

NEXPLAYER
Passion for High Quality Video Services

NexPlayer™ Plugin for Unity
Version 1.0.5.33
Technical Reference Manual
Updated June 3rd, 2019

Index

1.Introduction	4
1.1 Abstract	4
2. Capabilities and Limitations	5
2.1 System Requirements	5
2.2 Summary of Protocols	5
2.3 Android and iOS Feature List	6
2.4 Windows and Unity Editor Feature List	10
3. Player Integration	13
3.1 Installing the Package	13
3.2 Package Content	13
3.3 Sample Scenes	22
3.4 API Graphic Integration	23
3.5 Android Build Configuration	23
3.6 iOS Build Configuration	24
4. Quickstart Guide	25
5. Advanced Integration	26
5.1 Creating the Player	27
5.2 Releasing the Player	32
5.3 Background Status Handling	34
5.4 Subtitles Integration	35
5.5 Additional Properties	40
5.6 Debugging Log Level	41
5.7 Playback Modes	42
5.8 DRM	46
5.9 HTTP Headers	48
5.10 Log Level Property	49
5.11 Low Latency	50
5.12 Buffering Time	52
5.13 Support ABR	53
5.14 Render Mode	54
5.15 Add / Remove Renderer	57
5.16 Change Render Mode	58
5.17 Change Aspect Ratio	59
5.18 Change the Video Texture Background	61

5.19 360 Video Settings	62
5.20 Video Transparency	67
5.21 Spread Video Across Objects	68
5.22 Timed Metadata in HLS	69
5.23 Add a Callback to get an AES External Key	71
5.24 Get Content Information and Content Statistics Information	72
6. FAQ	74
7. Detailed API	77
8. Technical Support	78
9. Legal Notes	79
9.1 Disclaimer for Intellectual Property	79
9.2 Copyright	79

1.Introduction

1.1 Abstract

NexPlayer™ plugin for Unity is an interactive multiscreen streaming player for Unity apps that support both standard and 360 video playbacks across all Android, iOS and PC devices. NexPlayer™ plugin for Unity is the only playback solution for Unity that supports HLS & DASH live streaming across all devices, advanced events and out of the box 360 video navigation.

NexPlayer™ plugin for Unity was developed in collaboration with the native NexPlayer™ SDK and it has access to all of its features, such as Intelligent ABR, HTTP live streaming(HLS), DASH, progressive download, local playback, customizable APIs, and more.

The plugin was built to be swift and reliable without sacrificing performance and it has proven compatibility with international standards.

This Integration Guide is updated periodically.
Additional details and sample code will be added to this document.

For testing and development purposes, the Android/iOS emulator should not be used with NexPlayer™ as there are known differences and issues between the emulator and the actual devices. For example, the OpenGL renderer does not run properly within the emulators environment.

Frequent testing on actual devices is strongly recommended during development and all apps should be tested on actual devices prior to release.

2. Capabilities and Limitations

2.1 System Requirements

NexPlayer™ Plugin for Unity supports:

- Unity 2018.3 Personal Edition or Unity Pro and above
- Android 4.40 (Kit Kat, API level 19) and above using OpenGL ES version 2.0 or 3.0
- iOS 9.0 and above using OpenGL ES version 2.0 and Metal (iOS 11.0 and above to use VR features).

2.2 Summary of Protocols

Platform	Supported Graphics APIs	HLS	DASH	PD	Local	Streaming Assets
Android (armeabi-v7a and x86)	OpenGL ES 2, OpenGL ES3	✓	✓	✓	✓	✓
iOS	OpenGL ES2, OpenGL ES3, Metal	✓	✓	✓	✓	✓
Windows	DirectX 11	✓	✓	✓	✓	✓

2.3 Android and iOS Feature List

Basic Functionalities

- Start (from the beginning or anywhere)
- Pause/Resume
- External Subtitle On/Off
- Seek
- Scale
- Audio On/Off
- Audio Volume Adjustment
- Loop Playback
- Stop
- Portrait/Landscape Screen
- Buffering

Advanced Features

- Adaptive Bitrate (ABR)
- Low Latency
- Widevine DRM
- Widevine Optional Headers
- 360 Video
- IPv6
- Set Video Output Position
- Adjust Audio Output Volume
- Video Rotation
- Screen Scaling

Subtitles

- SMI
- SRT
- WebVTT
- CEA-608 (HLS only)
- CEA-708 (HLS only)

Protocols

- **HTTP Live Streaming (HLS)**
 - Support HLS V6, HLS V7
 - VOD
 - Live
 - Fragmented MP4 Chunk
 - CMAF
 - I-Frame Only Track (HLS V4)
 - Multi-Video/Audio Track (HLS V4)
 - Offline Playback
 - Offline Store Without Playback
 - HTTP Redirection
 - HTTP Additional Headers
 - GZip
 - Proxy
 - Low Latency
 - Live and VOD in Unicast ABR

- **DASH (isobmff live/on demand profile only)**

- VOD
- Live
- Offline Playback
- Offline Store Without Playback
- HTTP Redirection
- HTTP Digest Authentication
- HTTP Additional Headers
- GZip
- Low Latency
- Proxy

- **Progressive Download**

- **Local Playback (Supported File Formats)**

- MP4/PIFF/3GPP/3GP2
- ASF/WMV
- AVI
- AAC/AAC+/eAAC+
- MPEG-TS
- MKV

Codecs

- **Video Codecs**

- H.264 (Android 4.4 and above, iOS 8.0 and above)
- HEVC/H.265 (Android 5.0 and above, iOS 11.0 and above)
- MPEG-4

- **Audio Codecs**

- AAC-LC
- HE-AAC
- HE-AAC v2

2.4 Windows and Unity Editor Feature List

Basic Functionalities

- Start (from the beginning or anywhere)
- Pause/Resume
- External Subtitle On/Off
- Seek
- Scale
- Audio On/Off
- Audio Volume Adjustment
- Loop Playback
- Stop
- Portrait/Landscape Screen
- Buffering

Advanced Features

- Adaptive Bitrate (ABR)
- Low Latency
- 360 Video
- IPv6
- Set Video Output Position
- Adjust Audio Output Volume
- Video Rotation
- Screen Scaling

Subtitles

- WebVTT
- SMI
- SRT
- CEA-608 (HLS only)
- CEA-708 (HLS only)

Protocols

- **HTTP Live Streaming (HLS)**
 - Support HLS V6, HLS V7
 - VOD
 - Fragmented MP4 Chunk
 - I-Frame Only Track (HLS V4)
- **DASH (isobmff live/on demand profile only)**
 - VOD
- **Progressive Download**
- **Local Playback (Supported File Formats)**
 - MP4/PIFF/3GPP/3GP2
 - ASF/WMV
 - AVI
 - AAC/AAC+/eAAC+
 - MPEG-TS
 - MKV

Codecs

- **Video Codecs**

- H.264 (Android 4.4 and above & iOS 8.0 and above)
- MPEG-4

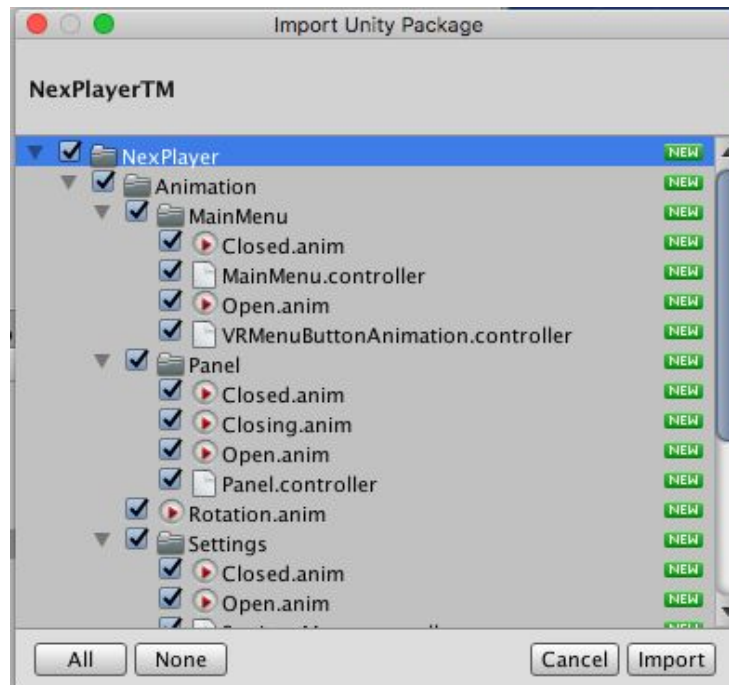
- **Audio Codecs**

- AAC-LC

3. Player Integration

3.1 Installing the Package

The fully operational NexPlayer™ Plugin for Unity SDK is provided as a Unity Package and can be imported into your Unity project.



3.2 Package Content

All the necessary files to integrate the player with any Unity Project are included in this package. The files are distributed in the following directories:

- | Integration_Guide.pdf
- | License.txt
- | NexPlayer_Plugin_for_Unity_Reference_Manual.zip
- | NexPlayer_Plugin_for_Unity_Release_Notes.txt
- |

```

+---Animation
| | Rotation.anim
| | SF Button.controller
| |
| +---MainMenu
| |   Closed.anim
| |   MainMenu.controller
| |   Open.anim
| |   VRMenuButtonAnimation.controller
| |
| +---Panel
| |   Closed.anim
| |   Closing.anim
| |   Open.anim
| |   Panel.controller
| |
| \---Settings
|   Closed.anim
|   Open.anim
|   SettingsMenu.controller
|
+---Materials
|   MaterialLeft.mat
|   MaterialMono.mat
|   MaterialOver.mat
|   MaterialRight.mat
|   MaterialUnder.mat
|   Nexplayer Unity.mat
|   RenderTexture 1.mat
|   RenderTexture 2.mat
|   RenderTexture 3.mat
|   RenderTexture.mat
|   Transparent.mat
|   WorldSpaceTilling.mat
|

```

```

+---NexPlayer360
| +---Resources
| | | cardboard.png
| | | font-awesome_4-7-0_compass_396_50_ffffff_none.png
| | | font-awesome_4-7-0_hand-paper-o_256_0_ffffff_none.png
| | | font-awesome_4-7-0_hand-rock-o_256_0_ffffff_none.png
| | | gray_dot.png
| | | SpherePoints.FBX
| | |
| | \---Materials
| |         No Name.mat
| |
| +---Scenes
| |     NexPlayer360.unity
| |
| \---Scripts
|     Nex360DLL.dll
|     NexPlayer360.cs
|     NexPlayer360KeyControls.cs
|     NexVRInteractable.cs
|     NexVRInteractableSeekBar.cs
|
+---Prefabs
|     CaptionUI.prefab
|     HeaderValuePanel.prefab
|     Main Camera.prefab
|     PlaybackSetting.prefab
|     Sample Controls.prefab
|     SF Button.prefab
|     SF Scene Elements.prefab
|     SF Title.prefab
|     StereoCameras.prefab
|
|
|

