Contents

About the Mesh SDK

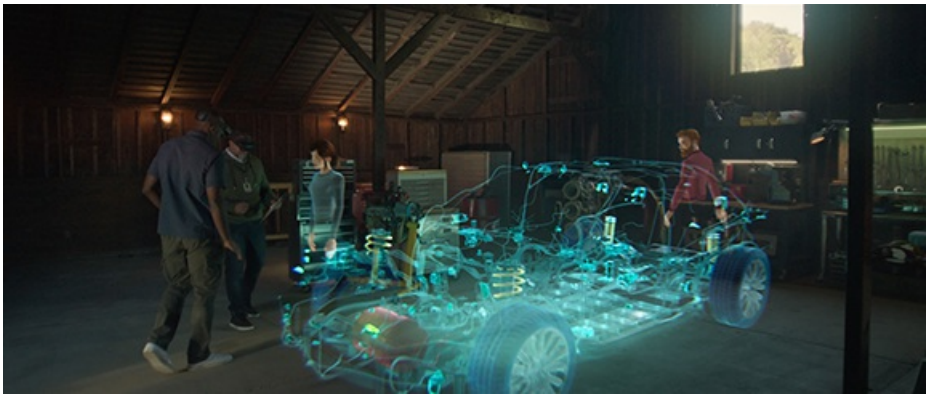
3/4/2021 • 4 minutes to read • Edit Online

IMPORTANT

The **Mesh SDK** is currently in private preview.

Mesh SDK Overview

The Mesh SDK provides services and client components that power social-first, rich, and immersive experiences on a trustworthy platform that fits effortlessly into the lives of your users. The goal is to increase the developer base and provide momentum for the cycle between applications, content, devices, and users. The platform welcomes HoloLens 2 and VR headset users and provides exceptional, tightly integrated support for these devices.



The SDK was created to provide opportunities for developers to lower the adoption bar for delightful and immersive 3D experiences. If you're in the business of creating experiences where remote and local collaborators can play, create, and collaborate, you're in the right place!

Who Mesh SDK is for

The SDK is for developers who build Mixed Reality applications that allow users (both Commercial and Consumers) to seamlessly be present, connect and collaborate across time & space.

Build fast with a rich set of client libraries and helpers.

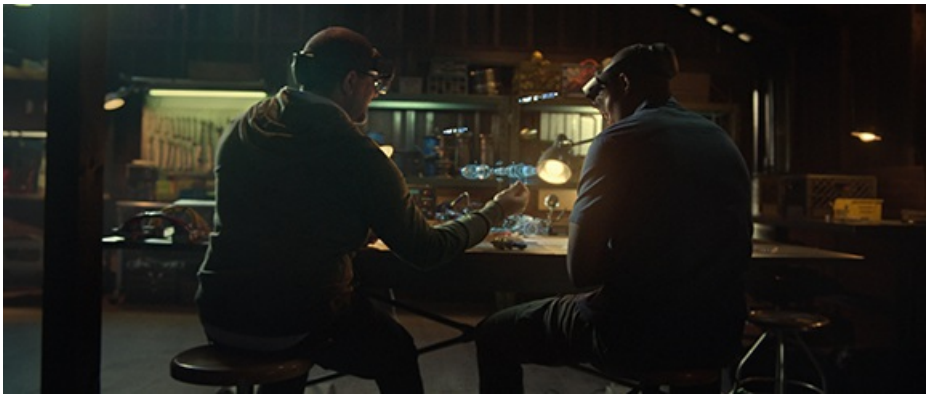
- The SDK leverages Microsoft Identity platform removing the burden of authentication challenges and privacy controls around this data.

- Build powerful apps using various SDK Helpers that handle Billing, Transport, Spatial and other common matters.
- Developer simplicity. No complex provisioning & configuration, no need to reason through Pricing tiers and resulting feature seams.

Why use Mesh SDK?

The SDK provides all the tools to build apps where people can interact with 3D content or engage with each other across any platform or device. Users feel like they are physically present with each other, even when they are not, so that they can:

- Collaborate, create, and communicate
- Help each other solve problems
- Train together from anywhere
- Design new things
- Share fun experiences



Here are some key features offered by the SDK:

- **Identity, Profile, & Privacy** - Identify users (federated or MSA/AAD) across devices, with privacy controls around this data.
- **Friends & Invitations** - Integrate users' social graph into mixed reality experiences, including availability to join experiences.
- **Avatars + Holoportation** - Utilize digital representations of remote users in physical spaces.
- **Sessions** - Lifecycle management around collaboration sessions, including per-session service management.
- **Spatial Awareness** - Localize sessions/content to a point in space, AR navigation, etc.
- **Shared Content** - Stores and recall persistent mixed-reality content associated with users and/or real-world spaces.
- **Subscriptions** - Subscription management including payments, billing, etc.
- **Transport** - Audio, video, and data transfer from device-to-device.
- **Communication** - Integration with device telephony, chat, messaging, etc.

The Mesh ecosystem

Mesh is innovating deeply in the following areas:

- Developing an understanding of 3D spaces, their content, and the devices that live in them.
- Optimizing local and cloud-based resources to work well together, including specialized workloads like Holoportation and other high-quality presence representations.

Supported platforms

Mesh is expected to run on multiple platforms, but currently only runs on *HoloLens 2* (HL2) hardware and *Windows 10* desktops, laptops and devices. For development, Mesh currently supports:

- C++ on DirectX for HL2 and PC
- Unity

Core concepts

Like all systems, Mesh has several core concepts, or building blocks, that power its features:

- Avatars: Learn about how people look and move in a session.
- Client: The entry point for the entire Mesh SDK.
- Identity: Identify your users and applications within the Mesh system.
  - This currently relies on the Microsoft identity platform.
- Sessions: Manage the life cycle of collaboration sessions, which includes co-localization and media.
- Session Origin: Learn about how participants share objects in a session.
- Spatial Audio: Learn about 3D sound works.
- Transport: Manage voice, video, and basic data/state sync.

It's easier to see these core concepts in action than as distributed bits of information. The next section provides a sample scenario that connects everything into a bigger and more efficient picture. You can find a more detailed version of this scenario, with graphics, in the Session Origin article.

A sample scenario

This section describes a sample scenario that touches on all the areas introduced above while showing how your users and applications will interact with the SDK services.

Alice and Bob walk into a cafe wearing their AR devices to share a holographic cup of coffee. Alice has a subscription to Mesh services; she is already signed in and has set her presence choice to "available" in her privacy settings. This means the system knows her identity and will show her as available to any friends in the area.

Alice creates a new session which can include nearby friends. In this case there is a single friend available: Bob. The service notifies Bob that he is close to an active session, and he decides to join in. With both parties involved, a session is set up with a trusted, direct transport option that minimizes latency and packet loss.

After Alice shows Bob how attractive her new coffee cup is in a real cafe setting, she saves the coffee hologram in the session in case she wants to show it later.

Services breakdown

The application that Alice and Bob are using leverages the Mesh services by:

- Relying on the system to verify identity and subscription status.
- Creating a session and inviting others.
- Sharing data through a secure real-time transport.
- Relying on the system to enforce security and to share data.

Get Started

Start building your first app using Mesh

- Get started with Mesh SDK

Next Steps

System requirements

System requirements and setup

4/10/2021 • 2 minutes to read • Edit Online

Device support

SDK	HOLOLENS 1ST GEN	HOLOLENS 2	WINDOWS PC
Mesh SDK	✕	✔	✔

> **NOTE**
> The Mesh SDK is constantly evolving, so check back for updates on additional platform support.

Required hardware

Windows PC

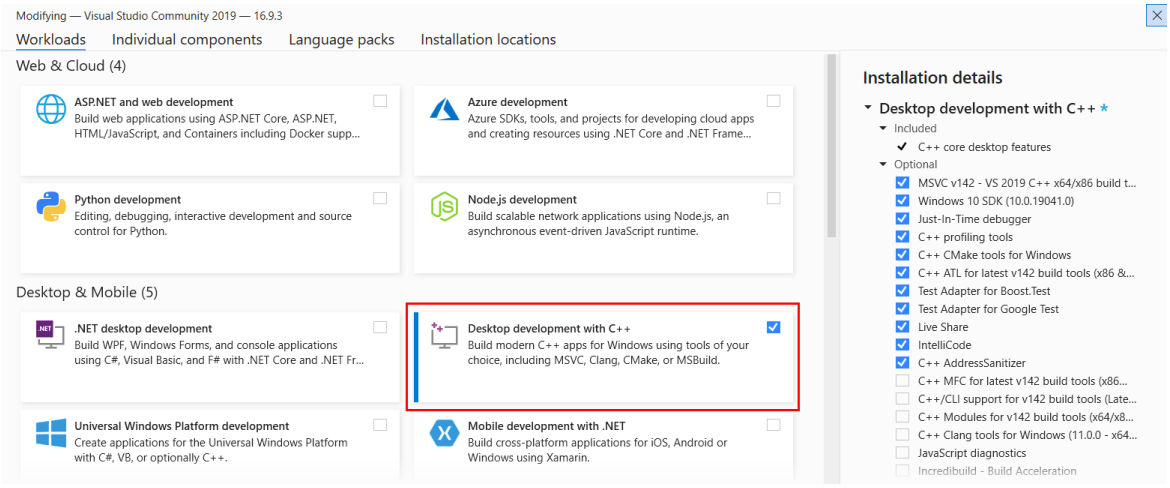
- Windows 10 version 1903 or higher.
- Up-to-date graphics drivers.
- Optional: H265 hardware video decoder, if you want to use local preview of remotely rendered content (for example, in Unity).

> **IMPORTANT**
> Windows update doesn't always deliver the very latest GPU drivers; check your GPU manufacturer's website for latest drivers:
>
> - AMD drivers
> - Intel drivers
> - NVIDIA drivers

Install Visual Studio (required)

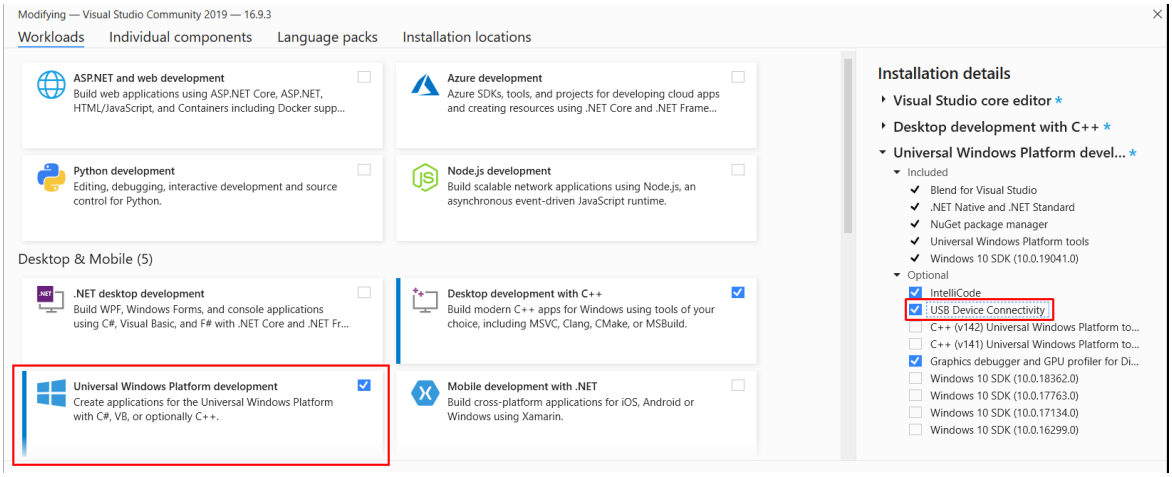
Select Workloads

1. Select the **Desktop Development with C++** workload.



2. Select the **Universal Windows Platform development** workload.

3. In the right-side **Installation Details** panel under **Optional**, select **USB Device Connectivity**.

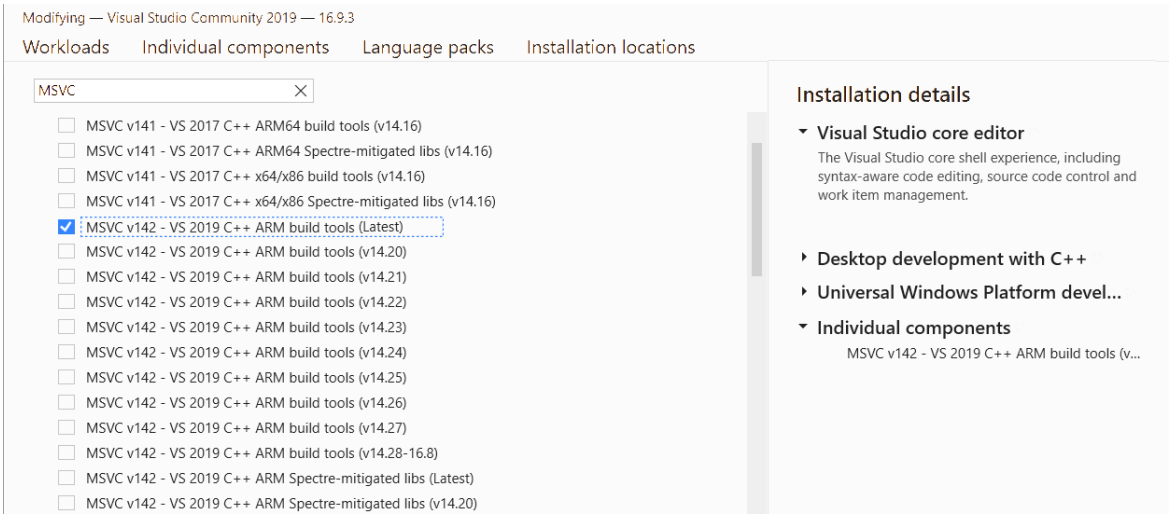


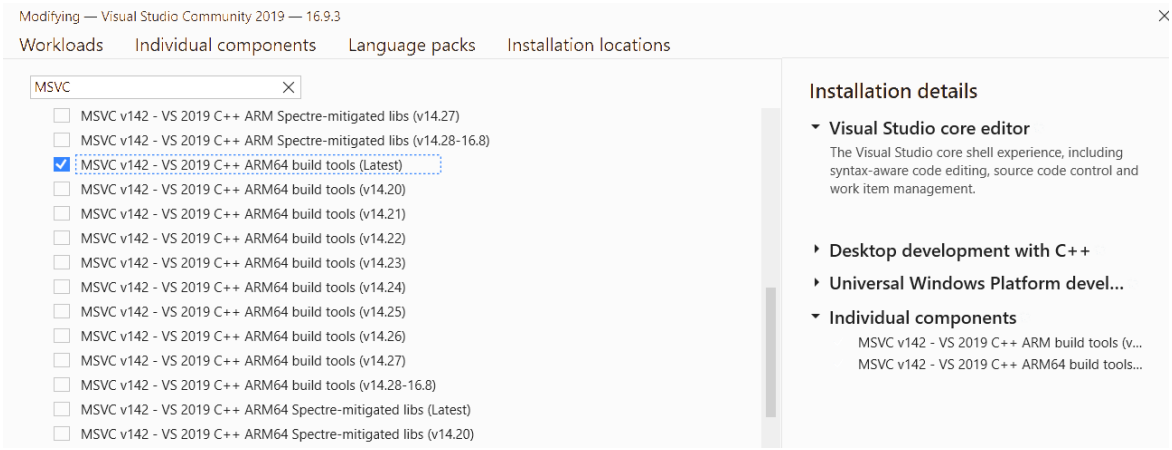
> **IMPORTANT**
>
> In order to communicate with a HoloLens over USB, during Visual Studio installation, you need to select **IP over USB**. This is not selected by default.

4. Select the **Individual Components** tab.

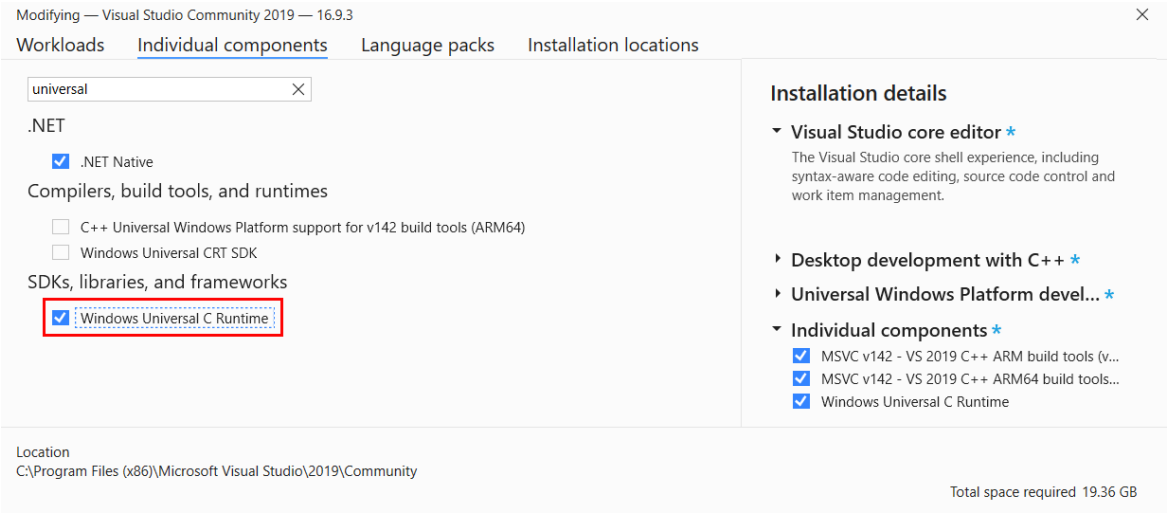
5. In the search box, enter "MSVC."

6. In the list, select the latest versions of **MSVC v142 - VS 2019 C++ ARM build tools** and **MSVC v142 - VS 2019 C++ ARM64 build tools**.





7. In the search box, enter "universal."
8. In the list, select **Windows Universal C Runtime**. Note that this isn't required, but would be needed for writing plugins and for lower-level C++ development.



The Visual Studio Unity workload

Unless you're intentionally trying to install a newer (non-LTS) version of Unity, we recommend **not** installing the Unity workload as part of Visual Studio installation. Install the Unity 2019 LTS stream instead.

Known issues

There are some known issues with debugging Mixed Reality apps in Visual Studio 2019 version 16.0. Ensure that you update to Visual Studio 2019 version 16.8 or higher.

Install Unity (optional)

If you're looking for the easiest way to get started creating Mesh apps, the Unity game engine is your best choice. We recommend the Unity LTS (Long Term Support) stream as the best version to use when starting new projects. To pick up the latest stable fixes, you can update to its latest revision.

- We recommend that you use **Unity 2019**, which is the LTS build required for the Mixed Reality Toolkit (MRTK) v2.
- If you need to use a different version of Unity for specific reasons, Unity supports side-by-side installs of different versions.

Install Git

You must also download and install Git.

More install instructions

For more details about installing the Windows SDK, Unity, the HoloLens 2 Emulator, and the Mixed Reality Feature Tool, as well as HoloLens tips and immersive (VR) headset requirements, see our Install the tools page.

Limitations

- UWP/x64 and Win32/x64 are currently the only supported binaries.

Next steps

Getting Started

See also

- Release notes

Getting started with Mesh SDK

4/12/2021 • 3 minutes to read • Edit Online

The Mesh platform provides services and client components to power social-first, rich and immersive experiences where remote and local collaborators can play, create, and collaborate.

Development checkpoints

Use the following checkpoints to get started on your Mesh development journey. If you're new to Mixed Reality, own a HoloLens, and haven't already explored the Designing Holograms sample application, we recommend downloading and using it to familiarize yourself with the basics of Mixed Reality UX.

1. Getting started

By the end of this section, you'll have a basic understanding what the SDK can do, a system requirements checklist, and a properly configured development environment for Mixed Reality apps.

CHECKPOINT	OUTCOME
What is the Mesh SDK	Begin your journey with services and client components that power social-first, rich, and immersive experiences on a trustworthy platform
System requirements	Make sure your development environment and hardware meet all the Mesh SDK requirements, and then set up your Mixed Reality project with Visual Studio and (optionally) Unity.

2. Setting up

Before you can start integrating Mesh SDK features into your Mixed Reality projects, you'll need to properly register your tenant, users, and application. By the end of this section, your app will be ready to authenticate users with server tokens and set up Mesh sessions.

CHECKPOINT	OUTCOME
Register your tenant	Link your organization and users to the Microsoft Identity Platform so Mesh can access authentication resources
Register your application with the tenant	Register your application with your tenant organization through Microsoft Azure
Request project scopes	Enable Mesh to read user and application information from your registered tenant
Set up redirect URIs	Set up your application redirect URIs to receive tokens when your users are successfully authenticated

NOTE

See the HoloLens 2 authentication tips for more details.

3. Core concepts

We recommend that you get familiar with each of these topics before you move on to any implementations of your own. After diving into the core concepts listed below, you'll have a toolbox full of Mesh features you can integrate into your own projects.

CONCEPT	CAPABILITIES
Client	Start configuring your client object with information about connection specifics and the client owner's user identity
Identity	Learn how to sign in your users with Microsoft Accounts (MSA) or Azure Active Directory (AAD)
Sessions	Manage your app's active participants and enable them to communicate with each other securely and efficiently
Session Origin	Synchronize spatial interactions of participants who may be at your physical location or in different locations
Transport	Set up a connecting layer between your users in the same session, transferring audio and video data to all parties

4. Quickstarts and Tutorials

Unity

We recommend the Unity quickstarts and tutorials for newer developers who are getting started with Unity programming.

Create a Session and Invite a Guest (Windows PC Only).

In this quickstart, you'll open a Mesh/Unity project and give the project a way to identify itself to Mesh. Next, you'll build an app on the Universal Windows Platform. Finally, you'll start a Mesh session in the Unity Editor as User #1, run the app you built as User #2, and then invite User #2 to the session.

Create a Session and Invite a Guest (Windows PC and HoloLens).

This quickstart is similar to the one above, except that instead of running the app as User #2 in the Unity player, you build and deploy it to the HoloLens.

Intro to Mesh Tools for Unity

The Mesh Tools for Unity are pre-configured tools that address common needs for collaboration. These tools help accelerate app creation, providing you with more time to focus on collaborating instead of creating the app.

C++

We recommend this quickstart path for experienced developers who are proficient with C++ programming. In this quickstart, you'll launch two instances of the app, then authenticate in both, and then start a session between the two instances.

Create a session and invite a guest.

Next steps

System Requirements and Setup

Learn about the Unity editor

Learn about the Unity scripting API

Learn about converting Photon PUN2 to Mesh

Register your tenant with Mesh

4/26/2021 • 2 minutes to read • Edit Online

> **NOTE**
> This is setup step 1 of 5

What is a tenant, and why do I need one?

Hereinafter, a *tenant* is simply a dedicated instance of an Azure Active Directory (Azure AD) that an app developer or organization receives and owns when it signs up for a Microsoft cloud service such as Azure, Microsoft Intune or Microsoft 365. For more information on Azure AD tenants, see the Azure AD documentation.

In addition to work and school identities, each tenant can also contain consumer identities (if it's an Azure AD B2C tenant), and app registrations.

The Mesh SDKs take a dependency on the Microsoft Identity Platform's consent and authentication infrastructure. For your application to use Mesh and have access to Mesh OAuth scopes, you will need to create (or already have) an Azure AD tenant that represents your organization that is building the application. Because this is a pre-release SDK, we must flight Mesh features to your Azure AD tenant.

Prerequisites

You will need:

- An Azure subscription.
- An Azure Active Directory tenant in which to register applications. If you don't already have one, this quickstart can help you learn how to set one up.
- If you are working with commercial users, one or more Work or School accounts.

Next steps

Register your application with the tenant

Register your application with the tenant

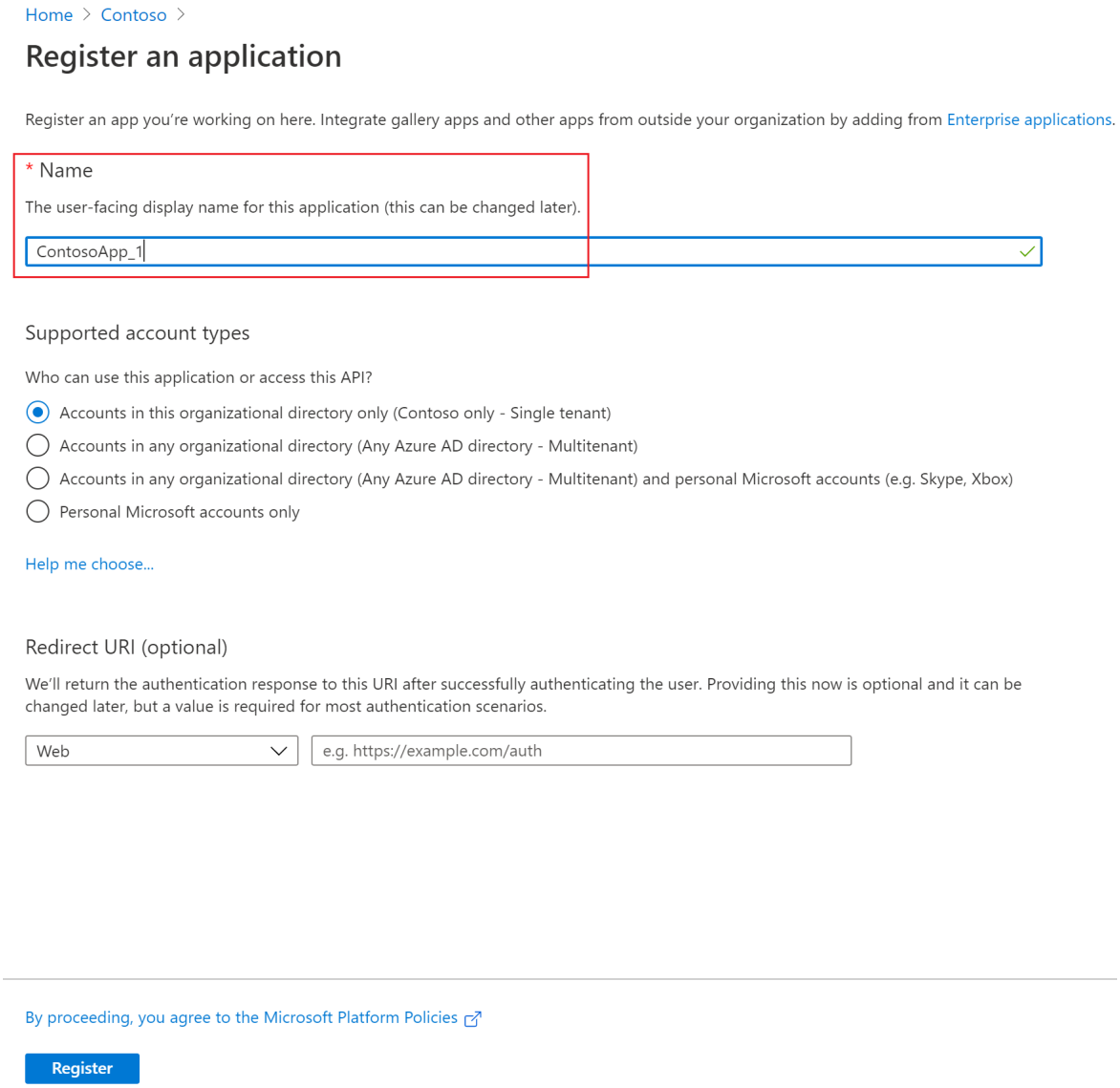
4/8/2021 • 2 minutes to read • Edit Online

> **NOTE**
> This is setup step 2 of 5

To use Mesh APIs, you must register your app with the tenant.

Register a new application using the Azure portal

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.
3. Search for and select **Azure Active Directory**. In the left-side navigation under **Manage**, select **App registrations**.
4. Select **New registration**.
5. On the **Register an application** page, enter a meaningful application name to display to customers.



6. Under **Supported account types**, specify who can use the application. Select the first option:

7. When finished, select **Register**. This takes you to your application's **Overview** page.

To add capabilities to your application, you can select other configuration options including branding, certificates and secrets, API permissions, and more.

Next steps

Request Mesh scopes for your project

Request permissions for your project

4/26/2021 • 3 minutes to read • Edit Online

> **NOTE**
> This is setup step 3 of 5

At this stage of the Mesh private preview, you must manually add Mesh to your tenant:

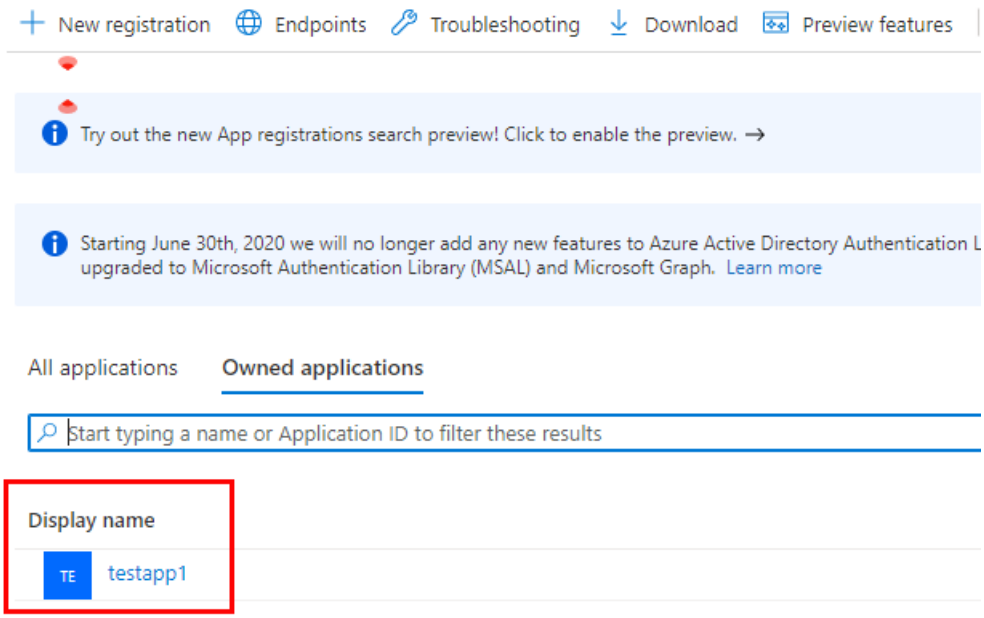
1. Visit
   `https://login.microsoftonline.com/tenant-ID/oauth2/authorize?client_id=98e6160b-4308-432a-b82d-ed6fce38dfbf&response_type=code`
   , replacing *tenant-ID* with the ID of the Azure Active Directory tenant where [you registered your app][register].
   - You can find a tenant's ID from its Azure Active Directory overview blade in the Azure portal. Make sure you have selected the tenant where you registered your app in the upper right corner.
2. When prompted to log in, do so with an account in that tenant. > [!Note] > If you have accounts in more than one tenant, we recommend that you use your browser's private browsing mode so that you're not automatically logged in and can pick the account you want to log into
3. Accept the permissions requested for "Microsoft Mesh (Preview)."
4. After consenting, you will be taken to a page confirming that Microsoft Mesh (Preview) has been configured in your tenant.
5. If Mesh has already been registered in the tenant, you may see the prompt: "Are you trying to sign in to Microsoft Mesh (Preview)?" If so, select **Continue**.

Add permissions

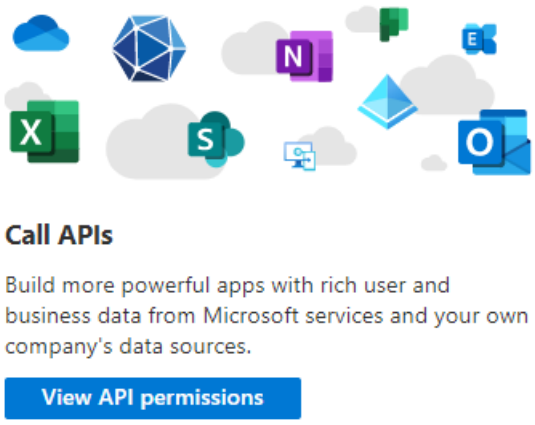
For your app to access Mesh services and other related services, it must request certain permissions. You'll request these in the steps below.

Navigate to the "API permissions" page

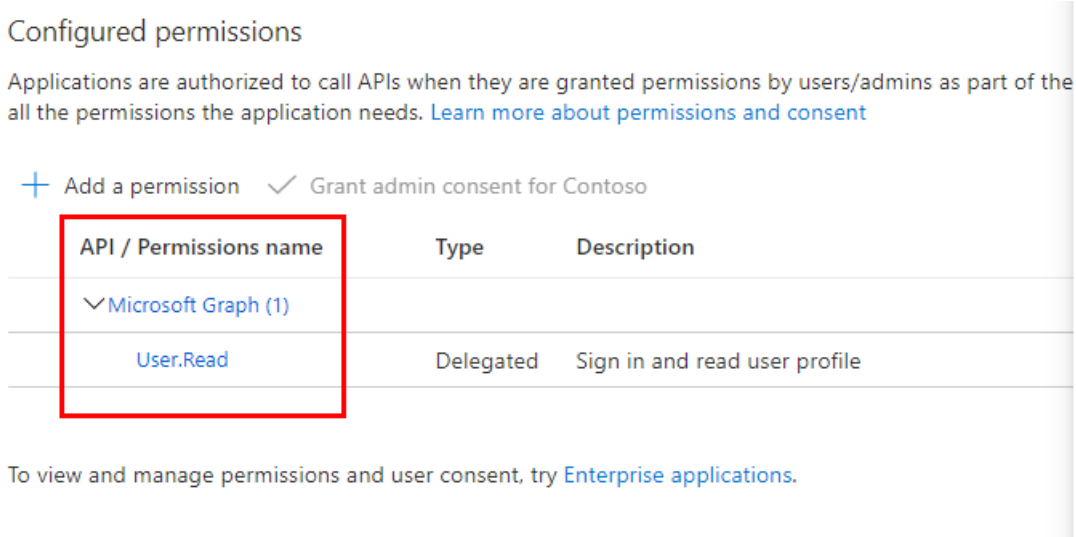
1. In the upper right corner of the [Azure portal][azure-portal] screen, click the account popup and then select the tenant where [you registered your app][register].
2. Search for and select **Azure Active Directory**. In the left-side navigation under **Manage**, select **App registrations**.
3. Find your app by either using the search box or looking in the **All Applications** or **Owned Application** tabs.



4. Click the app name, and then click **View API permissions**.



Note that in the **Configured permissions** table, there has already been one permission added by default: **Microsoft Graph / User.Read**.



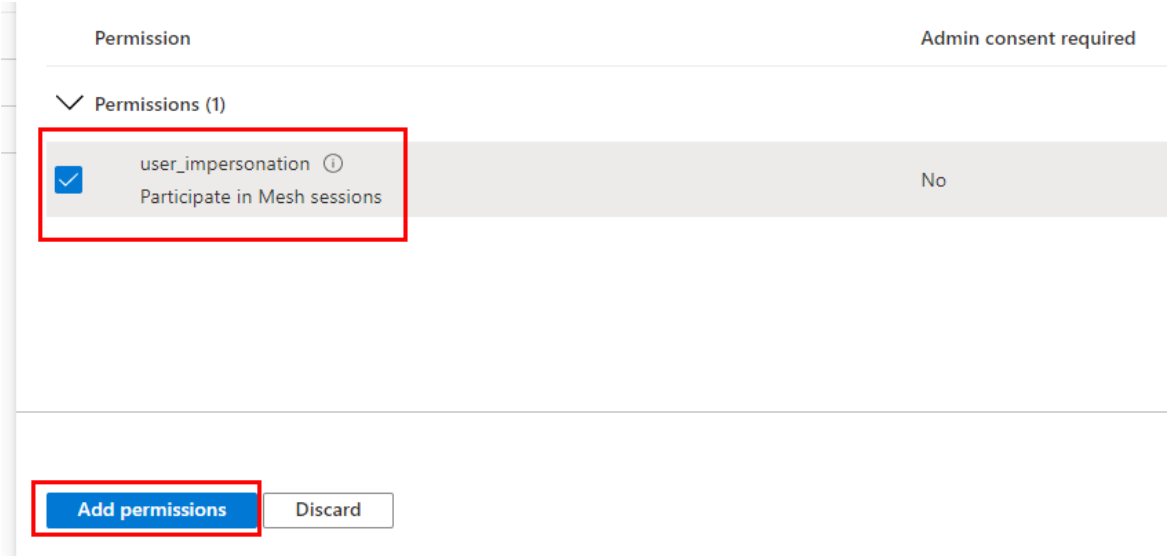
Add the Mesh permission

1. Select **Add a permission.**

2. In the **Request API permissions** panel, select **APIs my organization uses**.
3. In the search box, enter "Microsoft Mesh."
4. Select **Microsoft Mesh (Preview)**. (The application id should be "98e6160b-4308-432a-b82d-ed6fce38dfbf").

> **NOTE**
>
> If the "Microsoft Mesh (Preview)" API doesn't show up in the list, there are a few possible causes:
>
> - Mesh was added to the wrong tenant. If you have access to multiple Azure tenants, make sure you added the permissions to the correct tenant by following the "Add Mesh to your tenant" instructions above.
> - There's a longer-than-normal delay between the time you added Mesh to the tenant and its eventual appearance in the portal. This delay is usually only a few minutes, but can be up to twelve hours.
> - If you're using an early preview of Mesh, your tenant may have an older name for Mesh such as "Salvador Services App" or "Microsoft Mixed Reality Collaboration Services." If the "Microsoft Mesh (Preview)" API doesn't show up in the list, search by its application ID: "98e6160b-4308-432a-b82d-ed6fce38dfbf."

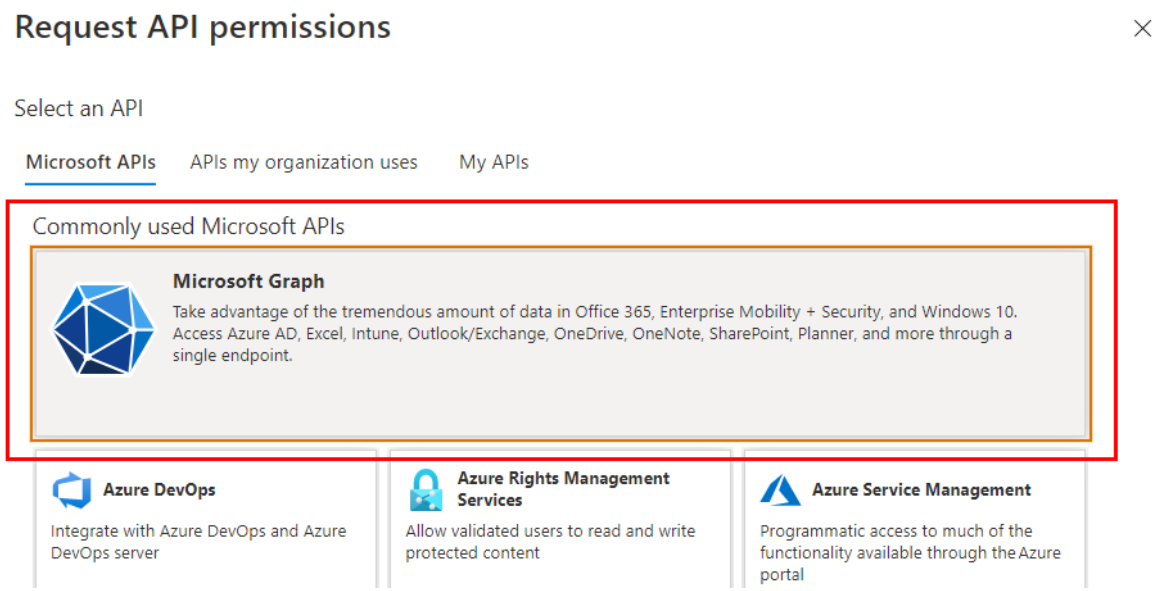
5. Select **Delegated permissions**.
6. There is only one available permission: **user_impersonation**. Select this, and then select the **Add permissions** button.



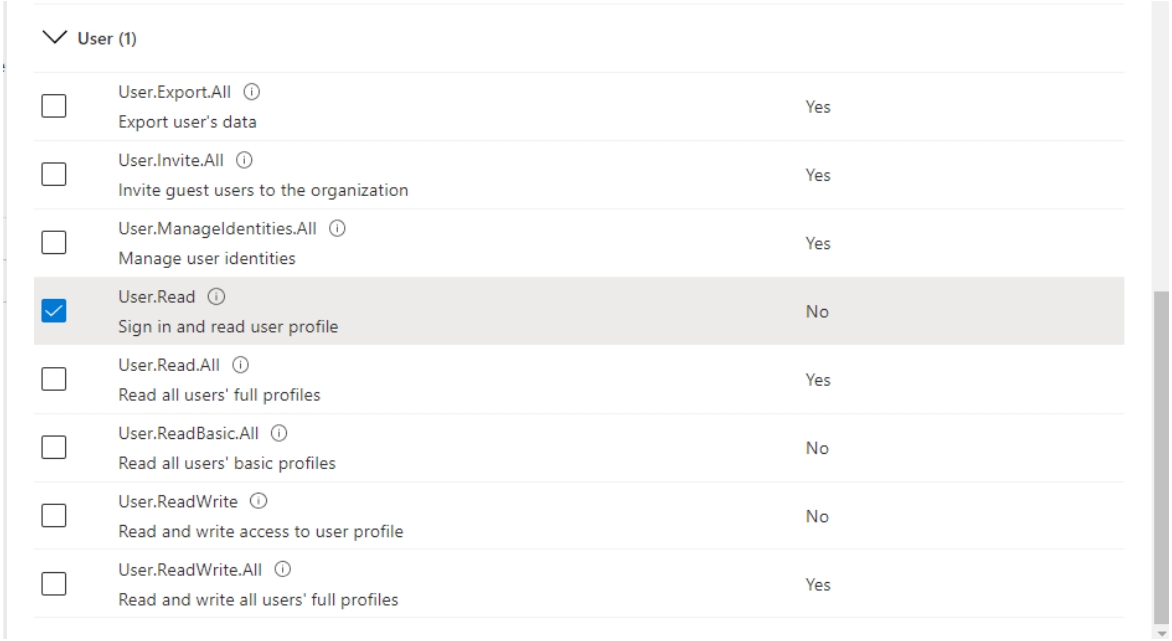
Add the Microsoft Graph permission

The Microsoft Graph User.ReadBasic.All permission allows users to view basic information about other users in the tenant, which is useful for a multi-user collaborative application. Our quickstarts and tutorials use this permission, and many Mesh apps will likely need it, too.

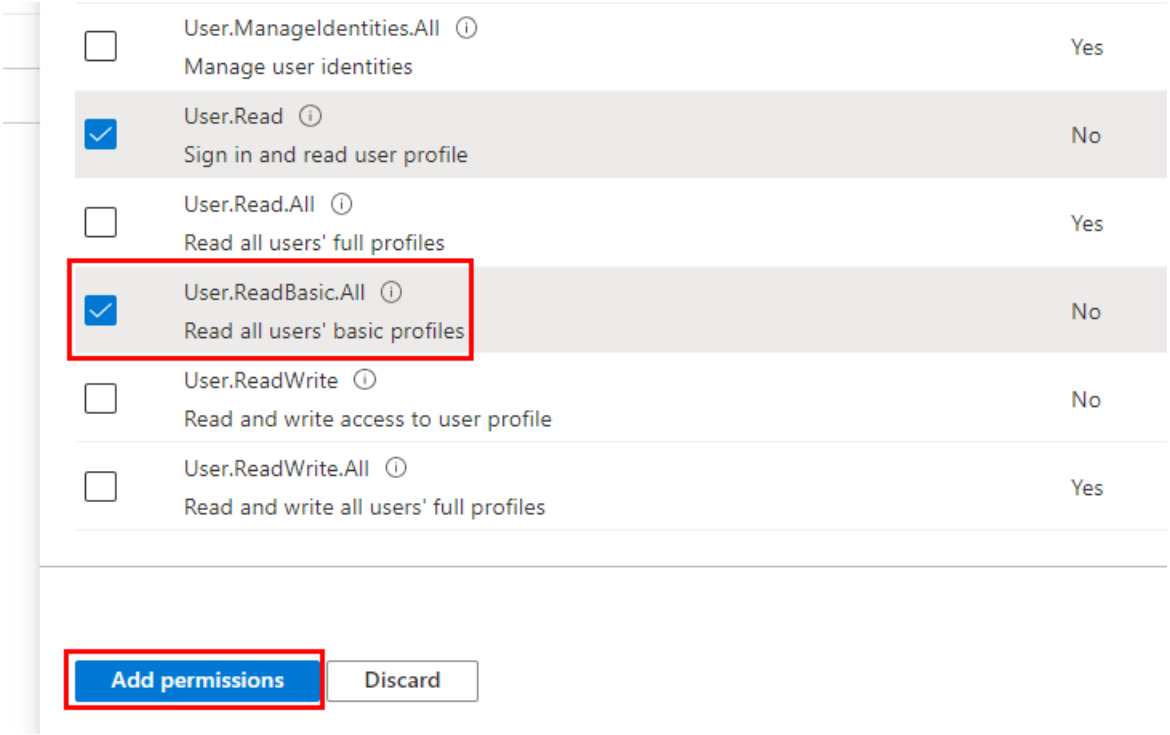
1. Select **Add a permission.**
2. In the **Request API permissions** panel, select **Microsoft Graph**.



3. Select **Delegated permissions**.
4. In the **Select permissions** box, enter "user."
5. Scroll down until you see the **User** permissions.



6. Select **User.ReadBasic.All**, and then select the **Add Permissions** button.



Note that User.ReadBasic.All works for enterprise apps but not MSAs.

Redirect errors

If you receive the following error, have an Azure Active Directory app or tenant administrator perform the registration for you:

> **Microsoft Mesh (Preview) needs permission to access resources in your organization that only an admin can grant. Please ask an admin to grant permission to this app before you can use it.**

Tenant admins should use the following URL to add the Mesh app to your tenant.

```
https://login.microsoftonline.com/tenant-ID/oauth2/authorize?client_id=98e6160b-4308-432a-b82d-ed6fce38dfbf&response_type=code&prompt=admin_consent
```

Next steps

Set up redirect URIs

Set up redirect URIs

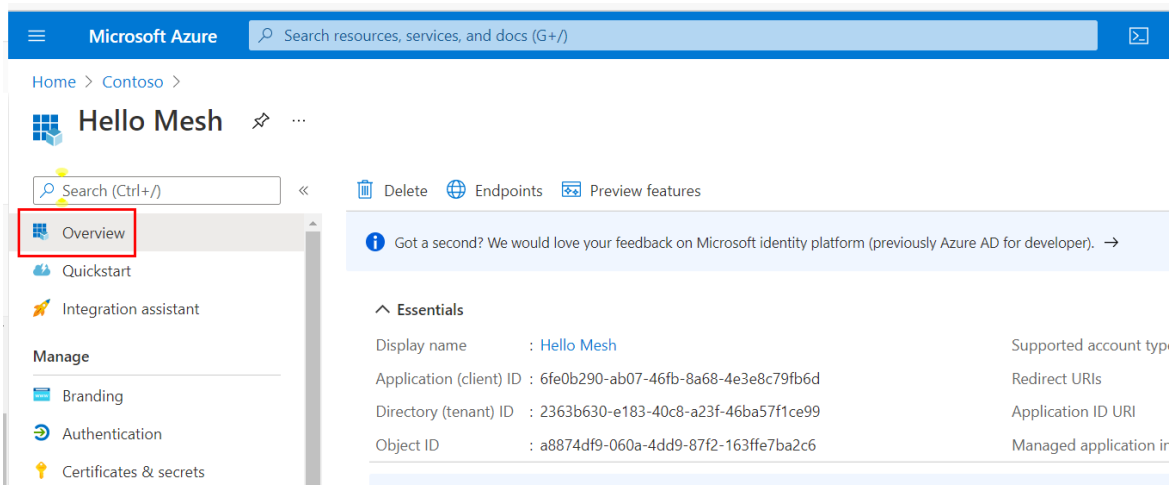
4/13/2021 • 2 minutes to read • Edit Online

NOTE

This is setup step 4 of 5

After Azure authenticates a user, it returns an authentication response (in other words, a *token*) to a *redirect URI.* You may sometimes see redirect URIs referred to as *reply URLs*.

1. In the Azure portal, go to your application's **Overview** page.



2. In the left-side navigation under **Manage**, select **Authentication**.
3. In the **Platform Configurations** section, click **Add a platform**.

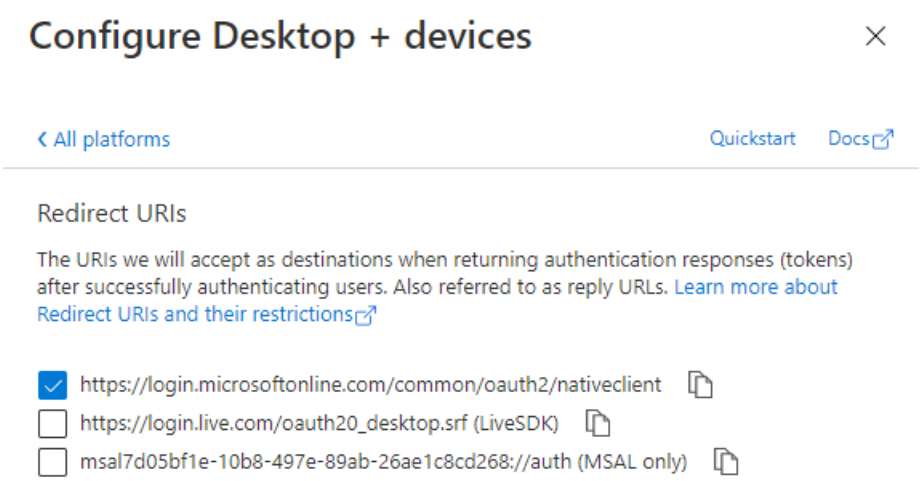
As this point, the instructions are different for Windows and Android.

For Windows:

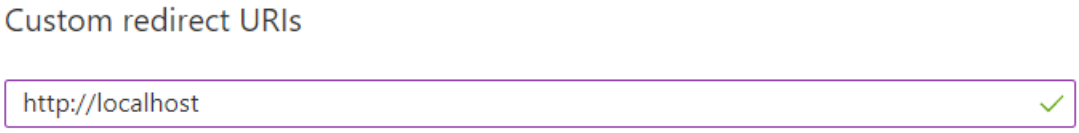
1. In the **Configure platforms** panel, click **Mobile and desktop applications**.
2. Do one of the following:

   If your app uses an embedded browser: Under **Configure Desktop + devices**, select the first redirect URI:

   `https://login/microsoftonline.com/common/oauth2/nativeclient`.



3. **If your app uses a system browser**: In the **Custom redirect URIs** box, enter `http://localhost`.

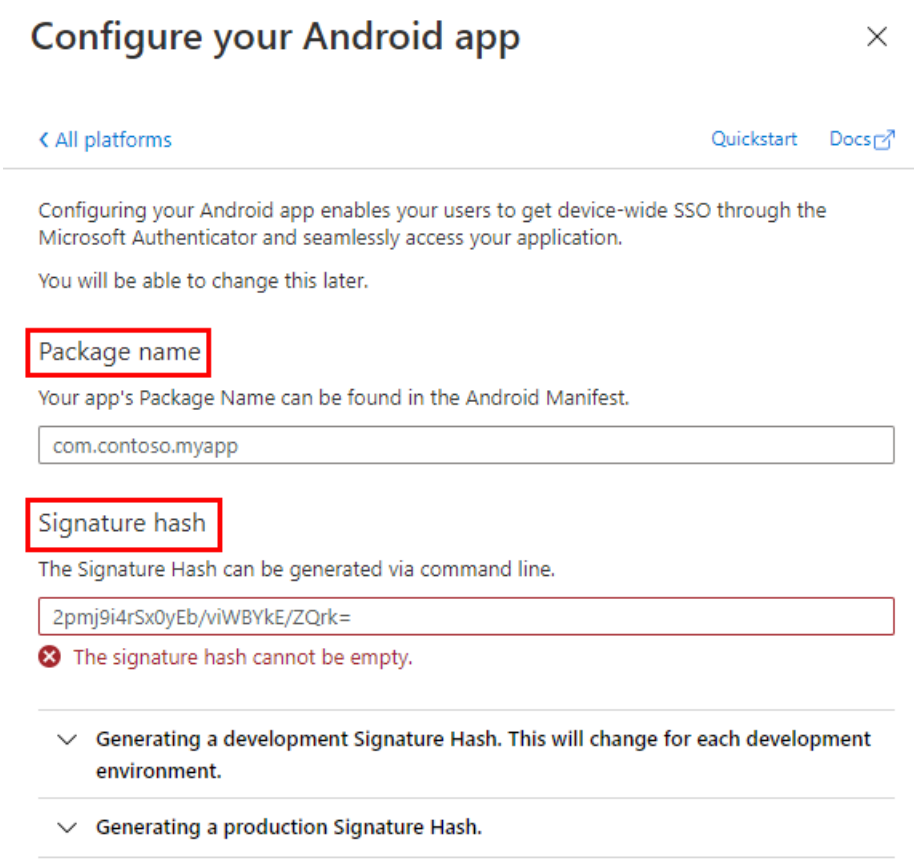


> **IMPORTANT**
>
> - If you're a Unity user, this makes authentication work in Editor mode.
> - Make sure you enter **http**, and not **https**, in the box.

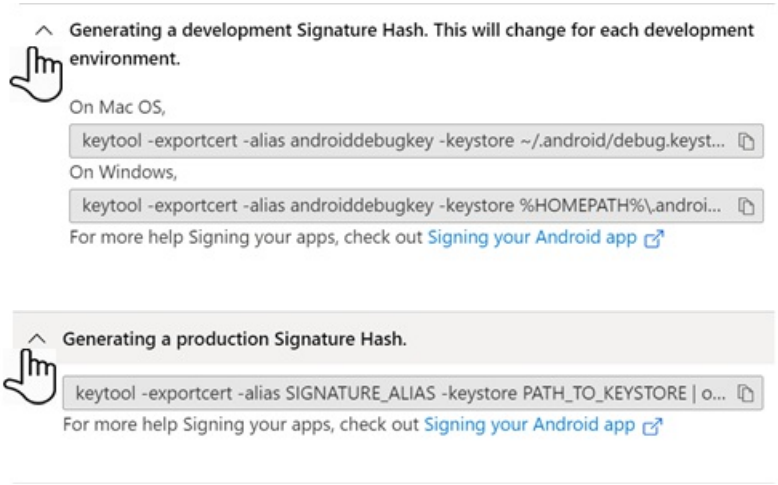
For Android

1. In the **Configure platforms** panel, click **Android**.

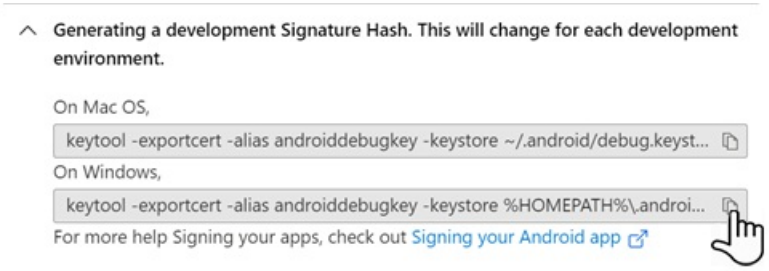
   This opens the **Configure your Android App** panel, where you can add your app's package name and signature hash. The **Signature hash** section contains commands that you can copy and then run in your Command Prompt/Terminal/shell to generate the hash.



2. In the **Package name** section, add the package name to the text box. **Note for Unity users**: the package name needs to be **com.unity3d.player**, which is generated by Unity.
3. In the **Signature hash** section, select **Generating a development Signature Hash ...** or **Generating a production Signature Hash**, depending on which one you want.

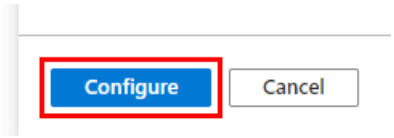


4. Click the Copy icon for the command string you want.



> **NOTE**
>
> The command strings make use of two tools. **keytool** is available through the Java SDK (standalone, or with Android Studio). **openssl** is available through Git and can also be installed independently.

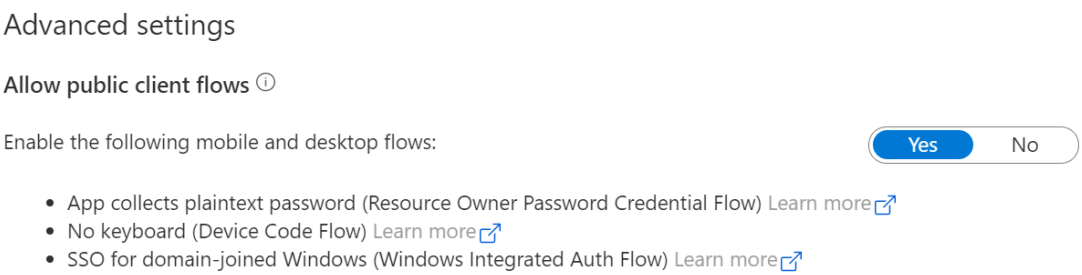
5. In your Command Prompt/Terminal/shell, paste the string and then generate the hash.
6. Copy the hash and then, in Azure, paste it into the text box in the **Signature hash** section.
7. Click **Configure**.



8. Oculus Quest users only: The Microsoft Authentication Library (MSAL) can't launch Oculus built-in Web browsers for the interactive login process. The device code flow is used instead.

To enable the device code flow

- Under **Advanced settings**, for **Allow public client flows**, select **Yes**.



Next steps

Learn about authentication on HoloLens 2

View our C++ Quickstart.

HoloLens 2 authentication tips

4/21/2021 • 2 minutes to read • Edit Online

> **NOTE**
> Follow these instructions **after** you've built your app.

When you deploy your application on the HoloLens 2 platform, your app automatically works with authentication based on MSAL. If you prefer, you can add WAM support, which streamlines the log-in process for users.

MSAL:

- Web-based UI that many users of Windows applications will be familiar with.
- The only authentication option available for use inside the Unity editor.
- Can be used with UWP applications for desktop or other devices.
- Requires no additional steps in the Azure portal.

Web Account Manager (WAM):

- Non-Web UI that can only be used with UWP applications.
- Provides a more OS-integrated authentication experience on HoloLens 2. It makes use of the user credentials already on the device so the user does not need to type a password in order to log in.
- Requires registration of an authentication ID as described below.

Using WAM for authentication on HoloLens 2

The following steps are required when using WAM.

Windows 10 provides each application with a unique URI and ensures that messages sent to that URI are only sent to that application.

To determine the redirect URI for your app:

1. Download the AppPackageInfo tool.
2. Get your app package path and run the tool on the command line. The tool will take any of the following:

- package.appxmanifest
- appxmanifest.xml
- .appx
- .appxbundle
- .msix
- .msixbundle

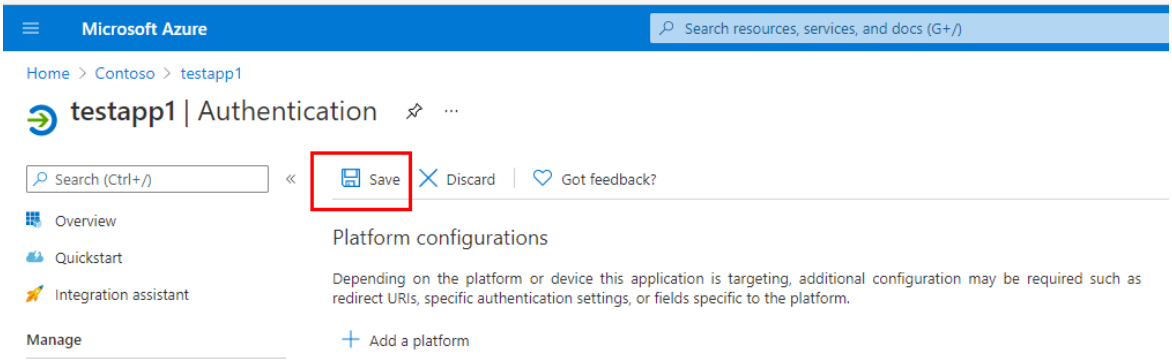
1. Copy the Package SID value to the clipboard.

```
Package SID = S-1-15-2-135638902-2382270379-399363827-2450642689-887555748-1760319418-140502344
```

> **NOTE**
>
> There is a risk of getting a wrong value if you use just the appxmanifest.xml or package.appxmanifest where if the signing cert doesn't match the publisher, the package publisher will prefer the cert. If it isn't working, try using `.appx`, `.appxbundle`, `.msix` or `.msixbundle`.

2. Sign in to the Azure portal.
3. Search for and select **Azure Active Directory**. In the left-side navigation under **Manage**, select **App registrations**.
4. Under **Display Name**, select your app.
5. In the left-side navigation under **Manage**, select **Authentication**.
6. Under **Mobile and desktop applications**, paste the following into the editable box located below the three listed options.

   `ms-appx-web://Microsoft.AAD.BrokerPlugIn/` <your Package SID>
7. Select **Save**.



See also

- Microsoft HoloLens

Quickstart: Create a session and invite a guest in Unity (Windows PC only)

4/23/2021 • 6 minutes to read • Edit Online

Get started in Mesh by creating a session and inviting participants. This is how you collaborate with your team members on an architectural project, or meet up with friends for a few rounds of virtual Checkers or Go, or host an Augmented Reality birthday party complete with virtual balloons, games, and a clown! In this quickstart, you'll open a Mesh/Unity project and give the project a way to identify itself to Mesh. Next, you'll build an app on the Universal Windows Platform and run it in the Unity player. Finally, you'll start a Mesh session in the Unity Editor as User #1, run the app you built as User #2, and then invite User #2 to the session. This is the first step in establishing the **presence** of each participant in a session so you can start interacting, collaborating, etc.

If you have a HoloLens 2 and want to use it to run the app that you will build, see our other other Unity quickstart.

Prerequisites

You will need:

- An Azure account with an active subscription. Create an account for free.
- A Windows computer withUnity 2019.4 (LTS)andVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with setting up your developer environment.
- If you are working with commercial users, one or more Work or School accounts.
- The latest version of the Mesh private preview zip file.

Caution

Caution: When working on Windows, there is a MAX_PATH limit of 255 characters. Unity is affected by these limits and may fail to compile if any file path is longer than 255 characters. Therefore, we strongly recommend that you save your Unity project as close to the root of the drive as possible and keep file and folder names short.

Contents of your private preview zip file

You can find the Unity project in the zip file here:

`PrivatePreviewZip\Samples\Unity\MeshQuickStart`.

There is a PDF version of the Mesh online documentation here:

`PrivatePreviewZip\Documentation`.

Create a session and invite a guest

Open the sample scene and import TextMeshPro

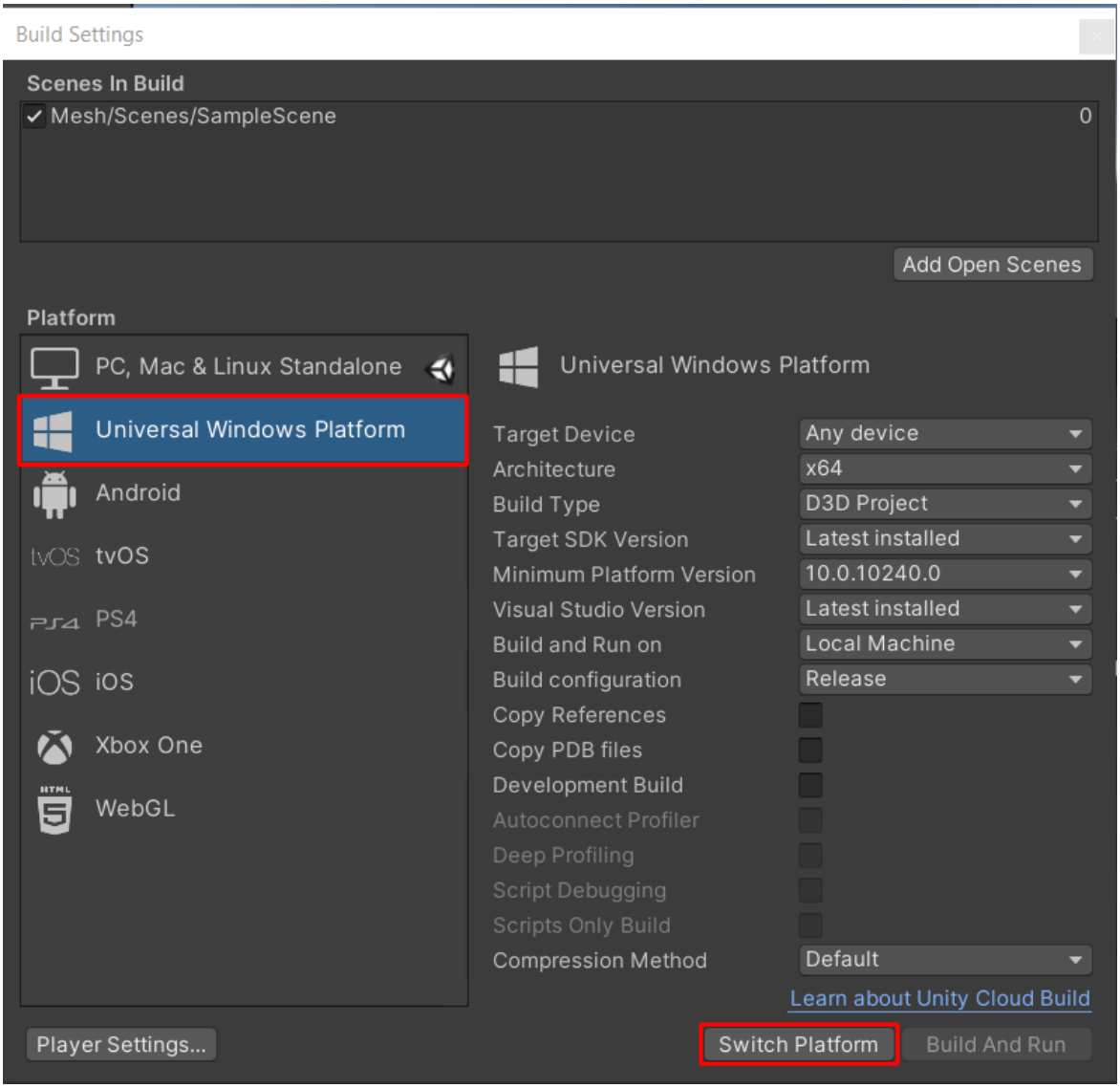
1. In the Unity Editor, open the sample project.
2. In the **Project** window, navigate to the **Assets** > **Mesh** > **Scenes** folder, and then open the scene

SampleScene.

