Quickstart Guide

Quickstart Guide Version 1.2 (2021-01-20)

Version History

Version	Date	SpaceHub Version	Changes
1.2	2021-01-20	0.19	Added Addressables documentation and 0.19 upgrade guide
1.1	2020-11-04	0.18	Added SpaceHub Groups and Laravel setup chapters
1.0	2020-09-04	0.15	Initial version

Table of contents

Upgrade guide: 0.18 to 0.19

- Upgrade to 2019.4.16f1
- Close Unity
- Delete the following files and folders and their corresponding .meta files:
    - folder Assets\NavMeshComponents
    - file Assets\Scripts\Avatars\Customization\CustomizationArtCategory.cs
    - file Assets\Scripts\Avatars\Customization\CustomizationArtPack.cs

    - file Assets\Scripts\Avatars\Customization\CustomizationAvatarPresets.cs
    - file Assets\Scripts\Interactables\ElevatorFloorsData.cs
    - file Assets\Editor\ConferenceBuildProcessor.cs
- Copy 0.19.240 to your project
- Open Unity
- Open Conference Server Settings Prefab (either through project or through any instance in the spacehub scenes)
- Press "Create" button to create a new Data object which holds your server settings
- Press "Migrate Old AppIDs to Scriptable Object" to migrate your settings
- Build Addressables once:
    - Open Window > Asset Management > Addressables > Groups
    - Select Play mode Script > Use existing build
    - Select Build > New Build > Default Build Script

Now you are ready to go

About SpaceHub Conference

SpaceHub Conference is a script reference and code architecture showcase to demonstrate how you can use Photon to support many users at the same time in a realtime application. Our goal is to provide you with an example project which you can study to get inspired for your own project and to show you how to implement a bunch of different features.

SpaceHub Conference is not a ready to use product. It is intended solely for educational purposes and provided As-Is! You will need expertise in Unity, C# Development and Photons Multiplayer technology to implement your own solutions.

This guide gives you a very brief overview of the different features we implemented in SpaceHub Conference and some pointers where you can start to dig in to to learn about each of these features.

SpaceHub Conference is built using the Photon products Realtime, Voice and Chat. With SpaceHub we demonstrate how to best utilize these products to create an interactive communications experience for large scale applications.