```

+---Resources

- | audio-mute.png
- | audio.png
- | font-awesome_4-7-0_arrow-left_396_50_ffffff_none.png
- | font-awesome_4-7-0_link_32_28_ffffff_none.png
- | font-awesome_4-7-0_pause_256_100_d9d5d4_none.png
- | font-awesome_4-7-0_play_256_100_d9d5d4_none.png
- | icons8-closed-captioning-64.png
- | icons8-expand-50.png
- | icons8-height-filled-50.png
- | icons8-original-size-filled-50.png
- | icons8-resize-50.png
- | icons8-stretch-uniform-50.png
- | icons8-width-50.png
- | icon_pause.png
- | icon_play.png
- | icon_stop.png
- | menu.png
- | more-options.png
- | Nexplayer Unity.png
- | no-repeat.png
- | RenderTexture.renderTexture
- | repeat.png
- | Roboto-Bold.ttf
- | Roboto-Regular.ttf
- | seekbar_background.png
- | skip-end-filled.png
- | skip-start-filled.png
- | stars.jpg
- | UIButtonDefault.png
- | unchecked-checkbox.png
- |
- |
- |
- |


```

+---Scenes
|   MainMenu.unity
|   NexPlayer_ChangeRenderMode_Sample.unity
|   NexPlayer_MaterialOverride_Sample.unity
|   NexPlayer_PlaybackMultipleRenderer_Sample.unity
|   NexPlayer_PlaybackSetting_Sample.unity
|   NexPlayer_RawImage_Sample.unity
|   NexPlayer_RenderTexture_Sample.unity
|   NexPlayer_VideoSpreadRenderTexture_Sample.unity
|
+---Scripts
| | Activation.cs
| | NexEditorHelper.cs
| | NexPlayer.cs
| |
| +---3rdParty
| |   AlticastWVDRM.cs
| |
| +---Editor
| | | AssetCatalog.cs
| | | iOS Build.cs
| | | JsonParser.cs
| | | NexCustomEditor.cs
| | | NexPlayerEditor.cs
| | | PBXCapabilityType.cs
| | | PBXPath.cs
| | | PBXProject.cs
| | | PBXProjectData.cs
| | | PBXProjectExtensions.cs
| | | PlistParser.cs
| | | ProjectCapabilityManager.cs
| | |
| |
| |
| |
| |

```

```

| | +---PBX
| | |     Elements.cs
| | |     Lexer.cs
| | |     Objects.cs
| | |     Parser.cs
| | |     Sections.cs
| | |     Serializer.cs
| | |     Utils.cs
| | |
| | \---Properties
| |     AssemblyInfo.cs
| |
| +---SDK
| |     NexPlayerAndroid.cs
| |     NexPlayerBase.cs
| |     NexPlayerCommon.cs
| |     NexPlayerFactory.cs
| |     NexPlayeriOS.cs
| |     NexPlayerPrefsHelper.cs
| |     NexPlayerTypes.cs
| |     NexPlayerWindowsEditor.cs
| |     NexSubtitleParser.cs
| |
| +---UI
| |     AdditionalValueManager.cs
| |     DRMHeaderPanelManager.cs
| |     Main.cs
| |     NexMainCube.cs
| |     NexSeekBar.cs
| |     NexUIController.cs
| |     NexVideoObject.cs
| |     NexVRUIController.cs
| |     PanelManager.cs
| |     PlaybackSettings.cs
| |     StereoMode.cs

```

```

| | TiltWindow.cs
| |
| \---Utility
|     NexUtil.cs
|     StreamingAssetFileHelper.cs
|     VideoCanvasHelper.cs
|
+---Shaders
|     InsideVisible-3D-LEFT.shader
|     InsideVisible-3D-OVER.shader
|     InsideVisible-3D-RIGHT.shader
|     InsideVisible-3D-UNDER.shader
|     InsideVisible.shader
|     TransparencyShader.shader
|     WorldSpaceTiling.shader
|
\---VRMenu
    +---Models
    | | MenuInfo.fbx
    | | MenuItem360.fbx
    | | MenuItemHLS.fbx
    | | MenuSelector.fbx
    | |
    | \---Materials
    |     MenuElements.mat
    |     MenuLogo.mat
    |
    +---Prefabs
    |     MenuItem360.prefab
    |     MenuItemHLS.prefab
    |     MenuSelectorChild.prefab
    |
    +---Resources
    | | bkg1_back.png
    | | bkg1_bot.png

```

```

| | bkg1_front.png
| | bkg1_left.png
| | bkg1_right.png
| | bkg1_top.png
| | GUIReticle.tif
| |
| +---360 Preview
| |   360_1.png
| |   360_2.png
| |   360_3.png
| |   360_Text.PNG
| |
| +---HLS Preview
| |   HLS_1.png
| |   HLS_2.png
| |   HLS_3.png
| |   HLS_Text.PNG
| |
| \---Materials
|   360_Preview_Material.mat
|   360_Text_Material.mat
|   GUIOverlay.mat
|   GUIReticle.tif
|   GUIDTargetReticle.tif
|   HLS_Preview_Material.mat
|   HLS_Text_Material.mat
|   MenuGUISlider.mat
|   Nexplayer Unity.mat
|   SeparableAlpha.shader
|   SimpleClear.shader
|   SlidingUV.shader
|   Stars_Material.mat
|   SunShaftsComposite.shader
|   UIOverlay.shader
|   VRCameraFade.mat

```

|
\\---Scripts
MenuAnimator.cs
MenuButton.cs
MenuItemPopout.cs
MenuSelectorMover.cs
NexMainCube.cs
PostEffectsBase.cs
Reticle.cs
SelectionRadial.cs
SelectionSlider.cs
SunShafts.cs
UIFader.cs
VRCameraFade.cs
VRCameraUI.cs
VREyeRaycaster.cs
VRInput.cs
VRInteractivItem.cs

3.3 Sample Scenes

Sample scenes are available in the Unity package in the following folder:

`./Assets/NexPlayer/Scenes/` in the file `NexPlayer.unity`

Scene	Description
MainMenu.unity	The main menu to access to all the scenes
NexPlayer_RawImage_Sampler.unity	Video player using a Raw Image component to display the video through a static image
NexPlayer_MaterialOverride_Sample.unity	Video player using a Material Override component to display the video through a rotating cube
NexPlayer_RenderTexture_Sample.unity	Video player using a Render Texture component to display the video through a static cube
NexPlayer_VideoSpreadRenderTexture_Sample.unity	Video player using a Render Texture component to display the video across multiple objects
NexPlayer360.unity	Video player using a Material Override component in a sphere to display the video in 360

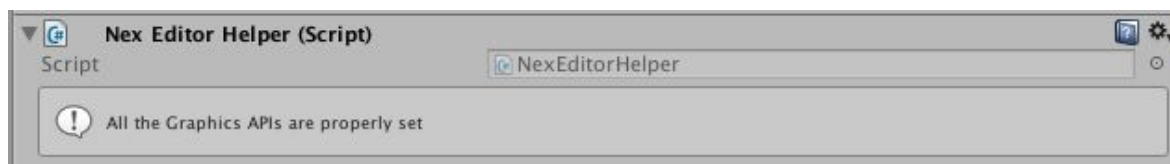
3.4 API Graphic Integration

To integrate NexPlayer™, a compatible graphics API must be selected.

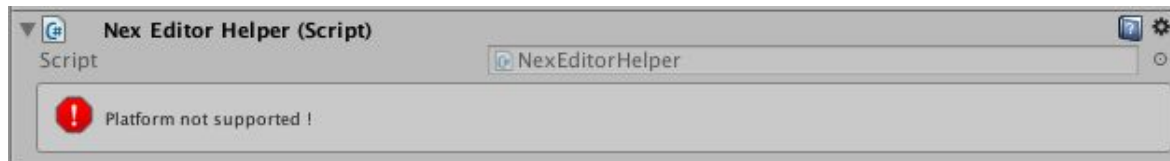
This can be done manually in the following Unity section:

File → Build Settings... → Player Settings

To make this process easier, we've created a helper component called **"NexEditorHelper.cs"**. When attached to an object, this component will include a graphic UI to automatically detect and inform you of any conflicts regarding the graphics API. This component will also promptly solve it.



Message that appears when the graphics API is correct



Message that appears when the graphics API is incorrect

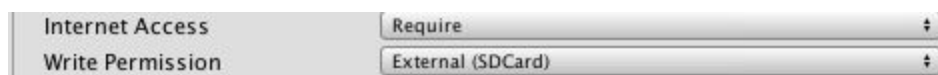
3.5 Android Build Configuration

To create a new APK file that includes the NexPlayer™ Plugin for Unity, the default configurations must be changed.

In order to allow any remote videos on Android, the option **Internet Access** needs to be set to 'Require' in the Unity player settings and the option **Write Permission** should be set to 'External (SD Card)'. This configuration is needed to save DRM certification data in the Android SDCard.

These configurations can be set up in the following Unity section:

File → Build Settings → Player Settings (Android) → Other Settings



3.6 iOS Build Configuration

To create a new IPA file that includes the NexPlayer™ Plugin for Unity, the default configurations must be changed.

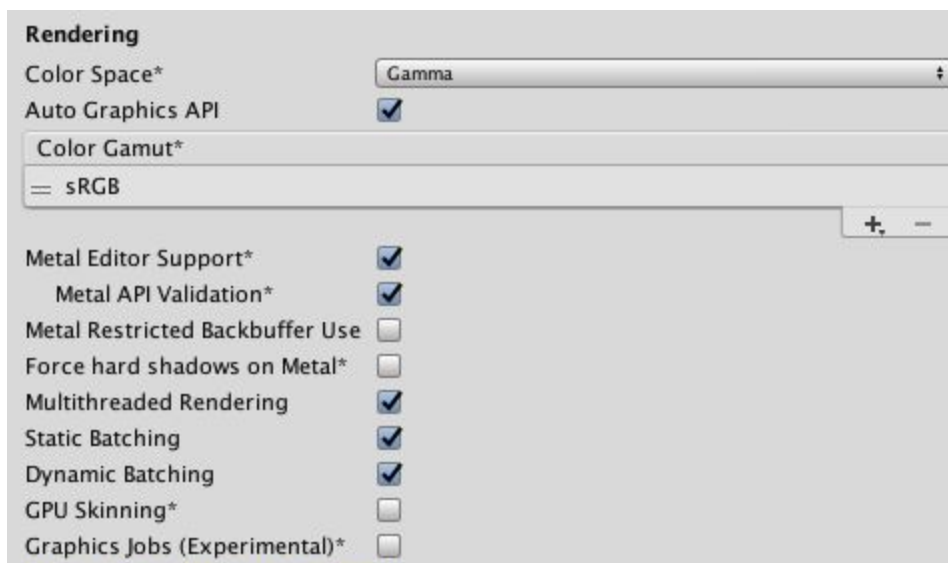
To display HTTP videos on iOS, the option **Allow downloads over HTTP** needs to be enabled.

A fast and easy way to enable these settings is using the helper component **NexEditorHelper.cs**.

We highly recommend using the following configuration, especially enabling the **Auto Graphics API** option.

This configuration can be set up in the following Unity section:

File → Build Settings → Player Settings (iOS) → Other Settings



4. Quickstart Guide

The NexPlayer™ Plugin for Unity can be quickly integrated into a Unity project using the files included in the Unity package.