3. On the menu bar, select **Window** > **TextMeshPro** > **Import TMP Essential Resources**. This is required for displaying text.
4. In the **Import Unity Package** window, click the **All** button, and then click **Import**.

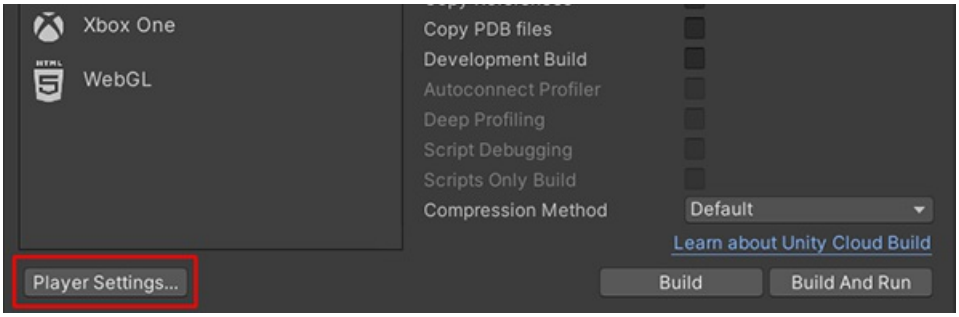
Switch the build platform

1. On the menu bar, select **File** > **Build settings...** .
2. In the **Platform** panel, select **Universal Windows Platform**, and then click **Switch Platform**. Unity will begin the process to switch the platform.

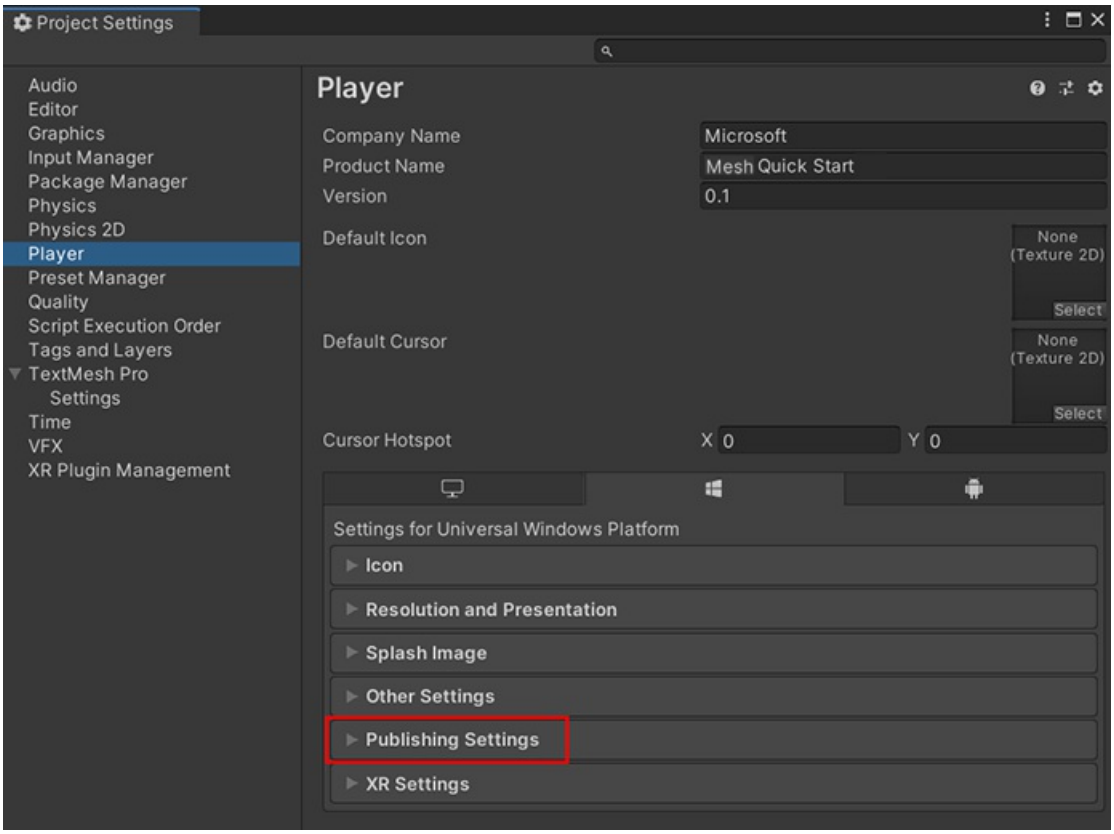


Add Capabilities

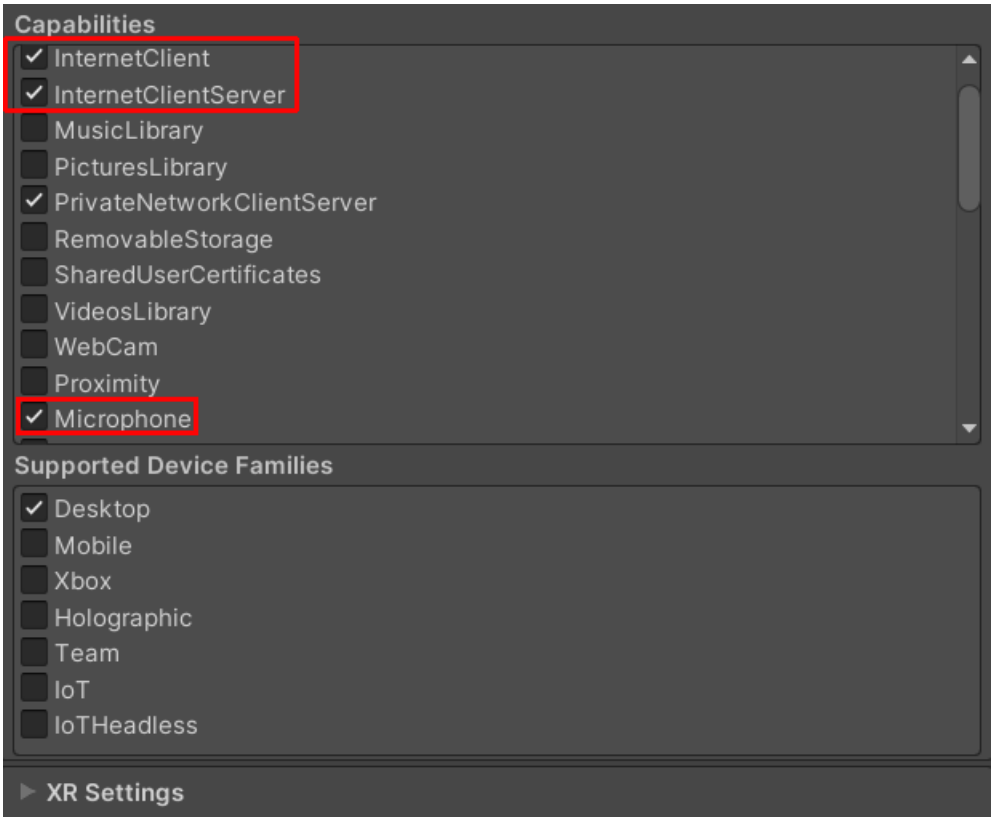
1. Click **Player Settings**.



2. In the **Player** panel, click the **Publishing Settings** drop-down.



3. In the **Publishing Settings** section under **Capabilities**, make sure you have **InternetClient**, **InternetClientServer**, and **Microphone** selected.



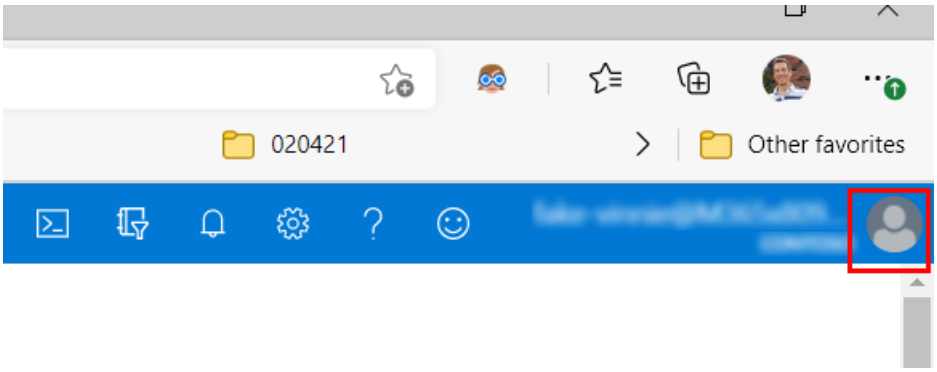
4. Close the **Project Settings** dialog.

Add the client ID and tenant ID to your project

When your application connects to the Mesh service, the service needs a way to identity it. You provide this capability by adding the client ID that was created earlier for your application when you registered it with your tenant, along with your tenant ID.

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.

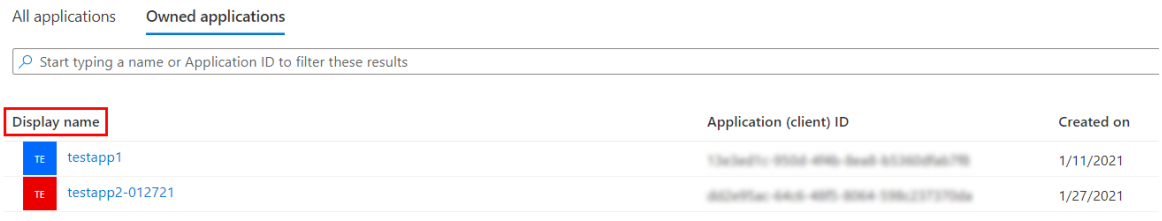
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.



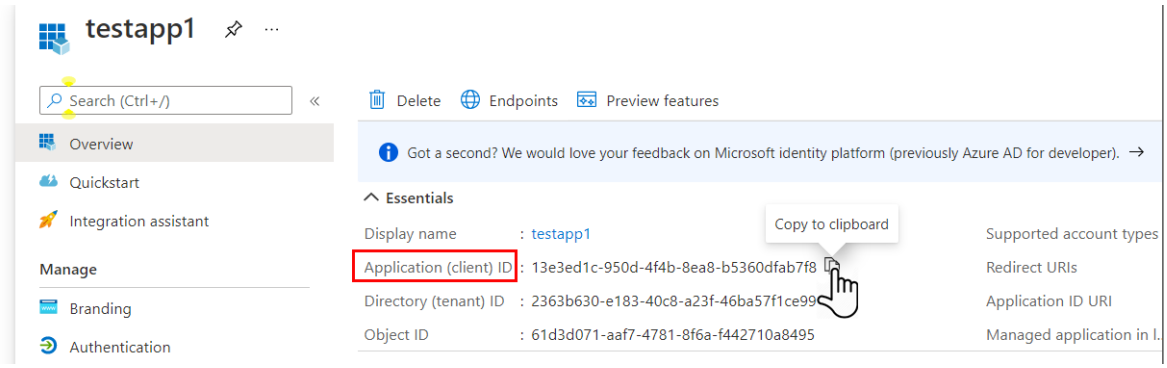
3. Search for and select **Azure Active Directory**. The easiest way is to click the **View** button on the page.

4. In the left-side navigation under **Manage**, select **App registrations**.

5. Under **Display Name**, click the name of your application.

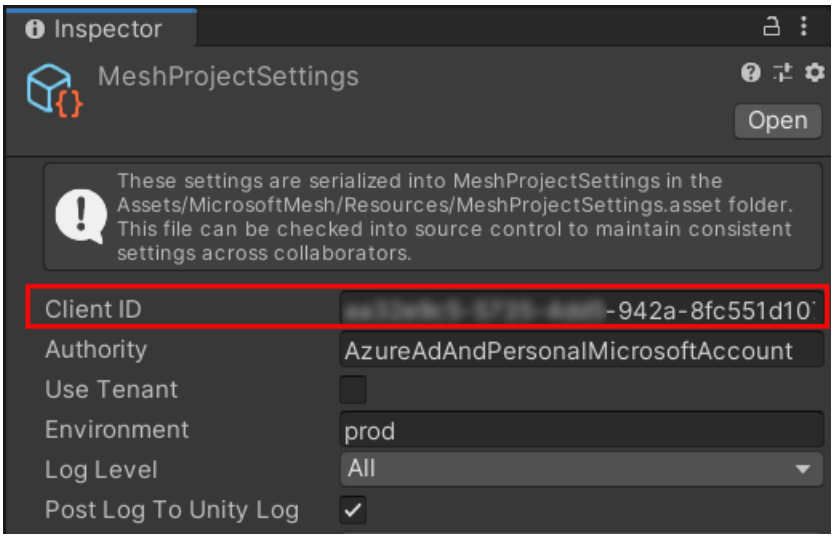


6. In the **Essentials** section, copy the **Application (client ID)** string to the clipboard. To do this, move your cursor just past the end of the ID, and then click the **Copy to clipboard** icon that appears. You can also select the ID and then press Ctrl + C.

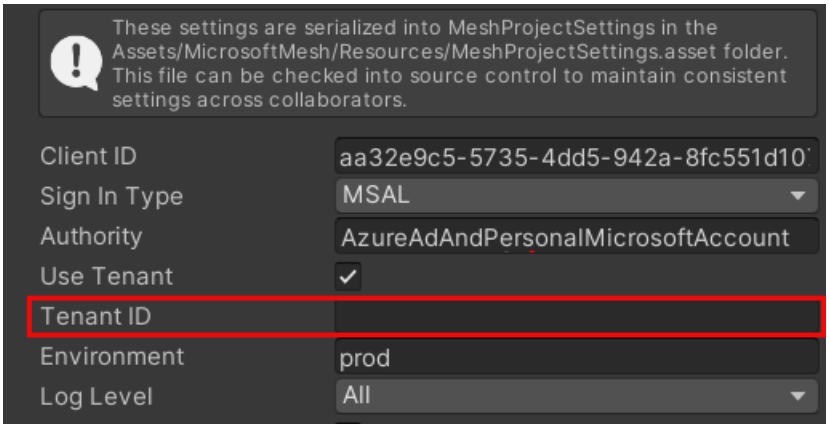


7. In Unity on the menu bar, select **Microsoft Mesh** > **Settings**.

8. In the **Inspector**, paste the client ID into the **Client ID** box.

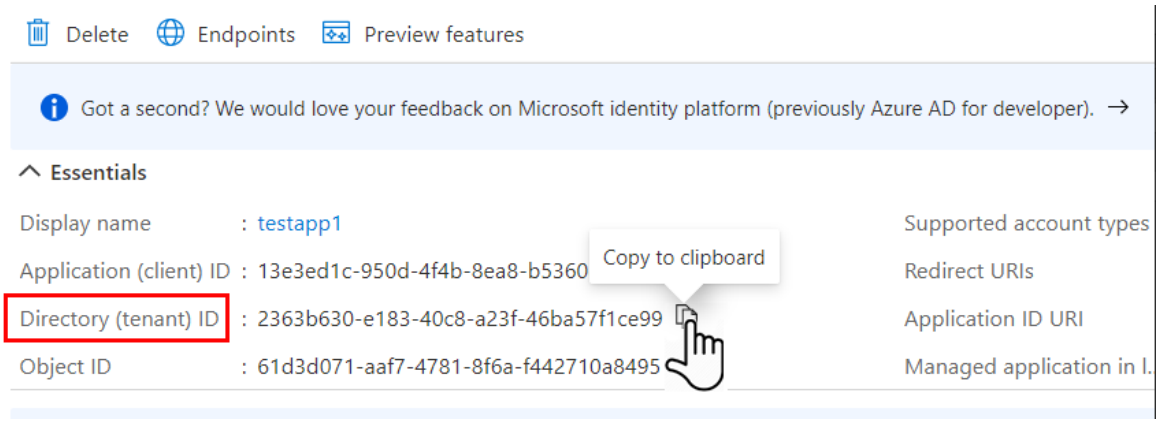


9. Select the **Use Tenant** checkbox. This causes the **Tenant ID** box to appear.

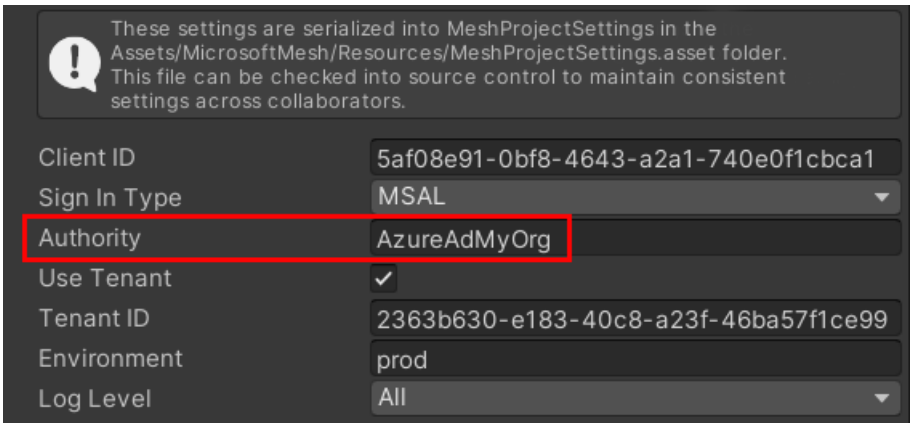


> **NOTE**
>
> The default **Sign in Type** is **MSAL**. This is the right option for this tutorial since you'll be building the app to the Unity Player. If you were building and deploying to the HoloLens, you would select **WAM** for the sign-in type. To learn more about MSAL and WAM, see HoloLens 2 Authentication Tips

10. Return to the app registration page in your browser, and then in the **Essentials** section, copy the **Directory (tenant) ID** string to the clipboard. You can do this in the same ways you copied the client ID.



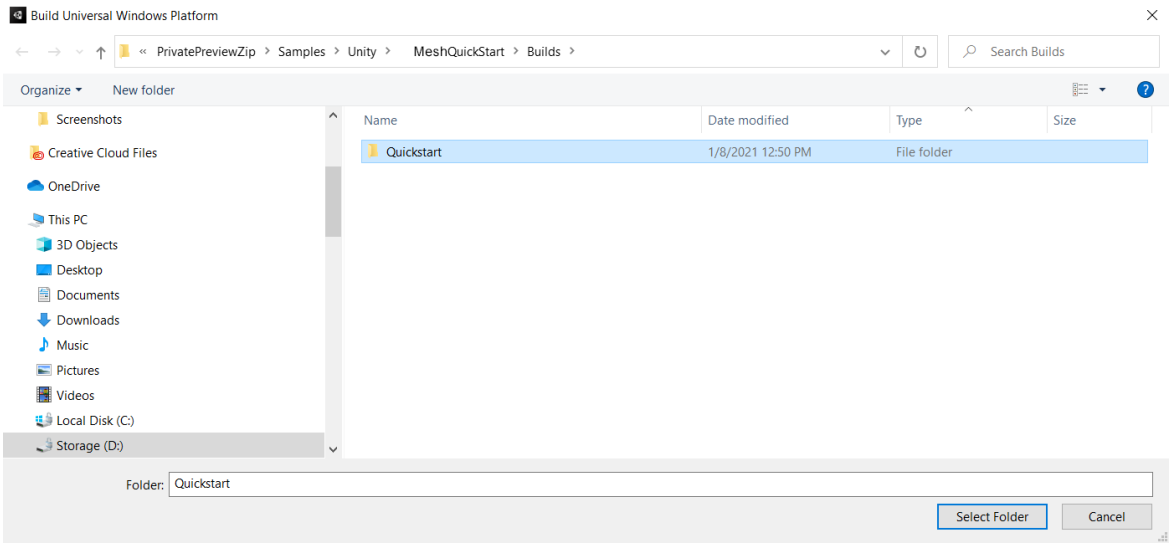
11. In Unity in the **Inspector**, paste the tenant ID into the **Tenant ID** box.
12. In the **Authority** box, enter an authority that matches the account type you selected when you registered your app in the Azure portal.



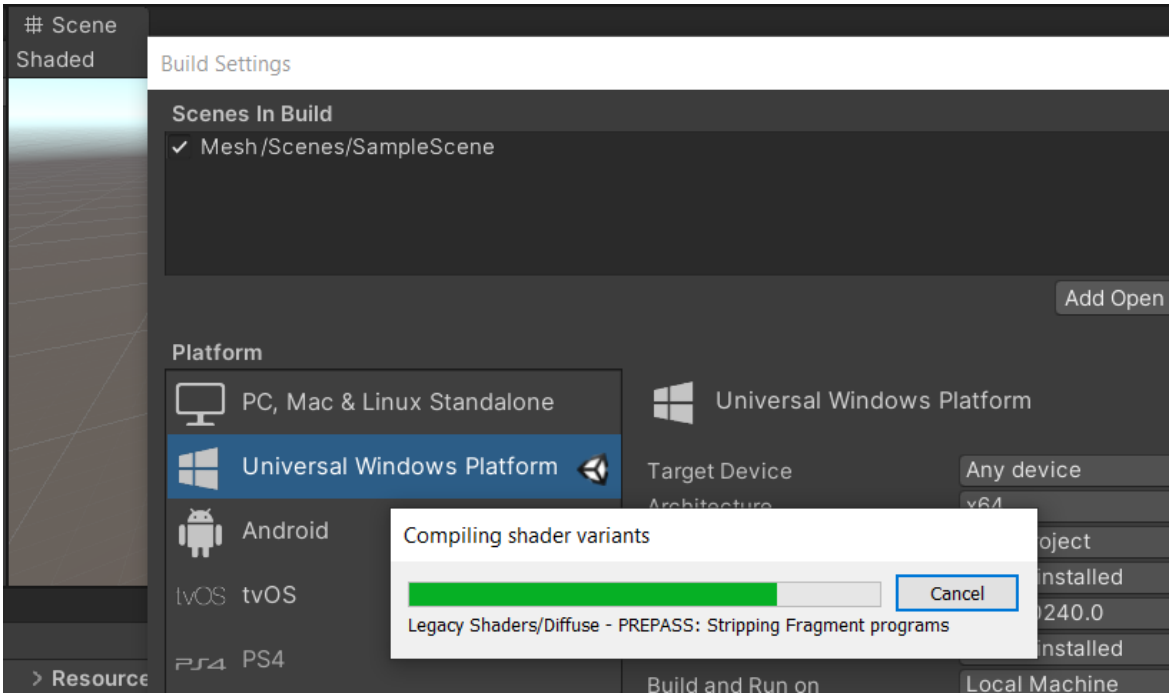
- If you selected **Accounts in the organizational directory only (Mesh Test Tenant only - Single tenant)**, in the **Authority** box, enter "AzureADMyOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**, in the **Authority** box, enter "AzureADMultipleOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)**, in the **Authority** box, enter "AzureADandPersonalMicrosoftAccount"

Build your application

1. On the menu bar, select **File** > **Build settings...** .
2. In the **Build Settings** dialog, click **Build and Run**.
3. In the **Build Universal Windows Platform** dialog, choose a suitable location to store your build (for example, you may want to create a "Builds" folder in your project). Create a new folder and give it a suitable name (for example, "Quickstart"), then select the folder, and then click the **Select Folder** button to start the build process.



A status bar appears and keeps you updated on your build progress.

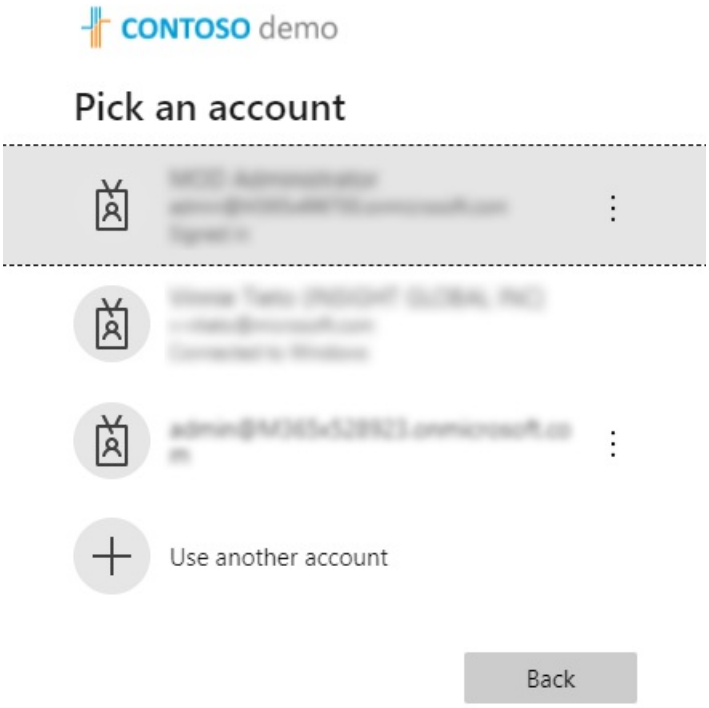


4. When the build finishes, the Mesh Quickstart app appears in a new Unity Player window. If a dialog appears asking you to log in, do so with your user name and password. In the Player window, you can see that your newly-built app has connected to the Mesh service.

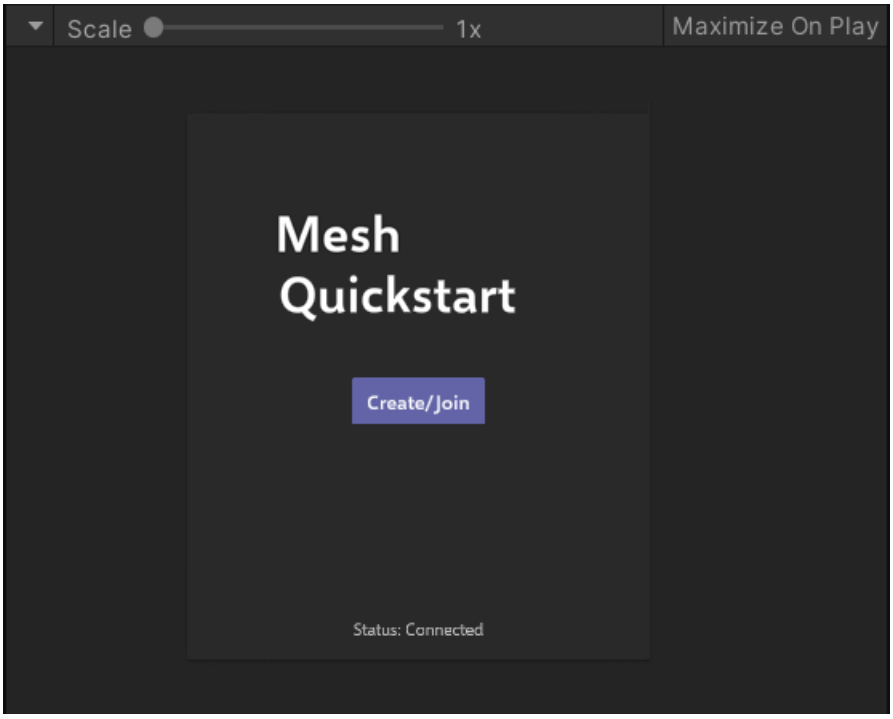
5. Minimize the Player window. We'll return to it in a moment.

Start a session in your Unity project

1. In the Unity Editor, close the **Build Settings** dialog.
2. Click the Play button. If the **Pick an account** log-in dialog appears in your browser, supply your user name and password as you did earlier.

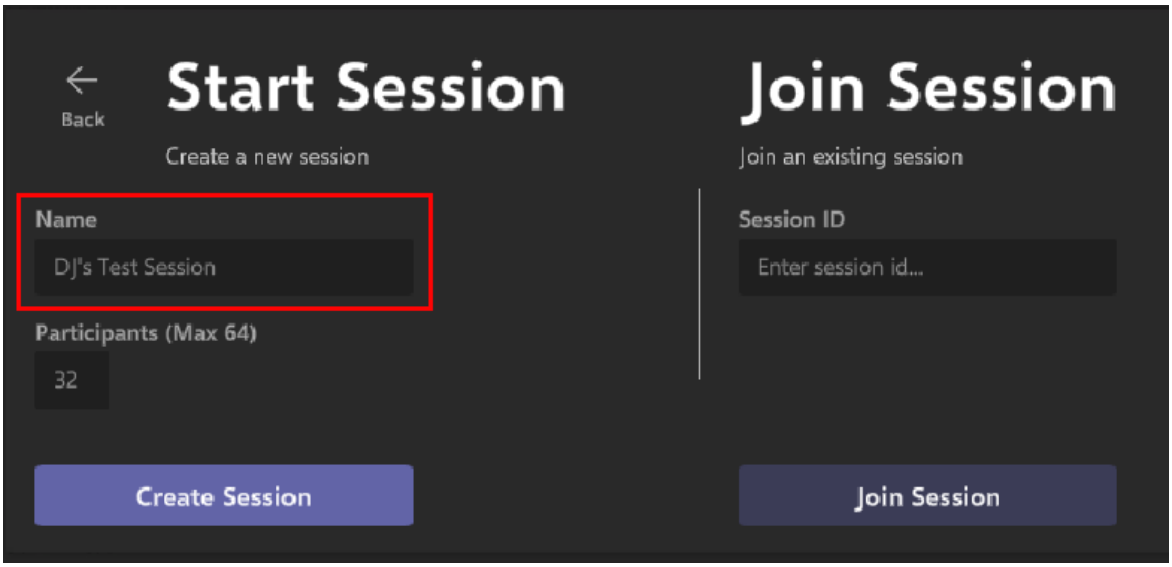


3. In the Unity editor, the project connects to the Mesh Service. After that connection is made, the Mesh UI appears in the **Game** window. It displays a **Status: Connected** message.

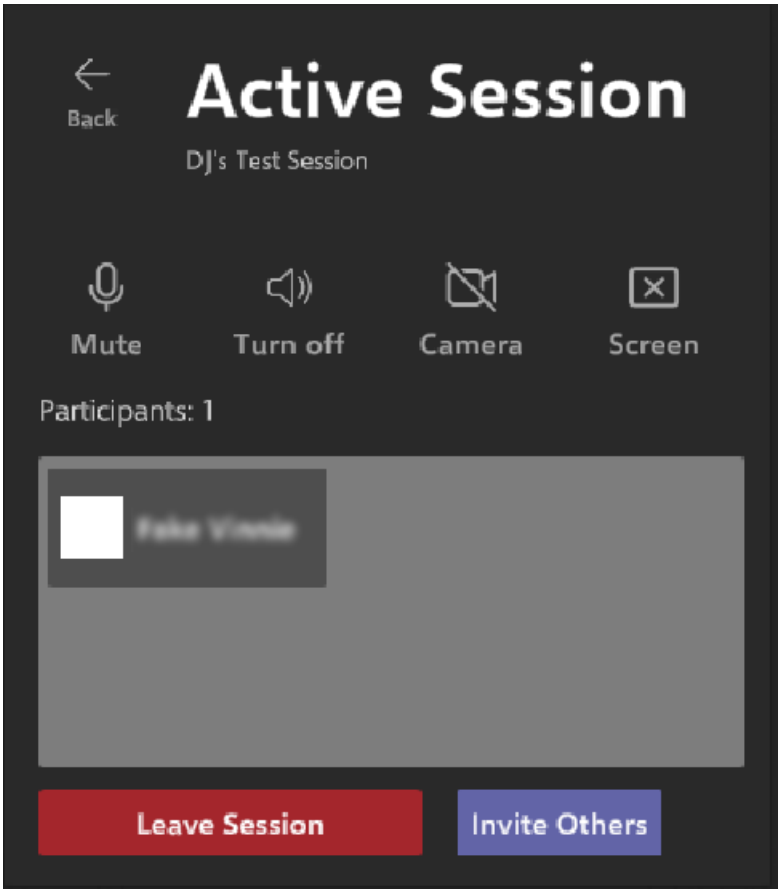


4. Click **Create-Join**.

5. In the **Name** box, type a name for your session.

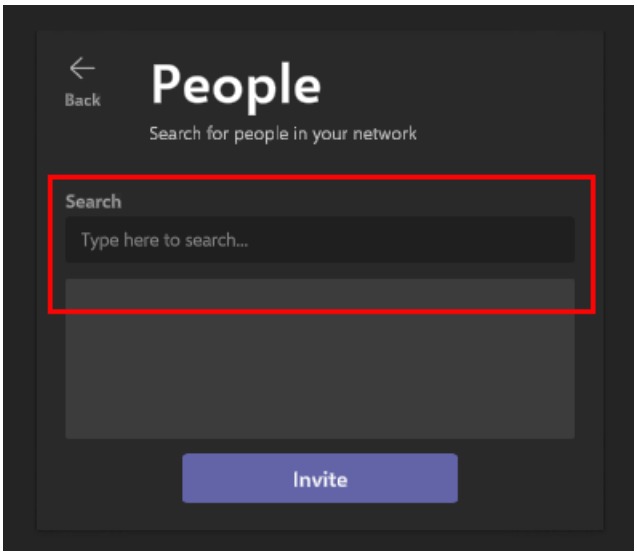


6. Click **Create Session**. The **Active Session** dialog appears and lists you as the session's first participant.



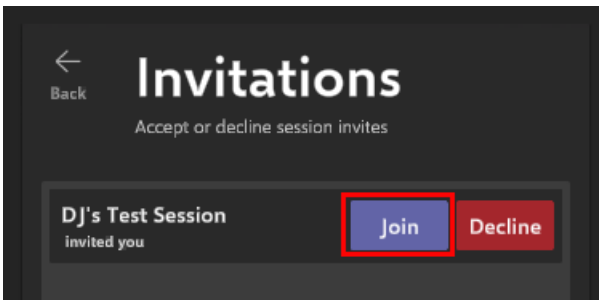
7. Click **Invite Others**. This opens the **People** dialog.

8. In the **Search** box, enter your user name. After it's found, select it, and then select **Invite**.

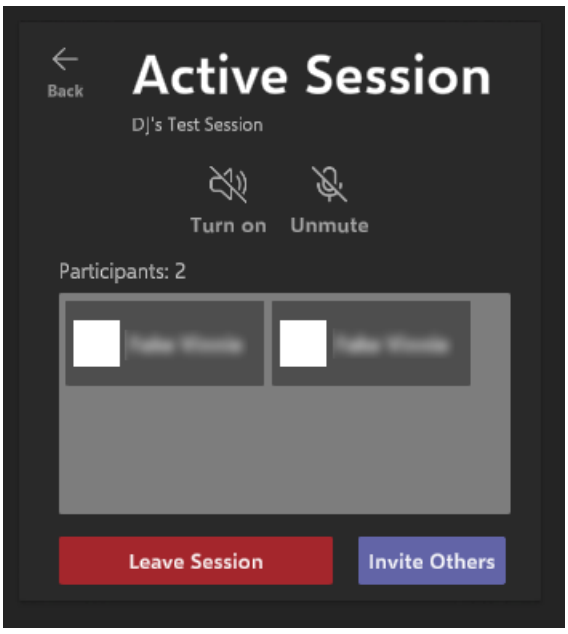


Accepting the invitation in your application

1. Maximize the Unity Player which contains the application that you built earlier. Note that it now contains an **Invitations** button. Click this button.
2. Note that you now see an invitation to join the session that you just initiated in the Unity Editor. Click **Join**.

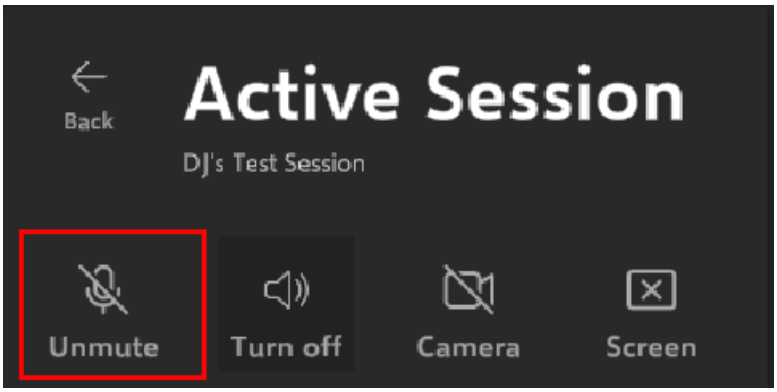


3. If you see a dialog that requests Mesh access to your microphone, click **Yes**.
4. In the Unity Editor, note that your invitation was accepted and that there are now two participants in the session.

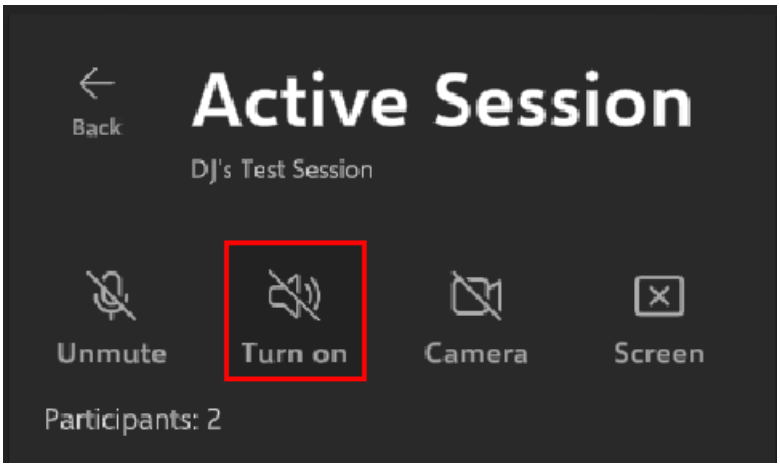


Turning on voice communication

1. In the Unity Player window, to enable the microphone in your application, click the Mute/Unmute button.



2. In the Unity Editor, to enable sound, click the Turn Sound On/Off button.



Congratulations! You now have a Mesh session running with two guests and voice communication.

Next Steps

Explore some of Mesh's core concepts, starting with the Client object.

Client

Quickstart: Create a session and invite a guest in Unity (Windows PC and HoloLens 2)

4/23/2021 • 8 minutes to read • Edit Online

Get started in Mesh by creating a session and inviting participants. This is how you collaborate with your team members on an architectural project, or meet up with friends for a few rounds of virtual Checkers or Go, or host an Augmented Reality birthday party complete with virtual balloons, games, and a clown! In this quickstart, you'll open a Mesh/Unity project and give the project a way to identify itself to Mesh. Next, you'll build an app on the Universal Windows Platform and deploy it to the HoloLens 2. Finally, you'll start a Mesh session in the Unity Editor as User #1, run the app you built and deployed to the HoloLens 2 as User #2, and then invite User #2 to the session. This is the first step in establishing the **presence** of each participant in a session so you can start interacting, collaborating, etc.

If you don't have a HoloLens 2, or you prefer to run the app in the Unity Player instead of the HoloLens 2, see our other other Unity quickstart.

Prerequisites

You will need:

- An Azure account with an active subscription. Create an account for free.
- A Windows computer withUnity 2019.4 (LTS)andVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with registering your tenant.
- If you are working with commercial users, one or more Work or School accounts.
- A HoloLens 2.
- The latest version of the Mesh private preview zip file.

Caution

Caution: When working on Windows, there is a MAX_PATH limit of 255 characters. Unity is affected by these limits and may fail to compile if any file path is longer than 255 characters. Therefore, we strongly recommend that you save your Unity project as close to the root of the drive as possible and keep file and folder names short.

Contents of your private preview zip file

You can find the Unity project in the zip file here:

`PrivatePreviewZip\Samples\Unity\MeshQuickStart`.

There is a PDF version of the Mesh online documentation here:

`PrivatePreviewZip\Documentation`.

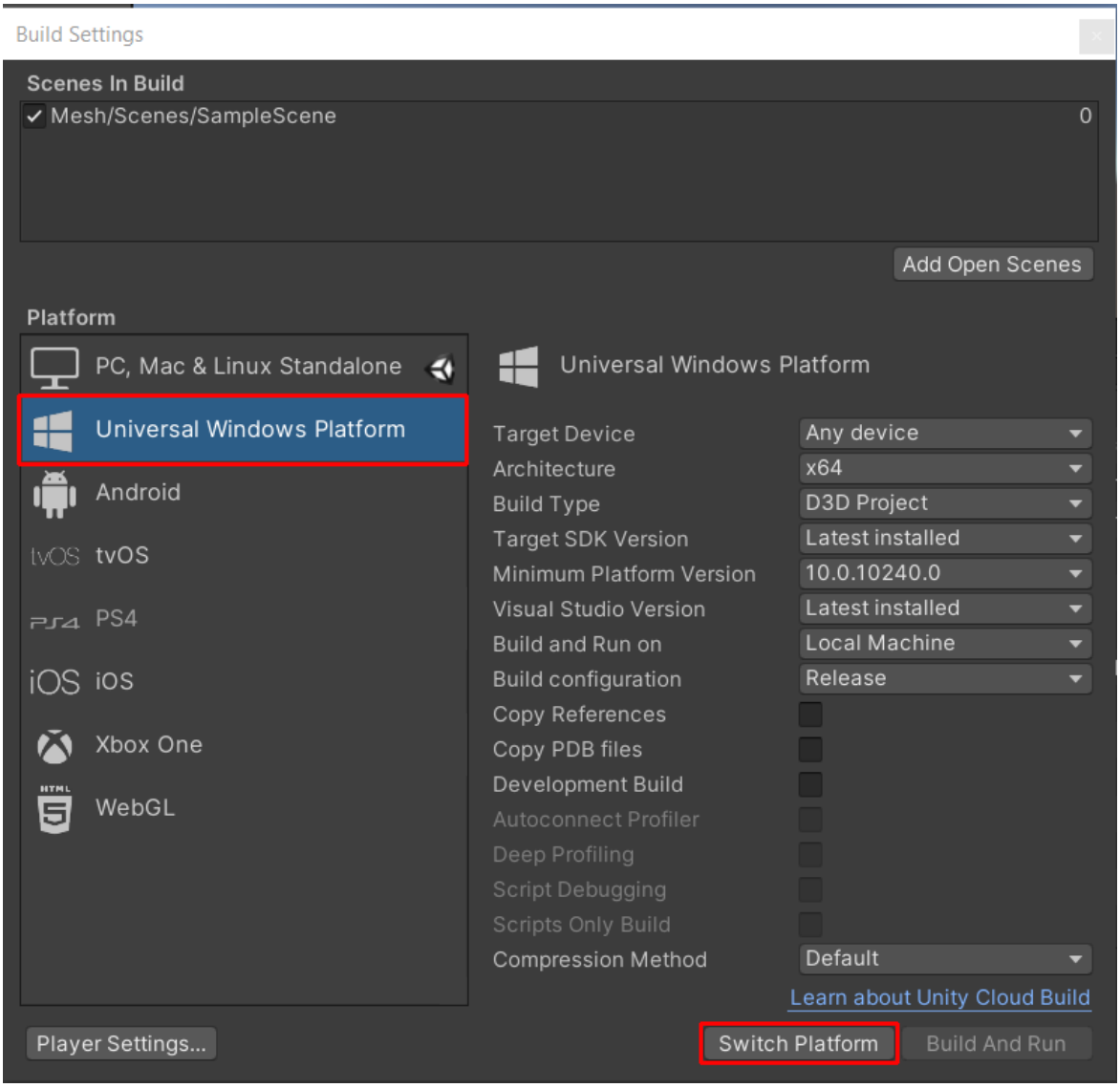
Create a session and invite a guest

Open the sample scene and import TextMeshPro

1. In the Unity Editor, open the sample project.
2. In the **Project** window, navigate to the **Assets** > **Mesh** > **Scenes** folder, and then open the scene **SampleScene**.
3. On the menu bar, select **Window** > **TextMeshPro** > **Import TMP Essential Resources**. This is required for displaying text.
4. In the **Import Unity Package** window, click the **All** button, and then click **Import**.

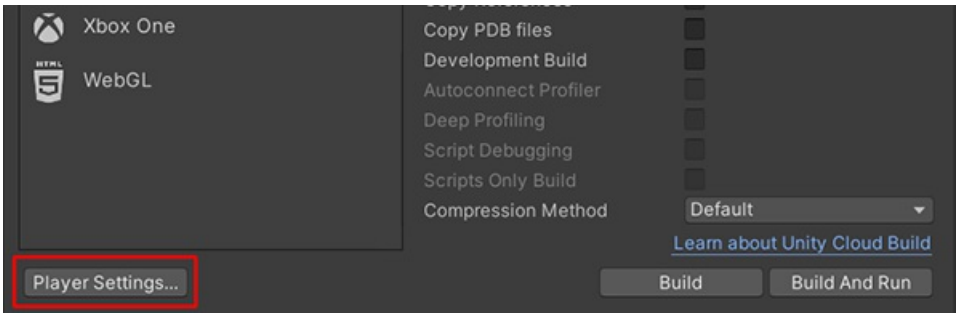
Switch the build platform

1. On the menu bar, select **File** > **Build settings...** .
2. In the **Platform** panel, select **Universal Windows Platform**, and then click **Switch Platform**.

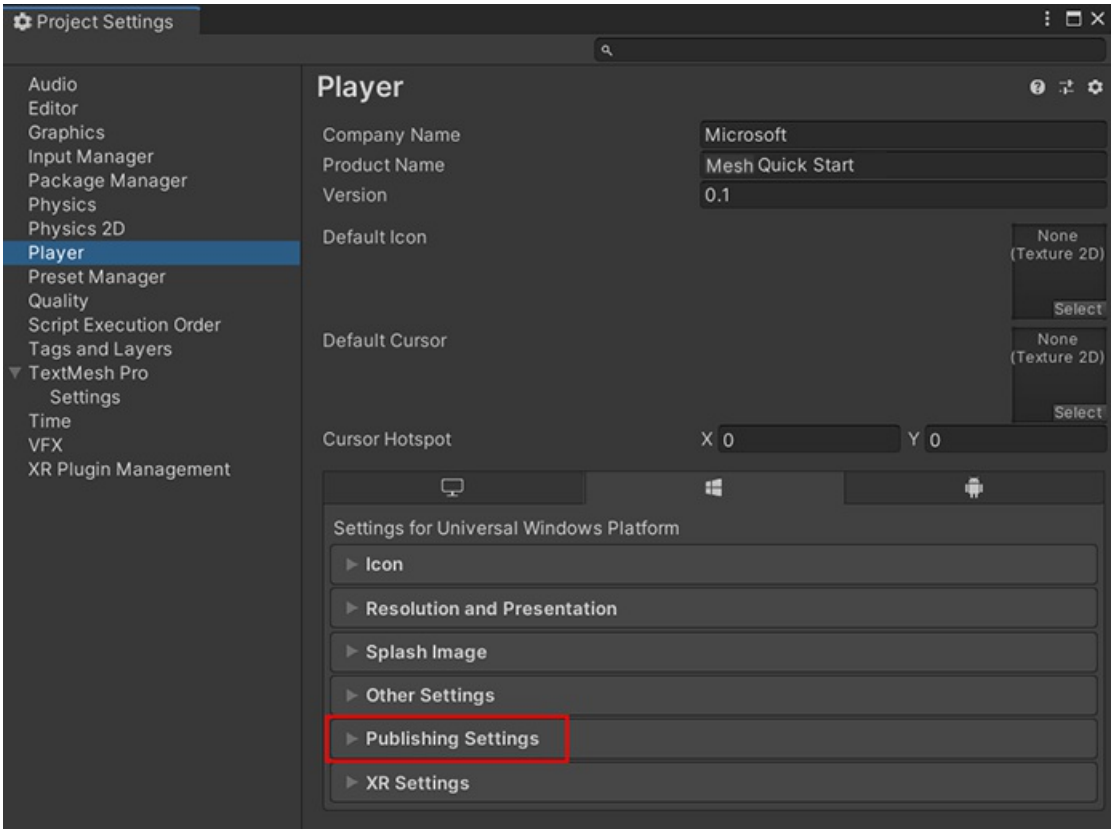


Add capabilities

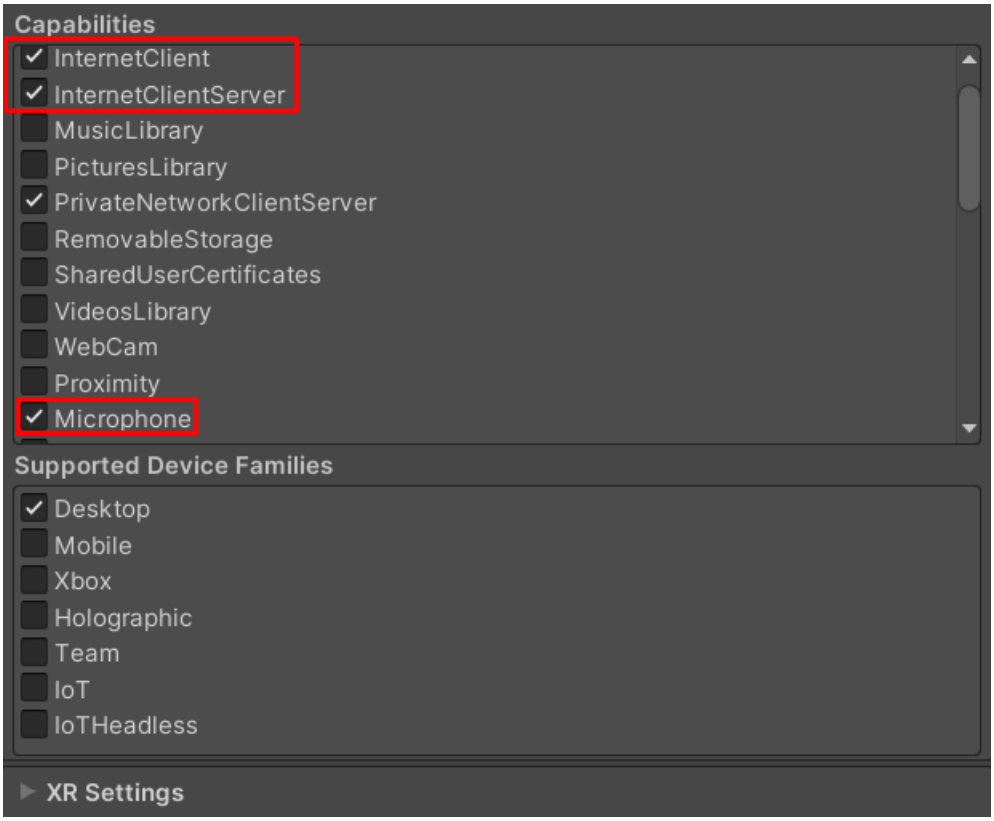
1. Click **Player Settings**.



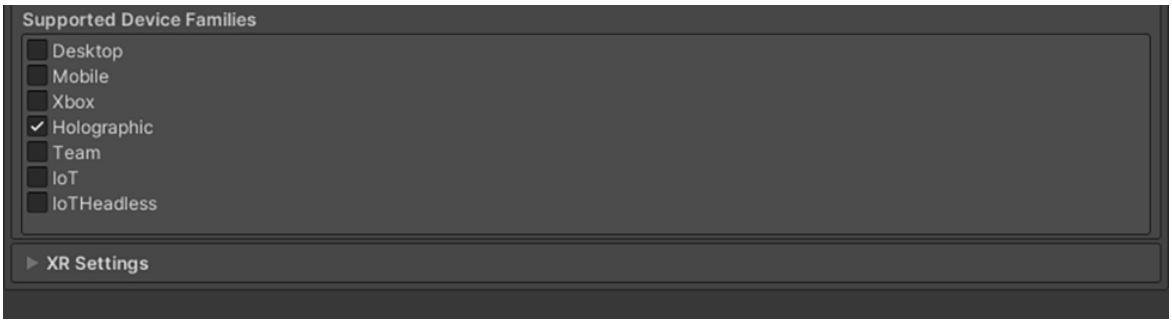
2. In the **Player** panel, click the **Publishing Settings** drop-down.



3. In the **Publishing Settings** section under **Capabilities**, make sure you have **InternetClient**, **InternetClientServer**, and **Microphone** selected.



4. In the **Publish Settings** section under **Supported Device Families**, select **Holographic**.

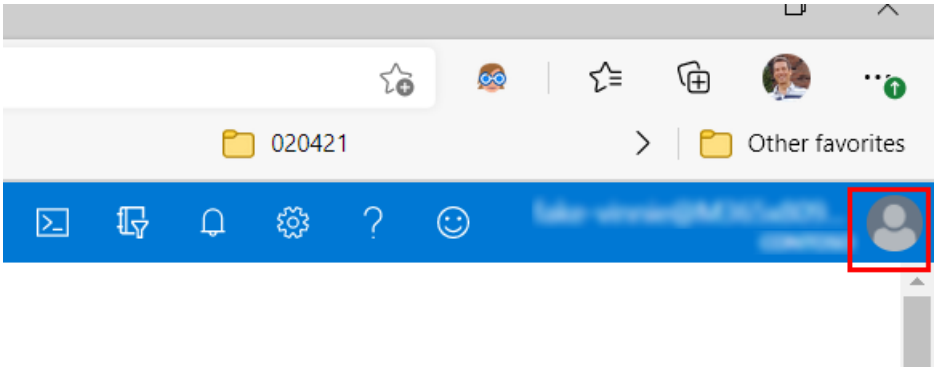


5. Close the **Project Settings** dialog.

Add the client ID and tenant ID to your project

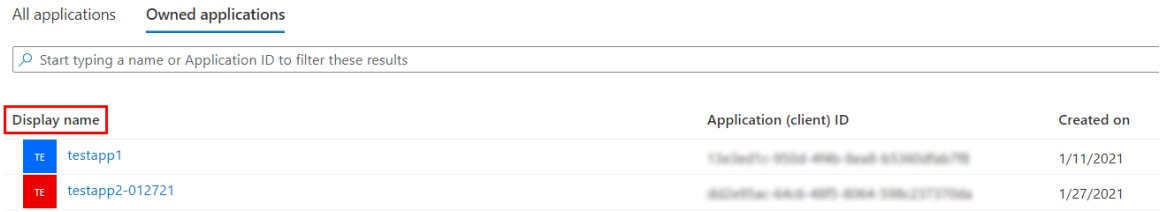
When your application connects to the Mesh service, the service needs a way to identity it. You provide this capability by adding the client ID that was created earlier for your application when you registered it with your tenant, along with your tenant ID.

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.

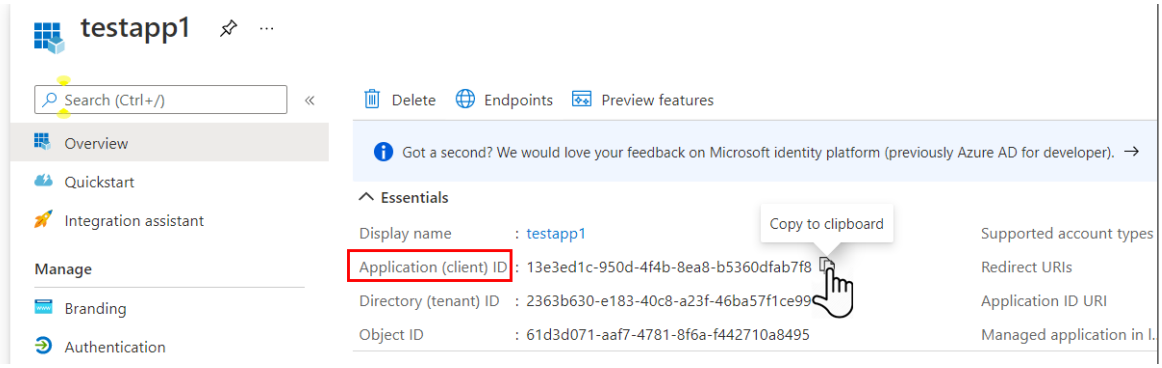


3. Search for and select **Azure Active Directory**. The easiest way is to click the **View** button on the page.

4. In the left-side navigation under **Manage**, select **App registrations**.
5. Under **Display Name**, click the name of your application.

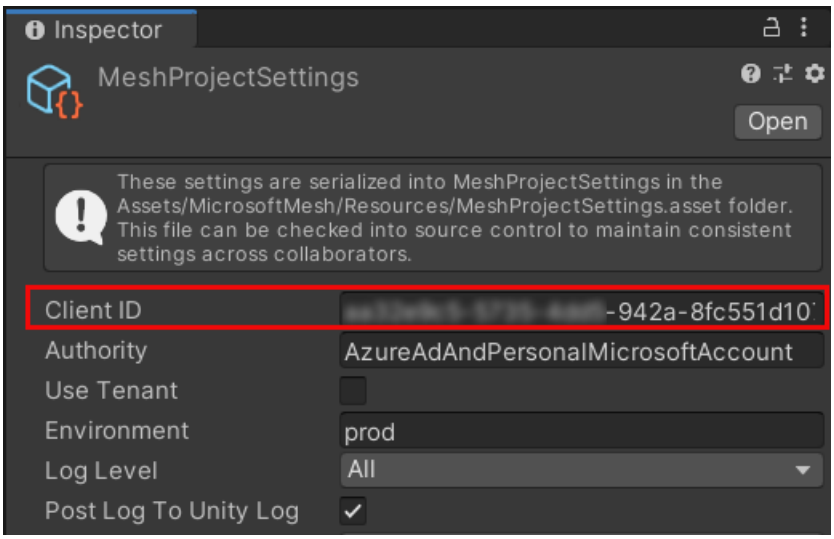


6. In the **Essentials** section, copy the **Application (client ID)** string to the clipboard. To do this, move your cursor just past the end of the ID, and then click the **Copy to clipboard** icon that appears. You can also select the ID and then press Ctrl + C.

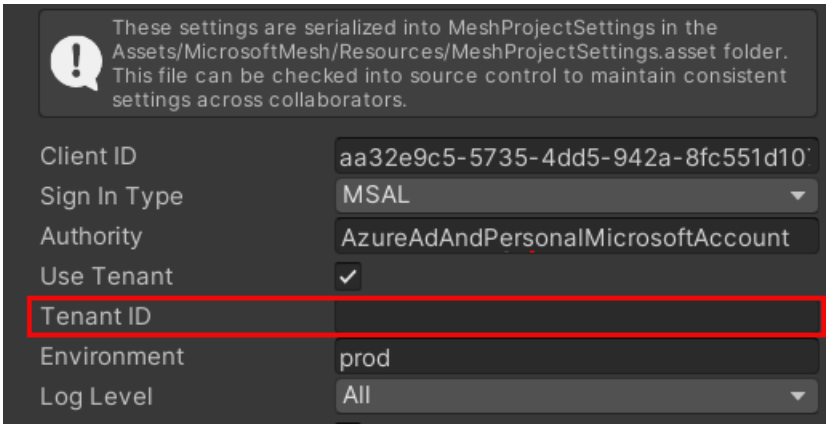


7. In Unity on the menu bar, select **Microsoft Mesh** > **Settings**.

8. In the **Inspector**, paste the client ID into the **Client ID** box.

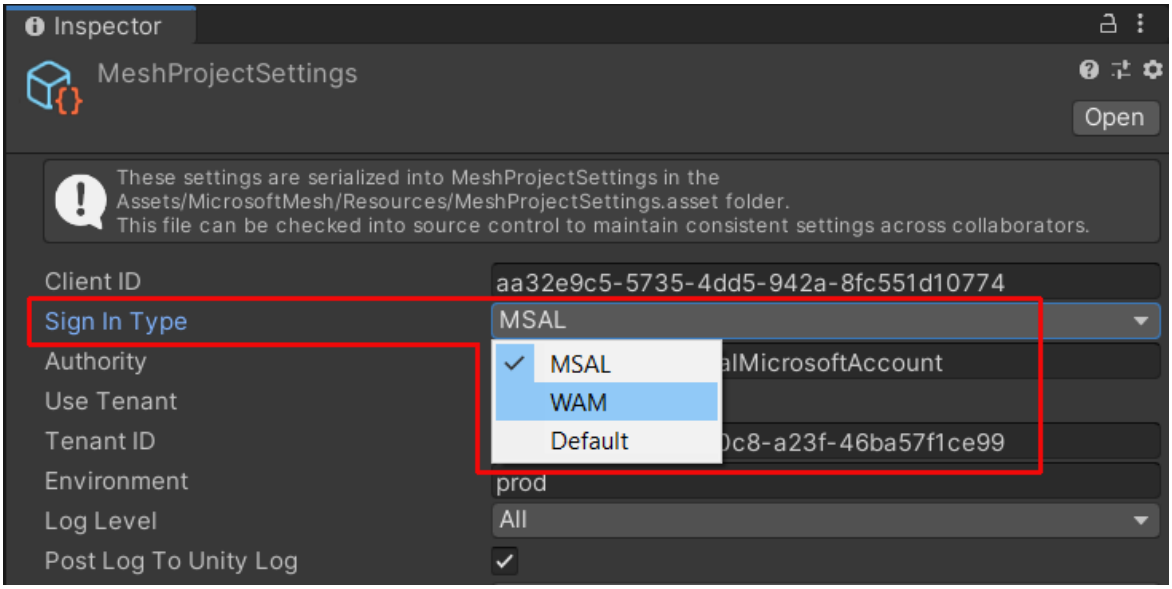


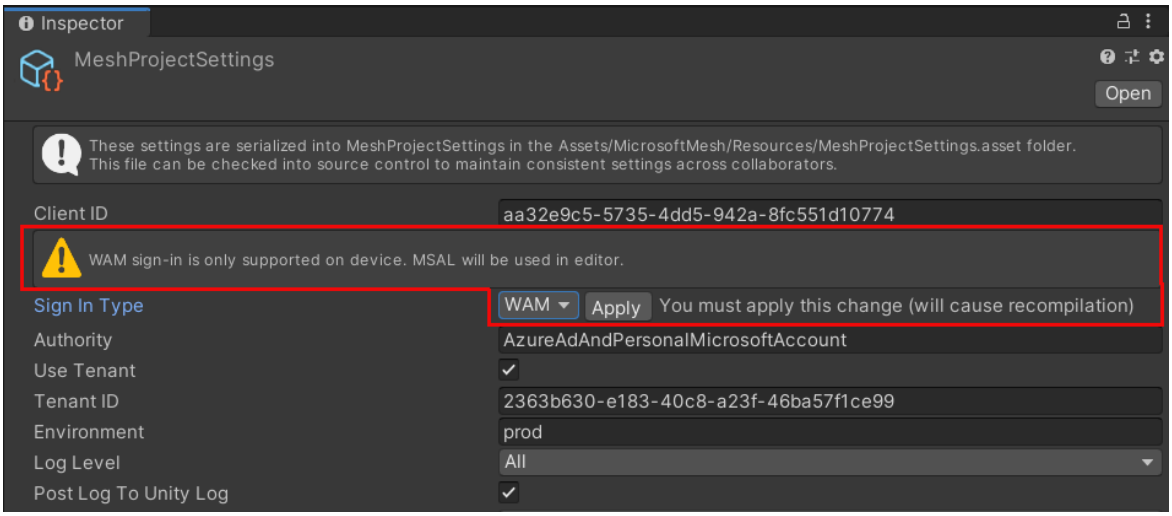
9. Select the **Use Tenant** checkbox. This causes the **Tenant ID** box to appear.



10. The **Sign in Type** is set to **MSAL** by default. Click this drop-down and then choose **WAM**. This is the preferred option when you build and deploy to the HoloLens. To learn more about MSAL and WAM, see

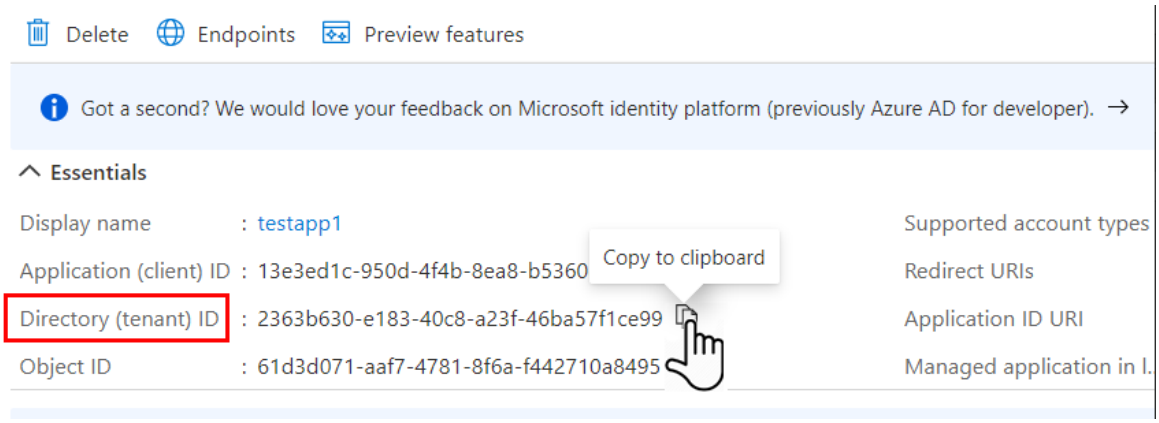
HoloLens 2 Authentication Tips





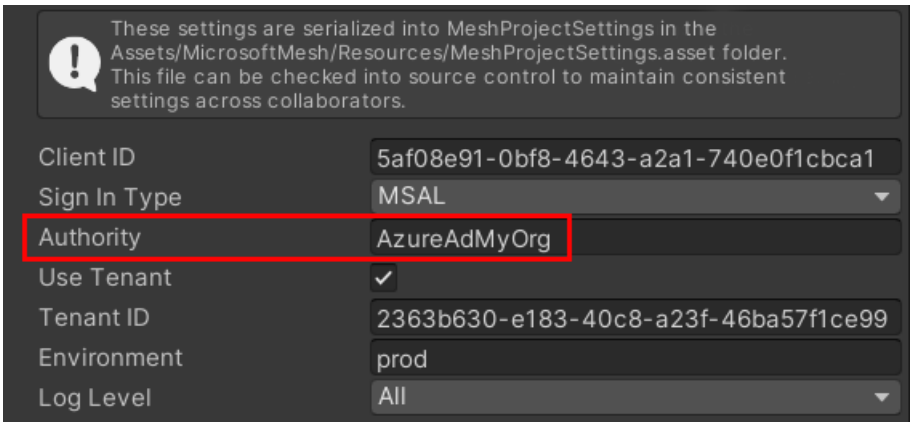
11. To apply the change to WAM, click the **Apply** button.

