parameter(Mandatory = $true)][string]$PathToFile
)

try {
    $body = @{
        grant_type = "client_credentials";
        scope = "https://graph.microsoft.com/.default";
        client_id = $ClientID;
        client_secret = $ClientSecret
    }

    $response = Invoke-RestMethod -Uri https://login.microsoftonline.com/$TenantId/oauth2/v2.0/token -Method Post -Body $body
    $token = $response.access_token

    $headers = New-Object "System.Collections.Generic.Dictionary[[String],[String]]"
    $headers.Add("Content-Type", 'application/xml')
    $headers.Add("Authorization", 'Bearer ' + $token)

    $graphuri = 'https://graph.microsoft.com/beta/trustframework/policies/' + $PolicyId + '/$value'
    $policycontent = Get-Content $PathToFile
    $response = Invoke-RestMethod -Uri $graphuri -Method Put -Body $policycontent -Headers $headers

    Write-Host "Policy" $PolicyId "uploaded successfully."
}
catch {
    Write-Host "StatusCode:" $_.Exception.Response.StatusCode.value_
    $_

    $streamReader = [System.IO.StreamReader]::new($_.Exception.Response.GetResponseStream())
    $streamReader.BaseStream.Position = 0
    $streamReader.DiscardBufferData()
    $errResp = $streamReader.ReadToEnd()
    $streamReader.Close()

    $errResp

    exit 1
}

exit 0

```

Configure your Azure pipeline

With your repository initialized and populated with your custom policy files, you're ready to set up the release pipeline.

Create pipeline

1. Sign in to your Azure DevOps Services organization and navigate to your project.
2. In your project, select **Pipelines > Releases > New pipeline**.
3. Under **Select a template**, select **Empty job**.
4. Enter a **Stage name**, for example *DeployCustomPolicies*, then close the pane.
5. Select **Add an artifact**, and under **Source type**, select **Azure Repository**.
 - a. Choose the source repository containing the *Scripts* folder that you populated with the PowerShell script.
 - b. Choose a **Default branch**. If you created a new repository in the previous section, the default branch is *master*.
 - c. Leave the **Default version** setting of *Latest from the default branch*.

- d. Enter a **Source alias** for the repository. For example, *policyRepo*. Do not include any spaces in the alias name.
6. Select **Add**
7. Rename the pipeline to reflect its intent. For example, *Deploy Custom Policy Pipeline*.
8. Select **Save** to save the pipeline configuration.

Configure pipeline variables

1. Select the **Variables** tab.
2. Add the following variables under **Pipeline variables** and set their values as specified:

NAME	VALUE
clientId	Application (client) ID of the application you registered earlier.
clientSecret	The value of the client secret that you created earlier. Change the variable type to secret (select the lock icon).
tenantId	<code>your-b2c-tenant.onmicrosoft.com</code> , where <i>your-b2c-tenant</i> is the name of your Azure AD B2C tenant.

3. Select **Save** to save the variables.

Add pipeline tasks

Next, add a task to deploy a policy file.

1. Select the **Tasks** tab.
2. Select **Agent job**, and then select the plus sign (+) to add a task to the Agent job.
3. Search for and select **PowerShell**. Do not select "Azure PowerShell," "PowerShell on target machines," or another PowerShell entry.
4. Select newly added **PowerShell Script** task.
5. Enter following values for the PowerShell Script task:

- **Task version:** 2.*
- **Display name:** The name of the policy that this task should upload. For example, *B2C_1A_TrustFrameworkBase*.
- **Type:** File Path
- **Script Path:** Select the ellipsis (...), navigate to the *Scripts* folder, and then select the *DeployToB2C.ps1* file.
- **Arguments:**

Enter the following values for **Arguments**. Replace `{alias-name}` with the alias you specified in the previous section.

```
# Before
-ClientId $(clientId) -ClientSecret $(clientSecret) -TenantId $(tenantId) -PolicyId
B2C_1A_TrustFrameworkBase -PathToFile $(System.DefaultWorkingDirectory)/{alias-
name}/B2CAssets/TrustFrameworkBase.xml
```

For example, if the alias you specified is *policyRepo*, the argument line should be:

```
# After  
-ClientID $(clientId) -ClientSecret $(clientSecret) -TenantId $(tenantId) -PolicyId  
B2C_1A_TrustFrameworkBase -PathToFile  
$(System.DefaultWorkingDirectory)/policyRepo/B2CAssets/TrustFrameworkBase.xml
```

6. Select **Save** to save the Agent job.

The task you just added uploads *one* policy file to Azure AD B2C. Before proceeding, manually trigger the job (**Create release**) to ensure that it completes successfully before creating additional tasks.

If the task completes successfully, add deployment tasks by performing the preceding steps for each of the custom policy files. Modify the `-PolicyId` and `-PathToFile` argument values for each policy.

The `PolicyId` is a value found at the start of an XML policy file within the `TrustFrameworkPolicy` node. For example, the `PolicyId` in the following policy XML is *B2C_1A_TrustFrameworkBase*:

```
<TrustFrameworkPolicy  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"  
PolicySchemaVersion="0.3.0.0"  
TenantId="contoso.onmicrosoft.com"  
PolicyId= "B2C_1A_TrustFrameworkBase"  
PublicPolicyUri="http://contoso.onmicrosoft.com/B2C_1A_TrustFrameworkBase">
```

When running the agents and uploading the policy files, ensure they're uploaded in this order:

1. *TrustFrameworkBase.xml*
2. *TrustFrameworkExtensions.xml*
3. *SignUpOrSignin.xml*
4. *ProfileEdit.xml*
5. *PasswordReset.xml*

The Identity Experience Framework enforces this order as the file structure is built on a hierarchical chain.

Test your pipeline

To test your release pipeline:

1. Select **Pipelines** and then **Releases**.
2. Select the pipeline you created earlier, for example *DeployCustomPolicies*.
3. Select **Create release**, then select **Create** to queue the release.

You should see a notification banner that says that a release has been queued. To view its status, select the link in the notification banner, or select it in the list on the **Releases** tab.

Next steps

Learn more about:

- [Service-to-service calls using client credentials](#)
- [Azure DevOps Services](#)

Manage Azure AD B2C custom policies with Azure PowerShell

2/18/2020 • 4 minutes to read • [Edit Online](#)

Azure PowerShell provides several cmdlets for command line- and script-based custom policy management in your Azure AD B2C tenant. Learn how to use the Azure AD PowerShell module to:

- List the custom policies in an Azure AD B2C tenant
- Download a policy from a tenant
- Update an existing policy by overwriting its content
- Upload a new policy to your Azure AD B2C tenant
- Delete a custom policy from a tenant

Prerequisites

- [Azure AD B2C tenant](#), and credentials for a user in the directory with the [B2C IEF Policy Administrator](#) role
- [Custom policies](#) uploaded to your tenant
- [Azure AD PowerShell for Graph preview module](#)

Connect PowerShell session to B2C tenant

To work with custom policies in your Azure AD B2C tenant, you first need to connect your PowerShell session to the tenant by using the [Connect-AzureAD](#) command.

Execute the following command, substituting `{b2c-tenant-name}` with the name of your Azure AD B2C tenant. Sign in with an account that's assigned the [B2C IEF Policy Administrator](#) role in the directory.

```
Connect-AzureAD -Tenant "{b2c-tenant-name}.onmicrosoft.com"
```

Example command output showing a successful sign-in:

```
PS C:\> Connect-AzureAD -Tenant "contosob2c.onmicrosoft.com"

Account          Environment TenantId          TenantDomain          AccountType
-----          -----      -----          -----
azureuser@contoso.com  AzureCloud  00000000-0000-0000-0000-000000000000  contosob2c.onmicrosoft.com  User
```

List all custom policies in the tenant

Discovering custom policies allows an Azure AD B2C administrator to review, manage, and add business logic to their operations. Use the [Get-AzureADMSTrustFrameworkPolicy](#) command to return a list of the IDs of the custom policies in an Azure AD B2C tenant.

```
Get-AzureADMSTrustFrameworkPolicy
```

Example command output:

```
PS C:\> Get-AzureADMSTrustFrameworkPolicy
```

```
Id  
--  
B2C_1A_TrustFrameworkBase  
B2C_1A_TrustFrameworkExtensions  
B2C_1A_signup_signin  
B2C_1A_ProfileEdit  
B2C_1A_PasswordReset
```

Download a policy

After reviewing the list of policy IDs, you can target a specific policy with [Get-AzureADMSTrustFrameworkPolicy](#) to download its content.

```
Get-AzureADMSTrustFrameworkPolicy [-Id <policyId>]
```

In this example, the policy with ID *B2C_1A_signup_signin* is downloaded:

```
PS C:\> Get-AzureADMSTrustFrameworkPolicy -Id B2C_1A_signup_signin
<TrustFrameworkPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.microsoft.com/online/cpim/schemas/2013/06"
PolicySchemaVersion="0.3.0.0" TenantId="contosob2c.onmicrosoft.com" PolicyId="B2C_1A_signup_signin"
PublicPolicyUri="http://contosob2c.onmicrosoft.com/B2C_1A_signup_signin" TenantObjectId="00000000-0000-0000-
0000-000000000000">
<BasePolicy>
  <TenantId>contosob2c.onmicrosoft.com</TenantId>
  <PolicyId>B2C_1A_TrustFrameworkExtensions</PolicyId>
</BasePolicy>
<RelyingParty>
  <DefaultUserJourney ReferenceId="SignUpOrSignIn" />
  <TechnicalProfile Id="PolicyProfile">
    <DisplayName>PolicyProfile</DisplayName>
    <Protocol Name="OpenIdConnect" />
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="displayName" />
      <OutputClaim ClaimTypeReferenceId="givenName" />
      <OutputClaim ClaimTypeReferenceId="surname" />
      <OutputClaim ClaimTypeReferenceId="email" />
      <OutputClaim ClaimTypeReferenceId="objectId" PartnerClaimType="sub" />
      <OutputClaim ClaimTypeReferenceId="identityProvider" />
      <OutputClaim ClaimTypeReferenceId="tenantId" AlwaysUseDefaultValue="true" DefaultValue="
{Policy:TenantObjectId}" />
    </OutputClaims>
    <SubjectNamingInfo ClaimType="sub" />
  </TechnicalProfile>
</RelyingParty>
</TrustFrameworkPolicy>
```

To edit the policy content locally, pipe the command output to a file with the `-OutputFilePath` argument, and then open the file in your favorite editor.

Example command sending output to a file:

```
# Download and send policy output to a file
Get-AzureADMSTrustFrameworkPolicy -Id B2C_1A_signup_signin -OutputFilePath C:\RPPolicy.xml
```

Update an existing policy

After editing a policy file you've created or downloaded, you can publish the updated policy to Azure AD B2C by using the [Set-AzureADMSTrustFrameworkPolicy](#) command.

If you issue the `Set-AzureADMSTrustFrameworkPolicy` command with the ID of a policy that already exists in your Azure AD B2C tenant, the content of that policy is overwritten.

```
Set-AzureADMSTrustFrameworkPolicy [-Id <policyId>] -InputFilePath <inputpolicyfilePath> [-OutputFilePath <outputFilePath>]
```

Example command:

```
# Update an existing policy from file
Set-AzureADMSTrustFrameworkPolicy -Id B2C_1A_signup_signin -InputFilePath C:\B2C_1A_signup_signin.xml
```

For additional examples, see the [Set-AzureADMSTrustFrameworkPolicy](#) command reference.

Upload a new policy

When you make a change to a custom policy that's running in production, you might want to publish multiple versions of the policy for fallback or A/B testing scenarios. Or, you might want to make a copy of an existing policy, modify it with a few small changes, then upload it as a new policy for use by a different application.

Use the [New-AzureADMSTrustFrameworkPolicy](#) command to upload a new policy:

```
New-AzureADMSTrustFrameworkPolicy -InputFilePath <inputpolicyfilePath> [-OutputFilePath <outputFilePath>]
```

Example command:

```
# Add new policy from file
New-AzureADMSTrustFrameworkPolicy -InputFilePath C:\SignUpOrSignIn2.xml
```

Delete a custom policy

To maintain a clean operations life cycle, we recommend that you periodically remove unused custom policies. For example, you might want to remove old policy versions after performing a migration to a new set of policies and verifying the new policies' functionality. Additionally, if you attempt to publish a set of custom policies and receive an error, it might make sense to remove the policies that were created as part of the failed release.

Use the [Remove-AzureADMSTrustFrameworkPolicy](#) command to delete a policy from your tenant.

```
Remove-AzureADMSTrustFrameworkPolicy -Id <policyId>
```

Example command:

```
# Delete an existing policy
Remove-AzureADMSTrustFrameworkPolicy -Id B2C_1A_signup_signin
```

Troubleshoot policy upload

When you try to publish a new custom policy or update an existing policy, improper XML formatting and errors in the policy file inheritance chain can cause validation failures.

For example, here's an attempt at updating a policy with content that contains malformed XML (output is truncated for brevity):

```
PS C:\> Set-AzureADMSTrustFrameworkPolicy -Id B2C_1A_signup_signin -InputFilePath C:\B2C_1A_signup_signin.xml
Set-AzureADMSTrustFrameworkPolicy : Error occurred while executing PutTrustFrameworkPolicy
Code: AADB2C
Message: Validation failed: 1 validation error(s) found in policy "B2C_1A_SIGNUP_SIGNIN" of tenant
"contosob2c.onmicrosoft.com".Schema validation error found at line
14 col 55 in policy "B2C_1A_SIGNUP_SIGNIN" of tenant "contosob2c.onmicrosoft.com": The element 'OutputClaims'
in namespace
'http://schemas.microsoft.com/online/cpim/schemas/2013/06' cannot contain text. List of possible elements
expected: 'OutputClaim' in namespace
'http://schemas.microsoft.com/online/cpim/schemas/2013/06'.
...
...
```

For information about troubleshooting custom policies, see [Troubleshoot Azure AD B2C custom policies and Identity Experience Framework](#).

Next steps

For information about using PowerShell to deploy custom policies as part of a continuous integration/continuous delivery (CI/CD) pipeline, see [Deploy custom policies from an Azure DevOps pipeline](#).

Accessing Azure AD B2C audit logs

2/20/2020 • 6 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) emits audit logs containing activity information about B2C resources, tokens issued, and administrator access. This article provides a brief overview of the information available in audit logs and instructions on how to access this data for your Azure AD B2C tenant.

Audit log events are only retained for **seven days**. Plan to download and store your logs using one of the methods shown below if you require a longer retention period.

NOTE

You can't see user sign-ins for individual Azure AD B2C applications under the **Users** section of the [Azure Active Directory](#) or [Azure AD B2C](#) pages in the Azure portal. The sign-in events there show user activity, but can't be correlated back to the B2C application that the user signed in to. You must use the audit logs for that, as explained further in this article.

Overview of activities available in the B2C category of audit logs

The **B2C** category in audit logs contains the following types of activities:

ACTIVITY TYPE	DESCRIPTION
Authorization	Activities concerning the authorization of a user to access B2C resources (for example, an administrator accessing a list of B2C policies).
Directory	Activities related to directory attributes retrieved when an administrator signs in using the Azure portal.
Application	Create, read, update, and delete (CRUD) operations on B2C applications.
Key	CRUD operations on keys stored in a B2C key container.
Resource	CRUD operations on B2C resources. For example, policies and identity providers.
Authentication	Validation of user credentials and token issuance.

For user object CRUD activities, refer to the **Core Directory** category.

Example activity

This example image from the Azure portal shows the data captured when a user signs in with an external identity provider, in this case, Facebook:

Activity Details: Audit log X

Activity

Date : 2019-09-14T18:13:17.0618117+00:00
 Name : Issue an id_token to the application
 CorrelationId : 00000000-0000-0000-0000-000000000000
 Category : B2C

Activity Status

Status : Success
 Reason : N/A

Initiated By (Actor)

Type : Application
 ObjectId : 00000000-0000-0000-0000-000000000000
 Spn : 00000000-0000-0000-0000-000000000000

Target(s)

Target
 Type : User
 ObjectId : 00000000-0000-0000-0000-000000000000

Additional Details

TenantId : test.onmicrosoft.com
 PolicyId : B2C_1A_signup_signin
 ApplicationId : 00000000-0000-0000-0000-000000000000
 Client : Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36
 IdentityProviderName : facebook
 IdentityProviderApplicationId : 0000000000000000
 ClientIpAddress : 127.0.0.1

The activity details panel contains the following relevant information:

SECTION	FIELD	DESCRIPTION
Activity	Name	Which activity took place. For example, <i>Issue an id_token to the application</i> , which concludes the actual user sign-in.
Initiated By (Actor)	ObjectId	The Object ID of the B2C application that the user is signing in to. This identifier is not visible in the Azure portal, but is accessible via the Microsoft Graph API.
Initiated By (Actor)	Spn	The Application ID of the B2C application that the user is signing in to.
Target(s)	ObjectId	The Object ID of the user that is signing in.
Additional Details	TenantId	The Tenant ID of the Azure AD B2C tenant.
Additional Details	PolicyId	The Policy ID of the user flow (policy) being used to sign the user in.

SECTION	FIELD	DESCRIPTION
Additional Details	ApplicationId	The Application ID of the B2C application that the user is signing in to.

View audit logs in the Azure portal

The Azure portal provides access to the audit log events in your Azure AD B2C tenant.

1. Sign in to the [Azure portal](#)
2. Switch to the directory that contains your Azure AD B2C tenant, and then browse to [Azure AD B2C](#).
3. Under **Activities** in the left menu, select **Audit logs**.

A list of activity events logged over the last seven days is displayed.

The screenshot shows the Azure portal's Audit logs interface. At the top, there are buttons for 'Columns', 'Refresh', 'Download', and 'Troubleshoot'. Below these are four filter dropdowns: 'Category' (set to 'All'), 'Activity Resource Type' (set to 'Authentication'), 'Activity' (set to 'Issue an id_token to the application'), and two 'Target' filters ('Enter target name (case-sensitive)' and 'Enter actor name (case-sensitive)'). A blue 'Apply' button is below the filters. A search bar labeled 'Search to filter items...' is present. The main area displays a table with columns: DATE, TARGET(S), INITIATED BY (ACTOR), and ACTIVITY. Two rows of data are shown:

DATE	TARGET(S)	INITIATED BY (ACTOR)	ACTIVITY
9/14/2019, 11:13:17 AM	User : 11111111-1111-1111-1111-111111111111	00000000-0000-0000-0000-000000000000	Issue an id_token to the application
9/12/2019, 8:47:57 PM	User : 11111111-1111-1111-1111-111111111111	00000000-0000-0000-0000-000000000000	Issue an id_token to the application

Several filtering options are available, including:

- **Activity Resource Type** - Filter by the activity types shown in the table in the [Overview of activities available](#) section.
- **Date** - Filter the date range of the activities shown.

If you select a row in the list, the activity details for the event are displayed.

To download the list of activity events in a comma-separated values (CSV) file, select **Download**.

Get audit logs with the Azure AD reporting API

Audit logs are published to the same pipeline as other activities for Azure Active Directory, so they can be accessed through the [Azure Active Directory reporting API](#). For more information, see [Get started with the Azure Active Directory reporting API](#).

Enable reporting API access

To allow script- or application-based access to the Azure AD reporting API, you need an application registered in your Azure AD B2C tenant with the following API permissions. You can enable these permissions on an existing application registration within your B2C tenant, or create a new one specifically for use with audit log automation.

- Microsoft Graph > Application permissions > AuditLog > AuditLog.Read.All

Follow the steps in the following article to register an application with the required permissions:

[Manage Azure AD B2C with Microsoft Graph](#)

After you've registered an application with the appropriate permissions, see the PowerShell script section later in this article for an example of how you can get activity events with a script.

Access the API

To download Azure AD B2C audit log events via the API, filter the logs on the **B2C** category. To filter by category, use the `filter` query string parameter when you call the Azure AD reporting API endpoint.