After importing the Unity package into your Unity project, you should perform the following steps:

1. Create a new Raw Image or a Cube (This is where the video will be rendered).
2. Create a new empty GameObject and rename it NexPlayer
3. Add the component NexPlayer(Script) to the NexPlayer GameObject.
4. In the component select the Players Additional Properties (Autoplay, Loop...).
5. Select a PlayMode (StreamingPlay, AssetPlay, LocalPlay).
6. Fill in the Input Values (Keyserver URL, LicenseRequestTimeout...).
7. (Optional) Create an empty GameObject and rename it ToggleableUI.
8. (Optional) Add the prefab SampleControls as a child of the ToggleableUI GameObject.
9. (Optional) Add the corresponding elements to the children of the SampleControls GameObject.(See Quickstart Tutorial Video)
10. Assign the Rendering GameObjects. The minimum that the player needs are the Camera and the Render Object (Renderer, RawImage, RenderTexture).
11. (Optional) To display a video in a RenderTexture object, remember to add a material to the object that contains the same texture as you referenced in the player. Standard materials can be found in Assets → NexPlayer → Materials.
12. If necessary, add the name of your scene to the switch structure located in the Awake() method in the NexPlayer.cs script. If your scene has the same name as the scenes provided in the SDK, it will work without changing the code.

You can find a visual tutorial of how to integrate the player here:

[Quickstart Tutorial Video](#)

5. Advanced Integration

We allow you to integrate the NexPlayer™ Plugin for Unity in your own code. In the following sections you will learn how to implement different functionalities into your codes, from the player creation to the implementation of DRM.

If you need more information about the different functions used in the NexPlayer™ Plugin for Unity, our Unity Package includes a detailed Reference Manual:

NexPlayer → NexPlayer_Plugin_for_Unity_Reference_Manual → Index.html

5.1 Creating the Player

You must include the following variables and methods to create the player:

```
//Variables to Create the Player
private NexPlayerBase player;

private VideoCanvasHelper videoCanvasHelper;

[SerializeField]
public Renderer renderer;

[SerializeField]
public RenderTexture renderTexture;

[SerializeField]
public Renderer renderTextureRenderer;

[SerializeField]
public RawImage rawImage;

[SerializeField]
public Camera mainCamera;

[SerializeField]
public int playType = 0;

[Tooltip("URI used to play the video")]
[SerializeField]
public string URL = "";

[SerializeField]
public int logLevel = -1;

public static NexRenderMode m_RenderMode = NexRenderMode.None;

private NexPlayerError initResult = 0;
```

The awake method runs the application in the background, creates a canvas helper, selects a render mode, and sets up NexPlayer:

```
//Awake Method
```

```
private void Awake() {
    Application.runInBackground = true;

    videoCanvasHelper = new VideoCanvasHelper(rawImage, renderer, renderTexture,
    renderTextureRenderer, mainCamera);

    if (m_RenderMode == NexRenderMode.None) {
        //Select a Render Mode (In NexPlayer.cs is selected by the Scene Name)

        m_RenderMode = NexRenderMode.RenderTexture;
        m_RenderMode = NexRenderMode.MaterialOverride;
        m_RenderMode = NexRenderMode.RawImage;
    }

    if (videoCanvasHelper != null)
        videoCanvasHelper.setRenderMode(m_RenderMode);

    Screen.sleepTimeout = SleepTimeout.NeverSleep;
    SetupNexPlayer();
}
```

The update method of NexPlayer needs to be called at the Update callback of the MonoBehaviour object:

```
//Update Method.
```

```
private void Update()
{
    if (player != null)
    {
        player.Update();
    }
}
```

The SetupNexPlayer() method creates the player. First it takes the player from NexPlayerFactory, then it sets a color for the texture that appears while the video is loading, it also sets the Render Mode, and assigns a Target Material for the player. Lastly it sets the main properties, and if there are no errors it opens the player.

//Setup Player Method

```
public void SetupNexPlayer() {  
  
    try {  
        // Creation of the NexPlayer instance  
        if (player == null)  
            player = NexPlayerFactory.GetNexPlayer();  
  
        //Default renderMode is MaterialOverride  
        NexRenderMode renderMode = videoCanvasHelper.GetRenderMode();  
  
        player.renderMode = renderMode;  
  
        player.SetTextureColor(Color.black);  
  
        switch (renderMode) {  
            case NexRenderMode.MaterialOverride:  
            case NexRenderMode.VR:  
                player.targetMaterialRenderer = renderer;  
                player.targetMaterialRenderer.material.mainTexture = player.GetTexture();  
                break;  
            case NexRenderMode.RenderTexture:  
                player.targetTexture = renderTexture;  
                Graphics.Blit(player.GetTexture(), player.targetTexture);  
                break;  
            case NexRenderMode.RawImage:  
                player.targetRawImage = rawImage;  
                player.targetRawImage.texture = player.GetTexture();  
                break;  
        }  
  
        SetProperties(); // For more information check the point Additional Properties  
  
        // Initialize NexPlayer  
  
        initResult = player.Init(logLevel);  
    }  
}
```

```

    if (initResult == NexPlayerError.NEXPLAYER_ERROR_NONE) {
        OpenPlayer();
    }
} catch (System.Exception e) {
    Debug.LogError("Error while initializing the player. Please check that your platform is supported.");
}
}
}

```

The OpenPlayer() method retrieves the full path of the file that is going to be played and opens the content of the file.

```

//Open Player Method

private int OpenPlayer() {
    int ret = 0;

    // The coroutine needs to be started after the player is created an initialized
    // Test routine
    // add complete path to content as play type, since current url includes only the name

    if (initResult == NexPlayerError.NEXPLAYER_ERROR_NONE) {
        string url = NexUtil.GetFullUri(playType, URL);
        string subtitle = null;
        OpenContent(url, subtitle);

    } else {
        player = null;
    }

    return ret;
}

```

The OpenContent() method opens the content of a url depending on the play type that is set. If the play type is 1, then the content must be located in the Streaming Assets folder. Finally, it starts the coroutine of the player.

```
//Open Content Method

private void OpenContent(String url, String subtitle_url) {
    if (!String.IsNullOrEmpty(url)) {
        if (playType == 1)
            player.OpenWithFD(url, subtitle_url);
        else
            player.OpenWithPath(url, subtitle_url);
    }

    StartCoroutine(player.CoroutineEndOfTheFrame());
}
```

5.2 Releasing the Player

Releasing the player refers to closing the player in a controlled way to exit the current scene.

To release the player, you have to call the `Release()` method and wait for the `NEXPLAYER_EVENT_CLOSED` callback:

```
//ToggleQuit Method
```

```
public void ToggleQuit()
{
    if (this.gameObject.activeSelf == false)
    {
        return;
    }
    FinishGame();
}
```

```
//Finish Game Method
```

```
private void FinishGame()
{
    if (player != null)
    {
        if (Application.platform == RuntimePlatform.WindowsEditor)
        {
            player.Close();
            player.Release();
            player = null;
            GoBack();
        }
        else
        {
            GoBack();
        }
    }
    else
    {
        GoBack();
    }
}
```


//EventNotify Method

```
private void EventNotify(NexPlayerEvent paramEvent, int param1, int param2)
```

```
{
```

```
    ...
```

```
    switch (paramEvent)
    {
```

```
        ...
```

```
        case NexPlayerEvent.NEXPLAYER_EVENT_CLOSED:
```

```
        {
```

```
            ResetPlayerUI();
```

```
        }
```

```
        break;
```

```
    }
```

```
}
```

//GoBack Method

```
private void GoBack() {
```

```
    UnityEngine.SceneManagement.SceneManager.LoadScene("MainMenu");
```

```
}
```

5.3 Background Status Handling

The NexPlayer™ Plugin for Unity supports to check the state change (background/foreground) via the OnApplicationPause() function's parameter value. If the application state is running in the background, it calls the Pause function, pausing the current video. When the application state comes to the foreground, it calls the Resume function, playing the video again.

```
//Variables to Handle the Background Status
```

```
private NexPlayerBase player;
private NexPlayerStatus playerStatus = NexPlayerStatus.NEXPLAYER_STATUS_NONE;

private void OnApplicationPause(bool pauseStatus) {
    if (player != null) {
        if (pauseStatus) {
            playerStatus = player.GetPlayerStatus();
            bApplicationPaused = true;

            if (player.GetPlayerStatus() > NexPlayerStatus.NEXPLAYER_STATUS_STOP) {
                player.Pause();
            }
        }
    }
}

//Return to Foreground
else {
    if (bApplicationPaused) {
        if (player.GetPlayerStatus() > NexPlayerStatus.NEXPLAYER_STATUS_STOP &&
            playerStatus == NexPlayerStatus.NEXPLAYER_STATUS_PLAY) {
            player.Resume();
        }
        playerStatus = NexPlayerStatus.NEXPLAYER_STATUS_NONE;
        bApplicationPaused = false;
    }
}
```

5.4 Subtitles Integration

The NexPlayer™ Plugin for Unity supports local and external subtitles.

```
//Variables to Support Subtitles
```

```
[SerializeField]  
    public string streamingSubtitleUri =  
    "http://webs.nexstreaming.com/hls/i/subtitle/sample.smi";
```

```
[SerializeField]  
    public string assetSubtitleUri = "";
```

```
[SerializeField]  
    public string localSubtitleUri = "";
```

```
[SerializeField]  
    public int subtitleFileIndex = 0;
```

```
[Tooltip("Subtitle URL is used to loading subtitle")]
```

```
[SerializeField]  
    public string subtitleURL = null;
```

```
public static string sharedSubtitleURL = null;
```

```
public static NexSubtitleElement subtitleElement;
```

```
public string savedCaption = null;
```

```
public int savedStartCaptionTime = 0;
```

```
public int savedEndCaptionTime = 0;
```

```
private bool bOnSubtitle = true;
```

```
[SerializeField]  
    public Text caption;
```

```
[SerializeField]  
    Toggle captionToggle;
```

In `OpenContent()` you have to include this parameter to add the subtitle URL.

```
// At OpenContent()
private void OpenContent(String url, String subtitle_url) {
    if (!String.IsNullOrEmpty(url)) {
        if (playType == 1)
            player.OpenWithFD(url, subtitle_url);
        else
            player.OpenWithPath(url, subtitle_url);
    }

    StartCoroutine(player.CoroutineEndOfTheFrame());
}
```

In `OpenPlayer()` you have to include the subtitles full URL, open the content and return the content.

```
//Open Player Method

private int OpenPlayer() {
    string url = NexUtil.GetFullUri(playType, URL);
    string subtitle = null;
    if (subtitleURL != null && !String.IsNullOrEmpty(subtitleURL) &&
        !playerPrefs.lowLatencyEnable)
        subtitle = NexUtil.GetFullUri(playType, subtitleURL);
    ret = OpenContent(url, subtitle);

    if (ret == 0)
    {
        string playerStatusText = "Opening";
        SetPlayerStatus(playerStatusText);
    }

    return ret;
}
```

To manage the subtitles you have to wait for the following callbacks:

`NexPlayerEvent.NEXPLAYER_EVENT_TEXT_INIT`

`NexPlayerEvent.NEXPLAYER_EVENT_TEXT_RENDER`

`NexPlayerEvent.NEXPLAYER_EVENT_PLAYBACK_STARTED`

//EventNotify Method

```
private void EventNotify(NexPlayerEvent paramEvent, int param1, int param2)
{
    ...
    switch (paramEvent)
    {
        ...

        case NexPlayerEvent.NEXPLAYER_EVENT_PLAYBACK_STARTED:
        {
            if (playPauseImage != null)
                playPauseImage.sprite = pauseSprite;

            startToPlayEvent.Invoke();

            if (captionToggle != null)
            {
                player.bOnSubtitle = captionToggle.isOn;
                bOnSubtitle = captionToggle.isOn;
                player.OnOffSubtitle(captionToggle.isOn);
            }
        }
        break;

        case NexPlayerEvent.NEXPLAYER_EVENT_TEXT_INIT:
        {
        }
        break;
        case NexPlayerEvent.NEXPLAYER_EVENT_TEXT_RENDER:
        {
            if (player != null && caption != null)
            {
                subtitleElement = player.GetCurrentSubtitleElement();
                if (subtitleElement.caption != null)
                {
                    caption.text = subtitleElement.caption;
                    savedCaption = subtitleElement.caption;
                    savedStartCaptionTime = subtitleElement.startTime;
                    savedEndCaptionTime = subtitleElement.endTime;
                }
            }
        }
        break;
    }
}
```

Finally, you can set up the ToggleCaption() method to manage the subtitles from the interface.

```
public void ToggleCaption(bool bToggle)
{
    if (this.gameObject.activeSelf == false)
        return;

    if (Application.platform == RuntimePlatform.IPhonePlayer || Application.platform ==
RuntimePlatform.Android || Application.platform == RuntimePlatform.WindowsEditor ||
Application.platform == RuntimePlatform.WindowsPlayer)
    {
        if (player != null)
        {
            player.OnOffSubtitle(bToggle);
            player.bOnSubtitle = bToggle;

            if (bToggle == false)
            {
                if (caption != null)
                {
                    caption.text = "";
                }
            }
        }
    }
    else
    {
        if (captionToggle != null)
            captionToggle.isOn = player.bOnSubtitle;
    }

    if(bToggle)
    {
        if(savedCaption != null)
        {
            int currentTime = player.GetCurrentTime();
            if(savedStartCaptionTime <= currentTime && savedEndCaptionTime >=
currentTime)
            {
                caption.text = savedCaption;
            }
        }
        captionImage.GetComponent<Image>().color = Color.white;
    }
}
```

```
    }  
    else  
    {  
        captionImage.GetComponent<Image>().color = Color.gray;  
    }  
}
```

5.5 Additional Properties

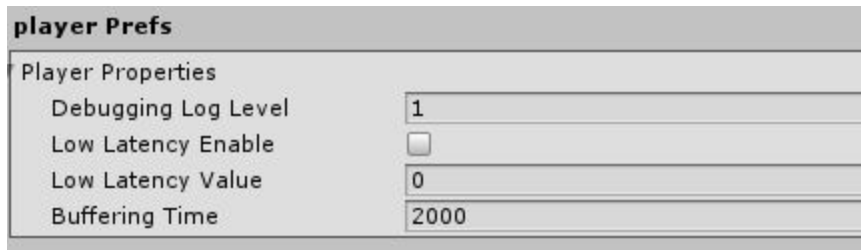


Properties	Description
AutoPlay	Starts the playback automatically after the video is buffered
Loop	When the playback reaches the end position it jumps to the start and plays again
MutePlay	Turns off the audio
Thumbnail	NexPlayer waits for playback in the first frame of the video
Volume	Adjust the NexPlayer audio volume
DRM Cache	Enables caching the DRM license, but the server side must support it
LogLevel	Debug log level at NexProtocol, NexPlayer Engine, and Unity

```
private void SetProperties()
{
    if (player != null) {
        player.mute = mutePlay;
        player.volume = volumeSize;
        player.loop = loopPlay;
        player.autoPlayback = autoPlay;
        player.firstFrame = thumbnail;
        player.bOnSubtitle = bOnSubtitle;
    }
}
```


5.6 Debugging Log Level

Debug log level to get information about NexPlayer and 3rd party libraries (e.g. Widevine).



The screenshot shows a dialog box titled "player Prefs". Under the "Player Properties" section, there are four settings: "Debugging Log Level" set to 1, "Low Latency Enable" with an unchecked checkbox, "Low Latency Value" set to 0, and "Buffering Time" set to 2000.

player Prefs	
Player Properties	
Debugging Log Level	1
Low Latency Enable	<input type="checkbox"/>
Low Latency Value	0
Buffering Time	2000

Properties	Description
Debugging Log Level	Set log level for debugging (protocol, Nex AL Libraries). This value must be an integer between 1 and 4.

```
At SetPlayerPrefs(NexPlayerPrefsHelper.NexPlayerPrefs playerPrefs) in OpenPlayer()
```

```
if (player != null && playerPrefs != null)
{
    //Set log level for debugging
    if(playerPrefs.lowLatencyEnable)
    {
        player.logLevelForDebugging = playerPrefs.debuggingLogLevel;
    }
}
```

5.7 Playback Modes

The NexPlayer™ Plugin for Unity supports three playback modes:

- 1) Streaming Play: plays the streaming content

The screenshot shows a configuration window for the NexPlayer plugin. It includes the following fields and controls:

- URL :** A text field containing the URL `https://webs.nexstreaming.com/hls/i/bipbop_16x9/bipbop_16x9_variant.m3u`.
- KeyServer URL**: An empty text field.
- LicenseRequestTimeout(sec)**: A text field containing the value `30`.
- Additional Http Headers**: A section containing:
 - Number of Additional Headers:** A control with a value of `0`, plus and minus buttons, and a **Reset** button.
 - Http Header Keys**: A table with one row showing **Size** as `0`.
 - Http Header Values**: A table with one row showing **Size** as `0`.

URL	The URL of the streaming content
KeyServer URL	The URL of the key server
License Request TimeOut	The timeout value until a request is dismissed
Additional Http Headers	Set additional HTTP headers for HLS or Smooth Streaming, etc.
Subtitle URL	The URL of the external subtitles

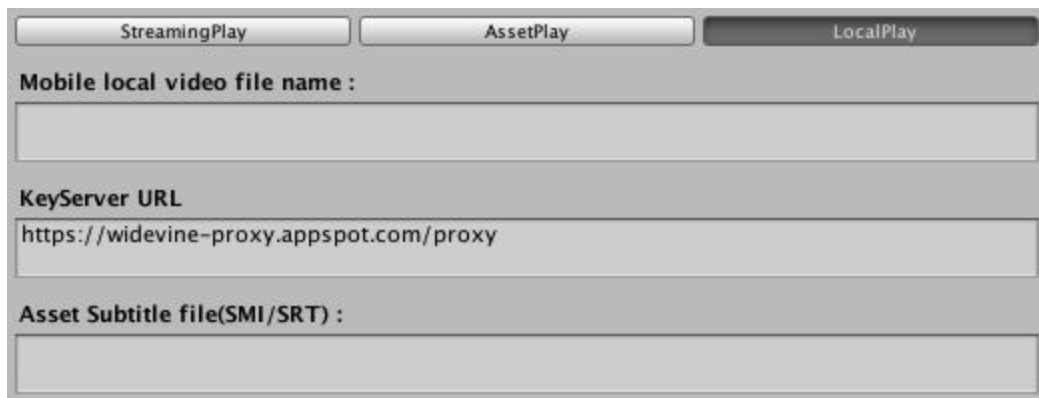
2) AssetPlay: plays the video file in the Streaming folder



Streaming Assets Directory

Asset video file	Selects the file to play among the mp4 files in the streaming folder (if there are no files in the folder, a warning message will be displayed)
KeyServer URL	The URL of the Key Server
Subtitle URL	Selects the file to display subtitles among the SRT and SMI files in the streaming folder (if there are no files in the folder, a warning message will be displayed)

3) Local Play: plays a video file downloaded on a mobile phone



Mobile local video file name	The name of the video file
KeyServer URL	The URL of the key server
Subtitle file	The name of the subtitle file







Android Example




SDCARD/Android/data/Bundle Identifier/files/FileName.mp4

- The Bundle Identifier is different for each app.
- If the file exists in the above path based on the file name entered in the Inspector, it can be played.
- If you put a subtitle file in the same path as the video file, then the subtitles will be displayed on the screen

iOS Example

- After building in Unity, turn on the application and enable support for iTunes file sharing in the info.plist file in the generated Xcode project.
- Using iTunes, put the file in the file name you entered in the Inspector
- If you put a subtitle file in the same path as the video file, then the subtitles will be displayed on the screen

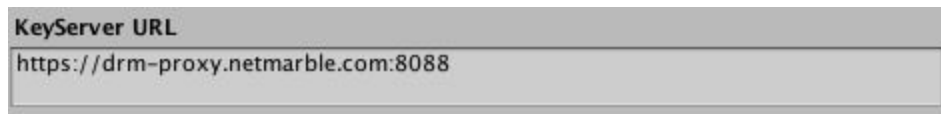
	iMovie
	Mipmapping
	NexPlayer Plugin
	NexPlayerSample2_devel
	NexPlayerSample2_Widevine_rele...
	NVRencode iOS

	Keystore	0KB	오늘 오후 7:18
	sampleVideo.mp4	17.3MB	2018. 6. 27. 오후 3:...
	Unity	20KB	오늘 오후 7:18

5.8 DRM

The NexPlayer™ Plugin for Unity supports DRM content in Streaming, Asset Play, and local videos. The DRM can be accessed with a KeyServer URL and Widevine custom headers. This feature is only supported on Android and iOS.

KeyServer URL

A screenshot of a Unity Inspector field for 'KeyServer URL'. The field is a text box containing the URL 'https://drm-proxy.netmarble.com:8088'.

KeyServer URL
https://drm-proxy.netmarble.com:8088

Widevine Custom Headers

A screenshot of a Unity Inspector control for 'Widevine Optional DRM Headers'. It includes a 'Number of Optional Headers' field set to 0, with '+' and '-' buttons, and a 'Reset' button. Below is a table with columns 'Header Key' and 'Header value'.

Widevine Optional DRM Headers	
Number of Optional Headers:	0 + - Reset
Header Key	Header value

Variables to use DRM

```
[SerializeField]
public string keyServerURI;

[SerializeField]
public string[] widevineHeaderKeys;

[SerializeField]
public string[] widevineHeaderValues;

[SerializeField]
public bool drmCache = true;

[SerializeField]
public int licenseRequestTimeout;

private Dictionary<string, string> drmHeaderData = new Dictionary<string, string>();

private AlticastWVDRM alticastWVDrm;
```

```

Awake() {

//make drm header

if (widevineHeaderKeys != null && widevineHeaderValues != null) {
    for (int i = 0; i < wwHeaderSize; i++) {
        if (!String.IsNullOrEmpty(widevineHeaderKeys[i]) &&
            !String.IsNullOrEmpty(widevineHeaderValues[i]))
            drmHeaderData.Add(widevineHeaderKeys[i], widevineHeaderValues[i]);
    }
}

OpenPlayer() {

    // drm setting & player properties setting

    if (Application.platform == RuntimePlatform.Android || Application.platform ==
        RuntimePlatform.IPhonePlayer) {
        //Setting DRM
        alticastWVDrm = gameObject.GetComponent < AlticastWVDRM > ();

        if (alticastWVDrm != null) {
            //Initialize the dictionary

            drmHeaderData = null;

            drmHeaderData = alticastWVDrm.GenerateRequestHeaders();

            player.SetDRMSecretKey(alticastWVDrm.secretKey);
        }

        player.UseLicenseCacheMode(drmCache);
        player.SetWVDRMConfig(keyServerURI, drmHeaderData, licenseRequestTimeout);
    }

    .....

    if (initResult == NexPlayerError.NEXPLAYER_ERROR_NONE)
    {
        StartPlay(URL, subtitleURL);
    }
}

```

5.9 HTTP Headers

The NexPlayer™ Plugin for Unity supports additional HTTP headers that allow the client and the server to pass additional information with the request or the response.



