Video Player Documentation

Introduction

Cross-Platform Video Player Plugin is a powerful tool designed to take your video playback capabilities in Unity to the next level. With this plugin, you can effortlessly integrate high-quality video playback functionalities into your Unity projects, providing your users with a seamless and immersive video experience. This plug-and-play plugin is incredibly easy to set up, allowing you to save time and effort while integrating advanced video player features. Whether you're developing a game, an interactive application, or a multimedia project, the Cross-Platform Video Player Plugin offers a range of essential functionalities to enhance your project's video playback capabilities.

One significant advantage of the Cross-Platform Video Player Plugin is its ability to play video in WebGL builds. Unity's native VideoPlayer component cannot play videos in WebGL builds, which can be a significant setback for developers looking to create web-based experiences. Fortunately, the Cross-Platform Video Player Plugin resolves this limitation by providing seamless video playback on WebGL builds. This means that you can now create web-based projects and include video playback functionalities without any restrictions, unlocking new possibilities for interactive websites, web-based games, e-learning applications, and more. This is a feature-rich video playback tool which utilizes the native VideoPlayer component and offers a comprehensive set of features, including seek controls, play/pause functionality, volume controls, next and previous video navigation, Playback Speed, Fullscreen mode, maintain video aspect ratio, thumbnail support, etc. It is cross-platform and compatible with WebGL, Android, iOS, and standalone builds.

Installation

To integrate the Video Player into your Unity project, follow these steps:

1. Import the Unity package (.unitypackage) into your Unity game project.
2. After successful import, locate the "Video Player Canvas" prefab.
3. Add the "Video Player Canvas" prefab to your scene. Make sure it serves as the root canvas and is not nested within another canvas. Keeping the Video Player in a

separate canvas is recommended as the "Controls Size" property affect the canvas scaler's resolution.

4. Congratulations! The Video Player is now added to your scene.

Adding Video Clips

To add video clips for playback, follow these steps:

1. Select the "Video Player" GameObject in the Hierarchy window.
2. In the Inspector tab, locate the "Video Clip Info Array" property of the "VideoPlayerCrossPlatform" script.
3. Edit the "Video Clip Info Array" property to add the video clips you want to play.

Programmatic Video Clip Management

In addition to assigning clips via the Inspector, you can also programmatically add video clips to the player's list at runtime using the `AddVideoClips()` method:

- **`AddVideoClips(VideoClipInfo[] videoClipInfoArray)`:** This public method allows you to add an array of `VideoClipInfo` objects to the player's internal list of videos. This is useful for dynamic content loading.

How to Play a Video (Basic Setup)

To get a video playing in your scene, follow these general programmatic steps (Check the demo scene and the demo script VideoPlayerDemo.cs for a better understanding.):

1. **Instantiate the Video Player Prefab:** Create an instance of the "Video Player Canvas" prefab in your scene.

```
videoPlayerInstance =
Instantiate(videoPlayerPrefab).GetComponentInChildren<Video
PlayerCrossPlatform>(true);
```

   a. *Note:* The `true` argument in `GetComponentInChildren` ensures that even inactive child GameObjects within the prefab are searched for the `VideoPlayerCrossPlatform` component.

2. **Add Video Clips:** Provide the `VideoPlayerCrossPlatform` instance with the list of videos it should manage.

```
videoPlayerInstance.AddVideoClips(videoClipInfoArray.ToArray());
```

3. **Show the Video Player UI:** Activate the video player's container to make its UI visible. This is crucial before loading the clip, as Unity's native `VideoPlayer` component requires its GameObject to be active for proper preparation.

```
videoPlayerInstance.ShowVideoPlayer();
```

4. **Load the First Video Clip:** Load the initial video by its index in the `VideoClipInfoArray`.

```
videoPlayerInstance.LoadVideoClip(0, autoPlayVideo);
```

    a. The `autoPlayVideo` boolean determines if the video starts playing immediately after loading (set this in the Inspector if `[SerializeField]` is used).

Showing and Hiding the Video Player

You can programmatically show or hide the Video Player by calling the provided functions in the "VideoPlayerBase" script.

1. Use the "ShowVideoPlayer()" function to display the Video Player.
2. Use the "CloseVideoPlayer()" function to hide the Video Player.

Disabling Control Buttons

If you prefer not to display certain control buttons (e.g., Replay, Next, Prev Buttons), you can simply disable them from the hierarchy. Locate the corresponding GameObjects in the hierarchy window and disable them as needed.