```
https://graph.microsoft.com/v1.0/auditLogs/directoryAudits?$filter=loggedByService eq 'B2C' and activityDateTime gt 2019-09-10T02:28:17Z
```

PowerShell script

The following PowerShell script shows an example of how to query the Azure AD reporting API. After querying the API, it prints the logged events to standard output, then writes the JSON output to a file.

You can try this script in the [Azure Cloud Shell](#). Be sure to update it with your application ID, client secret, and the name of your Azure AD B2C tenant.

```
# This script requires an application registration that's granted Microsoft Graph API permission
# https://docs.microsoft.com/azure/active-directory-b2c/microsoft-graph-get-started

# Constants
$ClientID      = "your-client-application-id-here"          # Insert your application's client ID, a GUID
$ClientSecret   = "your-client-application-secret-here"       # Insert your application's client secret
$tenantdomain  = "your-b2c-tenant.onmicrosoft.com"           # Insert your Azure AD B2C tenant domain name

$loginURL      = "https://login.microsoftonline.com"
$resource       = "https://graph.microsoft.com"                  # Microsoft Graph API resource URI
$7daysago      = "{0:s}" -f (get-date).AddDays(-7) + "Z"        # Use 'AddMinutes(-5)' to decrement minutes, for example
Write-Output "Searching for events starting $7daysago"

# Create HTTP header, get an OAuth2 access token based on client id, secret and tenant domain
$body          =
@{grant_type="client_credentials";resource=$resource;client_id=$ClientID;client_secret=$ClientSecret}
$oauth         = Invoke-RestMethod -Method Post -Uri $loginURL/$tenantdomain/oauth2/token?api-version=1.0 -Body
$body

# Parse audit report items, save output to file(s): auditX.json, where X = 0 through n for number of nextLink pages
if ($oauth.access_token -ne $null) {
    $i=0
    $headerParams = @{'Authorization'="$($oauth.token_type) $($oauth.access_token)"}
    $url = "https://graph.microsoft.com/v1.0/auditLogs/directoryAudits?$filter=loggedByService eq 'B2C' and activityDateTime gt " + $7daysago

    # loop through each query page (1 through n)
    Do {
        # display each event on the console window
        Write-Output "Fetching data using Uri: $url"
        $myReport = (Invoke-WebRequest -UseBasicParsing -Headers $headerParams -Uri $url)
        foreach ($event in ($myReport.Content | ConvertFrom-Json).value) {
            Write-Output ($event | ConvertTo-Json)
        }

        # save the query page to an output file
        Write-Output "Save the output to a file audit$i.json"
        $myReport.Content | Out-File -FilePath audit$i.json -Force
        $url = ($myReport.Content | ConvertFrom-Json).'@odata.nextLink'
        $i = $i+1
    } while($url -ne $null)
} else {
    Write-Host "ERROR: No Access Token"
}
```

Here's the JSON representation of the example activity event shown earlier in the article:

```
{
  "id": "B2C_DQ03J_4984536",
  "category": "Authentication",
  "correlationId": "00000000-0000-0000-0000-000000000000",
  "result": "success",
  "resultReason": "N/A",
  "activityDisplayName": "Issue an id_token to the application",
  "activityDateTime": "2019-09-14T18:13:17.0618117Z",
  "loggedByService": "B2C",
  "operationType": "",
  "initiatedBy": {
    "user": null,
    "app": {
      "appId": "00000000-0000-0000-0000-000000000000",
      "displayName": null,
      "servicePrincipalId": null,
      "servicePrincipalName": "00000000-0000-0000-0000-000000000000"
    }
  },
  "targetResources": [
    {
      "id": "00000000-0000-0000-0000-000000000000",
      "displayName": null,
      "type": "User",
      "userPrincipalName": null,
      "groupType": null,
      "modifiedProperties": []
    }
  ],
  "additionalDetails": [
    {
      "key": "TenantId",
      "value": "test.onmicrosoft.com"
    },
    {
      "key": "PolicyId",
      "value": "B2C_1A_signup_signin"
    },
    {
      "key": "ApplicationId",
      "value": "00000000-0000-0000-0000-000000000000"
    },
    {
      "key": "Client",
      "value": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36"
    },
    {
      "key": "IdentityProviderName",
      "value": "facebook"
    },
    {
      "key": "IdentityProviderApplicationId",
      "value": "0000000000000000"
    },
    {
      "key": "ClientIpAddress",
      "value": "127.0.0.1"
    }
  ]
}
```

Next steps

You can automate other administration tasks, for example, [manage Azure AD B2C user accounts with Microsoft Graph](#)

Graph.

Use the Azure portal to create and delete consumer users in Azure AD B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

There might be scenarios in which you want to manually create consumer accounts in your Azure Active Directory B2C (Azure AD B2C) directory. Although consumer accounts in an Azure AD B2C directory are most commonly created when users sign up to use one of your applications, you can create them programmatically and by using the Azure portal. This article focuses on the Azure portal method of user creation and deletion.

To add or delete users, your account must be assigned the *User administrator* or *Global administrator* role.

NOTE

This feature is in public preview.

Types of user accounts

As described in [Overview of user accounts in Azure AD B2C](#), there are three types of user accounts that can be created in an Azure AD B2C directory:

- Work
- Guest
- Consumer

This article focuses on working with **consumer accounts** in the Azure portal. For information about creating and deleting Work and Guest accounts, see [Add or delete users using Azure Active Directory](#).

Create a consumer user

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Under **Manage**, select **Users**.
5. Select **New user**.
6. Select **Create Azure AD B2C user**.
7. Choose a **Sign in method** and enter either an **Email** address or a **Username** for the new user. The sign in method you select here must match the setting you've specified for your Azure AD B2C tenant's *Local account* identity provider (see **Manage > Identity providers** in your Azure AD B2C tenant).
8. Enter a **Name** for the user. This is typically the full name (given and surname) of the user.
9. (Optional) You can **Block sign in** if you wish to delay the ability for the user to sign in. You can enable sign in later by editing the user's **Profile** in the Azure portal.
10. Choose **Auto-generate password** or **Let me create password**.
11. Specify the user's **First name** and **Last name**.
12. Select **Create**.

Unless you've selected **Block sign in**, the user can now sign in using the sign in method (email or username) that you specified.

Delete a consumer user

1. In your Azure AD B2C directory, select **Users**, and then select the user you want to delete.
2. Select **Delete**, and then **Yes** to confirm the deletion.

For details about restoring a user within the first 30 days after deletion, or for permanently deleting a user, see [Restore or remove a recently deleted user using Azure Active Directory](#).

Next steps

For automated user management scenarios, for example migrating users from another identity provider to your Azure AD B2C directory, see [Azure AD B2C: User migration](#).

Secure an Azure API Management API with Azure AD B2C

11/5/2019 • 7 minutes to read • [Edit Online](#)

Learn how to restrict access to your Azure API Management (APIM) API to clients that have authenticated with Azure Active Directory B2C (Azure AD B2C). Follow the steps in this article to create and test an inbound policy in APIM that restricts access to only those requests that include a valid Azure AD B2C-issued access token.

Prerequisites

You need the following resources in place before continuing with the steps in this article:

- [Azure AD B2C tenant](#)
- [Application registered](#) in your tenant
- [User flows created](#) in your tenant
- [Published API](#) in Azure API Management
- [Postman](#) to test secured access (optional)

Get Azure AD B2C application ID

When you secure an API in Azure API Management with Azure AD B2C, you need several values for the [inbound policy](#) that you create in APIM. First, record the application ID of an application you've previously created in your Azure AD B2C tenant. If you're using the application you created in the prerequisites, use the application ID for *webapp1*.

You can use the current **Applications** experience or our new unified **App registrations (Preview)** experience to get the application ID. [Learn more about the new experience](#).

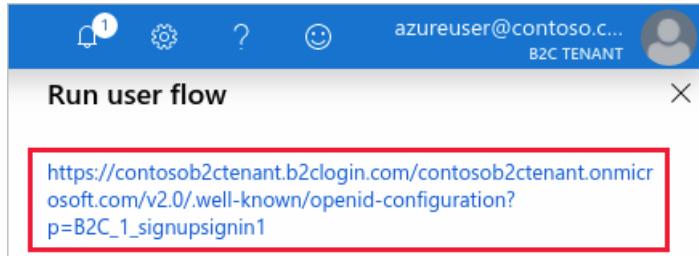
- [Applications](#)
 - [App registrations \(Preview\)](#)
1. Sign in to the [Azure portal](#).
 2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
 3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
 4. Under **Manage**, select **Applications**.
 5. Record the value in the **APPLICATION ID** column for *webapp1* or another application you've previously created.

Get token issuer endpoint

Next, get the well-known config URL for one of your Azure AD B2C user flows. You also need the token issuer endpoint URI you want to support in Azure API Management.

1. Browse to your Azure AD B2C tenant in the [Azure portal](#).
2. Under **Policies**, select **User flows (policies)**.
3. Select an existing policy, for example *B2C_1_signupsignin1*, then select **Run user flow**.
4. Record the URL in hyperlink displayed under the **Run user flow** heading near the top of the page. This

URL is the OpenID Connect well-known discovery endpoint for the user flow, and you use it in the next section when you configure the inbound policy in Azure API Management.



5. Select the hyperlink to browse to the OpenID Connect well-known configuration page.
6. In the page that opens in your browser, record the `issuer` value, for example:

```
https://your-b2c-tenant.b2clogin.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/v2.0/
```

You use this value in the next section when you configure your API in Azure API Management.

You should now have two URLs recorded for use in the next section: the OpenID Connect well-known configuration endpoint URL and the issuer URI. For example:

```
https://yourb2ctenant.b2clogin.com/yourb2ctenant.onmicrosoft.com/v2.0/.well-known/openid-configuration?p=B2C_1_signupsignin1  
https://yourb2ctenant.b2clogin.com/99999999-0000-0000-0000-999999999999/v2.0/
```

Configure inbound policy in Azure API Management

You're now ready to add the inbound policy in Azure API Management that validates API calls. By adding a [JWT validation](#) policy that verifies the audience and issuer in an access token, you can ensure that only API calls with a valid token are accepted.

1. Browse to your Azure API Management instance in the [Azure portal](#).
2. Select **APIs**.
3. Select the API that you want to secure with Azure AD B2C.
4. Select the **Design** tab.
5. Under **Inbound processing**, select `</>` to open the policy code editor.
6. Place the following `<validate-jwt>` tag inside the `<inbound>` policy.
 - a. Update the `url` value in the `<openid-config>` element with your policy's well-known configuration URL.
 - b. Update the `<audience>` element with Application ID of the application you created previously in your B2C tenant (for example, *webapp1*).
 - c. Update the `<issuer>` element with the token issuer endpoint you recorded earlier.

```

<policies>
  <inbound>
    <validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-validation-
error-message="Unauthorized. Access token is missing or invalid.">
      <openid-config
        url="https://yourb2ctenant.b2clogin.com/yourb2ctenant.onmicrosoft.com/v2.0/.well-known/openid-
configuration?p=B2C_1_signupsignin1" />
      <audiences>
        <audience>44444444-0000-0000-444444444444</audience>
      </audiences>
      <issuers>
        <issuer>https://yourb2ctenant.b2clogin.com/99999999-0000-0000-0000-
999999999999/v2.0/</issuer>
      </issuers>
    </validate-jwt>
    <base />
  </inbound>
  <backend> <base /> </backend>
  <outbound> <base /> </outbound>
  <on-error> <base /> </on-error>
</policies>

```

Validate secure API access

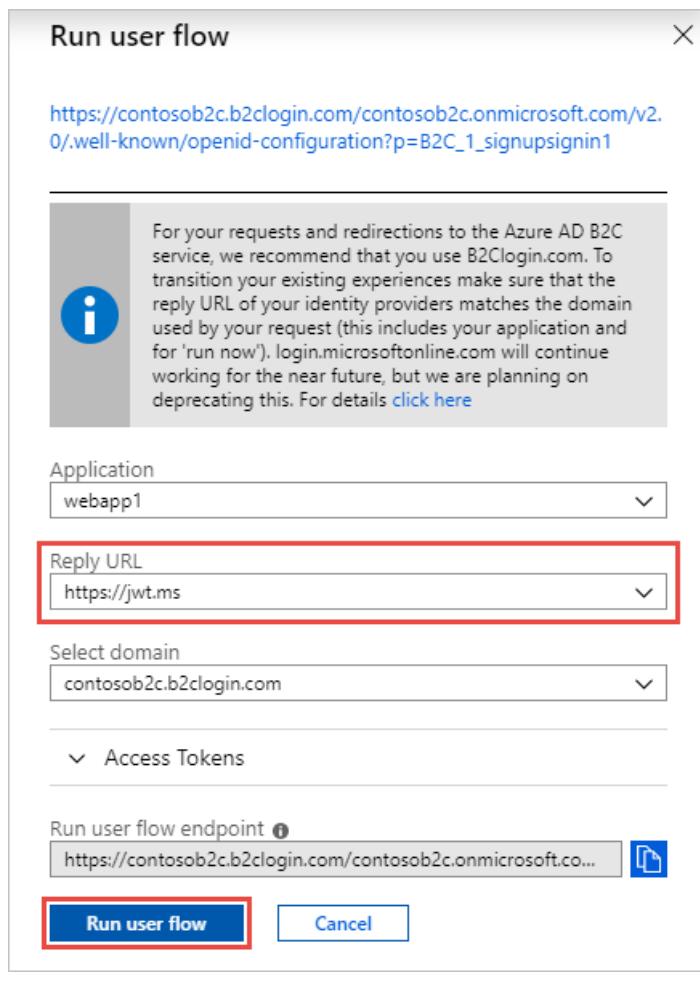
To ensure only authenticated callers can access your API, you can validate your Azure API Management configuration by calling the API with [Postman](#).

To call the API, you need both an access token issued by Azure AD B2C, and an APIM subscription key.

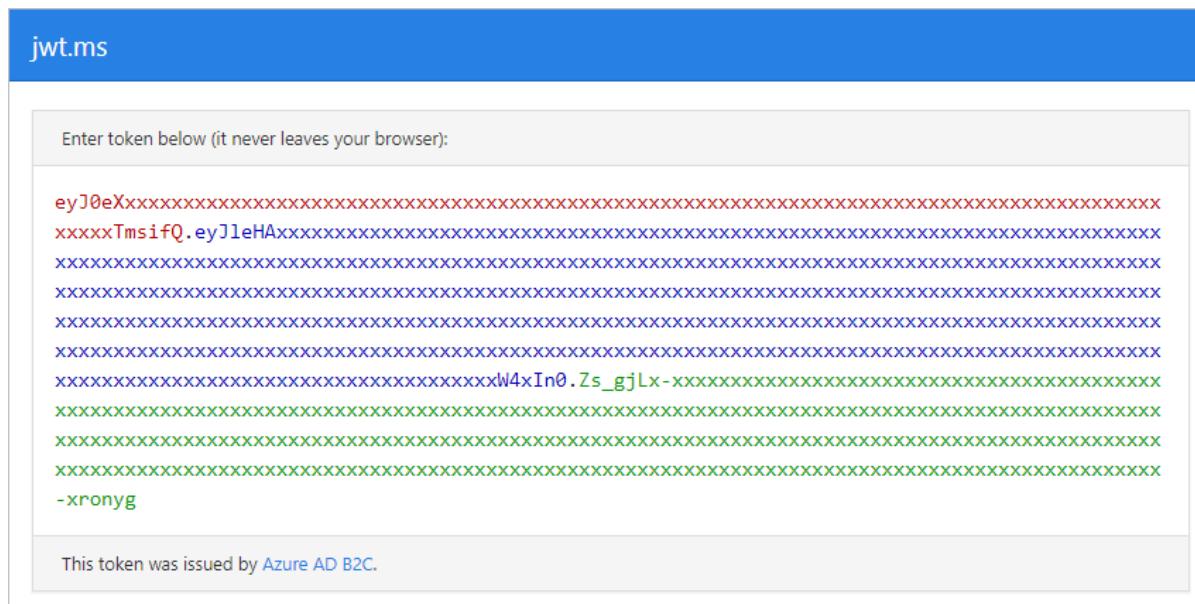
Get an access token

You first need a token issued by Azure AD B2C to use in the `Authorization` header in Postman. You can get one by using the **Run now** feature of your sign-up/sign-in user flow you should have created as one of the prerequisites.

1. Browse to your Azure AD B2C tenant in the [Azure portal](#).
2. Under **Policies**, select **User flows (policies)**.
3. Select an existing sign-up/sign-in user flow, for example *B2C_1_signupsignin1*.
4. For **Application**, select *webapp1*.
5. For **Reply URL**, choose `https://jwt.ms`.
6. Select **Run user flow**.



7. Complete the sign-in process. You should be redirected to <https://jwt.ms>.
8. Record the encoded token value displayed in your browser. You use this token value for the Authorization header in Postman.



Get API subscription key

A client application (in this case, Postman) that calls a published API must include a valid API Management subscription key in its HTTP requests to the API. To get a subscription key to include in your Postman HTTP request:

1. Browse to your Azure API Management service instance in the [Azure portal](#).
2. Select **Subscriptions**.

3. Select the ellipsis for **Product: Unlimited**, then select **Show/hide keys**.
4. Record the **PRIMARY KEY** for the product. You use this key for the `Ocp-Apim-Subscription-Key` header in your HTTP request in Postman.

DISPLAY NAME	PRIMARY KEY	SECONDARY KEY	SCOPE	STATE	OWNER	ALLOW TRACING
.....	Product: Starter	Active	Administrator	<input checked="" type="checkbox"/>
.....	Product: Unlimited	Active	Administrator	Show/hide keys
Built-in all-access	Service	Active		Activate subscription

Test a secure API call

With the access token and APIM subscription key recorded, you're now ready to test whether you've correctly configured secure access to the API.

1. Create a new `GET` request in [Postman](#). For the request URL, specify the speakers list endpoint of the API you published as one of the prerequisites. For example:

```
https://contosoapim.azure-api.net/conference/speakers
```

2. Next, add the following headers:

KEY	VALUE
Authorization	Encoded token value you recorded earlier, prefixed with <code>Bearer</code> (include the space after "Bearer")
Ocp-Apim-Subscription-Key	APIM subscription key you recorded earlier

Your **GET** request URL and **Headers** should appear similar to:

3. Select the **Send** button in Postman to execute the request. If you've configured everything correctly, you should be presented with a JSON response with a collection of conference speakers (shown here truncated):

```
{
  "collection": {
    "version": "1.0",
    "href": "https://conferenceapi.azurewebsites.net:443/speakers",
    "links": [],
    "items": [
      {
        "href": "https://conferenceapi.azurewebsites.net/speaker/1",
        "data": [
          {
            "name": "Name",
            "value": "Scott Guthrie"
          }
        ],
        "links": [
          {
            "rel": "http://tavis.net/rels/sessions",
            "href": "https://conferenceapi.azurewebsites.net/speaker/1/sessions"
          }
        ]
      },
      [...]
    ]
  }
}
```

Test an insecure API call

Now that you've made a successful request, test the failure case to ensure that calls to your API with an *invalid* token are rejected as expected. One way to perform the test is to add or change a few characters in the token value, then execute the same `GET` request as before.

1. Add several characters to the token value to simulate an invalid token. For example, add "INVALID" to the token value:

▼ Headers (2)		
	KEY	VALUE
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6...
<input checked="" type="checkbox"/>	Ocp-Apim-Subscription-Key	bb5b2f8703df4cc292271a4ba01a0530

2. Select the **Send** button to execute the request. With an invalid token, the expected result is a `401` unauthorized status code:

```
{
  "statusCode": 401,
  "message": "Unauthorized. Access token is missing or invalid."
}
```

If you see the `401` status code, you've verified that only callers with a valid access token issued by Azure AD B2C can make successful requests to your Azure API Management API.

Support multiple applications and issuers

Several applications typically interact with a single REST API. To enable your API to accept tokens intended for multiple applications, add their application IDs to the `<audiences>` element in the APIM inbound policy.

```
<!-- Accept tokens intended for these recipient applications -->
<audiences>
    <audience>44444444-0000-0000-0000-444444444444</audience>
    <audience>66666666-0000-0000-0000-666666666666</audience>
</audiences>
```

Similarly, to support multiple token issuers, add their endpoint URIs to the `<issuers>` element in the APIM inbound policy.

```
<!-- Accept tokens from multiple issuers -->
<issuers>
    <issuer>https://yourb2ctenant.b2clogin.com/99999999-0000-0000-0000-999999999999/v2.0/</issuer>
    <issuer>https://login.microsoftonline.com/99999999-0000-0000-0000-999999999999/v2.0/</issuer>
</issuers>
```

Migrate to b2clogin.com

If you have an APIM API that validates tokens issued by the legacy `login.microsoftonline.com` endpoint, you should migrate the API and the applications that call it to use tokens issued by [b2clogin.com](#).

You can follow this general process to perform a staged migration:

1. Add support in your APIM inbound policy for tokens issued by both b2clogin.com and `login.microsoftonline.com`.
2. Update your applications one at a time to obtain tokens from the b2clogin.com endpoint.
3. Once all of your applications are correctly obtaining tokens from b2clogin.com, remove support for `login.microsoftonline.com`-issued tokens from the API.

The following example APIM inbound policy illustrates how to accept tokens issued by both b2clogin.com and `login.microsoftonline.com`. Additionally, it supports API requests from two applications.

```
<policies>
    <inbound>
        <validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-validation-error-message="Unauthorized. Access token is missing or invalid.">
            <openid-config url="https://yourb2ctenant.b2clogin.com/yourb2ctenant.onmicrosoft.com/v2.0/.well-known/openid-configuration?p=B2C_1_signupsignin1" />
            <audiences>
                <audience>44444444-0000-0000-0000-444444444444</audience>
                <audience>66666666-0000-0000-0000-666666666666</audience>
            </audiences>
            <issuers>
                <issuer>https://login.microsoftonline.com/99999999-0000-0000-0000-999999999999/v2.0/</issuer>
                <issuer>https://yourb2ctenant.b2clogin.com/99999999-0000-0000-0000-999999999999/v2.0/</issuer>
            </issuers>
        </validate-jwt>
        <base />
    </inbound>
    <backend> <base /> </backend>
    <outbound> <base /> </outbound>
    <on-error> <base /> </on-error>
</policies>
```

Next steps

For additional details on Azure API Management policies, see the [APIM policy reference index](#).

You can find information about migrating OWIN-based web APIs and their applications to b2clogin.com in

[Migrate an OWIN-based web API to b2clogin.com.](#)

Manage user access in Azure Active Directory B2C

2/20/2020 • 9 minutes to read • [Edit Online](#)

This article discusses how to manage user access to your applications by using Azure Active Directory B2C (Azure AD B2C). Access management in your application includes:

- Identifying minors and controlling user access to your application.
- Requiring parental consent for minors to use your applications.
- Gathering birth and country/region data from users.
- Capturing a terms-of-use agreement and gating access.

NOTE

In Azure Active Directory B2C, [custom policies](#) are designed primarily to address complex scenarios. For most scenarios, we recommend that you use built-in [user flows](#).

Control minor access

Applications and organizations may decide to block minors from using applications and services that are not targeted to this audience. Alternatively, applications and organizations may decide to accept minors and subsequently manage the parental consent, and deliver permissible experiences for minors as dictated by business rules and allowed by regulation.

If a user is identified as a minor, you can set the user flow in Azure AD B2C to one of three options:

- **Send a signed JWT id_token back to the application:** The user is registered in the directory, and a token is returned to the application. The application then proceeds by applying business rules. For example, the application may proceed with a parental consent process. To use this method, choose to receive the **ageGroup** and **consentProvidedForMinor** claims from the application.
- **Send an unsigned JSON token to the application:** Azure AD B2C notifies the application that the user is a minor and provides the status of the user's parental consent. The application then proceeds by applying business rules. A JSON token does not complete a successful authentication with the application. The application must process the unauthenticated user according to the claims included in the JSON token, which may include **name**, **email**, **ageGroup**, and **consentProvidedForMinor**.
- **Block the user:** If a user is a minor, and parental consent has not been provided, Azure AD B2C can notify the user that they are blocked. No token is issued, access is blocked, and the user account is not created during a registration journey. To implement this notification, you provide a suitable HTML/CSS content page to inform the user and present appropriate options. No further action is needed by the application for new registrations.

Get parental consent

Depending on application regulation, parental consent might need to be granted by a user who is verified as an adult. Azure AD B2C does not provide an experience to verify an individual's age and then allow a verified adult to grant parental consent to a minor. This experience must be provided by the application or another service provider.

The following is an example of a user flow for gathering parental consent:

1. A [Microsoft Graph API](#) operation identifies the user as a minor and returns the user data to the application

in the form of an unsigned JSON token.

2. The application processes the JSON token and shows a screen to the minor, notifying them that parental consent is required and requesting the consent of a parent online.
3. Azure AD B2C shows a sign-in journey that the user can sign in to normally and issues a token to the application that is set to include **legalAgeGroupClassification** = “**minorWithParentalConsent**”. The application collects the email address of the parent and verifies that the parent is an adult. To do so, it uses a trusted source, such as a national ID office, license verification, or credit card proof. If verification is successful, the application prompts the minor to sign in by using the Azure AD B2C user flow. If consent is denied (for example, if **legalAgeGroupClassification** = “**minorWithoutParentalConsent**”), Azure AD B2C returns a JSON token (not a login) to the application to restart the consent process. It is optionally possible to customize the user flow so that a minor or an adult can regain access to a minor’s account by sending a registration code to the minor’s email address or the adult’s email address on record.
4. The application offers an option to the minor to revoke consent.
5. When either the minor or the adult revokes consent, the Microsoft Graph API can be used to change **consentProvidedForMinor** to **denied**. Alternatively, the application may choose to delete a minor whose consent has been revoked. It is optionally possible to customize the user flow so that the authenticated minor (or parent that is using the minor’s account) can revoke consent. Azure AD B2C records **consentProvidedForMinor** as **denied**.

For more information about **legalAgeGroupClassification**, **consentProvidedForMinor**, and **ageGroup**, see [User resource type](#). For more information about custom attributes, see [Use custom attributes to collect information about your consumers](#). When you address extended attributes by using the Microsoft Graph API, you must use the long version of the attribute, such as *extension_18b70cf9bb834edd8f38521c2583cd86_dateOfBirth: 2011-01-01T00:00:00Z*.

Gather date of birth and country/region data

Applications may rely on Azure AD B2C to gather the date of birth (DOB) and country/region information from all users during registration. If this information does not already exist, the application can request it from the user during the next authentication (sign-in) journey. Users cannot proceed without providing their DOB and country/region information. Azure AD B2C uses the information to determine whether the individual is considered a minor according to the regulatory standards of that country/region.

A customized user flow can gather DOB and country/region information and use Azure AD B2C claims transformation to determine the **ageGroup** and persist the result (or persist the DOB and country/region information directly) in the directory.

The following steps show the logic that is used to calculate **ageGroup** from the user's date of birth:

1. Try to find the country by the country code in the list. If the country is not found, fall back to **Default**.
2. If the **MinorConsent** node is present in the country element:
 - a. Calculate the date that the user must have been born on to be considered an adult. For example, if the current date is March 14, 2015, and **MinorConsent** is 18, the birth date must be no later than March 14, 2000.
 - b. Compare the minimum birth date with the actual birth date. If the minimum birth date is before the user's birth date, the calculation returns **Minor** as the age group calculation.
3. If the **MinorNoConsentRequired** node is present in the country element, repeat steps 2a and 2b using the value from **MinorNoConsentRequired**. The output of 2b returns **MinorNoConsentRequired** if the minimum birth date is before the user's birth date.

4. If neither calculation returns true, the calculation returns **Adult**.

If an application has reliably gathered DOB or country/region data by other methods, the application may use the Graph API to update the user record with this information. For example:

- If a user is known to be an adult, update the directory attribute **ageGroup** with a value of **Adult**.
- If a user is known to be a minor, update the directory attribute **ageGroup** with a value of **Minor** and set **consentProvidedForMinor**, as appropriate.

For more information about gathering DOB data, see [Use age gating in Azure AD B2C](#).

Capture terms of use agreement

When you develop your application, you ordinarily capture users' acceptance of terms of use within their applications with no, or only minor, participation from the user directory. It is possible, however, to use an Azure AD B2C user flow to gather a user's acceptance of terms of use, restrict access if acceptance is not granted, and enforce acceptance of future changes to the terms of use, based on the date of the latest acceptance and the date of the latest version of the terms of use.

Terms of Use may also include "Consent to share data with third parties." Depending on local regulations and business rules, you can gather a user's acceptance of both conditions combined, or you can allow the user to accept one condition and not the other.

The following steps describe how you can manage terms of use:

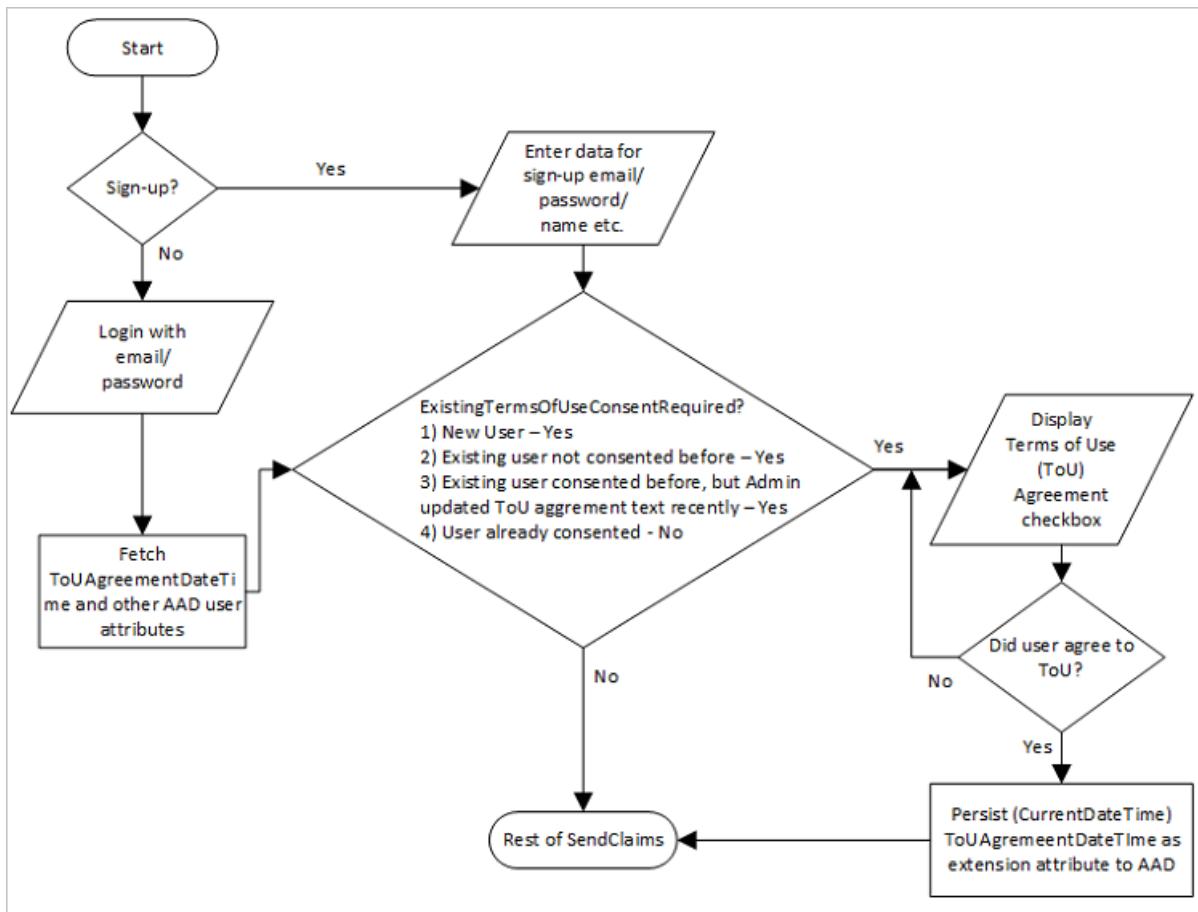
1. Record the acceptance of the terms of use and the date of acceptance by using the Graph API and extended attributes. You can do so by using both built-in and custom user flows. We recommend that you create and use the **extension_termsOfUseConsentDateTime** and **extension_termsOfUseConsentVersion** attributes.
2. Create a required check box labeled "Accept Terms of Use," and record the result during signup. You can do so by using both built-in and custom user flows.
3. Azure AD B2C stores the terms of use agreement and the user's acceptance. You can use the Graph API to query for the status of any user by reading the extension attribute that's used to record the response (for example, read **termsOfUseTestUpdateDateTime**). You can do so by using both built-in and custom user flows.
4. Require acceptance of updated terms of use by comparing the date of acceptance to the date of the latest version of the terms of use. You can compare the dates only by using a custom user flow. Use the extended attribute **extension_termsOfUseConsentDateTime**, and compare the value to the claim of **termsOfUseTextUpdateDateTime**. If the acceptance is old, force a new acceptance by displaying a self-asserted screen. Otherwise, block access by using policy logic.
5. Require acceptance of updated terms of use by comparing the version number of the acceptance to the most recent accepted version number. You can compare version numbers only by using a custom user flow. Use the extended attribute **extension_termsOfUseConsentDateTime**, and compare the value to the claim of **extension_termsOfUseConsentVersion**. If the acceptance is old, force a new acceptance by displaying a self-asserted screen. Otherwise, block access by using policy logic.

You can capture terms of use acceptance under the following scenarios:

- A new user is signing up. The terms of use are displayed, and the acceptance result is stored.
- A user is signing in who has previously accepted the latest or active terms of use. The terms of use are not displayed.
- A user is signing in who has not already accepted the latest or active terms of use. The terms of use are displayed, and the acceptance result is stored.

- A user is signing in who has already accepted an older version of the terms of use, which are now updated to the latest version. The terms of use are displayed, and the acceptance result is stored.

The following image shows the recommended user flow:



The following is an example of a DateTime based terms of use consent in a claim:

```

<ClaimsTransformations>
    <ClaimsTransformation Id="GetNewUserAgreeToTermsOfUseConsentDateTime"
TransformationMethod="GetCurrentDateTime">
        <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="extension_termsOfUseConsentDateTime"
TransformationClaimType="currentDateTime" />
        </OutputClaims>
    </ClaimsTransformation>
    <ClaimsTransformation Id="IsTermsOfUseConsentRequired" TransformationMethod="IsTermsOfUseConsentRequired">
        <InputClaims>
            <InputClaim ClaimTypeReferenceId="extension_termsOfUseConsentDateTime"
TransformationClaimType="termsOfUseConsentDateTime" />
        </InputClaims>
        <InputParameters>
            <InputParameter Id="termsOfUseTextUpdateDateTime" DataType="dateTime" Value="2098-01-30T23:03:45" />
        </InputParameters>
        <OutputClaims>
            <OutputClaim ClaimTypeReferenceId="termsOfUseConsentRequired" TransformationClaimType="result" />
        </OutputClaims>
    </ClaimsTransformation>
</ClaimsTransformations>

```

The following is an example of a Version based terms of use consent in a claim:

```

<ClaimsTransformations>
  <ClaimsTransformation Id="GetEmptyTermsOfUseConsentVersionForNewUser"
TransformationMethod="CreateStringClaim">
    <InputParameters>
      <InputParameter Id="value" DataType="string" Value="" />
    </InputParameters>
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="extension_termsOfUseConsentVersion"
TransformationClaimType="createdClaim" />
    </OutputClaims>
  </ClaimsTransformation>
  <ClaimsTransformation Id="GetNewUserAgreeToTermsOfUseConsentVersion"
TransformationMethod="CreateStringClaim">
    <InputParameters>
      <InputParameter Id="value" DataType="string" Value="V1" />
    </InputParameters>
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="extension_termsOfUseConsentVersion"
TransformationClaimType="createdClaim" />
    </OutputClaims>
  </ClaimsTransformation>
  <ClaimsTransformation Id="IsTermsOfUseConsentRequiredForVersion"
TransformationMethod="CompareClaimToValue">
    <InputClaims>
      <InputClaim ClaimTypeReferenceId="extension_termsOfUseConsentVersion"
TransformationClaimType="inputClaim1" />
    </InputClaims>
    <InputParameters>
      <InputParameter Id="compareTo" DataType="string" Value="V1" />
      <InputParameter Id="operator" DataType="string" Value="not equal" />
      <InputParameter Id="ignoreCase" DataType="string" Value="true" />
    </InputParameters>
    <OutputClaims>
      <OutputClaim ClaimTypeReferenceId="termsOfUseConsentRequired"
TransformationClaimType="outputClaim" />
    </OutputClaims>
  </ClaimsTransformation>
</ClaimsTransformations>

```

Next steps

- To learn how to delete and export user data, see [Manage user data](#).
- For an example custom policy that implements a terms of use prompt, see [A B2C IEF Custom Policy - Sign Up and Sign In with 'Terms of Use' prompt](#).

Manage user data in Azure Active Directory B2C

2/20/2020 • 3 minutes to read • [Edit Online](#)

This article discusses how you can manage the user data in Azure Active Directory B2C (Azure AD B2C) by using the operations that are provided by the [Microsoft Graph API](#). Managing user data includes deleting or exporting data from audit logs.

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Delete user data

User data is stored in the Azure AD B2C directory and in the audit logs. All user audit data is retained for 7 days in Azure AD B2C. If you want to delete user data within that 7-day period, you can use the [Delete a user](#) operation. A DELETE operation is required for each of the Azure AD B2C tenants where data might reside.

Every user in Azure AD B2C is assigned an object ID. The object ID provides an unambiguous identifier for you to use to delete user data in Azure AD B2C. Depending on your architecture, the object ID can be a useful correlation identifier across other services, such as financial, marketing, and customer relationship management databases.

The most accurate way to get the object ID for a user is to obtain it as part of an authentication journey with Azure AD B2C. If you receive a valid request for data from a user by using other methods, an offline process, such as a search by a customer service support agent, might be necessary to find the user and note the associated object ID.

The following example shows a possible data-deletion flow:

1. The user signs in and selects **Delete my data**.
2. The application offers an option to delete the data within an administration section of the application.
3. The application forces an authentication to Azure AD B2C. Azure AD B2C provides a token with the object ID of the user back to the application.
4. The token is received by the application, and the object ID is used to delete the user data through a call to the Microsoft Graph API. The Microsoft Graph API deletes the user data and returns a status code of 200 OK.
5. The application orchestrates the deletion of user data in other organizational systems as needed by using the object ID or other identifiers.
6. The application confirms the deletion of data and provides next steps to the user.

Export customer data

The process of exporting customer data from Azure AD B2C is similar to the deletion process.

Azure AD B2C user data is limited to:

- **Data stored in the Azure Active Directory:** You can retrieve data in an Azure AD B2C authentication user journey by using the object ID or any sign-in name, such as an email address or username.
- **User-specific audit events report:** You can index data by using the object ID.

In the following example of an export data flow, the steps that are described as being performed by the application can also be performed by either a backend process or a user with an administrator role in the directory:

1. The user signs in to the application. Azure AD B2C enforces authentication with Azure Multi-Factor Authentication if needed.
2. The application uses the user credentials to call a Microsoft Graph API operation to retrieve the user attributes. The Microsoft Graph API provides the attribute data in JSON format. Depending on the schema, you can set the ID token contents to include all personal data about a user.
3. The application retrieves the user audit activity. The Microsoft Graph API provides the event data to the application.
4. The application aggregates the data and makes it available to the user.

Next steps

To learn how to manage how users access your application, see [Manage user access](#).

Migrate users to Azure AD B2C

2/20/2020 • 4 minutes to read • [Edit Online](#)

Migrating from another identity provider to Azure Active Directory B2C (Azure AD B2C) might also require migrating existing user accounts. Two migration methods are discussed here, *bulk import* and *seamless migration*. With either approach, you're required to write an application or script that uses the [Microsoft Graph API](#) to create user accounts in Azure AD B2C.

Bulk import

In the bulk import flow, your migration application performs these steps for each user account:

1. Read the user account from the old identity provider, including its current credentials (username and password).
2. Create a corresponding account in your Azure AD B2C directory with the current credentials.

Use the bulk import flow in either of these two situations:

- You have access to a user's plaintext credentials (their username and password).
- The credentials are encrypted, but you can decrypt them.

For information about programmatically creating user accounts, see [Manage Azure AD B2C user accounts with Microsoft Graph](#).

Seamless migration

Use the seamless migration flow if plaintext passwords in the old identity provider are not accessible. For example, when:

- The password is stored in a one-way encrypted format, such as with a hash function.
- The password is stored by the legacy identity provider in a way that you can't access. For example, when the identity provider validates credentials by calling a web service.

The seamless migration flow still requires bulk migration of user accounts, but then uses a [custom policy](#) to query a [REST API](#) (which you create) to set each users' password at first sign-in.

The seamless migration flow thus has two phases: *bulk import* and *set credentials*.

Phase 1: Bulk import

1. Your migration application reads the user accounts from the old identity provider.
2. The migration application creates corresponding user accounts in your Azure AD B2C directory, but *does not set passwords*.

Phase 2: Set credentials

After bulk migration of the accounts is complete, your custom policy and REST API then perform the following when a user signs in:

1. Read the Azure AD B2C user account corresponding to the email address entered.
2. Check whether the account is flagged for migration by evaluating a boolean extension attribute.
 - If the extension attribute returns `True`, call your REST API to validate the password against the legacy identity provider.
 - If the REST API determines the password is incorrect, return a friendly error to the user.

- If the REST API determines the password is correct, write the password to the Azure AD B2C account and change the boolean extension attribute to `False`.
- If the boolean extension attribute returns `False`, continue the sign-in process as normal.

To see an example custom policy and REST API, see the [seamless user migration sample](#) on GitHub.

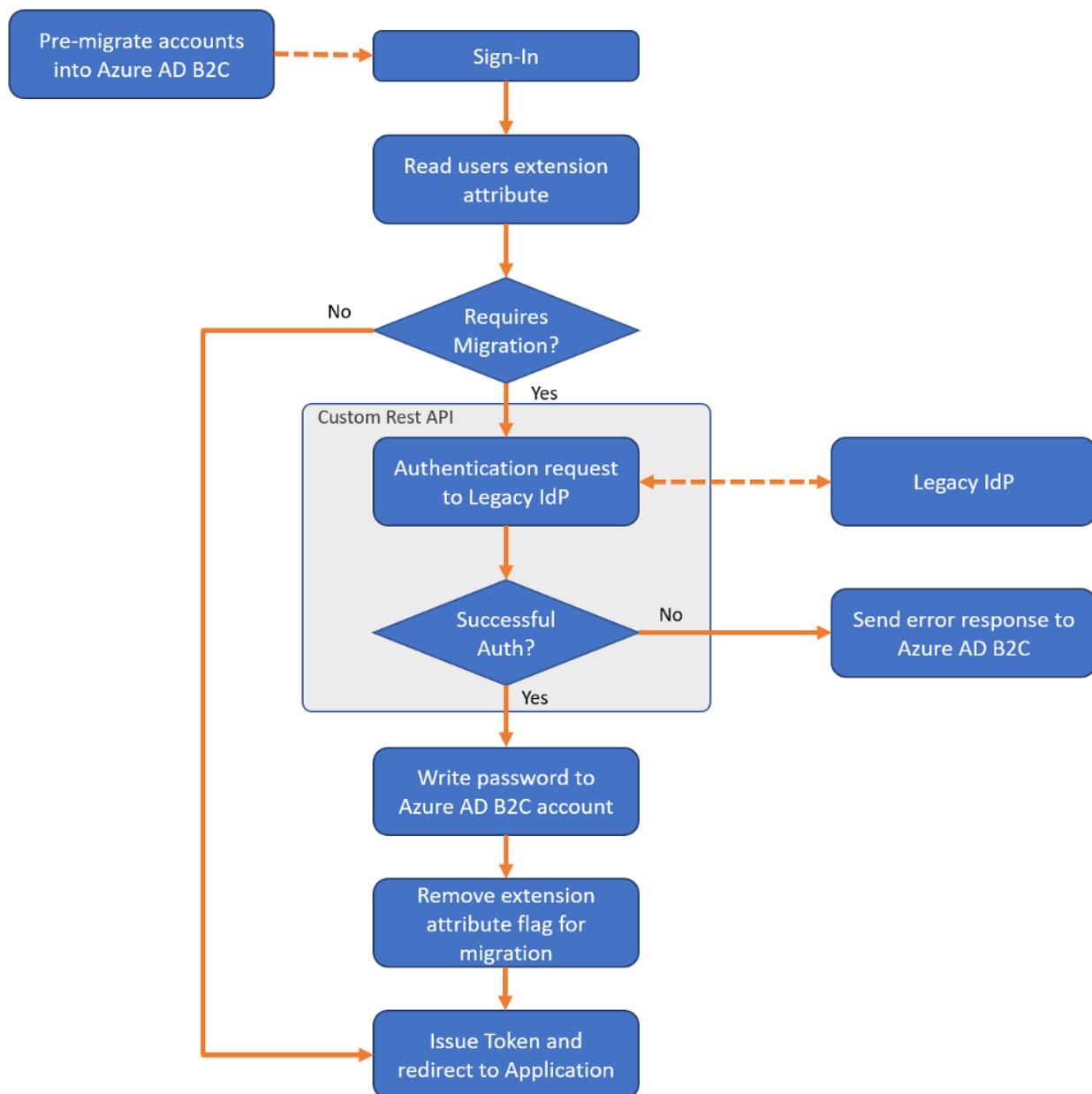


Diagram: Seamless migration flow

Best practices

Security

The seamless migration approach uses your own custom REST API to validate a user's credentials against the legacy identity provider.

You must protect your REST API against brute-force attacks. An attacker can submit several passwords in the hope of eventually guessing a user's credentials. To help defeat such attacks, stop serving requests to your REST API when the number of sign-in attempts passes a certain threshold. Also, secure the communication between Azure AD B2C and your REST API by using a [client certificate](#).

User attributes

Not all information in the legacy identity provider should be migrated to your Azure AD B2C directory. Identify

the appropriate set of user attributes to store in Azure AD B2C before migrating.

- **DO** store in Azure AD B2C
 - Username, password, email addresses, phone numbers, membership numbers/identifiers.
 - Consent markers for privacy policy and end-user license agreements.
- **DO NOT** store in Azure AD B2C
 - Sensitive data like credit card numbers, social security numbers (SSN), medical records, or other data regulated by government or industry compliance bodies.
 - Marketing or communication preferences, user behaviors, and insights.

Directory clean-up

Before you start the migration process, take the opportunity to clean up your directory.

- Identify the set of user attributes to be stored in Azure AD B2C, and migrate only what you need. If necessary, you can create [custom attributes](#) to store more data about a user.
- If you're migrating from an environment with multiple authentication sources (for example, each application has its own user directory), migrate to a unified account in Azure AD B2C.
- If multiple applications have different usernames, you can store all of them in an Azure AD B2C user account by using the identities collection. With regard to the password, let the user choose one and set it in the directory. For example, with the seamless migration, only the chosen password should be stored in the Azure AD B2C account.
- Remove unused user accounts before migration, or do not migrate stale accounts.

Password policy

If the accounts you're migrating have weaker password strength than the [strong password strength](#) enforced by Azure AD B2C, you can disable the strong password requirement. For more information, see [Password policy property](#).

Next steps

The [azure-ad-b2c/user-migration](#) repository on GitHub contains a seamless migration custom policy example and REST API code sample:

[Seamless user migration custom policy & REST API code sample](#)

Developer notes for custom policies in Azure Active Directory B2C

2/12/2020 • 4 minutes to read • [Edit Online](#)

Custom policy configuration in Azure Active Directory B2C is now generally available. This method of configuration is targeted at advanced identity developers building complex identity solutions. Custom policies make the power of the Identity Experience Framework available in Azure AD B2C tenants. Advanced identity developers using custom policies should plan to invest some time completing walk-throughs and reading reference documents.

While most of the custom policy options available are now generally available, there are underlying capabilities, such as technical profile types and content definition APIs that are at different stages in the software lifecycle. Many more are coming. The table below specifies the level of availability at a more granular level.

Features that are generally available

- Author and upload custom authentication user journeys by using custom policies.
 - Describe user journeys step-by-step as exchanges between claims providers.
 - Define conditional branching in user journeys.
- Interoperate with REST API-enabled services in your custom authentication user journeys.
- Federate with identity providers that are compliant with the OpenIDConnect protocol.
- Federate with identity providers that adhere to the SAML 2.0 protocol.

Responsibilities of custom policy feature-set developers

Manual policy configuration grants lower-level access to the underlying platform of Azure AD B2C and results in the creation of a unique, trust framework. The many possible permutations of custom identity providers, trust relationships, integrations with external services, and step-by-step workflows require a methodical approach to design and configuration.

Developers consuming the custom policy feature set should adhere to the following guidelines:

- Become familiar with the configuration language of the custom policies and key/secrets management. For more information, see [TrustFrameworkPolicy](#).
- Take ownership of scenarios and custom integrations. Document your work and inform your live site organization.
- Perform methodical scenario testing.
- Follow software development and staging best practices with a minimum of one development and testing environment and one production environment.
- Stay informed about new developments from the identity providers and services you integrate with. For example, keep track of changes in secrets and of scheduled and unscheduled changes to the service.
- Set up active monitoring, and monitor the responsiveness of production environments. For more information about integrating with Application Insights, see [Azure Active Directory B2C: Collecting Logs](#).
- Keep contact email addresses current in the Azure subscription, and stay responsive to the Microsoft live-site team emails.
- Take timely action when advised to do so by the Microsoft live-site team.

Terms for features in public preview

- We encourage you to use the public preview features for evaluation purposes only.
- Service level agreements (SLAs) do not apply to the public preview features.
- Support requests for public preview features can be filed through regular support channels.

Features by stage and known issues

Custom policy/Identity Experience Framework capabilities are under constant and rapid development. The following table is an index of features and component availability.

Identity Providers, Tokens, Protocols

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
IDP-OpenIDConnect			X	For example, Google+.
IDP-OAUTH2			X	For example, Facebook.
IDP-OAUTH1 (twitter)		X		For example, Twitter.
IDP-OAUTH1 (ex-twitter)				Not supported
IDP-SAML			X	For example, Salesforce, ADFS.
IDP-WSFED	X			
Relying Party OAUTH1				Not supported.
Relying Party OAUTH2			X	
Relying Party OIDC			X	
Relying Party SAML		X		
Relying Party WSFED	X			
REST API with basic and certificate auth			X	For example, Azure Logic Apps.

Component Support

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Azure Multi Factor Authentication			X	
Azure Active Directory as local directory			X	

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Azure Email subsystem for email verification			X	
Multi-language support			X	
Predicate Validations			X	For example, password complexity.
Using third party email service providers		X		

Content Definition

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Error page, api.error			X	
IDP selection page, api.idpselections			X	
IDP selection for signup, api.idpselections.signup			X	
Forgot Password, api.localaccountpasswordreset			X	
Local Account Sign-in, api.localaccountsigin			X	
Local Account Sign-up, api.localaccountsigup			X	
MFA page, api.phonefactor			X	
Self-asserted social account sign-up, api.selfasserted			X	