Variables to use HTTP Headers

```
[SerializeField]
public string[] httpHeaderKeys;

[SerializeField]
public string[] httpHeaderValues;

[SerializeField]
public int httpHeaderSize = 0;

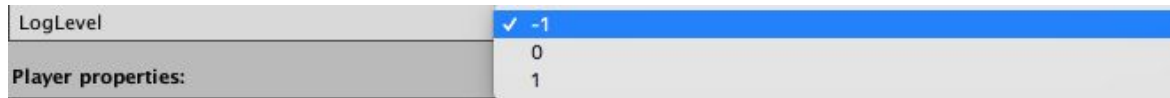
[SerializeField]
public int wvHeaderSize = 0;

private Dictionary<string, string> additionalHttpHeaders = new Dictionary<string,
string>();

Awake() {
if (httpHeaderKeys != null && httpHeaderValues != null)
{
    Log("key length : " + httpHeaderKeys.Length);
    for(int i = 0; i < httpHeaderKeys.Length;i++)
    {
        if(!String.IsNullOrEmpty(httpHeaderKeys[i]) &&
            !String.IsNullOrEmpty(httpHeaderValues[i]))
        {
            additionalHttpHeaders.Add(httpHeaderKeys[i], httpHeaderValues[i]);
        }
    }
}
```


5.10 Log Level Property

Debug log level about NexProtocol, NexPlayer Engine, Unity, and C#.



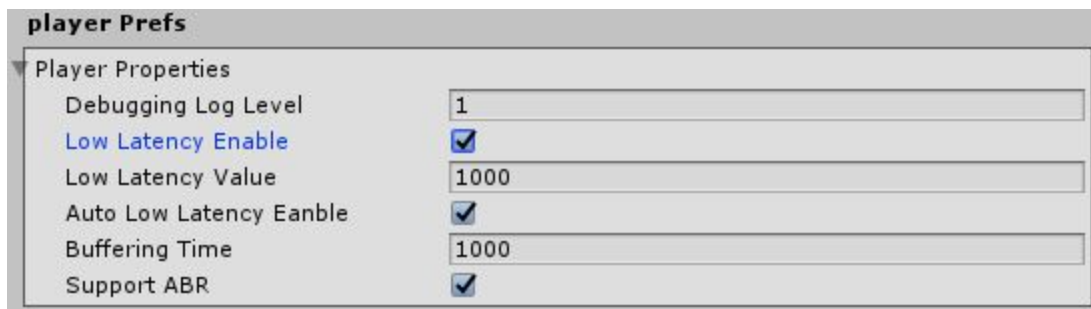
Log Level	Description
None	Do not use logs
0	Check only logs at C# level
Debug	View NexPlayer, NexAIFactory logs
RTP	Also, View RTP Logs (includes Debug Logs)
RCP	Also, View RCP Logs(includes RTP and Debug Logs)
Frame	Also, View Frame Logs(includes RTP and Debug Logs)

```
private void SetupNexPlayer() {  
    try  
    {  
        .....  
  
        player.Init(logLevel);  
    }  
}
```

5.11 Low Latency

The NexPlayer™ Plugin for Unity supports low latency video streaming. Low latency is a feature that displays a video stream with the lowest delay possible.

Properties	Description
Low Latency Enable	Enable/disable low latency
Low Latency Value	Set the number of milliseconds of media to buffer if additional buffering is required during streaming playback for low latency
Auto Low Latency Enable	Adjust automatically the number of milliseconds of media to buffer if additional buffering is required during streaming playback for low latency



// Variables to use Low Latency

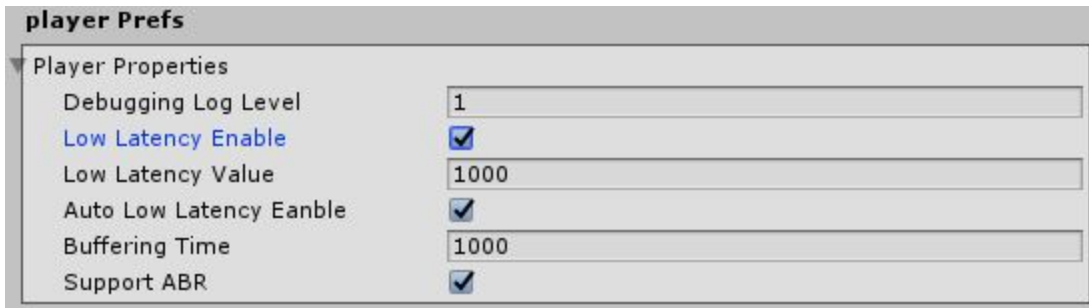
```
[SerializeField]
public NexPlayerPrefsHelper.NexPlayerPrefs playerPrefs = new
NexPlayerPrefsHelper.NexPlayerPrefs();

private void SetPlayerPrefs(NexPlayerPrefsHelper.NexPlayerPrefs playerPrefs)
{
    //Set LowLatency
    if (playerPrefs.lowLatencyEnable) {
        player.SetLowLatency(playerPrefs.lowLatencyEnable, playerPrefs.lowLatencyValue);
        //set LOW_LATENCY_AUTO_CONTROL property
        if (playerPrefs.AutoLowLatencyEnable)
            player.SetPlayerProperty(NexPlayerProperty.AUTO_LOW_LATENCY_ENABLE, 1);
    }
    //set LOW_LATENCY_AUTO_CONTROL property
```

```
if(playerPrefs.AutoLowLatencyEanble)
    player.SetPlayerProperty(NexPlayerProperty.AUTO_LOW_LATENCY_ENABLE, 1);
}
```

5.12 Buffering Time

The NexPlayer™ Plugin for Unity supports changing the rebuffering time.

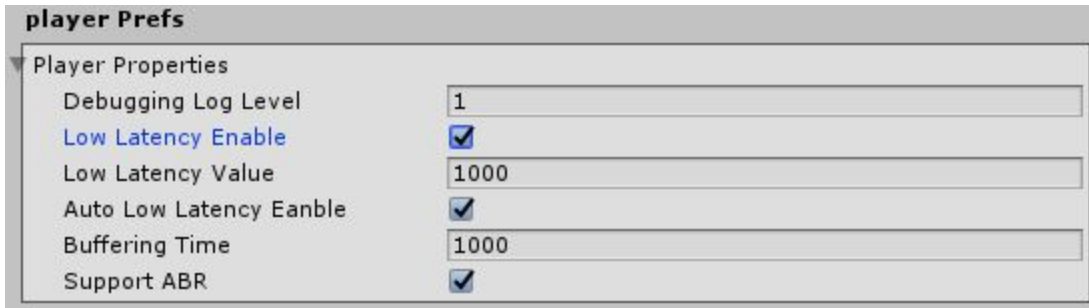


Properties	Description
Buffering Time	Set the average number of milliseconds to buffer if additional buffering is required during streaming playback (if this value is set to 0, it will be changed to 5000 by default)

```
//At SetPlayerPrefs(NexPlayerPrefsHelper.NexPlayerPrefs playerPrefs) in OpenPlayer()
if (player != null && playerPrefs != null)
{
    //Set buffering time
    if(playerPrefs.bufferingTime > 0)
    {
        player.SetPlayerProperty(NexPlayerProperty.INITIAL_BUFFERING_DURATION,
            playerPrefs.bufferingTime);
        player.SetPlayerProperty(NexPlayerProperty.RE_BUFFERING_DURATION,
            playerPrefs.bufferingTime);
    }
}
```

5.13 Support ABR

Adaptive bitrate is a feature that allows the player to dynamically switch between different resolution tracks on the video depending on the available bandwidth.



The screenshot shows a 'player Prefs' dialog box with a 'Player Properties' section. The settings are as follows:

Property	Value
Debugging Log Level	1
Low Latency Enable	<input checked="" type="checkbox"/>
Low Latency Value	1000
Auto Low Latency Enable	<input checked="" type="checkbox"/>
Buffering Time	1000
Support ABR	<input checked="" type="checkbox"/>

Properties	Description
Support ABR	Adaptive bitrate chooses the track with the highest resolution based on the available bandwidth (any track with a bandwidth greater than this value will not be played)

```
//At SetPlayerPrefs(NexPlayerPrefsHelper.NexPlayerPrefs playerPrefs) in OpenPlayer()
if (player != null && playerPrefs != null)
{
    //Set ABR Enabled

    if (playerPrefs.supportABR) {
        player.SetPlayerProperty(NexPlayerProperty.SUPPORT_ABR, 1);
    } else {
        player.SetPlayerProperty(NexPlayerProperty.SUPPORT_ABR, 0);
    }
}
```

5.14 Render Mode

The NexPlayer™ Plugin for Unity supports four render modes to display the video in different objects.

RenderMode	Description
RawImage	Plays the video with Raw Image component
Material Override	Plays the video with Renderer Material texture
RenderTexture	Plays the video with Render Texture
VR	Plays the video with VR

RenderMode.RawImage

1. Set the NexPlayer renderMode to RenderMode.RawImage
2. Set target RawImage to play video

```
private void SetupNexPlayer() {  
    try  
    {  
        .....  
  
        player.renderMode = RenderMode.RawImage;  
        player.targetRawImage = rawImage;  
    }  
}
```

rawImage

RawVideo (RawImage)

Example of how to set renderMode for RawImage in the script and inspector

RenderMode.MaterialOverride

1. Set NexPlayer renderMode to RenderMode.MaterialOverride
2. Set target Material Renderer to play video

```
private void SetupNexPlayer() {  
    try  
    {  
        .....  
  
        player.renderMode = RenderMode.MaterialOverride;  
        player.targetMaterialRenderer = renderer;  
    }  
}
```

renderer Video Cube (Mesh Renderer)

Example of how to set renderMode for Material Override in the script and the Inspector

RenderMode.RenderTexture

1. Set NexPlayer renderMode to RenderMode.RenderTexture
2. Set targetTexture to play video

```
private void SetupNexPlayer() {  
    try  
    {  
        .....  
  
        player.renderMode = RenderMode.RenderTexture;  
        player.targetTexture = renderTexture;  
    }  
}
```

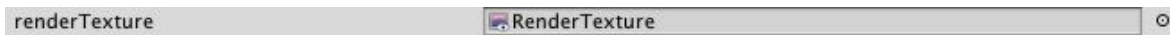
renderTexture RenderTexture

Example of how to set renderMode for renderTexture in the script and the Inspector

RenderMode.VR

1. Set NexPlayer renderMode to RenderMode.VR
2. Set target MaterialRenderer to play video

```
private void SetupNexPlayer() {  
    try  
    {  
        .....  
  
        player.renderMode = NexRenderMode.VR;  
        player.targetMaterialRenderer = renderer;  
    }  
}
```