12. Return to the app registration page in your browser, and then in the **Essentials** section, copy the **Directory (tenant) ID** string to the clipboard. You can do this in the same ways you copied the client ID.



13. In Unity in the **Inspector**, paste the tenant ID into the **Tenant ID** box.

14. In the **Authority** box, enter an authority that matches the account type you selected when you registered your app in the Azure portal.



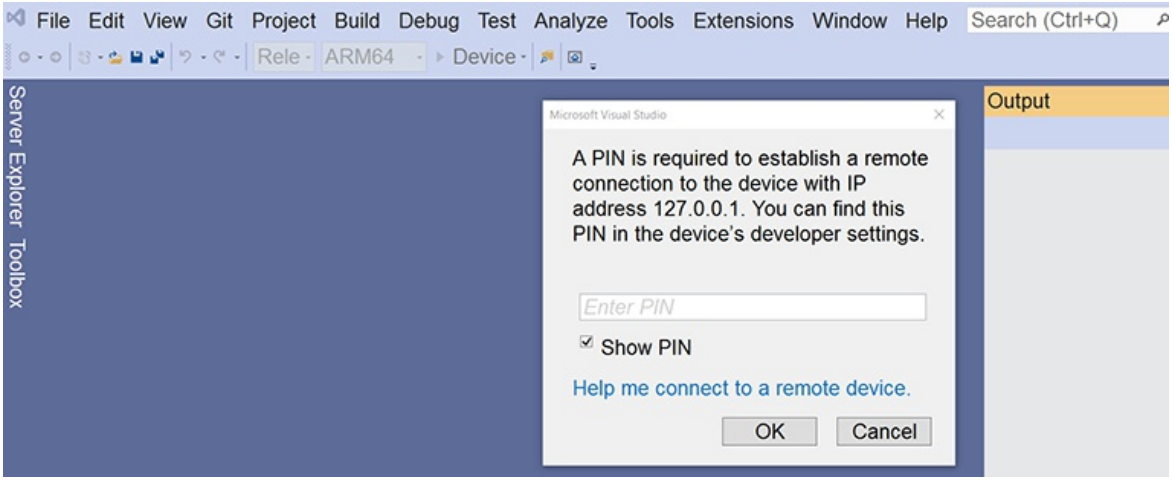
- If you selected **Accounts in the organizational directory only (Mesh Test Tenant only - Single tenant)**, in the **Authority** box, enter "AzureADMyOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**, in the **Authority** box, enter "AzureADMultipleOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)**, in the **Authority** box, enter "AzureADandPersonalMicrosoftAccount"

Building and deploying to your HoloLens 2

Prerequisites

Before building to your device, do the following:

1. Confirm that your device is in Developer Mode.
2. Confirm that your device is paired with your development computer. If it's not, you'll see the following dialog box in Visual Studio during the build process:

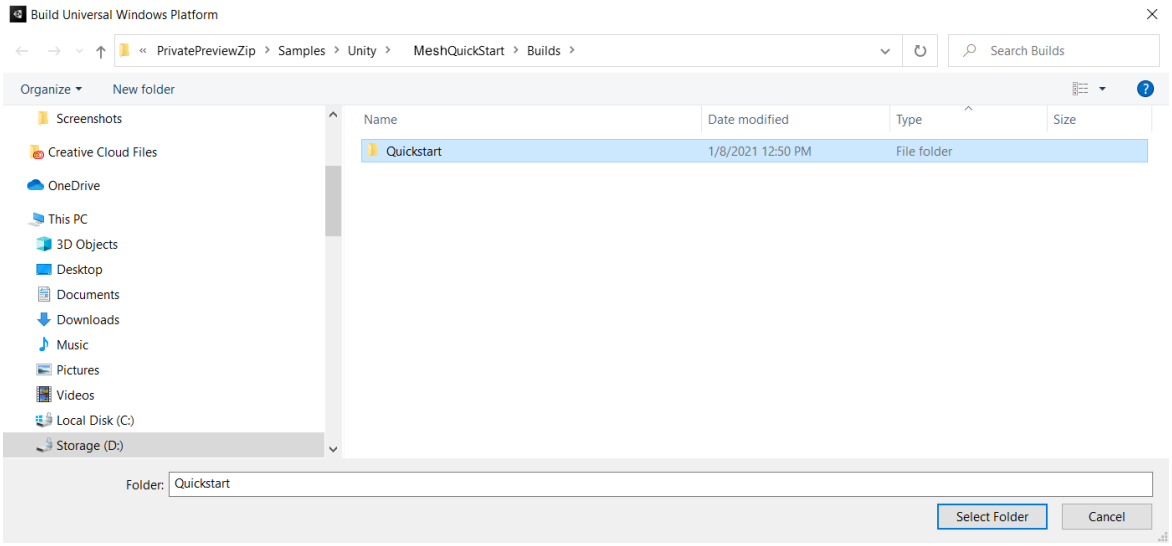


To learn more about these first two steps, see Using Visual Studio to deploy and debug.

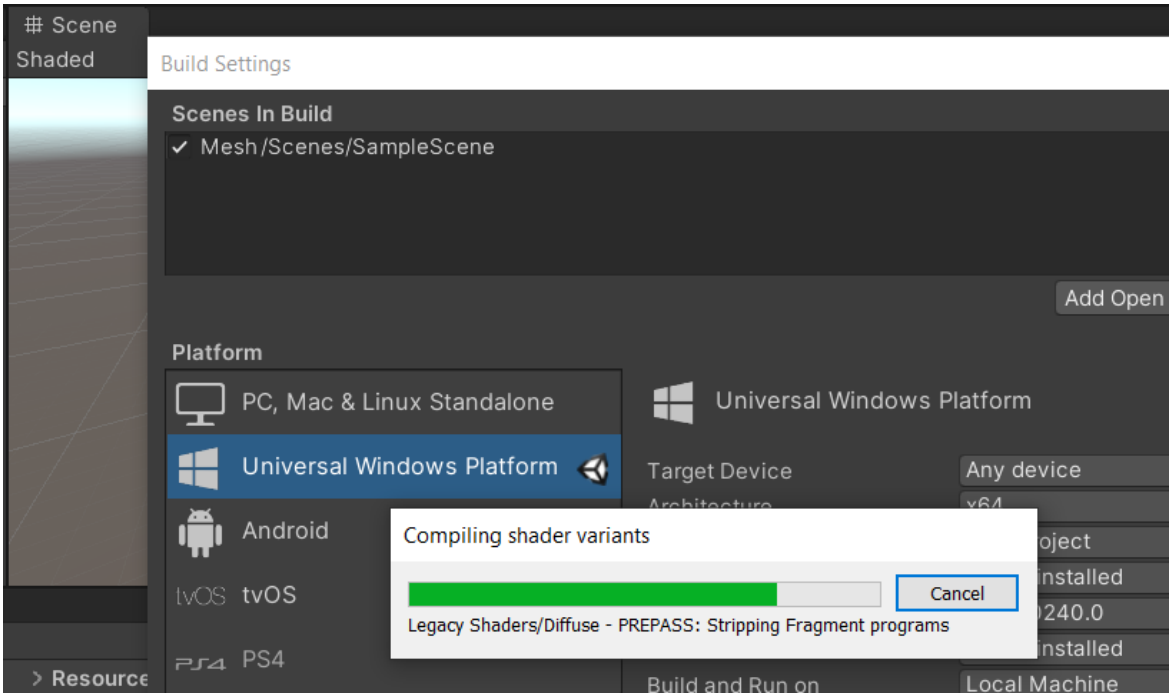
3. Review our HoloLens 2 Authentication Tips

Build your app

1. On the menu bar, select **File** > **Build settings...** .
2. In the **Build Settings** dialog, click **Build**.
3. In the **Build Universal Windows Platform** dialog, choose a suitable location to store your build (for example, you may want to create a "Builds" folder in your project). Create a new folder and give it a suitable name (for example, "Quickstart"), then select the folder, and then click the **Select Folder** button to start the build process.

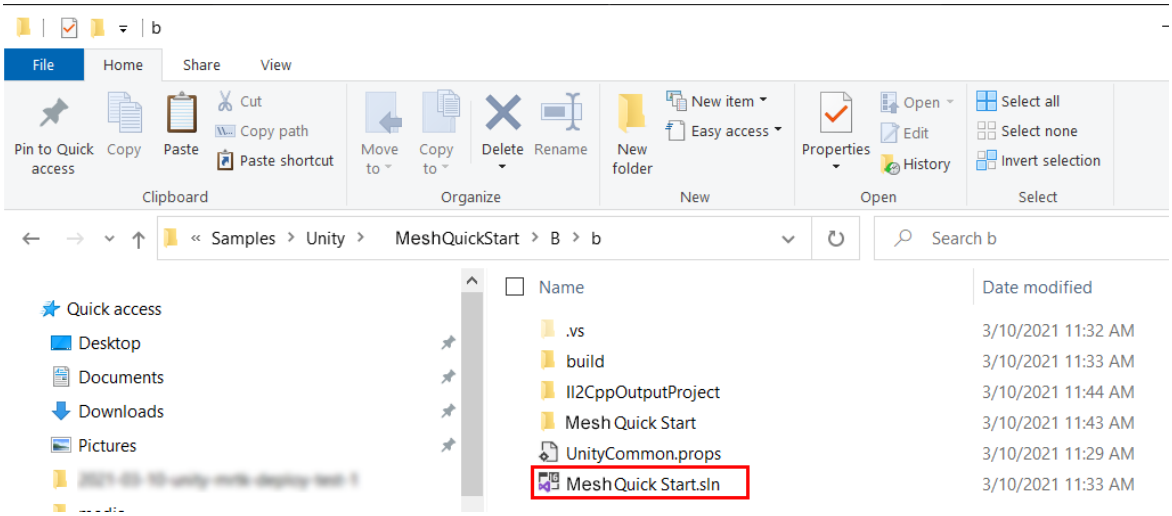


A status bar appears and keeps you updated on your build progress.

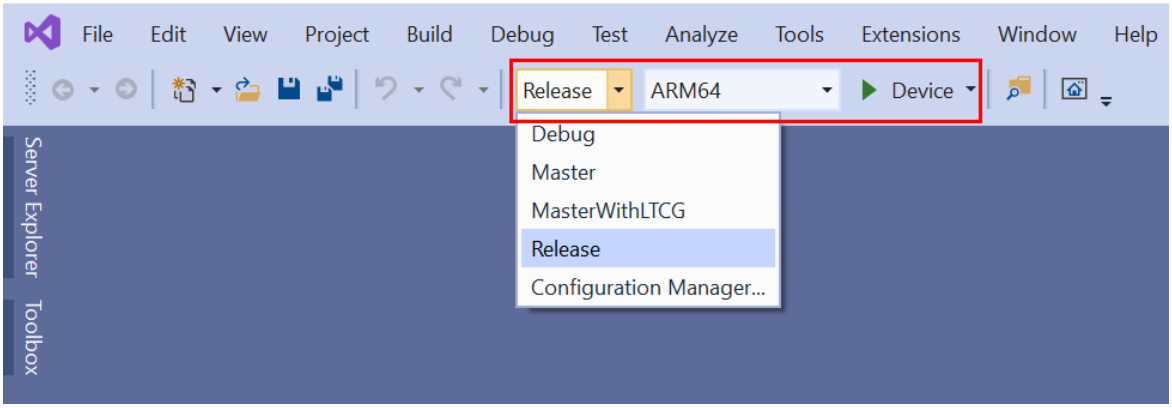


Deploy to the HoloLens 2

1. After the build finishes, in File Explorer, navigate to the location where you stored the build, and then double-click the solution file to open it in Visual Studio:



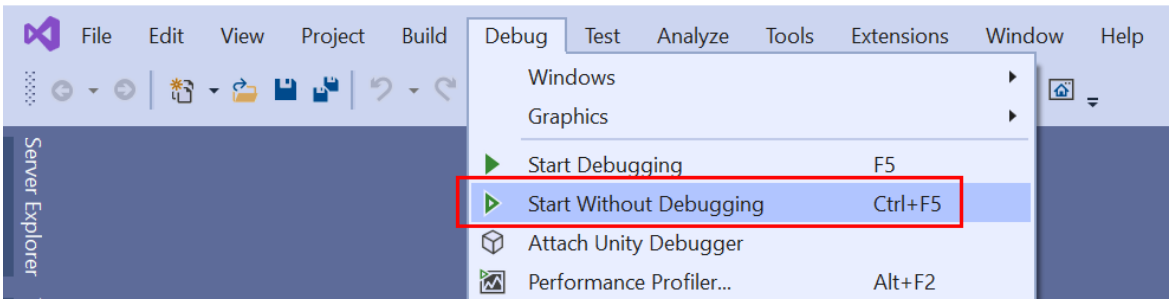
2. Configure Visual Studio for HoloLens 2 by selecting the **Master** or **Release** configuration, the **ARM64** architecture, and **Device** as the target.



> **NOTE**
>
> If you don't see Device as a target option, you may need to change the startup project for the Visual Studio solution from the IL2CPP project to the UWP project. To do this, in the Solution Explorer, right-click on YourProjectName (Universal Windows), and then select **Set as StartUp Project**.

3. Connect your HoloLens 2 to your computer, and then do one of the following:

- To build the app, deploy it to your HoloLens 2, and start it automatically without the Visual Studio debugger attached, on the menu bar, select **Debug** > **Start Without Debugging**.

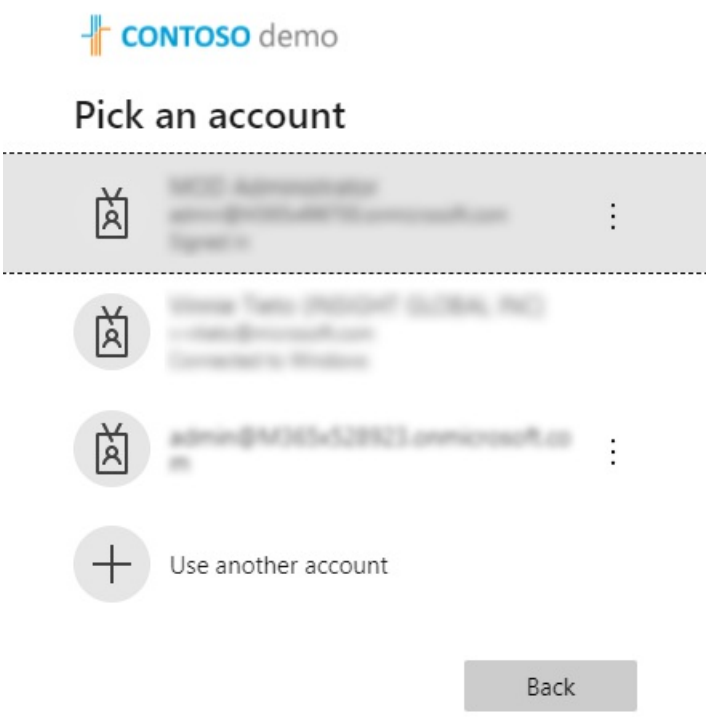


- To build and deploy the app to your HoloLens 2 but not have it start automatically, on the menu bar, select **Build** > **Deploy Solution**.

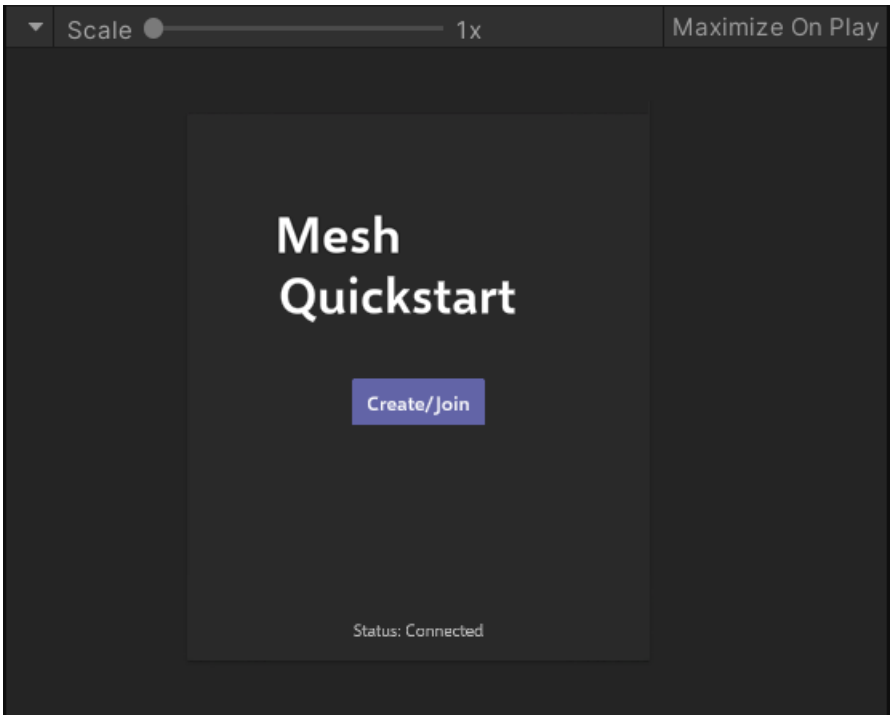
> **NOTE**
>
> You may notice the Diagnostics profiler in the app, which you can toggle on or off by using the speech command **Toggle Diagnostics**. It's recommended that you keep the profiler visible most of the time during development to understand when changes to the app may impact performance. For example, HoloLens 2 apps should continuously run at 60 FPS.

Start a session in the Unity Editor

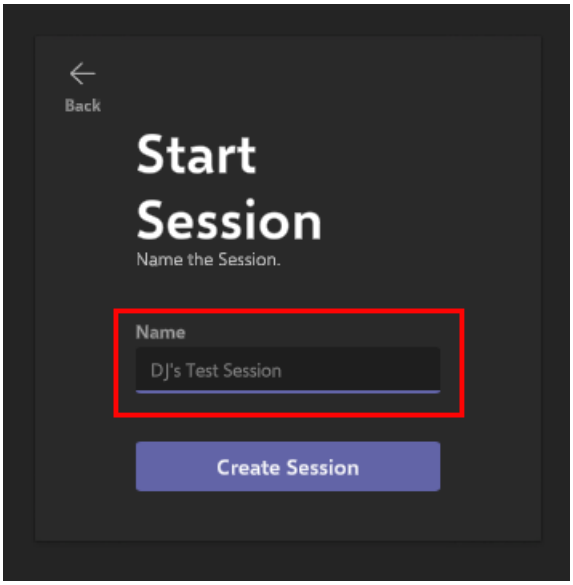
1. In the Unity Editor, close the **Build Settings** dialog.
2. Click the Play button. If the **Pick an account** log-in dialog appears in your browser, supply your user name and password.



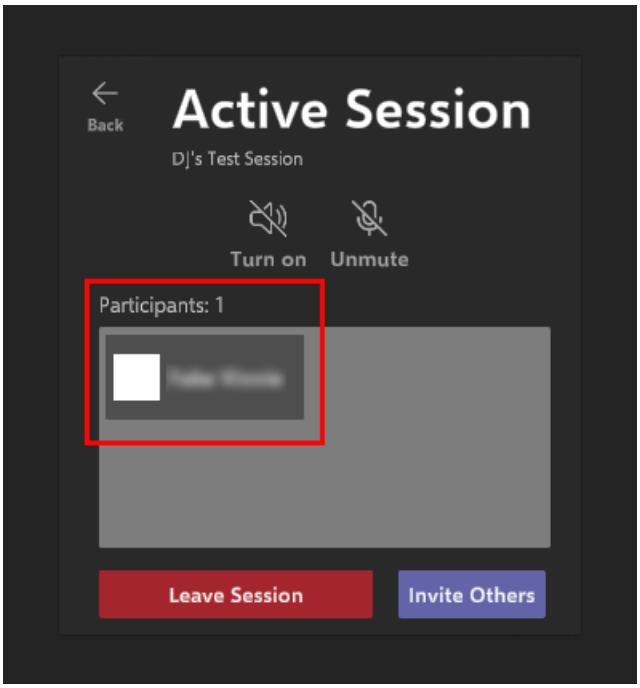
3. The project connects to the Mesh Service. After that connection is made, the Mesh UI appears in the **Game** window. It displays a **Status: Connected** message.



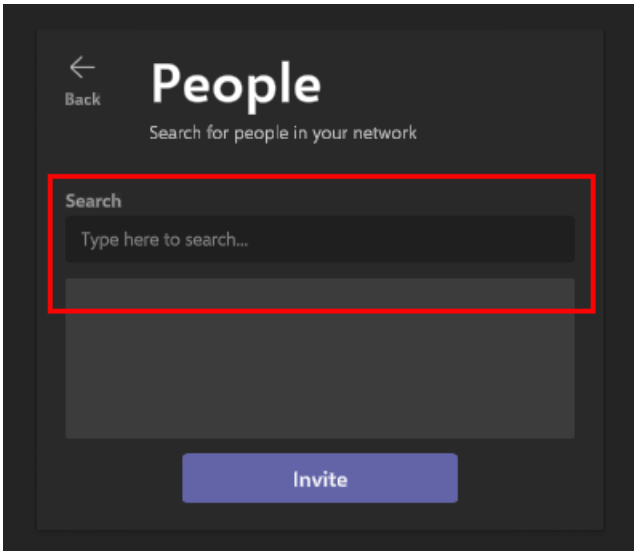
4. Click **New Session**.
5. In the **Name** box, type a name for your session.



6. Click **Create Session**. The **Active Session** dialog appears and lists you as the session's first participant.

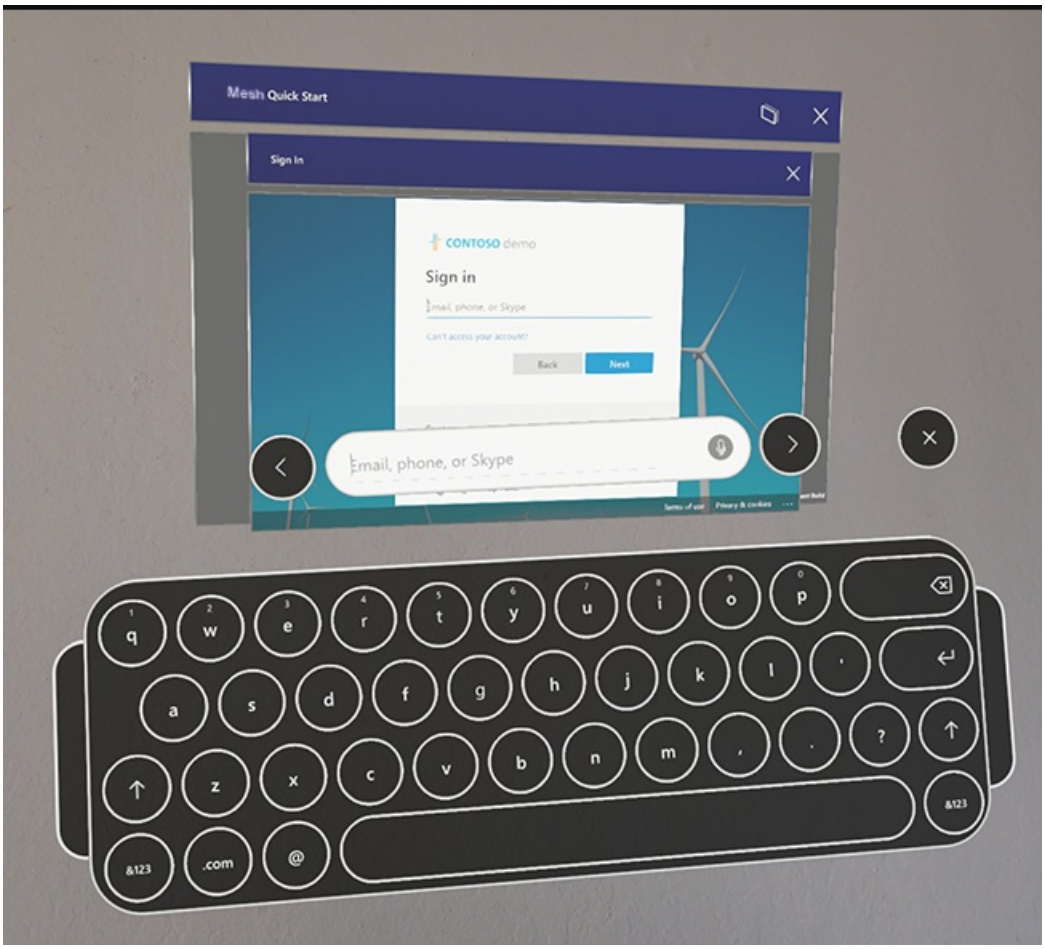


7. Click **Invite Others**. This opens the **People** dialog.
8. In the **Search** box, enter your user name, and then click **Invite**.

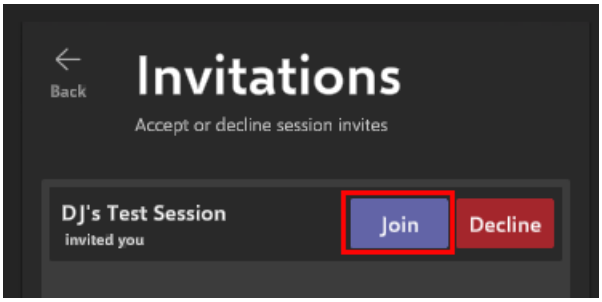


Accepting the invitation in your HoloLens 2 App

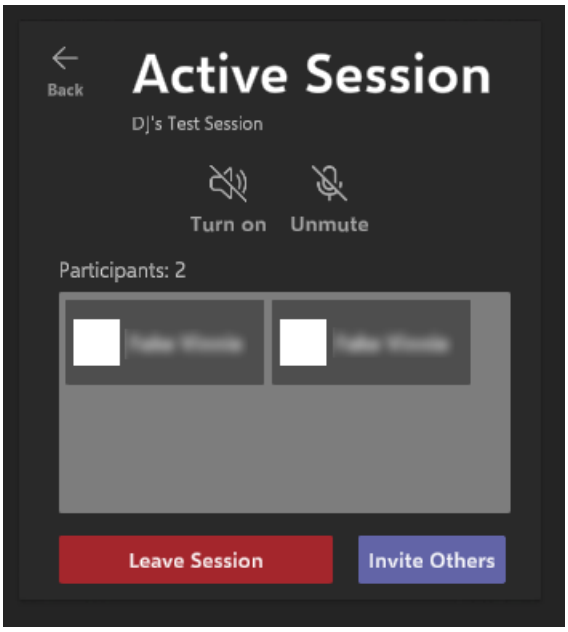
1. In your HoloLens 2, start the app you built earlier if it's not already running. If the **Sign In** log-in dialog appears, supply your user name and password.



2. Click the **Invitations** button.
3. Note that you now see an invitation to join the session that you just initiated in the Unity Editor. Click **Join**.

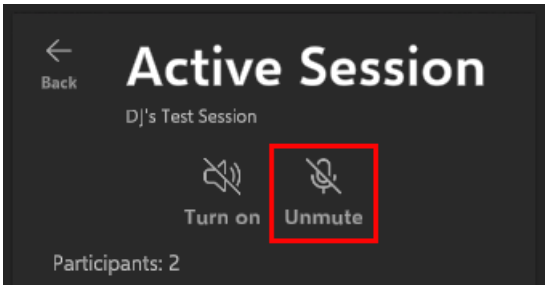


4. If you see a dialog that requests Mesh access to your microphone, click **Yes**.
5. In the Unity Editor, note that your invitation was accepted and that there are now two participants in the session.

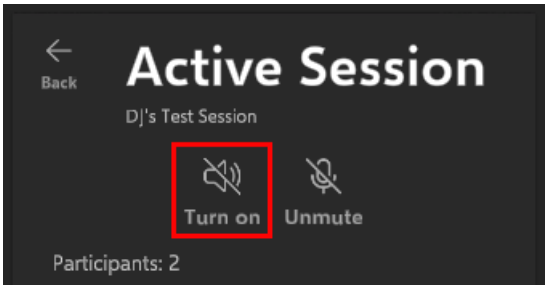


Turning on voice communication

1. In the HoloLens 2, to enable the microphone in your application, click the Mute/Unmute button.



2. In the Unity Editor, to enable sound, click the Turn Sound On/Off button.



Congratulations! You now have a Mesh session running with two guests and voice communication.

Next Steps

Explore some of Mesh's core concepts, starting with the Client object.

Client

Quickstart: Create a session and invite a guest using C++

3/31/2021 • 3 minutes to read • Edit Online

Get started in Mesh by creating a session and inviting participants. This is how you collaborate with your team members on an architectural project, or meet up with friends for a few rounds of virtual Checkers or Go, or host an Augmented Reality birthday party complete with virtual balloons, games, and a clown! The MR Collaboration app is a Visual Studio project that gives you a sample of the C++ APIs. In this quickstart, you'll launch two instances of the app, then authenticate in both, and then start a session between the two instances. This is the first step in establishing the **presence** of each participant in a session so you can start interacting, collaborating, etc.

Prerequisites

You will need:

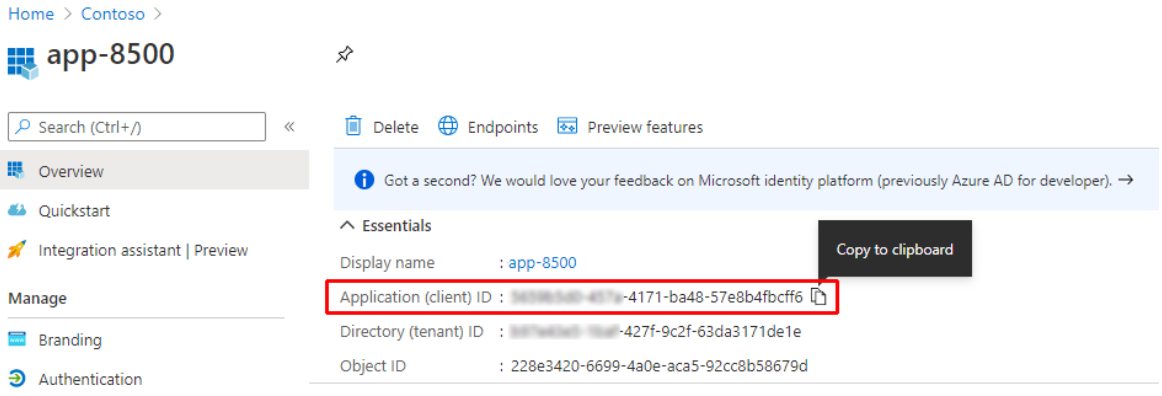
- An Azure account with an active subscription. Create an account for free.
- A Windows computer withVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with registering your tenant.
- If you are working with commercial users, one or more Work or School accounts.

You can find the MR Collaboration project in the zip file provided in your private preview drop under `Samples\Cpp\MeshQuickStart\MeshQuickStart.sln`.

Find and copy the client ID

When your application connects to the Mesh service, the service needs a way to identity it. You provide this capability by adding the client ID that was created earlier for your application when you registered it with your tenant.

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.
3. Search for and select **Azure Active Directory**. In the left-side navigation under **Manage**, select **App registrations**.
4. Under **Display Name**, click the name of your application.
5. In the **Essentials** section, copy your client ID to the clipboard.



Add the client ID to your project

1. In Visual Studio, open the solution file.
2. Open `ClientAppInfo.h`. You'll see the line:

```
const wchar_t* ClientId = L"==specify client id here==";
```

3. Replace `==specify client id here==` with the app's client ID which you copied in the previous section.

Deploy the application

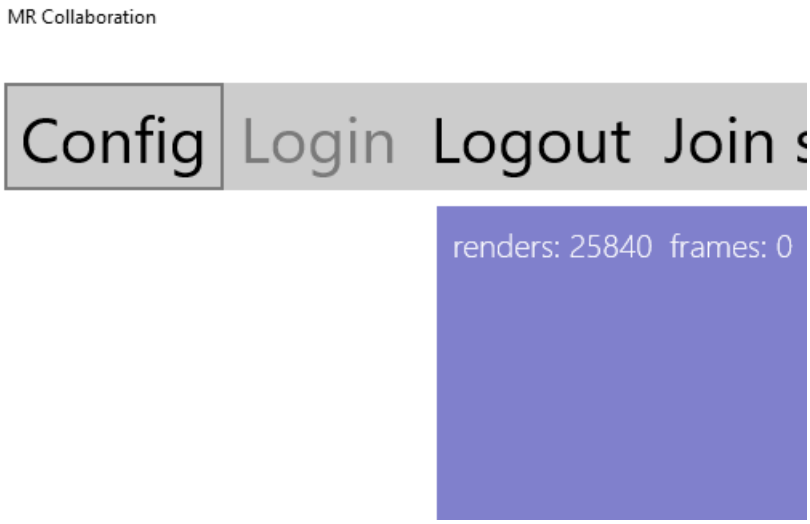
1. In the Solutions Configurations drop down, select **Release**. In the Solution Platforms drop down, select **x64**.
2. On the **Build** menu, select **Deploy Solution**.
3. After the build completes, in the Start menu, navigate to the **MR Collaboration** application and then open it.

Configure the application Redirect URIs

The reply URI tells our authentication process where to send tokens after successfully authenticating users. This URI will be different for each developer machine unless you use the same signing key, such as when the project is associated to the store. The MR Collab app will generate the URI that you can then use on the developer portal.

To determine the application's reply URL:

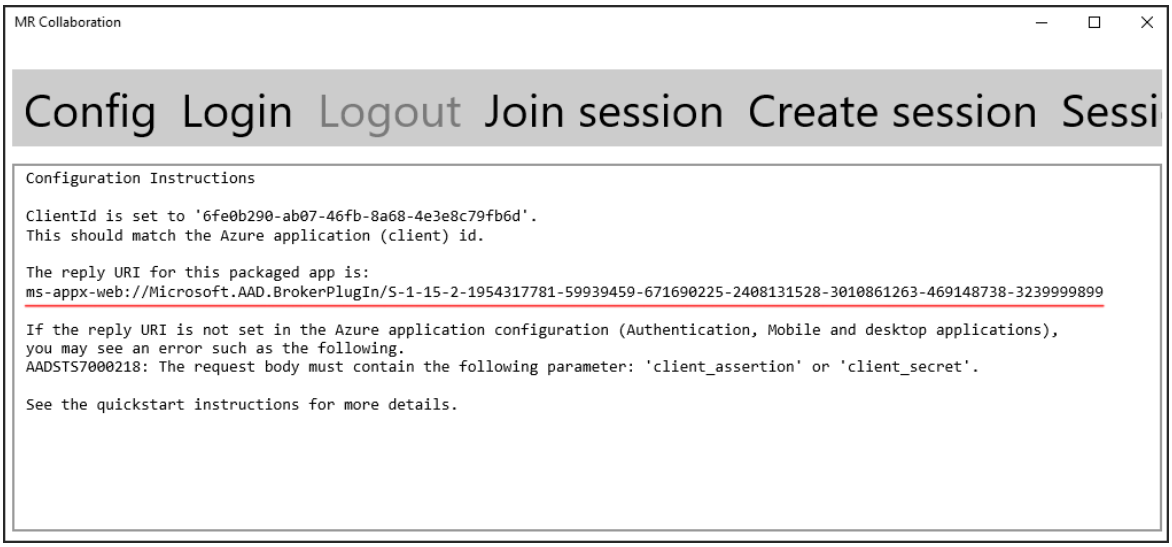
1. In the MR Collaboration app, select the **Config** menu.



2. Look for the line that says:

The reply URI for this packaged app is:

Copy the next line--this is the URI.



3. Navigate to the Azure portal and find your application registration. On the Overview page, under the **Manage** menu, select **Authentication**.
4. Select **Add a Platform** and make sure **Mobile and desktop applications** is selected.
5. Under **Redirect URIs**, select **Add URI**. Paste the value from the MR Collaboration app into the box.
6. Select **Save**.

Communicate with the Mesh Services

1. Launch two instances of the MR Collaboration application. (We'll call them Instance A and Instance B.)
2. In each instance, select **Login**. Sign in with one of the users in your tenant.
3. In Instance A, select **Create Session**. Wait for the application to create the session, and then select **Invite User**.
4. Invite your user to a session.
5. In Instance B, accept the invitation.

If you have more than one developer working on the system, you can provide the same end-user experience by running the app from Visual Studio in debug mode and then calling the other developer from your debug build.

Troubleshooting problems

- Troubleshooting common error messages

Next Steps

Explore some of Mesh's core concepts, starting with the Client object.

Client

Explore a pre-configured Mesh tools scene

4/26/2021 • 6 minutes to read • Edit Online

In this quickstart, you'll set Unity up to run the Mesh tools for Unity, and then open the `ToolkitDemo` scene, which gives you a complete user experience (UX) that addresses common needs for collaboration. A number of tools are already pre-configured, so you can press Play and quickly get into a collaborative space. There are also instructions for building and deploying the project to the HoloLens.

Prerequisites

You will need:

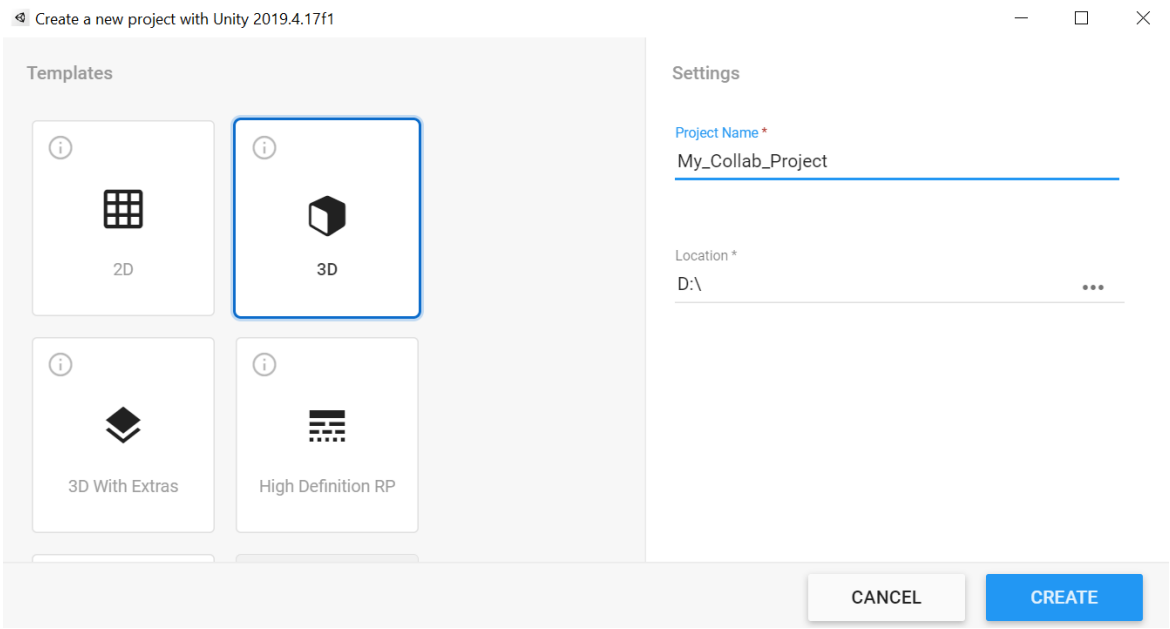
- A Windows computer withVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure account with an active subscription. Create an account for free.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with registering your tenant. You'll need a Client ID and Tenant ID for this tutorial.
- If you are working with commercial users, one or more Work or School accounts.
- A Unity 2019.4.x installation.
- The Mixed Reality Feature tool for Unity. This page explains how to download and use the tool to import packages.
- The latest version of the Mesh private preview zip file.

  Caution

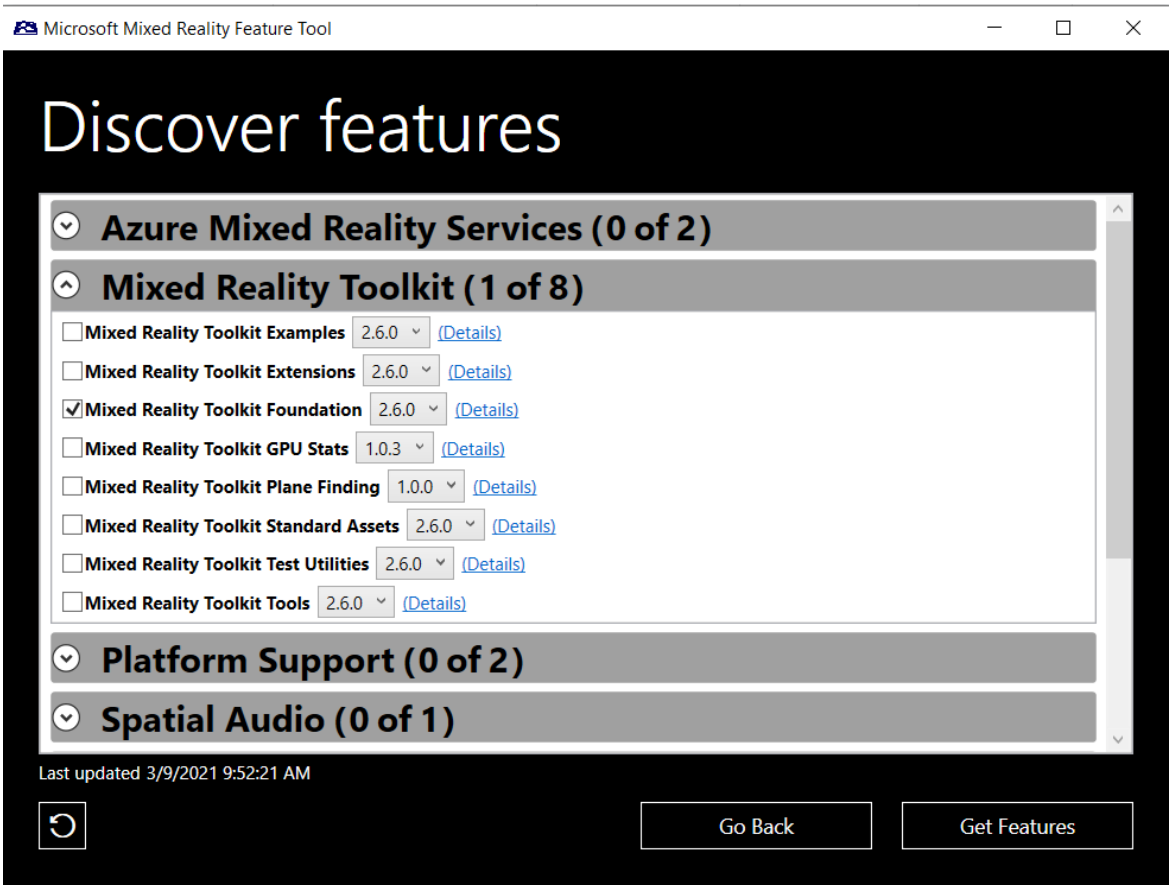
  Caution: When working on Windows, there is a MAX_PATH limit of 255 characters. Unity is affected by these limits and may fail to compile if any file path is longer than 255 characters. Therefore, we strongly recommend that you save your Unity project as close to the root of the drive as possible and keep file and folder names short.

Set up and run a new project

1. Create a new Unity project.



2. Follow the instructions on the Mixed Reality Feature Tool page to import a Mixed Reality package into your project. In this case, we want the **Mixed Reality Toolkit Foundation 2.6.x** package.



3. Unzip the private preview zip file, and then copy the unzipped folders **Documentation** and **Packages** and paste them into the root folder of your new project.

4. In Unity, on the **Project** tab, navigate to this folder:

   ```
   Packages\Mesh Tools for Unity (UX)\_Shared\Scenes\
   ```

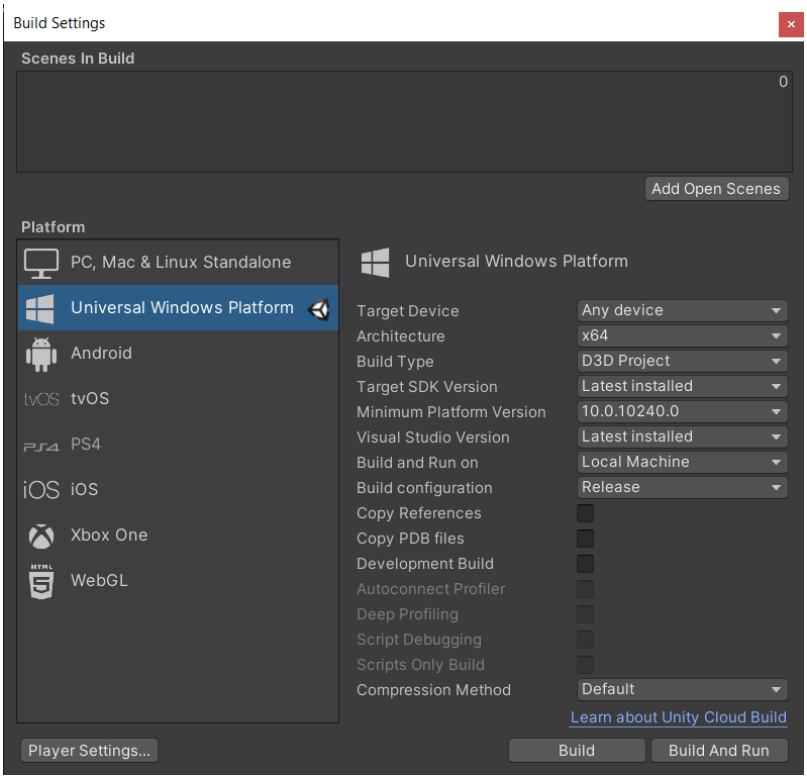
   ... and then open the **ToolkitDemo** scene.

   This causes the **TMP Importer** dialog to appear. In the dialog, click the **Import TMP Essentials** button. After the import is finished, close the dialog.

> **NOTE**
>
> Differences in TextMeshPro versions may cause your text to appear very large in the scene. Restart Unity and re-open the project if necessary.

Switch the build platform

1. In the menu bar, select **File** > **Build Settings**.
2. In the Build Settings window, select **Universal Windows Platform** and then click the **Switch Platform** button. Unity will begin the process to switch the platform.

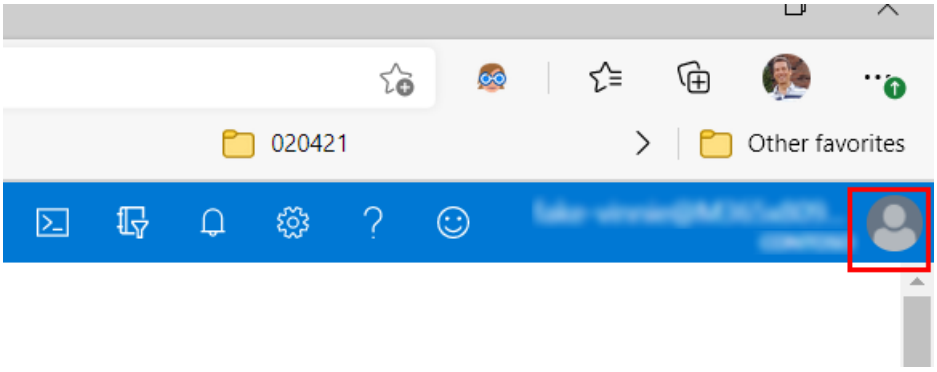


3. Close the **Build Settings** window.

Add the client ID and tenant ID to your project

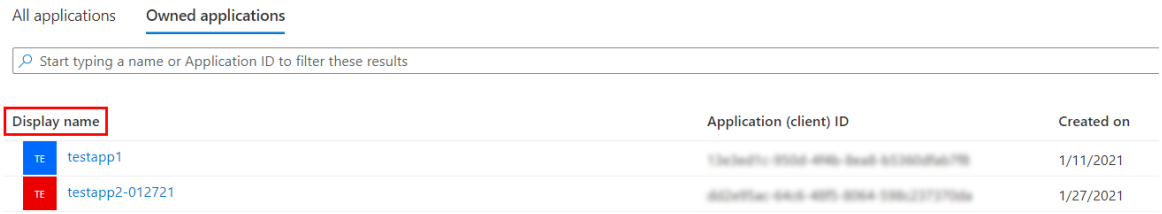
When your application connects to the Mesh service, the service needs a way to identity it. You provide this capability by adding the client ID that was created earlier for your application when you registered it with your tenant, along with your tenant ID.

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.

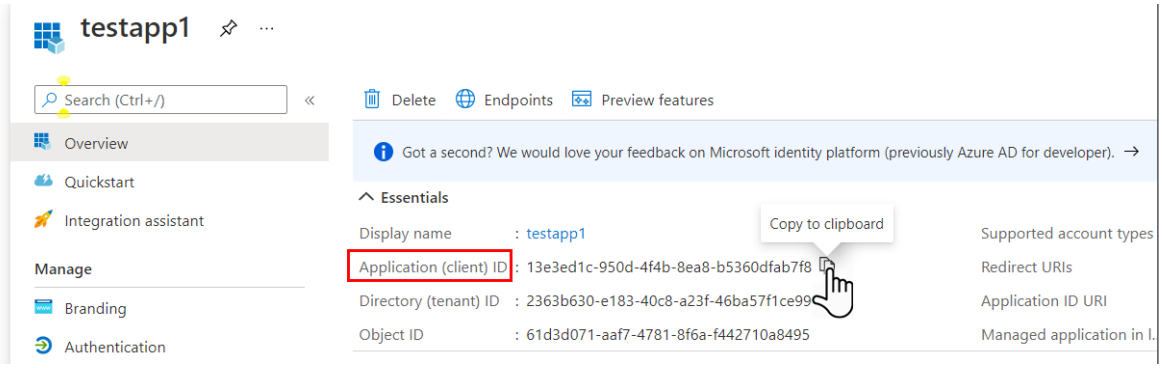


3. Search for and select **Azure Active Directory**. The easiest way is to click the **View** button on the page.

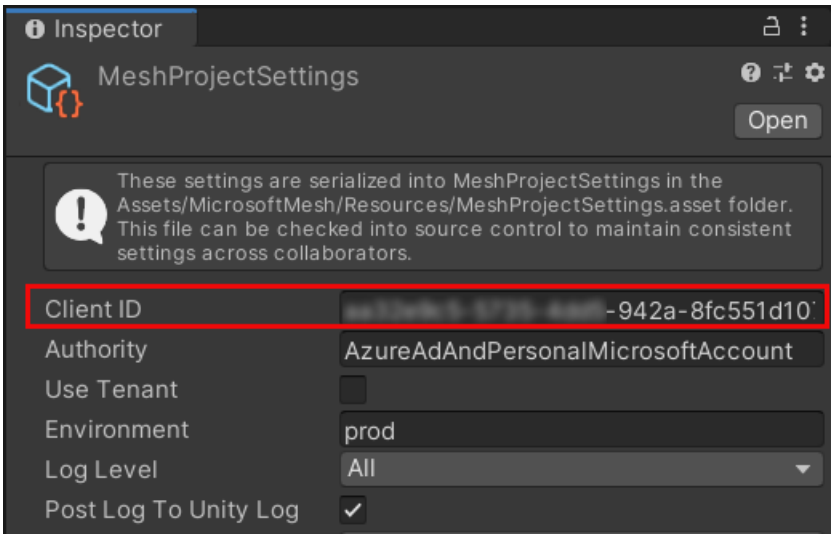
4. In the left-side navigation under **Manage**, select **App registrations**.
5. Under **Display Name**, click the name of your application.



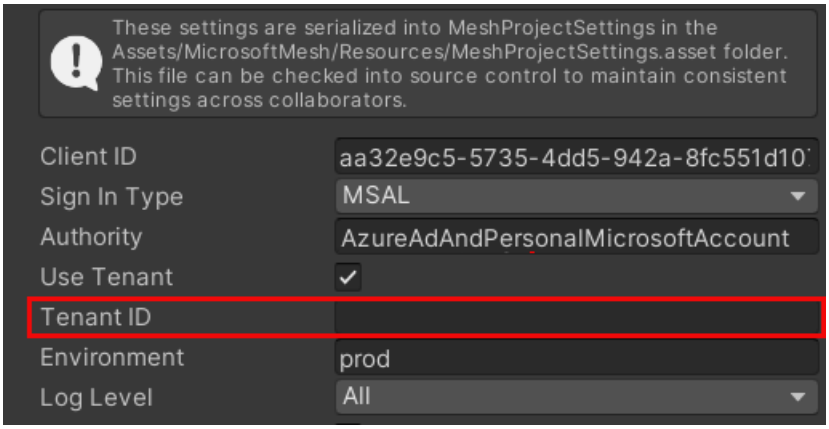
6. In the **Essentials** section, copy the **Application (client ID)** string to the clipboard. To do this, move your cursor just past the end of the ID, and then click the **Copy to clipboard** icon that appears. You can also select the ID and then press Ctrl + C.



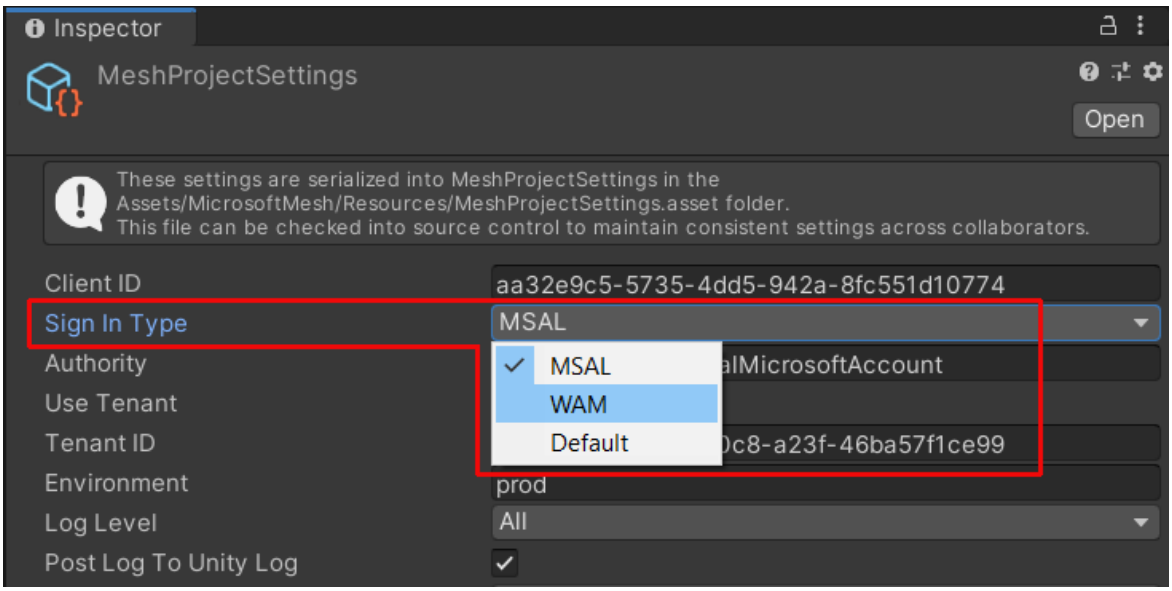
7. In Unity on the menu bar, select **Microsoft Mesh** > **Settings**.
8. In the **Inspector**, paste the client ID into the **Client ID** box.

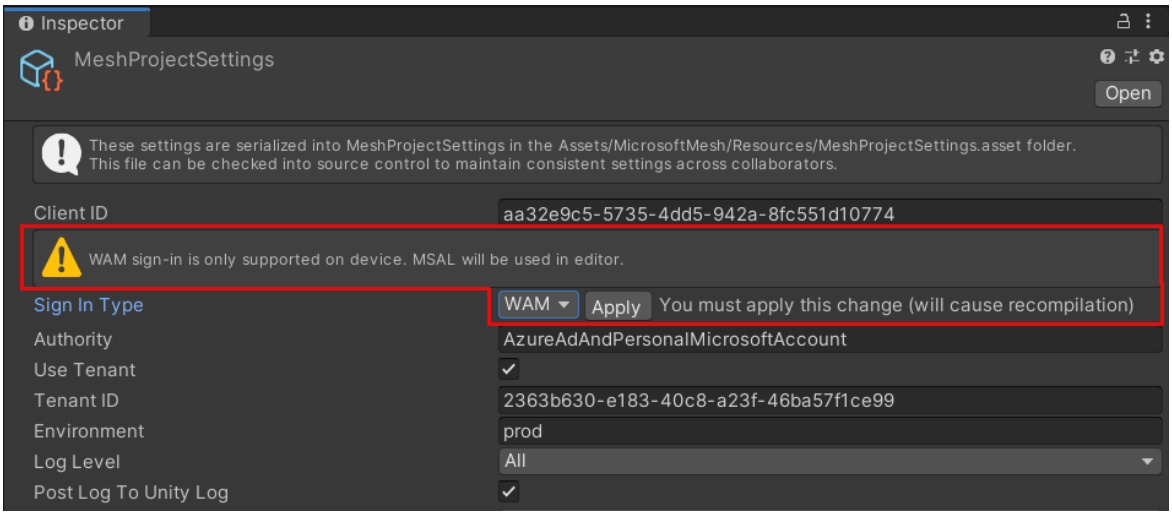


9. Select the **Use Tenant** checkbox. This causes the **Tenant ID** box to appear.



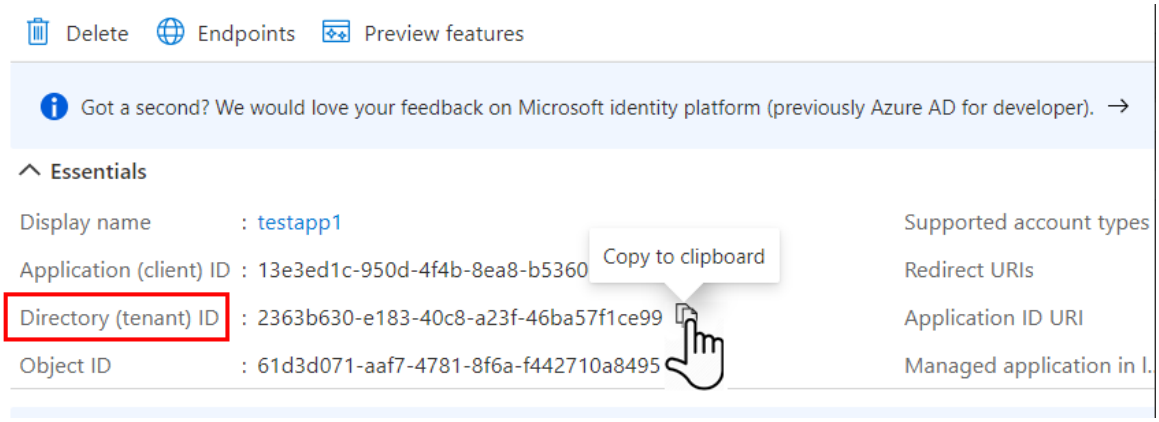
10. The **Sign in Type** is set to **MSAL** by default. Click this drop-down and then choose **WAM**. This is the preferred option when you build and deploy to the HoloLens. To learn more about MSAL and WAM, see HoloLens 2 Authentication Tips





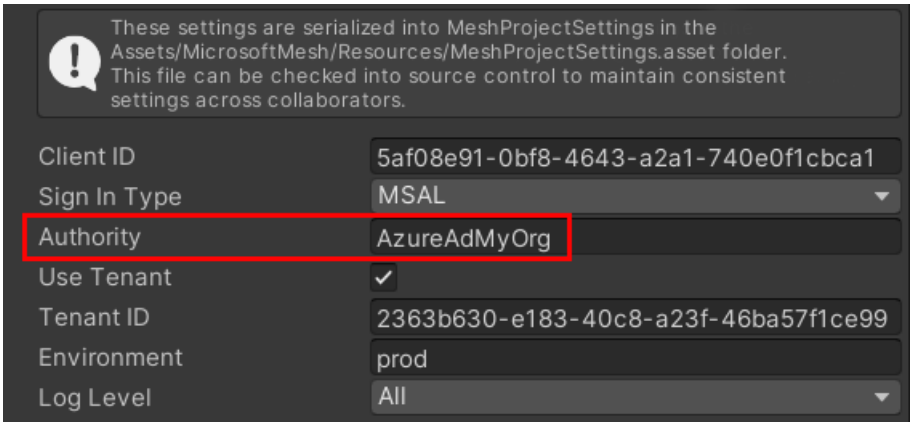
11. To apply the change to WAM, click the **Apply** button.

12. Return to the app registration page in your browser, and then in the **Essentials** section, copy the **Directory (tenant) ID** string to the clipboard. You can do this in the same ways you copied the client ID.



13. In Unity in the **Inspector**, paste the tenant ID into the **Tenant ID** box.

14. In the **Authority** box, enter an authority that matches the account type you selected when you registered your app in the Azure portal.



- If you selected **Accounts in the organizational directory only (Mesh Test Tenant only - Single tenant)**, in the **Authority** box, enter "AzureADMyOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**, in the **Authority** box, enter "AzureADMultipleOrg"
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)**, in the **Authority** box, enter "AzureADandPersonalMicrosoftAccount"

Supported account types

Who can use this application or access this API?

- ◯ Accounts in this organizational directory only (Contoso only - Single tenant)
- ◯ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ◯ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ◯ Personal Microsoft accounts only

Help me choose...

Run the demo scene

1. Click Play to run the demo scene, and then create a session with other collaborators.

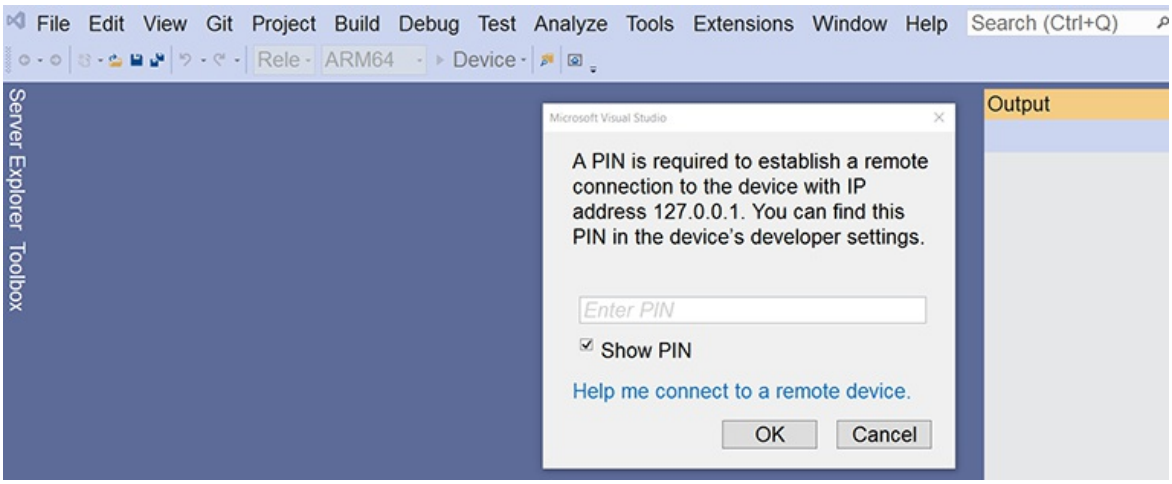
The ToolkitDemo scene gives you a complete user experience (UX) that addresses common needs for collaboration. A number of tools are already pre-configured, so you can press Play and quickly get into a collaborative space.

Build and deploy to your HoloLens 2

Prerequisites

Before building to your device, do the following:

1. Confirm that your device is in Developer Mode.
2. Confirm that your device is paired with your development computer. If it's not, you'll see the following dialog box in Visual Studio during the build process:

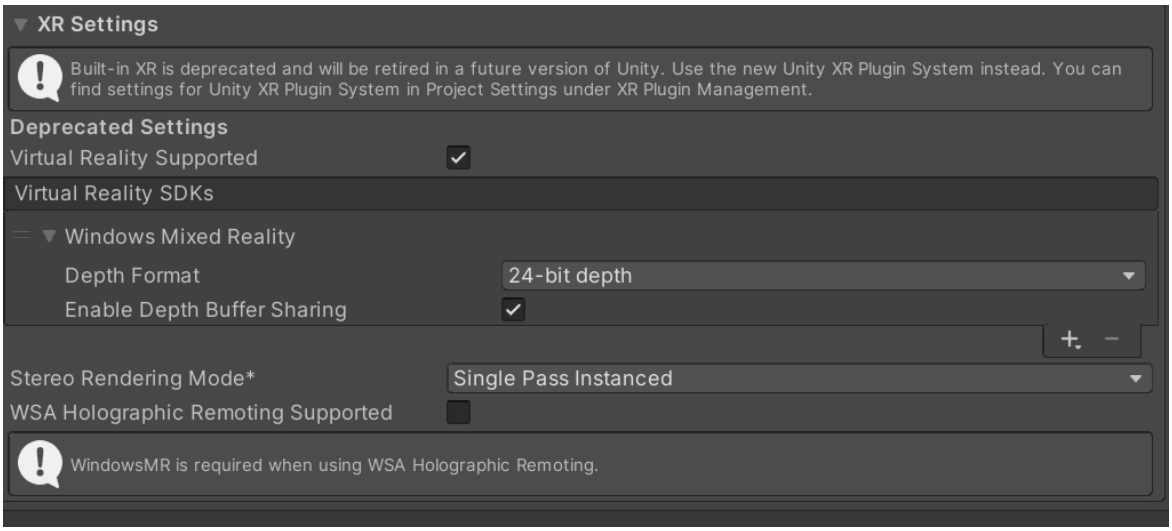


   To learn more about these first two steps, see Using Visual Studio to deploy and debug.