Helpful links to get started

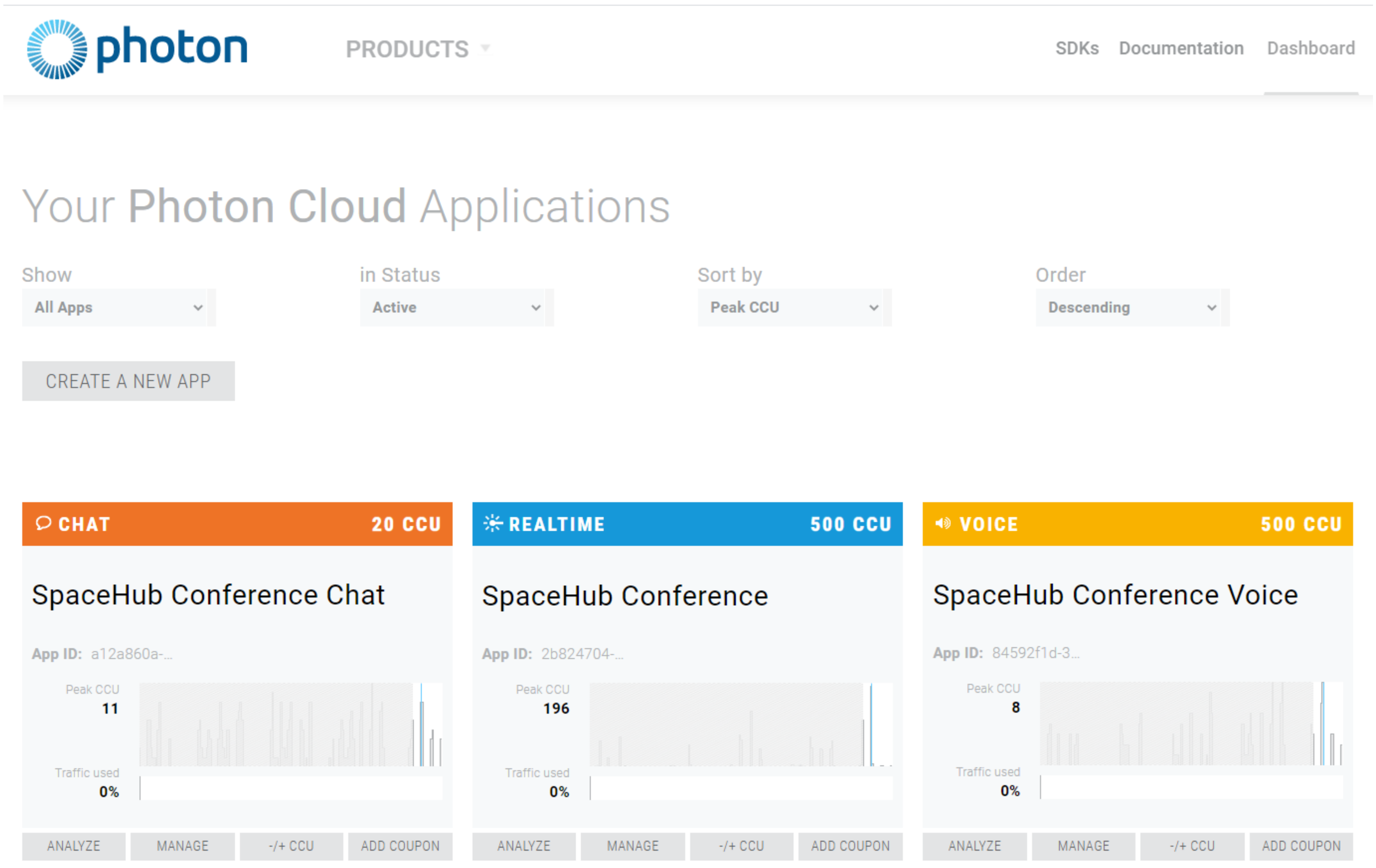
- spacehub.world - The main website for SpaceHub
- spacehub.world/changelog - Changelogs to quickly read what's new in each SpaceHub version
- photonengine.com - Main Photon website

Structure and setup

SpaceHub consists of three main building blocks: The Photon Cloud products, the Unity application and a website/REST service. This document will give you an overview of each building block and how they work together.

1. Photon Cloud

To be able to use each Photon product (Realtime, Voice and Chat), you will need to generate an AppId on the Photon website. After you have registered your own account, you can click on your profile in the top right and click on "Your applications". On this page you can create one application per Photon product. Afterwards your dashboard should look roughly like this:


PRODUCTS

SDKs Documentation Dashboard

Your **Photon Cloud** Applications

Show: All Apps

in Status: Active

Sort by: Peak CCU

Order: Descending

CREATE A NEW APP

CHAT 20 CCU
SpaceHub Conference Chat
App ID: a12a860a-...
Peak CCU 11
Traffic used 0%
ANALYZE MANAGE -/+ CCU ADD COUPON

REALTIME 500 CCU
SpaceHub Conference
App ID: 2b824704-...
Peak CCU 196
Traffic used 0%
ANALYZE MANAGE -/+ CCU ADD COUPON

VOICE 500 CCU
SpaceHub Conference Voice
App ID: 84592f1d-3...
Peak CCU 8
Traffic used 0%
ANALYZE MANAGE -/+ CCU ADD COUPON

You will get a unique App ID for each of the three products, which you will need to connect to the Photon services with your Unity application.

2. Unity Application

You can always download the most recent version of the Unity application by logging into your Photon Circle Member account on https://spacehub.world

In order to be able to press play and connect to your own SpaceHub Conference, you need to setup the Photon App Ids. Your main entry point into the application is the Expo scene (Assets/Scenes/Expo.unity). Here you find the "Conference Server Settings" Game Object which will prompt you for the App Ids of your Photon Products. You can leave the Api Url empty for now, we will introduce this in the Web Application Features section. Copy and paste your App Ids in the appropriate fields.

You should also build the included addressables once. If you don't want to use the addressables system (more info below), you can remove the packages for your project, but for demonstration purposes we included a short example how this works.