Self-asserted profile update, api.selfasserted.profileupdate			X	

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Unified signup or sign-in page, api.signuporsignin, with parameter "disableSignup"			X	
JavaScript / Page layout		X		

App-IEF integration

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Query string parameter domain_hint			X	Available as claim, can be passed to IDP.
Query string parameter login_hint			X	Available as claim, can be passed to IDP.
Insert JSON into UserJourney via client_assertion	X			Will be deprecated.
Insert JSON into UserJourney as id_token_hint		X		Go-forward approach to pass JSON.
Pass IDP TOKEN to the application		X		For example, from Facebook to app.

Session Management

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
SSO Session Provider			X	
External Login Session Provider			X	
SAML SSO Session Provider			X	
Default SSO Session Provider			X	

Security

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Policy Keys- Generate, Manual, Upload			X	

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Policy Keys- RSA/Cert, Secrets			X	
Policy upload			X	

Developer interface

FEATURE	DEVELOPMENT	PREVIEW	GA	NOTES
Azure Portal-IEF UX			X	
Application Insights UserJourney Logs		X		Used for troubleshooting during development.
Application Insights Event Logs (via orchestration steps)		X		Used to monitor user flows in production.

Next steps

Learn more about [custom policies](#) and the differences with user flows.

Page layout versions

2/26/2020 • 2 minutes to read • [Edit Online](#)

Page layout packages are periodically updated to include fixes and improvements in their page elements. The following change log specifies the changes introduced in each version.

NOTE

This feature is in public preview.

2.0.0

- Self-asserted page (`selfasserted`)
 - Added support for [display controls](#) in custom policies.

1.2.0

- All pages
 - Accessibility fixes
 - You can now add the `data-preload="true"` attribute [in your HTML tags](#) to control the load order for CSS and JavaScript.
 - Load linked CSS files at the same time as your HTML template so it doesn't 'flicker' between loading the files.
 - Control the order in which your `script` tags are fetched and executed before the page load.
 - Email field is now `type=email` and mobile keyboards will provide the correct suggestions
 - Support for Chrome translate
- Unified and self-asserted pages
 - The username/email and password fields now use the `form` HTML element to allow Edge and Internet Explorer (IE) to properly save this information.

1.1.0

- Exception page (`globalexception`)
 - Accessibility fix
 - Removed the default message when there is no contact from the policy
 - Default CSS removed
- MFA page (`multifactor`)
 - 'Confirm Code' button removed
 - The input field for the code now only takes input up to six (6) characters
 - The page will automatically attempt to verify the code entered when a 6-digit code is entered, without any button having to be clicked
 - If the code is wrong, the input field is automatically cleared
 - After three (3) attempts with an incorrect code, B2C sends an error back to the relying party
 - Accessibility fixes
 - Default CSS removed
- Self-asserted page (`selfasserted`)

- Removed cancel alert
- CSS class for error elements
- Show/hide error logic improved
- Default CSS removed
- Unified SSP (unifiedssp)
 - Added keep me signed in (KMSI) control

1.0.0

- Initial release

Next steps

For details on how to customize the user interface of your applications in custom policies, see [Customize the user interface of your application using a custom policy](#).

Cookies definitions for Azure AD B2C

1/30/2020 • 2 minutes to read • [Edit Online](#)

The following sections provide information about the cookies used in Azure Active Directory B2C (Azure AD B2C).

SameSite

The Microsoft Azure AD B2C service is compatible with SameSite browser configurations, including support for `SameSite=None` with the `Secure` attribute.

To safeguard access to sites, web browsers will introduce a new secure-by-default model that assumes all cookies should be protected from external access unless otherwise specified. The Chrome browser is the first to implement this change, starting with [Chrome 80 in February 2020](#). For more information about preparing for the change in Chrome, see [Developers: Get Ready for New SameSite=None; Secure Cookie Settings](#) on the Chromium Blog.

Developers must use the new cookie setting, `SameSite=None`, to designate cookies for cross-site access. When the `SameSite=None` attribute is present, an additional `Secure` attribute must be used so cross-site cookies can only be accessed over HTTPS connections. Validate and test all your applications, including those applications that use Azure AD B2C.

For more information, see:

- [Handle SameSite cookie changes in Chrome browser](#)
- [Effect on customer websites and Microsoft services and products in Chrome version 80 or later](#)

Cookies

The following table lists the cookies used in Azure AD B2C.

NAME	DOMAIN	EXPIRATION	PURPOSE
<code>x-ms-cpim-admin</code>	main.b2cadmin.ext.azure.com	End of browser session	Holds user membership data across tenants. The tenants a user is a member of and level of membership (Admin or User).
<code>x-ms-cpim-slice</code>	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used to route requests to the appropriate production instance.
<code>x-ms-cpim-trans</code>	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used for tracking the transactions (number of authentication requests to Azure AD B2C) and the current transaction.
<code>x-ms-cpim-sso:{Id}</code>	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used for maintaining the SSO session.

NAME	DOMAIN	EXPIRATION	PURPOSE
x-ms-cpim-cache:{id}_n	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session , successful authentication	Used for maintaining the request state.
x-ms-cpim-csrf	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Cross-Site Request Forgery token used for CSRF protection.
x-ms-cpim-dc	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used for Azure AD B2C network routing.
x-ms-cpim-ctx	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Context
x-ms-cpim-rp	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used for storing membership data for the resource provider tenant.
x-ms-cpim-rc	b2clogin.com, login.microsoftonline.com, branded domain	End of browser session	Used for storing the relay cookie.

Error codes: Azure Active Directory B2C

1/8/2020 • 9 minutes to read • [Edit Online](#)

The following errors can be returned by the Azure Active Directory B2C service.

ERROR CODE	MESSAGE
AADB2C90002	The CORS resource '{0}' returned a 404 not found.
AADB2C90006	The redirect URI '{0}' provided in the request is not registered for the client id '{1}'.
AADB2C90007	The application associated with client id '{0}' has no registered redirect URIs.
AADB2C90008	The request does not contain a client id parameter.
AADB2C90010	The request does not contain a scope parameter.
AADB2C90011	The client id '{0}' provided in the request does not match client id '{1}' registered in policy.
AADB2C90012	The scope '{0}' provided in request is not supported.
AADB2C90013	The requested response type '{0}' provided in the request is not supported.
AADB2C90014	The requested response mode '{0}' provided in the request is not supported.
AADB2C90016	The requested client assertion type '{0}' does not match the expected type '{1}'.
AADB2C90017	The client assertion provided in the request is invalid: {0}
AADB2C90018	The client id '{0}' specified in the request is not registered in tenant '{1}'.
AADB2C90019	The key container with id '{0}' in tenant '{1}' does not have a valid key. Reason: {2}.
AADB2C90021	The technical profile '{0}' does not exist in the policy '{1}' of tenant '{2}'.
AADB2C90022	Unable to return metadata for the policy '{0}' in tenant '{1}'.
AADB2C90023	Profile '{0}' does not contain the required metadata key '{1}'.
AADB2C90025	Profile '{0}' in policy '{1}' in tenant '{2}' does not contain the required cryptographic key '{3}'.

ERROR CODE	MESSAGE
AADB2C90027	Basic credentials specified for '{0}' are invalid. Check that the credentials are correct and that access has been granted by the resource.
AADB2C90028	Client certificate specified for '{0}' is invalid. Check that the certificate is correct, contains a private key and that access has been granted by the resource.
AADB2C90031	Policy '{0}' does not specify a default user journey. Ensure that the policy or its parents specify a default user journey as part of a relying party section.
AADB2C90035	The service is temporarily unavailable. Please retry after a few minutes.
AADB2C90036	The request does not contain a URI to redirect the user to post logout. Specify a URI in the post_logout_redirect_uri parameter field.
AADB2C90037	An error occurred while processing the request. Please contact administrator of the site you are trying to access.
AADB2C90039	The request contains a client assertion, but the provided policy '{0}' in tenant '{1}' is missing a client_secret in RelyingPartyPolicy.
AADB2C90040	User journey '{0}' does not contain a send claims step.
AADB2C90043	The prompt included in the request contains invalid values. Expected 'none', 'login', 'consent' or 'select_account'.
AADB2C90044	The claim '{0}' is not supported by the claim resolver '{1}'.
AADB2C90046	We are having trouble loading your current state. You might want to try starting your session over from the beginning.
AADB2C90047	The resource '{0}' contains script errors preventing it from being loaded.
AADB2C90048	An unhandled exception has occurred on the server.
AADB2C90051	No suitable claims providers were found.
AADB2C90052	Invalid username or password.
AADB2C90053	A user with the specified credential could not be found.
AADB2C90054	Invalid username or password.
AADB2C90055	The scope '{0}' provided in request must specify a resource, such as 'https://example.com/calendar.read'.

ERROR CODE	MESSAGE
AADB2C90057	The provided application is not configured to allow the OAuth Implicit flow.
AADB2C90058	The provided application is not configured to allow public clients.
AADB2C90067	The post logout redirect URI '{0}' has an invalid format. Specify an https based URL such as 'https://example.com/return' or for native clients use the IETF native client URI 'urn:ietf:wg:oauth:2.0:oob'.
AADB2C90068	The provided application with ID '{0}' is not valid against this service. Please use an application created via the B2C portal and try again.
AADB2C90075	The claims exchange '{0}' specified in step '{1}' returned HTTP error response with Code '{2}' and Reason '{3}'.
AADB2C90077	User does not have an existing session and request prompt parameter has a value of '{0}'.
AADB2C90079	Clients must send a client_secret when redeeming a confidential grant.
AADB2C90080	The provided grant has expired. Please re-authenticate and try again. Current time: {0}, Grant issued time: {1}, Grant sliding window expiration time: {2}.
AADB2C90081	The specified client_secret does not match the expected value for this client. Please correct the client_secret and try again.
AADB2C90083	The request is missing required parameter: {0}.
AADB2C90084	Public clients should not send a client_secret when redeeming a publicly acquired grant.
AADB2C90085	The service has encountered an internal error. Please reauthenticate and try again.
AADB2C90086	The supplied grant_type [{0}] is not supported.
AADB2C90087	The provided grant has not been issued for this version of the protocol endpoint.
AADB2C90088	The provided grant has not been issued for this endpoint. Actual Value : {0} and Expected Value : {1}
AADB2C90092	The provided application with ID '{0}' is disabled for the tenant '{1}'. Please enable the application and try again.
AADB2C90107	The application with ID '{0}' cannot get an ID token either because the openid scope was not provided in the request or the application is not authorized for it.

ERROR CODE	MESSAGE
AADB2C90108	The orchestration step '{0}' does not specify a CpiimIssuerTechnicalProfileReferenceId when one was expected.
AADB2C90110	The scope parameter must include 'openid' when requesting a response_type that includes 'id_token'.
AADB2C90111	Your account has been locked. Contact your support person to unlock it, then try again.
AADB2C90114	Your account is temporarily locked to prevent unauthorized use. Try again later.
AADB2C90115	When requesting the 'code' response_type, the scope parameter must include a resource or client ID for access tokens, and 'openid' for ID tokens. Additionally include 'offline_access' for refresh tokens.
AADB2C90117	The scope '{0}' provided in the request is not supported.
AADB2C90118	The user has forgotten their password.
AADB2C90120	The max age parameter '{0}' specified in the request is invalid. Max age must be an integer between '{1}' and '{2}' inclusive.
AADB2C90122	Input for '{0}' received in the request has failed HTTP request validation. Ensure that the input does not contain characters such as < or &.
AADB2C90128	The account associated with this grant no longer exists. Please reauthenticate and try again.
AADB2C90129	The provided grant has been revoked. Please reauthenticate and try again.
AADB2C90145	No unverified phone numbers have been found and policy does not allow a user entered number.
AADB2C90146	The scope '{0}' provided in request specifies more than one resource for an access token, which is not supported.
AADB2C90149	Script '{0}' failed to load.
AADB2C90151	User has exceeded the maximum number for retries for multi-factor authentication.
AADB2C90152	A multi-factor poll request failed to get a response from the service.
AADB2C90154	A multi-factor verification request failed to get a session id from the service.
AADB2C90155	A multi-factor verification request has failed with reason '{0}'.

ERROR CODE	MESSAGE
AADB2C90156	A multi-factor validation request has failed with reason '{0}'.
AADB2C90157	User has exceeded the maximum number for retries for a self-asserted step.
AADB2C90158	A self-asserted validation request has failed with reason '{0}'.
AADB2C90159	A self-asserted verification request has failed with reason '{0}'.
AADB2C90161	A self-asserted send response has failed with reason '{0}'.
AADB2C90165	The SAML initiating message with id '{0}' cannot be found in state.
AADB2C90168	The HTTP-Redirect request does not contain the required parameter '{0}' for a signed request.
AADB2C90178	The signing certificate '{0}' has no private key.
AADB2C90182	The supplied code_verifier does not match associated code_challenge
AADB2C90183	The supplied code_verifier is invalid
AADB2C90184	The supplied code_challenge_method is not supported. Supported values are plain or S256
AADB2C90188	The SAML technical profile '{0}' specifies a PartnerEntity URL of '{1}', but fetching the metadata fails with reason '{2}'.
AADB2C90194	Claim '{0}' specified for the bearer token is not present in the available claims. Available claims '{1}'.
AADB2C90205	This application does not have sufficient permissions against this web resource to perform the operation.
AADB2C90206	A time out has occurred initialization the client.
AADB2C90208	The provided id_token_hint parameter is expired. Please provide another token and try again.
AADB2C90209	The provided id_token_hint parameter does not contain an accepted audience. Valid audience values: '{0}'. Please provide another token and try again.
AADB2C90210	The provided id_token_hint parameter could not be validated. Please provide another token and try again.
AADB2C90211	The request contained an incomplete state cookie.
AADB2C90212	The request contained an invalid state cookie.

ERROR CODE	MESSAGE
AADB2C90220	The key container in tenant '{0}' with storage identifier '{1}' exists but does not contain a valid certificate. The certificate might be expired or your certificate might become active in the future (nbf).
AADB2C90223	An error has occurred sanitizing the CORS resource.
AADB2C90224	Resource owner flow has not been enabled for the application.
AADB2C90225	The username or password provided in the request are invalid.
AADB2C90226	The specified token exchange is only supported over HTTP POST.
AADB2C90232	The provided id_token_hint parameter does not contain an accepted issuer. Valid issuers: '{0}'. Please provide another token and try again.
AADB2C90233	The provided id_token_hint parameter failed signature validation. Please provide another token and try again.
AADB2C90235	The provided id_token is expired. Please provide another token and try again.
AADB2C90237	The provided id_token does not contain a valid audience. Valid audience values: '{0}'. Please provide another token and try again.
AADB2C90238	The provided id_token does not contain a valid issuer. Valid issuer values: '{0}'. Please provide another token and try again.
AADB2C90239	The provided id_token failed signature validation. Please provide another token and try again.