3. Review our HoloLens 2 Authentication Tips

Build and deploy your app

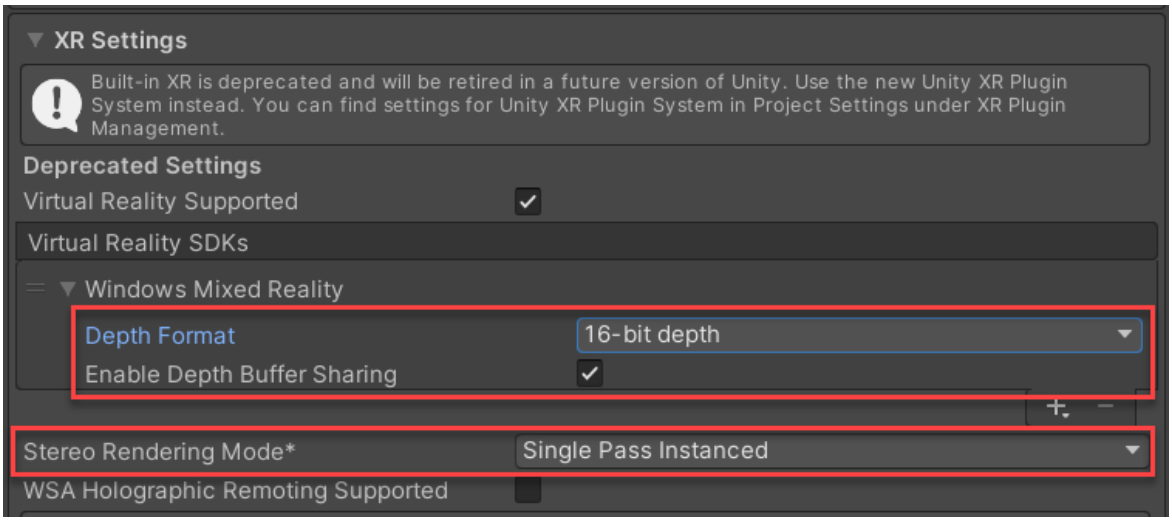
1. Open **File** > **Build Settings**.
2. Click the **Player Settings...** button.
3. Open the **XR Settings** section.
4. Select **Virtual Reality Supported**.



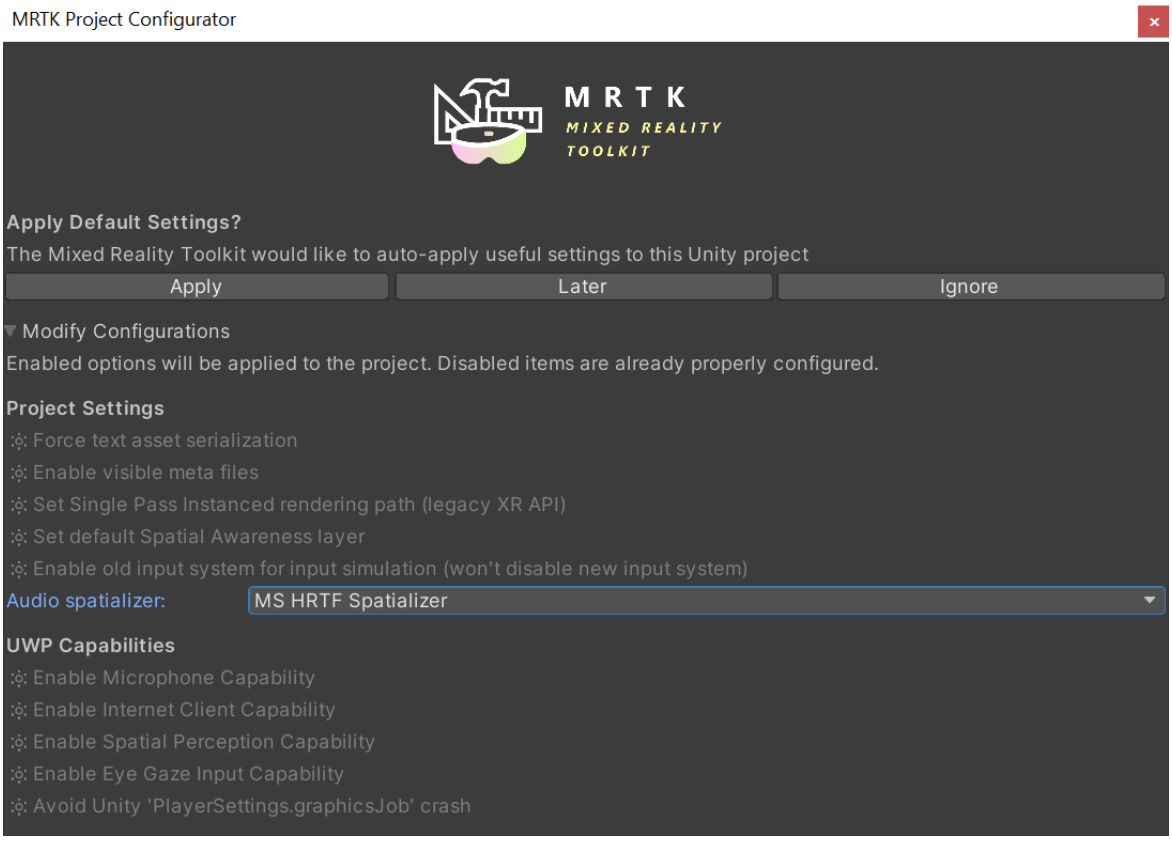
NOTE: This quickstart uses the Unity XR legacy system. To use XR with the XR SDK plugin system, follow the MRTK guide on how to set up your project for XR SDK.

5. In the **Windows Mixed Reality** section, choose the following options:

   - Set **Depth Format** to **16-bit depth**.
   - Check the **Enable Depth Buffer Sharing** check box.
   - Set **Stereo Rendering Mode** to **Single Pass Instanced**.



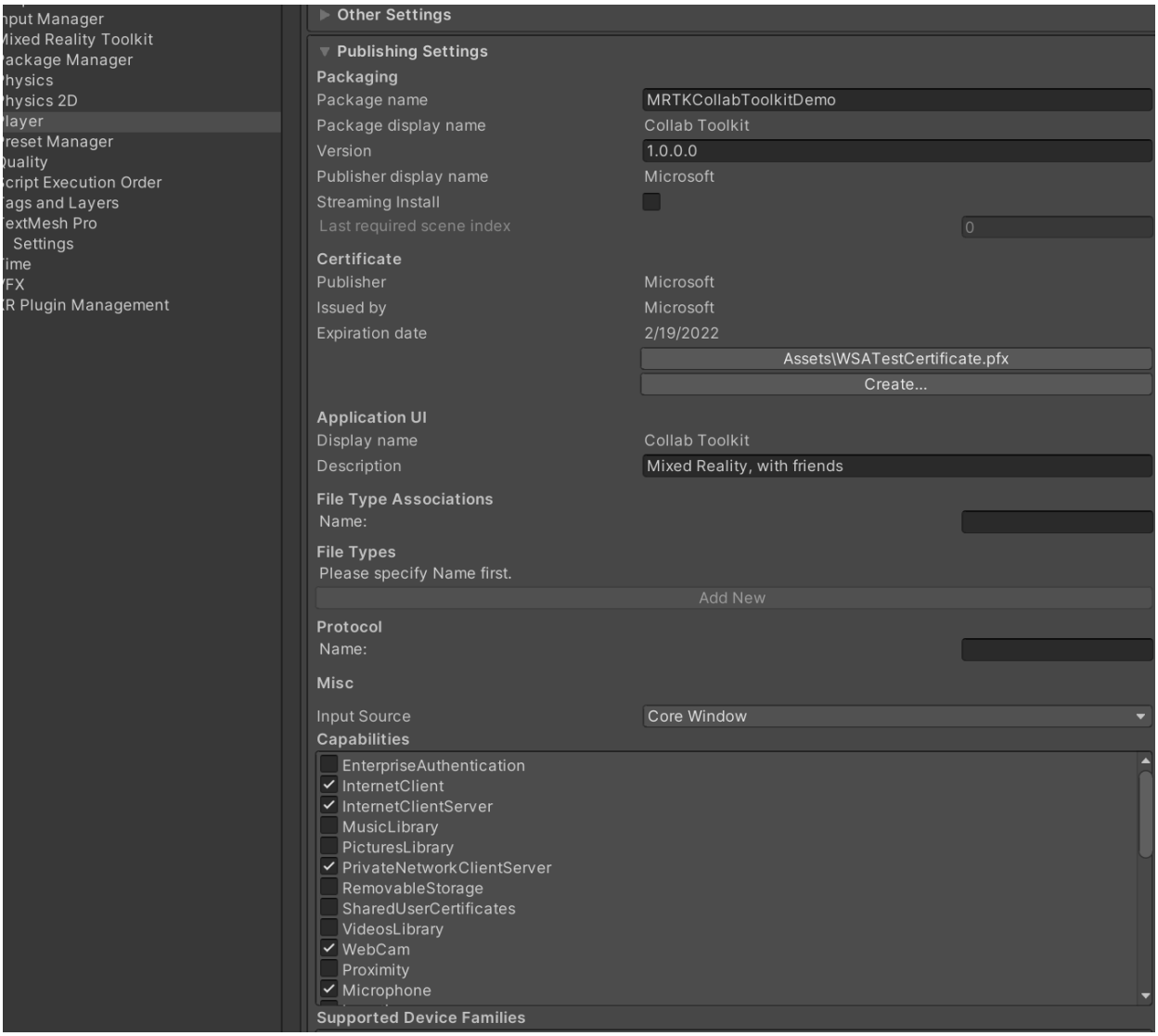
You should see the MRTK Project Configurator window:



6. Confirm any recommended settings from MRTK and ensure that the **Audio spatializer** is set to **MS HRTF Spatializer**.

7. Open the **Publishing Settings** section and ensure that the following options are selected (MRTK should have set them for you):

- InternetClientServer
- WebCam
- Microphone
- SpatialPerception
- GazeInput

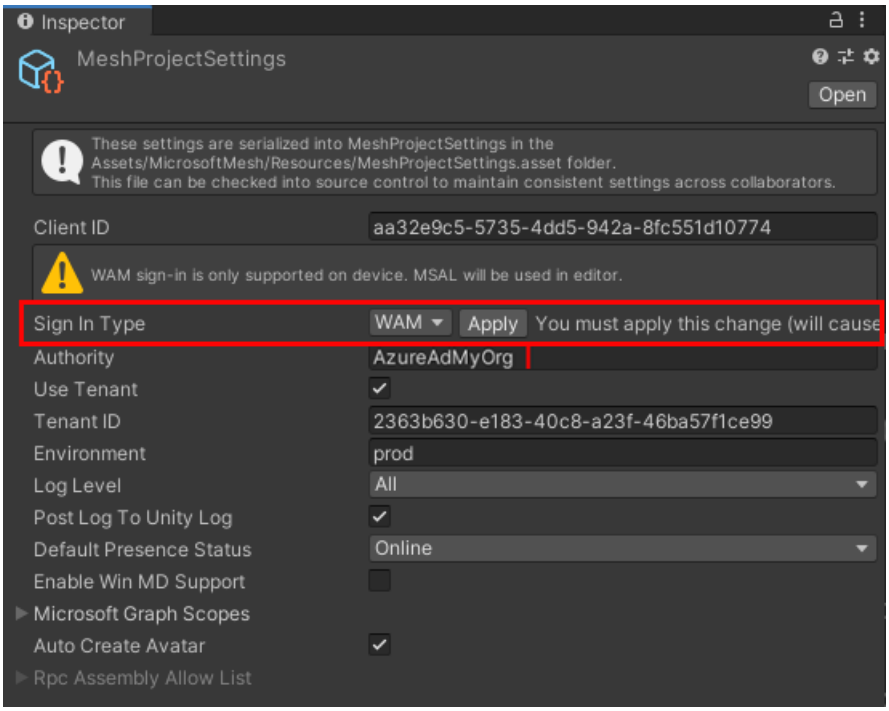


1. Close the **Player Settings** window.
2. In the **Build Settings** window, click the **Build** button.
3. Open the generated solution file in Visual Studio and change the architecture to ARM64.
4. On your HoloLens 2, click Play.

Azure Active Directory login

To use Azure Active directory sign-in:

1. In the menu bar, click **Microsoft Mesh** > **Settings**.
2. Enable **WAM** as the **Sign in Type**, and then apply your setting.



Redirect URI

When setting up your application you'll need to set the redirect URI. You can usually find this in the error code shown when you run the app before the redirect URI is configured correctly. You can also see this in your Unity logs. The output log contains the redirect URI to set in the Mobile and Desktop section of the authentications settings in your AAD application.

The output looks like this:

```
ReplyURL should be 'ms-appx-web://Microsoft.AAD.BrokerPlugIn/S-X-XX-X-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-
XXXXXXXXXX-XXXXXXXXX-XXXXXXXXXX-XXXXXXXXX'
```

Next Steps

Explore some of Mesh's core concepts, starting with the Client object.

Client

Mesh for Unity/Photon Developers

4/21/2021 • 10 minutes to read • Edit Online

If you're a Unity developer who's familiar with Photon Unity Networking (PUN or PUN2), you already have a strong foundation for learning the multiuser features of Mesh's Unity Networking Toolkit (which we'll simply refer to as "Mesh"). There are many similarities between the two systems, but also a few important differences. This article will help you to understand those differences so you can get up and running as quickly as possible.

Using PUN and Mesh Together

PUN provides a comprehensive networking integration with Unity that can be difficult to convert to a different low- or high-level networking solution. Fortunately, Mesh provides features that very closely map to PUN. If you want to use PUN features that aren't found in Mesh, or you prefer to use a specific PUN feature because you're more familiar with it, you can use the systems in parallel and get the best of both worlds. For example:

- Using PUN for data transport and game object synchronization.
- Using Mesh to create sessions, and to locate and invite participants.

Keep in mind that if you decide later that you prefer Mesh's transport system, you can switch to it provided you make certain adjustments related to code path and components.

One possible drawback to using PUN and Mesh together is that you may end up paying for two services and supporting two dependencies. With moderate effort, you can fully replace PUN with Mesh.

Similarities and Differences Between PUN and Mesh

Some Mesh features not currently found in PUN are:

- Video sharing
- Screen sharing
- Spatial anchor sharing
- Mesh avatar system integration. Highly optimized Mixed Reality avatars with built-in eye tracking and mouth synch to voice chat
- Highly optimized transform synchronization (coming soon)
- Graph API utilities

Some features Photon provides that Mesh currently does not:

- Text chat
- Session browsing and searching (maybe coming soon?)
- Dedicated server

Conversion Overview

To complete the full conversion from PUN to Mesh, you will need to:

- Update scripts which reference PUN types, implement with Mesh instead
- Update prefabs which include PUN components, adding Mesh network toolkit components
- Complete PhotonVoice conversion, if used

Prerequisites

To begin the process, you can work in an existing Unity project or start a new project and copy assets into it. Since Photon can co-exist with Mesh, either approach will work.

To sign up and connect your Unity app to the Mesh service, you need to follow the setup steps which start with registering your app with the tenant.

The Mesh SDK package, which is included in your project, allows your app to connect to the service and use all Mesh features. If you want to provide additional scripts and prefabs that you can drop in and use to get going even faster, add the following packages, found in the **Packages** folder, to your project:

com.microsoft.mesh.toolkit com.microsoft.mesh.toolkit.ux

The first package listed above includes the systems and scripts that make converting from PUN to Mesh much easier.

Before moving on, make sure to verify your app's connection to the service. There should be a message in the **Console** that confirms this.

Converting PUN Types to Mesh Types

PUN types and their equivalents in Mesh:

PUN	MESH
`PhotonView`	`NetObject`
Message Events	`MeshServiceManager`, `NetTransportBase`
`PhotonNetwork`	`MeshServiceManager`, `ParticipantManager`, `NetObjectManager`
`PhotonTransformView`	`NetLocalTransform`
Rooms	Sessions
Room and player properties	`MeshServiceManager.Session.State`
Text Chat	Not yet implemented
PhotonAnimatorView	Not yet implemented

Converting from PhotonView to NetObject

The `NetObject` is essentially the same concept as `PhotonView`. They both allow a game object to be synchronized over the network session by sending messages from one instance to the corresponding instances on the other apps. `PhotonView` also provides the concept of ownership; this is also supported by `NetObject`.

The `NetObject` script is located in Unity here:

Packages/com.microsoft.mesh.toolkit/Assets/Network/Scripts/

Replace `PhotonView` properties in your application code with `NetObject` and rename the following members as follows:

Properties

PUN	MESH
isMine	IsMine
viewID	NetId
ownerActorNr	OwnerId

Public Attributes

PUN	MESH
ownershipTransfer	OwnershipTransferMode
RPC()	CallRPC()
[PunRPC]	[NetRPC]
RequestOwnership()	TransferOwnership()
PhotonView.Find(int viewID)	NetObjectManager.Instance.GetNetObject(int netId)

Example:

```
// Photon
//   Find a PhotonView by view ID, check if it's owned by the local player,
//   then do "something."
int viewID = 10;
PhotonView photonView = PhotonView.Find(viewID);
if ( photonView != null && photonView.isMine )
{
    int viewID = photonView.viewID;
    int ownerNr = photonView.ownerActorNr;
    // Do something
}

// Mesh
//   Find a NetObject by net ID, check if it's owned by the local player,
//   then do "something"
int netId = 10;
NetObject netObject = NetObjectManager.Instance.GetNetObject(netID);
if ( netObject != null && netObject.IsMine )
{
    int netId = netObject.NetId;
    int ownerId = netObject.OwnerId;
    // Do something
}
```

Instantiating networked objects

In PUN, you instantiate a prefab with a `PhotonView` for the session with `PhotonNetwork.Instantiate()`.

In Mesh, you can replace this with `NetObjectManager.Instance.SpawnNetObject()` to instantiate a `NetObject` prefab.

Note that in PUN, you can specify the initial transform and custom data, which is used when instantiating the object for the other players. Mesh does not yet support this but is expected to do so in the future.

Note that in both cases, the prefab must be placed within a folder named "Resources."

Example:

```
// Photon
//   Instantiate a prefab with a PhotonView component on it
GameObject newGO = PhotonNetwork.Instantiate("MyPrefabName", new Vector3(0, 0, 0), Quaternion.identity,
null);

// Mesh
//   Instantiate a prefab with a NetObject component on it
NetObject newSO = NetObjectManager.Instance.SpawnNetObject(MyPrefab);
```

Destroying networked objects

In PUN, you destroy an instance of a `PhotonView` for the session with `PhotonNetwork.Destroy()`.

In Mesh, you can use `GameObject.Destroy()` on `NetObjects`, or `NetObjectManager.Instance.DestroyNetObject()`.

Example:

```
// Photon
//   Destroy a PhotonView and its game object
PhotonNetwork.Destroy(myPhotonView);


// Mesh
//   Destroy a NetObject and its game object
NetObjectManager.Instance.DestroyNetObject(myNetObject);
// or
GameObject.Destroy(myNetObject);
```

Remote Procedure Calls

In PUN, you do the following:

- Apply the `[PunRPC]` attribute to a method.
- Add a `PhotonView` to the same game object.
- Call the `RPC("<method name>")` function on the `PhotonView`` component, passing the name of the method to invoke.

In Mesh, you do the following:

- Apply the `[NetRPC]` attribute to a method.
- Add an instance of `NetObject` to the same game object.
- Call `CallRPC("<method name>")` on the `NetObject` component, passing the name of the method to invoke.

Example:

```
// Photon
//   Call an RPC to notify everyone about the new nickname
[PunRPC]
void RPCSetNickName(string newName)
{
     // Update a UI label, etc
}

void SetNickName(string newName)
{
    PhotonView photonView = PhotonView.Get(this);
    photonView.RPC("RPCSetNickName", RpcTarget.All, newName);
}

// Mesh
//   Call an RPC to notify everyone about the new nickname
[NetRPC]
void RPCSetNickName(string newName)
{
     // Update a UI label, etc.
}

void SetNickName(string newName)
{
    NetObject netObject = GetComponent<NetObject>();
    netObject.CallRPC(RPCFlags.LocalInvoke, "RPCSetNickName", newName);
}
```

State serialization block

`PhotonView` supports the `IPunObservable` interface, which provides a simple method to serialize and transmit some state.

The Mesh networking toolkit doesn't provide this, so you must replace this usage with an RPC call and call it at the desired frequency. Note that RPCs can be called unreliably.

Example:

```
// Photon
//   OnPhotonSerializeView is called several times per second to send the
//   data written to the PhotonStream from the owner to the remote copies
void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
       if ( stream.isWriting )
       {
           stream.SendNext(x);
           stream.SendNext(y);
           stream.SendNext(z);
       }
       else
       {
           x = (int)stream.ReceiveNext();
           y = (int)stream.ReceiveNext();
           z = (int)stream.ReceiveNext();
       }
}


// Mesh
//   Call an RPC at regular intervals to broadcast some state
[NetRPC]
void RPCUpdateState(float _x, float _y, float _z)
{
    x = _x;
    y = _y;
    z = _z;
}

// Start and stop using StartCoroutine/StopCoroutine at the appropriate times
IEnumerator CoSendState()
{
    NetObject netObject = GetComponent<NetObject>();
    while ( true )
    {
        if (netObject.IsMine)
        {
        // Note, could send as a Vector3 instead
        netObject.CallRPC("RPCUpdateState", x, y, z);
        }
        yield return null;
    }
}
```

Ownership Requests

`PhotonView` supports the `IPunOwnershipCallbacks` interface, which allows a script to respond to ownership requests.

In Mesh:

1. Remove the PUN interface, and add a method tagged with `[NetRPC]` with the signature `RPCRequestTransfer(int newOwnerNetId)`.
2. Call `TransferOwnership(newOwnerNetId)` to allow the transfer to the participant with net id `newOwnerNetId`, or to disallow the transfer, do not call it.

Example:

```
// Photon
//  Respond to ownership transfer request and transfer ownership
//  These callbacks get called for requests on any PhotonView
void OnOwnershipRequest(PhotonView targetView, Player requestingPlayer)
{
    if ( targetView.IsMine && targetView == myPhotonView )
    {
        // Note: this is how you request ownership transfer any time

        targetView.TransferOwnership(requestingPlayer);
    }
}

void OnOwnershipTransfered(PhotonView targetView, Player previousOwner)
{
    // Do something?
}

// Mesh
//  Respond to ownership transfer request and transfer ownership
//  These callbacks get called only for the game object with this shared object
[NetRPC]
void RPCRequestTransfer(int newOwnerId)
{
    // Note: this is how you request ownership transfer any time
    myNetObject.TransferOwnership(newOwnerId);
}

void Start()
{
    myNetObject.OnOwnershipTransferred += OnOwnershipTransferred;
}

void OnOwnershipTransferred(int prevOwnerId)
{
    // Do something?
}
```

Transform Synchronization

This works in a similar way for both systems:

PUN provides the `PhotonTransformView` component, which synchronizes a game object's world space transform.

Mesh provides the `NetLocalTransform` component, which synchronizes the configured transform (defaulting to the game object's transform) in its local space. Note that this allows the object transform to be synchronized relative to another transform, and not just the world. When running on HoloLens, the world origin may not always be where you intend, so it's useful to place your shared objects as children of a transform that can be placed wherever desired, and synchronized relative to it. This means that all participants will see the exact same scene relative to the parent transform.

Voice

PhotonVoice and Mesh both provide spatialized voice chat. See Transport for details on implementing voice chat in your Mesh experience.

Rooms and Friends

PUN uses the concept of "rooms" to get participants using the same app together. Mesh uses "sessions" for this purpose. There aren't many differences between the rooms and sessions themselves, but there are major differences between how rooms and sessions are managed and enumerated.

In PUN, you can create a room with a name and optional password. All other players can view the list of active rooms and choose one to join. PUN does not provide invites. With Mesh, anyone can create a session, but other

participants can only join if explicitly invited, and there is no built-in method to enumerate active sessions.

If you want to implement enumeration of active sessions with Mesh, you can use MSGraph APIs to register active sessions and then query that list.

PUN also provides methods to query the online status of a "friend" based on an id for that player. This information does not persist on the Photon side; it must be tracked by the app or other services, including any kind of "friend list." Mesh works with Graph contacts to invite and add them to sessions and to query their status on the service. The Toolkit provides utilities to enumerate and search for contacts within the organization.

Optimizations

PUN can compare data sent previously and then send less data, or, if the data hasn't changed, PUN can refrain from sending it again. PUN provides some built-in general optimization to decrease the amount of data that needs to be sent over the network.

Mesh doesn't currently provide as much optimization but can be modified as desired to be optimized for each application.

Room and Player Properties

Both Mesh and PUN support synchronization of key-value pairs of properties. There are some differences in the specifics of usage, but the general concept is the same. PUN provides `Room.CustomProperties`, `RoomInfo.SetCustomProperties`, `Player.CustomProperties` and `Player.SetCustomProperties`, which store key-value pairs using Hashtables.

Mesh provides MeshServiceManager.CurrentSession.State, which has methods for setting adding, getting, and setting string properties, as well as for incrementing and getting monotonically increasing integer properties. When a property is changed by anyone in the session, a StateChanged event is invoked for the key whose value changed when the new value is received.

More Comparisons:

FEATURE	PUN	MESH
Cloud/on premise hosting	Comes in two popular variants: Photon Cloud, which is a managed server offering, and Photon Server, where the backend server is managed by the developer	There is no managed SaaS offering. Developers must create their own environment on Azure
API references	.NET, C++, Objective C	C# .NET only
Identity provider	B2C only	Azure AD and Azure AD B2C
Cloud regions supported	Limited to 14 Photon cloud regions	All Azure public cloud regions
Support for RPC	Yes, using the PunRPC tag	Yes, using the NetRPC tag
Room and Lobby	Yes, supported	Rooms are called sessions. No support for Lobby, requires confirmation

Create and Experiment!

We recommend that you create a small project in Mesh and experiment with some of the features that are implemented differently from PUN.

Next Steps

Explore some of Mesh's core concepts, starting with the Client object.

Client

Intro to Mesh Tools for Unity

4/26/2021 • 12 minutes to read • Edit Online

The Mesh Tools for Unity are pre-configured tools for creating, starting, and sharing models in a Mesh session. These tools help accelerate app creation, providing you with more time to focus on collaborating instead of creating the app. In this tutorial, you'll create and start a Mesh session. Next, you'll add an avatar for yourself and then add the MRTK-UI collaboration tool which provides a session flow UI. Finally, you'll add a Centerpiece to the session as a common reference which can be used to relocate and reorient the session contents.

Instructions are also provided for building and deploying the app to a HoloLens 2.

Prerequisites

You'll need:

- A Windows computer withVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure account with an active subscription. Create an account for free.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with setting up your developer environment. You'll need a Client ID and Tenant ID for this tutorial.
- If you're working with commercial users, one or more Work or School accounts.
- A Unity 2019.4.x installation.
- The Mixed Reality Feature tool for Unity. This page explains how to download and use the tool to import packages.
- The latest version of the Mesh private preview zip file.

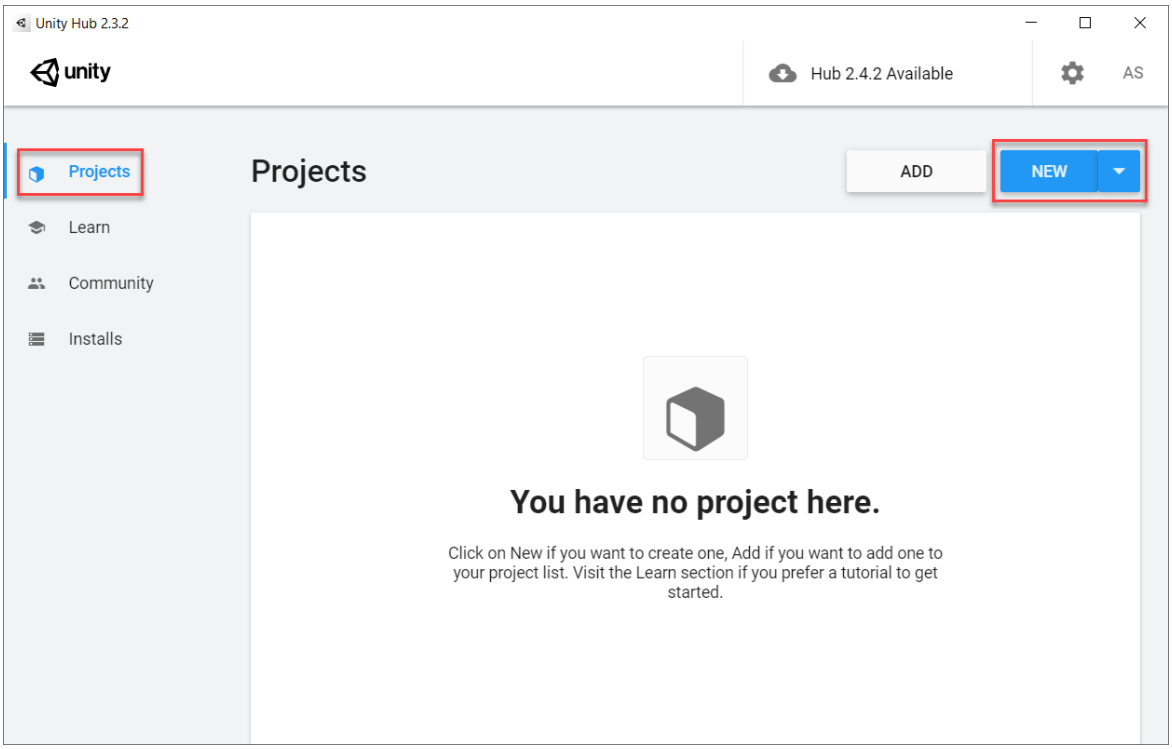
Caution

Caution: When working on Windows, there is a MAX_PATH limit of 255 characters. Unity is affected by these limits and may fail to compile if any file path is longer than 255 characters. Therefore, we strongly recommend that you save your Unity project as close to the root of the drive as possible and keep file and folder names short.

Set up a Unity project for Microsoft Mesh

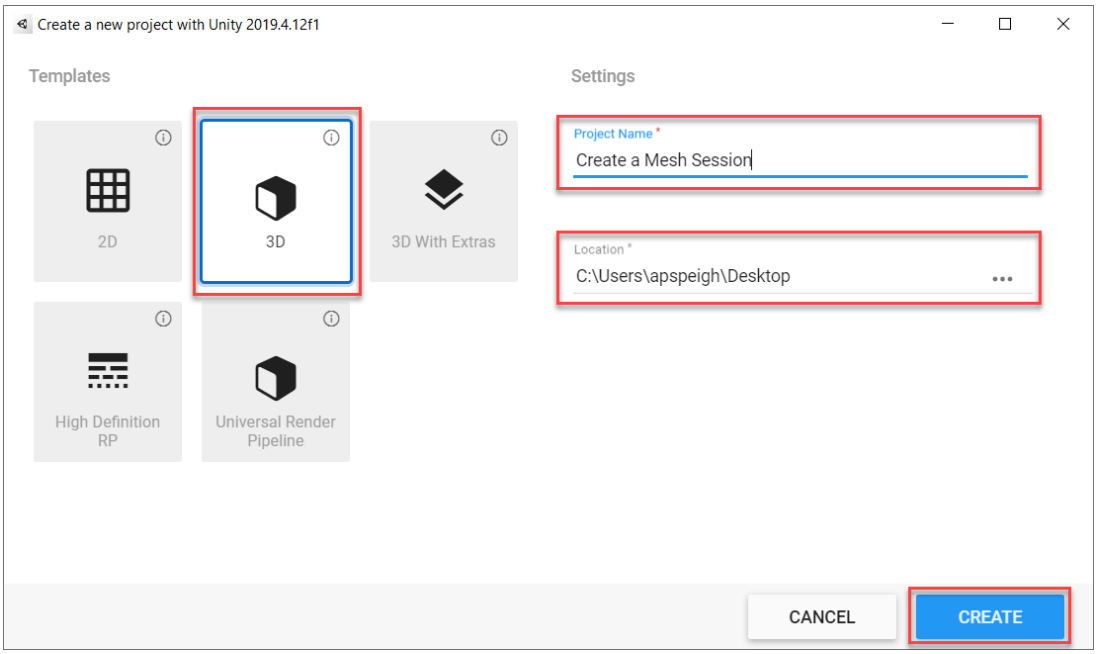
Create a new Unity project

1. Launch the **Unity Hub**, and then select the **Projects** tab. Next, click the down arrow next to the **New** button and select the Unity version specified in the tutorial Prerequisites section.

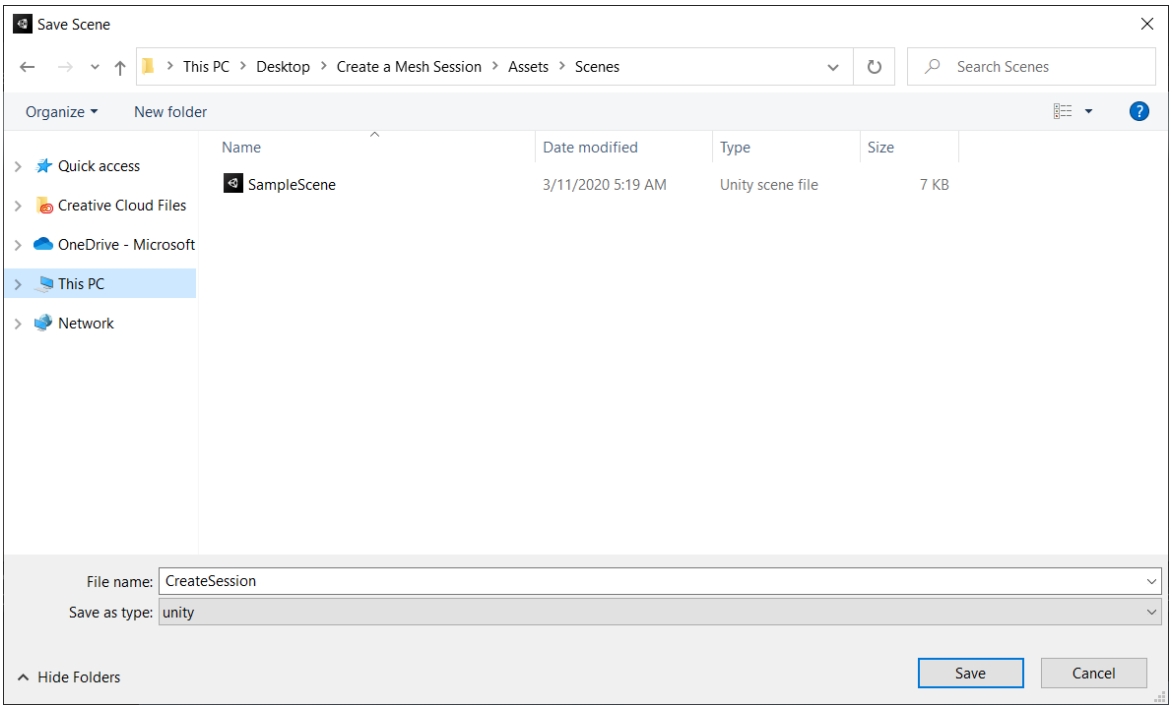


2. In the **Create a new project** window:

   - Ensure **Templates** is set to **3D**.
   - Enter a suitable **Project Name**--for example, *Create a Mesh Session*.
   - Choose a suitable **Location** to store your project (for example, *D:\MeshTutorials*).
   - Click the **Create** button to create and launch your new Unity project.



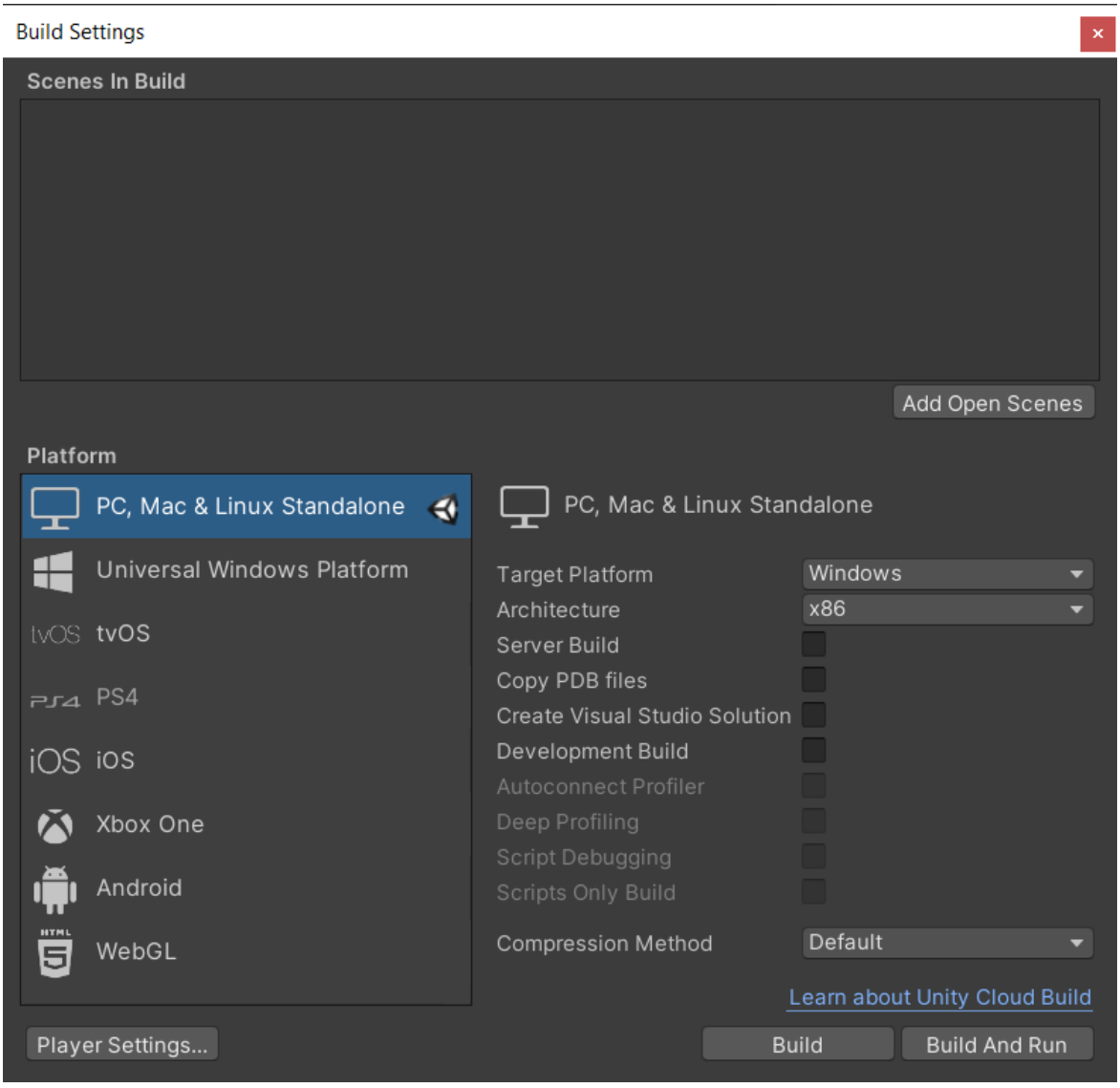
3. On the menu bar, select **File** > **Save As** and then select the **Scenes** folder. Enter a name for the scene (for example, *CreateSession*) and then select **Save**.



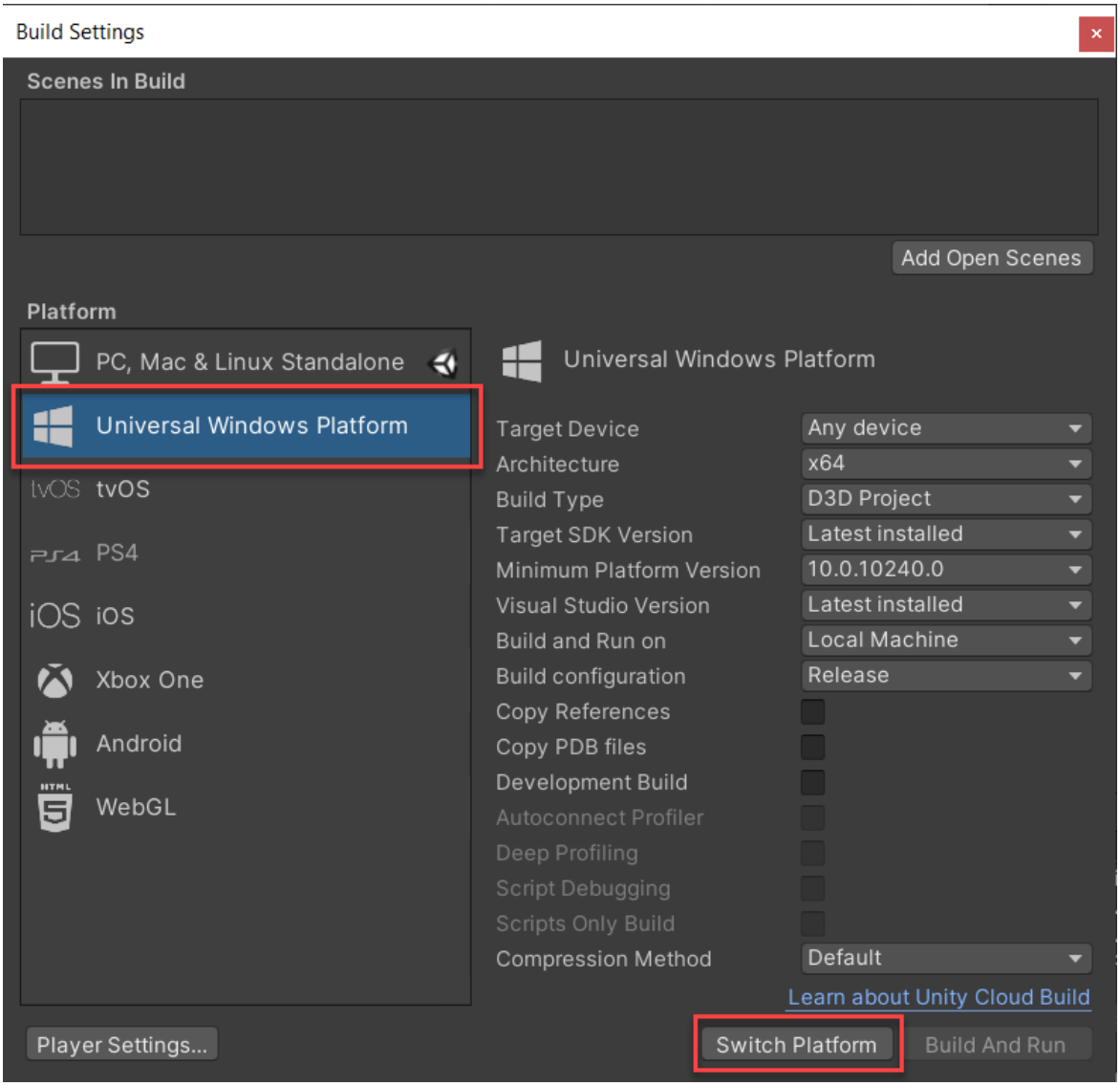
Configure Unity for Windows Mixed Reality

Switch Build Platform

1. In the menu bar, select **File** > **Build Settings...**.

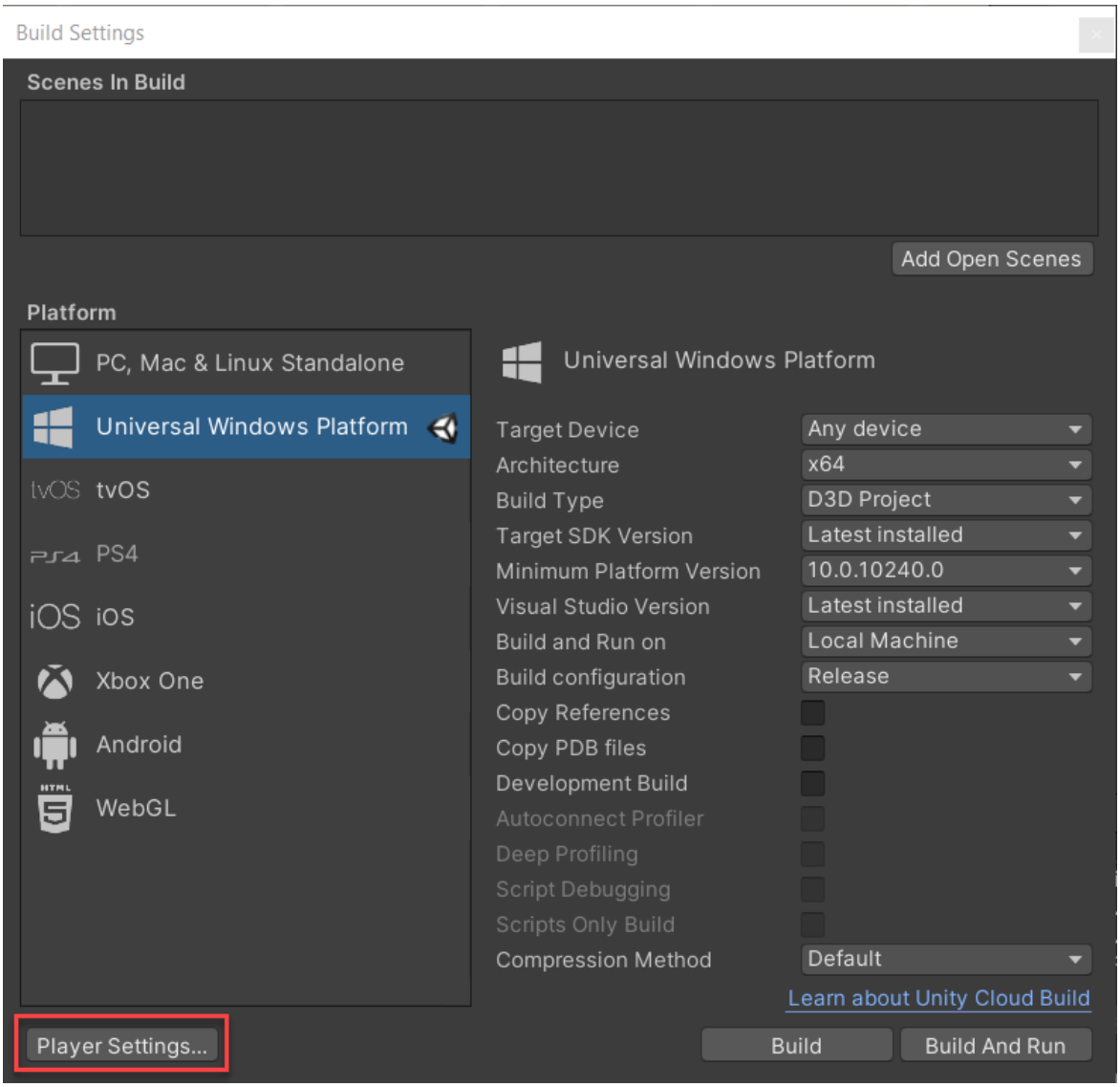


2. In the **Build Settings** window, select **Universal Windows Platform** and then click the **Switch Platform** button. Unity will begin the process to switch the platform.

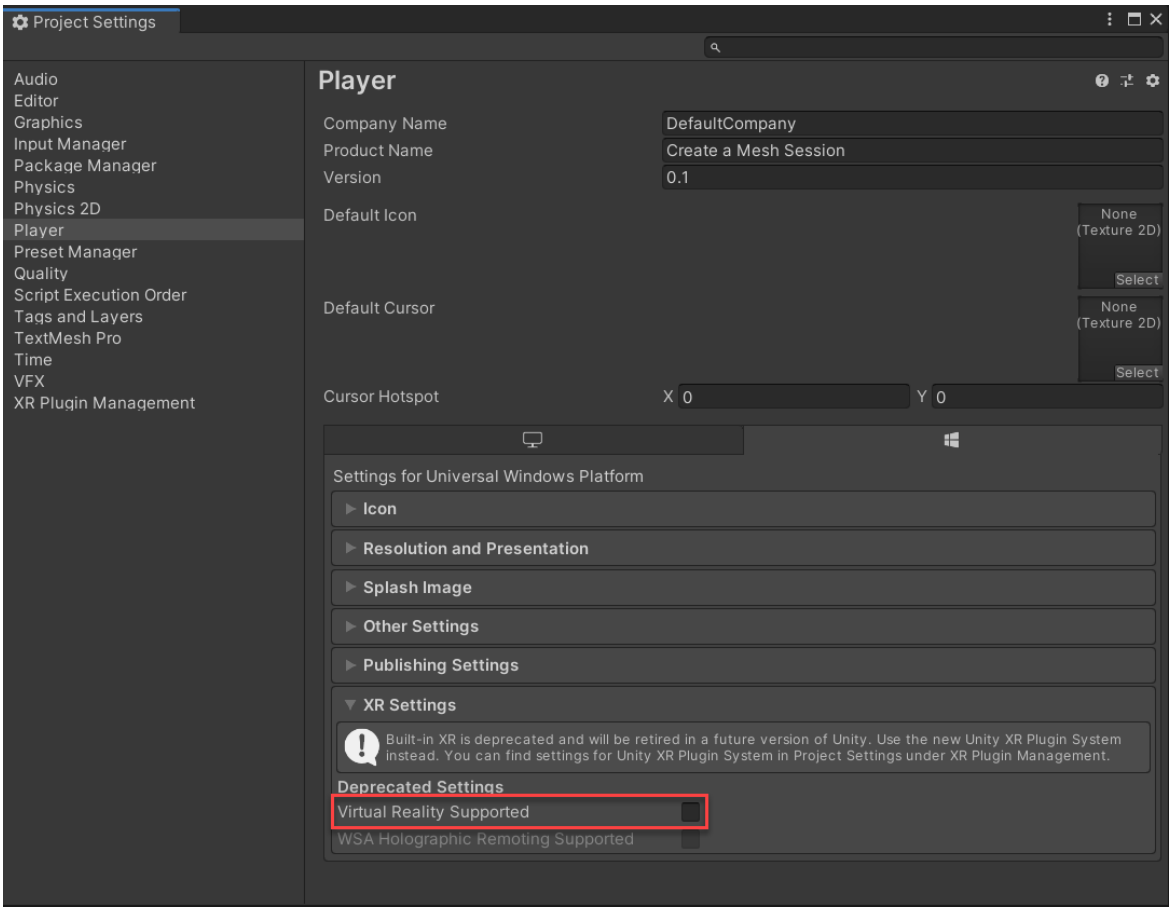


Enable Virtual Reality

1. In the **Build Settings** window, select the **Player Settings...** button.

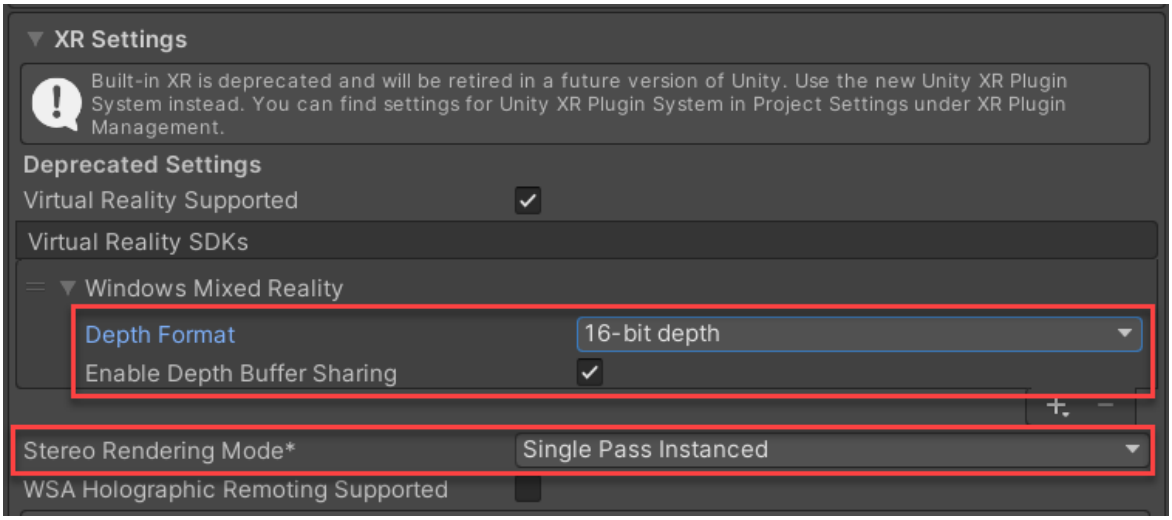


2. Select **Player**, and then expand the **XR Settings** section.
3. Select the **Virtual Reality Supported** check box to enable virtual reality. This adds the Windows Mixed Reality SDK.



4. In the **Windows Mixed Reality** section, choose the following options:

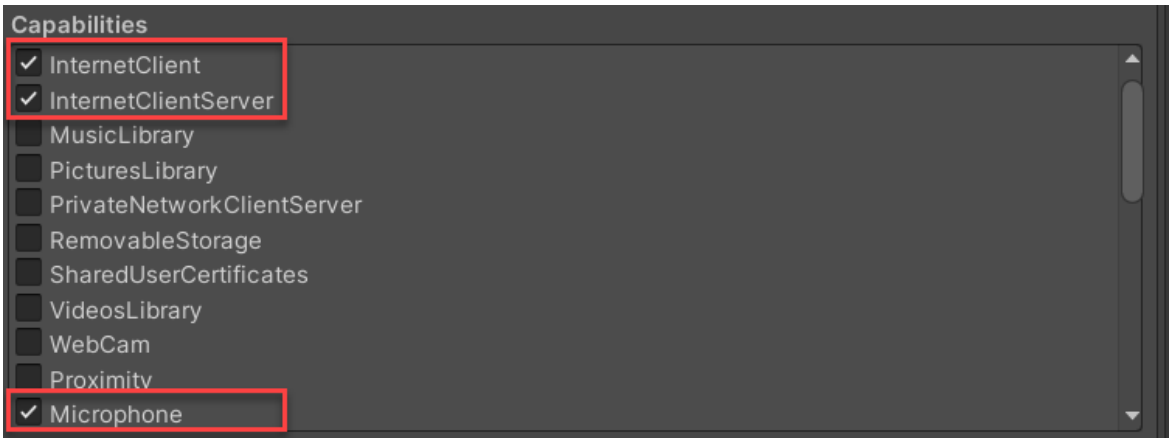
   - Set **Depth Format** to **16-bit depth**.
   - Check the **Enable Depth Buffer Sharing** check box.
   - Set **Stereo Rendering Mode** to **Single Pass Instanced**.



Add Capabilities

1. Expand the **Publishing Settings** panel, and then in the **Capabilities** section, select the following options:

   - InternetClient
   - InternetClientServer
   - Microphone



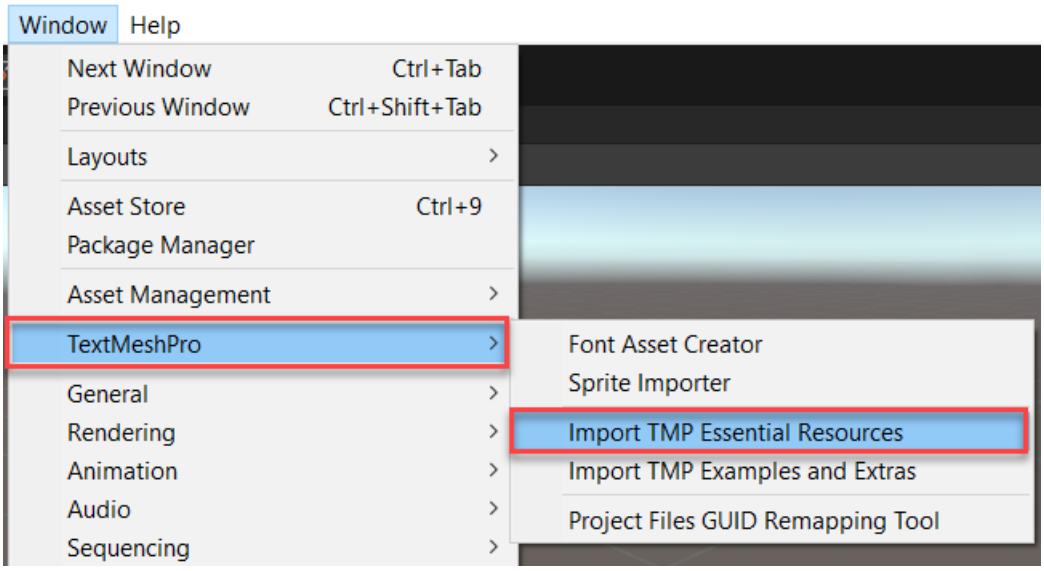
2. In the **Supported Device Families** section, select **Holographic**.

3. Close the **Project Settings** and **Build Settings** windows.

Import and Configure Resources

Import TextMeshPro (TMP) Essential Resources

1. In the menu bar, select **Window** > **TextMeshPro** > **Import TMP Essential Resources**.

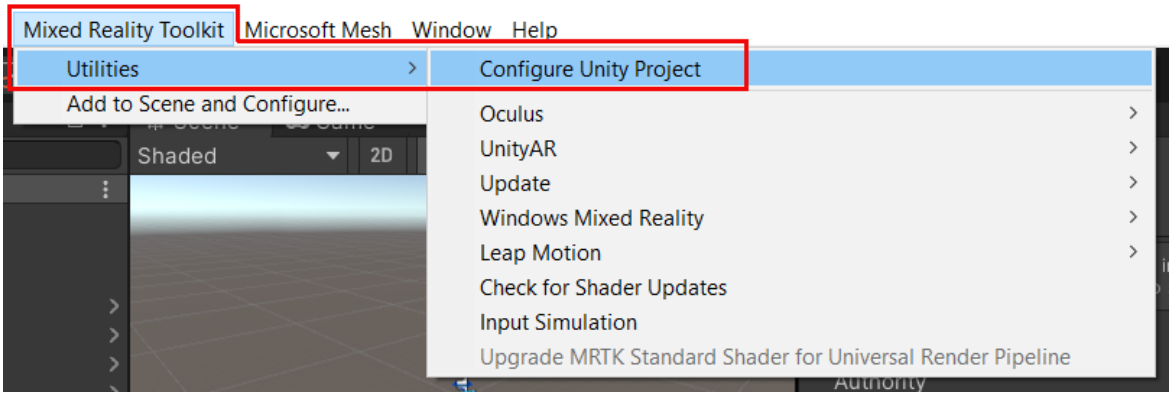


2. In the **Import Unity Package** window, click the **All** button to ensure that all assets are selected, and then click the **Import** button.

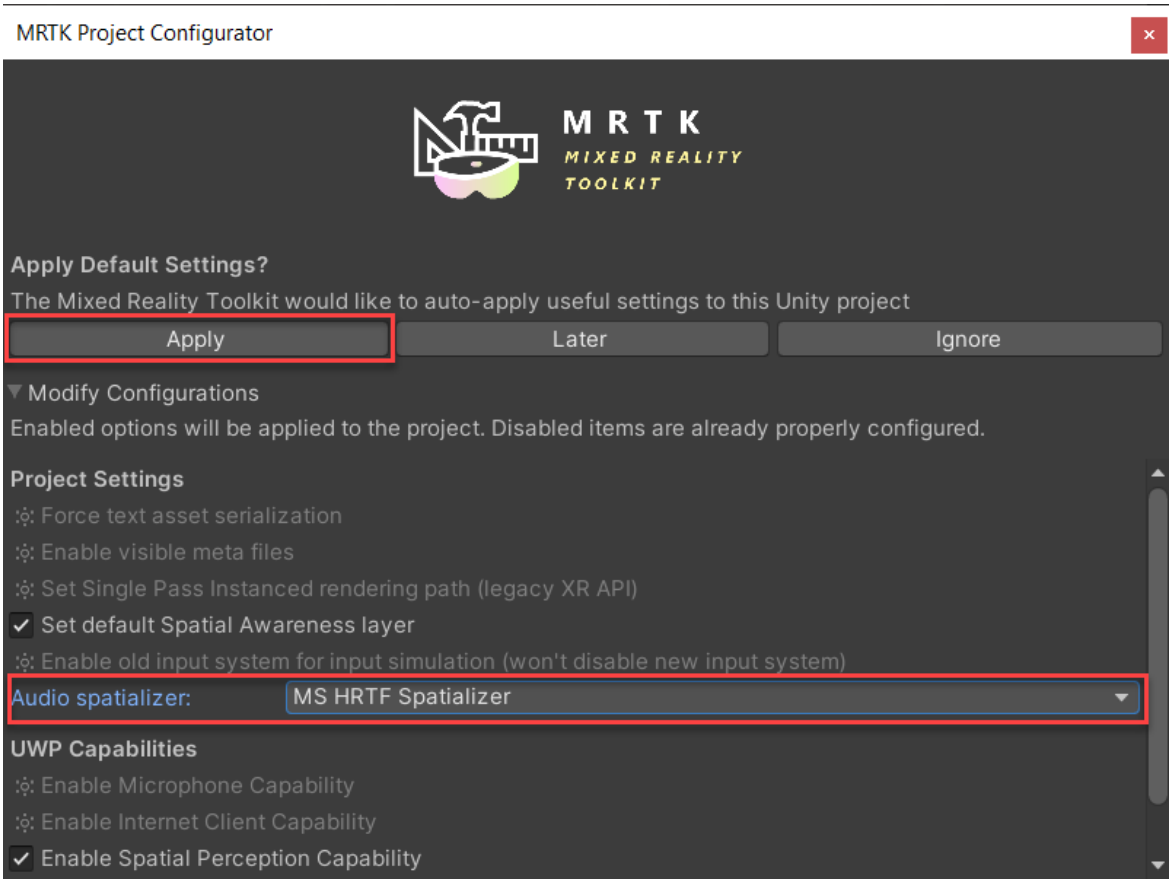
Import the MRTK Unity Foundation package

1. Follow the instructions on the Mixed Reality Feature Tool page to import the **Mixed Reality Toolkit Foundation 2.6.x** package into the project.

2. After the MRTK Foundation package is imported, the **MRTK Project Configurator** window will appear. If the window does not appear, you can open it from the menu bar--select **Mixed Reality Toolkit** > **Utilities** > **Configure Unity Project**.



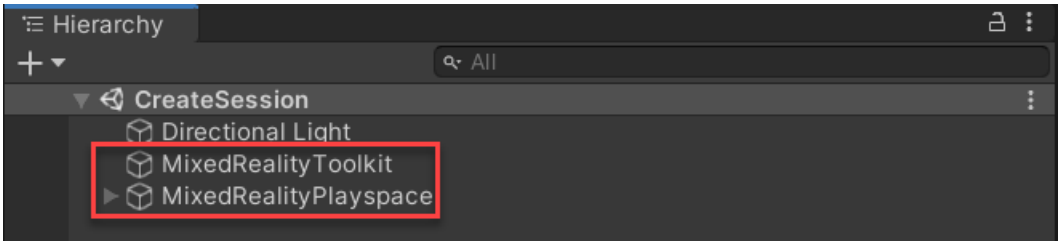
3. In the **MRTK Project Configurator** window, select the **Audio Spatializer** drop down and then select **MS HRTF Spatializer**.



4. Select the **Apply** button to apply the settings.

5. In the menu bar, select **Mixed Reality Toolkit** > **Add to Scene and Configure...** to add MRTK to the current scene. Note that two new objects are added to the **Hierarchy**:

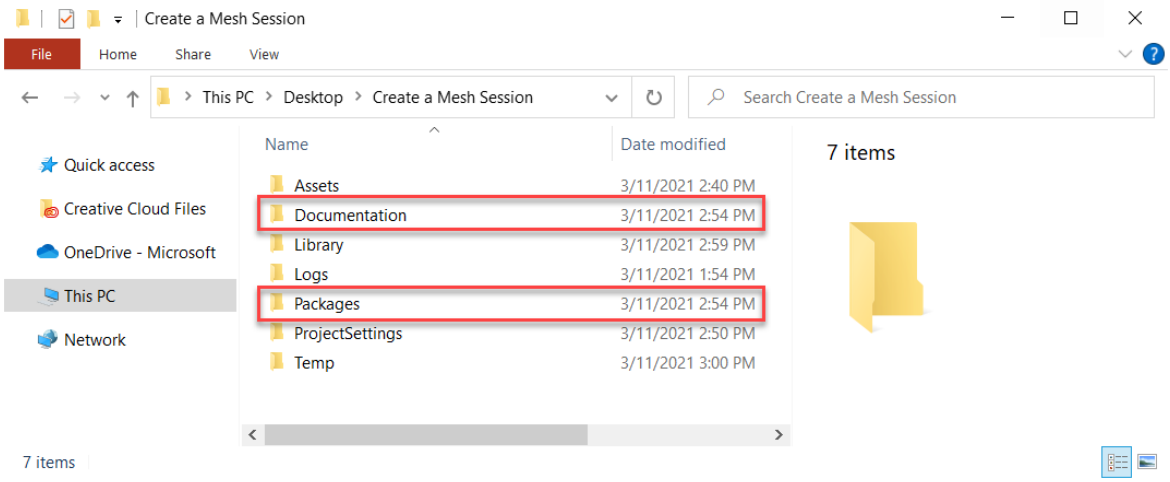
   - MixedRealityToolkit
   - MixedRealityPlayspace



Add the Private Preview folders to the project

1. Unzip the private preview zip file.

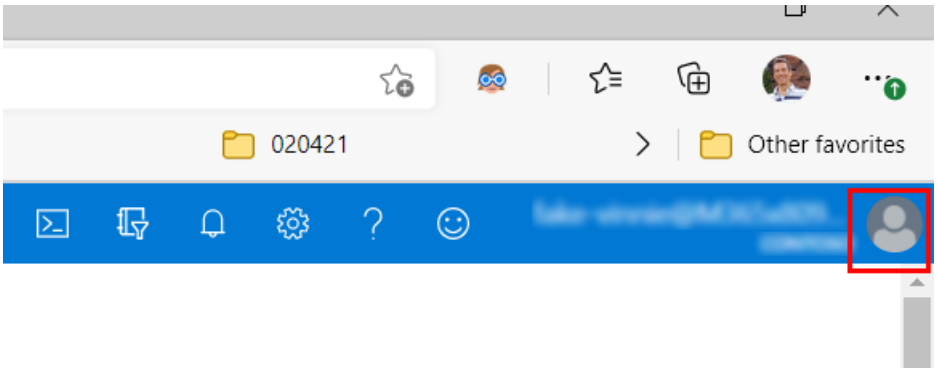
2. Copy the unzipped folders **Documentation** and **Packages** and paste them into the root folder of the project. This adds several packages to your project.