- Build Addressables once:
  - Open Window > Asset Management > Addressables > Groups
  - Select Play mode Script > Use existing build
  - Select Build > New Build > Default Build Script
- If you want to test the addessable demo scene:
  - Open your ConferenceServerSettings in the unity inspector
  - Make sure the "Default Addressable Catalogs" array is empty
  - Now a helper box appears for the addressables
  - Click "Add SpaceHub Demo Addressable Data" to fill out the demo data
  - Now you are able to enter the special "Island Booth" door, which loads an external scene via addressables

Now you are able to join your Photon server.

There are thee main scenes you can start to dig into:

Assets/Scenes/Funnel.unity

This scene implements the main entrypoint of our demo applications. Here users are prompted to setup their Graphics settings, Audio settings, Login into the service and select events.

Assets/Scenes/CustomizationRoom.unity

After the Funnel, users get into this scene to customize their avatar and how they will look in SpaceHub Conference. Once they're ready they can press "Join" to actually join the conference.

Assets/Scenes/Expo.unity

This is the main workhorse scene for SpaceHub Conference. It loads all the managers, server connectors and the player avatar setup and deals with all the complicated functionality. Each conference level (like "ExpoRoomMain") only contains the visuals and fuctionality for that specific scene and they're loaded additively on top of the Expo scene, which is always present.

C# Managers

We use a lot of manager classes, which are singleton instances, to separate the code into different responsibilities. Manager classes are often the main entry point to a feature and they usually have a public static Instance variable to access them.

For example

```
SpacesManager.Instance.RequestSpaces().
```

Most managers are setup in the Expo.unity scene in the Managers GameObject.

3. Website

The web part of SpaceHub Conference is **optional**, which means the Unity Application will run fine without it, but it allows for more advanced functionality like User or Exhibitor management. Just like the Unity application, the example source code can be downloaded on your spacehub.world Dashboard.

Our example website is a basic application built on top of the Laravel PHP Framework. It demonstrates how to use advanced Photon Features like WebHooks, WebRPCs and Custom Authentication. You don't have to use PHP or Laravel to implement the same features. As long as you can serve a RESTful API through an online website, the same principles will apply.

We won't go into details how to setup your own webserver, as this is beyond the scope of this document, so we simply focus on how to setup Photon to use your REST endpoints. The Web Application Features section will go into more details which API endpoints need to be setup and how to connect them to the Photon service.

Laravel setup

1. Create a new folder in your web directory and extract the web source code into that folder.
2. Rename .env.example to .env and fill in your specific configurations (you need to at least setup a database driver)
3. Run the following CLI commands in the root directory of your spacehub conference web app
    - Run `` `composer install` ``
    - Run `` `npm install` ``
    - Run `` `php artisan key:generate` ``
    - Run `` `php artisan migrate` ``
4. Setup directory permissions as described in the laravel installation manual
5. Make sure you setup your server to serve the laravel app via a top-level domain. For example: http://spacehub.test

On Windows (XAMPP)

Configure a virtual host in the `/xampp/apache/conf/extra/httpd-vhosts.conf` file like so:

```
<VirtualHost spacehub.test:80>
    ServerName spacehub.test
    DocumentRoot "c:/xampp/htdocs/spacehub-conference-admin/public"
    Redirect 301 / https://spacehub.test/
</VirtualHost>
```

```
<VirtualHost spacehub.test:443>
    ServerName spacehub.test
    DocumentRoot "c:/xampp/htdocs/spacehub-conference-admin/public"

    SSLEngine on
    SSLCertificateFile "conf/ssl.crt/server.crt"
    SSLCertificateKeyFile "conf/ssl.key/server.key"

    <Directory "c:/xampp/htdocs/spacehub-conference-admin">
    </Directory>
</VirtualHost>
```

Also edit the `C:\Windows\System32\drivers\etc\hosts` file to map your test domain name to your localhost ip:

```
127.0.0.1 spacehub.test
```

Mac or Linux

It's easiest to use Laravel Valet for your test deployment: https://laravel.com/docs/8.x/valet

Unity Application Features

Since SpaceHub Conference is an example implementation, we won't go very far into explaining the implementation details of each feature here. This list simply serves to give you a quick overview of all the features we implemented and which Unity scenes and C# classes you can dig into to get started to learn about these features.