AADB2C90240	The provided id_token is malformed and could not be parsed. Please provide another token and try again.
AADB2C90242	The SAML technical profile '{0}' specifies PartnerEntity CDATA which cannot be loaded for reason '{1}'.
AADB2C90243	The IDP's client key/secret is not properly configured.
AADB2C90244	There are too many requests at this moment. Please wait for some time and try again.
AADB2C90248	Resource owner flow can only be used by applications created through the B2C admin portal.
AADB2C90250	The generic login endpoint is not supported.
AADB2C90255	The claims exchange specified in technical profile '{0}' did not complete as expected. You might want to try starting your session over from the beginning.

ERROR CODE	MESSAGE
AADB2C90261	The claims exchange '{0}' specified in step '{1}' returned HTTP error response that could not be parsed.
AADB2C90272	The id_token_hint parameter has not been specified in the request. Please provide token and try again.
AADB2C90273	An invalid response was received : '{0}'
AADB2C90274	The provider metadata does not specify a single logout service or the endpoint binding is not one of 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect' or 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST'.
AADB2C90276	The request is not consistent with the control setting '{0}': '{1}' in technicalProfile '{2}' for policy '{3}' tenant '{4}'.
AADB2C90277	The orchestration step '{0}' of user journey '{1}' of policy '{2}' does not contain a content definition reference.
AADB2C90279	The provided client id '{0}' does not match the client id that issued the grant.
AADB2C90284	The application with identifier '{0}' has not been granted consent and is unable to be used for local accounts.
AADB2C90285	The application with identifier '{0}' was not found.
AADB2C90288	UserJourney with id '{0}' referenced in TechnicalProfile '{1}' for refresh token redemption for tenant '{2}' does not exist in policy '{3}' or any of its base policies.
AADB2C90289	We encountered an error connecting to the identity provider. Please try again later.
AADB2C90296	Application has not been configured correctly. Please contact administrator of the site you are trying to access.
AADB2C99005	The request contains an invalid scope parameter which includes an illegal character '{0}'.
AADB2C99006	Azure AD B2C cannot find the extensions app with app id '{0}'. Please visit https://go.microsoft.com/fwlink/?linkid=851224 for more information.
AADB2C99011	The metadata value '{0}' has not been specified in TechnicalProfile '{1}' in policy '{2}'.
AADB2C99013	The supplied grant_type [{0}] and token_type [{1}] combination is not supported.
AADB2C99015	Profile '{0}' in policy '{1}' in tenant '{2}' is missing all InputClaims required for resource owner password credential flow.

Microsoft Graph operations available for Azure AD B2C

2/21/2020 • 2 minutes to read • [Edit Online](#)

The following Microsoft Graph API operations are supported for the management of Azure AD B2C resources, including users, identity providers, user flows, custom policies, and policy keys.

Each link in the following sections targets the corresponding page within the Microsoft Graph API reference for that operation.

User management

- [List users](#)
- [Create a consumer user](#)
- [Get a user](#)
- [Update a user](#)
- [Delete a user](#)

For more information about managing Azure AD B2C user accounts with the Microsoft Graph API, see [Manage Azure AD B2C user accounts with Microsoft Graph](#).

Identity providers (user flow)

Manage the identity providers available to your user flows in your Azure AD B2C tenant.

- [List identity providers registered in the Azure AD B2C tenant](#)
- [Create an identity provider](#)
- [Get an identity provider](#)
- [Update identity provider](#)
- [Delete an identity provider](#)

User flow

Configure pre-built policies for sign-up, sign-in, combined sign-up and sign-in, password reset, and profile update.

- [List user flows](#)
- [Create a user flow](#)
- [Get a user flow](#)
- [Delete a user flow](#)

Custom policies

The following operations allow you to manage your Azure AD B2C Trust Framework policies, known as [custom policies](#).

- [List all trust framework policies configured in a tenant](#)
- [Create trust framework policy](#)
- [Read properties of an existing trust framework policy](#)
- [Update or create trust framework policy](#)

- [Delete an existing trust framework policy](#)

Policy keys

The Identity Experience Framework stores the secrets referenced in a custom policy to establish trust between components. These secrets can be symmetric or asymmetric keys/values. In the Azure portal, these entities are shown as **Policy keys**.

The top-level resource for policy keys in the Microsoft Graph API is the [Trusted Framework Keyset](#). Each **Keyset** contains at least one **Key**. To create a key, first create an empty keyset, and then generate a key in the keyset. You can create a manual secret, upload a certificate, or a PKCS12 key. The key can be a generated secret, a string you define (such as the Facebook application secret), or a certificate you upload. If a keyset has multiple keys, only one of the keys is active.

Trust Framework policy keyset

- [List the trust framework keysets](#)
- [Create a trust framework keysets](#)
- [Get a keyset](#)
- [Update a trust framework keysets](#)
- [Delete a trust framework keysets](#)

Trust Framework policy key

- [Get currently active key in the keyset](#)
- [Generate a key in keyset](#)
- [Upload a string based secret](#)
- [Upload a X.509 certificate](#)
- [Upload a PKCS12 format certificate](#)

Applications

- [List applications](#)
- [Create an application](#)
- [Update application](#)
- [Create servicePrincipal](#)
- [Create oauth2Permission Grant](#)
- [Delete application](#)

Application extension properties

- [List extension properties](#)

Azure AD B2C provides a directory that can hold 100 custom attributes per user. For user flows, these extension properties are [managed by using the Azure portal](#). For custom policies, Azure AD B2C creates the property for you the first time the policy writes a value to the extension property.

Audit logs

- [List audit logs](#)

For more information about accessing Azure AD B2C audit logs with the Microsoft Graph API, see [Accessing Azure AD B2C audit logs](#).

Azure Active Directory B2C: Region availability & data residency

1/28/2020 • 2 minutes to read • [Edit Online](#)

Region availability and data residency are two very different concepts that apply differently to Azure AD B2C from the rest of Azure. This article explains the differences between these two concepts, and compares how they apply to Azure versus Azure AD B2C.

Azure AD B2C is **generally available worldwide** with the option for **data residency** in the **United States, Europe, or Asia Pacific**.

Region availability refers to where a service is available for use.

Data residency refers to where user data is stored.

Region availability

Azure AD B2C is available worldwide via the Azure public cloud.

This differs from the model followed by most other Azure services, which typically couple *availability* with *data residency*. You can see examples of this in both Azure's [Products Available By Region](#) page and the [Active Directory B2C pricing calculator](#).

Data residency

Azure AD B2C stores user data in either United States, Europe, or the Asia Pacific region.

Data residency is determined by the country/region you select when you [create an Azure AD B2C tenant](#):

The screenshot shows the 'Azure AD B2C Create Tenant' step of the Azure AD B2C Create Tenant wizard. At the top, there is a breadcrumb navigation: Home > New > Azure Active Directory B2C > Create new B2C Tenant or Link to existing Tenant > Azure AD B2C Create Tenant. Below the breadcrumb, there are two required fields: 'Organization name' and 'Initial domain name'. The 'Country or region' dropdown menu is highlighted with a red border. The dropdown lists several options: France, Faroe Islands, Finland, France, Gabon, Gambia, The, and Germany. France is currently selected. To the right of the dropdown, the suffix '.onmicrosoft.com' is visible.

Data resides in the **United States** for the following countries/regions:

United States, Canada, Costa Rica, Dominican Republic, El Salvador, Guatemala, Mexico, Panama, Puerto Rico and Trinidad & Tobago

Data resides in **Europe** for the following countries/regions:

Algeria, Austria, Azerbaijan, Bahrain, Belarus, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Egypt, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Israel, Italy, Jordan, Kazakhstan, Kenya, Kuwait, Latvia, Lebanon, Liechtenstein, Lithuania, Luxembourg, North Macedonia, Malta, Montenegro, Morocco, Netherlands, Nigeria, Norway, Oman, Pakistan, Poland, Portugal, Qatar, Romania, Russia, Saudi Arabia, Serbia, Slovakia, Slovenia, South Africa, Spain, Sweden, Switzerland, Tunisia, Turkey, Ukraine, United Arab Emirates and United Kingdom.

Data resides in **Asia Pacific** for the following countries/regions:

Afghanistan, Hong Kong SAR, India, Indonesia, Japan, Korea, Malaysia, Philippines, Singapore, Sri Lanka, Taiwan, and Thailand.

The following countries/regions are in the process of being added to the list. For now, you can still use Azure AD B2C by picking any of the countries/regions above.

Argentina, Australia, Brazil, Chile, Colombia, Ecuador, Iraq, New Zealand, Paraguay, Peru, Uruguay, and Venezuela.

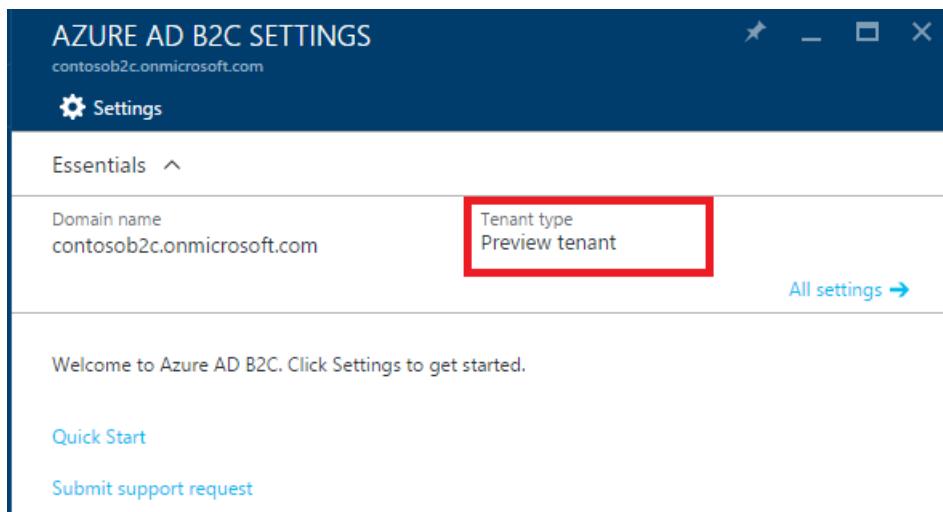
Preview tenant

If you had created a B2C tenant during Azure AD B2C's preview period, it's likely that your **Tenant type** says **Preview tenant**.

If this is the case, you must use your tenant ONLY for development and testing purposes. DO NOT use a preview tenant for production applications.

There is no migration path from a preview B2C tenant to a production-scale B2C tenant. You must create a new B2C tenant for your production applications.

There are known issues when you delete a preview B2C tenant and create a production-scale B2C tenant with the same domain name. *You must create a production-scale B2C tenant with a different domain name.*



Billing model for Azure Active Directory B2C

2/23/2020 • 6 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) usage is billed to a linked Azure subscription and uses a monthly active users (MAU) billing model. Learn how to link an Azure AD B2C resource to a subscription and how the MAU billing model works in the following sections.

IMPORTANT

This article does not contain pricing information. For the latest information about usage billing and pricing, see [Azure Active Directory B2C pricing](#).

Monthly active users (MAU) billing

Azure AD B2C billing is metered on the count of unique users with authentication activity within a calendar month, known as monthly active users (MAU) billing.

Starting **01 November 2019**, all newly created Azure AD B2C tenants are billed per-monthly active users (MAU). Existing tenants that are [linked to a subscription](#) on or after 01 November 2019 will be billed per-monthly active users (MAU).

If you have an existing Azure AD B2C tenant that was linked to a subscription prior to 01 November 2019, you can choose to do one of the following:

- Upgrade to the monthly active users (MAU) billing model, or
- Stay on the per-authentication billing model

Upgrade to monthly active users billing model

Azure subscription owners that have administrative access to the Azure AD B2C resource can switch to the MAU billing model. Billing options are configured in your Azure AD B2C resource.

The switch to monthly active users (MAU) billing is **irreversible**. Once you convert an Azure AD B2C resource to the MAU-based billing model, you cannot revert that resource to the per-authentication billing model.

Here's how to make the switch to MAU billing for an existing Azure AD B2C resource:

1. Sign in to the [Azure portal](#) as the subscription owner.
2. Select the **Directory + subscription** filter in the top menu, and then select the Azure AD B2C directory that you want to upgrade to MAU billing.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation menu with various options like 'Create a resource', 'Home', 'Dashboard', 'All services', etc. The 'Azure services' section in the center has icons for 'Create a resource', 'Azure Active Directory', 'Azure AD B2C', 'Resource groups', 'App Services', 'Storage accounts', 'SQL databases', and 'All services'. Below this is a 'Recent resources' table:

NAME	TYPE
contosob2c.onmicrosoft.com	B2C Tenant
contosob2cRG	Resource group
contosostorage	Storage account
vm01	Virtual machine
vm01PublicIP	Public IP address
vm01VMNic	Network interface
contoso-vnet	Virtual network
vm01NSG	Network security group
contosoRG	Resource group

To the right, the 'Directory + subscription' blade is open. It shows a 'Default subscription filter' section with a note about showing data for selected subscriptions. It also includes a 'Switch directory' section with a dropdown for 'Sign in to your last visited directory' and tabs for 'Favorites' and 'All Directories'. A search bar is at the bottom. A specific entry, 'Contoso B2C contosob2c.onmicrosoft.com 11111111-1111-1111-1111-111111111111', is highlighted with a red box.

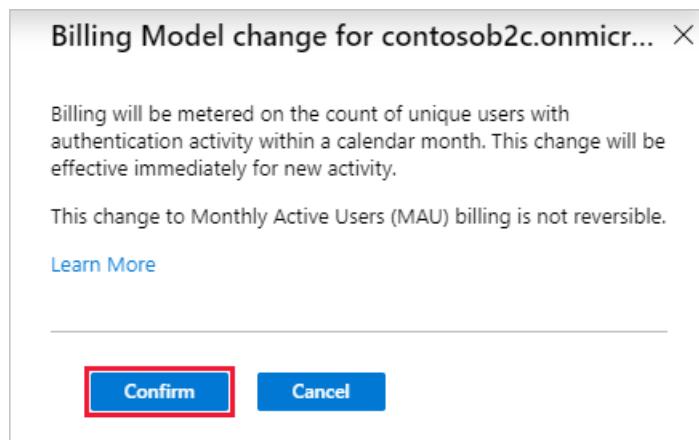
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. On the **Overview** page of the Azure AD B2C tenant, select the link under **Resource name**. You're directed to the Azure AD B2C resource in your Azure AD tenant.

The screenshot shows the 'Azure AD B2C' overview page for the tenant 'contosob2c.onmicrosoft.com'. The left sidebar has links for 'Overview', 'Manage', 'Applications', 'Identity providers', and 'User attributes'. The main area displays tenant details: Domain name (contosob2c.onmicrosoft.com), Tenant type (Production-scale tenant), Subscription status (Registered), and Resource name (contosob2c.onmicrosoft.com, highlighted with a red box). A purple banner at the top right says 'New! Pay by monthly active users plan available →'.

5. On the **Overview** page of the Azure AD B2C resource, under **Billable Units**, select the **Per Authentication (Change to MAU)** link.

The screenshot shows the 'contosob2c.onmicrosoft.com' overview page. The left sidebar has links for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. The main area shows the Azure AD B2C Tenant details: Resource group (contosob2cRG), Location (unitedstates), Subscription name (Visual Studio Enterprise with MSDN), Subscription ID (44444444-4444-4444-4444-444444444444), Tenant ID (88888888-8888-8888-8888-888888888888), and Billable Units (Per Authentication (Change to MAU, highlighted with a red box)).

6. Select **Confirm** to complete the upgrade to MAU billing.



What to expect when you transition to MAU billing from per-authentication billing

MAU-based metering is enabled as soon as you, the subscription/resource owner, confirm the change. Your monthly bill will reflect the units of authentication billed until the change, and new units of MAU starting with the change.

Users are not double-counted during the transition month. Unique active users who authenticate prior to the change are charged a per-authentication rate in a calendar month. Those same users are not included in the MAU calculation for the remainder of the subscription's billing cycle. For example:

- The Contoso B2C tenant has 1,000 users. 250 users are active in any given month. The subscription administrator changes from per-authentication to monthly active users (MAU) on the 10th of the month.
- Billing for 1st-10th is billed using the per-authentication model.
 - If 100 users sign in during this period (1st-10th), those users are tagged as *paid for the month*.
- Billing from the 10th (the effective time of transition) is billed at the MAU rate.
 - If an additional 150 users sign in during this period (10th-30th), only the additional 150 are billed.
 - The continued activity of the first 100 users does not impact billing for the remainder of the calendar month.

During the billing period of the transition, the subscription owner will likely see entries for both methods (per-authentication and per-MAU) appear in their Azure subscription billing statement:

- An entry for the usage until the date/time of change that reflects per-authentication.
- An entry for the usage after the change that reflects monthly active users (MAU).

For the latest information about usage billing and pricing for Azure AD B2C, see [Azure Active Directory B2C pricing](#).

Link an Azure AD B2C tenant to a subscription

Usage charges for Azure Active Directory B2C (Azure AD B2C) are billed to an Azure subscription. When an Azure AD B2C tenant is created, the tenant administrator needs to explicitly link the Azure AD B2C tenant to an Azure subscription.