Add the client ID and tenant ID to your project

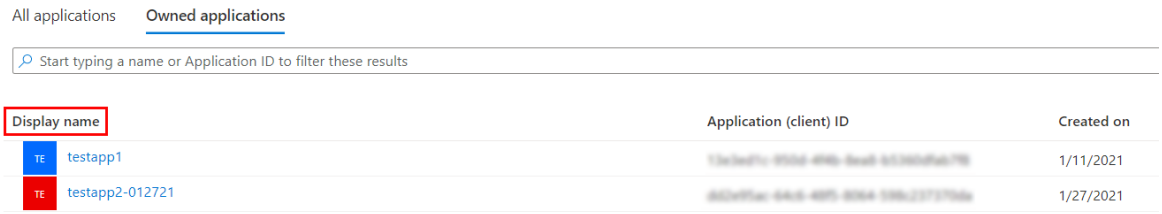
When your application connects to the Mesh service, the service needs a way to identity it. You provide this capability by adding the client ID that was created earlier for your application when you registered it with your tenant, along with your tenant ID.

1. Sign in to the Azure portal using either a work or school account or a personal Microsoft account.
2. If your account gives you access to more than one tenant, in the upper right corner of the Portal screen, click the account popup and then select the tenant you want.

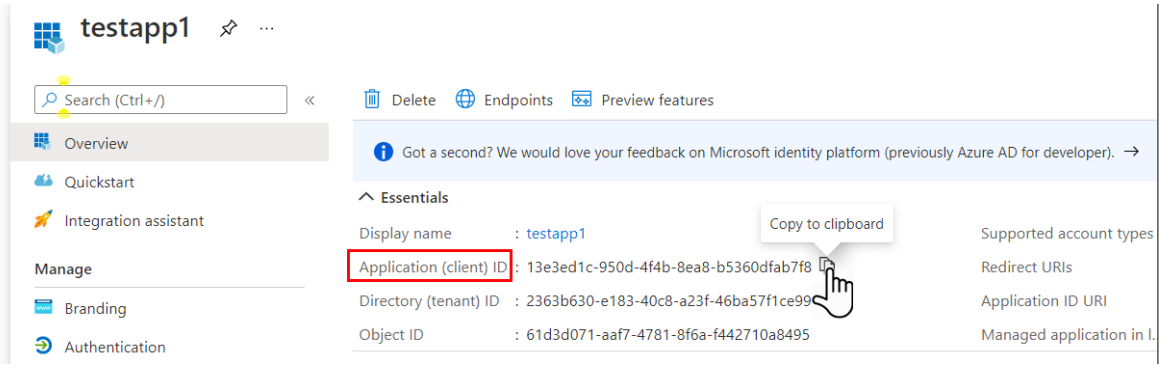


3. Search for and select **Azure Active Directory**. The easiest way is to click the **View** button on the page.

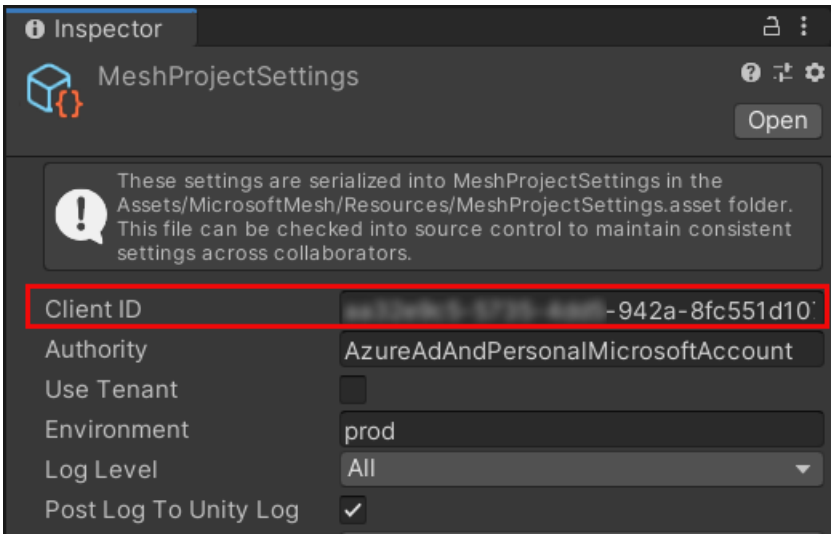
4. In the left-side navigation under **Manage**, select **App registrations**.
5. Under **Display Name**, click the name of your application.



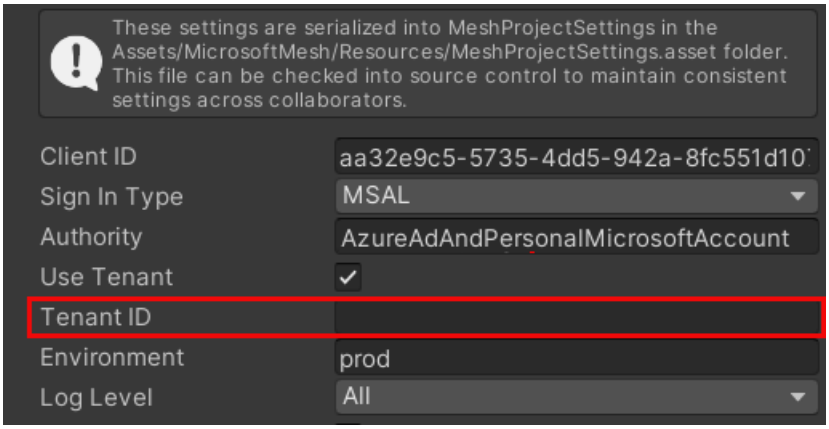
6. In the **Essentials** section, copy the **Application (client ID)** string to the clipboard. To do this, move your cursor just past the end of the ID, and then click the **Copy to clipboard** icon that appears. You can also select the ID and then press Ctrl + C.



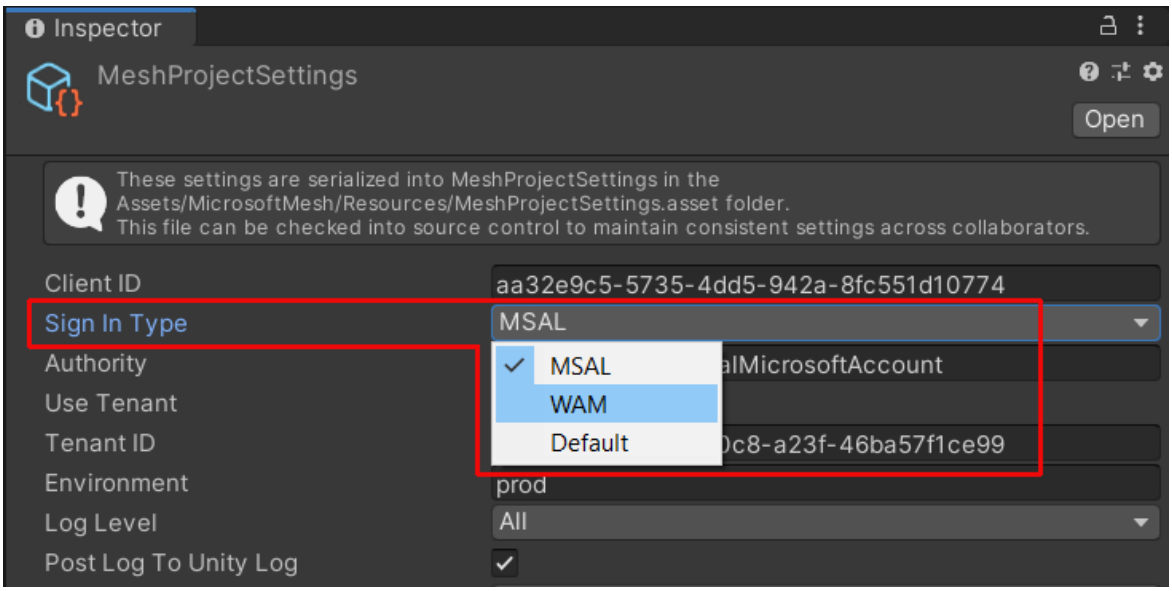
7. In Unity on the menu bar, select **Microsoft Mesh** > **Settings**.
8. In the **Inspector**, paste the client ID into the **Client ID** box.

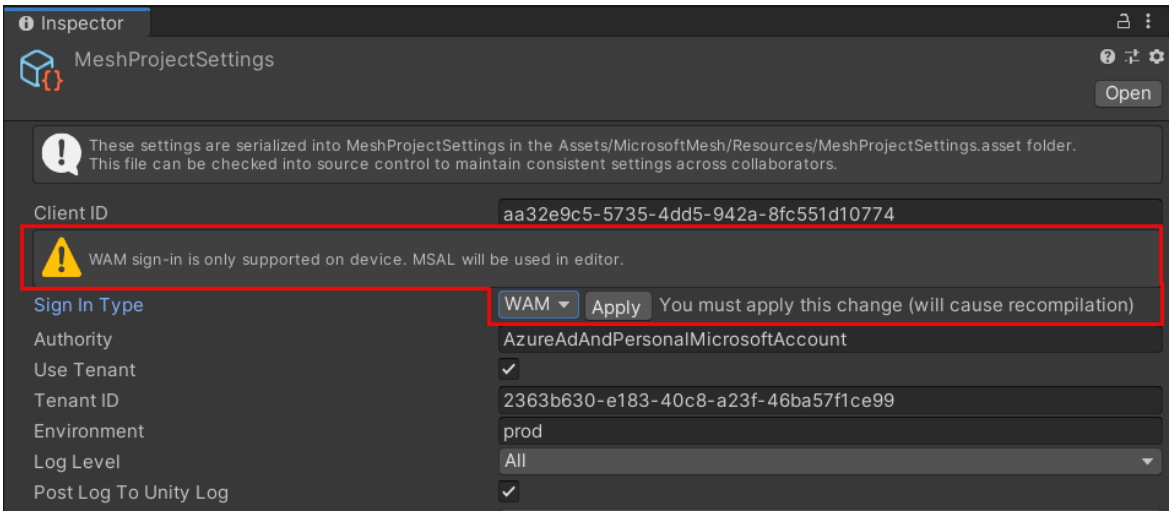


9. Select the **Use Tenant** checkbox. This causes the **Tenant ID** box to appear.



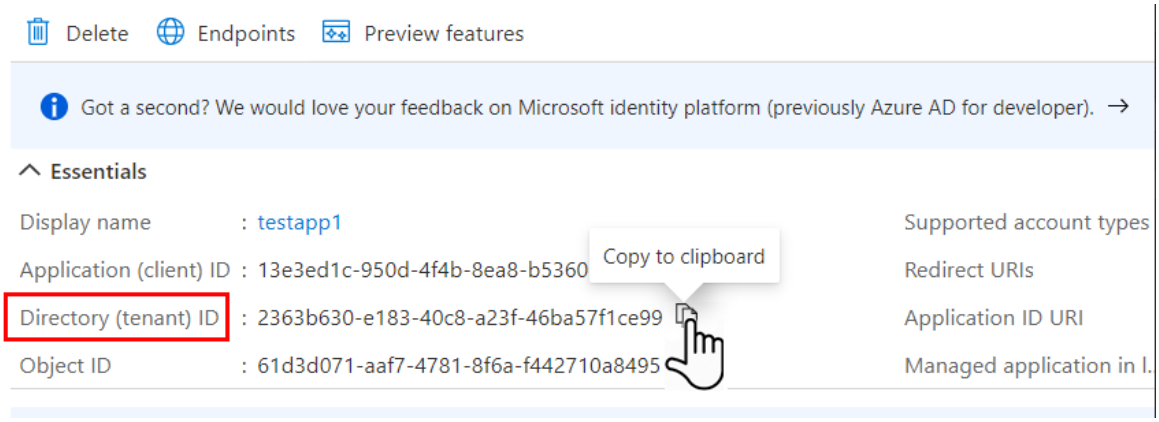
10. The **Sign in Type** is set to **MSAL** by default. Click this drop-down and then choose **WAM**. This is the preferred option when you build and deploy to the HoloLens. To learn more about MSAL and WAM, see HoloLens 2 Authentication Tips





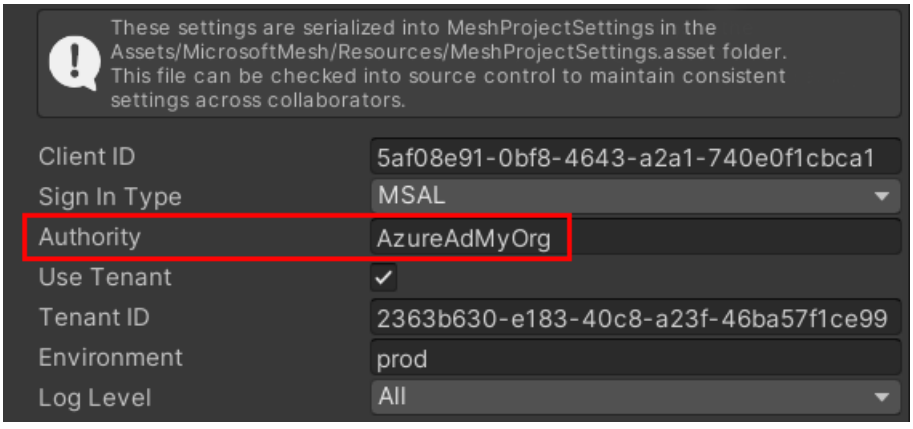
11. To apply the change to WAM, click the **Apply** button.

12. Return to the app registration page in your browser, and then in the **Essentials** section, copy the **Directory (tenant) ID** string to the clipboard. You can do this in the same ways you copied the client ID.



13. In Unity in the **Inspector**, paste the tenant ID into the **Tenant ID** box.

14. In the **Authority** box, enter an authority that matches the account type you selected when you registered your app in the Azure portal.



- If you selected **Accounts in the organizational directory only (Mesh Test Tenant only - Single tenant)**, in the **Authority** box, enter "AzureADMyOrg."
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**, in the **Authority** box, enter "AzureADMultipleOrg."
- If you selected **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)**, in the **Authority** box, enter "AzureADandPersonalMicrosoftAccount."

Supported account types

Who can use this application or access this API?

- ◯ Accounts in this organizational directory only (Contoso only - Single tenant)
- ◯ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ◯ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ◯ Personal Microsoft accounts only

Help me choose...

Add the MRTK UI Prefab to the scene

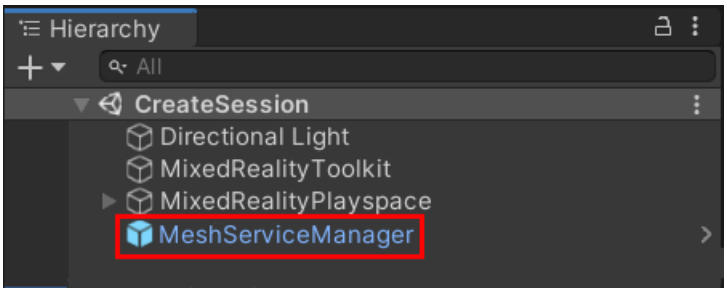
The MRTK UI prefab adds a session flow UI to the scene which allows you to create a collaborative session and invite users.

1. In the **Project** window, navigate to this folder:

   Packages > Mesh Tools for Unity > Assets > Manager > Resources

2. Drag the **MeshServiceManager** prefab to the **Hierarchy**.

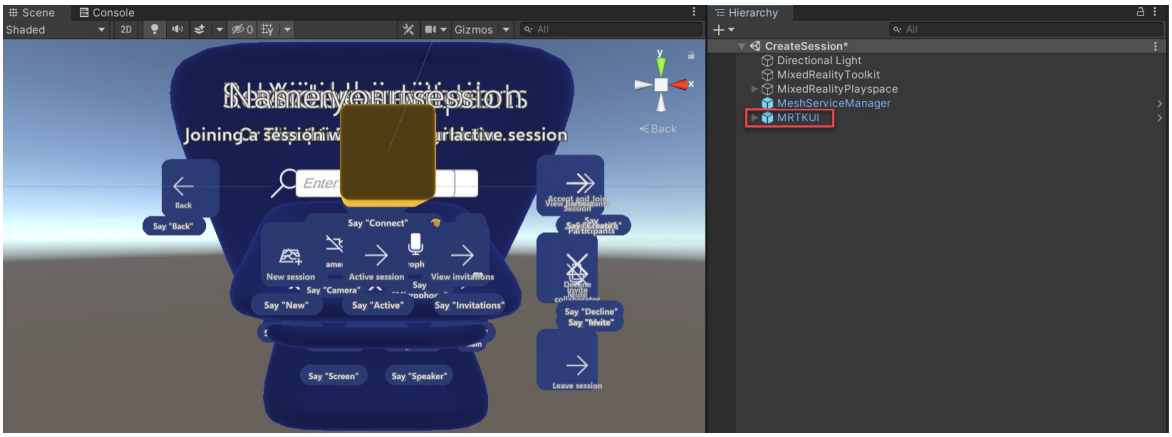
   This prefab gives you a session manager singleton that you can access from code. The UX and Mesh avatars rely on the prefab.



3. In the **Project** window, navigate to this folder:

   Packages > Mesh Tools for Unity (UX) > MRTKUI

4. Drag the **MRTKUI** prefab to the **Hierarchy**.



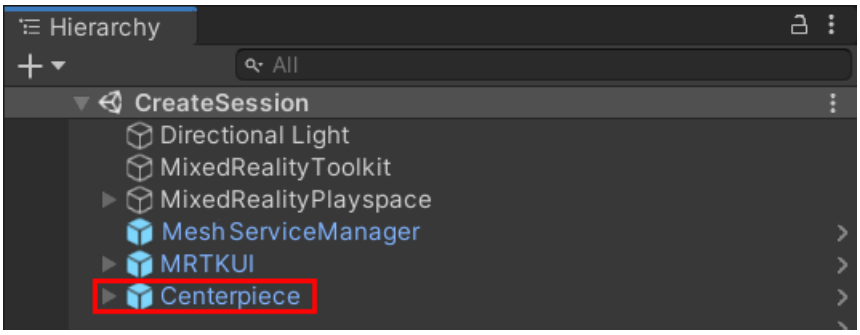
Add a Centerpiece to the scene

The Centerpiece enables participants to share a common holographic reference no matter where they are in the world or which device they're using. Participants can use the tabletop handles to relocate and reorient the entire session contents.

1. In the **Project** window, navigate to this folder:

Packages > Mesh Tools for Unity (UX) > Centerpiece > Prefabs

2. Drag the **Centerpiece** prefab to the **Hierarchy**.



3. In the **Hierarchy**, select the **Centerpiece** object.

4. In the **Inspector** window, set the **Transform Position** to **0, -0.8, 0**. This lowers the centerpiece in the scene.

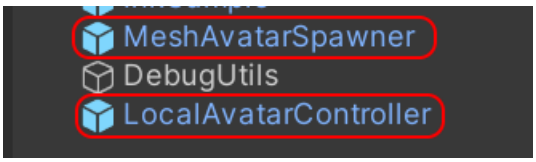
Add an Avatar to the scene

With the avatar package in your project, you can spawn and share a virtual representation of yourself with other participants in the session. Input from a participant's head, hands and eyes are processed by MRTK and projected onto an animated avatar representation in the collaborative session. In addition, the package provides spatial audio support for each of the spawned avatars.

1. In the **Project** window, navigate to this folder:

   Packages > Mesh Tools for Unity (Avatar) > Runtime > Prefabs

2. Drag the **MeshAvatarSpawner** and **LocalAvatarController** prefabs to the **Hierarchy**.

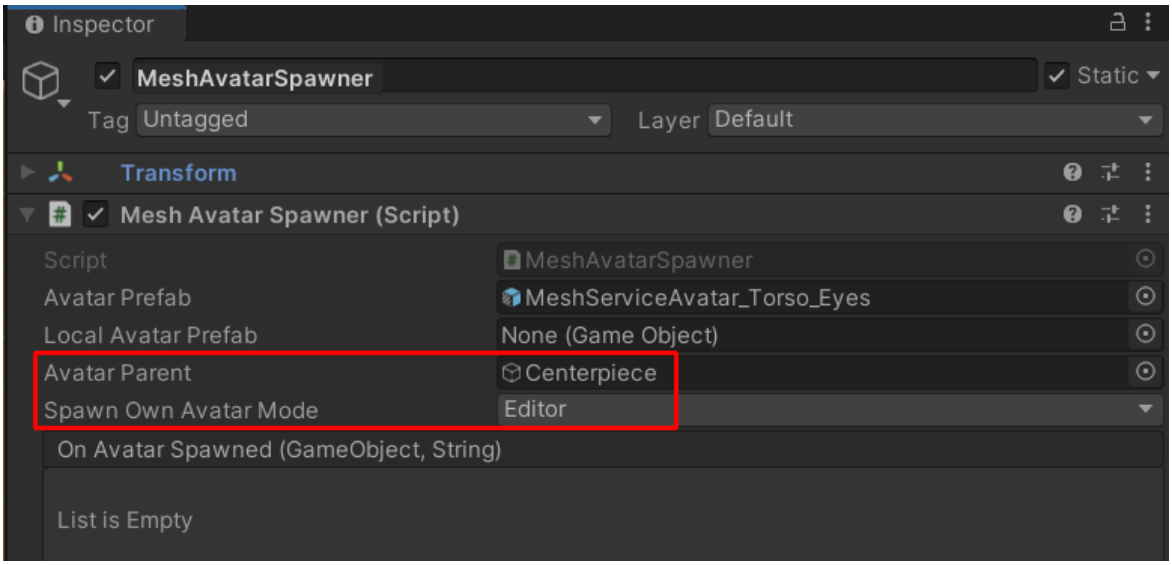


The **MeshAvatarSpawner** prefab comes with the base setup to spawn avatars with spatial audio support when participants join the session. The **LocalAvatarController** lets you animate your own avatar using sensor tracking from any supported input device.

1. In the **Hierarchy**, select the **MeshAvatarSpawner** object.

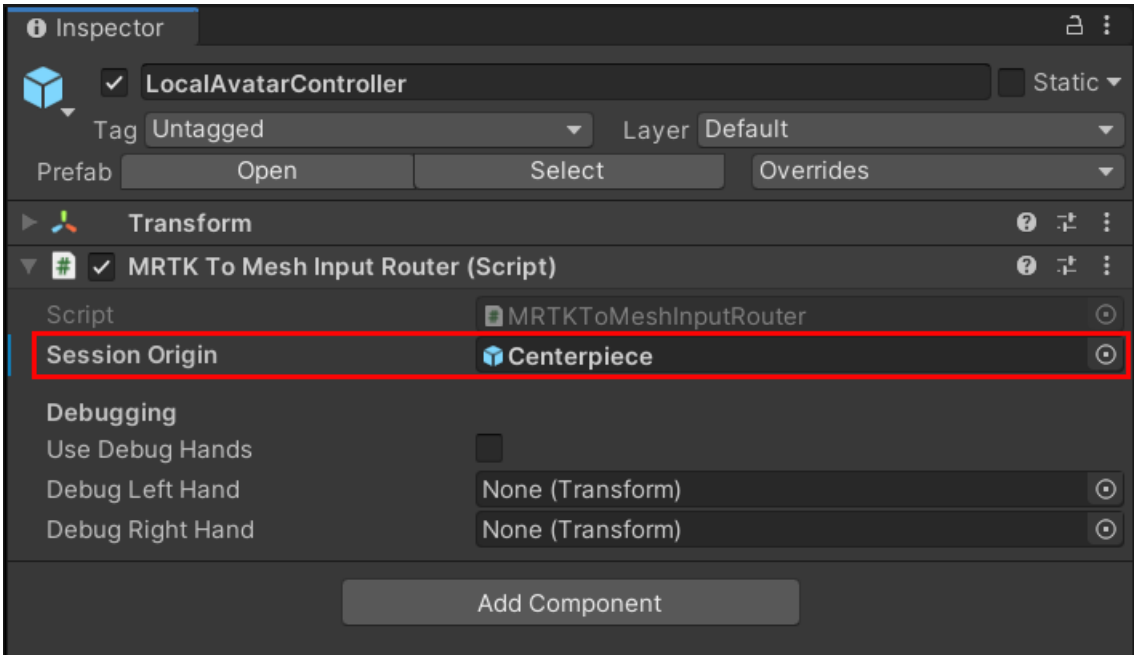
2. In the **Inspector** window, find the **Mesh Avatar Spawner (Script)** and then modify the following:

   - Drag the **Centerpiece** object from the **Hierarchy** to the **Avatar Parent** property.
   - For the **Spawn Own Avatar Mode** property, select **Nothing**, and then select **Editor**.



3. In the **Hierarchy**, select the **LocalAvatarController** object. In the **Inspector**, note that **LocalAvatarController** has a script named **MRTK To Mesh Input router** which contains a property called **Session Origin**.

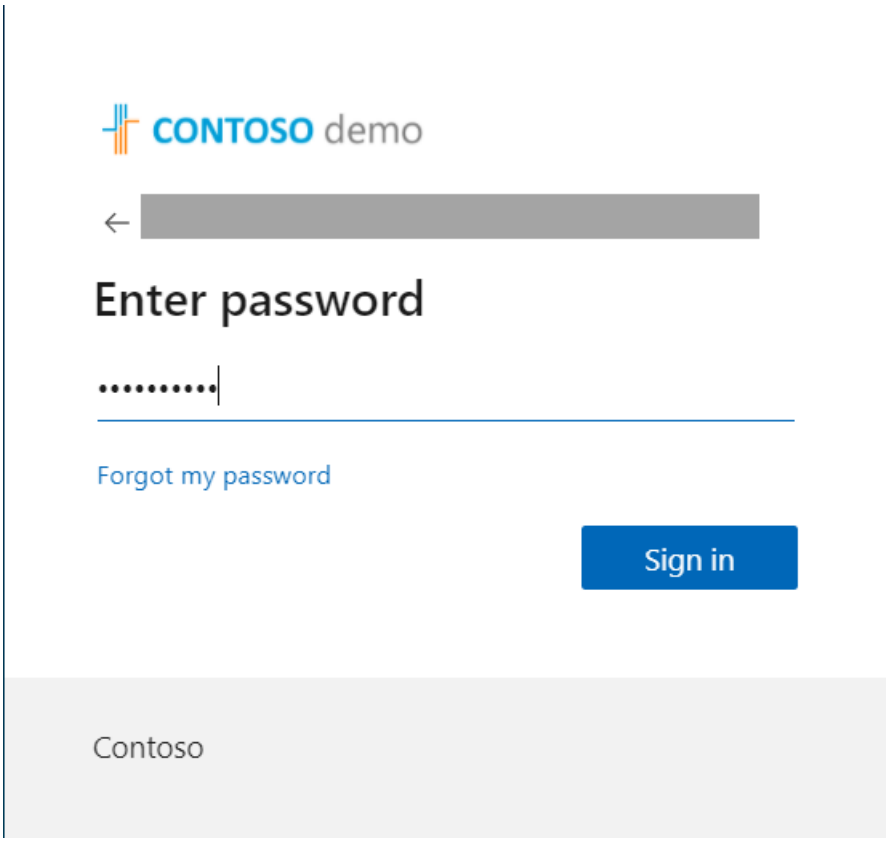
4. Drag the **Centerpiece** object from the **Hierarchy** to the **Session Origin** property.



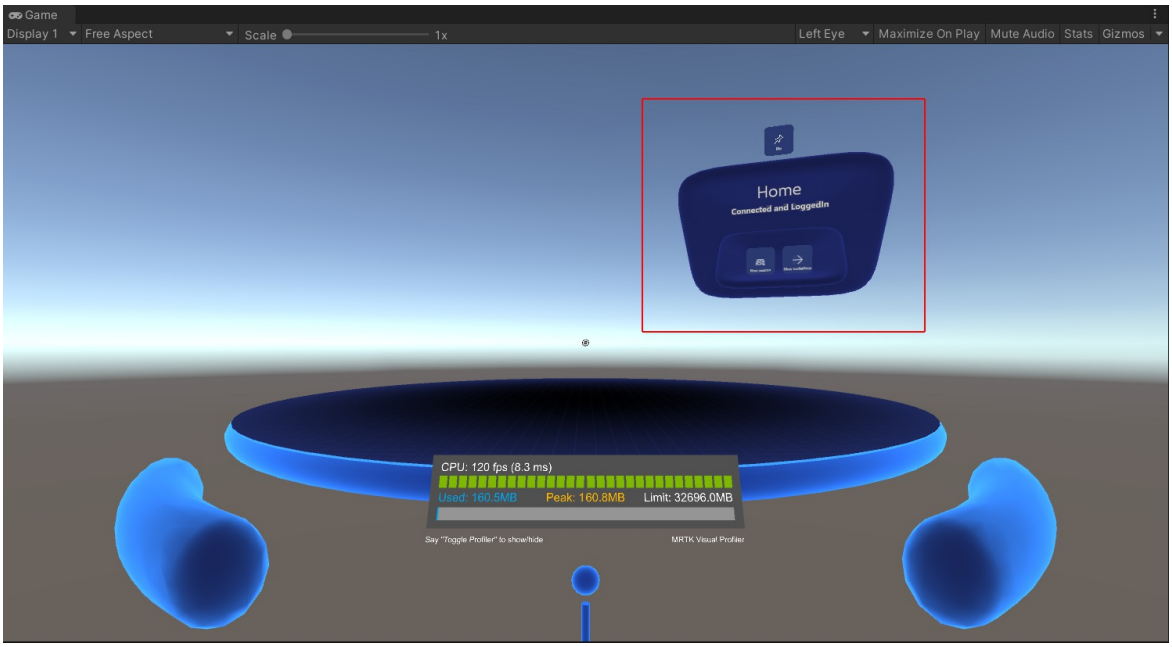
Test in the Unity editor

1. In the Unity toolbar, select the **Play** button to enter play mode.

2. If the **Pick an account** log-in dialog appears in your browser, enter your username and password.



3. In the Unity editor, the project connects to the Mesh Service. After the connection is made, the MRTK UI object appears in the **Game** window. If needed, adjust your view so that the MRK UI object is fully visible and you can read the buttons.



> **NOTE**
>
> You may notice the Diagnostics profiler in the app, which you can toggle on or off by using the speech command **Toggle Diagnostics**. We recommend that you keep the profiler visible most of the time during development to understand when changes to the app may impact performance. For example, HoloLens 2 apps should continuously run at 60 FPS.

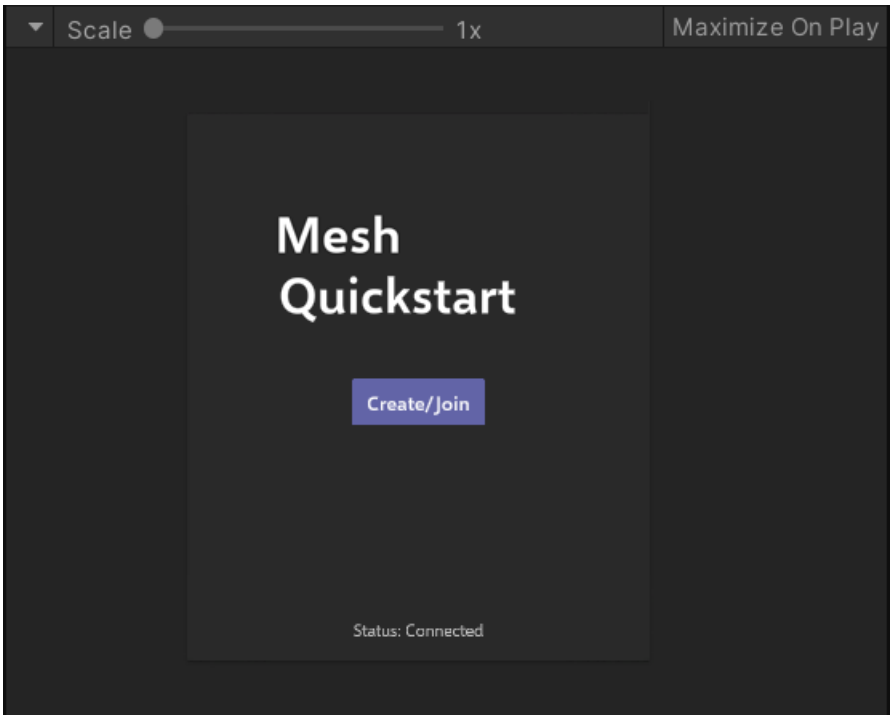
Here are some quick tips for input and navigation:

- To move around the scene, use the **W/A/S/D** keys. This moves the camera forward/left/back/right.

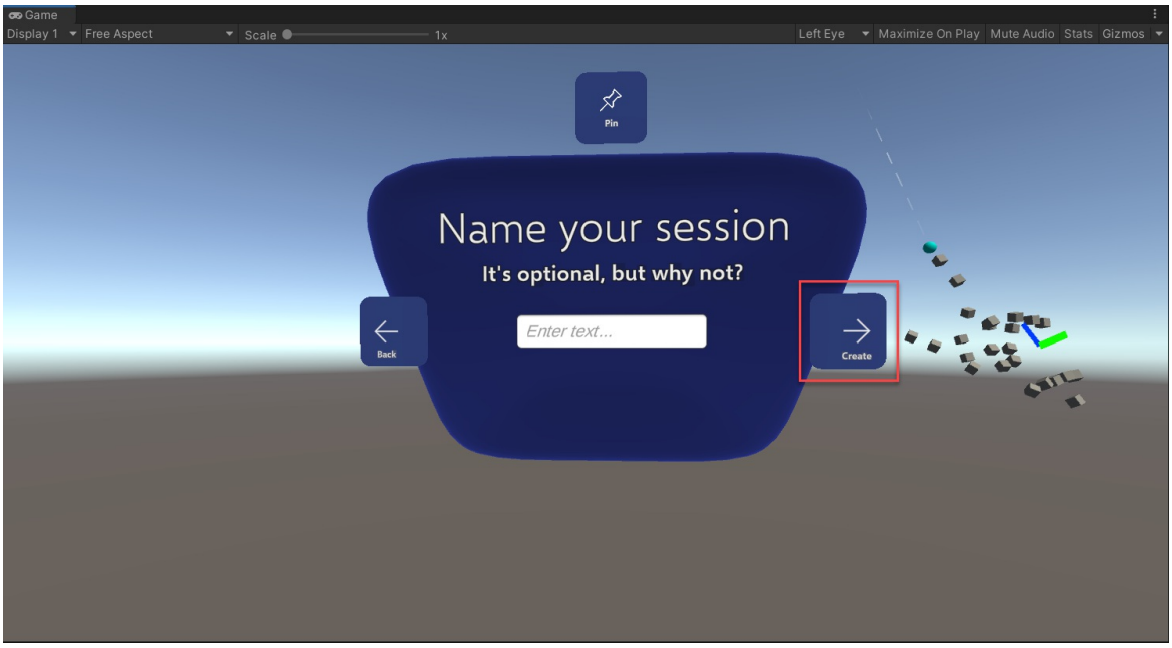
- To move the camera vertically, use the **Q** and **E** keys.
- To make simulated hands appear that you can use to do things like click buttons and drag objects, press and hold the Space bar (right hand) or left Shift key (left hand).
- To move a simulated hand towards or away from you , use the mouse scroll wheel.

4. Press and hold the Space bar to simulate hand input with the right hand.

5. On the **Home** screen of the **MRTK UI** object, select **New Session**. Tip: use the mouse scroll wheel to adjust the hand until it's pointing to the **New Session** button and highlighting it (see below), and then click your left mouse button.



6. On the **Name Your Session** screen, use the hand to select **Create**. If you're testing in the Unity editor, you must skip the step to **Name the Session**. If you're viewing on a HoloLens 2, select the text input field and use the MRTK keyboard to type a name.



> **NOTE**
>
> Note: Text input requires the MRTK keyboard, which is not supported in Unity. Thus, if you're working in Unity, you must skip the steps to **Name the Session** and **Invite Someone**.

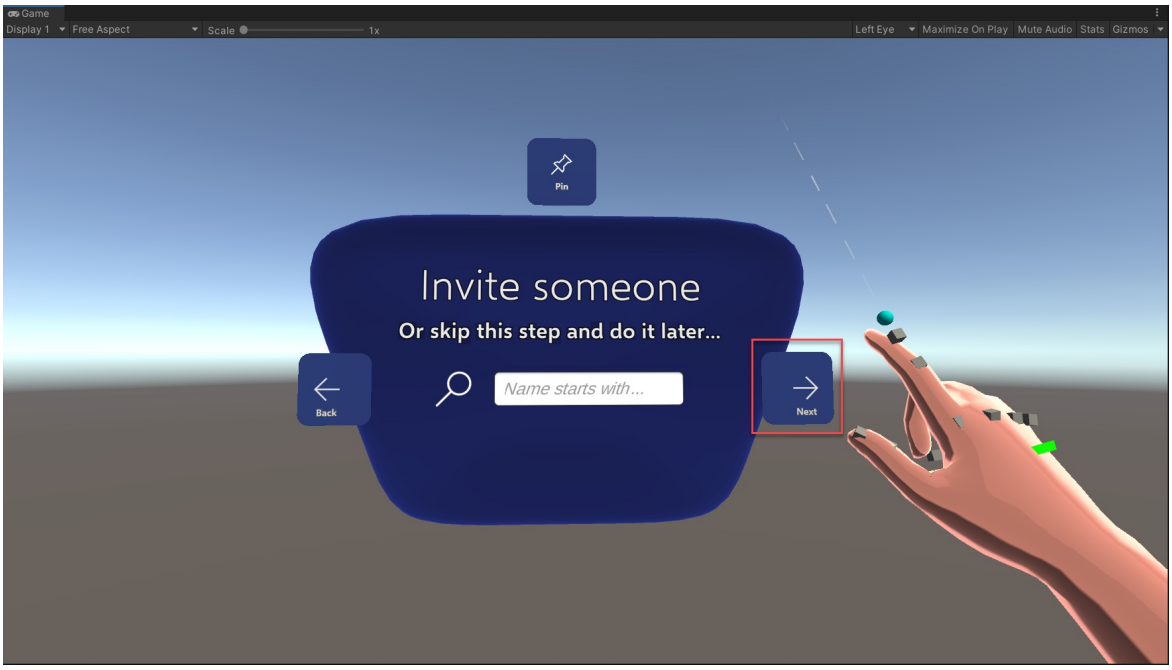
A yellow rotating cube appears which indicates the session is in the process of being created. Once the cube disappears, the UI will progress to the invitation screen indicated by the title showing **Invite Someone**.

In addition, your avatar is spawned. You can observe your avatar either in the **Scene** view or by using the simulated hands provided by MRTK in the **Game** view.

If you want to observe your full avatar as you move around, change your layout so that you can see the Scene and Game views at the same time.

7. On the **Invite Someone** screen, select **Next**. You'll skip this when testing in the Unity editor. If viewing on

a HoloLens 2, select the text input field and use the MRTK keyboard to type a name.



8. Once the **MRTK UI** object reflects an **Active Session**, you're part of an active session.

9. Using either the left or right hand, grab a handle on the Centerpiece object and then drag it to reposition the Centerpiece in the scene.

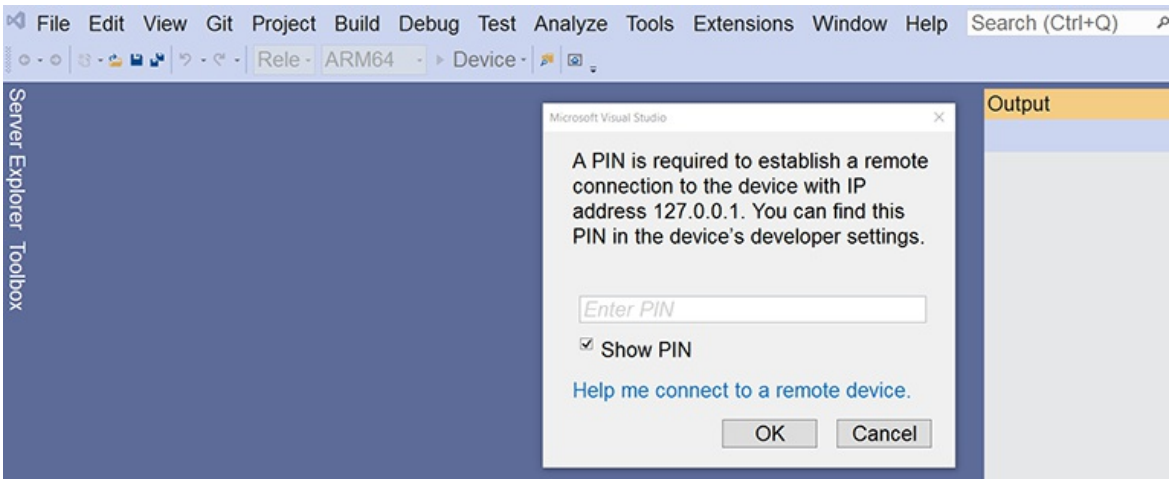
Build and Deploy to HoloLens 2

Prerequisites

Before building to your device, do the following:

1. Confirm that your device is in Developer Mode.

2. Confirm that your device is paired with your development computer. If it's not, you'll see the following dialog box in Visual Studio during the build process:



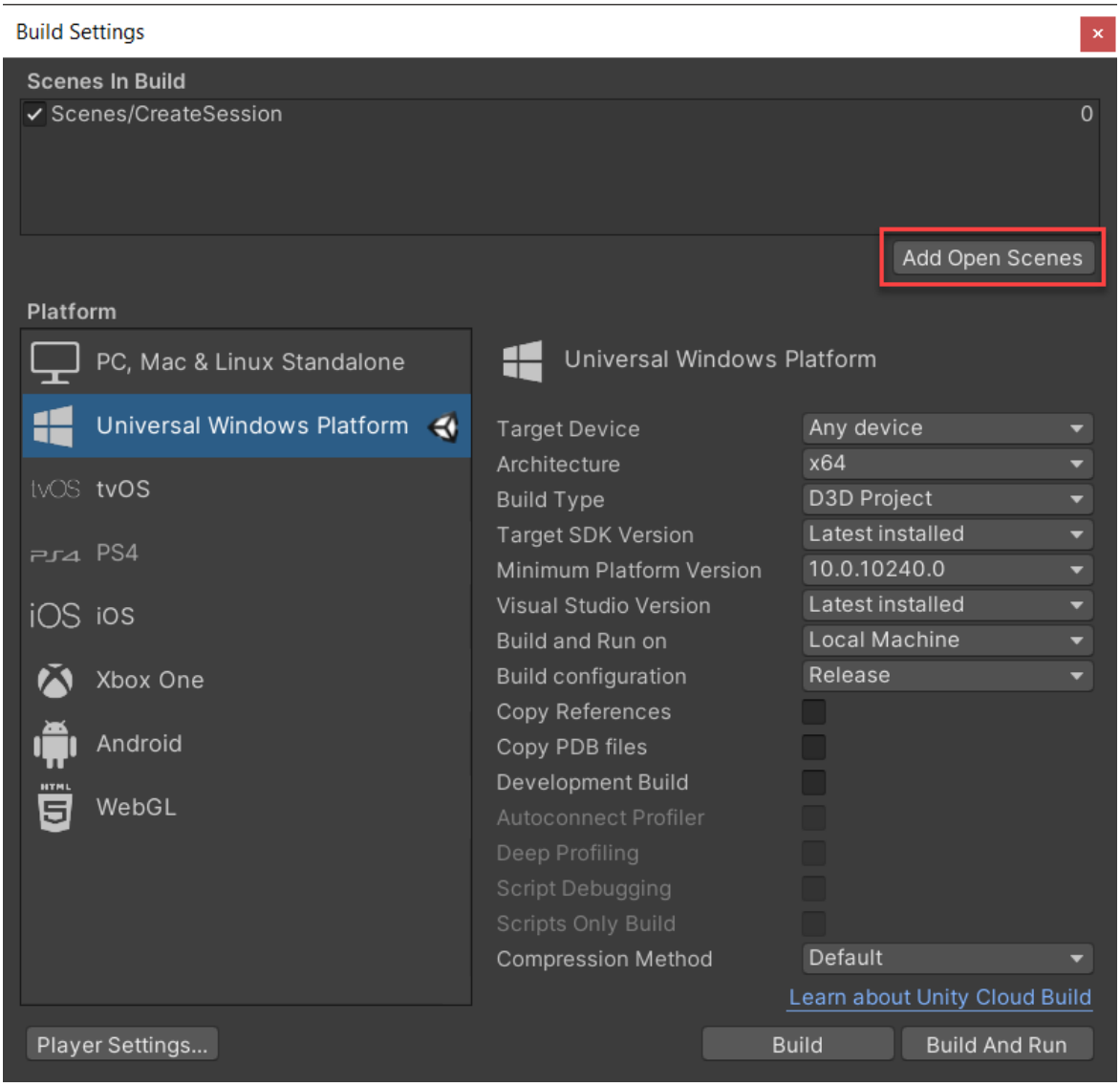
To learn more about these first two steps, see Using Visual Studio to deploy and debug.

3. Review our HoloLens 2 Authentication Tips

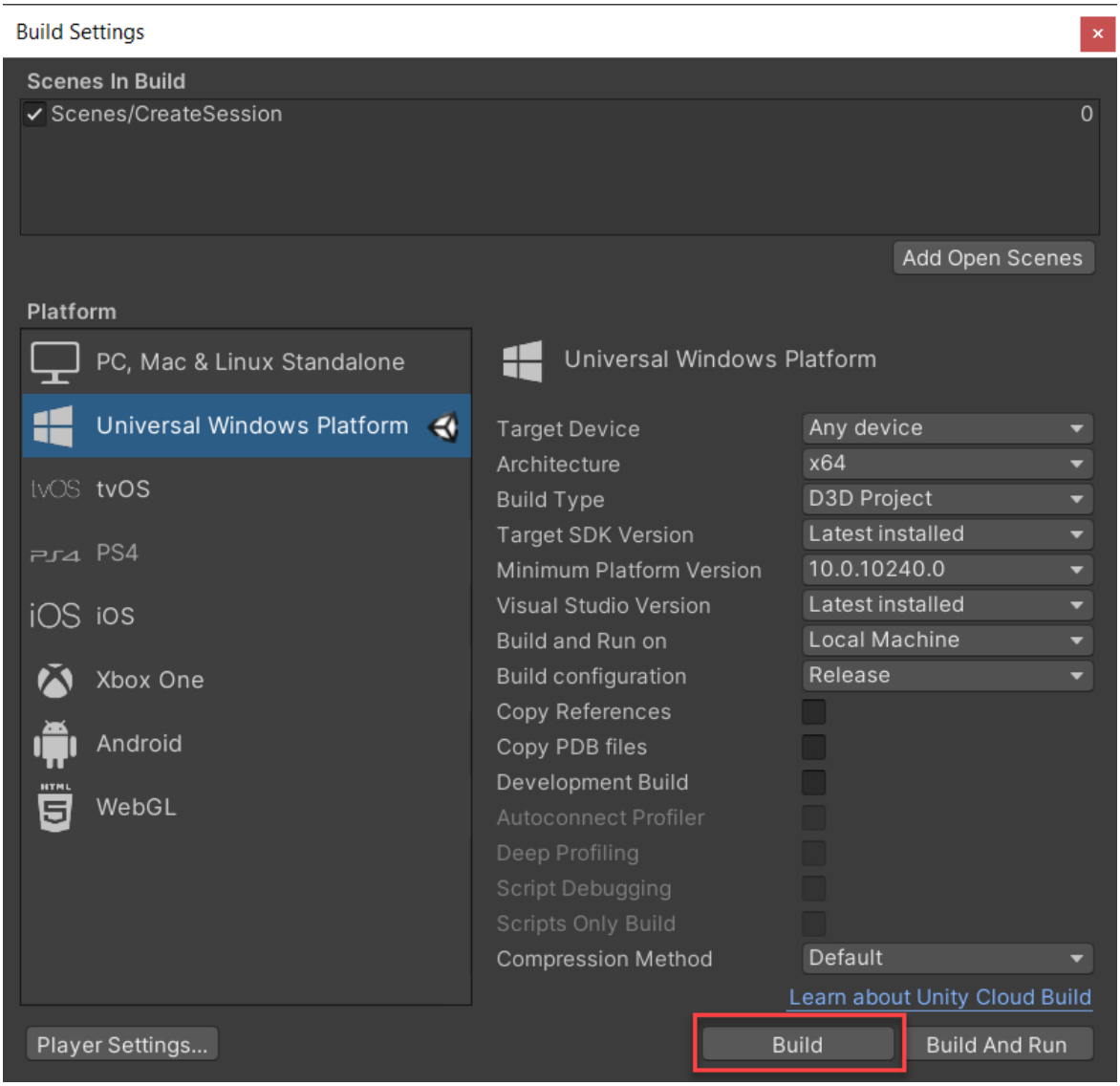
Build your app

1. In the menu bar, select **File** > **Build Settings...**.

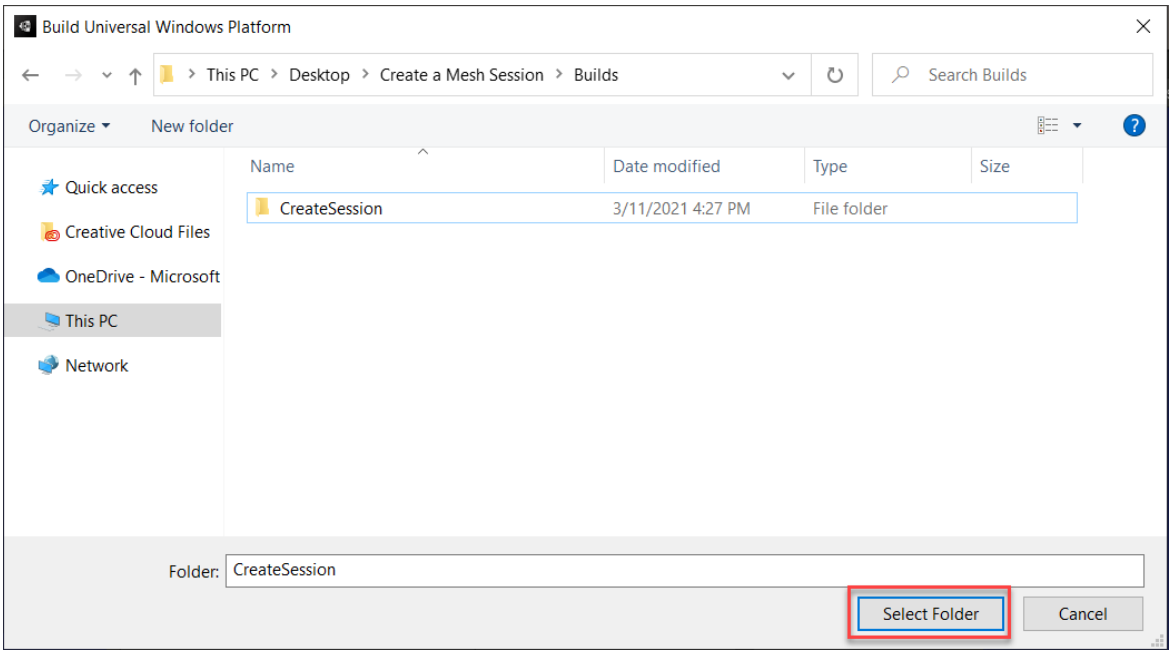
2. Select **Add Open Scenes** to add the scene to the **Scenes in Build** section.



3. In the **Build Settings** window, select **Build**.

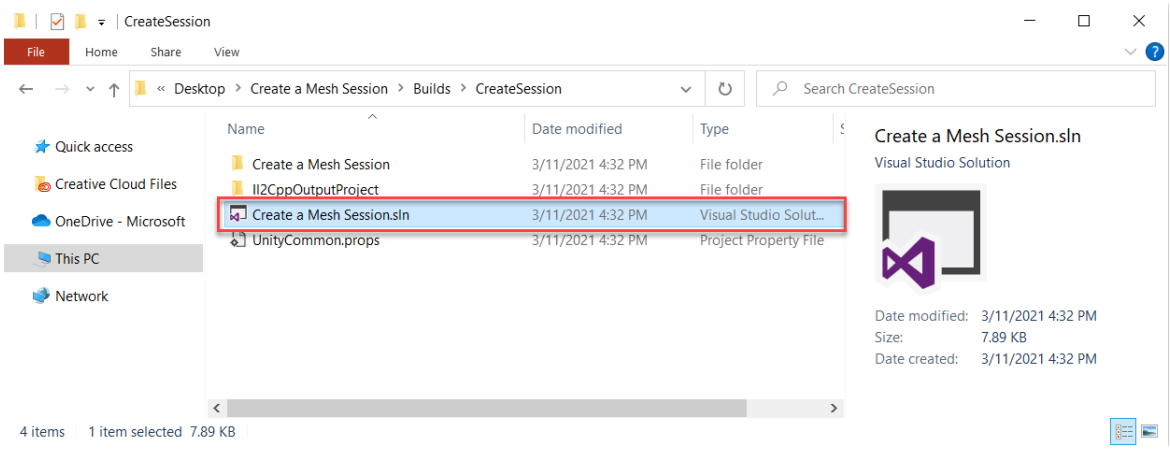


4. In the **Build Universal Windows Platform** dialog, choose a suitable location to store your build (for example, you may want to create a Builds folder in your project). Create a new folder and give it a suitable name (for example, *CreateSession*). Next, select the folder and then click the **Select Folder** button to start the build process.



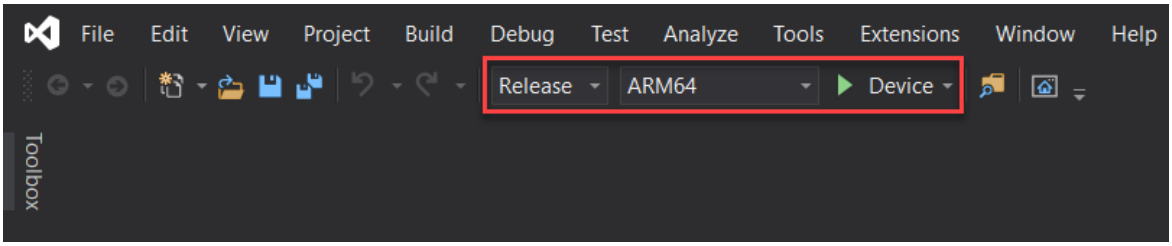
A status bar appears and keeps you updated on your build process.

5. When the build finishes, in the **File Explorer**, navigate to the location where you stored the build. Next, double-click the solution file (i.e. the **.sln** file) to open it in Visual Studio.



6. Configure Visual Studio for HoloLens 2 by selecting the following:

   - **Master** or **Release**
   - **Architecture**: **Arm64**
   - **Target**: **Device**

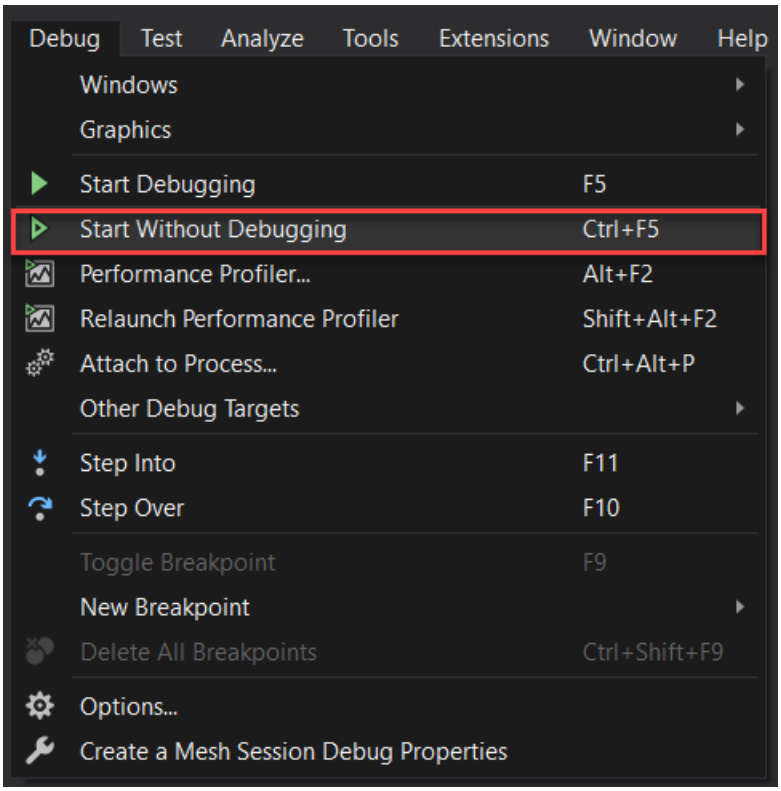


> **NOTE**
>
> Note: If you don't see Device as a target option, you may need to change the startup project for the Visual Studio solution from the IL2CPP project to the UWP project. To do this, in the Solution Explorer, right-click on YourProjectName (Universal Windows), and then select Set as StartUp Project.

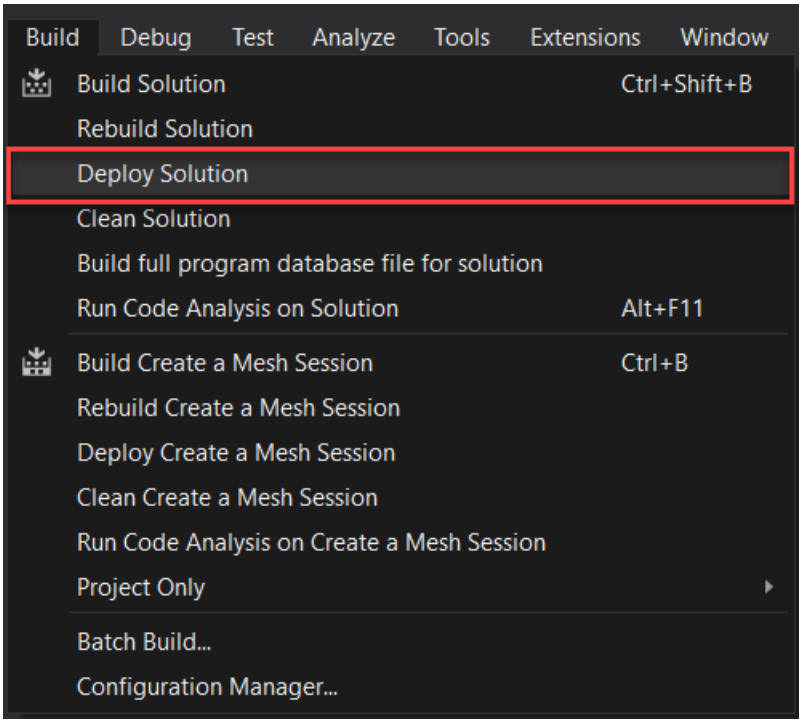
7. Connect your HoloLens 2 to your computer.

8. Build and deploy the app following one of the methods below:

   - **Start automatically without Visual Studio debugger attached**: In the Visual Studio menu, select **Debug** > **Start Without Debugging**.



- **Do not start app automatically**: In the Visual Studio menu, select **Build** > **Deploy Solution**.



Next steps

You've learned how to create and start a Mesh session using the MRTK-UI collaboration tool and add a Centerpiece to the app. Next, learn how to share a 3D model with participants in a session.

Share a Model with Participants

Share a Model with Participants

4/21/2021 • 3 minutes to read • Edit Online

Prerequisites

You will need:

- A Windows computer withVisual Studio 2019or later installed. Your Visual Studio installation must include theUniversal Windows Platform developmentworkload and the Windows 10 SDK (10.0.18362.0 or newer)component.
- An Azure account with an active subscription. Create an account for free.
- An Azure Active Directory tenant that is registered and configured to communicate with your application and the Mesh service. There is a series of topics in the documentation that explain how to set this all up, starting with registering your tenant.You'll need a Client ID and Tenant ID for this tutorial.
- If you are working with commercial users, one or more Work or School accounts.
- A Unity 2019.4.x installation.
- The Mixed Reality Feature tool for Unity. This page explains how to download and use the tool to import packages.
- The latest version of the Mesh private preview zip file.

Caution

Caution: When working on Windows, there is a MAX_PATH limit of 255 characters. Unity is affected by these limits and may fail to compile if any file path is longer than 255 characters. Therefore, we strongly recommend that you save your Unity project as close to the root of the drive as possible and keep file and folder names short.

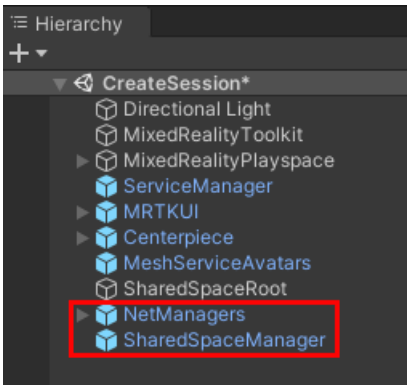
Add a display case to the scene

The Display Case allows participants in a session to share and view .glb models stored on their OneDrive cloud storage. Shared models can be moved by any participant.

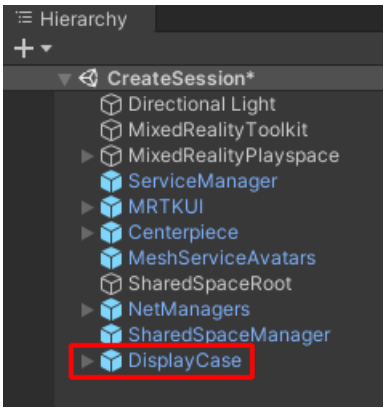
1. In the **Project** window, navigate to this folder:

   Packages > **Mesh Tools for Unity > **Assets** > **Network** > **Prefabs**.

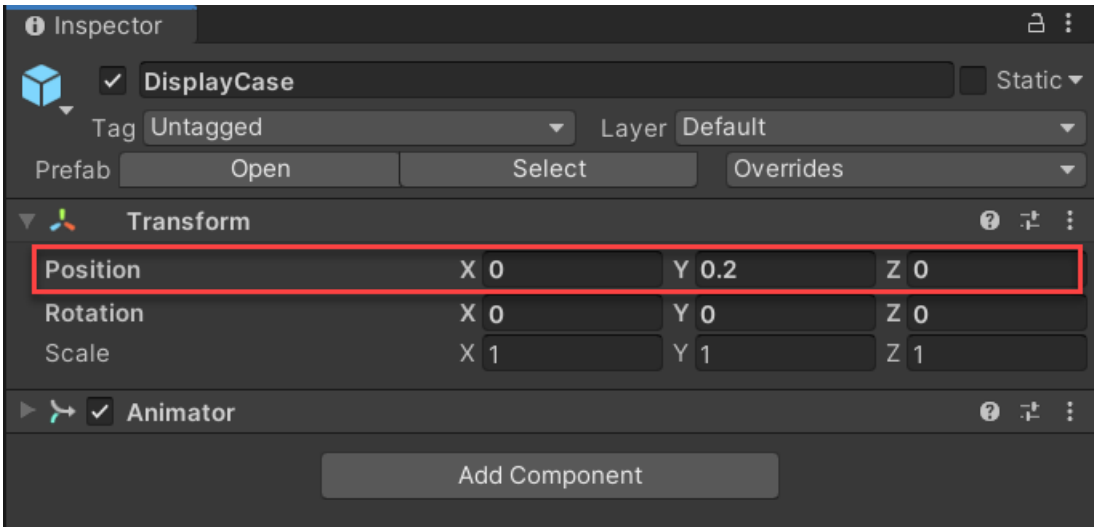
2. Drag the **NetManagers** and **SharedSpaceManager** prefabs to the **Hierarchy**.



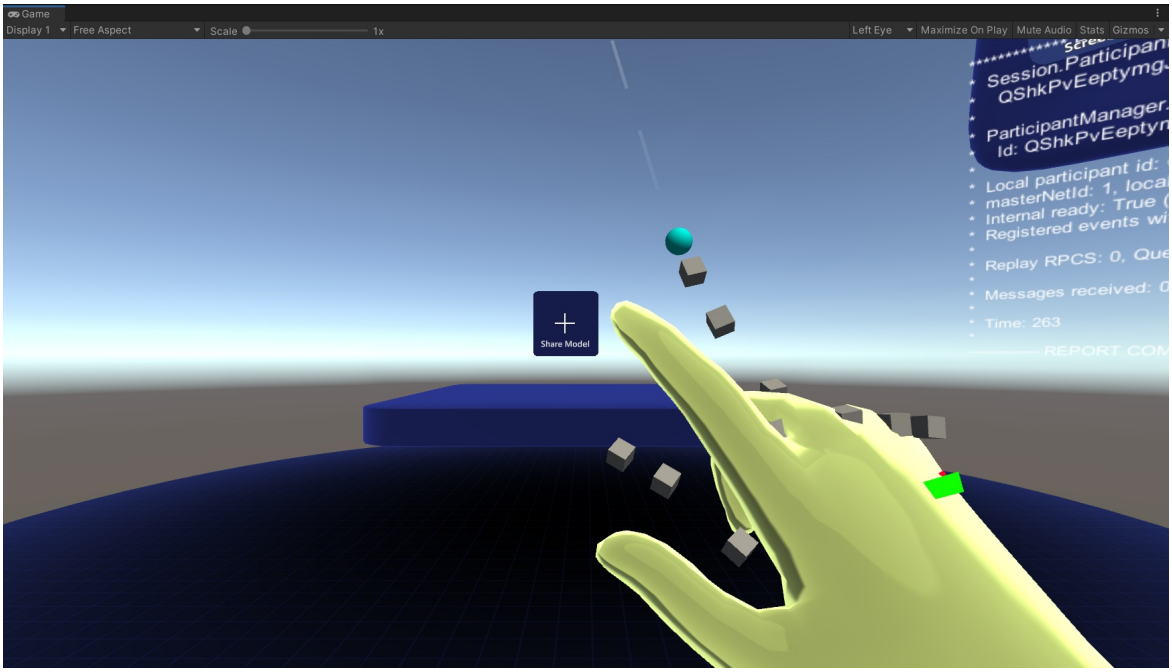
3. Add the **Display Case** prefab (**Packages > Microsoft Mesh UX Tools for Unity > DisplayCase > Prefabs**) to the **Hierarchy**.