List of features

- Expo
- XR for Oculus Quest
- SpaceHub Groups
  - GroupsManager
  - GroupsViewManager
  - Connections
  - IGroupPropertiesStorage
  - Group
  - Custom Chat Messages
  - Users
  - Media Stack
- Voice Circles

- Chatbubbles
- Sandbox
- Video playback with Vimeo
- Cinema
- Stage
- Avatar customization
- Avatar Art Packs
- User authentication
- Admin tools and user moderation
- Minimaps
- Spaces
- Space Elevator
- Addressables

Expo

This is the main meat of SpaceHub Conference. How do you design an app that handles large amounts of users? The key is to send as little data over the network as possible while also trying to send data to as few other users as possible. Photon measures and limits how many *messages per second* you send in each room in order to ensure the Photon Cloud is running for all users.

For example: For position updates and data we chose to only send one position update every 2 seconds to very close users and one position update every 15 seconds to users very far away.

We also use Photons InterestGroup feature to split users into different regions, which allows us to determine who to send high and low frequency updates to.

Files of interest:

- ConferenceNetwork.cs
- AvatarManager.cs
- PlayerBase.cs and PlayerLocal.cs (OnSerializeLowFrequency, OnSerializeHighFrequency)
- Helpers.GetHighFrequencyGroupId(position)
- ConferenceSceneSettings.cs (to setup how many high frequency interest groups are available)

XR for Oculus Quest

SpaceHub Conference is designed so it can be used in VR (using the Oculus Quest) or standalone PC and Mac. The View Mode Manager handles the different modes and loads the appropriate ones. The local player is also separated into a PC Rig and an XR Rig, which each handle their different inputs and interactions. For the XR mode we utilize Unitys XR Interaction Toolkit.

Files of interest:

- ViewModeManager.cs
- Assets/Prefabs/Local Player.prefab
- Unity XR Interaction Toolkit

Spacehub Groups

Spacehub Groups / Groups implements the basic Communication features used for Spacehub Conference. Users can join or be added to "Groups" with different features like text and voice chat. Conference features using Groups:

- ChatBubbles
- Voice Circles
- Global Floor Chat
- Emotes
- Signals (Moderation)

namespace Spacehub.Groups

GroupsManager

The GroupsManager is the common accessor for all Groups features and subclasses. It is instantiated on the local player and can be accessed via *PlayerLocal.Instance.GroupsManager* It collects the available connections as well as the Groups currently joined.

GroupsViewManager

The Groups View Manager is responsible for the UI side of Groups. It displays the Chatlog and Userlist of the selected group, lets you select different groups and show the mediastack.

Files of interest:

- GroupsViewManager.cs
- ChatChannelView.cs
- MediaControlViewStack.cs
- UserManagerView.cs
- UserNameList.cs
- ChatInput.cs
- ChatTopBar.cs

Connections

A Connection implements the actual network communication between clients. It wraps PhotonChat and PhotonVoice to be useable in Groups.

- GroupConnectionBase / IGroupConnection
- IVoiceGroupConnection
- IChatGroupConnection

Before using a connection it needs to be added to the GroupsManager via RegisterConnection(IGroupConnection);

To create a new Connection inherit from GroupConnectionBase and implement the interface for the connection type you are buildig (eg Chat or Voice) After a connection to a group has been established the Connection needs to tell the group that it was successful. (via GroupConnectionBase.OnGroupConnected(Group)) Only when all connections are established is the Group displayed in the UI.

Examples of implementations:

- PhotonChatGroupConnection.cs

- PhotonVoiceGroupConnection.cs
- PhotonChatConnector.cs // *This exists mostly to allow multiple IChatClientListener to listen to the same PhotonChat client*

Files of interest:

- GroupsDataStructures.cs

IGroupPropertiesStorage

Storage handles all properties that need to be saved and shared between users of groups. Eg Public names of Groups or Moderation stati of Users The default implementation uses PhotonChat but could be overwritten with anything that implements the IGroupPropertiesStorage interface.