Example of how to set renderMode for VR and renderTexture in the script and the Inspector

5.15 Add / Remove Renderer

The NexPlayer™ Plugin for Unity supports the addition and removal of renderers. This feature allows you to use the same renderer for different objects during runtime.

```
// Variables to Add / Remove Renderer
```

```
private int cubeIndex = 0;
```

```
private void AddRenderer() {  
    if(cube Index < 3 && cubeIndex >= 0)  
    {  
        cubeIndex++;  
        player.AddMaterialRenderer("Cube" + cubeIndex,  
        GameObject.FindGameObjectWithTag("Cube"+cubeIndex).GetComponent<Renderer  
        >());  
    }  
}
```

```
private void RemoveRenderer() {  
    if(cube Index < 3 && cubeIndex >= 0)  
    {  
        player.RemoveMaterialRenderer("Cube"+cubeIndex);  
        cubeIndex--;  
    }  
}
```

5.16 Change Render Mode

The NexPlayer™ Plugin for Unity supports changing the render mode during runtime.

```
public void ChangeRenderMode(int renderMode)
{
    switch (renderMode)
    {
        case 1:
            m_RenderMode = NexRenderMode.RenderTexture;
            player.targetTexture = renderTexture;
            break;

        case 2:
            m_RenderMode = NexRenderMode.MaterialOverride;
            player.targetMaterialRenderer = renderer;
            break;

        case 3:
            m_RenderMode = NexRenderMode.RawImage;
            player.targetRawImage = rawImage;
            break;

        default:
            break;
    }

    player.renderMode = m_RenderMode;
}
```

5.17 Change Aspect Ratio

The NexPlayer™ Plugin for Unity supports changing the aspect ratio of the screen during runtime.

Aspect Ratio	Description
OriginSize	Shows the video in its original size on the screen
FitToScreen	Shapes the size of the video to fit the screen proportionally
StretchToFullScreen	Extends the size of the video until it fills the entire screen
FitHorizontally	Extends the size of the video until it fills the x-axis of the screen
FitVertically	Extends the size of the video until it fills the y-axis of the screen

// Variables to Change the Aspect Ratio

```
private VideoAspectRatio ratio = VideoAspectRatio.FitToScreen;  
private Vector2 originalRendererSize = Vector2.zero;
```

```
public void ChangeAspectRatio(string aspectRatio)  
{  
    if (this.gameObject.activeSelf == false)  
        return;  
    switch (aspectRatio)  
    {  
        case "OriginSize":  
            ratio = VideoAspectRatio.OriginSize;  
            break;  
        case "FitToScreen":  
            ratio = VideoAspectRatio.FitToScreen;  
            break;  
        case "StretchToFullScreen":  
            ratio = VideoAspectRatio.StretchToFullScreen;  
            break;  
        case "FitHorizontally":  
            ratio = VideoAspectRatio.FitHorizontally;  
            Break;  
    }
```

```
        case "FitVertically":
            ratio = VideoAspectRatio.FitVertically;
            break;
    }

    if (originalRendererSize != Vector2.zero)
    {
        if (player != null)
        {
            if (m_RenderMode != NexRenderMode.VR)
            {
                player.SetPlayerOutputPosition(ratio, originalRendererSize);
            }
        }
    }
}
```

5.18 Change the Video Texture Background

The NexPlayer™ Plugin for Unity supports changing the background color of the texture where the video is shown while the video is opening.

This function must be called before the content is opened. By default this color is set to black.

```
// At SetupPlayer()

private void SetupNexPlayer()
{
    ...
    player.SetTextureColor(Color.black);
    ...
}
```

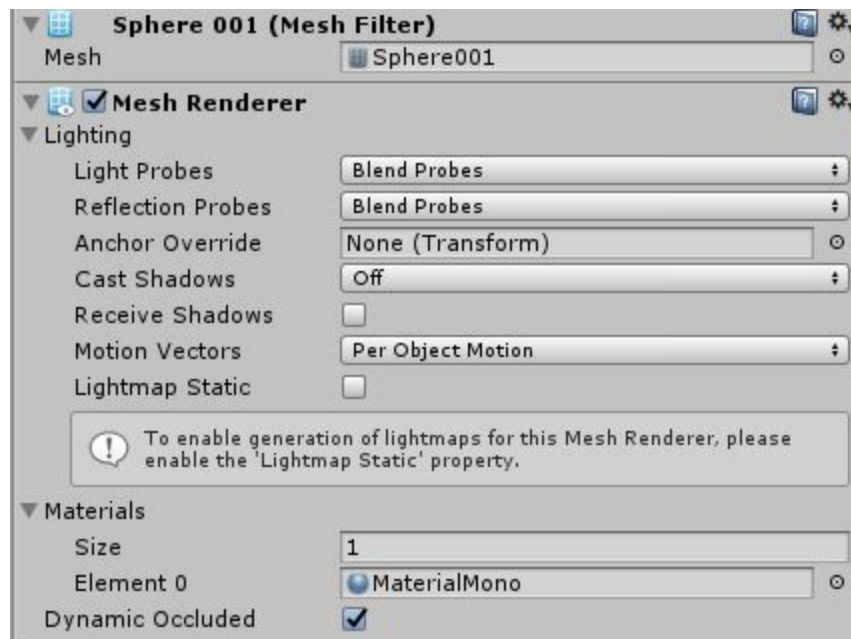
5.19 360 Video Settings

The NexPlayer™ Plugin for Unity supports 360 Video settings.

To set the 360 settings, you must create a sphere that will contain the player. You will need some components to set up the 360 video:

- **Mesh Renderer:** to set the material in which the video will be shown
- **NexPlayer Script:** controls the player settings
- **NexPlayer 360 Script:** controls the camera movement
- **NexPlayer 360 Key Controls Script:** controls the camera movement with a keyboard to test the 360 feature in the Unity Editor
- **NexUI Controller Script:** controls the interaction with the player using the 360 feature
- **Stereo Mode Script:** supports a stereo audio mode

Mesh Renderer Settings



NexPlayer Settings

Rendering GameObjects		
playPauseImage	PlayPause (Image)	○
totTime	TotTime (Text)	○
currentTime	CurrentTime (Text)	○
videoSize	VideoSize (Text)	○
status	Status (Text)	○
seekBar	NexSeekBar (NexSeekBar)	○
pauseSprite	font-awesome_4-7-0_pause_256_10i	○
playSprite	font-awesome_4-7-0_play_256_100_	○
caption	None (Text)	○
captionToggle	None (Toggle)	○
volume	None (Slider)	○
MuteButtonImage	None (Image)	○
volumeOnSprite	None (Sprite)	○
volumeOffSprite	None (Sprite)	○
loopButtonImage	None (Image)	○
loopOnSprite	None (Sprite)	○
loopOffSprite	None (Sprite)	○
renderer	Video Sample (Mesh Renderer)	○
rawImage	None (Raw Image)	○
renderTexture	None (Render Texture)	○
renderTextureRenderer	None (Renderer)	○
Camera	Main Camera (Camera)	○

NexPlayer 360 Settings

Nex Player 360 (Script)		
Script	NexPlayer360	○
Camera To Rotate	Main Camera (Camera)	○
Speed Mouse Movement	1.5	
Cursor Hand Grabbing	font-awesome_4-7-0_hand-rock-o_25i	○
Cursor Hand Hovering	font-awesome_4-7-0_hand-paper-o_2	○
Touch Speed	15	
Zoom Speed	0.2	
Rotation Speed	5	
Min Field Of View	40	
Max Field Of View	70	
Toggle UI	Video Sample (NexUIController)	○

Component	Description
Camera to Rotate	The main camera in the scene
Speed Mouse Movement	Mouse speed for PC
Cursor Hand Grabbing	Cursor icon that appears when the left mouse button is clicked or the screen is held/pressed
Cursor Hand Hovering	Default cursor icon
Touch Speed	Touch input speed
Zoom Speed	Zoom in/out speed
Min/Max Field of View	The range of the field of view
Toggle UI	Sets the UI

NexPlayer 360 Key Controls Settings



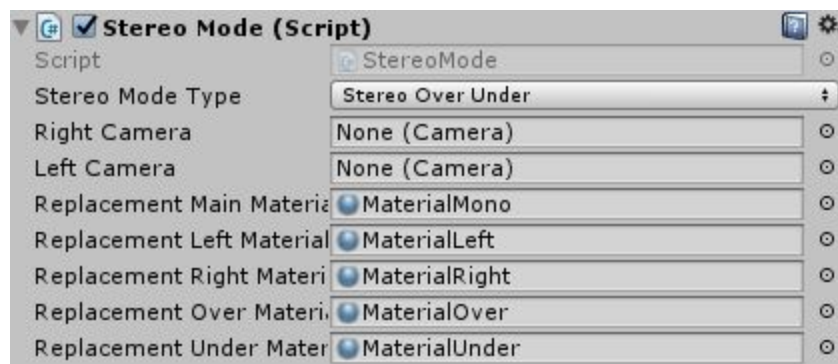
Component	Description
Camera to Rotate	The main camera in the scene
Right	Key used to represent a movement to the right
Left	Key used to represent a movement to the left
Up	Key used to represent a movement up
Down	Key used to represent a movement down

Nex UI Controller Settings



Component	Description
Game Object To Toggle	Sets the UI
Cardboard Button	VR Cardboard button (not yet supported)
Main Camera	The main camera in scene
Exclusion Layers	Layers containing non UI elements
Radial	Selection Radial to be toggled
Reticle	Reticle to be toggled

Stereo Mode Settings



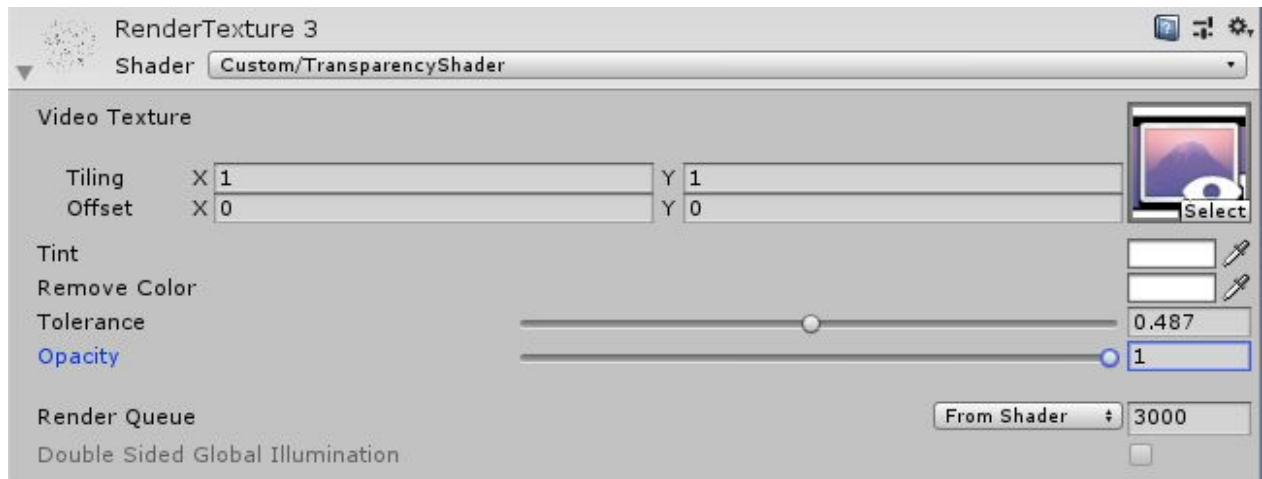
Component	Description
Stereo Mode Type	In order to correctly display the video on the sphere, this value must correspond to the 360 video format being used. The different values are: None, Mono, Stereo Over Under, Stereo Side by Side
Right Camera	This field is registered in app
Left Camera	This field is registered in app
Replacement Main Material	Renders material according to the field of view