4. In the Hierarchy, select the Display Case prefab.
5. In the **Inspector** window, set the **Transform Position** to **0, 0.2, 0**. This places the display case on top of the **Centerpiece** object.

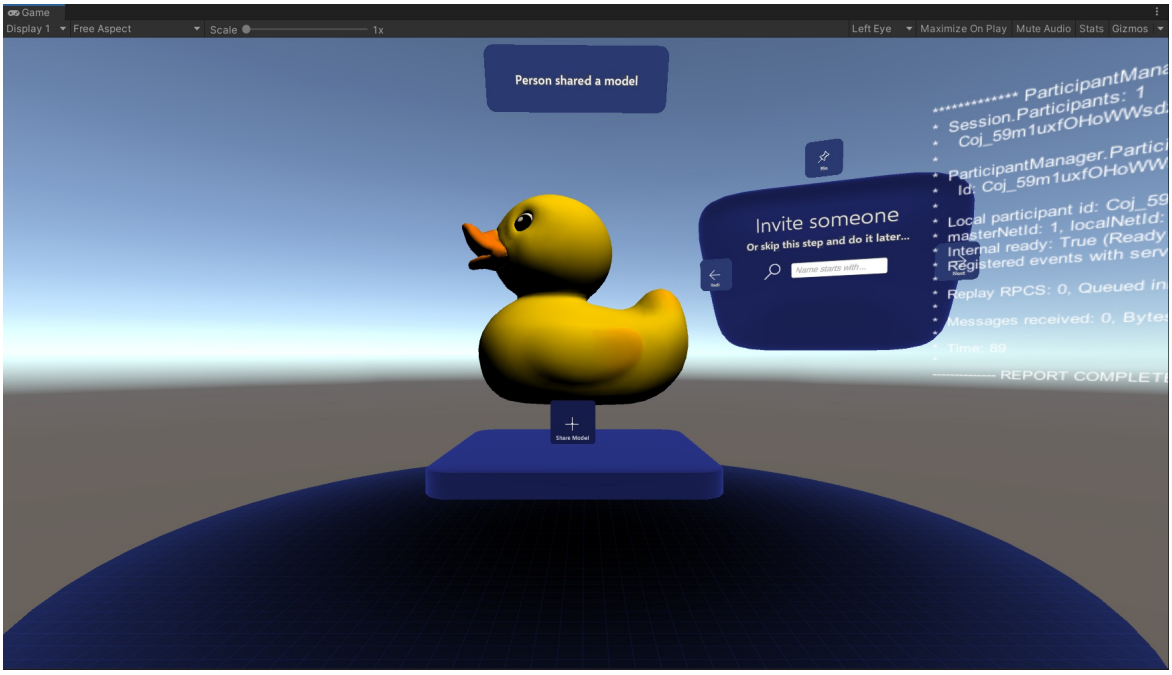


6. In the Unity toolbar, enter Play mode and start a new session.
7. After the session starts, select **Share Model** above the display case platform.

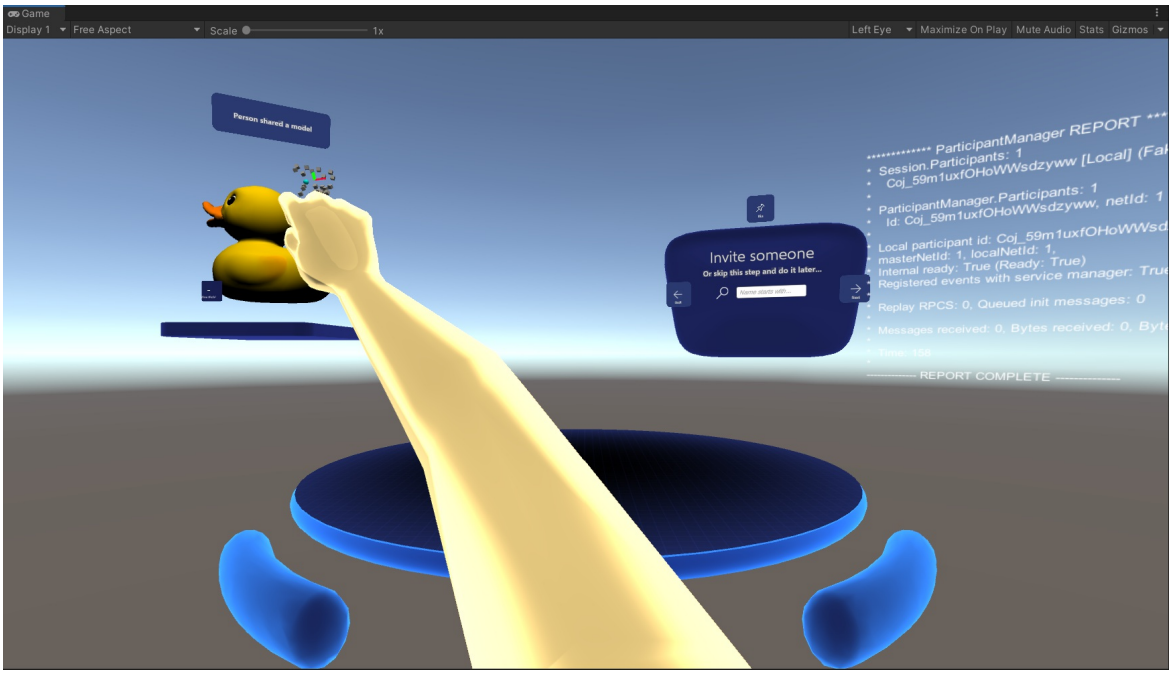


8. Wait for the model menu to refresh with the .glb models started in your OneDrive. Once the list loads, select a model.

9. Once a model is selected, a notification appears which states that a participant has shared a model.



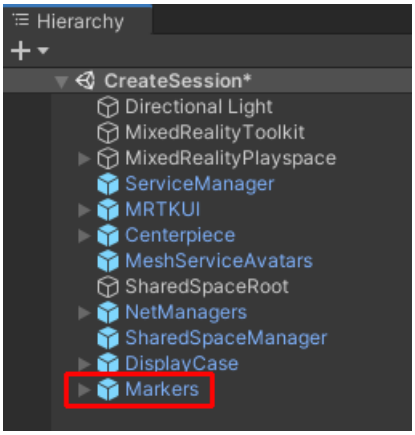
10. Using the Unity in-editor input simulation, press the **Y** key to simulate hand input with the right hand. Grab the model and position to a new location in the scene. Once the model is repositioned, exit Play mode.



Add a Marker to the scene

A Marker is a visual symbol that you can use to draw attention to a location, 3D model, or any other element in a scene. Control of Markers is not limited to the participant who created it; any participant in a session can move, rotate, hide, or remove Markers. Markers can be created and managed with the Markers panel. The Markers panel also provides the ability to create a Note. As you move around, the Markers Panel stays close to you and is always within easy reach. You can also create a marker using the Hand menu.

1. Navigate to the **Packages > Mesh Tools for Unity (UX) > Markers** folder.
2. Drag the **Markers** prefab to the **Hierarchy**.



3. In the Unity toolbar, enter Play mode and start a new session.
4. After the session starts, share a model and reposition it in the scene.
5. Use the Hand menu to access the Markers panel. First, press the **T** key to simulate hand input with the left hand. Next, turn the palm towards you. To do so, hold down the **Ctrl+Shift** keys and the **left mouse button** then drag to rotate the hand.
6. When Hand menu buttons appear, press the **space bar** to simulate hand input with the right hand. Using the right hand, select the top button.

7. When the Markers panel appears, select **Create New Button** to create a new marker.
8. Grab the marker and position next to the model that you previously repositioned.
9. In the Markers panel, select **Create New Note** to create a new note.
10. Grab the note and position next to the marker that you previously repositioned. Once the note is repositioned, exit Play mode.

Note: Text input requires the MRTK keyboard. Given that the MRTK keyboard is not supported in Unity, you will not be able to type a note. However, you can deploy the project to your HoloLens to test text input when creating a note.

Next Steps

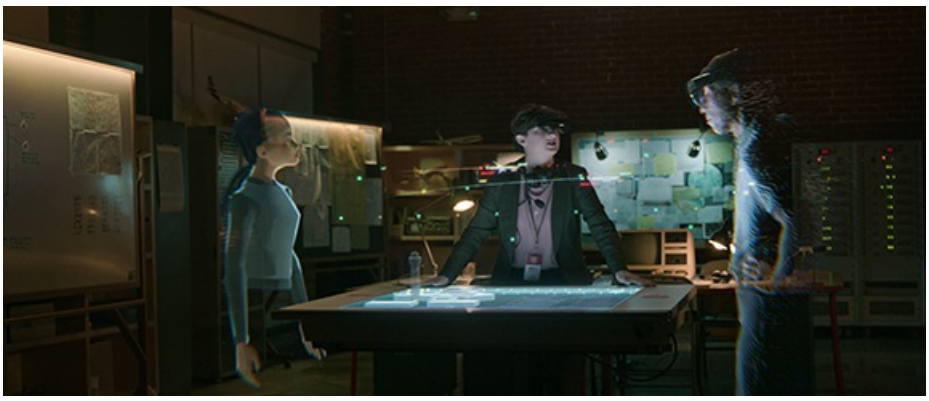
Learn more about the Display Case Mesh tool.

Learn more about the Markers Mesh tool.

Avatars

3/4/2021 • 6 minutes to read • Edit Online

An *avatar* is a virtual representation of a participant. Avatars are one of the most important concepts in Mesh because they represent how people look and move in a session. Understanding what kinds of avatars are available and how they work will help you get the most out of Mesh.



To start using avatars, participants in a Mesh collaboration session create and join a *spatial session*. A spatial session is an extension of a collaboration session which enables high-frequency, bidirectional streaming of spatial data between participants. Once you're inside the spatial session:

- You can spawn and share your avatar with the other participants. Your avatar will replicate your movements in real time when displayed on other participants' devices.
- You can receive other participants' avatars and display them locally.
- You can send and receive spatialized audio.

Communication in a spatial session is encrypted and optimized based on network characteristics, server load, and spatial proximity between participants.

Avatar Synchronization

An avatar contains a *skeleton* that consists of various *joints*. The joints correspond to the parts of the body that may rotate—a shoulder joint, an elbow joint, and so forth. In the 3D world, *joints* are sometimes referred to as *bones*, but we use the term *joints* in this documentation.

The position and rotation of the avatar's joints in 3D space at a particular time is called its *pose*. Mesh synchronizes the poses of avatars between all clients in a spatial session.

In Mesh, you may want to collaborate with your team members on an architectural project, or meet up with friends to play your favorite game, or attend an Augmented Reality party. In any type of Mesh session, the way you appear to others is determined by your avatar.

Currently, there's one avatar provided with the SDK: the standard Mesh avatar. It provides a basic look and cannot be customized.

You want your avatar to function as a plausible virtual human, which builds trust and understanding with your collaborators. To achieve this, Mesh uses *motion models*, a real-time system that provides life-like avatar motion by estimating your pose from limited inputs. A motion model treats each person as a unified whole in diverse sensing and rendering contexts. It fuses information from all available human understanding technologies to estimate the movement of a whole virtual person, including face, eyes, body, and hands.

Motion models have the following qualities:

- They're **robust** in compensating for gaps (time and space) in input. This guarantees consistent, realistic human behavior.
- They're **responsive** to input signals (for example, from hand tracking or head tracking). You can be confident the virtual human is doing what it should be doing.
- They're **adaptable** to sensor/input configuration. If network availability drops, and some signals aren't available to the motion model, it still produces plausible human behavior.

Participants may be joining a session from different kinds of devices: AR, VR, or even just their phone. Different devices will have a range of input capabilities. For example, a HoloLens 2 predicts fingertip locations, while other AR devices may not. In all cases, the device makes use of the best input signals available to provide a consistent representation.

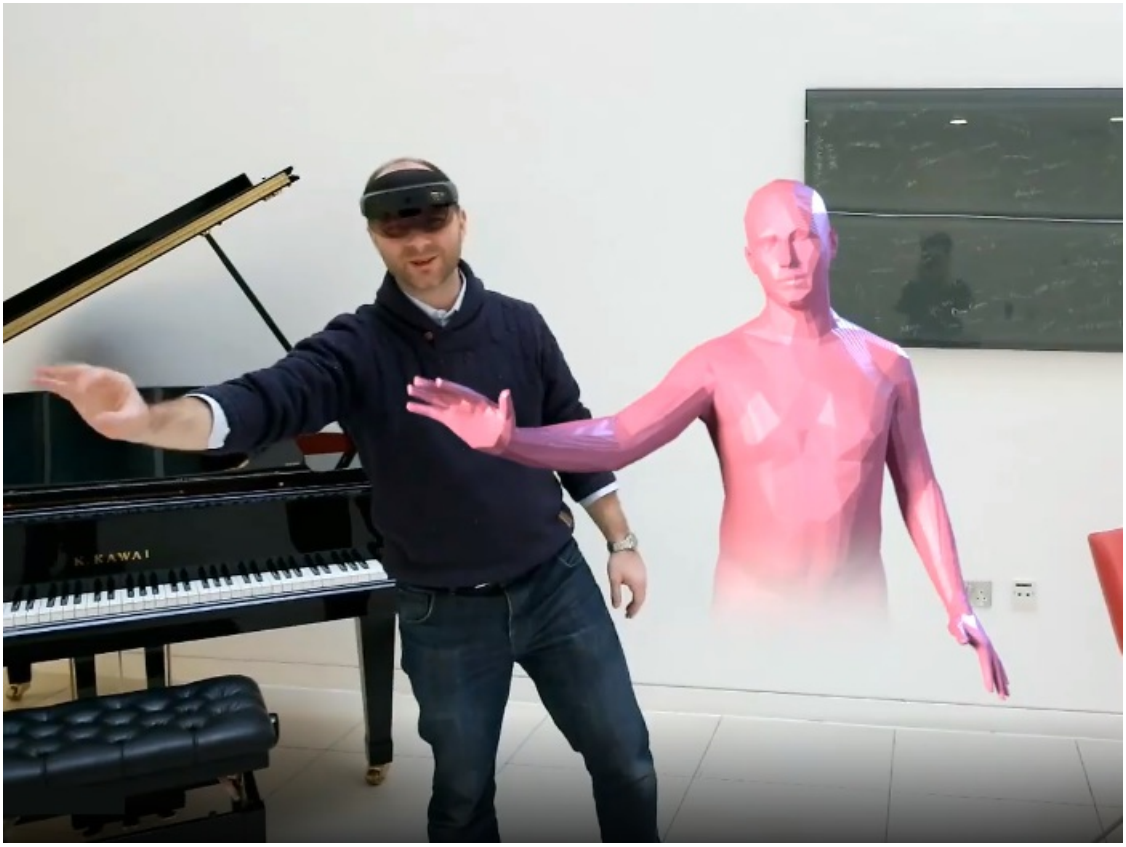
Let's look at an example of how transmitting pose data to avatars works:

Bob and Alice are in a Mesh session in different physical locations. They're each wearing a device, such as the HoloLens 2, that can capture their head and hand positions and record their voice.

The sensors on Bob's device capture the poses of Bob's head and hands. Mesh sends the sensor data to the server, which uses the data to construct a full body of Bob and estimate its pose. This pose contains joints that correspond to the parts of Bob that may rotate—a shoulder joint, an elbow joint, and so on.

In the image below, Bob's HoloLens 2 detects the transforms of Bob's head and hands, which are indicated by the three-color axes. To Bob's left, there is an upper body visualization of what Mesh generates internally when the motion model system constructs Bob's representation.

As Bob moves around, the internal representation of Bob's body is continually updated.



After the server constructs Bob's representation, it sends its pose of its internal skeleton to Alice's device, which contains an avatar that is identified with Bob. The pose received from the server is applied to the skeleton of this avatar, animating it.

Because this process occurs many times per second, if Bob moves, the matching movement of Bob's avatar appears smooth to Alice. Even though Alice can't see the physical Bob, she can be confident that his avatar is closely duplicating his movements in real time.

If you want to quickly add presence to your application with humanoid representation, the standard avatar is an

easy, out-of-the-box solution.

A broad range of input signals can be used to generate motion models, including:

- A head tracking signal (HoloLens, VR headset).
- Hand tracking, (HoloLens, VR controllers)
- Fingertip tracking (HoloLens 2)
- Eye gaze (HoloLens 2)
- Audio, which drives lip motion.

Local-only avatar

Avatar can be created locally for scenarios like looking into a mirror and customizing your avatar without joining Mesh and a spatial session. After creating an avatar using its constructor `Avatar(Skeleton)` you can call `Avatar.InjectInput()` and `Avatar.TryGetPose()` in update loop to get the pose estimated by Motion Models subsystem.

```
void Start()
{
    // Get the canonical skeleton, a built-in skeleton available with Mesh
    var skeleton = Skeleton.GetCanonical();
    // Create an avatar instance for local-only simulation
    m_avatar = new Avatar(skeleton);
}
void Update()
{
    // Inject sensor data (head and hand tracking, ...) into the avatar
    ExternalInputData inputData = new ExternalInputData();
    inputData.AddHead(...);
    m_avatar.InjectInput(inputData);
    // Retrieve the estimated pose (skeleton joints transforms)
    if (m_avatar.TryGetPose(lod: 0, out Vector3 rootPosition, out Quaternion[] poseJoints))
    {
        // Animate the avatar by applying the new pose to the skeleton
    }
}
```

Examples

Joining the spatial session

```
var options = new SpatialSessionOptions
{
    ExpectedFrameRate = 60,
    IsFrameRateFixed = false,
};
// All we need is an active collaboration session.
var spatialSession = await SpatialSession.CreateFromSessionAsync(collaborationSession, options);
// To leave the spatial session, dispose the object
spatialSession.Dispose();
// Note that you have to be part of the collaboration session the whole time you are using a spatial session
```

Sharing your own pose

```
void Start()
{
    // Create avatar for the current participant
    selfAvatar = spatialSession.SetAvatar(collaborationSession.CurrentParticipant, Skeleton.GetCanonical());
}


void Update()
{
    var timestamp = Time.time;
    var headTransform = Camera.main.transform;
    ExternalInputData inputData = new ExternalInputData();
    inputData.AddHead(timestamp, new PoseTransform() {
        Position = headTransform.Position,
        Rotation = headTransform.Rotation});
    // AddHand/Fingertips/EyeGaze...
    selfAvatar.InjectInput(inputData);
}
```

Receiving other participant's poses

```
void Start()
{
    // Subscribe to changes in spatial session participants
    spatialSession.ParticipantChanged += (object sender, SpatialParticipantChangedEventArgs args) =>
    {
        if (args.Operation == SpatialParticipantChangedOperation.ParticipantAdded)
        {
            // Instantiate an avatar for the newly joining participant
            spatialSession.SetAvatar(args.Participant, Skeleton.GetCanonical());
        }

    }
}

void Update()
{
    // Advancing time on a spatial session updates the Mesh avatars poses
    spatialSession.AdvanceTimeTo(Time.time);

    foreach (var participant in collaborationSession.Participants)
    {
        var avatar = spatialSession.GetParticipantAvatar(participant);
        if (avatar == nullptr)
        {
            continue;
        }

        // Get the received pose from Mesh avatar.
        if (avatar.TryGetPose(lod: 0, out Vector3 rootPosition, out Quaternion[] poseJointsBuffer))
        {
            // Animate the avatar by applying the new pose to the skeleton
        }
    }
}
```

For details on how to apply the pose please see the sample avatar in Avatars toolkit.

Next steps

Try our Unity or C++ quickstarts, or explore some of Mesh's core concepts, starting with the Client object.

Unity quickstart

C++ quickstart

Client

See Also

- Session origin
- Spatial audio
- Avatar package

Client

3/29/2021 • 2 minutes to read • Edit Online

Overview

The entry point for the Mesh platform is a client object of type `CollaborationClient`. This client can be configured with information about connection specifics and the user identity of the person who owns the client.

Once you have a client, the following major objects and operations are available:

- Connections can be customized with the `Configuration` property.
- The identity of the user can be accessed with the `Identity` property.
- Sessions can be created, joined or shared through the `Sessions` property.
  - Other endpoints are represented as `Participants` with associated users.
  - The application can share simple state with others with the `State` property.
  - Use the `CreateTransportAsync` method to access data channels and control audio and media.

Each of these topics is covered with example code in the following sections.

Setting up a connection

Creating a client is the first task when using the Mesh SDK. The following code declares and initializes a client and ensures they are logged in with a default sign-in experience.

```
// 1. Create a new client with a default configuration.
CollaborationClient client = new CollaborationClient();

// 2. Subscribe to the authentication TokenRequested event
client.TokenRequired += new TokenRequiredDelegate(object sender, TokenRequiredEventArgs args) {
    var deferral = args.GetDeferral();
    // 3. Retrieve an authentication token for the Mesh scope.
    var token = await signin.GetTokenAsync(CollaborationClient.GetTokenScopes());
    args.AuthenticationToken = token;
    deferral.Complete();
};

// - This is how you know if you lose connectivity to core services.
client.ConnectionStateChanged += new ConnectionStateChangedDelegate(object sender,
ConnectionStateChangedEventArgs args) {
    if (client.ConnectionStatus != ConnectionStatus.Connected) {
        // 4. Set state in your application to reflect that there is no Mesh connection.
    }
};

// 5. Check that the client connection status is active and connect if not.
if (client.ConnectionStatus != ConnectionStatus.Connected) {
    await client.ConnectAsync();
}
```

Getting diagnostics

You can gather diagnostic information by setting your client's `LogLevel` property and subscribing to the `LogDelegate` event.

```
// 1. Change clients log level.
client.LogLevel = ClientLogLevel.Information;

// 2. Subscribe to log event handler.
client.Logged += new LogDelegate((object sender, LogEventArgs args) => {
    // This event handler may be called on any thread, unlike other callbacks.
    Debugger.WriteLine(args.Message);
});
```

You can select from the following options when setting the client's `LogLevel`:

- `None`: Specifies that logging should not write any messages.
- `Error`: Specifies logs that indicate when the current flow of execution stops due to a failure.
- `Warning`: Specifies logs that highlight an abnormal or unexpected event, but do not otherwise cause execution to stop.
- `Information`: Specifies logs that track the general activity of the client.
- `Debug`: Specifies logs used for investigation during development.
- `All`: Specifies that all messages should be logged.

Showing collaborator status

The following properties can be accessed from the client object:

> **NOTE**
>
> These are currently unsupported in the SDK.

```
// User's avatar
Avatar PreviewAvatar { get; }

// Online/offline
PresenceStatus Presence { get; }
```

Next steps

Identity < Sessions

Identity

3/4/2021 • 2 minutes to read • Edit Online

Overview

Mesh clients always connect with a signed-in user. Users can sign in with a Microsoft Account (MSA) or an Azure Active Directory (AAD) user.

The identity of the user is represented by the `Identity` property of the `CollaborationClient` object. Applications must subscribe to the `TokenRequired` event and provide an authentication token during the callback.

Next steps

Session

Sessions

3/18/2021 • 2 minutes to read • Edit Online

Overview

Mesh sessions are the mixed reality counterpart of a group call between friends over the phone. Any Mesh user can create a session and invite others; there are no special permissions required. These sessions manage the active participants and allow them to communicate with each other securely and efficiently.

Using Mesh sessions, participants can share media such as audio and video.

Creating a session

The `Sessions` property of the `CollaborationClient` object supports methods to create, find and join sessions:

```
// 1. Set options to help identify and filter this session.
SessionOptions options = new SessionOptions()
{
    Name = "Marcelo's work session",
};

// 2. Create an asynchronous session using the current client.
CollaborationSession session = await client.Sessions.CreateAsync(options);
```

`SessionOptions` specifies information that can help others find a session and decide to join:

- `Name` is a required property that's used for user display.

When `CreateAsync` is invoked, the session manager tells Mesh to create a session on behalf of the user and immediately join it. Other users can then find this session and join whenever they want.

Sending invitations

After a session is created, you'll want to use the client to invite others to join in. Use `InviteToSessionAsync` to send a notification to other collaborators to join a specified session.

```
// Send an invitation with the client Session object.
// - Pass in the contacts you want to invite and the session reference.
var invitee = CollaborationClient.FormatIdentifierFromOrganizationAccount(tenantId, userObjectId);

var contacts = new string[] { invitee };
await client.Sessions.InviteToSessionAsync(contacts, session.SessionReference);
```

> **NOTE**
>
> A `CollaborationSession` object is used when a session has been joined. When referring to a session in a different context--for example, when discovering sessions that can be joined--a `SessionReference` object is used instead. The reference holds an identifier along with the name for the sessions.

Joining a session

In order to join a session, you need to subscribe your client to the `OnSessionInviteReceived` event, and then

explicitly accept the invitation using `args.Invite.ResolveAsync`:

```
// 1. Set up a handler to receive invitations.
client.OnSessionInviteReceived += new SessionInviteReceivedDelegate(async (sender, args) =>
{
    // 2. Join the session asynchronously.
    CurrentSession = await args.Invite.ResolveAsync(InviteResolution.Accept);
});
```

Displaying participants

```
// Subscribe to event and handle changes through a delegate.
CurrentSession.ParticipantsChanged += new ParticipantsChangedDelegate((sender, args) => {

});
```

Monitoring connection changes

```
// 1. Monitor for connection changes with an event handler.
CurrentSession.ConnectionChanged += new ConnectionChangedDelegate(sender, args) {
    // 2. Check for specific changes to connection status.
    if (client.ConnectionStatus != ConnectionStatus.Connected) {

    }
};
```

Working with session state

Subscribe to the `StateChanged` event to monitor changes to your session:

```
// 1. Subscribe to event and handle changes through a delegate.
CurrentSession.State.StateChanged += new SessionStateChangedDelegate((sender, args) => {

});
```

You can also save common data types to your session state for the duration of the session. However, these will not carry over or persist.

```
// 2. Save, remove, or query variables from short-term storage.
// - `false` result usually means this was a no-op because someone beat you to it.
int value = await CurrentSession.State.IncrementValueAsync("foo", 1);

// Add values
await CurrentSession.State.AddToListAsync("foo", "value");
bool added = await CurrentSession.State.AddToSetAsync("foo", "value");

// Remove values
bool removed = await CurrentSession.State.RemoveFromListAsync("foo", "value");
bool removed = await CurrentSession.State.RemoveFromSetAsync("foo", "value");

// Get values
string[] values = await CurrentSession.State.GetValuesAsync("foo");
string value = await CurrentSession.State.GetValueAsync("foo");

// Set values
await CurrentSession.State.SetValueAsync("foo", "new value");

// Delete values
bool deleted = await CurrentSession.State.DeleteKeyAsync("foo");
```

Leaving a session

```
await CurrentSession.CloseAsync();
CurrentSession = null;
```

Next steps

Session Origin

See also

- Client

Session Origin

4/7/2021 • 15 minutes to read • Edit Online

In a Mesh-enabled application on your chosen platform you can create a session, invite participants who may be at your physical location or in different locations, and share audio, video, or data. By creating a *spatial session*, you can also transmit spatial audio and avatar poses between participants, and call up holograms that all participants view. In this article, we'll explain how participants share holograms that they perceive to be located at the same locations within a shared coordinate system.

This session assumes familiarity with the concepts of coordinate systems and frames of reference.

Session coordinate system and origin

Every Mesh spatial session defines a 3D coordinate system shared by all session participants. When participants share avatars and other holographic 3D content in the session, they express the shared positions and orientations with respect to this system. The system follows the conventions of Windows spatial coordinate systems (right-handed; Y-axis is aligned to gravity and points up; one unit corresponds to 1 meter). All participants and devices use the session coordinate system for shared content, and the position/orientation of each hologram in this system is the same for any participant/device that queries it.

Each instance of the Mesh-enabled holographic app, running for a participant on a specific device, typically also defines a local 3D coordinate system. This system is generally provided by the specific Mixed Reality platform that the app is based on (Unity, Windows Mixed Reality, ARCore, and so forth). For example, the local rendering engine uses a predefined coordinate system to express the transforms of the objects to be rendered; tracking data from sensors is also usually expressed with respect to this system. Differently from the session coordinate system, the local coordinate system is only used by the local app instance, and it is meaningless for other app instances.

In order to collaborate in the shared session coordinate system, each participant/app instance must be able to convert:

- the coordinates of holographic content created locally from the local coordinate system to the session one, so that they can be correctly shared with the other participants
- the positions and orientations contained in the input sensor data from the local coordinate system to the session one, for the same reason
- the coordinates of shared content from the session coordinate system to the local one, so that the content can be rendered locally.

Each app instance must therefore establish a mapping between the session coordinate system and the local coordinate system. This mapping can be usually described as a 3D transform matrix between the origin and axes of the local coordinate system (*local origin* in short) to the origin and axes of the session coordinate system (*session origin* in short). Once chosen, this transform matrix should be used to transform the coordinates of avatars, input poses, and other holographic content between the two systems.

Placing the session origin

Choosing the transform matrix is equivalent to choosing the position and orientation of the session origin in the local coordinate system; in practice, "pinning" the session origin (and, with it, all the session content) to a point in the participant's physical space. Mesh defines this process, in short, *placing* the session origin in the local system.

There are various strategies for having the participants in a session place the session origin, and how it's done depends on the app and its scenarios. An app might let each user manipulate and place the session origin freely, it might place the origin in a fixed position for each participant, or it might use more complex logic based on the participants' surrounding environment and their proximity to each other. We provide examples for a few basic scenarios below.

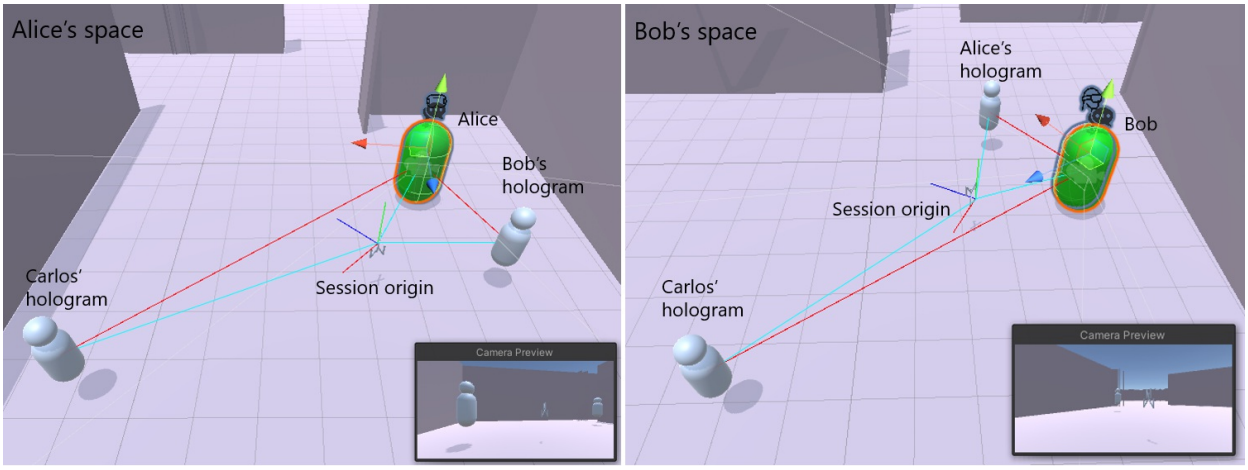
In Unity, you can attach the `MeshSessionOrigin` component from the Mesh Prototype Unity Toolkit Avatar package to a game object, and then move/let the user manipulate that game object, to place the session origin in the scene. The position of the remote avatars and the local input poses will be automatically adjusted according to the game object transform.

Example: independent participants

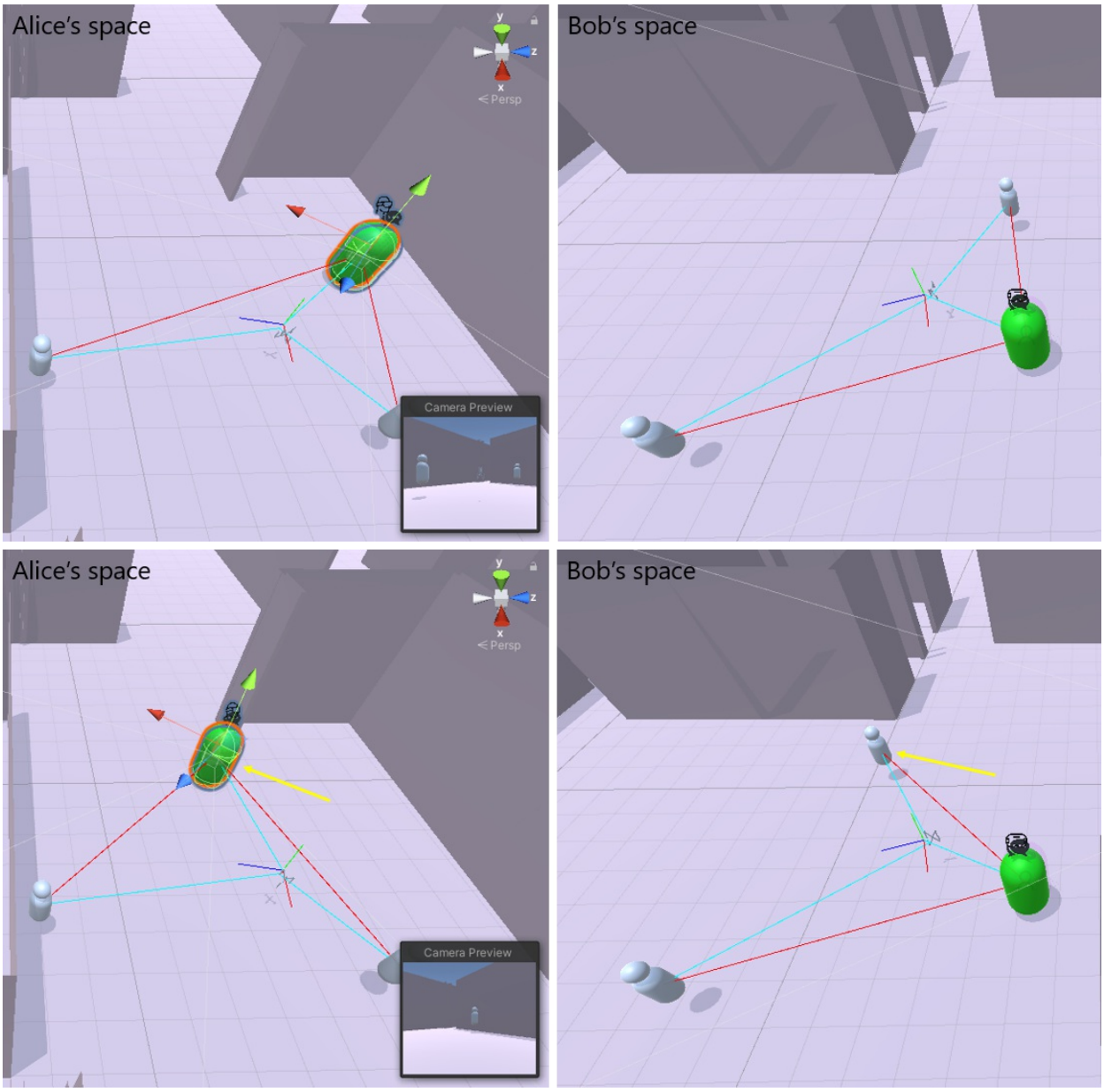
In the simplest case, every participant places the session origin independently in the local space. This strategy can be adopted regardless of whether participants are in proximity of each other. The session origin can be placed wherever the participant (or the app) wants it to be—on top of a desk, in the middle of the floor, and so on. After each app instance has placed the origin, all of them can render the same set of avatars and content (possibly from a different perspective, depending on the device position) superimposed on the surrounding physical space. In this case, no participant needs to know about where the other participants have placed the session origin.

For this example, imagine you are in a session with someone in a different physical location; they're in their home, and you are in your office. They place the session origin on the top of a table in their home. You place the session origin in the middle of your desktop in your office. If they call up a hologram and place it at (0,0,0) in the session coordinate system; it appears for them at the top of their table, and it appears for you in the middle of your desk. If the other participant moves the hologram on their table five units to the right on the y axis in the session coordinate frame, this change is synchronized with you and the hologram in your view moves on your desktop five units to the right on the y axis in the session coordinate frame.

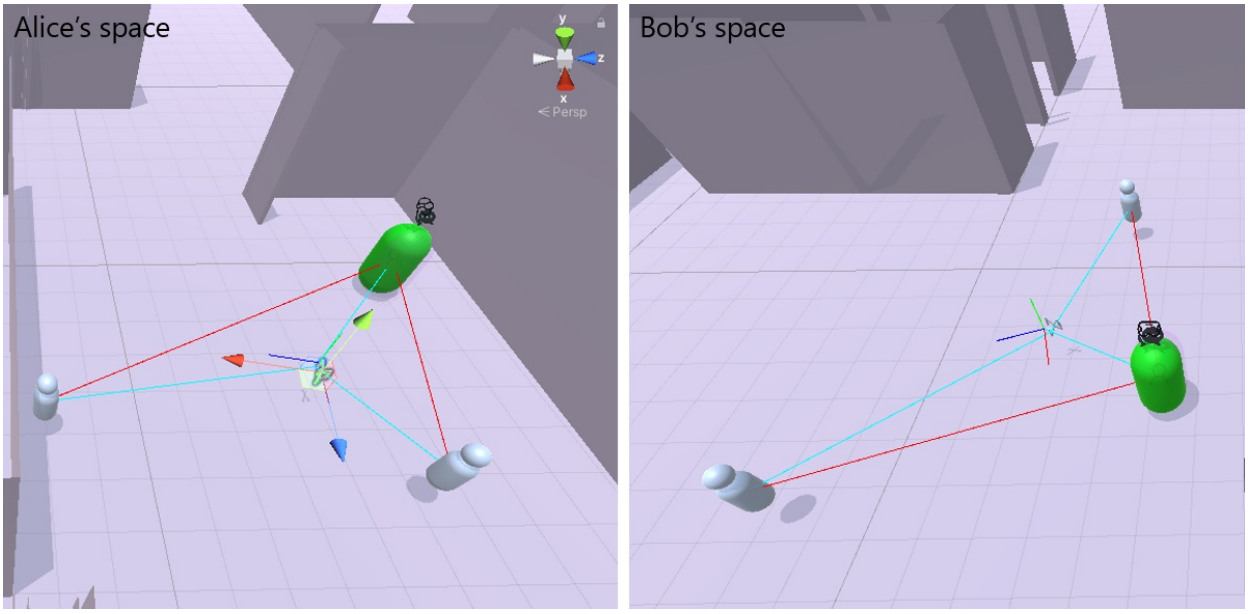
In the picture below, Alice, Bob, and Carlos have joined a session, each from separate locations.

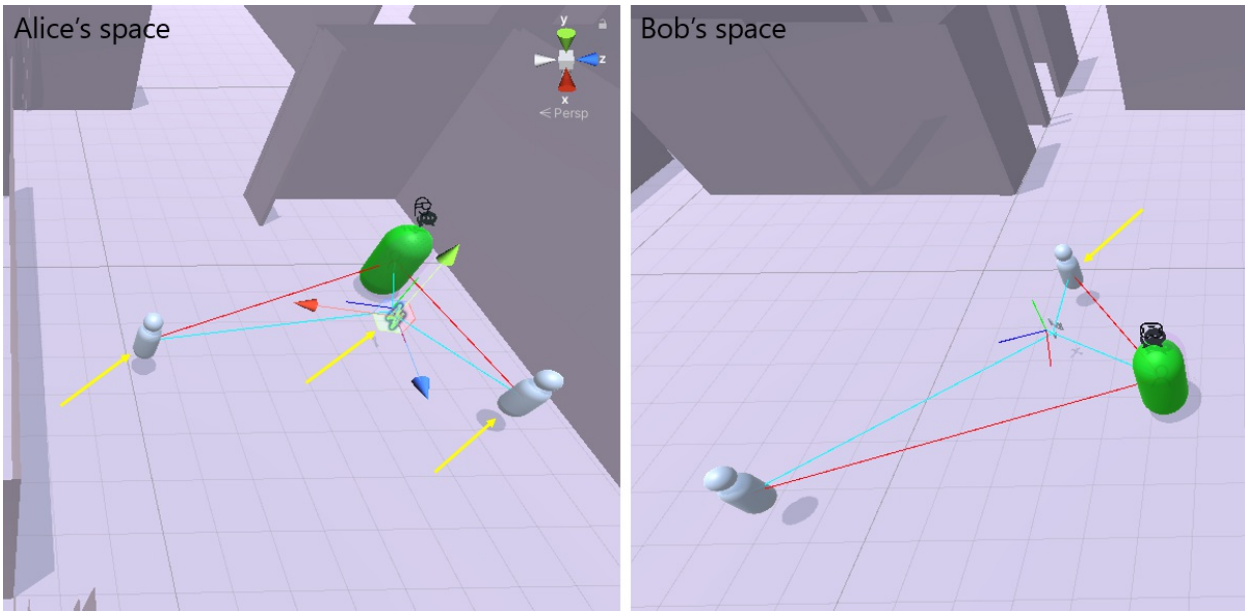


The position of each participant in respect to the session origin is unique and shared in real time with all participants: when a participant moves around, the movement is shared with the other participants.



At the same time, the placement of the session origin in a participant's local system determines where the avatars of the other participants appear in the local space. Moving the session origin in your space will move all the session content accordingly. Note also that when you move the position of the session origin relative to yourself, you are moving your position relative to the session origin, so from the other participants' point of view your avatar will move in the opposite direction.





Placing the session origin at an anchor

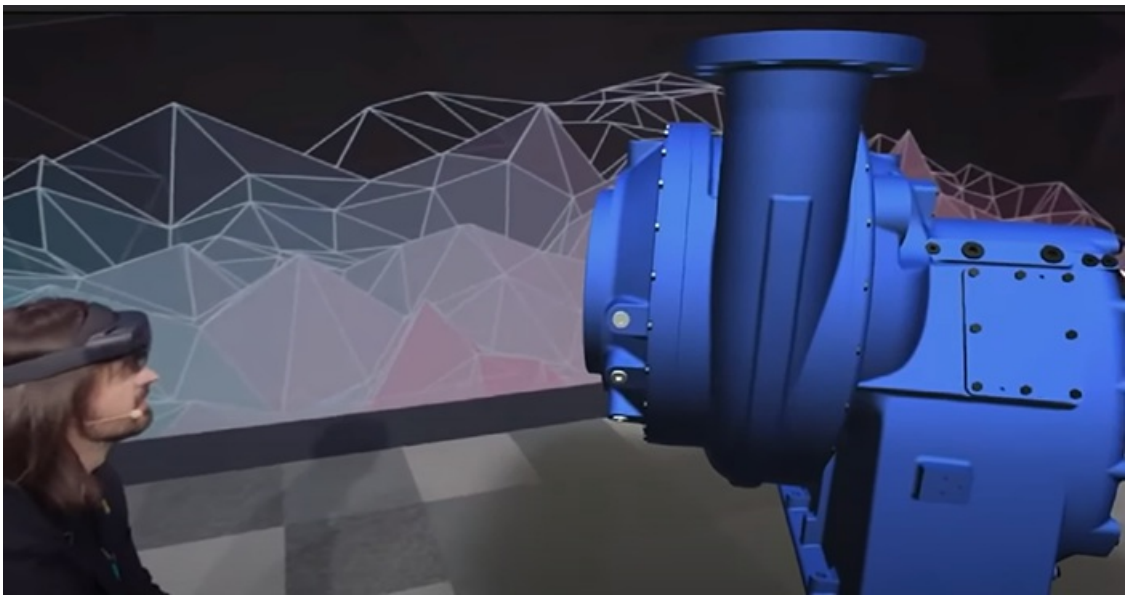
In some cases, it might be desirable to center the session coordinate system in correspondence of a tracking anchor, rather than at a fixed 3D position/orientation in the local coordinate system. For example, it can be done to make the session origin appear at a precise point in the physical space, and ensure that it appears stably anchored to that point as the device moves. Also, while a transform in the local coordinate system is only meaningful for the local app instance and cannot be shared as it is, a tracking anchor corresponds to visual data that can be localized by other devices. For this reason, it can be used as a common reference to indicate where the session origin has been placed in the real world, and to share this information with other participants (for example through Azure Spatial Anchors).

When the session origin is placed at an anchor, the application still needs to get a transform to map the local coordinate system to the session one, and to transform the spatial data back and forth as described above. On most MR platforms, this transform can be easily retrieved from the anchor itself. Note that the value of the transform will typically change as the platform tracking gathers data, so an app should query the new value on every frame in order to keep the session origin and content stably aligned to the physical space.

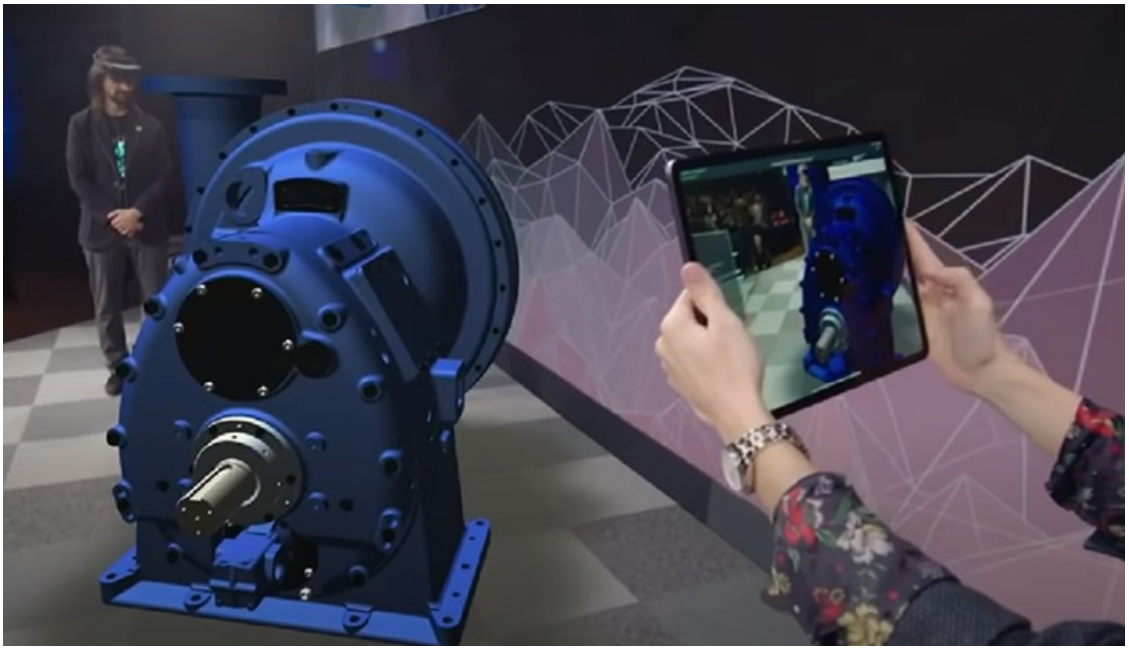
Example: two participants in the same location

In this example, two participants are in the same physical location. The first participant places a tracking anchor at a point in their coordinate frame, establishing that point as the session origin. He then creates an Azure Spatial Anchor at that location, and shares it with the rest of the session as a session address. The second participant queries the session addresses, finds the shared Azure Spatial Anchor, and tries to locate it in her local space. Since she is close, she manages to locate it successfully, and places the session origin on it. The participants are now operating in the same coordinate frame; since they have both placed the session origin on the same Azure Spatial Anchor, the position and orientation of the session coordinate frame *in the physical space* is the same for both. All the holograms that they share with each other will then appear to have the same position/orientation in physical space for both. When participants share the same physical location/orientation for the session like this, they are said to be *co-located*.

In the session below, the participants are co-located and the devices they are using determine whether they view avatars and objects in 2D or 3D. The session creator has placed the session origin in the middle of the room and called up a hologram there. He then views the hologram in 3D using his HoloLens 2.



The session creator shares his tracking anchor with a second participant, who can then view the hologram, in the same place in the real world, in 2D on her tablet.



Because the two participants are in different positions (relative to the session origin, and in the physical space), they see the hologram from different angles.

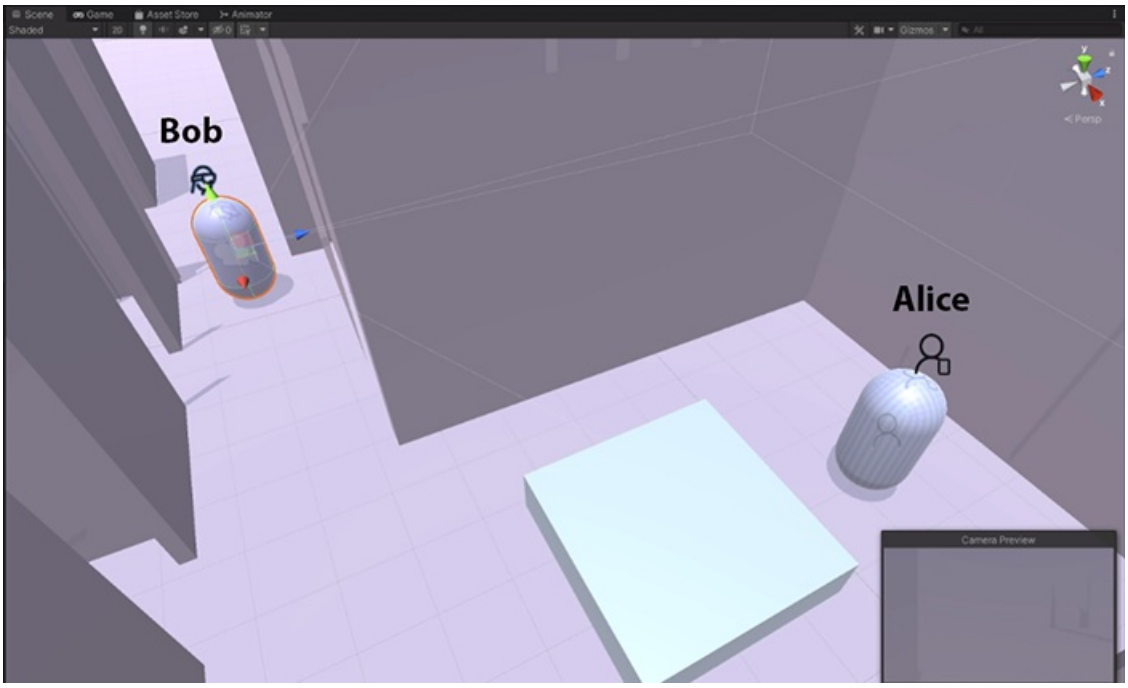
An application can exploit the fact that two participants are co-located to implement some optimizations. For example, it can avoid displaying each other's avatar on their devices, or skip voice transmission between them (since they see and hear each other in real life).

> **NOTE**
>
> Mesh doesn't *require* participants in the same physical area to place the session origin at the same location. You could have each participant place their session origin at a different point within the area. In such a scenario, they could still view a shared hologram, but it would be perceived as being in different places in the real world by each of the participants.

Example: local and remote participants

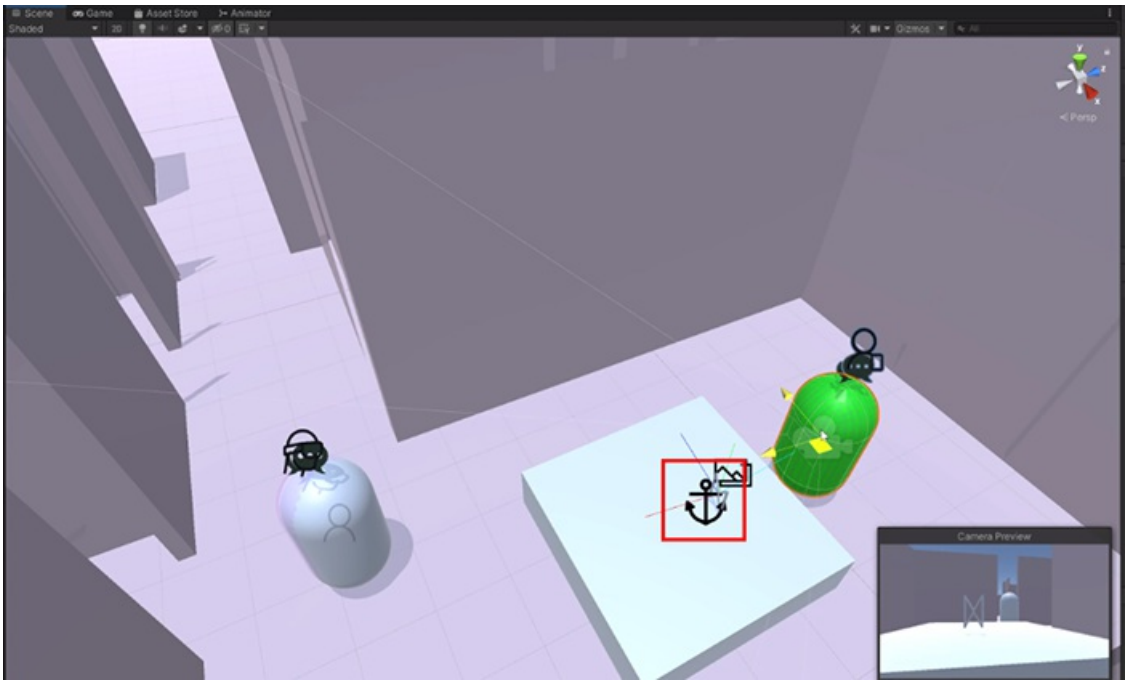
Mesh also allows creating sessions where participants grouped around multiple different locations in physical space share the same session content.

In this example, Alice and Bob work together at an architectural firm and want to view a hologram.

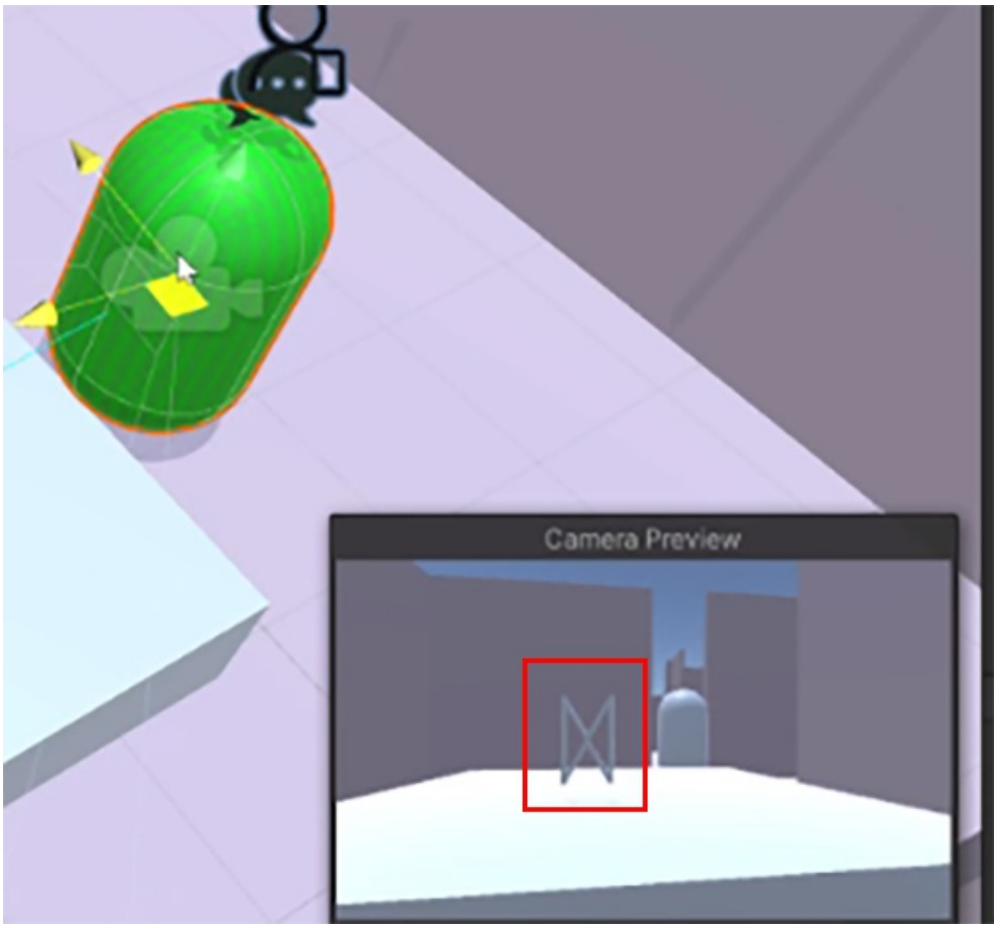


Bob starts a Mesh session. He is in the same physical location as Alice. Bob sends a session invite to Alice, and she accepts.

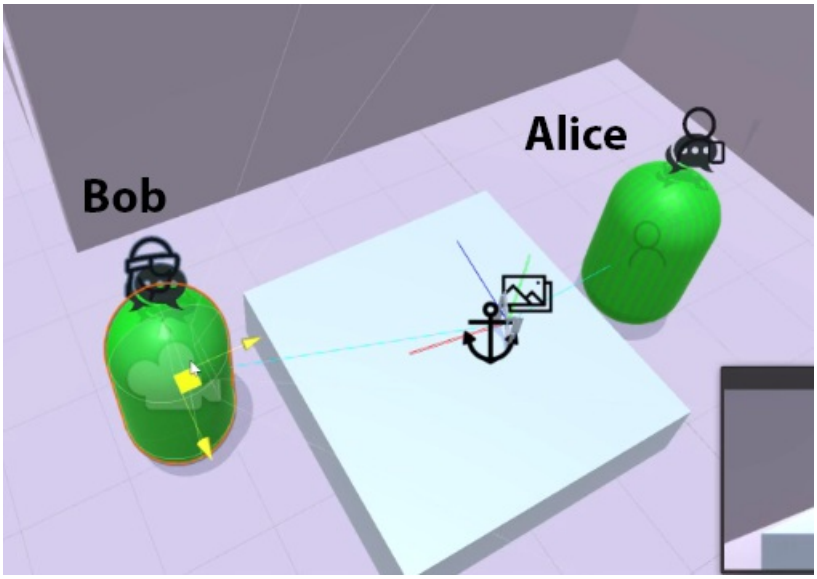
Alice decides that she wants the session origin to be located in the middle of her tabletop, so she places a local tracking anchor there, establishing that point as the session origin.



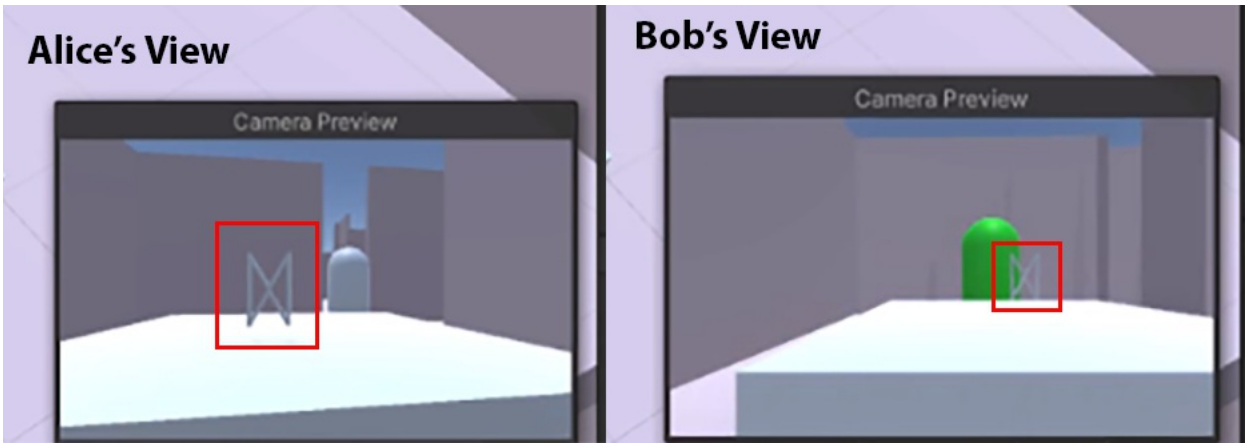
She calls up a hologram and places it at the session origin.



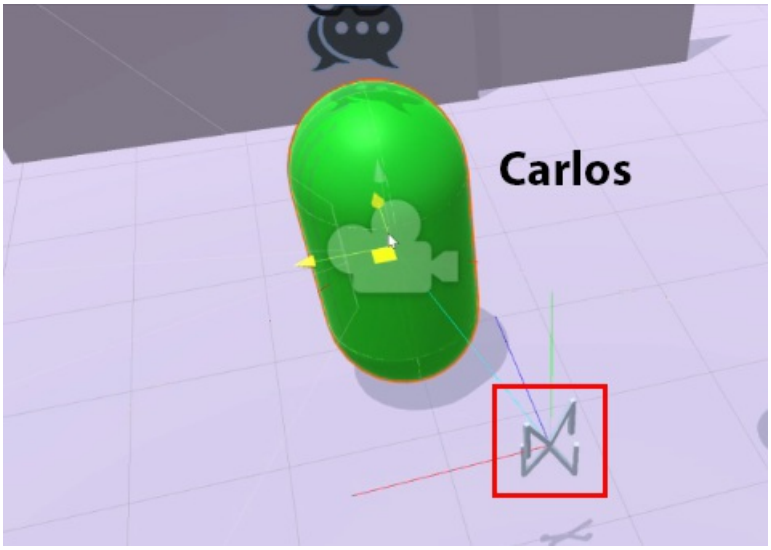
As in the previous example, Alice shares her placement of the session origin by creating an Azure Spatial Anchor and publishing it as a session address. Bob approaches Alice's table, and the Mesh client running on his device detects the anchor shared by Alice for him to place the session origin.



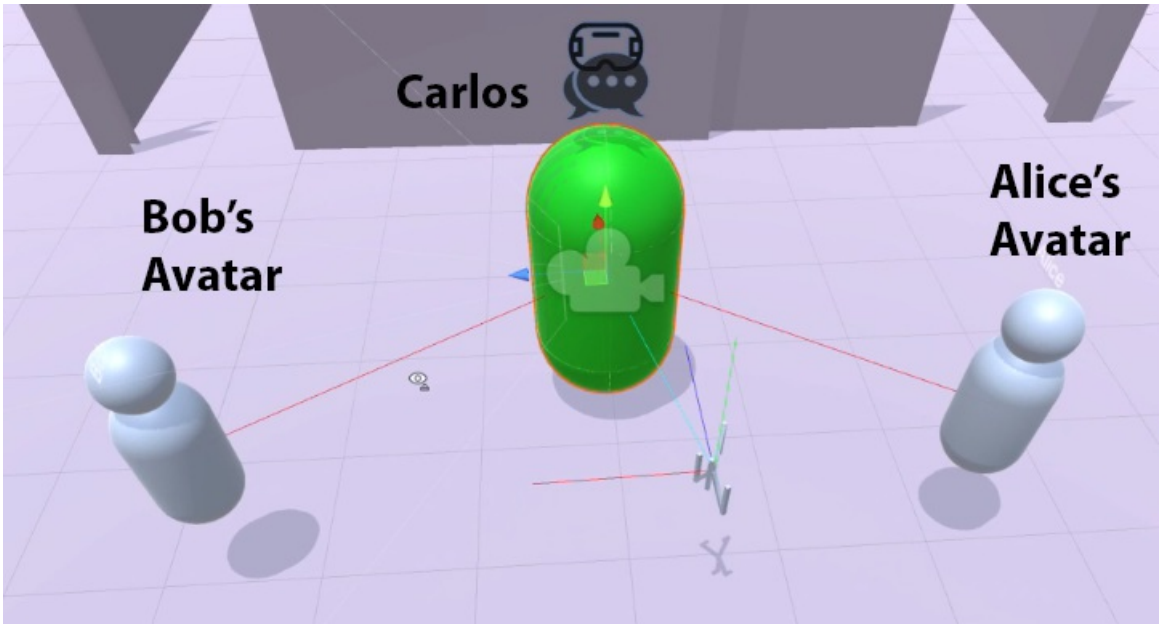
They are now sharing the same session coordinate system and view holograms in the same physical locations (each from his/her viewpoint).



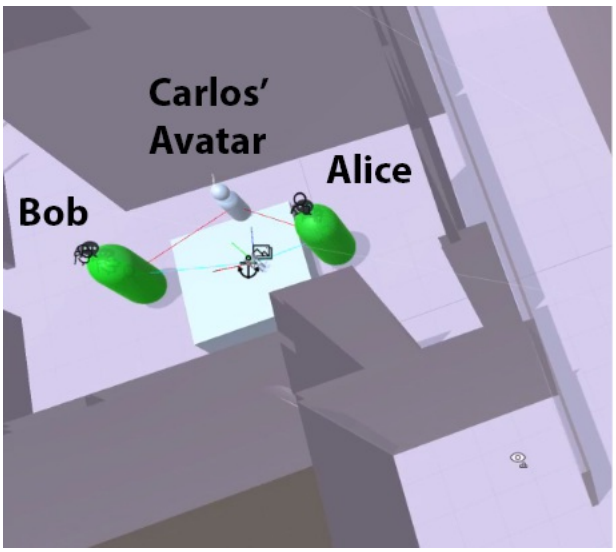
Bob decides to invite Carlos, a designer who is currently working at home in a different city, to the session. Carlos needs to determine a point in his local physical space that corresponds to the session origin. He adds a platform native anchor—just above the floor in his home office—and places the session origin there. The same hologram that Bob and Alice are viewing on Alice's tabletop at their location now appears in Carlos' office on his device.