Properties are adressed by two enums as Keys:

- GroupProperty
- UserProperty

Callbacks can be registered at different points:

- Group Property
  - Group -> PropertyChangedCallback
- UserProperty
  - UserManager -> PropertyChangedCallback
  - User -> PropertyChangedCallback

Files of interest:

- GroupsDataStructures.cs
- PhotonChatGroupStorage.cs

Group

A Group is a collection of users connected by one or more Connections (eg chat and/or voice)

Create New Group

```
    var connections = new Dictionary<Groups.ConnectionType, Groups.SendType>();
    connections.Add( Groups.ConnectionType.Chat, Groups.SendType.On );
    connections.Add( Groups.ConnectionType.Voice, Groups.SendType.On ); // add the connections you want
to use and the default sendtype for the joining user.

    var myGroup = m_GroupsManager.CreateOrGetGroup( new Groups.GroupCreationParameter()
    {
        Connections = connections,
        InstanceId = ConferenceRoomManager.Instance.CurrentRoom.InstanceId,
        InterestGroup = 0, // Unique(per room) Interestgroup. Used for Photon Voice
        Name = "UniqueGroupName",
        DefaultPublicName = "Name Visible To End-User",
        AlwaysVisibleInChannelList = true,
        ShowGroupInChatChannelList = true,
        CanRenameTitle = true, // Any User in a Group can rename the public name of the group
        CanBeClosedByUser = true, // Users can leave the group at will
        ChatMessagesFromHistoryCount = 0,
        HasPublicUserList = true,
        SortOrderInChannelList = 9999,
        SetSelectedOnConnect = true,
        StorageOverride = null,
    } );
```

Leave Group

```
    m_GroupsManager.LeaveGroup( myGroup );
    m_GroupsManager.LeaveGroup( "UniqueGroupName" );
```

Files of interest:

- Group.cs
- GroupsManager.cs
- GroupsDataStructures.cs

Custom Chat Messages

Custom chat messages can be used to send arbitrary data via Chat groups. You can register to a callback in Group.ReceivedChatMessageCallback to receive all chat messages and filter for Groups.GroupsChatMessageBase.GetMessageType() == ChatMessageType.Custom.

Examples of implementations:

- EmoteHandler.cs
- SignalManager.cs

Files of interest:

- GroupsChatMessages.cs
- GroupsManager.cs
- Group.cs

Users

Users are created and collected in the UserManager whenever a user connects through any connection. They are identified by a common UserId. The user Id is provided by IGroupUserIdProvider set in the GroupsManager. The default implementation can be found in the ConferenceChatListener.cs and uses the Photon Realtime UserId.

Examples of implementations:

- ConferenceChatListener.cs

Files of interest:

- UserManager.cs
- User.cs
- GroupsDataStructures.cs

Media Stack

Classes can implement the IMediaControllable to make Volume and Mute toggles available through the Media Stack UI To show up in the Media Stack IMediaControllable classes need to be added to the Media Manager PlayerLocal.Instance.GroupsManager.MediaManager.Add(this); PlayerLocal.Instance.GroupsManager.MediaManager.Remove(this);

Available Media Types: public enum MediaType { AudioIn, AudioOut, VideoIn, VideoOut, }

Examples of implementations:

- Group.cs
- VimeoVideoCanvas.cs

Files of interest:

- IMediaControllable.cs
- MediaManager.cs

Voice Circles

A very intuitive way to start voice chats with other users. Simply walk up to them and a voice circle will spawn around two or more users, who can now chat.

Files of interest:

- Assets/Scripts/VoiceCircles

Chatbubbles

A big blue dome in the scene which players can enter to start a voice and text chat.

Files of interest:

- Chatbubble.cs
- PlayerLocalChatbubble.cs

Sandbox

A Sandbox is using an InterestGroup to send object interaction data to players who also joined the same sandbox. As an example, we implemented the board game solitaire

Files of interest:

- Assets/Scripts/Sandbox
- Assets/Prefabs/Sandbox

Video playback with Vimeo

We implemented the Vimeo Unity SDK to stream video files from video into a video player. A Vimeo Pro Account is needed for this to work.