Playing Local Videos on WebGL

To play a locally stored video on the WebGL platform, follow these steps:

1. Place the video file inside the StreamingAssets folder. The StreamingAssets folder should be located at the root of the Project, directly within the Assets folder.
2. In the inspector, under the 'Video Clip Info Array' property of the Video PlayerCrossPlatform script, add a new element.
3. Change the Source of the newly added element to 'Streaming Asset'.
4. Input the file path of the video in the 'Url' field. Always include the subfolder path in the path used to reference the streaming asset if your video file is in subfolders. For example, If your video is directly in the StreamingAssets folder like StreamingAssets/intro.mp4, set the URL field to "intro.mp4" and If your video is in a subfolder like StreamingAssets/Videos/intro.mp4, set the URL field to "Videos/intro.mp4".
5. Congratulations! Your WebGL Video Player is now setup.

Subtitle Support

The plugin supports displaying subtitles from `.srt` files.

To add subtitles to your video clips:

1. **Place the Subtitle File:** Place your `.srt` subtitle file within the `StreamingAssets` folder (or any subfolder within it).
2. **Assign URL in VideoClipInfo:** For the corresponding `VideoClipInfo` entry for your video, locate the `subtitlesUrl` property.
3. **Enter File Path:** Input the file path of the `.srt` file relative to the `StreamingAssets` folder into the `subtitlesUrl` field. For example, if your subtitle file is `StreamingAssets/subtitles/mysubtitle.srt`, you would enter `"subtitles/mysubtitle.srt"`.

The `ParseSubtitles()` function is responsible for loading and parsing these files. If the file is not found, a warning will be logged to the console.

Keyboard Shortcuts

The plugin offers customizable keyboard shortcuts for various video player functionalities.

- **Enabling Shortcuts:** Ensure the `enableKeyboardShortcuts` property (part of `VideoPlayerCrossPlatform` properties, often found under a "SETTINGS" header) is enabled in the Inspector for keyboard input to be processed.

- **Default Actions (configurable via Inspector):**
  - **Play/Pause:** Configured by `playPauseToggleKey`.
  - **Seek Forward:** Skips forward by 5 seconds. Configured by `seekForwardKey`.
  - **Seek Backward:** Skips backward by 5 seconds. Configured by `seekBackwardKey`.
  - **Volume Up:** Increases volume by 0.1. Configured by `volumeUpKey`.
  - **Volume Down:** Decreases volume by 0.1. Configured by `volumeDownKey`.
  - **Next Video:** Plays the next video in the list. Configured by `nextVideoKey`.
  - **Previous Video:** Plays the previous video in the list. Configured by `prevVideoKey`.
  - **Restart Video:** Restarts the current video from the beginning. Configured by `restartVideoKey`.
  - **Toggle Mute:** Mutes/unmutes the audio. Configured by `muteKey`.
  - **Toggle Fullscreen:** Enters/exits fullscreen mode. Configured by `fullScreenKey`.

Caution: Using the Space Key for Shortcuts in Unity

In Unity's UI system, when a Button is clicked, it becomes the currently selected UI object. By default, Unity maps the Space and Enter keys to activate the selected object, as these keys are assigned as alternate inputs for the "Submit" axis in the Input Manager (Project Settings > Input Manager > Submit). This behavior can lead to unintended double-triggering when:

- You assign the Space key as a global shortcut (e.g., to toggle video playback).
- A UI button was previously clicked and is still selected.
- Pressing Space then both triggers the UI button and runs your custom input logic.

Solutions

1. **Deselect the UI object after button clicks:** Call `EventSystem.current.SetSelectedGameObject(null);` after a button is clicked to prevent the UI from responding to keyboard input like Space or Enter.
2. **Remove Space from the Submit input axis:** If you want to completely stop the UI system from responding to the Space key:
   a. Go to **Project Settings > Input Manager > Submit**.

b. Remove "space" from the Alt Positive Button field. This tells Unity to stop using Space as a default activation key for UI buttons.

Recommendation:

Avoid using the Space key as a global shortcut unless you're certain no UI elements are focused. If you must use it, ensure UI buttons are properly deselected after use or reconfigure the input bindings to avoid conflicts.

Video Drifting Concept and Correction

Video drifting refers to a scenario where the video playback time becomes desynchronized from an internal reference clock. This commonly occurs if the video gets stuck due to issues like buffering, or if the game goes into a pause mode (e.g., due to application minimization or loss of focus). When this happens, the native VideoPlayer component skips/fast-forward the video to match the internal clock time.