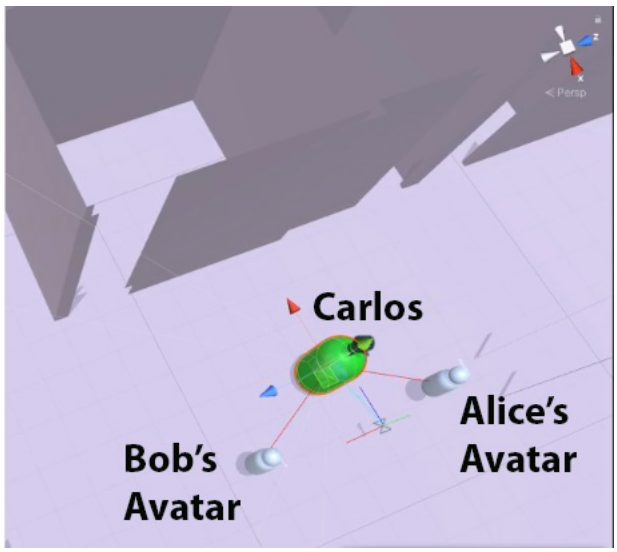


Bob and Alice's avatars are also now visible in Carlos' view. As Bob and Alice move around, the poses of their avatars are updated in Carlos' view. In addition, Carlos now has voice communication with Bob and Alice. Alice is standing off to the left of Carlos, so when she speaks, Carlos perceives the sound as coming from his left. In other words, the audio is spatialized.

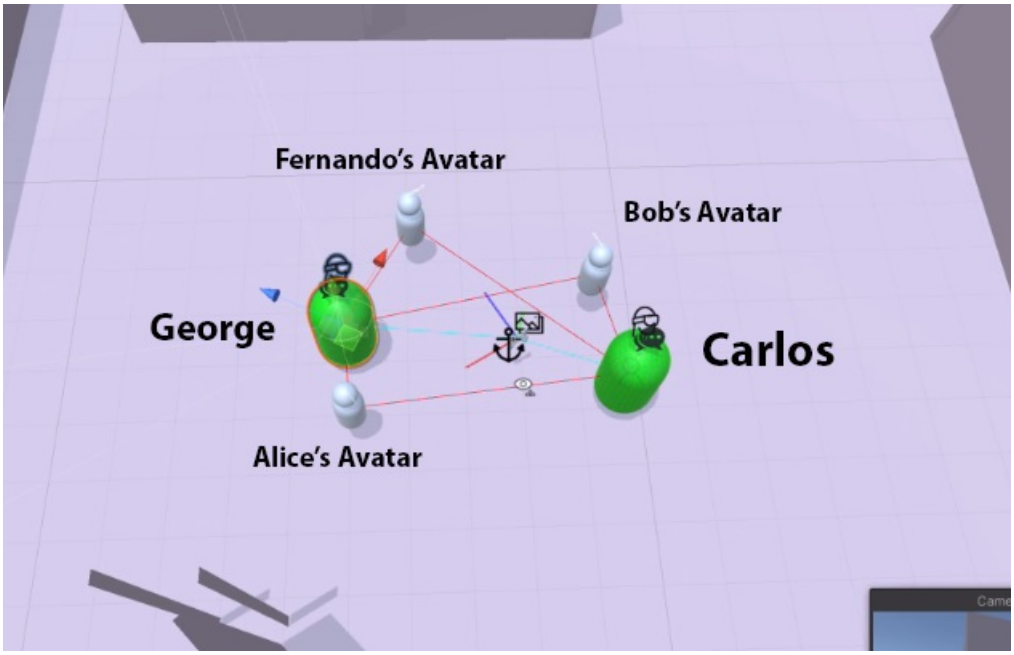


Carlos' avatar can now be seen by Bob and Alice on their devices. If Carlos walks in a circle around the hologram he sees in his view, Bob and Alice see Carlos' avatar walking around the hologram in their views.





Carlos decides to invite his partner George, who is in the next room, to the session. George accepts the invite, finds Carlos' anchor, and sees the hologram. George invites Fernando, another participant in a third physical location, to the session. So from Carlos' perspective, there are two co-located participants (he and George) and three remote participants (Bob, Alice and Fernando). Carlos can see the avatars of Bob, Alice and Fernando, and hear their voices through Mesh, and can also see and hear George physically near him. The app correctly recognizes that Carlos and George are co-located, so it skips avatar and audio communication between them.



Session addresses

Mesh allows applications to annotate a session origin with *addresses*. Addresses are identifiers which can be used to describe a location and orientation in real or virtual space. An address may be, for example, the ID of an Azure Spatial Anchor or a fiducial marker, as well as containing a mixture of components like gps, WiFi fingerprints and others.

Participants can share addresses with the rest of the session members, retrieve addresses shared by others, and try to *locate* addresses in their local space. On success, the location process produces a tracking anchor or a transform (depending on the platform and on the address) which the participant can use to place the session origin. The participant may then notify others of the address which has been located, and that the session origin has been placed at that address. As seen in the examples above, this allows the application to reason about

which participants are co-located.

There can be a 1:N mapping from sessions to addresses. That is, a session origin can simultaneously be placed in several different locations in the real world by different users. Note however that each participant places or locates the origin in a single location.

Creating an address

Each address has a *type*, indicating what the address represents, and a *data payload*, that identifies the address within its type.

You can create addresses using methods on the `SessionAddress` class:

```
// Create an address corresponding to an Azure Spatial Anchor with a known GUID.
SessionAddress asaAddress = SessionAddress.CreateAsaAddress("1b047be1-3d84-406a-9253-7c563d86bf29");

// Create an address corresponding to the location of a QR marker with a specific data string.
SessionAddress qrAddress = SessionAddress.CreateQrAddress("https://www.microsoft.com");

// Create an address of a custom type.
SessionAddress customAddress = SessionAddress.CreateCustomAddress("MyType", "SomeData");
```

On Windows UWP platforms, you can also call `CollaborationClient.AddressFromAnchorAsync` to create an ASA address from a native tracking anchor directly. For example:

```
// Create a native tracking anchor; details depend on application.
Windows.Perception.Spatial.SpatialAnchor anchor = CreateAnchorAtCursor();

// Create an ASA corresponding to the native tracking anchor, then create an address from the ASA GUID.
SessionAddress asaAddress = await collaborationClient.AddressFromAnchorAsync(anchor);
```

> **NOTE**
> Support for creating ASA addresses within Mesh is currently unavailable in Unity apps.

Sharing addresses

The API entry point for sharing addresses is the `SessionOrigin` class.

A participant can annotate the session with an address by simply calling `ShareAddressAsync` on the current session's frame.

```
// Get the origin object for the current session.
SessionOrigin origin = collaborationSession.Origin;

// Share the address.
SessionAddress address = BuildAddress();
await origin.ShareAddressAsync(address);
```

Once the address is shared, all participants can retrieve it by calling `SessionOrigin.GetAllAddressesAsync` on the local `SessionOrigin` object. Participants can subscribe to new addresses being added through the `SessionOrigin.AddressChanged` event.

A participant can also share that they have placed the session origin at a specific address by calling `SessionOrigin.SharePlacementAddressAsync` with the address. After this method is called, the `SessionParticipant.SessionAddress` property for that specific participant will return the passed address for everyone in the session. The address will also be added to the shared list, like when calling `ShareAddressAsync`.

Apps can use `SessionOrigin.SharePlacementAddressAsync` and the `SessionParticipant.SessionAddress` property to understand which participants are co-located with the local one and with each other. In the general case where an address uniquely identifies a location in the real world (like with ASA addresses), two participants can be assumed to be co-located if their `SessionAddress` value is the same. Note that this might not be true if an address can correspond to multiple real locations (for example, if an address corresponds to a marker of which multiple copies exist); in those cases, the application will need to use additional data to disambiguate.

Locating an address

The `AddressSearch` API allows for simultaneously searching for the location of several addresses concurrently. Currently, it supports locating the following kinds of addresses on Windows UWP platforms:

- Azure Spatial Anchors
- QR codes

> **NOTE**
> Support for locating ASA addresses is currently unavailable in Unity apps.

Trying to locate an unsupported kind of address will produce no results. Apps that use custom mechanisms for co-location can create and share custom addresses through Mesh, but need to implement the location logic themselves.

Once an `AddressSearch` starts, it will report each address status change (located, rejected, etc.) until it is stopped, together with the located tracking anchor if available. See below for an example:

```
private AddressSearch addressSearch;

public void StartSearch()
{
    // Get the addresses to locate (for example, use SessionOrigin.GetAllAddressesAsync).
    SessionAddress[] addresses = GetAddressesToLocate();

    this.addressSearch = collaborationClient.CreateAddressSearch();
    this.addressSearch.Updated += SearchUpdated;
    this.addressSearch.Start(addresses);
}

private void SearchUpdated(object sender, AddressSearchUpdatedEventArgs args)
{
    foreach (AddressSearchResult res in this.addressSearch.Result)
    {
        if (res.Status == AddressSearchStatus.Located)
        {
            Log($"Address found: {res.Address.Type}:{Res.Address.Data}");
            ShowMarkerAtAnchor(res.Anchor);
        }
    }
}

public void StopSearch()
{
    this.addressSearch.Stop();
}
```

Out of scope

The Mesh SDK does not try to resolve races in address sharing, such as when two participants in the same space try to create an address for both at the same time (the address list might end up containing two different addresses close to each other). It does not choose where the session origin should be placed if several addresses

are located. It is assumed that the application will implement the appropriate logic for resolving such issues, depending on its specific scenarios.

Next steps

Spatial Audio

See Also

- Avatars
- Spatial audio

Spatial Audio

4/12/2021 • 4 minutes to read • Edit Online

When people speak in a 3D environment, the audio can be *spatialized*, which means that the audio from each sound source is processed to give the listeners the sense that it's coming from a specific point in space. In Unity, there are different audio spatializers the user can choose from.

The spatial audio system in the Mesh SDK can be easily integrated with audio spatializers. Mesh sends an audio stream from each participant in the session. These streams contain information on their origin (participant, 3D transform), which can be used as input to the spatializer.

Technical Details

Format

The audio data is sent and received as float PCM. Only one channel per stream is currently allowed. Sending and receiving sample rates can be set independently, choosing from one of the available ones (8, 12, 16, 24, or 48 kHz).

Compression

The audio data is compressed using the Opus codec. The bitrate can be customized, to favor audio quality over bandwidth.

Latency and Jitter Control

The audio stream compensates every jitter in the network by dynamically controlling the latency when it's received. Lost and out-of-order packets are replaced using forward error correction and packet loss concealment. These techniques are applied in a transparent way, to maximize the quality of the playback.

Noise Gate

The spatial audio system uses a noise gate to detect if the user is talking. When sending audio, an *audio frame* worth of a few milliseconds of audio data, is created and analyzed. Only if its "loudness" is above a certain level, the audio frame is encoded in a packet and sent over to the other participants. This way you can save bandwidth and ensure only relevant audio is transmitted. The parameters of the noise gate can be customized by the user.

The `SpatialAudioManager` object

To access spatial audio from a Mesh session, first you need to create and join a *spatial session*. You can find information on how to do it the avatars overview page.

The `SpatialAudioManager` object is exposed as a property of `SpatialSession`.

You can configure the `SpatialAudioManager` object to match the format used for capturing and playing audio in your application, as in the sample code below.

How to send audio

To send audio, you can call `SendAudio` from the SpatialAudioManager object. Sending can happen as soon as you capture PCM data from the microphone, either on the main update or from the audio thread.

```
private AudioClip _micClip = null;
private float[] _audioFrameData = null;
private int _sentMicrophonePosition = 0;

void OnConnected()
{
    _spatialAudio = spatialSession.SpatialAudioManager;
    _spatialAudio.CaptureSampleRate = SampleRate.Full; // 48kHz PCM for microphone

    // Microphone capture start: write to a 1 second long looping buffer
    // Note: the capturing frequency (48000 in this case) has to match _spatialAudio.CaptureSampleRate
    _micClip = Microphone.Start(deviceName: null, loop: true, lengthSec: 1, frequency: 48000);

    // PCM buffer for microphone data
    _audioFrameData = new float[_spatialAudio.CaptureFrameSamples];
}

void Update()
{
    spatialSession.AdvanceTimeTo(Time.time);

    if ( _micClip != null)
    {
        int micPos = Microphone.GetPosition(null);

        int toSend = ((micPos + _micClip.samples) - _sentMicrophonePosition) % _micClip.samples;
        while (toSend >= _audioFrameData.Length)
        {
            // Retrieve the PCM data for one audio frame, copying it from the microphone clip
            _micClip.GetData(_audioFrameData, _sentMicrophonePosition);
            _sentMicrophonePosition = (_sentMicrophonePosition + _audioFrameData.Length) % _micClip.samples;
            toSend -= _audioFrameData.Length;

            // Send the whole audio frame to spatial audio
            _spatialAudio.SendAudio(_audioFrameData, 0, -1);
        }
    }
}
```

How to receive audio

To receive audio, the application has to subscribe to the event `AudioStateChanged` in the spatial audio manager. The event is called whenever a new audio source is added or removed into the spatial session. In the callback you can access any new *IncomingSource* object. The app can hold onto this object to get information on that specific audio source, like the source participant or the location of the audio source, and to consume the incoming audio stream.

```
private AudioClip _micClip = null;
private float[] _audioFrameData = null;
private int _sentMicrophonePosition = 0;

void OnConnected()
{
    _spatialAudio = spatialSession.SpatialAudioManager;

    // Note: set the playback sample rate to match the sample rate of the application
    _spatialAudio.PlaybackSampleRate = SampleRate.Full;

    _spatialAudio.AudioStateChanged += AudioStateChanged;
}

void AudioStateChanged(object sender, SourceStateChangeArgs args)
{
    var source = args.AudioSource;

    switch( args.State )
    {
        case SourceState.Added:
        {
            if (source.Participant != null)
            {
                TryAttachSoundSourceToAvatar(source.Participant, source);
            }
            else
            {
                CreateNewObjectForUnboundSoundSource(source);
            }
            break;
        }
        case SourceState.AboutToBeRemoved:
        {
            if (source.Participant != null)
            {
                TryDetachSoundSourceFromAvatar(source.Participant, source);
            }
            else
            {
                RemoveUnboundSoundSource(source);
            }
            break;
        }
    }
}
```

The audio stream is usually consumed in an audio callback for the object the incoming source is attached to. For example, in Unity we can override MonoBehaviour.OnAudioFilterRead.

```
void OnAudioFilterRead(float[] dspOutBuffer, int channels)
{
    var audioSource = _incomingAudioSource;
    if (audioSource == null)
    {
        return;
    }

    Array.Resize(ref _incomingBuffer, dspOutBuffer.Length / channels);

    // Pull the audio data from the incoming source, to fill up the DSP audio buffer
    audioSource.ReadPcm(_incomingBuffer);

    // Note: the PCM format returned by ReadPCM is determined by SpatialAudioManager.PlaybackSampleRate
    int outIdx = 0;
    for (int i = 0; i < _incomingBuffer.Length; ++i)
    {
        for (int c = 0; c < channels; ++c)
        {
            dspOutBuffer[outIdx++] = _incomingBuffer[i];
        }
    }
}
```

For a more detailed Unity example, and to know how to associate audio to avatars see the sample scene in Avatars toolkit.

Next steps

Transport

See Also

- Avatars
- Session origin

Transport

4/23/2021 • 2 minutes to read • Edit Online

Overview

The Transport class is a connecting layer between users in the same session, transferring audio and video data to all parties. This includes audio and streaming video, as well as data messages between users.

Creating a transport object

```
// 1. Create a network transport object.
// - Enables defaults for incoming audio to default speakers.
Transport transport = await CurrentSession.CreateTransportAsync();

// 2. Keep the transport object around for access outside of current scope.
CurrentTransport = transport;
```

NOTE

`CreateTransportAsync` will throw an exception if a timeout occurs.

Sharing audio

```
// Start sharing audio with everyone in the session using the default microphone.
transport.SessionMedia.IsMicrophoneMuted = false;
```

Streaming video

```
// Start sharing video with everyone in the session using the default camera.
CameraShare share = await transport.SessionMedia.CreateCameraShareAsync(null);
await share.StartAsync();
```

Rendering incoming video

```
VideoStream video = default;

// 1. Set the video device.
MeshServiceManager.Instance.SetVideoDevice(MeshVideoPlugin.GetMediaDevicePointer());
// IMPORTANT NOTE: To receive video, this call is required.
// It must occur before you start receiving video,
// and preferably before you've created a session.

// 2. Setup frame ready event handler.
var onFrameReady = new FrameReadyDelegate((sender, args) =>
{
    // Acquire VideoFrame object.
    // IMPORTANT NOTES:
    // To prevent memory leaks in your application, acquired VideoFrame objects should be explicitly
disposed of.
    // If AcquireVideoFrame() isn't called at least once before your event handler(s) return, the frame data
is automatically destroyed.
    using (var videoFrame = args.AcquireVideoFrame())
    {
        // Read and render frame data
    }
});

// 3. Subscribe to media changed event handler.
CurrentTransport.SessionMedia.MediaStateChanged += new MediaChangedDelegate((sender, args) => {
    if (args.VideoStream != null) {
        if (args.MediaState == MediaState.Started) {
            // Subscribe to frame ready event
            video = args.VideoStream;
            video.FrameReady += onFrameReady;

            // You can also set the preferred (but not guaranteed)
            // resolution of the video. In this case, we use 720p
            video.SetVideoPreference(1280, 720);

            // Start receiving video
            await video.StartAsync();
        } else if (args.MediaState == MediaState.Removed) {
            video.FrameReady -= onFrameReady;
            video = null;
        }
    }
});
```

Sending data messages

```
// 1. Subscribe and handle message data when it's received.
CurrentTransport.MessageReceived += new MessageReceivedDelegate((sender, args) => {

    // 2. Read all data in one go.
    // - Best practice is to recycle all of these buffers.
    byte[] bytes = new byte[args.DataLength];
    // This next line is pseudocode:
    ui.ShowBytesFrom(args.RemoteParticipant, args.ReadData(0, bytes, 0, bytes.Length));
});

// 3. Send or broadcast media.
CurrentTransport.Broadcast(TransportMessageOptions.None, new byte[] { 0, 1, 2 });
CurrentTransport.Send(mySelectedParticipant, TransportMessageOptions.Reliable, new byte[] { 0, 1, 2 });
```

Next steps

Unity quickstart

C++ quickstart

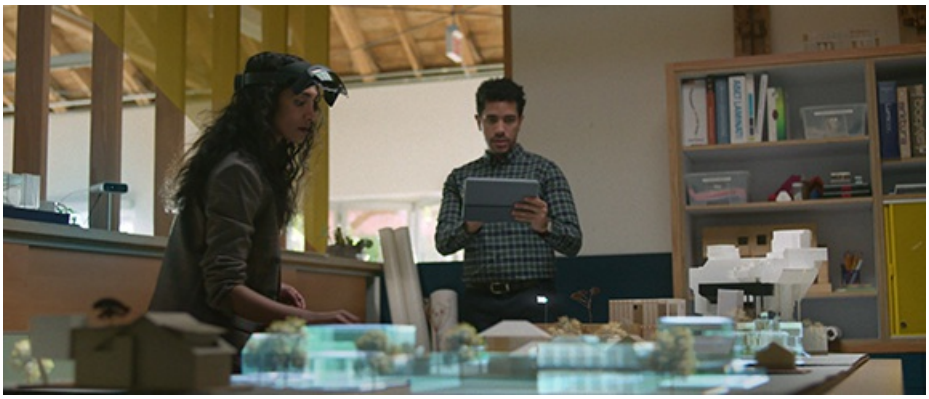
See also

- Client

ToolkitDemo for Unity

4/23/2021 • 3 minutes to read • Edit Online

If you start a project with the default scene and want to take advantage of the Mesh tools for Unity, you must add prefabs to your scene and adjust settings. As an alternative, you can open the `ToolkitDemo` scene which gives you a complete user experience (UX) that addresses common needs for collaboration. A number of tools are already pre-configured, so you can press Play and quickly get into a collaborative space.



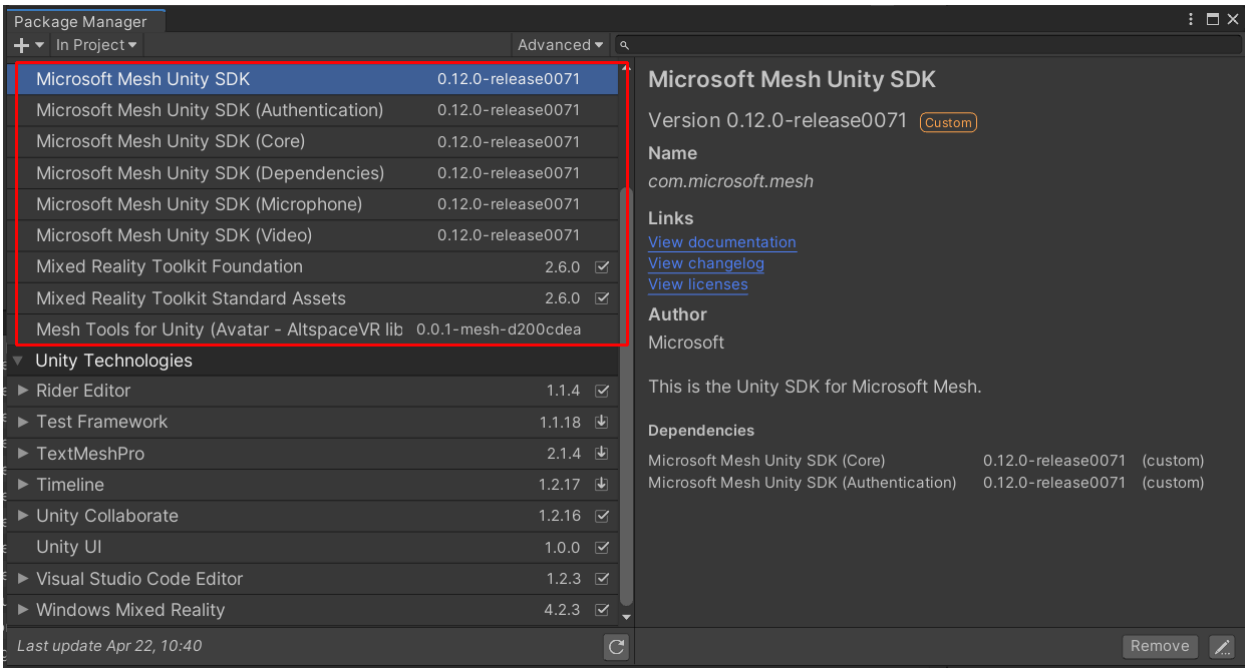
The `ToolkitDemo` scene goes beyond the SDK surface to provide examples for working with a variety of services. You can use the available components and prefabs "as-is", or use them as a starting point for your own ideas.

User features

- Invite people to join you in a *Session* from a variety of devices
- See others virtually as 3D avatars
- Talk with other participants
- Stream your camera or desktop to the session
- Share 3D models from OneDrive
- Call out a point in space with markers
- Draw something that everyone can see

Basic usage

1. Make sure you have the packages below installed.



1. Copy the `ToolkitDemo.unity` scene to your project's `Assets` folder.
2. On the menu bar, select **Microsoft Mesh** > **Settings**, and then in the **Inspector**, add your Client and Tenant ID.
3. Press the Play button to connect.
4. Look for authentication prompts in your web browser.
5. Use the UI to create a session and invite people to join you.

Developer features

- A "Service Manager" that wraps the SDK in Unity functionality and addresses development pain points such as service life-cycle and threading.
- UX that exercises the SDK features for sessions and its participants.
- A network programming model to support rapid prototyping.
- Builds on MRTK.
- Contains examples for loading content from OneDrive.
- Utilizes Graph for identity, search and storage of personal preferences.
- Includes helpers to get you authenticated with Azure Active Directory.
- Input routing and joint mapping for animating avatars.

Sample UX

The `ToolkitDemo` scene is composed of prefabs that provides a variety of tools and features to participants in a session. All of the prefabs below can be found in the `toolkit.ux` package.

- MRTK UI service, session, participant, and invitation management.
- Centerpiece controls scene orientation.
- Display Case shares 3D models from OneDrive.
- Markers let you call out a place in space.
- Ink gives you expressive capabilities for when words aren't enough.

Developer components

You can use the tools in the `ToolkitDemo` scene "as is" or view them as a starting point for your own ideas.

Packaging philosophy

The `ToolkitDemo` relies on several packages:

SDK/Helpers toolkit toolkit.avatar (Requires MRTK) toolkit.ux (Requires MRTK)

Each package depends on the one above it. This gives you the flexibility to choose only what you need for your purposes.

SDK/Helpers

COMPONENT	USE	DESCRIPTION
`MeshServiceManager.cs`	*Everywhere*	Singleton
`GraphUtilites.cs`	Display case	REST queries for Graph
`MSAL.cs`	ServiceManager.cs	Authentication
`WAM.cs`	ServiceManager.cs	Authentication on Hololens

Toolkit

MANAGERS	USE	DESCRIPTION
`VideoManager.cs`	MRTKUI	Coordinates transport for streaming video

NETWORKING	USE	DESCRIPTION
`ParticipantManager.cs`	Display Case, Markers, Ink	Associates participants with network data
`PersistenceManager.cs`	Display Case, Markers	Services for saving and loading state from OneDrive using REST
`NetObjectManager.cs`	Sample UX	
`NetLocalTransform.cs`	Display Case	
`NetMutex.cs`	Display Case	
`NetObject.cs`	Display Case	
`SharedSpaceRoot.cs`	ToolkitDemo.unity scene root	

Toolkit.Avatar

This package provides support to display the meshes of participants in a session. It includes a sample implementation that uses a Hololens 2 via MRTK to provide input to the service for pose prediction. It also comes with a rig animator that applies motion data from the service.

TOOLKIT.AVATAR	USE	DESCRIPTION
`MeshAvatarSpawner.cs`	Avatars prefab	Handles generating meshes for participants when they join
`MRTKToMeshInputRouter.cs`	Avatars prefab	Forwards head and hands on HoloLens to the service for pose prediction
`SpatialAudioSpawner.cs`	Avatars prefab	

Known issues

Composition

The `ToolkitDemo` is monolithic and work is needed to separate the concerns of the components, distilling them into something more friendly for reuse in other projects.

Redistributed DLLs

Many common development libraries aren't easily consumed by Unity; as a result, we are redistributing them to ease developer adoption. This presents a risk of conflicts so in the future, the libraries will be provided as a separate package.

Next steps

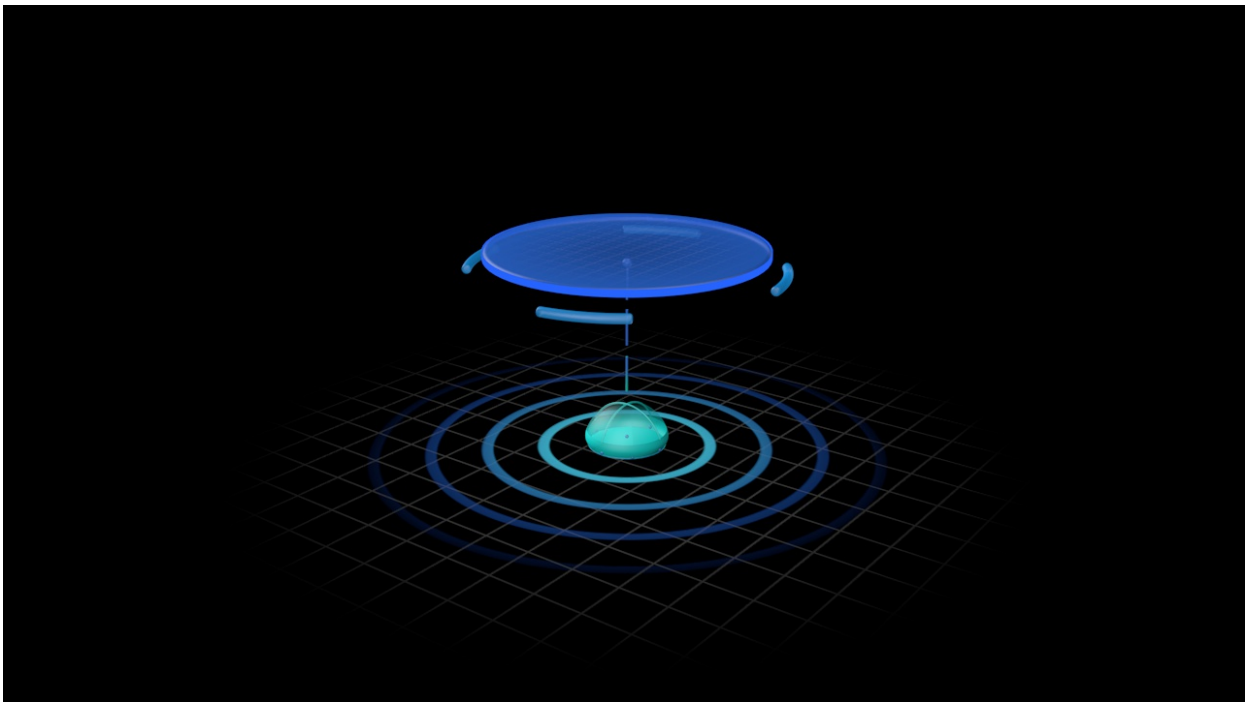
Centerpiece

The Centerpiece Mesh tool for Unity

3/29/2021 • 2 minutes to read • Edit Online

Overview

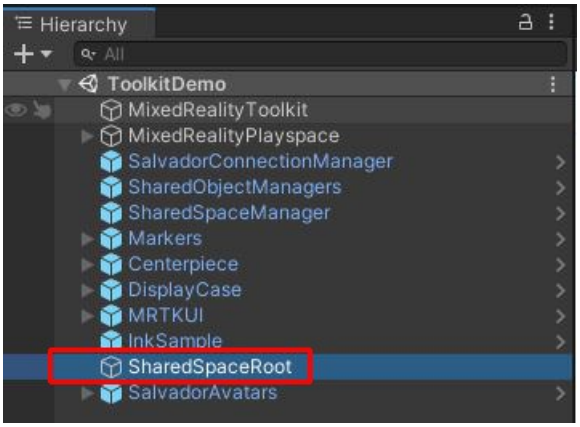
When participants are in a session, it's important for them to be able to share a common holographic reference no matter where they are in the world or which device they're using. The Centerpiece serves this purpose. Participants can use it to relocate and reorient the entire session contents.



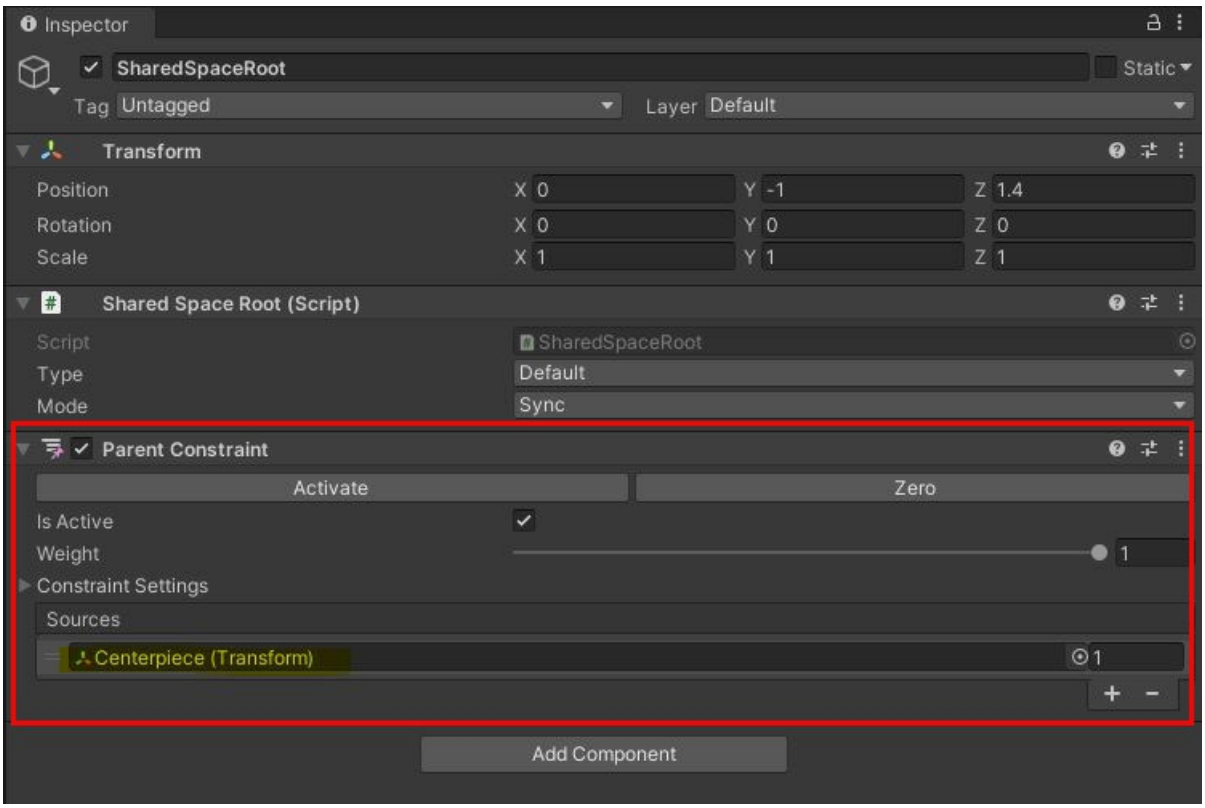
As a session participant, you may be in a physical room with a lot of furniture, making it difficult to walk around and view content from different perspectives. You can use the Centerpiece to rotate and reposition the contents of your session for a better view. This doesn't affect any other participant's view of the content or their own local Centerpiece; if you move or rotate your Centerpiece, other participants simply see your avatar change location. Another use of the Centerpiece is when you may want to change your position from standing to sitting at a desk. In this scenario, the ability to reposition your session contents with the Centerpiece is helpful.

Spatial location

For the Centerpiece to fulfill its role in a session, its spatial location (x,y,z) is passed to the `SharedSpaceRoot` game object.



This game object has a component named **Parent Constraint** that references the Centerpiece. This ensures that when the Centerpiece is moved during a session, the scene is moved along with it.



To learn more about spatial location, see Session Origin

Using the Centerpiece

To use the Centerpiece in your project:

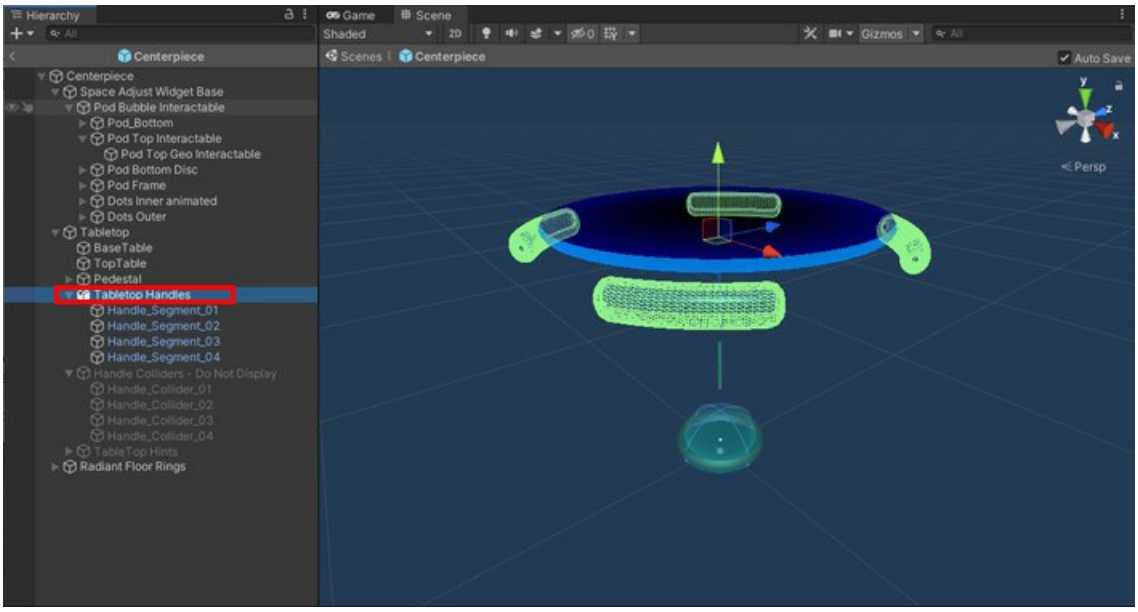
1. Make sure your scene is MRTK-enabled.
2. Drag the **Centerpiece** prefab into your scene. It's located in the **Azure Unity Toolkit UX folder**: **\Packages\com.microsoft.mesh.toolkit.ux\Centerpiece\Prefabs**

The Tabletop surface

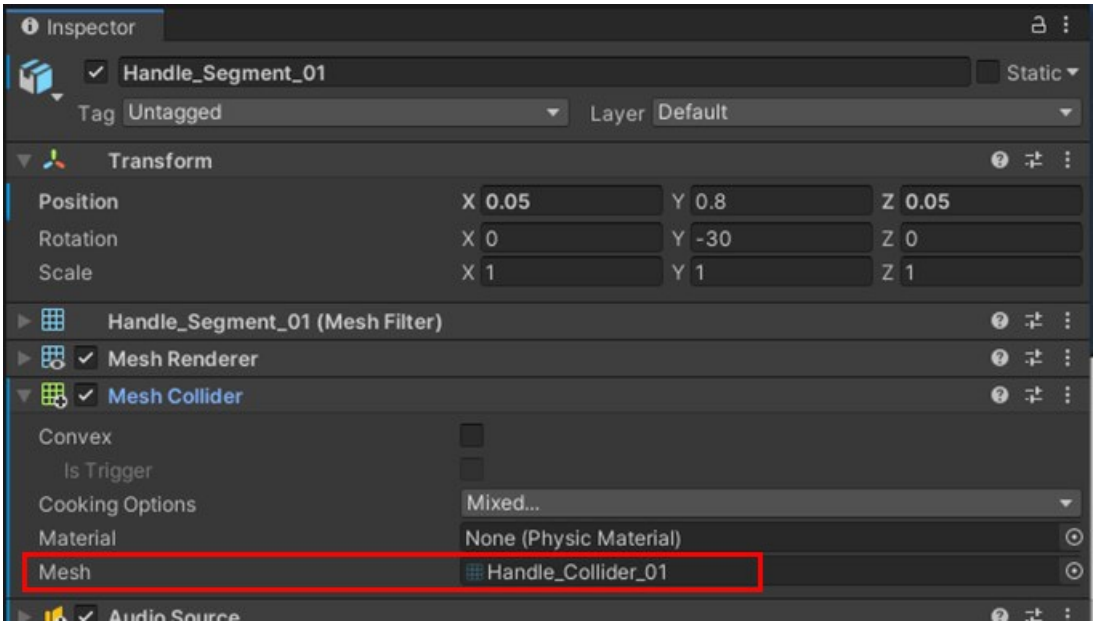
The Tabletop surface is where you share content and gather around with other participants for conversation. The default Tabletop is great for small groups and sharing table-sized content, and it can also be adapted for other purposes--for example, an informational kiosk that lists available sessions, conveys information about an event, or provides the background for a self-guided tour of a project. There are many possibilities.

Tabletop handles

The Tabletop game object contains a child object named **Tabletop Handles**. This optional set of handles lets people move and reorient the session. These handles are especially convenient when a person is close to the tabletop; they can simply reach out, grab a handle, and use it to move or rotate the Centerpiece.



Each visible handle references a **Handle_Collider_#**. This expands the targetable area of the handles and makes them easier to grab from a distance.



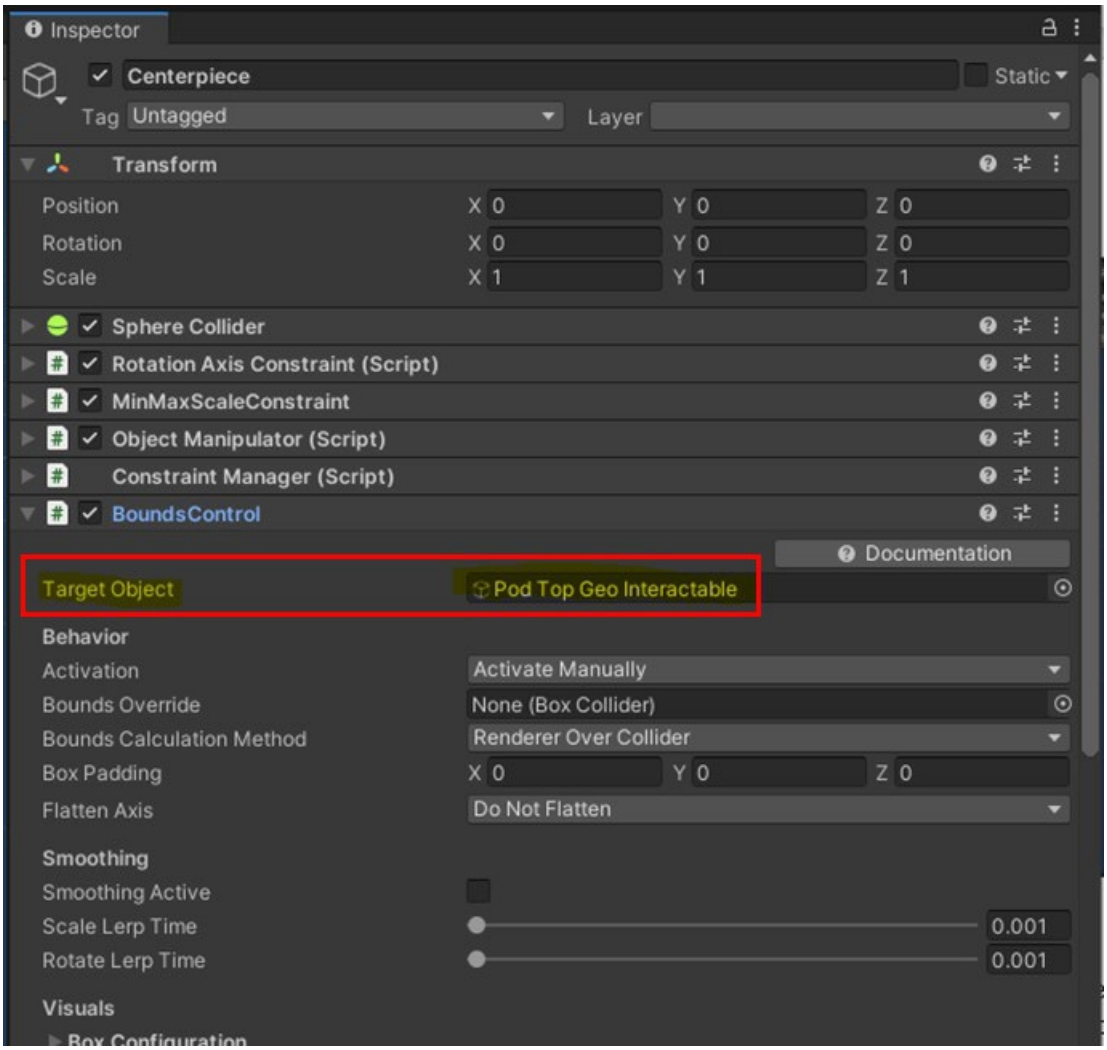
The Tabletop handles are optional. In the Hierarchy, the **Handle Collider** game objects are disabled because Unity only needs to reference their geometry. Disabling them doesn't affect the functionality of the Centerpiece; the base widget will still allow the scene to be relocated and rotated.

Centerpiece settings

Although Centerpiece works without the need to adjust any settings, it contains several scripts and child objects that are worth taking a look at, especially if you plan on doing customizations.

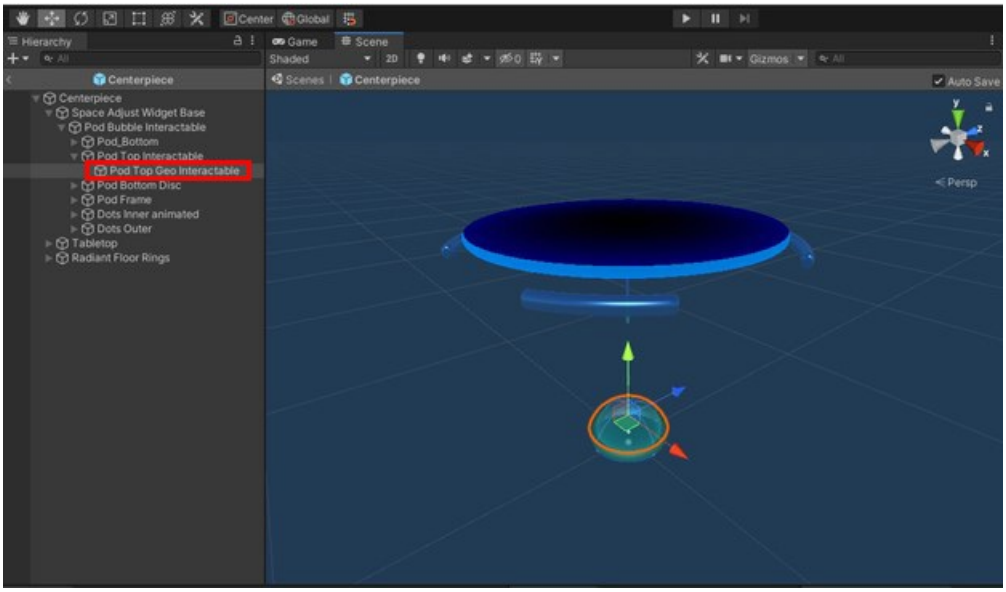
Rotation Axis Constraint (Script): Limits rotation to the Y-axis.

BoundsControl (Script): Note that its **Target Object** field points to **Pod Top Geo Interactable**. This prevents the Tabletop from being moved accidentally, which can happen when participants interact with content placed on top of it.



Pod Top Geo Interactable (child object): Use this to grab, move and reorient the session. This object makes up the dome of the base; we recommend that you align this with the floor plane.

[LINK to section/doc on snapping Centerpiece to floor plane coming soon]



Next steps

display-case

Mesh Tools for Unity UX Display Case

4/26/2021 • 2 minutes to read • Edit Online

> **WARNING**
> **This is an experimental, in-preview feature**. Some of the features of the Mesh Tools for Unity, while not yet fully fleshed out, have enough value at this stage that we can recommend that you take a look. We label them "experimental" to indicate that they're still evolving and subject to change over time.

Features

The Display Case allows anyone in a session to share content from OneDrive with others.

> **WARNING**
> ***Content shared in a session is available to everyone in your organization that has access to the link.*** To stop sharing the content, you must revoke permissions using OneDrive.

Prerequisites

- MRTK added to the scene
- MeshServiceManager present in the scene and configured.
- NetObjectManager present in the scene.
- Supported content uploaded to OneDrive.
- An active session.

> **TIP**
> You can use the *Mesh Service Manager Window* to create a session.

Setup

Add `Packages/com.microsoft.mesh.toolkit.ux/DisplayCase/Prefabs/DisplayCase.prefab` to your scene.

Declare Microsoft Graph scopes

To fetch the contents from OneDrive, the Display Case needs permissions. This can be set up ahead of time in the Azure portal or just-in-time using the *MeshServiceManager*.

To set up permissions in the Azure portal:

- Add the `Files.Read.All` to the *API Permission* of the registered app in AAD.

> **NOTE**
> Updated scopes in the portal may not propagate immediately due to the nature of cached tokens. Refreshing a token may be required.

To set up permissions in the Service Manager:

- Find the *MeshServiceManager* component in your scene and add the `Files.Read.All` permission to the *Microsoft Graph Scopes* if it's not already present.

Usage

Any participant in a session can use the *share button* to select a file.

After content is found, it's presented in a list.

To share an item:

- Choose the item to and then click the *share button*. The content is loaded and accessible to everyone in the session.

How it works

The Display Case uses OneDrive and Graph (via REST) to create a shared resource. Mesh is used to distribute the link to everyone in the session.

Supported content formats

- GLB
  - As implemented in MRTK
  - *GLTF isn't supported!*
- JPG
- PNG
- Unity Asset Bundles

GLB content

You can export GLB models from Paint 3D. In addition, the KhronosGroup provides examples.

Known Issues

GLB Models with complex hierarchies may not appear centered inside the display case.

Troubleshooting

If no item names appear in the selection menu, check for the following:

- Content must be stored on the OneDrive associated with the account used to log into the service.
- GLB content must have a lowercase `.glb` extension.

The Ink Mesh tool for Unity

3/29/2021 • 2 minutes to read • Edit Online

With the Ink Mesh tool for Unity, participants in a session can use a virtual pen to draw lines with "ink" in 3D space. Participants can see each others' ink but can only modify their own ink.

To use the Ink Mesh tool for Unity:
Drag and drop the Ink prefab from Ink\Prefabs directory into your Unity scene.

Once connected to a session, raise the right hand, and observe a button on the back of the right hand. The button is labeled "Toggle Ink." Use the left hand to press the button, to enable or disable the 'pen'.

When enabled, a cursor will appear between the right thumb and index finger. The color of the cursor represents the color of the ink to be drawn. When disabled, the cursor will not be visible.

Pinch and move the right hand to draw ink from the cursor location. Release the pinch to 'lift up' the pen, and end the stroke. Pinch again to draw another stroke, etc.

Known bugs or missing features:

- Offline drawing is not yet supported
- Currently only one color is supported
- Currently, there is no way to remove ink

Next steps

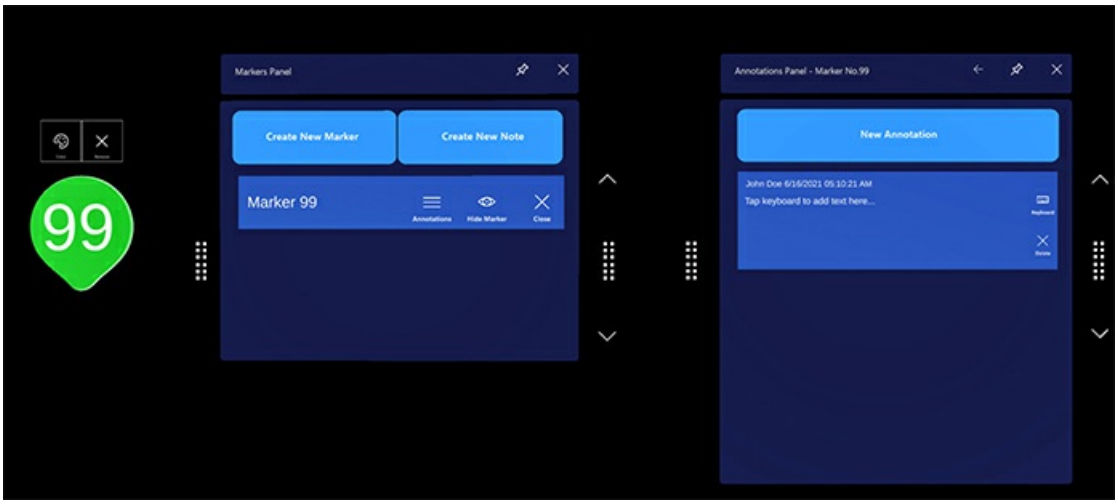
Markers

The Markers Mesh tool for Unity

4/23/2021 • 6 minutes to read • Edit Online

Overview

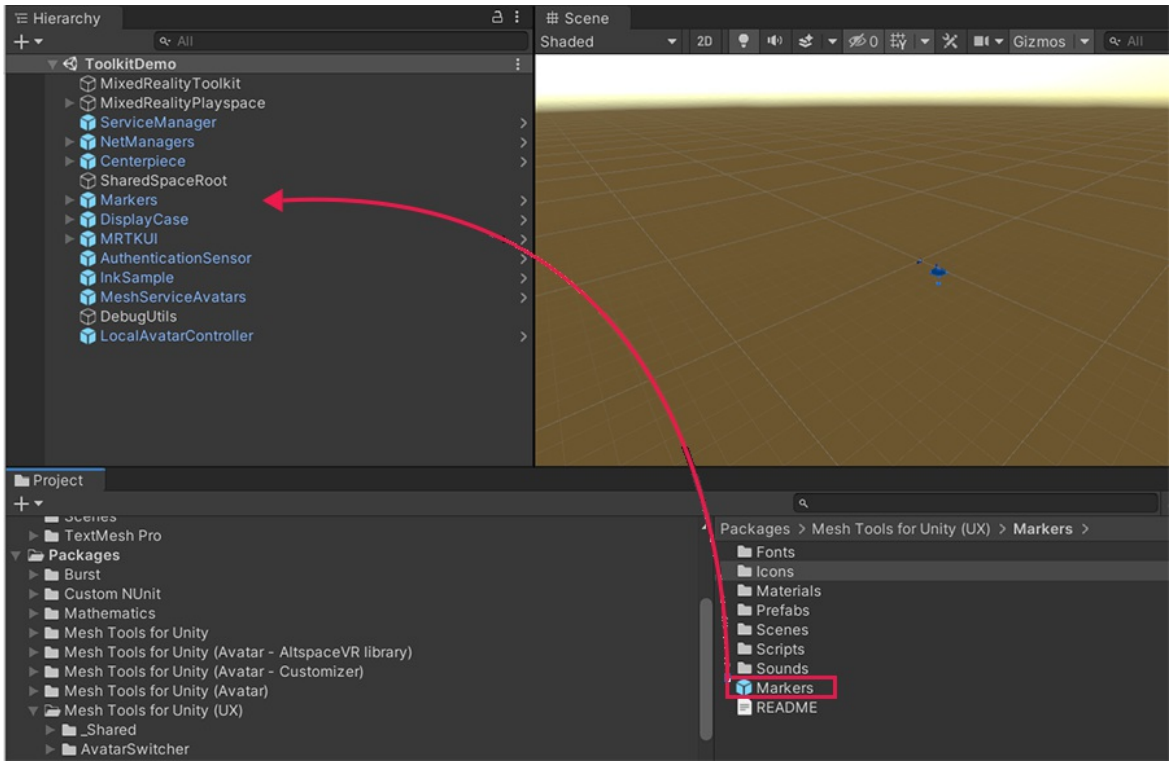
A Marker is a visual symbol that you can use to draw attention to a location, 3D model, or any other element in a scene. Control of Markers is not limited to the participant who created it; any participant in a session can move, rotate, hide or remove Markers.



Using Markers

To add a Marker to your scene:

1. Make sure your scene is MRTK-enabled.
2. Navigate to the **Packages** > **Mesh Tools for Unity (UX)** > **Markers** folder.
3. Drag the **Markers** prefab to the **Hierarchy**.



4. After the prefab has been added to the scene, the Hand menu becomes available. Use this to access the Markers panel and create a new Marker.

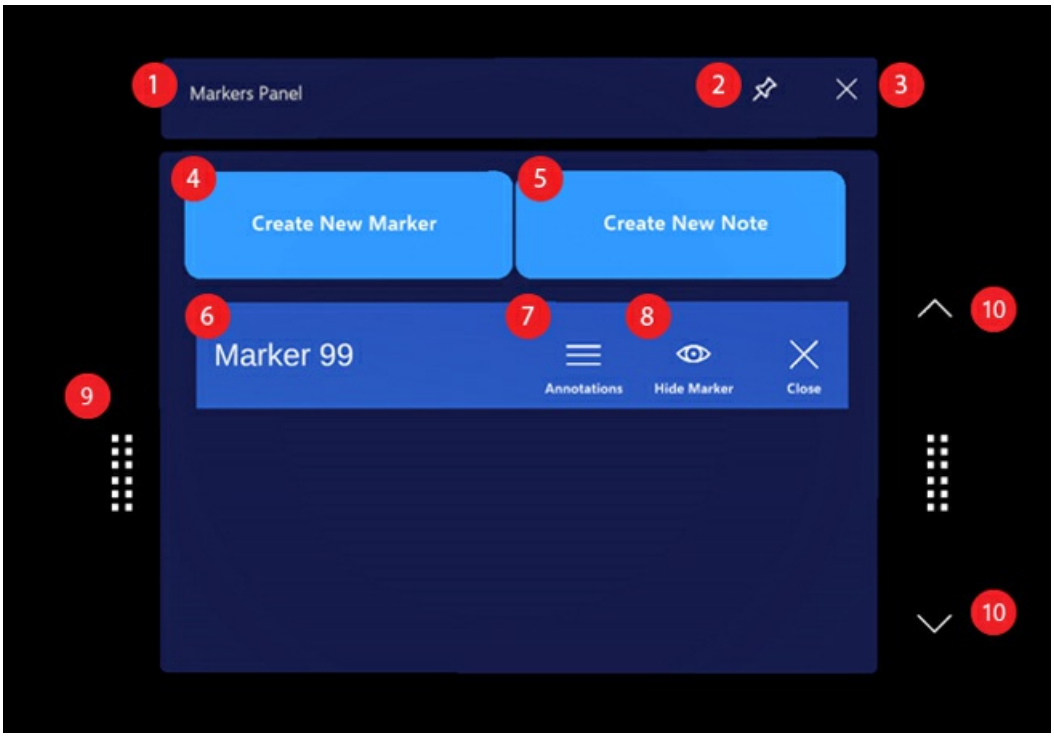
To move a Marker:

- Click and drag the Marker. You can change its position and rotation.

The Markers panel

To create and control Markers, use the Markers panel. As you move around, the Markers Panel stays close to you and is always within easy reach.

UI elements



1. **Title bar**: Displays the name of the panel. To move the panel, click the title bar and then drag.

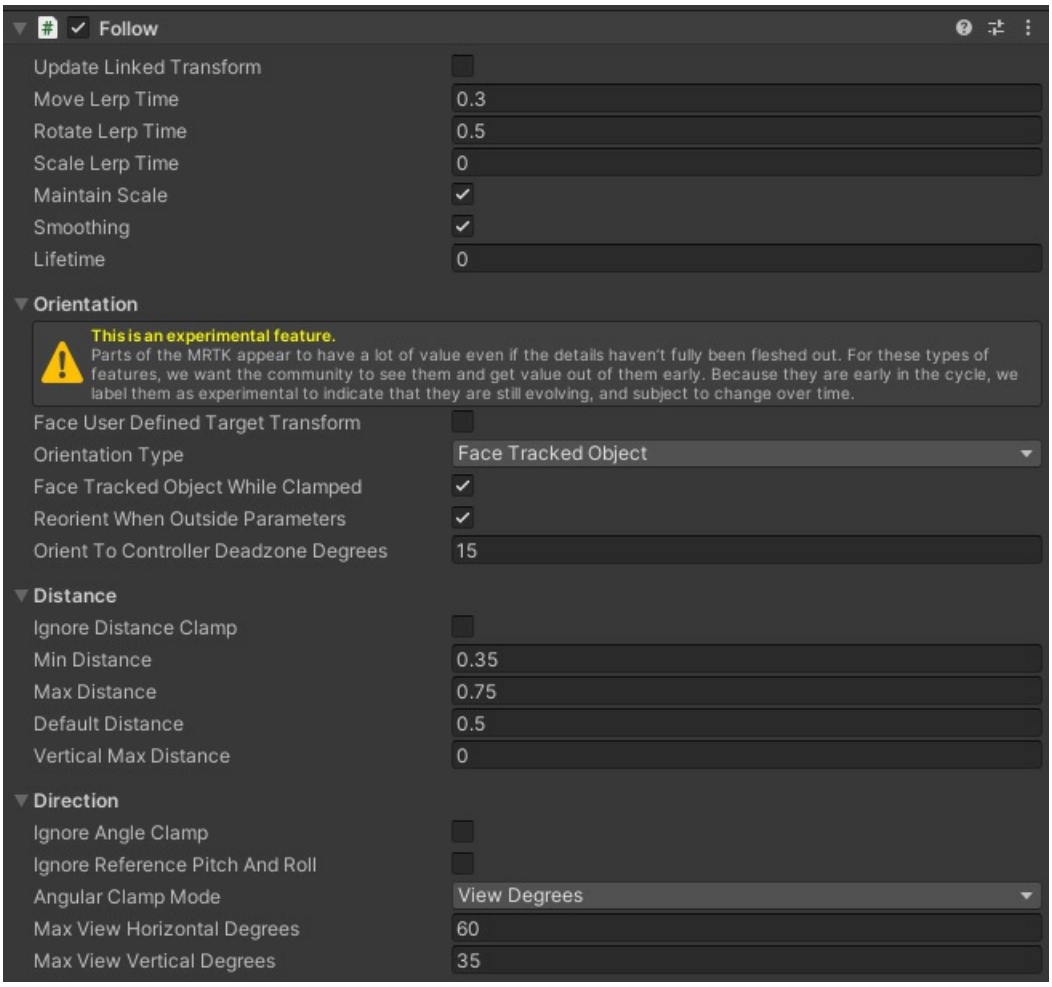
2. **Pin button**: Select this to stop the panel from following you as you move. The panel stays in the location it was in when you clicked.
3. **Close button**: If the panel is obstructing your view in the session, you can click this button to close it. Closing the panel does not remove any Markers from the scene.
4. **Create New Marker**: Creates a new marker in the scene. The new marker appears 0.5m in front of you, facing you, and in the middle of your field of view. You can also create a marker using the Hand menu, which is explained in the next section.
5. **Create New Note**: Creates a new note in the scene.
6. **Marker Row**: Each time you create a new Marker, the panel adds a row that contains settings and controls for that Marker. The first box in the row displays the Marker's number, which corresponds to its prefab icon.
7. **Annotations**: You can create, hide, or remove Annotations.
8. **Show/Hide**: Show or hide the current marker in the scene.
9. **Drag handles**: These appear when your hand gets close to the panel, or a hand ray intersects with it. Use the drag handles to change the position or rotation of the panel.
10. **Up and Down buttons**: If you have more than three Markers in your scene, you can't see them all at the same time in the panel. You can click the up and down buttons to scroll through all the Markers.

Changing a Marker's "Follow" Behavior

As mentioned earlier, the Marker panel will follow you if you move around in the session. The settings for this behavior are found in the **Follow** component. To view these settings:

1. In the **Hierarchy**, click the **Marker** prefab to view its child objects.
2. Click the **MarkerScrollParent** game object.

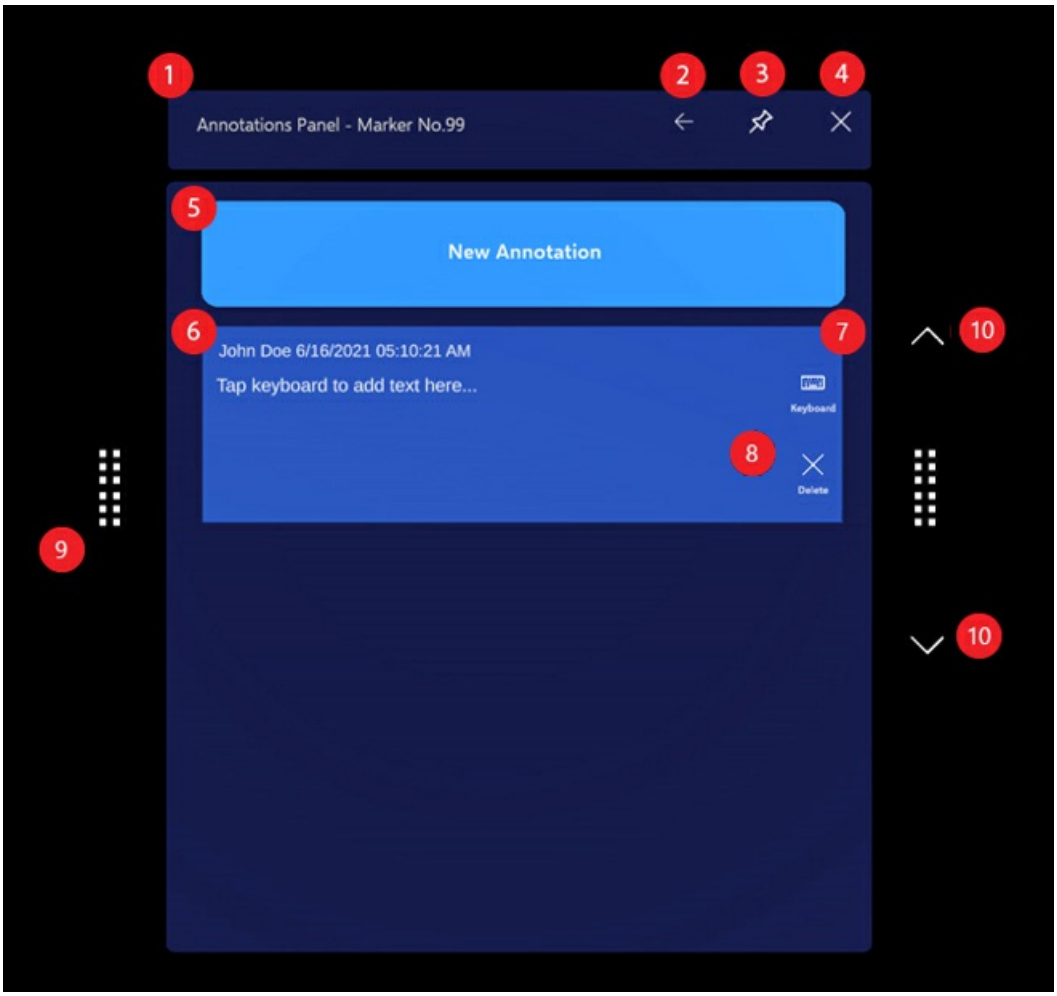
For comfortable follow behavior, we recommend the following settings:



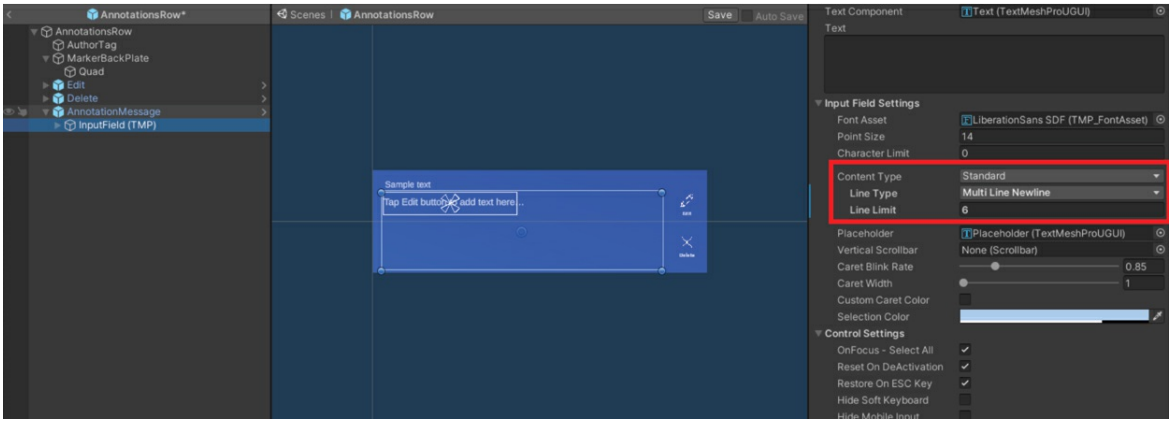
The Annotations Panel

With Annotations, users can share comments about objects synchronously or asynchronously. After a user adds an Annotation locally, it propagates over the network and can be seen by all participants. Like the Markers panel, the Annotations panel stays close to you as you move around and is always within easy reach.