Files of interest:

- https://github.com/vimeo/vimeo-unity-sdk

Cinema

Our cinema scene demonstrates how to synchronize video playback between all users using Vimeo and the AVPro Video Plugin from the Unity Asset Store

Files of interest:

- Assets/Scenes/Cinema.unity
- SynchedVideo.cs

Stage

Stage demonstrates how you can use Photon Voice to potentially broadcast a few speakers to very many listeners. This can be used, for example, for talks, panels. etc. We also implemented a basic Slideshow with controls for the speaker, a Q&A system via chat groups and emote icon feedback buttons for the audience.

Files of interest:

- Assets/Scenes/StageRealtime.unity
- Assets/Scripts/Stage

Avatar customization

In the beginning of SpaceHub Conference, users can choose an outfit and look for themselves with is then synchronized to all visitors of the conference.

Files of interest:

- Assets/Scenes/CustomizationRoom.unity
- Assets/Scripts/Avatars/Customization

Avatar Art Packs

We wrote the avatar system in such a way that it is easy for you to extend and add your own customizations. Different avatar assets are separated by type into Art Packs and Art Categories (for example Hair, Face, UpperBody etc.)

Files of interest:

- Assets/Avatars/ArtPacks/Default

User authentication

In combination with the Web Application you can implement your own User Authentication so visitors can or have to register with a username and password. Also read the Custom Authentication section in Web Application Features of this document.

Files of interest:

- CustomizationMenuLogin.cs

Admin tools and user moderation

In order to retain full control over an event and it's visitors, we built several administration and moderation features. These features also rely heavily on the Web Application for User Management.

Files of interest:

- Assets/Scripts/Moderation

Minimaps

Each scene is displayed as a minimap in the main menu. This helps users find their way through the convention. A minimap for a scene is designed as a UI Prefab and then added to the ConferenceRoomManager in the Expo scene.

Files of interest:

- ConferenceRoomManager.cs
- Assets/Scripts/Minimap
- Assets/Prefabs/Minimaps

Spaces

Spaces demonstrate how you can stream assets (images, text etc.) from the Web Application into the Unity Application. You can assign a user to a Space so they can upload their own artwork or logo, which is then displayed in the conference world.

Files of interest:

- SpacesManager.cs
- Assets/Scripts/Spaces
- PhotonWebRpcController.php (Web Application, see the functions spaces() and spacesoverview())

Space Elevator

The Space elevator shows a way to use the same scene for multiple different spaces. Users from each floor are separate from each other while still loading the same Unity scene. Spaces can be assigned a floor ID so the same Space can be used for multiple exhibitors on different floors.

Files of interest:

- Assets/Prefabs/UI/Elevator Popup.prefab

Addressables

Using addressables, you can split your main content from the projects visuals and load them separately. This helps to reduce download times for WebGL builds or it enables you to be able to change visual content after your customers have downloaded the main application. It is very useful when hosting live events to be able to change you scene without having to re-compile the whole project.

This is an advanced feature from Unity which allows you to split your project into multiple bundles. You can enable it by included the Addressables package in your project through the package manager. You can read more about Unity Addressables in the official documentation:
https://docs.unity3d.com/Packages/com.unity.addressables@1.16/manual/index.html

There is also a fantastic community guide to help you get started: https://github.com/mikerochip/addressables-training-manual

The addressables content can be in the same Unity Project as your main code, or you can even split it into a completely separate Unity Project. This is very benificial if you want to swap your visuals for different events while keeping the main code as it already is. It reduces the initial download time, significantly improves your control over memory management and enables a scalable, multi-tenant architecture for your application.

Using this feature is entirely optional. Our implementation is supposed to serve as an example for developers who want to develop large scale conference solutions.

To use addressables you have to execute the following steps:

- Build Addressables once:
  - Open Window > Asset Management > Addressables > Groups
  - Select Play mode Script > Use existing build
  - Select Build > New Build > Default Build Script
- If you want to test the addessable demo scene:
  - Open your ConferenceServerSettings in the unity inspector
  - Make sure the "Default Addressable Catalogs" array is empty
  - Now a helper box appears for the addressables
  - Click "Add SpaceHub Demo Addressable Data" to fill out the demo data
  - Now you are able to enter the special "Island Booth" door, which loads an external scene via addressables

Making assets addessable

Once the addressable package is loaded, every asset will have an "Addressable" checkmark box at the very top in the inspector. Simply click this box and the asset can now be loaded and unloaded via an address. The address is a string that often represents a filepath, but can be anything that fits your project. Our example showcases this by making the scenes and the avatar art packs addressable.