The Cross-Platform Video Player Plugin incorporates a robust drift correction mechanism to ensure a smooth and synchronized playback experience:

- **Reference Clock:** The plugin maintains an internal `referenceClock` that tracks the expected playback time, independent of the native Unity `VideoPlayer`'s time. This clock advances only when the video is actively playing and no other interfering operations (like seeking or buffering) are occurring.
- **Drift Detection (`CheckForDrifting()`):** A dedicated coroutine continuously compares the `VideoPlayer`'s actual current time with the `referenceClock`. If the absolute difference (`timeDelta`) between these two times exceeds a small threshold (e.g., 0.5 seconds), it's considered a potential drift.
- **Accumulated Drift (`driftedDuration`):** To prevent constant minor adjustments, the plugin accumulates `driftedDuration` whenever the `timeDelta` exceeds the threshold.
- **Resynchronization:** If the `driftedDuration` surpasses a secondary threshold (e.g., 0.2 seconds), the system identifies a significant drift. It then performs a `Seek()` operation to move the native `VideoPlayer`'s playback head to the `referenceClock`'s time, effectively resynchronizing the video and audio. A short pause (e.g., 1 second) is introduced after seeking to allow the player to stabilize.

This proactive drift detection and correction ensure a consistent and high-quality user experience, especially important for longer videos or systems under varying load.

VideoPlayerCrossPlatform Class Properties

The `VideoPlayerBase` script provides various parameters to control the behaviour and appearance of the Video Player. Here's an explanation of each parameter:

1. **VideoClipInfoArray:** An array of `VideoClipInfo` objects that stores information about each video clip to be played.
2. **VideoPlayerContainer:** The GameObject that acts as the container for the video player.
3. **VideoPlayer:** Unity's Native VideoPlayer component responsible for video playback.
4. **VideoFrame:** A Button component used to control the play/pause functionality. The `VideoFrame` button provides a convenient way for users to toggle the video player's play/pause state. By clicking the video frame, users can easily play or pause the video content without the need for additional play/pause buttons.
5. **Video Display:** A RawImage component used to display the video content. This defines the area in which the video will be rendered.
6. **Controls:** The parent gameobject of video player's controls UI elements.
7. **BottomControls:** A RectTransform component that defines the bottom controls area in the Video Player. The bottom controls area typically contains UI elements such as progress bar, time display, and playback controls.
8. **ThumbnailImage:** An Image component used to display the thumbnail of the currently selected video.
9. **LoadingIcon:** A Transform component representing the loading icon. This loading icon is displayed when the video is loading or buffering.
10. **PlayButton:** A Button used to control the play/pause functionality.
11. **PrevButton:** A Button component used to play the previous video clip.
12. **NextButton:** A Button component used to play the next video clip.
13. **ReplayButton:** A Button component used to replay the current video clip.
14. **FullscreenButton:** A Button component used to toggle Fullscreen mode.
15. **AudioButton:** A Button component used to toggle audio mute/unmute.
16. **AudioMuteImage:** An Image component used to indicate the audio mute/unmute state.
17. **AudioSlider:** A Slider component used to control the volume level.

18. **CenterPlayButton:** A Button component used to toggle the play/pause state of the video player. As the name suggests this button is positioned at the centre of the video player.
19. **CenterPlayButtonImage:** An Image component of the centre play button.
20. **CloseButton:** A Button component used to close the video player.
21. **TimeText:** A TextMeshProUGUI component used to display the elapsed time and video duration.
22. **ProgressBar:** A Slider component utilized to visually represent the video playback progress and can also be utilized to seek to a specific position in the video.
23. **CanvasScaler:** A CanvasScaler component used to control the size of the video player's controls.
24. **PlaySprite:** When the video player is playing, the sprite of the play button image will be replaced with this sprite, visually indicating that the video is currently playing.
25. **PauseSprite:** When the video player is paused, the sprite of the play button image will be replaced with this sprite, visually indicating that the video is currently paused.
26. **FullscreenEnterSprite:** When the video player is in windowed mode, the sprite of the Fullscreen button image will be replaced with the Fullscreen EnterSprite, indicating that clicking the button will make the video player enter the Fullscreen mode.
27. **FullscreenExitSprite:** When the video player is in Fullscreen mode, the sprite of the Fullscreen button image will be replaced with the Fullscreen ExitSprite, indicating that clicking the button will make the video player exit the Fullscreen mode.
28. **IsFullscreen:** Determines whether the video player starts in Fullscreen mode.
29. **PreserveVideoAspectRatio:** Determines whether the video's aspect ratio is maintained.
30. **NativeBackButton Closes Player:** Determines whether pressing the native back button closes the video player.
31. **DefaultVideo Duration:** The default duration string displayed when the video duration is not available.
32. **ControlsAutoHide Delay:** The delay in seconds before the controls automatically hide.
33. **ControlsFadeSpeed:** The speed at which the controls fade in and out.
34. **FullscreenAnimSpeed:** The speed at which the Fullscreen animation occurs.
35. **ReprepareInterval:** The time interval in seconds for attempting to re-prepare the video once an error has occurred. The error can be due to Internet connection or some other reason.