The subscription link is achieved by creating an Azure AD B2C *resource* within the target Azure subscription. Several Azure AD B2C resources can be created in a single Azure subscription, along with other Azure resources like virtual machines, Storage accounts, and Logic Apps. You can see all of the resources within a subscription by going to the Azure Active Directory (Azure AD) tenant that the subscription is associated with.

A subscription linked to an Azure AD B2C tenant can be used for the billing of Azure AD B2C usage or other Azure resources, including additional Azure AD B2C resources. It cannot be used to add other Azure license-based services or Office 365 licenses within the Azure AD B2C tenant.

Prerequisites

- Azure subscription
- Azure AD B2C tenant that you want to link to a subscription
 - You must be a tenant administrator
 - The tenant must not already be linked to a subscription

Create the link

1. Sign in to the [Azure portal](#).
2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains the Azure subscription you'd like to use (*not* the directory containing the Azure AD B2C tenant).
3. Select **Create a resource**, enter `Active Directory B2C` in the **Search the Marketplace** field, and then select **Azure Active Directory B2C**.
4. Select **Create**
5. Select **Link an existing Azure AD B2C Tenant to my Azure subscription**.
6. Select an **Azure AD B2C Tenant** from the dropdown. Only tenants for which you are a global administrator and that are not already linked to a subscription are shown. The **Azure AD B2C Resource name** field is populated with the domain name of the Azure AD B2C tenant you select.
7. Select an active Azure **Subscription** of which you are an administrator.
8. Under **Resource group**, select **Create new**, and then specify the **Resource group location**. The resource group settings here have no impact on your Azure AD B2C tenant location, performance, or billing status.
9. Select **Create**.

The screenshot shows the Azure portal interface for creating a new B2C tenant or linking an existing one. The top navigation bar includes Home, New, Marketplace, Azure Active Directory B2C, Create new B2C Tenant or Link to existing Tenant, and Azure AD B2C Resource. The main form has two sections: 'Create new B2C Tenant or Link to existing Tenant' and 'Azure AD B2C Resource'. In the 'Create new B2C Tenant or Link to existing Tenant' section, there is a 'Create a new Azure AD B2C Tenant' button and a 'Link an existing Azure AD B2C Tenant to my Azure subscription' button, which is highlighted with a red box. In the 'Azure AD B2C Resource' section, fields include 'Azure AD B2C Tenant' (contosob2c.onmicrosoft.com), 'Azure AD B2C Resource name' (contosob2.onmicrosoft.com), 'Subscription' (Visual Studio Enterprise with MSDN), 'Resource group' ((New) contosob2cRG), 'Resource group location' (East US 2), and a 'Create' button at the bottom right, which is also highlighted with a red box.

After you complete these steps for an Azure AD B2C tenant, your Azure subscription is billed in accordance with your Azure Direct or Enterprise Agreement details, if applicable.

Manage your Azure AD B2C tenant resources

After you create the Azure AD B2C resource in an Azure subscription, you should see a new resource of the type "B2C tenant" appear with your other Azure resources.

You can use this resource to:

- Navigate to the subscription to review billing information
- Get the Azure AD B2C tenant's tenant ID in GUID format
- Go to your Azure AD B2C tenant
- Submit a support request
- Move your Azure AD B2C tenant resource to another Azure subscription or resource group

Resource group (change)
[contosob2cRG](#)

Location
unitedstates

Subscription name (change)
[Visual Studio Enterprise with MSDN](#)

Subscription ID
00000000-0000-0000-0000-000000000000

Azure AD B2C Tenant
[contosob2c.onmicrosoft.com](#)

Tenant ID
11111111-1111-1111-1111-111111111111

Regional restrictions

If you've established regional restrictions for Azure resource creation in your subscription, that restriction may prevent you from creating the Azure AD B2C resource.

To mitigate this issue, relax your regional restrictions.

Azure Cloud Solution Providers (CSP) subscriptions

Azure Cloud Solution Providers (CSP) subscriptions are supported in Azure AD B2C. The functionality is available using APIs or the Azure portal for Azure AD B2C and for all Azure resources. CSP subscription administrators can link, move, and delete relationships with Azure AD B2C as done with other Azure resources.

The management of Azure AD B2C using role-based access control is not affected by the association between the Azure AD B2C tenant and an Azure CSP subscription. Role-based access control is achieved by using tenant-based roles, not subscription-based roles.

Change the Azure AD B2C tenant billing subscription

Azure AD B2C tenants can be moved to another subscription if the source and destination subscriptions exist within the same Azure Active Directory tenant.

To learn how to move Azure resources like your Azure AD B2C tenant to another subscription, see [Move resources to new resource group or subscription](#).

Before you initiate the move, be sure to read the entire article to fully understand the limitations and requirements for such a move. In addition to instructions for moving resources, it includes critical information like a pre-move checklist and how to validate the move operation.

Next steps

For the latest pricing information, see [Azure Active Directory B2C pricing](#).

Manage threats to resources and data in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) has built-in features that can help you protect against threats to your resources and data. These threats include denial-of-service attacks and password attacks. Denial-of-service attacks might make resources unavailable to intended users. Password attacks lead to unauthorized access to resources.

Denial-of-service attacks

Azure AD B2C defends against SYN flood attacks using a SYN cookie. Azure AD B2C also protects against denial-of-service attacks by using limits for rates and connections.

Password attacks

Passwords that are set by users are required to be reasonably complex. Azure AD B2C has mitigation techniques in place for password attacks. Mitigation includes detection of brute-force password attacks and dictionary password attacks. By using various signals, Azure AD B2C analyzes the integrity of requests. Azure AD B2C is designed to intelligently differentiate intended users from hackers and botnets.

Azure AD B2C uses a sophisticated strategy to lock accounts. The accounts are locked based on the IP of the request and the passwords entered. The duration of the lockout also increases based on the likelihood that it's an attack. After a password is tried 10 times unsuccessfully (the default attempt threshold), a one-minute lockout occurs. The next time a login is unsuccessful after the account is unlocked (that is, after the account has been automatically unlocked by the service once the lockout period expires), another one-minute lockout occurs and continues for each unsuccessful login. Entering the same password repeatedly doesn't count as multiple unsuccessful logins.

The first 10 lockout periods are one minute long. The next 10 lockout periods are slightly longer and increase in duration after every 10 lockout periods. The lockout counter resets to zero after a successful login when the account isn't locked. Lockout periods can last up to five hours.

Manage password protection settings

To manage password protection settings, including the lockout threshold:

1. Sign in to the [Azure portal](#)
2. Use the **Directory + subscription** filter in the top menu to select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Under **Security**, select **Authentication methods (Preview)**, then select **Password protection**.
5. Enter your desired password protection settings, then select **Save**.

Save **Discard**

Custom smart lockout

Lockout threshold ✓

Lockout duration in seconds ✓

Custom banned passwords

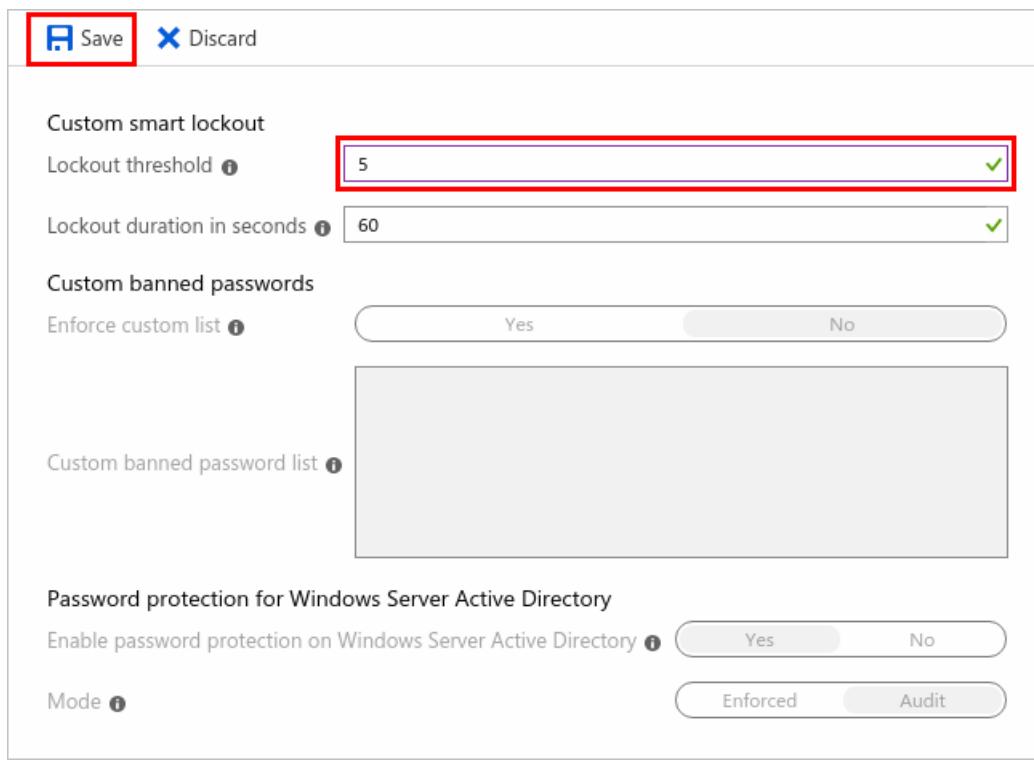
Enforce custom list Yes No

Custom banned password list

Password protection for Windows Server Active Directory

Enable password protection on Windows Server Active Directory Yes No

Mode Enforced Audit



Setting the lockout threshold to 5 in **Password protection** settings.

View locked-out accounts

To obtain information about locked-out accounts, you can check the Active Directory [sign-in activity report](#). Under **Status**, select **Failure**. Failed sign-in attempts with a **Sign-in error code** of **50053** indicate a locked account:

Status	Failure
Sign-in error code	50053
Failure reason	Account is locked because user tried to sign in too many times with an incorrect user ID or password.

To learn about viewing the sign-in activity report in Azure Active Directory, see [Sign-in activity report error codes](#).

Azure AD B2C: Extensions app

1/28/2020 • 2 minutes to read • [Edit Online](#)

When an Azure AD B2C directory is created, an app called

b2c-extensions-app. Do not modify. Used by AADB2C for storing user data. is automatically created inside the new directory. This app, referred to as the **b2c-extensions-app**, is visible in *App registrations*. It is used by the Azure AD B2C service to store information about users and custom attributes. If the app is deleted, Azure AD B2C will not function correctly and your production environment will be affected.

IMPORTANT

Do not delete the b2c-extensions-app unless you are planning to immediately delete your tenant. If the app remains deleted for more than 30 days, user information will be permanently lost.

Verifying that the extensions app is present

To verify that the b2c-extensions-app is present:

1. Inside your Azure AD B2C tenant, click on **All services** in the left-hand navigation menu.
2. Search for and open **App registrations**.
3. Look for an app that begins with **b2c-extensions-app**

Recover the extensions app

If you accidentally deleted the b2c-extensions-app, you have 30 days to recover it. You can restore the app using the Graph API:

1. Browse to <https://graphexplorer.azurewebsites.net/>.
2. Log in to the site as a global administrator for the Azure AD B2C directory that you want to restore the deleted app for. This global administrator must have an email address similar to the following:
username@{yourTenant}.onmicrosoft.com .
3. Issue an HTTP GET against the URL https://graph.windows.net/myorganization/deletedApplications with api-version=1.6. This operation will list all of the applications that have been deleted within the past 30 days.
4. Find the application in the list where the name begins with 'b2c-extension-app' and copy its objectid property value.
5. Issue an HTTP POST against the URL
<https://graph.windows.net/myorganization/deletedApplications/{OBJECTID}/restore> . Replace the {OBJECTID} portion of the URL with the objectid from the previous step.

You should now be able to [see the restored app](#) in the Azure portal.

NOTE

An application can only be restored if it has been deleted within the last 30 days. If it has been more than 30 days, data will be permanently lost. For more assistance, file a support ticket.

User flow versions in Azure Active Directory B2C

1/28/2020 • 2 minutes to read • [Edit Online](#)

User flows in Azure Active Directory B2C (Azure AD B2C) help you to set up common [policies](#) that fully describe customer identity experiences. These experiences include sign-up, sign-in, password reset, or profile editing. In Azure AD B2C, you can select from a collection of both recommended user flows and preview user flows.

New user flows are added as new versions. As user flows become stable, they'll be recommended for use. User flows are marked as **Recommended** if they've been thoroughly tested. User flows will be considered in preview until marked as recommended. Use a recommended user flow for any production application, but choose from other versions to test new functionality as it becomes available. You shouldn't use older versions of recommended user flows.

IMPORTANT

Unless a user flow is identified as **Recommended**, it is considered to be in *preview*. You should use only recommended user flows for your production applications.

V1

USER FLOW	RECOMMENDED	DESCRIPTION
Password reset	Yes	<p>Enables a user to choose a new password after verifying their email. Using this user flow, you can configure:</p> <ul style="list-style-type: none">• Multi-factor authentication• Token compatibility settings• Password complexity requirements
Profile editing	Yes	<p>Enables a user to configure their user attributes. Using this user flow, you can configure:</p> <ul style="list-style-type: none">• Token lifetime• Token compatibility settings• Session behavior
Sign in using ROPC	No	<p>Enables a user with a local account to sign in directly in native applications (no browser required). Using this user flow, you can configure:</p> <ul style="list-style-type: none">• Token lifetime• Token compatibility settings

USER FLOW	RECOMMENDED	DESCRIPTION
Sign in	No	<p>Enables a user to sign in to their account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token lifetime • Token compatibility settings • Session behavior • Block sign-in • Force password reset • Keep Me Signed In (KMSI) <p>You can't customize the user interface with this user flow.</p>
Sign up	No	<p>Enables a user to create an account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token lifetime • Token compatibility settings • Session behavior • Password complexity requirements
Sign up and sign in	Yes	<p>Enables a user to create an account or sign in to their account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token lifetime • Token compatibility settings • Session behavior • Password complexity requirements

V1.1

USER FLOW	RECOMMENDED	DESCRIPTION
Password reset v1.1	No	<p>Allows a user to choose a new password after verifying their email (new page layout available). Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token compatibility settings • Password complexity requirements

V2

USER FLOW	RECOMMENDED	DESCRIPTION
-----------	-------------	-------------

User flow	Recommended	Description
Password reset v2	No	<p>Enables a user to choose a new password after verifying their email. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token compatibility settings • Age gating • password complexity requirements
Profile editing v2	Yes	<p>Enables a user to configure their user attributes. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Token lifetime • Token compatibility settings • Session behavior
Sign in v2	No	<p>Enables a user to sign in to their account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token lifetime • Token compatibility settings • Session behavior • Age gating • Sign-in page customization
Sign up v2	No	<p>Enables a user to create an account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Token lifetime • Token compatibility settings • Session behavior • Age gating • Password complexity requirements
Sign up and sign in v2	No	<p>Enables a user to create an account or sign in their account. Using this user flow, you can configure:</p> <ul style="list-style-type: none"> • Multi-factor authentication • Age gating • Password complexity requirements

Azure AD B2C: Frequently asked questions (FAQ)

2/20/2020 • 9 minutes to read • [Edit Online](#)

This page answers frequently asked questions about the Azure Active Directory B2C (Azure AD B2C). Keep checking back for updates.

Why can't I access the Azure AD B2C extension in the Azure portal?

There are two common reasons for why the Azure AD extension is not working for you. Azure AD B2C requires your user role in the directory to be global administrator. Please contact your administrator if you think you should have access. If you have global administrator privileges, make sure that you are in an Azure AD B2C directory and not an Azure Active Directory directory. You can see instructions for [creating an Azure AD B2C tenant](#).

Can I use Azure AD B2C features in my existing, employee-based Azure AD tenant?

Azure AD and Azure AD B2C are separate product offerings and cannot coexist in the same tenant. An Azure AD tenant represents an organization. An Azure AD B2C tenant represents a collection of identities to be used with relying party applications. By adding **New OpenID Connect provider** under **Azure AD B2C > Identity providers** or with custom policies, Azure AD B2C can federate to Azure AD allowing authentication of employees in an organization.

Can I use Azure AD B2C to provide social login (Facebook and Google+) into Office 365?

Azure AD B2C can't be used to authenticate users for Microsoft Office 365. Azure AD is Microsoft's solution for managing employee access to SaaS apps and it has features designed for this purpose such as licensing and Conditional Access. Azure AD B2C provides an identity and access management platform for building web and mobile applications. When Azure AD B2C is configured to federate to an Azure AD tenant, the Azure AD tenant manages employee access to applications that rely on Azure AD B2C.