In our implementation we follow an address naming guideline: The address of a scene should be: "Scenes/{SceneName}.unity" The address of an avatar art pack should be: "Artpack/{prefix}" (Use the same prefix you used in the art packs scriptable object, for example "default" for our SpaceHub avatars)

Building your addressables

Once you setup your addressables, you can build a so called "catalog", which stores the paths to all addressable object which you can load at runtime. If you want to be able to load your addressables from a remote server, you need to tell Unity to build a remote catalog. Find the "AddressableAssetSettings" asset in your project (or navigate to Window -> Asset Management ->

Addressables -> Settings) and enable "Build remote catalog".

Now you can use the addressable editor windows (Window -> Asset Management -> Addressables -> Groups) to build your addressable bundles. Please refer to the Unity documentation for this step as this process is to complex to fully describe in this Quickstart Guide.

Splitting Content from executable project

Using addressables you can split your content entirely from the executable and even author them in different Unity project. This way you could, for example, serve your visual scenes from your own file server or deliver them through a CDN. Unity introduced its own CDN service for this purpose: https://unity.com/products/cloud-content-delivery

To make this work it requires a bit of setup as described below.

Proxy scripts

Addressables cannot contain code, which isn't already present in the main project. So in order to be able to place MonoBehaviours in external scenes, we developed a system to split our MonoBehaviours into two files, one containing just the public properties, which you would want to setup in the Unity inspector, and one that contains all the logic. Using C#s partials feature, these two files then get merged into one class during compilation.

This allows us to only copy the partials with public properties into the external project to setup the public properties.

As a general convention we gave those two script files specific names, so it is easier to work with them. For example the Chatbubble script is split into Chatbubble.cs and Chatbubble_Behaviour.cs. Both contain the class definition `` `public partial class Chatbubble : VoiceArea` `` so the compiler knows to combine the two.

If you want to add you own code that can be used in external projects, please refer to the "Writing your own Partial classes" section of this document.

Prepare the content project

Create a new, empty Unity project which stores your content (visuals, audio, etc.).

Required Packages:

- Addressables (1.8.5)
- Cinemachine
- TextMeshPro (2.1.3)
- Universal RP (7.3.1)

Next, you'll need to add the "Scripting Define Symbols" *SPACEHUB_CONTENT_PROJECT* to your project settings.

- ProjectSettings -> Player -> Other Settings -> Scripting Define Symbols -> Add SPACEHUB_CONTENT_PROJECT

In order to use scripts and prefabs you'll need to copy a folder over to the new project.

- Copy "Assets\SpaceHubCore"

These folders contain the basic prefabs and scripts used in Spacehub scenes. If you have more you can also copy these over or even create new ones in the content project without problem. Adding code to the content project does not work as easily though. See "Writing your own Partial classes" for more info.

We use this to define stub classes for the content project so its not neccessary to import certain packages there. eg XRBaseInteractable Look at "ContentProjectStubs.cs"

When using Unity Cloud Delivery, setup Addressable Profile

Go to addressable profiles window and create a new profile called UCD

```
RemoteBuildPath: ServerData/[BuildTarget]
RemoteLoadPath: {SpaceHub.Conference.AddressablesManager.RemoteLoadPath}
```

Writing your own Partial classes

If you want to create your addressable bundles within a different Unity content project than the Spacehub executable project you'll need to split up some of your classes using the keyword partial. Partial classes allow you to seperate the public properties and the actual implementation of functionallity. This is neccessary because it is not possible to load code through addressables, but scripts with the same name will still be matched between the addressable from the content project and the executable project. We only copy the "header" script with the public properties into the content project so we can set the properties in the inspector without needing to have all the dependencies and script references in the content project.