UI Elements



1. **Title bar**: Displays the name of the panel and the related Marker number. To move the panel, click the title bar and then drag.
2. **Back button**: Returns you to the Markers panel.
3. **Pin button**: Select this to stop the panel from following you as you move. The panel stays in the location it was in when you clicked.
4. **Close button**: If the panel is obstructing your view in the session, you can click this button to close it. Closing the panel does not remove any Annotations from the scene.
5. **New Annotation**: Creates a new annotation row to add Annotations related to a Marker.
6. **Annotation content**: Every time a user creates a new Annotation, the system posts a user name and the date and time of the annotation. One item can handle up to six lines of text; if you need more lines, you can adjust the component.

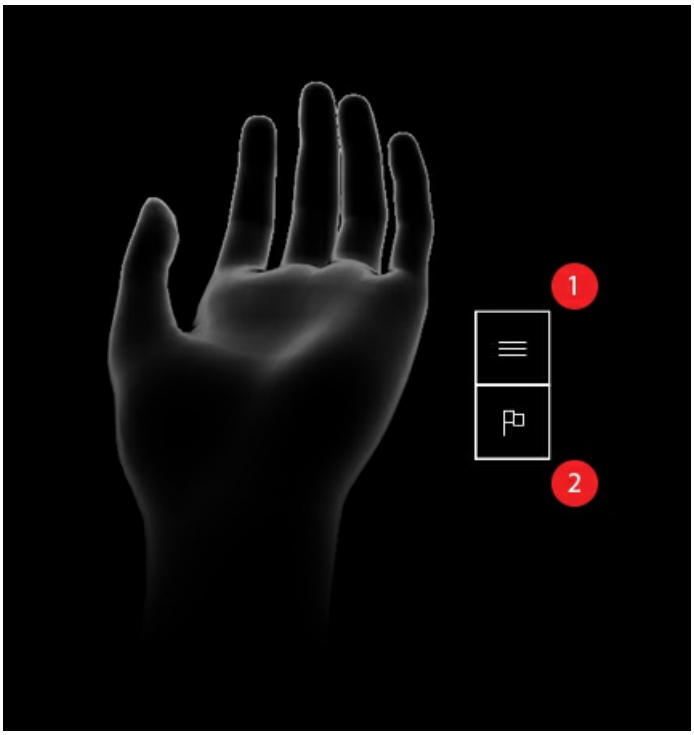


7. **Keyboard**: Opens a keyboard that you can use to add text to the annotation.

8. **Delete**: Deletes the current Annotation row. After a user deletes a row, the deletion propagates on the network and removes the same row on all other clients.
9. **Drag handles**: These appear when your hand gets close to the panel, or a hand ray intersects with it. Use the drag handles to change the position or rotation of the panel.
10. **Up and Down buttons**: Each time you create an Annotation, the panel creates a new row for it. If you have more than three Annotations in your scene, you can't see them all at the same time in the panel. You can click the up and down buttons to scroll through all the Annotations.

The Hand menu

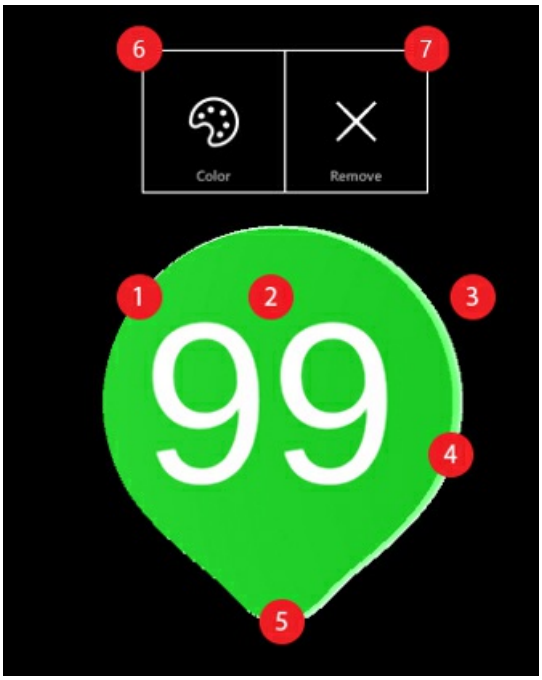
The Hand menu in the HoloLens is your main way to access the Markers panel. Learn more about the Hand menu.



1. Hide and show the Markers panel.
2. Shortcut to creating new Markers in a scene.

The Marker Icon

The Marker icon has a number of important design elements for you to keep in mind.

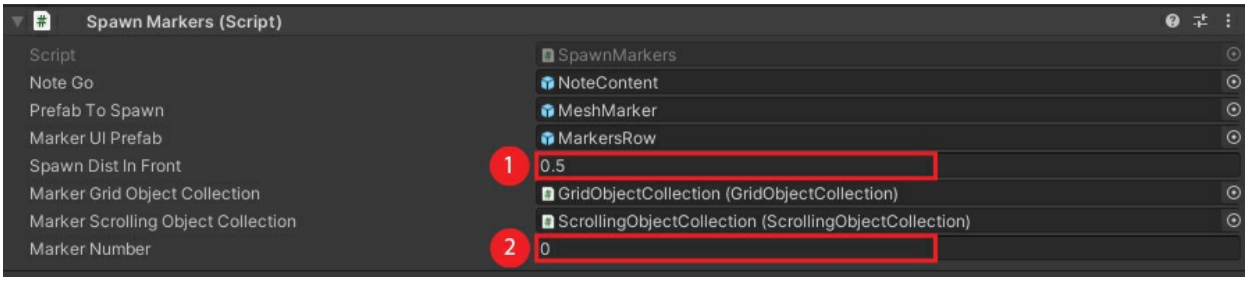


1. **Color**: Use a color that makes the Marker stand out in the scene.
2. **Reference number**: Each marker has a reference number. This number is displayed on both sides of the Marker so that all participants in the session can see it.
3. **Circular shape**: This shape makes it easy to grab the Marker and change its position or rotation.
4. **Volume**: A 3D object with volume has a strong and solid presence.
5. **Point**: The point on the Marker helps to make it clear which object or location you're calling attention to.
6. **Color Palette**: Use this to change the Marker's color. There are two default options.
7. **Remove**: Click this to remove the Marker.

Marker settings

To view the Marker settings:

1. In the **Hierarchy**, click the **Markers** prefab.
2. In the **Inspector**, view the **Spawn Markers (Script)** component.



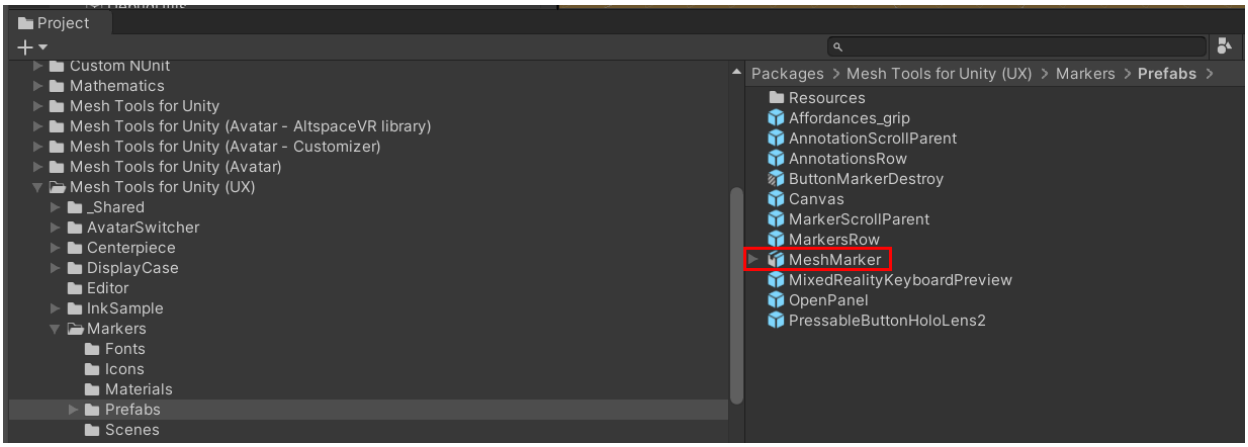
The two most important settings are:

1. **Spawn Dist In Front**: Determines how far in front of you the Marker will appear. In order to provide a comfortable position within easy reach of your hand, we recommend a setting of 0.5 from the user's head. Anything closer may cause discomfort.
2. **Marker Number**: This is the number for the first Marker you add to the scene. You can change it to anything you want.

Using a Custom Marker

The default 3D model that the Marker is based on is called **bevelPolygon1**. You can replace this with any 3D

model you want. The settings for this are found in the **MeshMarker** prefab.



If you decide to make this change, we recommend that you use a 3D model that you can point at the objects or locations you wish to draw attention to.

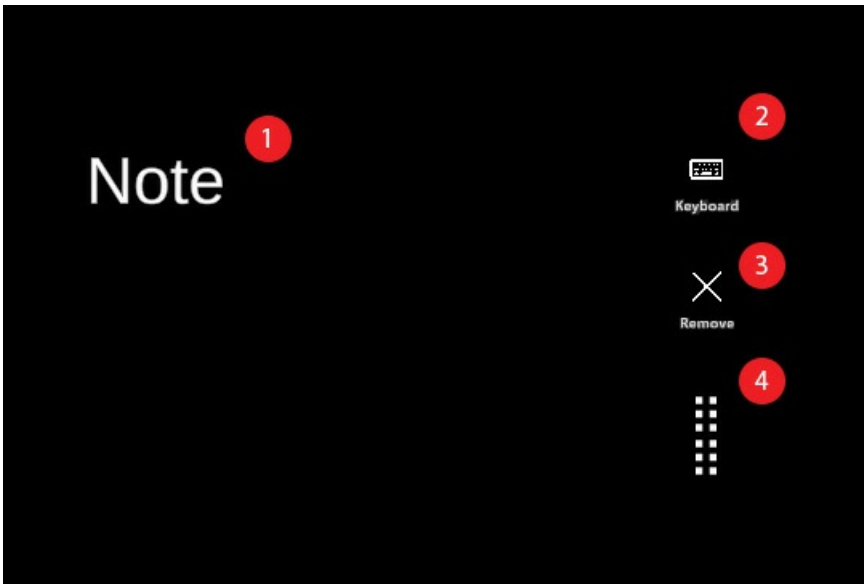
Mixed Reality Notes

You can create Notes which are visible to all participants in the session. This can used, for example, to provide comments about a hologram.

To create a Note:

- On the **Markers Panel**, click **Create New Note**.

You can find Notes on the Marker panel.



1. The area where users see the Notes.
2. The **Keyboard** button shows or hides the MRTK keyboard. Use this to type a Note.
3. To delete a Note, click **Remove**.
4. Click and drag here to change the Note's position or rotation.

Next steps

MRTK UI

The MRTK-UI Mesh tool for Unity

4/26/2021 • 2 minutes to read • Edit Online

Overview

MRTK-UI makes it easy to add collaboration features to your existing applications with 3D UI and UX that demonstrate the core platform features.

Features

- View the connection status to the service
- Start and name a new session
- Review pending invitations to sessions and respond to them
- Invite others to your session
- Search for users in your Azure Active Domain (AAD) to invite
- Start and stop streaming for your camera, desktop, microphone and speakers when in an active session
- View participants in a session and mute them
- Invite additional people to an active session
- Receive new invitations to join existing sessions
- Accessible by design

Directions

Add the `Packages/com.microsoft.mesh.toolkit.ux/MRTKUI/MRTKUI.prefab` to your scene.

This Mesh tool for Unity is optimized for Mixed Reality and HoloLens. It assumes that the MeshServiceManager is configured to connect to an Azure Active Directory.

For a similar tool built with Unity's 2D UI, see
`Packages/com.microsoft.mesh.toolkit.ux/UnityUI/Prefabs/UnityUIUserFlow.prefab`

Objectives

UX that gets a developer into a session with others, as quickly as possible. The user flow is card-based.

Home view

From the Home view you can do a few things depending on your connected state to the service. Once you are "connected and logged in" you have the option of starting a new session or checking your received invitations.

Create a new session

It's a simple as naming it - which is optional. People invited to your session will see this name. This assume their client is using this MRTK UI prefab.

Invite people

This optional step can be skipped. If you choose, you can enter some text into the input field and begin searching for participants. Select one or more people and click next to go to your active session.

The active session

Buttons are available to mute and unmute your microphone (which is on by default when you join a session). In addition you can share your desktop from PC or your camera. A toggle for turning down the session volume all the way is present.

From here you can view session participants as well as invite others. Of course, you may leave the active session at anytime.

A mic gain control is available to adjust your personal volume level.

Session participants

You can see a list of people in your session. Note it's possible for a person to join more than once from different clients. Select people and press the mute button to silence them.

Unresolved invitations

Displays a list of invitations that have been received but not acted upon. You may accept an invitation, leaving your current session. Decline, or discard the invitation - the sender is not notified of this action.

Next steps

Network

The Networking Toolkit

4/26/2021 • 18 minutes to read • Edit Online

Overview

Mesh functionality exists at many levels of the application. The core SDK plugin is available for multiple platforms and engines, but only provides the core Mesh service functionality. The Mesh Unity Toolkit, available through several packages, provides features that can be used to accelerate design and development of Mesh experiences built with Unity. The core Mesh SDK provides the lowest level networking functionality, while the Network portion of the Mesh Unity Toolkit provides several higher level networking features. These features also provide a much easier migration from Photon PUN.

Mesh SDK Data Transport Summary

Mesh provides many ways to send data over the network. The different features are optimized for different types of data, and the features exist at many levels. For more information on specific features, refer to the documentation. For a Unity app (not including writing your own native plugin) the features are:

MeshServiceManager

`ActiveMedia`: Camera sharing, screen sharing `CurrentTransport.Send/Broadcast` and `MessageReceived`: Send/receive arbitrary data messages to one or all participants `CurrentSession.State`: Read and write persistent, shared, key-value-pair strings and ints `CurrentSession.Frame`: Synchronize a shared spatial anchor

NetTransportBase (to subclass)

`Send/BroadcastMessage` and `OnMessageReceived`: Send/receive arbitrary data messages to one or all participants, routed by your class type `CallRPC`: Call a method in your class with arbitrary parameters, on one or all participants

NetObject (add component to game object)

`CallRPC`: Call a method on any component on the corresponding game object plus `NetObject` instance, with arbitrary parameters, on one or all participants

Components and systems

NetTransportBase

ParticipantManager

NetObjectManager

NetObject

NetMutex

NetLocalTransform

NetManipulable

Shared Space System

NetTransportBase

Overview

`NetTransportBase` encapsulates a wide variety of network message sending and receiving functionality. It does

not route network messages to specific game objects (for that, you can use `NetObject`), but it does route network messages to the class type. Thus, to use network messages from a singleton, inherit from `NetTransportBase`. Otherwise, to route messages to instances, use `NetObject` as a component. You can refer to `ParticipantManager.cs` and `NetObjectManager.cs` for examples of using `NetTransportBase`.

Prerequisite

- Inherit from `NetTransportBase`.
- Override `GetClassMessageType` to return a unique value for your class. You can add a constant to `NetTransportTypeIDs.cs` to prevent the same ID from being used for different classes. (In future versions of Mesh, this step may no longer be necessary.)

Sending messages

- `BroadcastMessage(int messageType, byte[] data, bool bReliable = true)` sends `messageType` and data to all other session participants.
- `BroadcastObject(int messageType, object obj, bool bReliable = true)` sends `messageType` and the object `obj` to all other session participants.
- `CallRPC(string method, params object[] parms)` calls this script's method, named `method`, on all other session participants, with parameters `parms`. The method must be tagged with the `[NetRPC]` attribute.
- `CallRPC(RPCFlags flags, int targetNetId, string method, params object[] parms)` calls this script's method, named `method`, on the session participant with net ID `targetNetId`, with parameters `parms`. See the `RPCFlags` section for more information. The method must be tagged with the `[NetRPC]` attribute.

Receiving messages

- Override `OnMessageReceived(int messageType, SessionParticipant sender, byte[] data)` to handle an incoming message to this script from sender. The message type can be read from `messageType`, and the message data is in `data`. It's up to the application logic to decode the data--for example, if an object was sent using `BroadcastObject`, then the data byte array can be decoded back into an object.

NetId getters

- `LocalNetId`: The network ID of the local participant (the one running the instance of the app).
- `MasterNetId`: The network ID of the session's master participant. See `ParticipantManager` for more info.
- `IsMaster`: True if the local participant is the session's master participant.
- `IsValidNetID(int netId)`: Returns true if `netId` is valid (in other words, it has been initialized and is recognized by everyone in the session).

Constants

- `InvalidNetId`: `IsValidNetID` compares network IDs against this constant.
- `NetIdBroadcast`: Use this constant with any message-sending method which takes a `targetNetId` parameter to send the message to every participant in the session.

RPC Utilities

- `InvokeRPC(object inst, string methodName, object[] parms)` invokes method named `methodName` on object `inst` with parameters `parms`.
- `InvokeGameObjectRPC(GameObject inst, string methodName, object[] parms)` invokes first method found named `methodName` among all MonoBehavior components on GameObject `inst`, with parameters `parms`.

Property Bag wrappers

- `async Task AddStringPropertyAsInt(string name, int value, System.Action<int> onNewValue = null)` attempts to add a new key/value pair of `name` and `value` to the session state. The value is stored internally as a string, so it cannot be used with `IncrementIntProperty`. If the property already exists in the session state, the callback `onNewValue` will be invoked with the existing value.
- `async Task GetStringPropertyAsInt(string name, System.Action<int> onValue)` gets the value of the string property `name`, as an integer value. `onValue` will be invoked with the current value, parsed as an integer.

- `DeleteProperty(string name)` removes a property from the session state.
- `async Task IncrementIntProperty(string name, int delta = 1, System.Action<int> onNewValue = null)` adds the value delta to the integer property `name`. `onNewValue` will be invoked with the resulting value of the operation, which may include changes made by other session participants' calls to `IncrementIntProperty`.

RPCFlags

The following enums can be bitwise OR'd together to control how RPCs are sent and invoked:

- `Default`: Does not invoke locally, is sent reliably.
- `LocalInvoke`: Invoke the RPC locally.
- `Unreliable`: Send the RPC message unreliably.
- `RemoteRPC`: Allows a remotely owned `NetObject` to invoke an RPC.
- `ReplayRPC`: Saves the RPC call in a list on the master participant (must be broadcast, reliable), which will be replayed for participants who join the session later.

ParticipantManager

Overview

`ParticipantManager` is a singleton (you can use the included prefab, or create your own instance) which assigns a unique integer network ID to each session participant. It also is responsible for selecting one of the session participants to be the "master." The master participant is responsible for recording and resending RPC calls made with the `ReplayRPC` RPC flag. `ParticipantManager` can also be used to conveniently associate custom application data with each session participant.

A participant or application is called "local" on the machine associated with that participant and application. All other participants or applications are "remote."

- `GetLocalParticipantId()` returns the local participant's full Mesh participant ID.
- `int GetParticipantNetId(string spId)` returns the network ID associated with the session participant with ID `spId`.
- `ParticipantData GetParticipantData(string spId)` returns the `ParticipantData` associated with the session participant with ID `spId`.
- `IsReady: True` when connected to a session and network IDs have been established for the current participants.

ParticipantData

This class associates the participant's Mesh participant ID, network ID, and `CustomData`. You can get the `ParticipantData` for a session participant using `GetParticipantData(string spId)`. Set and read `CustomData` as desired. Note that the value of this field is not automatically replicated over the network to other session participants. It is provided merely as a convenience to associate arbitrary application data with the session's participants.

NetObjectManager

Overview

`NetObjectManager` is a singleton (you can use the included prefab, or create your own instance) which enables the use of "Shared Objects." It is responsible for maintaining a map of network IDs to shared game object instances. This allows messages to be routed to the specific instances of a networked object in each participant's application. Shared Objects can be pre-placed in the scene hierarchy, and will be assigned a network ID automatically by `NetObjectManager`. The inspector for the `NetObjectManager` instance will show the shared object instances in the scene and their assigned network IDs. `GameObject.Destroy` can be used to destroy all instances of a shared object automatically. It is not necessary to use `DestroyNetObject(NetObject so)` to do this.

- `SpawnNetObject(GameObject prefab)` spawns an instance of `prefab` (which must have a `NetObject` component) for all participants in the session.

- `SpawnNetObject(NetObject prefab)` spawns an instance of `prefab` for all participants in the session.
- `SpawnNetObject(string prefabPath)` spawns an instance of the prefab at `prefabPath` (which must have a `NetObject` component) for all participants in the session.
- `NetObject GetNetObject(int netId)` returns the `NetObject` with network ID `netId`, if it exists.
- `IsReady`: True when connected to a session and network IDs have been established for the current participants.

NetObject

Overview

`NetObject` allows an instance of an object to be "shared" by all participants in a session. Each participant is running an instance of the application, which means that there is an instance of a particular "shared" object on each application. `NetObject` and `NetObjectManager` connect each of these shared object instances, so that their creation, destruction, and state can be synchronized across all instances of the application. For example:

- One participant loads a ball into the scene to share with the other participants. The instance of the ball needs to be created on each application.
- When the user moves the ball, a message needs to be sent to notify the other applications that the ball is moving.
- The user can create additional balls, and the instance of a particular ball must remain associated with its instances on the other applications.
- `NetObject` provides this by assigning a unique integer "network id" to all corresponding instances of a shared object. For example, the first ball created may have network ID 1, and the second ball created may have the network ID of 2. Each application in the session has two balls, one with net ID 1 and another with net ID 2.
- Messages sent to the ball with net ID 1 will be received by all applications and the message will be routed and handled only by the corresponding ball instances having the net ID of 1.

Each shared object is owned by one participant at a time, defaulting to the participant who spawned it. All participants have a copy of each shared object. If the shared object is owned by the local participant, it is considered "locally owned." If it corresponds to a remote participant, it is considered a "remote copy" and "remotely owned." Ownership doesn't imply any particular responsibility of the owner for the object; instead, it is a useful property to write distributed application logic. For example, typically, only locally owned objects should run logic that checks for user input and send messages to inform the remote copies what to do. The remote copies usually run logic to respond to those messages and do not check for user input or send messages.

- `IsReady`: The shared object is not ready to use until it has a valid net ID and a valid owner ID.
- `IsMine`: True if the shared object is locally owned. That is, returns true if the owner is the local participant.
- `NetId`: The network ID of the shared object.
- `OwnerId`: The network ID of the owning participant.
- `OwnershipTransferMode`: May be `Allow`, `Deny`, or `Request`. `Allow` allows the owner to be changed by anyone when they want. `Deny` will preemptively block requests made by remote copies, but if the object is locally owned, it may transfer ownership to any other owner when desired. `Request` is most typical, and when a remote copy requests ownership, a request is sent to the owner, who may then explicitly transfer ownership to the requesting participant if desired.
- `OnOwnershipTransferred(int [prevOwnerId])` is invoked when the owner ID changes. The `OwnerId` property represents the new and current participant when this is invoked.
- `OnReady()` is invoked when `IsReady` becomes true. That is, when it has been assigned a valid net ID and a valid owner ID.
- `TransferOwnership(int newOwnerNetId)` will request an ownership transfer to the participant with net ID `newOwnerNetId`. See `OwnershipTransferMode` for more information on how the request will be handled.

- To respond to an ownership transfer request, add a method tagged with `[NetRPC]` with signature `RPCRequestTransfer(int newOwnerNetId)` to any script on the game object with the shared object on it. Then call `TransferOwnership(newOwnerNetId)` to allow the transfer to the participant with net ID `newOwnerNetId`, or to disallow the transfer, do not call it.

RPCs

- `CallRPC(string method, params object[] parms)` invokes the method named `method` on all remote copies with parameters `parms`.
- `CallRPC(RPCFlags flags, string method, params object[] parms)` same as above, but with modified behavior based on flags. See `NetTransportBase.RPCFlags`.
- `CallRPC(RPCFlags flags, int targetNetId, string method, params object[] parms)` same as above, but can be used to invoke the RPC for the copy owned by the participant with the specified `targetNetId`. Pass `NetTransportBase.NetIdBroadcast` for `targetNetId` to call the RPC on all other participants.

NetMutex

Overview

`NetMutex` provides a mechanism for allowing multiple participants to robustly coordinate access to a specific resource, much like a mutex does in multithreaded programming. A networked application consists of multiple instances of the application running at the same time, communicating over the network, sharing data, and occasionally writing to the same pieces of data. Problems may arise if two participants simultaneously modify a portion of the shared state data. Instead, they must first establish a "lock" to determine who has permission to modify the data. `NetMutex` provides functionality that can be used to enforce safe patterns for accomplishing this, support common usage patterns, and help reduce boilerplate code necessary to support those patterns.

`NetMutex` works with a `NetObject` to manage the lock via the `NetObject`'s `Owner` property. Only the owner of the `NetObject` may safely modify its state. `NetMutex` helps wrap the flow which requests the lock via ownership transfer and provides callbacks to handle the various outcomes of that operation. For example, if participant's application B wishes to modify an object owned by participant's application A, they may request the ownership to be transferred. Application A may be currently using the resource and therefore can disallow the transfer, or it might not be using the resource and will allow the transfer. Application B may do something while waiting for the response and then appropriately handle the received response.

Only one `NetMutex` is needed to support any number of different actions on a shared object, but the shared object can only be owned or "locked" by one participant at a time. For example, you can support translation and painting in the same object behavior with one `NetMutex`, but you would need multiple shared objects and NetMutexes to support these operations for multiple participants simultaneously.

- `bool TryActionBegin(NetMutexAction action)` should be called when the application wants to perform a particular action on a shared resource. It will initiate the ownership transfer, and the callbacks configured on action will be executed to notify the caller when the appropriate events occur. See `NetMutexAction`.
- `OnActionEnd(NetMutexAction action)` should be called when the application finishes the action associated with action that was initiated with a call to `TryActionBegin`.
- `IsRemotelyActioned` returns true if the shared object is remotely owned and an action is currently occurring remotely.

NetMutexAction

This utility class wraps the events associated with one action that can be performed on a shared object. `BeginAction` and `CancelAction` are useful when your interactions begin or end in response to other event callbacks.

- `BeginAction(bool [isMine])` is invoked when the action can begin in various circumstances. `BeginAction` is invoked when `TryActionBegin` is called, with `isMine` set to true if the object is locally owned, and set to false if not. `BeginAction` will also be invoked with `isMine` set to true, when and if ownership is successfully

transferred to the local participant. The recommended pattern is to formally start your action when `BeginAction` is called. Cosmetic feedback should be provided if `isMine` is false-- however, if the action can be easily canceled, then just begin the real action as if the object is owned, and cancel it if the ownership lock fails.

- `CancelAction` is invoked when and if the ownership request fails, and also when `TryActionBegin` is called and the action has already begun remotely. See `NetManipulable` for an example of using `BeginAction` and `CancelAction` for an interaction which begins and ends from events.
- `startAction(bool [isMine])` is invoked when `TryActionBegin` is called, with `isMine` set to true if the object is locally owned, and false if not and ownership was requested.
- `ownership(bool [result])` is invoked when the ownership transfer is resolved, with the result being true if successfully transferred to the local participant, and false if not.
- `remotelyActioned` is invoked when `TryBeginAction` is called but the object is remotely owned and an action is known to be taking place.

NetLocalTransform

Overview

You can use the `NetLocalTransform` script to automatically synchronize a shared object's transform across all participants in the session. The current owner of the shared object informs the remote copies what the current local transform is, and the remote copies smoothly lerp to the received value. Note that this is a placeholder implementation, and that Mesh will soon provide a very robust and optimized system for synchronizing transforms. Ideally, the new system will replace the `NetLocalTransform` implementation, and the usage will remain as similar as possible. Note that synchronization of the transform uses the local transform properties. For mixed reality multiuser experiences, it is rare that the content will have the same placement relative to the Unity app's world space origin, so synchronizing transforms in the world space is rarely desired.

- `Transform Root`: This property can be overridden to synch the position of an arbitrary transform. If null, the transform on the shared object's game object will be synchronized.
- `float MotionSmoothness`: The value to use for lerping. Higher values result in faster interpolation.
- `float SendRate`: The frequency of transform updates. If the transform has not changed, nothing will be sent.
- `float ReliableSendDelay`: When the transform has stopped changing, no more unreliable updates will be sent. A reliable update will be sent after `ReliableSendDelay` seconds have elapsed when the transform stops changing.
- `float MaxReliableSendRate`: To avoid sending reliable updates too frequently, this is the maximum rate they will be sent.

NetManipulable

Overview

If you have an object that you can manipulate with the MRTK ObjectManipulator, and want that object to be a shared object which everyone in the session can manipulate, you must synchronize the transform, as well as handle the cases where multiple participants attempt to manipulate the object simultaneously. Simply add a `NetObject` and `NetManipulable` component (`NetMutex` will be added automatically by `NetManipulable`) to your object, and connect the `ObjectManipulator` to the NetManipulable's objectManipulator property.

`NetManipulable` will interface with the `ObjectManipulator` events and the `NetMutex` to ensure that only one participant can manipulate the object at a time. Ownership is transferred to the participant who begins manipulating the object, but the transfer request will be denied if the current owner is currently manipulating the object. A participant who begins manipulating a remotely owned object will observe the initial manipulation as normal, but if the ownership transfer request is denied due to another participant starting to manipulate first, then the first participant observes their manipulation terminating and the object will update to the proper state of being manipulated by the other participant.

Note that the default behavior can be modified so that the object can be "stolen" even while a participant is

manipulating it. The current owner would listen for the ownership change request (in `NetManipulable`, for example), and terminate their own current manipulation when they approve the transfer request.

Note that the application can implement "user permission" concepts such as requiring an object's owner to explicitly grant permission to other participants before allowing them to manipulate the object. These behaviors can be implemented by adding the appropriate checks when responding to the ownership transfer request, adding the appropriate checks when the interaction callbacks begin, etc.

Shared Space System

Overview

Most multiuser experiences involve the participants seeing essentially the same thing, and a big part of that is *where* the things are. Even if participants are not in the same physical location, it's still likely that the design will anticipate that they should see the same virtual scene. For example, if someone places a car model next to a tree model, and draws an arrow pointing to the car, then everyone else would ideally see the same thing--in other words, they should see a car next to a tree with an arrow pointing to the car. Furthermore, we want users to be able to navigate through the virtual scene, or to position the virtual scene within their physical space as desired (see the Centerpiece sample for more info on this).

These important aspects can be elegantly supported by placing all virtual objects in the same coordinate system; in Unity, place all shared game objects as children of a common *shared world root* transform. A participant can move their virtual scene to wherever they like in their physical space and it will not change how the code calculates or handles the shared object transforms. And when communicating locations, directions, transforms, and so forth, the local versions of these can be directly sent in messages without requiring calculations to convert to another space. There may certainly be certain situations where conversion is necessary, but this technique works in most cases.

A few helper classes have been provided to make setting this up very easy. They are:

- `SharedSpaceManager`
- `SharedSpaceRoot`
- `SharedSpaceObject`
- `Centerpiece`
- `MeshSessionOrigin`

SharedSpaceManager

`SharedSpaceManager` provides some handy functionality for working with the shared spaces and supporting the other shared space components. It will automatically look for a `SharedSpaceRoot` object in the scene and, if not found, will create a default one. This root object can be manipulated by the application or the user. It has a method:

- `ParentUnderSharedSpaceRoot(GameObject sharedSpaceObject, ReparentTransformOptions reparentModes)`: Call this to manually reparent a game object to the current shared space root (`SharedSpaceObject` automatically does this). Use `reparentMode` to define what happens to the objects transform.
- `ReparentTransformModes`:
  - `KeepWorldTransform`: Local transform will be recalculated so that the object remains at the same world location at which it was spawned.
  - `KeepLocalTransform`: Local transform will not be altered; therefore, the object will likely move to a new world location when reparenting.
  - `ZeroLocalTransform`: Object is placed at the shared space root's transform.

SharedSpaceRoot

`SharedSpaceRoot` can be added to a game object in your scene to tag it as the object that you want to be used as the root transform for `SharedSpaceObjects`. It has `Type` and `Mode` options that are not currently implemented.

SharedSpaceObject

If you add the `SharedSpaceObject` to a game object, it automatically reparents itself as a child of the current shared space root via `SharedSpaceManager.ParentUnderSharedSpaceRoot()`.

- `ReparentMode`: Set the mode by which the object's transform will be updated when reparenting (see `SharedSpaceManager.ReparentTransformModes`).

Next steps

Avatar package

The Avatar Package

4/26/2021 • 10 minutes to read • Edit Online

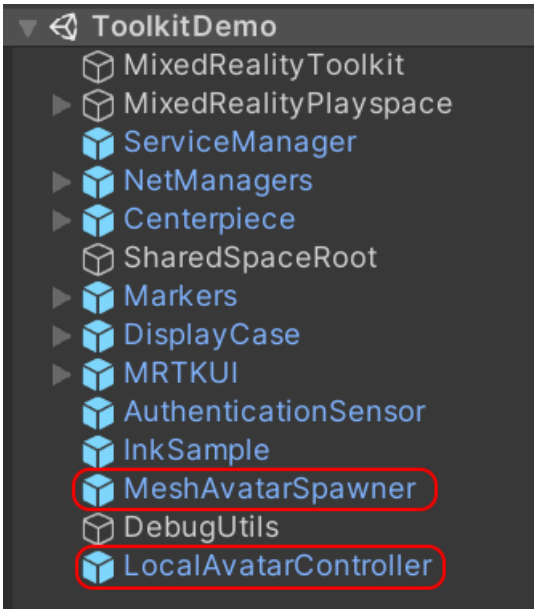
Overview

With the avatar package in your project, you can spawn and share a virtual representation of yourself with other participants in the session. Input from a participant's head, hands, and eyes are projected onto an animated avatar representation. That avatar is shared with all participants in the collaborative session. The package also provides spatial audio support.

Avatars in the ToolkitDemo scene

To start, import the Mesh Tools for Unity packages. Your project will have all the required components for a spatial session with avatars and spatial audio.

To use ToolkitDemo avatars in your project, drag the **MeshAvatarSpawner** and **LocalAvatarController** prefabs into the scene. The former comes with the base setup to spawn avatars with spatial audio support when participants join the session. The latter allows animating your own avatar using sensor tracking from any supported input device.



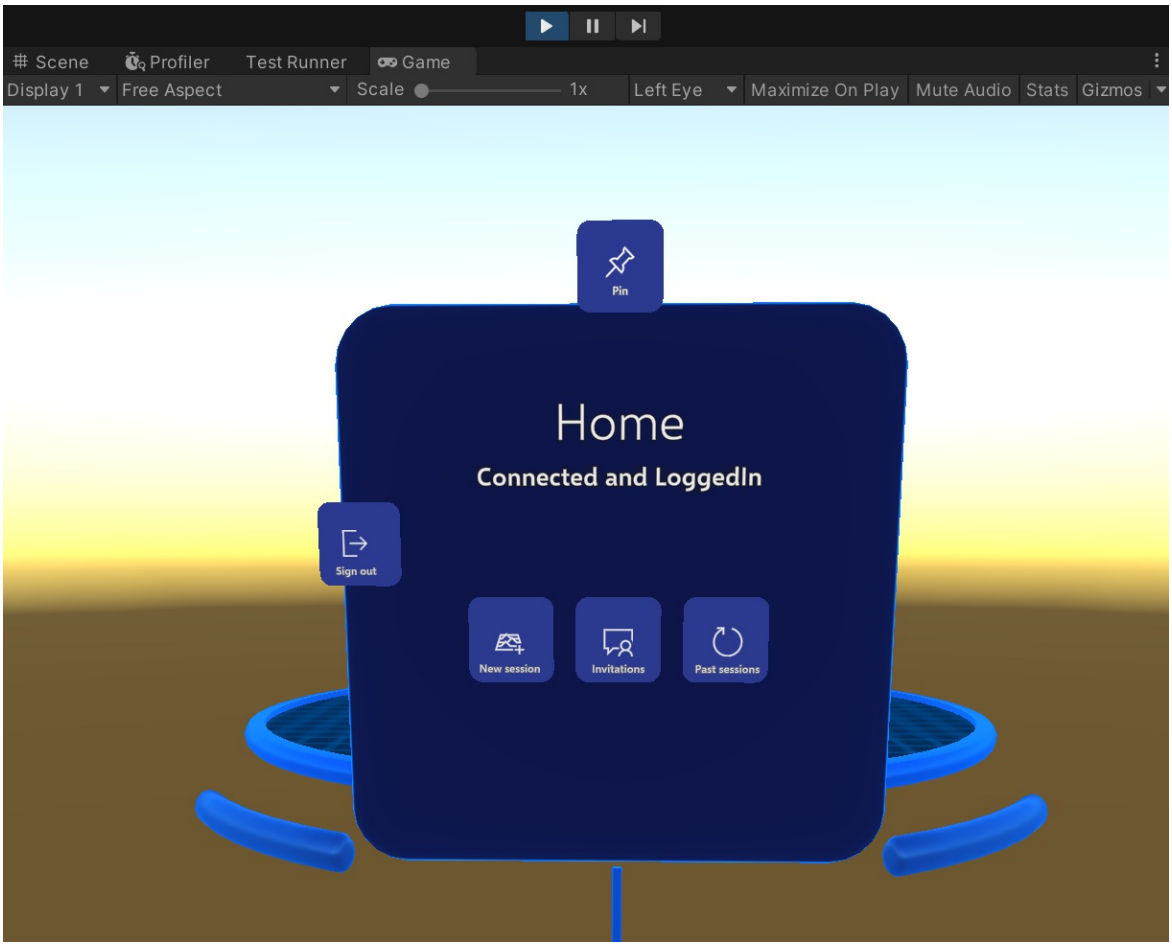
Running the ToolkitDemo scene

1. In the Project window of the Unity editor, navigate to **Packages** > **Mesh Tools for Unity (UX)** > **_Shared** > **Scenes**, and open the ToolkitDemo scene.

   > **NOTE**
   > If you haven't opened a scene with TMP in this project before you will be prompted to import TMP Essentials. Confirm the importing of TMP Essentials and **reopen the scene**.

2. Click the Play button.

3. Note that the session menu in the scene displays the title **Home**. If the panel doesn't appear in front of your camera, it won't be visible in the **Game** window. In that case, you can look around to find it. In the hierarchy, the GameObject is called *MRTKUI*.



4. In the session menu, click **New Session**. The title in the object changes to "Name your session." Naming your session is optional and can be skipped by leaving the input field blank and clicking **Create**.

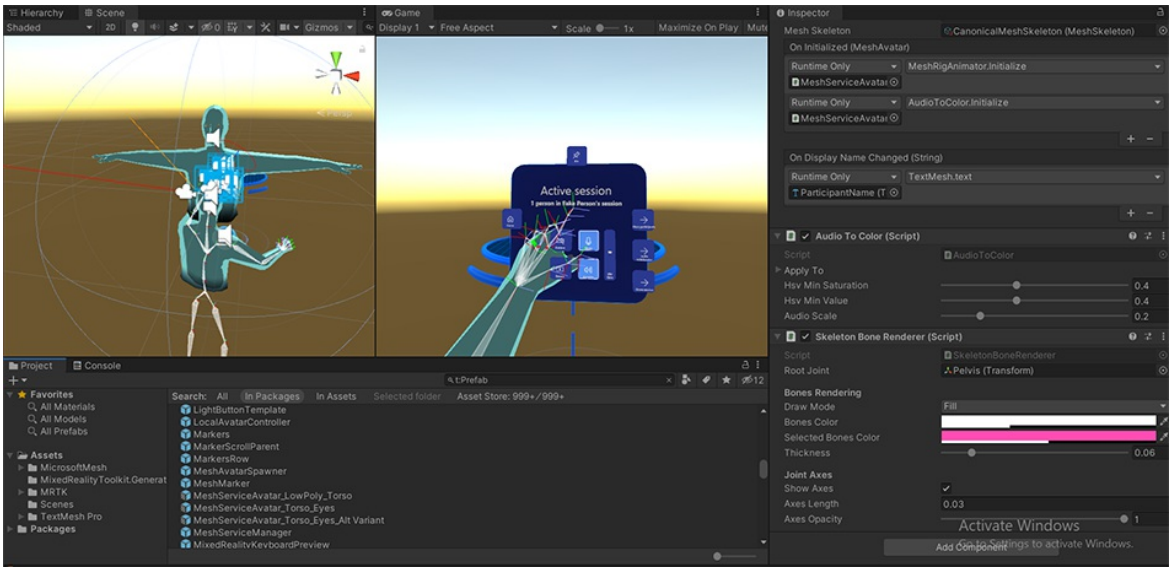
5. A yellow rotating cube will indicate that the collaboration session is being created. Once the cube disappeared, the UI will progress to the invitation screen indicated by the title showing **Invite someone**. You can skip this step and invite other participants at a later time. Confirm by clicking **Next**.

6. Once the session UI title switches to **Active session** you're part of an active session and your local avatar will be spawned. You can observe your avatar in the **Scene** view. You can also activate the simulated hands provided by MRTK to observe it in the **Game** view.

7. Use MRTK input simulation to enable the simulated hands and have the avatars hands appear in the game. (Simulated hands can be spawned by pressing the **T** key for left and **Y** key for the right hand).

8. You can observe the full skeleton by enabling the `SkeletonBoneRenderer` component located on the spawned avatar, which is a GameObject parented to the Centerpiece. By default this component is disabled in the demo scene and prefabs, as it's a debug tool.

9. If you want to observe your full avatar as you move around, change your layout so that you can see the **Scene** and **Game** views at the same time:



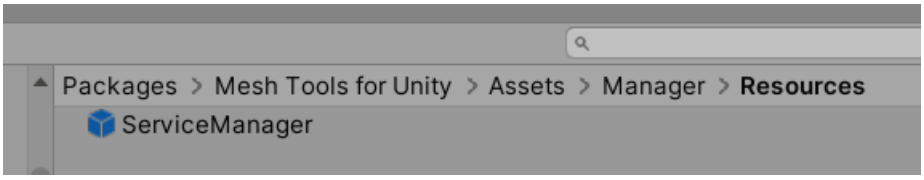
When running the **ToolkitDemo** scene on your HoloLens 2, the spawned avatars are mapped to the head, hands, and eye pose of each participant. By default, an avatar is also spawned for the local participant for consistency. Alternatively, the device hand mesh can be activated in MRTK to reduce the visual lag between the actual hand movement and the rendered mesh. In that case, the local participant avatar spawning can be avoided with the **Spawn Own Avatar Mode** property of the MeshAvatarSpawner component.

Adding avatars to an existing scene

Mesh Tools for Unity come with a range of preconfigured prefabs. They allow you to drag and drop spatial session support, including avatars, into your own scene. The following guide will help you getting started integrating avatars into your own scene:

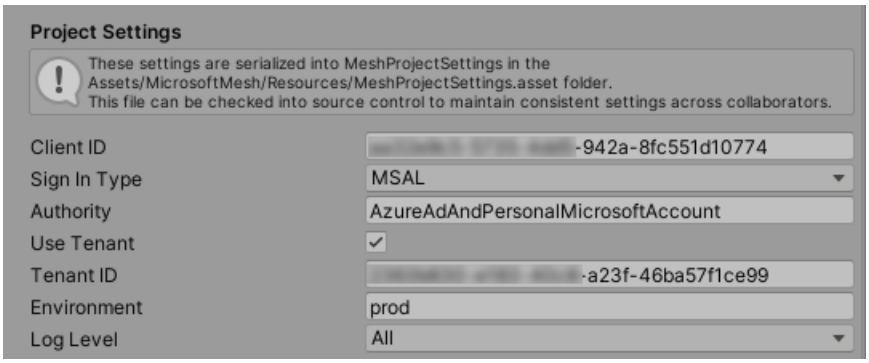
1. Follow the quickstart guide to set up all prerequisites and import the MRTK Collab packages into your project.

2. In the project panel, navigate to **Packages** > **Mesh Tools for Unity** > **Assets** > **Manager** > **Resources**, and then drag the **MeshServiceManager** prefab into the **Hierarchy**. This prefab gives you a session manager singleton that you can access from code. This manager is relied upon by the UX and Mesh Avatars.

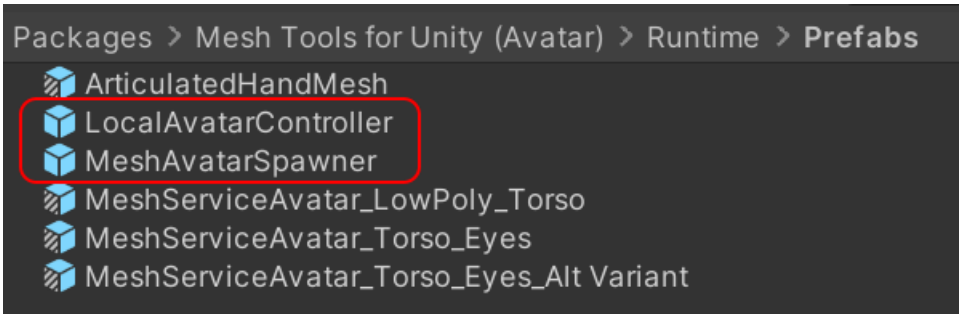
   `MeshServiceManager` comes with various responsibilities related to creating a collaborative session and session media like video and audio. Most components don't need any configuration and just have to be present in the scene. However the `MeshServiceManager` needs adjustments to connect to the Microsoft Mesh service.

3. With the **MeshServiceManager** prefab still selected, look in the **Inspector** and, in the **Project Settings** section inside the Mesh service manager script, add your credentials (client ID, tenant ID).



4. In the **Project** panel, navigate to **Mesh Tools for Unity (UX)** > **MRTKUI** >, and then drag the **MRTKUI** prefab to the **Hierarchy**. This adds a session flow UI to the scene, which will allow to create a collaborative session and invite users.

5. In the **Project** panel, navigate to **Mesh Tools for Unity (Avatar)** > **Runtime** > **Prefabs**, and then drag the **LocalAvatarController** and the **MeshAvatarSpawner** prefabs to the **Hierarchy**.

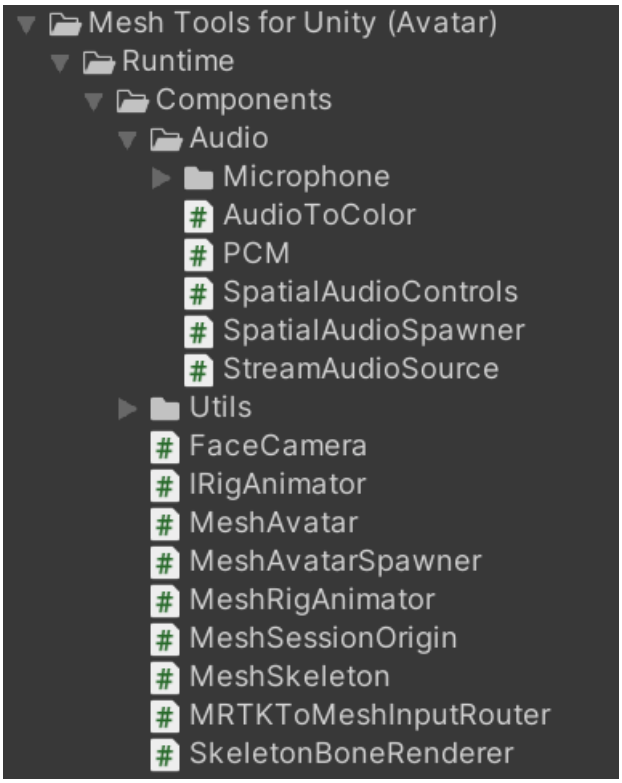


   The MeshAvatarSpawner prefab consists of two spawner components. The MeshAvatarSpawner component is responsible for spawning avatars in the scene. The `SpatialAudioSpawner` component adds spatial audio support for each avatar spawned by the avatar spawner. The LocalAvatarController prefab contains a single component, `MRTKToMeshInputRouter`. The component forwards inputs like head and hands transforms to the Mesh service. It relies on the MRTK input abstraction. Those inputs are then used to animate the avatar of the local user, and shared with other participants in the sessions.

6. At this point, you can click the **Play** button and continue on with the steps listed in the previous section.

Customizing avatars

The **Packages** > **Mesh Tools for Unity (Avatar)** > **Runtime** > **Components** folder contains the scripts that are used to implement the **MeshAvatarSpawner** prefab. You can use some of them as a starting point for your own customizations.



Let's look at a few of the scripts in detail.

Mesh Avatar

The `MeshAvatar` component is a central "glue" component that binds the other avatar components to the avatar and session participant. This component should be added to the prefab used for the session participant avatars. The `MeshAvatarSpawner` (see below) will automatically initialize this component if present on the instantiated prefab.

Mesh Skeleton

The `MeshAvatar.MeshSkeleton` property references an asset of type `MeshSkeleton` (derived from `ScriptableObject`) which contains the description of the skeleton to use when creating the avatar. This description is composed of a JSON file that contains the destination-to-source joint mapping, and the set of joint transforms forming the bind pose of the skeleton. The only supported assets are the ones found in **Runtime** > **JointMapping**:

- `CanonicalMeshSkeleton.asset` for the default development avatar used in the ToolkitDemo scene
- `AltspaceMeshSkeleton.asset` for the customizable avatar based on the AltspaceVR avatar library (see the **Mesh Tools for Unity (Avatar - Customizer)** package)

Initialized Event

An event raised when the MeshAvatarSpawner spawns a new avatar. The event handler receives as arguments the newly initialized MeshAvatar object. It's critical to connect this event to the MeshRigAnimator.Initialize() method.

Display Name Changed Event

An event raised when the display name of the avatar associated with the component changed. This event allows updating any text object showing the participant name.

Mesh Avatar Spawner

The `MeshAvatarSpawner` component is responsible for spawning avatars in the scene. It's attached to the **MeshAvatarSpawner** game object. Spawned avatars are either parented to a custom defined game object that can be linked in the `AvatarParent` property or if none is selected they'll be attached to the **MeshAvatarSpawner** game object owning the component. The Mesh Avatar Spawner listens to the session events for participants joining or leaving the session to spawn or destroy their avatars.

The following fields are exposed by the Mesh Avatar Spawner and can be modified for custom avatar spawning behavior:

Avatar Prefab

Defines the prefab the spawner will use to create the avatar instance per participant of the session.

Local Avatar Prefab

Defines an override prefab used for the local participant in place of `AvatarPrefab`. If not specified, `AvatarPrefab` is used by default.

Avatar Parent

The avatar parent will indicate the game object the spawned avatars will be parented under. This parenting is useful when working with a session origin to allow participants to anchor their scene on custom points in their environment. For example, the center piece GameObject can be used as parent to allow participants to move the entire collaboration scene in their real life room. This feature is useful if the initial scene starting position is blocked by walls or other obstacles. The default behavior for this field, in case no parent game object is linked, is to use the game object the `MeshAvatarSpawner` component is attached to.

Spawn Own Avatar Mode

This property can be modified to control if there should be an avatar spawned for the current user. You might encounter situations or devices where the generated avatar for your own user is not as suitable as, for example, a device provided hand mesh. In those cases, you can control the spawning behavior with this flag. The setting differentiates between player and editor builds to allow testing with a generated avatar and hand mesh in unity editor while running the app with device hands only on, for example, a HoloLens 2 device.

Avatar Spawned Event

An event raised when the `MeshAvatarSpawner` spawns a new avatar. The event handler receives as arguments the newly instantiated GameObject, and a string containing the identifier of the `SessionParticipant` the avatar was spawned for.

Avatar Removed Event

An event raised when the `MeshAvatarSpawner` despawns an existing avatar. The event handler receives the same arguments as `OnAvatarSpawned`.

Spatial Audio Spawner

The `SpatialAudioSpawner` component manages spatial audio in the scene by attaching audio stream sources to spawned avatars. It further sends microphone data from the current participant to the server on update so it's available for other participants to playback on the generated avatar.

MeshAvatar spawner

This property should be linked to the avatar spawner component so the spatial audio spawner can register to its `OnAvatarSpawned` and `OnAvatarRemoved` events to attach or detach the spatial audio components. If nothing is set, the `SpatialAudioSpawner` component will look for the `MeshAvatarSpawner` component on the same GameObject it is attached to.

Master Mixer

The master mixer that will be plugged into the audio source attached to the avatars. The master mixer has to include a 'Master' `AudioMixerGroup`. If master mixer is not set the default master mixer of the avatar package will be used.

MRTK to Mesh Input Router

This script takes the sensor data from device (for example, HoloLens 2) by querying MRTK and feeds it into the mesh service in order to run the pose estimation on the server. For most use cases, you won't have to change this component as the default configuration will work out of the box. However if you're working with for example, a moveable session origin like the center piece, the property **Session Origin** needs to be linked to the session origin game object.

MeshRigAnimator

The `MeshRigAnimator` component animates a simple rig with the pose data coming out of the Microsoft Mesh service. `MeshRigAnimator` also holds the voice source property, which will allow to play spatialized audio on an avatar. To enable this feature, set the voice source to a GameObject with an attached `AudioSource` inside the avatar prefab (usually around the location of the mouth). This component should be part of your avatar prefab with the following property configured:

Root Transform

Link the root `Transform` component of the skeleton hierarchy here, which is in most cases the pelvis or the hips.

Joint Mapping File

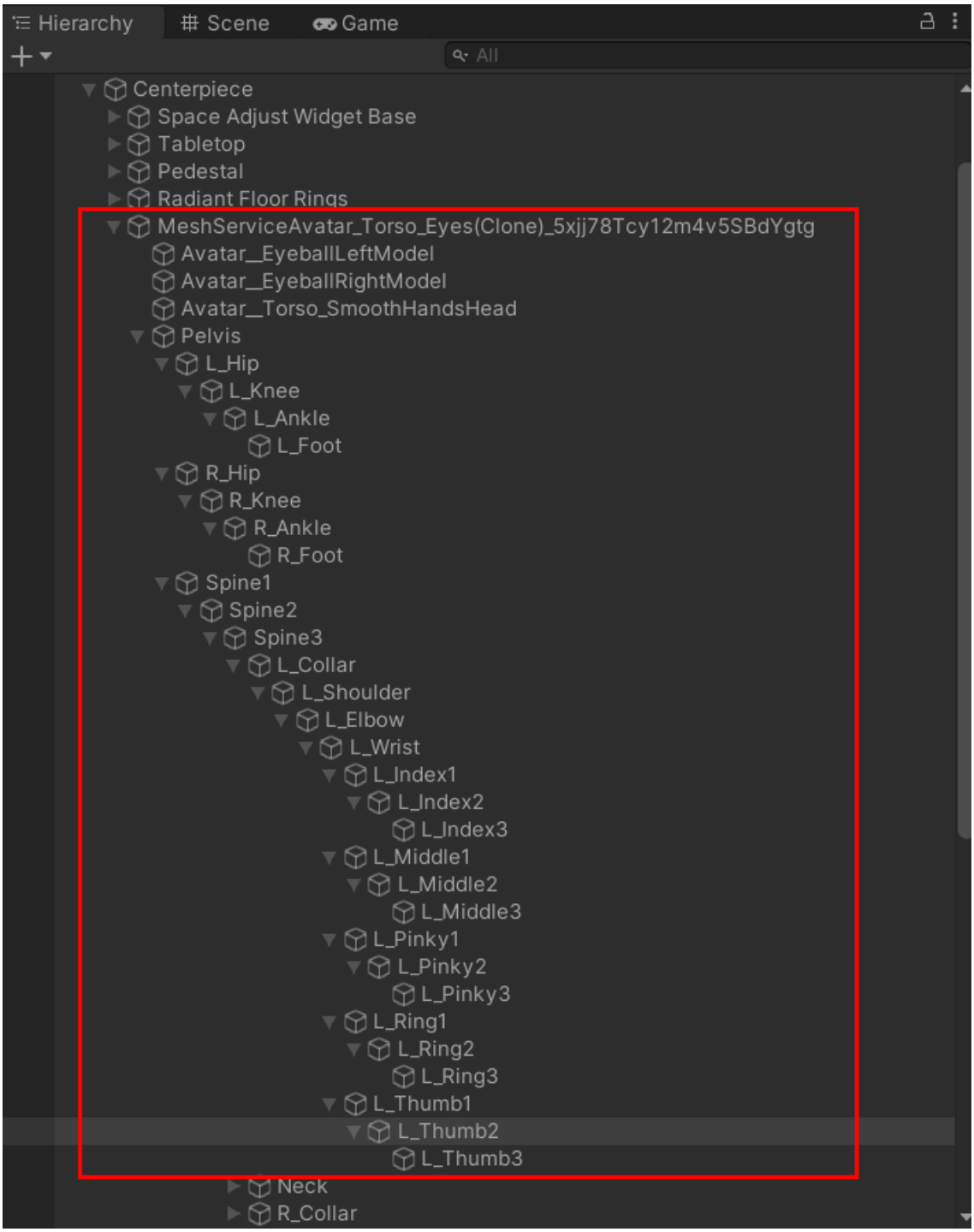
The JSON file that contains the destination-to-source joint mapping. For an example look at the `Identity_mapping.json` file in the MRTK toolkit avatar package.

Eye Transform Left / Right

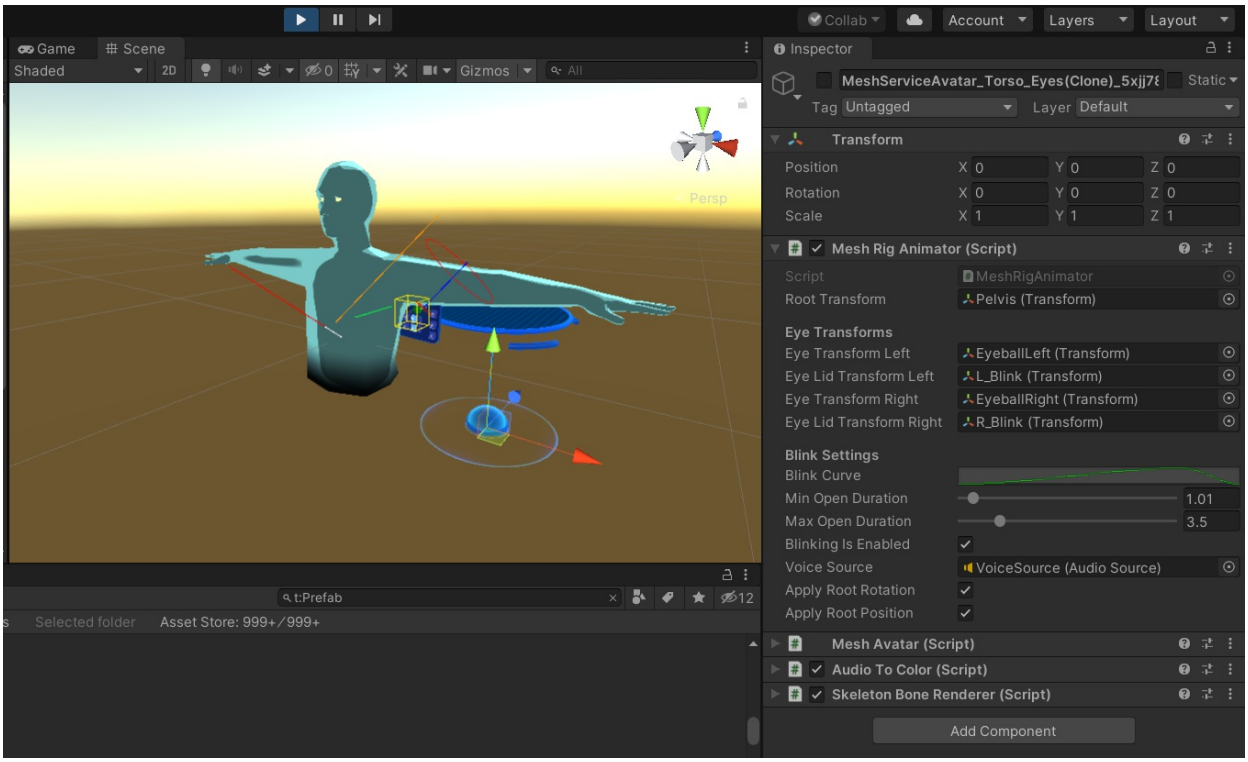
This needs to link to the left and right eye mesh of the avatar to allow for eye tracking values to be applied.

The Sample Avatar

The sample avatar is the default avatar spawned for every participant who joins a session. It is based on the canonical Microsoft Mesh skeleton:



The avatar contains a few of the scripts just mentioned:



In Summary: With `MeshAvatar`, `MeshRigAnimator`, and a simple-skeletoned asset, you have the minimum you need to have a fully posed avatar animated with Mesh.

You can animate this avatar without having to think about its rig.

Next steps

Avatars Overview

Mesh Tools for Unity Service Manager

4/26/2021 • 2 minutes to read • Edit Online

> **WARNING**
>
> **This is an experimental, in-preview feature**. Some of the features of the Mesh Tools for Unity, while not yet fully fleshed out, have enough value at this stage that we can recommend that you take a look. We label them "experimental" to indicate that they're still evolving and subject to change over time.

The *MeshServiceManager* is the gateway to using Mesh with Unity in a way that feels familiar to Unity developers.

> **IMPORTANT**
>
> Any scene that wants to access Mesh services using the Mesh Tools must ensure that there is at least one instance of the `MeshServiceManager` component on a GameObject in the scene.

Features

- Client lifetime management
- Connection to the service
- Participant Presence
- Authentication and Sign-in *(MeshHelpers)*
- Transport setup
- Session creation and join
- Session invitation management
- Spatial session management
- Participant management
- Graph profile query
- Logging
- Video device setup *(VideoManager)*
- Spatial session update (AdvanceTimeTo)
- Unity event queue for dispatching messages to main Unity thread

Prerequisites

- Mesh SDK

Setup

Add *MeshServiceManager* to a GameObject in the scene.

> **TIP**
>
> A prefab is provided that contains a number of components useful for typical session needs:
> `Packages/com.microsoft.mesh.toolkit/Assets/Manager/Resources/MeshServiceManager.prefab`

Configuration

Changes to some of the settings are stored as a *ScriptedObject* in the project.
`Assets/MicrosoftMesh/Resources/MeshProjectSettings.asset`

The most critical of these are *ClientID* and *UseTenant* and *TenantID*. These values can be retrieved from the Azure Portal. Choose an App from the App registrations and note the *Application (client) ID* and *Directory (tenant) ID*

Related

- How to find your Azure Active Directory tenant ID

Release Notes

4/12/2021 • 2 minutes to read • Edit Online

SDK release notes

Mesh tools release notes

Troubleshooting Common Error Messages

4/15/2021 • 2 minutes to read • Edit Online

You may run into the following errors throughout development.

If you've need additional assistance, contact your Microsoft representative with details. If you can, collect a log with `ClientLogLevel.All` while reproducing the error.

Customer cannot consent to "Microsoft Mesh (Preview)" app

```
AADSTS50020: User account 'account' from identity provider 'live.com ' does not exist in tenant 'tenant
name' and cannot access the application '98e6160b-4308-432a-b82d-ed6fce38dfbf'(Microsoft Mesh (Preview)) in
that tenant. The account needs to be added as an external user in the tenant first. Sign out and sign in
again with a different Azure Active Directory user account.
```

You are trying to log in to an Azure Active Directory tenant with a Microsoft Account (MSA). Instead, log in as a user from the Azure AD tenant.

Client app cannot acquire token

```
AADSTS650052: The app needs access to a service (api://98e6160b-4308-432a-b82d-ed6fce38dfbf) that your
organization tenant-id has not subscribed to or enabled. Contact your IT Admin to review the configuration
of your service subscriptions."
```

The Microsoft Mesh (Preview) app has not been registered in your Azure AD tenant. Make sure that you've granted the Microsoft Mesh (Preview) app access to your tenant. When the "Microsoft Mesh (Preview)" app (id: 98e6160b-4308-432a-b82d-ed6fce38dfbf) has been granted access, you can find it in your AAD tenant under **Enterprise Apps**.

401 Unauthorized responses from the Mesh service

- The user's token has expired: ensure that your `TokenRequiredDelegate` acquires a new token when it is called.
- The user's token might not have the right permissions for Mesh: make sure when acquiring a token that you request one with the permissions returned by `CollaborationClient.GetTokenScopes()`.

403 Forbidden responses from the Mesh service

- The user's token does not have the right permissions: make sure when logging the user in that you request a token with the permissions returned by `CollaborationClient.GetTokenScopes()`.

Client requests access to resource that is not listed

```
AADSTS650057: Invalid resource. The client has requested access to a resource which is not listed in the
requested permissions in the client's application registration. Client app ID: e9a72d39-9a8e-4035-a818-
ec7387e0f624(test-cpp-getting-started). Resource value from request: api://98e6160b-4308-432a-b82d-
ed6fce38dfbf. Resource app ID: 98e6160b-4308-432a-b82d-ed6fce38dfbf. List of valid resources from app
registration: 00000003-0000-0000-c000-000000000000.
```

The app is missing permissions. Make sure the `user_impersonation` scope for the "Microsoft Mesh (Preview)"

API is requested as part of the application registration.