GVAX SA DEVELOPMENT

uCAST

for Android, iOS

Unity plugin for fast and flexible Google Cast® implementation in your App

Version 1.0

Released 8th of May 2020

The plugin allows you to display different files and streams on the cast receiver devices.

This plugin works on Android and iOS devices and does not work in the Editor or on desktop platforms! The plugin has been developed with Unity 2018.4.20f1. The core functionality should work on the previous versions, but you will not be able to check out the demo scene.

Setup

You will need to create the **Cast application** (https://cast.google.com/publish/) and obtain an **ID** in order to use the Cast functionality. You should provide the necessary information for the application in order for cast to work (package name for Android, and iTunes ID and bundle ID for iOS). You do not have to provide the information for the platform you do not plan to use your application on.

While the application is not yet published in the cast console, you will have to manually add the Serial Numbers for the Cast devices you plan to test on. Therefore, you should publish the Cast application (in the Cast console) as soon as possible for the Unity application to be able to detect and interact with all Cast devices.

Note! It can take up to several days after the App ID creation in the Cast console before the applications using it will be able to detect the Cast devices. It will also take several days after the Cast application publishing before the Unity application will be able to detect all of the

nearby Cast devices, not only the ones that are manually added in the Cast console.

After the initial setup is complete, you can move on to the Cast functionality integration into your Unity application. Make sure you provide the same package name in the Unity project that is registered in the Cast console.

Integration

CastManager class is the main entry point for the Cast integration into your Unity project.

Note! You should add the Access Wifi Information capability in XCode project after building with Unity in order for the Cast functionality to work correctly!

For Android you will need to enter the correct package name in the Cast/Android/AndroidManifest.xml file.

CastManager class is the main entry point for the Cast integration into your Unity project. *CastManager* should be initialized before you try to interact with the Cast functionality!

```
private void Initialize()
{
    CastManager.Initialize(CastApplicationId, OnCastConnected, OnCastDisconnected);
}

...

private void OnCastConnected(CastSession session)
{
    //Handle current CastSession here
}

private void OnCastDisconnected(string message)
{
    //Handle message here
}
```

CastManager is initialized with the application ID obtained from the Cast console after registering the application. At this point, you also provide two callbacks to handle a successful connection and disconnection.

After a successful connection, you will be able to work with the

CastSession object to extract the necessary information about the connection, as well as interact directly with the connected device.

Call the *CastManager.ShowCastController* method when you want to show the native view, containing the cast button. You can provide a short title, as well as an optional image to display on the native view:

```
CastManager.ShowCastController("Cast button will be shown once a device is discovered", imageLoader.ImageTexture);
```

The native view can be dismissed by user (native back button on Android and custom back button on the view for iOS). The native view will be automatically closed after a successful connection.

You can also manually finish the current CastSession with an option to either keep or stop casting on the current device:

```
CastManager.EndCurrentSession(true);
```

CastSession is the main interaction point while the application remains connected with the Cast device. You obtain a reference to it after a successful connection to a Cast device. You can read device attributes by obtaining a *CastDevice* object from the session:

```
CastDevice device = session.GetDevice();
```

You can also obtain a reference to the Remote media client object that is responsible for the media playback on the Cast device:

```
CastRemoteMediaClient mediaClient = session.GetRemoteMediaClient();
```

You can also check the active input state, device standby state, as well as read and change the volume level and mute state.

RemoteMediaClient object is responsible for the media playback on the connected Cast device.

To play a media file on the Cast device, you will have to load the *RemoteMediaClient*, providing the necessary data. You can also optionally provide whether to autoplay the media after it is loaded, as well as provide a time interval from which to play the data (for videos and audio files) and playback speed (from 0.5 to 2).

```
var mediaData = new CastMediaData
{
        title = _selectedVideoInfo.title,
        subtitle = _selectedVideoInfo.subtitle,
        dataUrl = _selectedVideoInfo.videoUrl,
        previewImageUrl = _selectedVideoInfo.imageUrl,
        mimeType = "videos/mp4",
        mediaType = MediaType.Movie,
        streamType = StreamType.Buffered
};
```

```
var mediaClient = _currentCastSession.GetRemoteMediaClient();
mediaClient.LoadMedia(mediaData);
```

You can also call *Play*, *Pause* and *Stop* methods to influence the playback of the media.

> **Note!** After *Stop* is called, you will have to *LoadMedia* again before playing it.

You can also modify the playback rate at any time using the *SetPlaybackRate* method and providing a value between 0.5 and 2.