Note: This is only neccessary if you want to split your content into a different unity project. You can also use addressables without this and will not have to worry about splitting your scripts.

Example: File: MyScript.cs public partial class MyScript : MonoBehaviour { public string myProperty = "Default"; }

File: MyScript_Behaviour.cs public partial class MyScript: MonoBehaviour { private void Awake() { Debug.Log(myProperty); } //More code here. }

Note: The file with the public properties has the same name as the class. This is because this is the one that actually gets attached as a component by unity.

Web Application Features

Introduction

There are four different ways how SpaceHub Conference (the Unity App) interacts with SpaceHub World (the website):

1. Custom Authentication

Through a special setup of your Application in the Photon dashboard, the Unity application can send authentication data (username/password) to the Photon server, which in turn will connect to your webserver to ask if the credentials are valid. We use it to authenticate users, who register an account on https://spacehub.world. This way they can use the same username and password on the website and in the Unity application. This process is described in the Photon Documentation - Custom Authentication.

Files of interest:

- ConferenceNetwork.cs (Unity App)
- PhotonAuthenticationController.php (Web App)

2. WebHooks

This is an advanced Photon feature which requires you to setup different paths in your Photon dashboard of your Realtime application so that the Photon Server can communicate with your webserver. This process is described in the Photon Documentation - WebHooks.

This is used for tracking where Players are in your world and logging this on the webserver.

Files of interest:

- PhotonWebhooksController.php (Web App)

3. WebRPCs

A WebRPC is an advanced Photon feature which works similarly to WebHooks. You need to setup the proper path in your Photon dashboard in order for WebRPCs to work. This process is described in the Photon Documentation - WebRPCs.

WebRPCs send a request from the Unity App to the Photon server, which then connects to your REST Server to receive a response and deliver it to the Unity application.

Files of interest:

- SpacesManager.cs (Unity App)
- ModUserList.cs (Unity App)
- PhotonWebRpcController.php (Web App)

4. REST Api Calls

This is not a Photon Feature. We are simply calling a web url from the Unity Application via UnityWebRequest and listening for the response. We designed the Web server in such a way, that it responds with JSON data, which we then parse and handle. But this is an architecture choice and you can return whatever data you want.

Files of interest:

- Networking.UnityWebRequest (Unity Documentation)
- EventsPanel.cs (Unity App)
- ApiController.php (Web App)

List of features

- User registration and login
- Photon Custom User Authentication
- Player profiles and banning
- Player logging with WebHooks
- Admin client and user management with WebRPCs
- Event management with REST API calls
- Exhibitor management and streaming assets

User registration and login

We are using basic Laravel authentication features for User accounts

Files of interest:

- app/Http/Controllers/Auth

Photon Custom User Authentication

Learn about this Photon feature here: Photon Documentation - Custom Authentication

Files of interest:

- app/Http/Controllers/PhotonAuthenticationController.php

Player profiles and banning

Player profiles can be filled out online with Company data, Social Media links etc. This data is then loaded by each user during the Custom Authentication process and then shared with others via player properties.

Files of interest:

- app/Http/Controllers/ProfileController.php
- app/Http/Controllers/PhotonAuthenticationController.php

Player logging with WebHooks

Learn about this Photon feature here: Photon Documentation - WebHooks

Files of interest:

- app/Http/Controllers/PhotonWebhooksController.php
- app/Player.php

Admin client and user management with WebRPCs

Learn about this Photon feature here: Photon Documentation - WebRPCs

Files of interest:

- app/Http/Controllers/PhotonWebRpcController.php
- app/Http/Controllers/UsersController.php
- app/User.php

Event management with REST API calls

Before the user has logged into Photon, we can't use WebRPCs to receive data from the serve. We use REST API calls to receive the event list from the server in the beginning of the Funnel.

Files of interest:

- app/Http/Controllers/ApiController.php

Exhibitor management (*Spaces*) and streaming assets

Users can be assigned a Space in the online backend and then upload graphical assets and setup some text, which is then streamed into SpaceHub Conference via WebRPCs and UnityWebRequests

- app/Http/Controllers/PhotonWebRpcController.php (spaces() and spacesoverview() functions)
- SpacesManager.cs (in the Unity Application)