5.20 Video Transparency

To implement this feature follow the steps below:

1. Select the game object where the Render Texture is referenced (e.g. a cube)
2. On the Render Texture component select the following shader:
Custom/TransparencyShader
3. Set the properties below to make the background of the video transparent

Properties	Description
Tint	All of the pixels will be multiplied to the tint color
Remove Color	The color in the background that will be removed from the video
Tolerance	This value determines how close in color the pixel must be in regards to the "Remove Color" property. It should be between 0 (less difference between the colors) and 1 (more difference between the colors)
Opacity	This value determines the opacity of the video. It should be between 0 (fully transparent) and 1 (fully opaque)

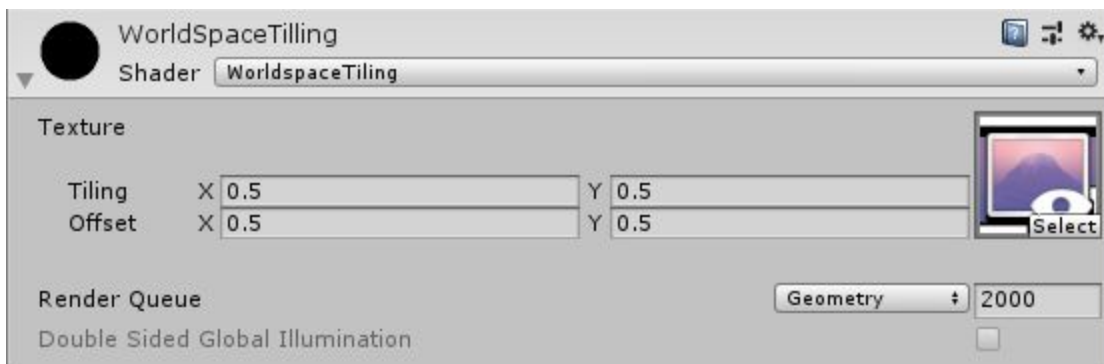


5.21 Spread Video Across Objects

To implement this feature follow the steps below:

1. Select the GameObject where the Render Texture is referenced (e.g. a cube)
2. On the Render Texture Component select the following shader: WorldSpaceTiling
3. Set the properties below to spread the video across a group of objects

Properties	Description
Tiling	This property modifies the size of the video
Offset	This property modifies the position of the video



5.22 Timed Metadata in HLS

If Timed Metadata is included in the content, then the player will get the information automatically.

Our default ID3 Tag set is shown in the table below. For more information, refer to `NexTimedMetadata` in `NexPlayerTypes.cs`.

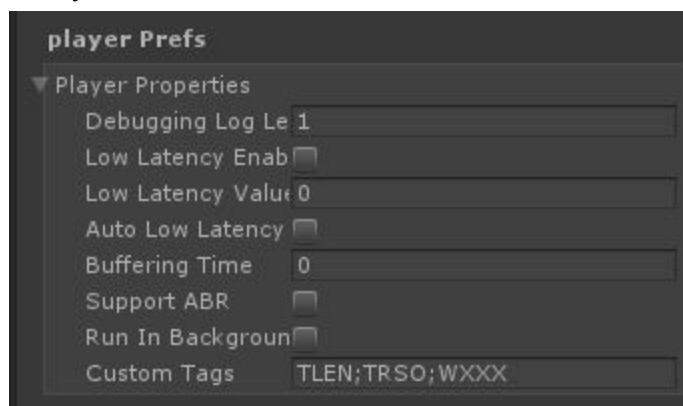
Name	ID3 Tag ID
Title	TIT2
Album	TALB
Artist	TPE1
Track	TRCK
Year	TDRC
Genre	TCON
PrivateFrame	PRIV
Text	TXXX

To add more Timed Metadata, you can set custom tags.

Custom tags must be set in the following formats:

Custom Tags : `{TAG1};{TAG2};{TAG3}`

For example, if you set Custom Tags (TLEN;TRSO;WXXX), then you can get TLEN, TRSO and WXXX if they are in the content.



In addition, the event occurs when the Timed Metadata needs to be rendered.

```
//EventNotify Method

private void EventNotify(NexPlayerEvent paramEvent, int param1, int param2)
{
    switch (paramEvent)
    {
        case NexPlayerEvent.NEXPLAYER_EVENT_TIMED_METADATA_RENDER:
        {
            NexTimedMetadata updateTimedMetadta =
player.GetTimedMetadata();
            Log("Title : " + updateTimedMetadta.Title);
            Log("Album : " + updateTimedMetadta.Album);
            Log("Artist : " + updateTimedMetadta.Artist);
            Log("Track : " + updateTimedMetadta.TrackNumber);
            Log("Year : " + updateTimedMetadta.Year);
            Log("Genre : " + updateTimedMetadta.Genre);
            Log("PrivateFrame : " + updateTimedMetadta.PrivateFrame);
            Log("TEXT : " + updateTimedMetadta.Text);

            foreach(KeyValuePair<string, string> extraTag in
updateTimedMetadta.ExtraTag)
            {
                Log("Extra Tag " + extraTag.Key + " : " + extraTag.Value);
            }
        }
        break;
    }
}
```

5.23 Add a Callback to get an AES External Key

Nexplayer™ provides a registered callback that adds a callback to retrieve the external key of an AES encrypted stream. The callback should be implemented using the following script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using AOT;
using System;
using NexPlayerAPI;

public class AESKeyCallback : MonoBehaviour
{
    public NexPlayer player;

    [MonoPInvokeCallback(typeof(NEXPLAYERGetKeyExtCallbackFunc))]
    public static int GetKeyExtCallback(string pKeyUrl, uint dwKeyUrlLen, string pKeyBuf,
    uint dwKeyBufSize, uint pdwKeySize, uint pUserData)
    {
        Debug.Log("GetKeyExtCallback Callback Function");
        return 3000;
    }

    public void AESKeyCallbackFunc()
    {
        NEXPLAYERGetKeyExtCallbackFunc callback = new
        NEXPLAYERGetKeyExtCallbackFunc(GetKeyExtCallback);

        player.RegisterGetKeyExtCallBackFunc(callback);
    }
}
```

You should add this script to an GameObject named AESKey and pass it through the Nexplayer GameObject as a parameter for the script.(Drag and drop in the Inspector).

The function `GetKeyExtCallback()` is called each time that a new playlist is received that includes a stream with an encryption key. There are six parameters in this function:

Parameter	Description
<code>pKeyUrl</code>	Pointer to the URL of the encryption key
<code>dwKeyUrlLen</code>	Length of the URL of the key
<code>pKeyBuf</code>	Pointer to the buffer where the encryption key information will be stored
<code>dwKeyBufSize</code>	Size of the key's information buffer
<code>pdwKeySize</code>	Pointer to the length of the encryption key
<code>pUserData</code>	Data that the user passed when the callback was originally registered

5.24 Get Content Information and Content Statistics Information

Nexplayer™ provides an interface that can get Content Information and Statistics Information such as Video frame rates, Video Average frame rates, Video Codec, etc.. The interface should be implemented using the following script:

```
//Getting Content Information
public void EventNotify(NexPlayerEvent paramEvent, int param1, int param2)
{
    case NexPlayerEvent.NEXPLAYER_EVENT_INIT_COMPLETE:
        {
            UpdateContentInfo();
            //Log for example about Video Codec and Resolution.

            Log("ContentInfo \n" + "\t Video Codec : " +
contentInfo.VideoCodec.ToString("X") + "\n" + "\t Video Resolution : " +
contentInfo.VideoWidth + " x " + contentInfo.VideoHeight + "\n");
        }
        break;
    }

    case NexPlayerEvent.NEXPLAYER_EVENT_ON_TIME:
        {
            SetCurrentTime();

            //Update Content Statistic Info(every 60fps) and player should be finished
wrap up initialization

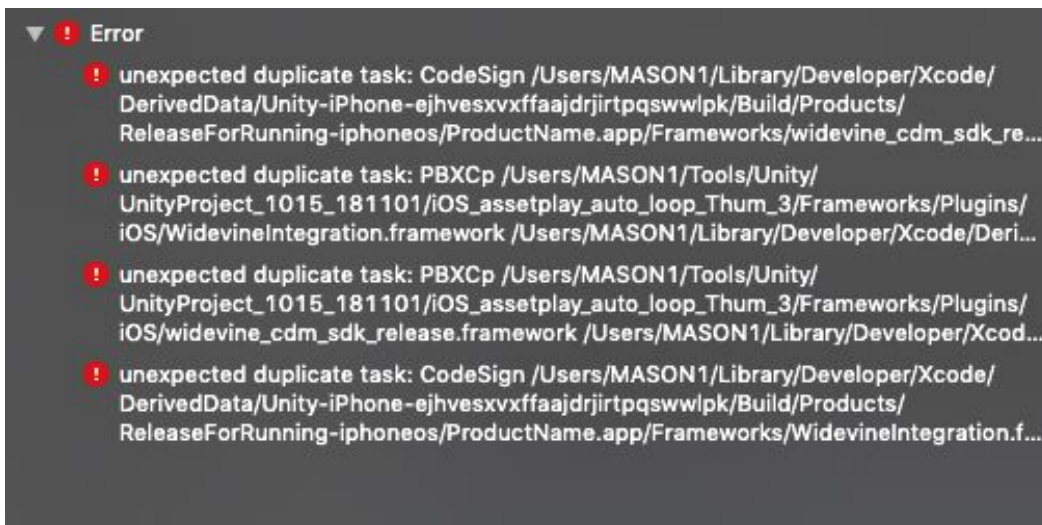
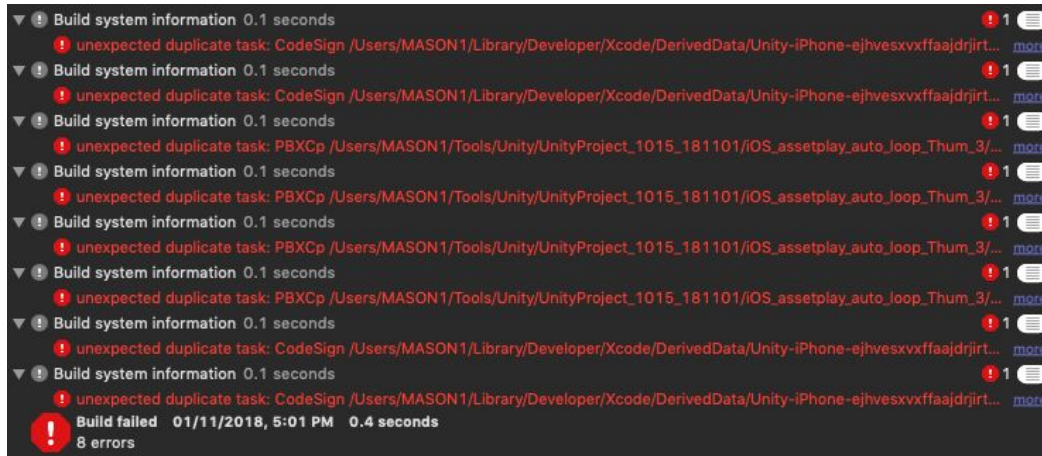
            if (player != null && player.GetPlayerStatus() >
NexPlayerStatus.NEXPLAYER_STATUS_CLOSE)
                {
                    UpdateContentStatisticInfo();
                    Log("Average Video FPS : " + statisticsInfo.avgFPS);
                    Log("Average Video Disp : " + statisticsInfo.avgDisp);
                }
        }
    }
}
```