36. **LoadingAppearance Threshold:** The time threshold in seconds after which the loading icon is displayed while the video is loading or buffering.
37. **LoadingIconRotation Speed:** The rotation speed of the loading icon.
38. **DefaultVolume:** The default volume level of the video player.
39. **ControlsSize:** The size of the controls.
40. **ModifyOrientation:** Determines whether the orientation of the device should be modified when the Video Player is in fullscreen mode. If enabled, the device's orientation can be adjusted based on the specified fullscreen Orientation. This setting is specific to mobile platforms.
41. **DefaultOrientation:** Default orientation of the device when the Video Player is not in fullscreen mode. This setting is specific to mobile platforms.
42. **Fullscreen Orientation:** This variable represents the desired orientation of the device when the Video Player enters fullscreen mode. This setting is specific to mobile platforms.
43. **EnableKeyboardShortcuts:** (New) A boolean that determines whether keyboard shortcuts are enabled for controlling the video player.
44. **PlayPauseToggleKey, SeekForwardKey, SeekBackwardKey, VolumeUpKey, VolumeDownKey, NextVideoKey, PrevVideoKey, RestartVideoKey, MuteKey, FullScreenKey:** (New) KeyCode variables that allow you to customize which keyboard keys trigger specific video player actions.

VideoClipInfo Class Properties

The `VideoClipInfo` class holds information about a video clip to be played by the Video Player. It contains the following properties:

1. **Source:** Specifies the video content source, indicating whether to use the locally stored video clip, URL, or streaming asset as the source.
2. **Clip:** The video clip to be used as the content source when the source is set to `VideoClip`. This property is not used when the source is URL or StreamingAsset.
3. **Url:** The URL of the video to be used as the content source when the source is set to URL. This property is not used when the source is set to `VideoClip`. When the source is set to `StreamingAssets`, you need to enter the video's file path in the `Url` field.
4. **Duration:** The duration of the video clip. This value represents the length of the video and can be set manually or retrieved automatically. If left empty `DefaultVideo Duration` is shown until the video length is retrieved.

5. **Thumbnail:** A sprite representing the thumbnail image associated with the video clip. This thumbnail can be displayed in the Video Player UI.
6. **SubtitlesUrl:** (New) The URL or file path (relative to StreamingAssets) of the `.srt` subtitle file associated with this video clip.

VideoClipSource Enum

The `VideoClipSource` enum defines the possible video content sources for the `VideoClipInfo` class. It has the following values:

1. **VideoClip:** This option uses a Unity `VideoClip` asset assigned to the `clip` property of the `VideoClipInfo` object to play the Video. **Note:** Directly playing `VideoClip` assets is not supported in WebGL.
2. **URL:** This option plays a video from a direct URL, which is specified in the `url` property of the `VideoClipInfo` object. This is suitable for playing online video files across all supported platforms, including WebGL.
3. **StreamingAsset:** Represents playing a video in WebGL using a streaming asset. Use this option to play local video files in WebGL builds. When using this source, the VideoPlayer will play the video from the StreamingAssets folder. You should provide the file path of the video within the StreamingAssets folder in the `Url` property.

Android App Unable To Play Video From Web URL

If the video player fails to play an online video from a web URL on Android, despite working correctly in the editor and other platforms, changing the following options under 'Player Settings\Other Settings' might resolve the issue:

1. 'Internet Access' to 'Require'
2. 'Allow downloads over HTTP*' to 'Always Allowed'
3. Android may not allow HTTP videos to play even when the 'Allow downloads over HTTP*' option is set to "Always Allowed." In such cases, switching to HTTPS URLs is recommended.

Contact

Email: nareshbishtbusiness@gmail.com

Please consider leaving a review on the Unity Asset Store if you appreciate the asset.

Thank you!