What are local accounts in Azure AD B2C? How are they different from work or school accounts in Azure AD?

In an Azure AD tenant, users that belong to the tenant sign-in with an email address of the form

`<xyz>@<tenant domain>`. The `<tenant domain>` is one of the verified domains in the tenant or the initial `<...>.onmicrosoft.com` domain. This type of account is a work or school account.

In an Azure AD B2C tenant, most apps want the user to sign-in with any arbitrary email address (for example, `joe@comcast.net`, `bob@gmail.com`, `sarah@contoso.com`, or `jim@live.com`). This type of account is a local account. We also support arbitrary user names as local accounts (for example, `joe`, `bob`, `sarah`, or `jim`). You can choose one of these two local account types when configuring identity providers for Azure AD B2C in the Azure portal. In your Azure AD B2C tenant, select **Identity providers**, select **Local account**, and then select **Username**.

User accounts for applications can be created through a sign-up user flow, sign-up or sign-in user flow, the Microsoft Graph API, or in the Azure portal.

Which social identity providers do you support now? Which ones do you plan to support in the future?

We currently support several social identity providers including Amazon, Facebook, GitHub (preview), Google, LinkedIn, Microsoft Account (MSA), QQ (preview), Twitter, WeChat (preview), and Weibo (preview). We evaluate adding support for other popular social identity providers based on customer demand.

Azure AD B2C also supports [custom policies](#). Custom policies allow you to create your own policy for any identity provider that supports [OpenID Connect](#) or SAML. Get started with custom policies by checking out our [custom policy starter pack](#).

Can I configure scopes to gather more information about consumers from various social identity providers?

No. The default scopes used for our supported set of social identity providers are:

- Facebook: email
- Google+: email
- Microsoft account: openid email profile
- Amazon: profile
- LinkedIn: r_emailaddress, r_basicprofile

Does my application have to be run on Azure for it work with Azure AD B2C?

No, you can host your application anywhere (in the cloud or on-premises). All it needs to interact with Azure AD B2C is the ability to send and receive HTTP requests on publicly accessible endpoints.

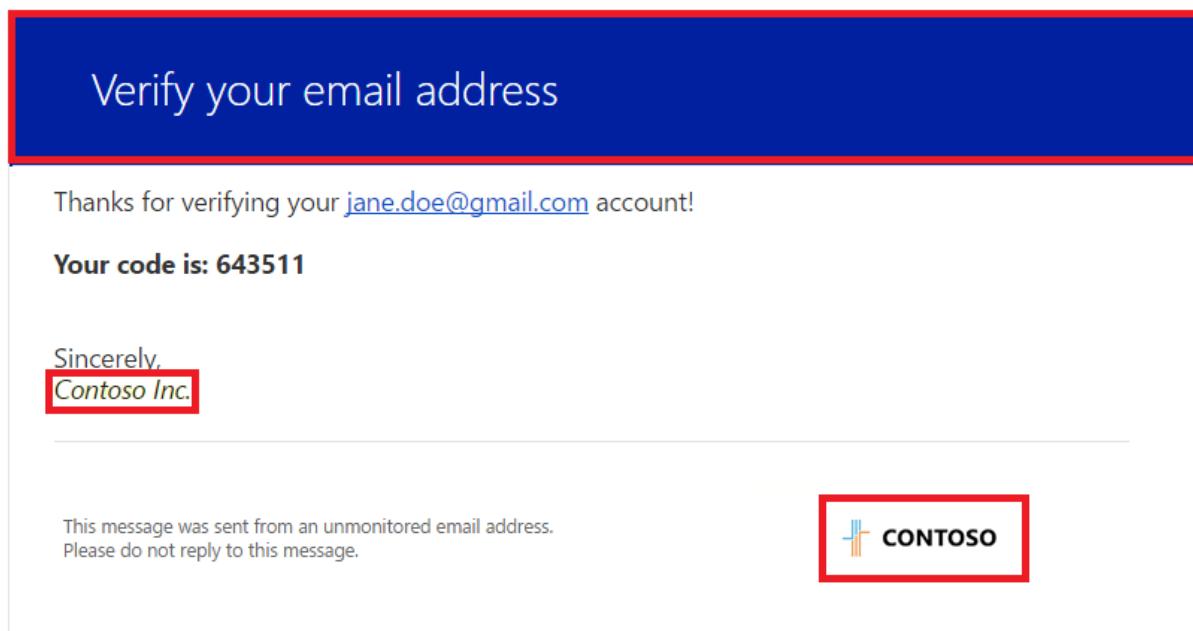
I have multiple Azure AD B2C tenants. How can I manage them on the Azure portal?

Before opening 'Azure AD B2C' in the left side menu of the Azure portal, you must switch into the directory you want to manage. Switch directories by clicking your identity in the upper right of the Azure portal, then choose a directory in the drop down that appears.

How do I customize verification emails (the content and the "From:" field) sent by Azure AD B2C?

You can use the [company branding feature](#) to customize the content of verification emails. Specifically, these two elements of the email can be customized:

- **Banner logo:** Shown at the bottom-right.
- **Background color:** Shown at the top.



The email signature contains the Azure AD B2C tenant's name that you provided when you first created the Azure AD B2C tenant. You can change the name using these instructions:

1. Sign in to the [Azure portal](#) as the Global Administrator.
2. Open the **Azure Active Directory** blade.
3. Click the **Properties** tab.
4. Change the **Name** field.
5. Click **Save** at the top of the page.

Currently there is no way to change the "From:" field on the email.

How can I migrate my existing user names, passwords, and profiles from my database to Azure AD B2C?

You can use the Microsoft Graph API to write your migration tool. See the [User migration guide](#) for details.

What password user flow is used for local accounts in Azure AD B2C?

The Azure AD B2C password user flow for local accounts is based on the policy for Azure AD. Azure AD B2C's sign-up, sign-up or sign-in and password reset user flows use the "strong" password strength and don't expire any passwords. For more details, see [Password policies and restrictions in Azure Active Directory](#).

For information about account lockouts and passwords, see [Manages threats to resources and data in Azure Active Directory B2C](#).

Can I use Azure AD Connect to migrate consumer identities that are stored on my on-premises Active Directory to Azure AD B2C?

No, Azure AD Connect is not designed to work with Azure AD B2C. Consider using the [Microsoft Graph API](#) for user migration. See the [User migration guide](#) for details.

Can my app open up Azure AD B2C pages within an iFrame?

No, for security reasons, Azure AD B2C pages cannot be opened within an iFrame. Our service communicates with the browser to prohibit iFrames. The security community in general and the OAuth2 specification, recommend against using iFrames for identity experiences due to the risk of click-jacking.

Does Azure AD B2C work with CRM systems such as Microsoft Dynamics?

Integration with Microsoft Dynamics 365 Portal is available. See [Configuring Dynamics 365 Portal to use Azure AD B2C for authentication](#).

Does Azure AD B2C work with SharePoint on-premises 2016 or earlier?

Azure AD B2C is not meant for the SharePoint external partner-sharing scenario; see [Azure AD B2B](#) instead.

Should I use Azure AD B2C or B2B to manage external identities?

Read [Compare B2B collaboration and B2C in Azure AD](#) to learn more about applying the appropriate features to your external identity scenarios.

What reporting and auditing features does Azure AD B2C provide? Are they the same as in Azure AD Premium?

No, Azure AD B2C does not support the same set of reports as Azure AD Premium. However there are many commonalities:

- **Sign-in reports** provide a record of each sign-in with reduced details.
- **Audit reports** include both admin activity as well as application activity.
- **Usage reports** include the number of users, number of logins, and volume of MFA.

Can I localize the UI of pages served by Azure AD B2C? What languages are supported?

Yes, see [language customization](#). We provide translations for 36 languages, and you can override any string to suit your needs.

Can I use my own URLs on my sign-up and sign-in pages that are served by Azure AD B2C? For instance, can I change the URL from contoso.b2clogin.com to login.contoso.com?

Not currently. This feature is on our roadmap. Verifying your domain in the **Domains** tab in the Azure portal does not accomplish this goal. However, with b2clogin.com, we offer a [neutral top level domain](#), and thus the external appearance can be implemented without the mention of Microsoft.

How do I delete my Azure AD B2C tenant?

Follow these steps to delete your Azure AD B2C tenant.

You can use the current **Applications** experience or our new unified **App registrations (Preview)** experience.

[Learn more about the new experience](#).

- [Applications](#)
- [App registrations \(Preview\)](#)

1. Sign in to the [Azure portal](#) as the *Subscription Administrator*. Use the same work or school account or the same

Microsoft account that you used to sign up for Azure.

2. Select the **Directory + subscription** filter in the top menu, and then select the directory that contains your Azure AD B2C tenant.
3. In the left menu, select **Azure AD B2C**. Or, select **All services** and search for and select **Azure AD B2C**.
4. Delete all the **User flows (policies)** in your Azure AD B2C tenant.
5. Delete all the **Applications** you registered in your Azure AD B2C tenant.
6. Select **Azure Active Directory** on the left-hand menu.
7. Under **Manage**, select **Users**.
8. Select each user in turn (exclude the *Subscription Administrator* user you are currently signed in as). Select **Delete** at the bottom of the page and select **YES** when prompted.
9. Under **Manage**, select **App registrations** (or **App registrations (Legacy)**).
10. Select **View all applications**
11. Select the application named **b2c-extensions-app**, select **Delete**, and then select **Yes** when prompted.
12. Under **Manage**, select **User settings**.
13. If present, under **LinkedIn account connections**, select **No**, then select **Save**.
14. Under **Manage**, select **Properties**
15. Under **Access management for Azure resources**, select **Yes**, and then select **Save**.
16. Sign out of the Azure portal and then sign back in to refresh your access.
17. Select **Azure Active Directory** on the left-hand menu.
18. On the **Overview** page, select **Delete directory**. Follow the on-screen instructions to complete the process.

Can I get Azure AD B2C as part of Enterprise Mobility Suite?

No, Azure AD B2C is a pay-as-you-go Azure service and is not part of Enterprise Mobility Suite.

How do I report issues with Azure AD B2C?

See [File support requests for Azure Active Directory B2C](#).

Solutions and Training for Azure Active Directory B2C

9/17/2019 • 2 minutes to read • [Edit Online](#)

Azure Active Directory B2C (Azure AD B2C) enables organizations to implement business solutions that help them connect with their customers. The following solution guides and training are downloadable documents that will walk you through these solutions.

TITLE	DESCRIPTION
Customer Identity Management with Azure AD B2C	In this overview of the service, Parakh Jain (@jainparakh) from the Azure AD B2C team provides us an overview of how the service works, and also show how we can quickly connect B2C to an ASP.NET Core application.
Benefits of using Azure AD B2C	Understand the benefits and common scenarios of Azure AD B2C, and how your application(s) can leverage this CIAM service.
Gaining Expertise in Azure AD B2C: A Course for Developers	This end-to-end course takes developers through a complete journey on developing applications with Azure AD B2C as the authentication mechanism. Ten in-depth modules with labs cover everything from setting up an Azure subscription to creating custom policies that define the journeys that engage your customers.
Enabling partners, Suppliers, and Customers to Access Applications with Azure active Directory	Every organization's success, regardless of its size, industry, or compliance and security posture, relies on organizational ability to collaborate with other organizations and connect with customers. Bringing together Azure AD, Azure AD B2C, and Azure AD B2B Collaboration, this guide details the business value and the mechanics of building an application or web experience that provides a consolidated authentication experience tailored to the contexts of your employees, business partners and suppliers, and customers.
Migrating Application Authentication to Azure AD B2C in a Hybrid Environment	In today's modern organizations, digital transformation and moving to the cloud happens in stages, requiring most organizations to at least temporarily operate in a hybrid identity environment. This guide focuses on creating the migration plan for moving your first application to Azure AD B2C, and covers the considerations for doing so while in a hybrid identity environment.
General Data protection Regulation (GDPR) Considerations for Customer Facing Applications	For any customer facing applications, GDPR must be taken into consideration by all organizations that embark on projects that hold personal data and serve EU citizens. This solution guide focuses on how Azure AD B2C can be used as a flexible component of your overall GDPR compliance approach, including how Azure AD B2C components support each of the key GDPR rights for individuals.

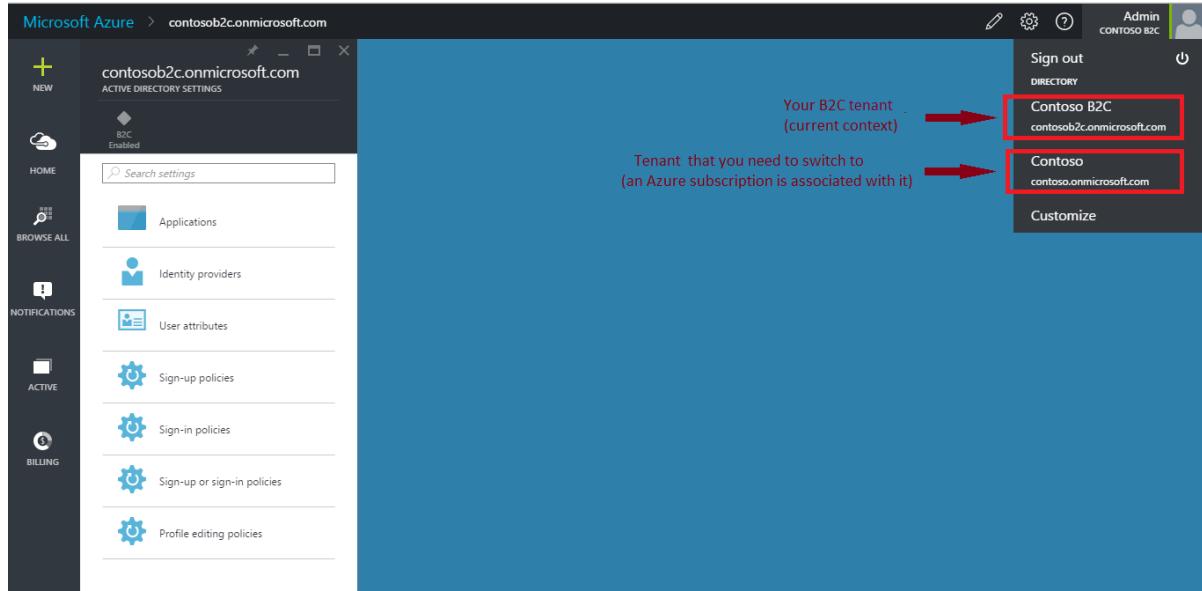
TITLE	DESCRIPTION
<p>Working with custom policies:</p> <ul style="list-style-type: none">• Custom policies introduction• Leverage custom policies in your tenant• Structure policies and manage keys• Bring your own identity and migrate users• Troubleshoot policies and audit access• Deep dive on custom policy schema	<p>This series of documents provides an end-to-end journey with the custom policies in Azure AD B2C, presenting in-depth the most common advanced identity scenarios.</p> <p>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.</p>

Azure Active Directory B2C: File Support Requests

1/28/2020 • 2 minutes to read • [Edit Online](#)

You can file support requests for Azure Active Directory B2C (Azure AD B2C) on the Azure portal using the following steps:

1. Switch from your B2C tenant to another tenant that has an Azure subscription associated with it. Typically, the latter is your employee tenant or the default tenant created for you when you signed up for an Azure subscription. To learn more, see [how an Azure subscription is related to Azure AD](#).



2. After switching tenants, click **Help + support**.

Microsoft Azure

Dashboard | Edit

Search resources

+ New

Resource groups

All resources

Recent

App Services

SQL databases

Virtual machines (classic)

Virtual machines

Cloud services (classic)

Subscriptions

Storage accounts (classic)

Storage accounts

DocumentDB Accounts

Browse >

The Microsoft Azure dashboard features a grid of tiles. The 'Help + support' tile is highlighted with a red border. Other visible tiles include 'Marketplace', 'Service health MY RESOURCES' (showing a world map with green checkmarks), 'Tour', 'How it works', 'Feedback', 'Azure classic portal', 'What's new', and 'Portal settings'.

3. Click **New support request**.

Help + support

Settings New support...

Support Add tiles +

New support request

Manage support requests

Resource health

Link Existing Benefits

MSDN forums

Stack Overflow

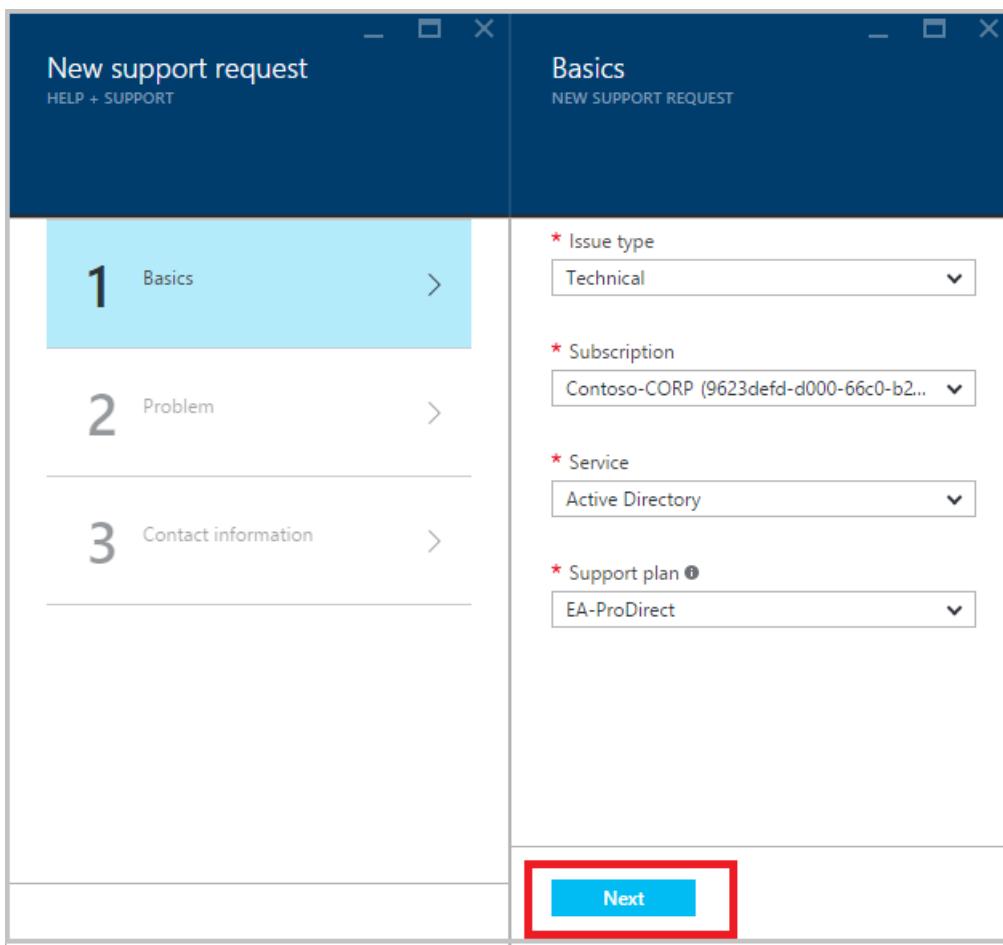
Service health MY RESOURCES

Feedback Diagnostics

The 'Help + support' blade contains several tiles: 'New support request' (highlighted with a red border), 'Manage support requests', 'Resource health', 'Link Existing Benefits', 'MSDN forums', 'Stack Overflow', 'Service health MY RESOURCES' (showing a world map with green checkmarks), 'Feedback', and 'Diagnostics'.

4. In the **Basics** blade, use these details and click **Next**.

- **Issue type** is **Technical**.
- Choose the appropriate **Subscription**.
- **Service** is **Active Directory**.
- Choose the appropriate **Support plan**. If you don't have one, you can sign up for one [here](#).



5. In the **Problem** blade, use these details and click **Next**.

- Choose the appropriate **Severity** level.
- **Problem type** is **B2C**.
- Choose the appropriate **Category**.
- Describe your issue in the **Details** field. Provide details such as the B2C tenant name, description of the problem, error messages, correlation IDs (if available), and so on.
- In the **Time frame** field, provide the date and time (including time zone) that the issue occurred.
- Under **File upload**, upload all screenshots and files that you think would assist in resolving the issue.

The screenshot shows the 'New support request' wizard in progress. The left pane displays a navigation path with three steps: 1. Basics (completed), 2. Problem (current step), and 3. Contact information. The right pane is titled 'Problem' and contains the following fields:

- * Severity: C - Minimal impact
- * Problem type: B2C
- * Category: Choose a category
- * Details: Enter the details of your issue

Below these fields is a note: 'If possible, please provide date, time, and time zone information for the most recent' with a checkmark. A red box highlights the 'Next' button at the bottom.

6. In the **Contact information** blade, add your contact information. Click **Create**.

The screenshot shows a two-panel interface for creating a new support request. The left panel is titled 'New support request' and 'HELP + SUPPORT'. It displays a progress bar with three steps: '1 Basics' (green checkmark), '2 Problem' (green checkmark), and '3 Contact information' (blue background, grey arrow). The right panel is titled 'Contact information' and 'NEW SUPPORT REQUEST'. It contains fields for 'First name' (text input), 'Last name' (text input), 'Email' (text input), 'Additional contact' (text input placeholder 'Who else should we email?'), 'Phone number' (text input), 'Country/region' (dropdown menu set to 'United States'), and a note about accepting terms and conditions. A red box highlights the 'Create' button at the bottom.

- After submitting your support request, you can monitor it by clicking **Help + support** on the Startboard, and then **Manage support requests**.

Known issue: Filing a support request in the context of a B2C tenant

If you missed step 2 outlined above and try to create a support request in the context of your B2C tenant, you will see the following error.

IMPORTANT

Don't attempt to sign up for a new Azure subscription in your B2C tenant.