```
        //Monitor change of the renderer size
        if (m_RenderMode != NexRenderMode.VR)
            MonitorRendererSizeChange();
    }
    Break;
}
```

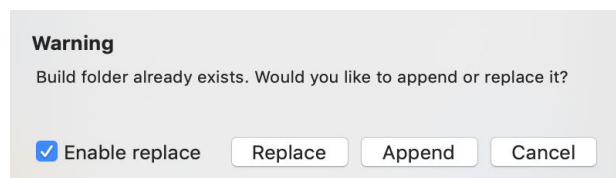
Content Info and Content Statistic Info type is defined at NexPlayerTypes.cs.

6. FAQ

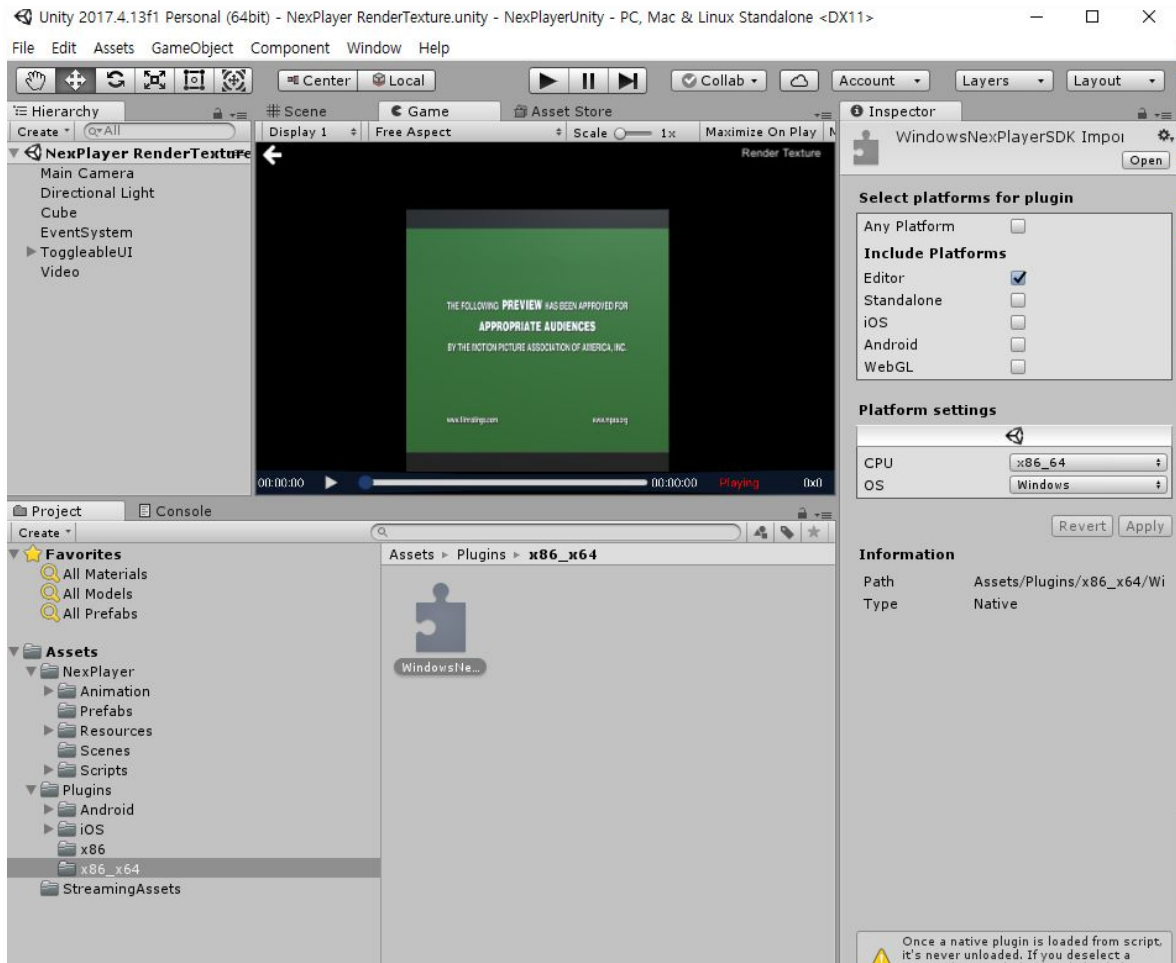
1. **Q.** How do I solve the problem below when building my project in iOS?



1. **A.** When you build your iOS project in the Unity Editor, create a new Xcode project, or if you want to overwrite your old project, check the 'Enable replace' checkbox and then select the option 'Replace'.



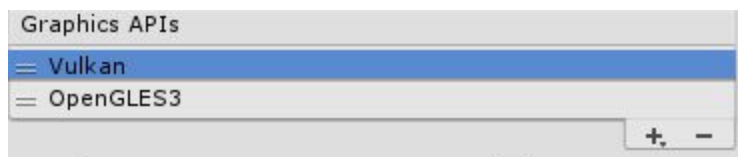
2. **Q.** Why is the video not working on the Windows Unity Editor?
2. **A.** This occurs when the Unity Editor platform is not specified in the Windows NexPlayer DLL. It should be specified as shown below:



3. **Q.** Why can't I play DRM content when I build my app on Android?
3. **A.** To allow any remote video on Android, the option 'Internet Access' needs to be set to 'Require' in the Unity player settings and the option 'Write Permission' should be set to External (SD Card). This configuration is needed to save the DRM certification data on the Android SDCard.

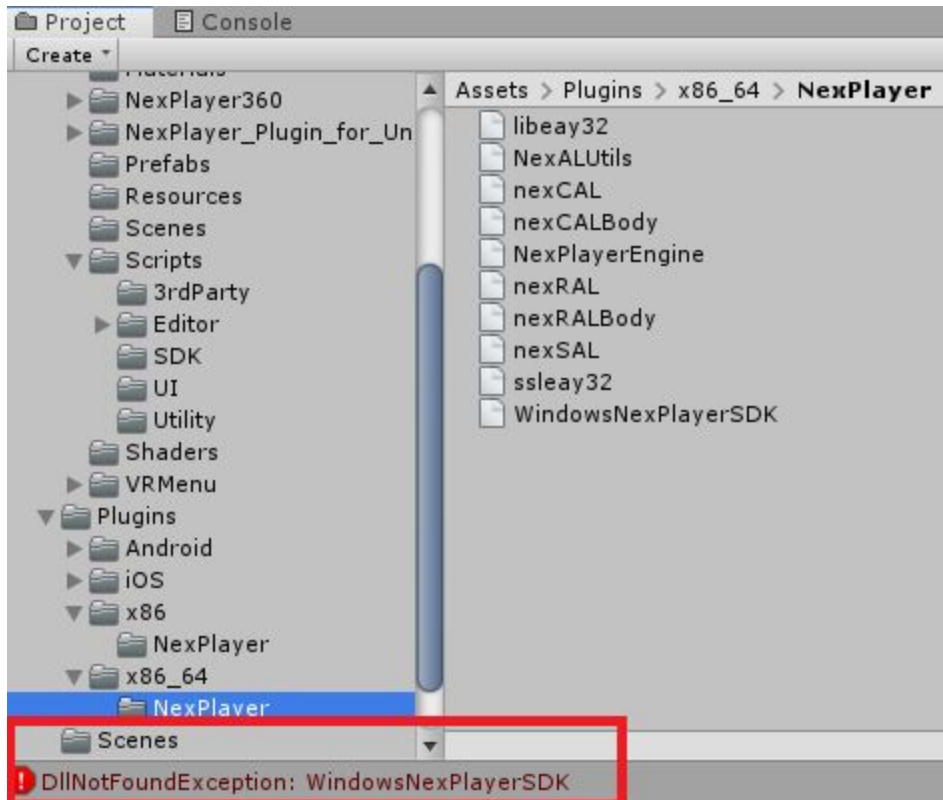


4. **Q.** Does the player support Widevine Auto License Renewal? How does it work?
4. **A.** Yes, our player supports that feature. Widevine Auto License Renewal is related to the server side. The server sends a 'message' event and if there is a renewal event in the message, then the license renewal is triggered by the Widevine CDM and initiates the 'message' event. The Widevine license validity is extended when the renewal event is triggered. There isn't a notable difference in the player's behaviour.
5. **Q.** Would I be able to load my videos from Streaming Assets if I enable the option Split Application Binary?
5. **A.** Yes, Split Application Binary is compatible with our player.
6. **Q.** Why can't I see the stream displayed on an Android device when I build a project with Unity 2019?
6. **A.** Review the configuration of the graphics APIs in the Project Settings. Unity sets Vulkan as the main graphics API by default. To use the NexPlayer™ Unity Plugin, you must select OpenGL ES3 as the main graphics API. Change the order or delete the Vulkan API option to solve this issue.



7. **Q.** Does the NexPlayer™ Unity Plugin support license files to verify the app ID?
7. **A.** Yes, let us know if your app needs this feature and we will provide you with a specific SDK that supports it.

8. **Q.** I have the following error on the console and I can't run the app:



8. **A.** There are missing compilation tools on the version of Visual Studio that you are using. Please restore or reinstall Visual Studio to solve this issue.

7. Detailed API

If you need more information about the different functions used in the NexPlayer™ Plugin for Unity, our Unity Package includes a detailed Reference Manual:

NexPlayer → NexPlayer_Plugin_for_Unity_Reference_Manual → Index.html

8. Technical Support

Our experts keep our player updated with the latest features and react very quickly to customer requests, providing excellent 24/7 technical support. If you are having problems that aren't answered in the FAQ chapter don't hesitate to contact us at supportmadrid@nexplayer.com and we will find a solution to your problem. We have specialists working around the clock to make sure that your problems get resolved as swiftly and efficiently as possible.

9. Legal Notes

9.1 Disclaimer for Intellectual Property

This product is designed for general purposes, and accordingly the customer is responsible for any and all intellectual property licenses required for actual application. NexStreaming Corp. does not provide any indemnification for any intellectual properties owned by third parties.

9.2 Copyright

Copyright for all documents, drawings, and programs related to this specification are owned by NexStreaming Corp. No part of the specification shall be reproduced or distributed without prior written approval by NexStreaming Corp. Content and configuration of any and all parts of the specification shall not be modified nor distributed without prior written approval by NexStreaming Corp.

© Copyright 2010-2019 NexStreaming Corp. All rights reserved.