

AudioLink Reference Guide

A reference guide for the AudioLink API.



! Learn to use this **Beta** feature, but use caution when shipping with it.

You can create a basic **AudioLink** implementation in just a few lines of code with just a `IAudioLinkFactory` definition and registration. However, other API types can help you further build out your implementation.

Below you'll find summaries of each of the core AudioLink C++ types and some code examples.

i You can find the AudioLink source code at `Engine\Source\Runtime\AudioLink\`.

IAudioLinkFactory

Every AudioLink implementation needs a factory class derived from `IAudioLinkFactory`, which has several pure virtual functions that define the implementation's entry points.

You must implement each function, as returning null results in an assert. Unreal Engine (UE) expects all calls dispatched to be thread-safe and ideally lockless to return efficiently.

You can use parameter structs instead of numerous individual parameters to simplify the extension of these function calls.



You can only register a single factory object; additional factories assert a fatal error on startup.

Plugin Implementation

AudioLink uses the `IModularFeature` interface to extend UE. The `IAudioLinkFactory` constructor handles registration, so creating an instance of your factory is sufficient for recognition. Handle this in the `StartupModule()` of your implementation.



There are different ways to implement plugins in UE, but the `IModularFeature` interface is the most common for audio extensions.

Example: Submix AudioLink Implementation

```
1 // Parameters use when creating a Submix Audio Link
2 struct FAudioLinkSubmixCreateArgs
3 {
4     TWeakObjectPtr<const USoundSubmix> Submix;
5     FAudioDevice* Device = nullptr;
6     TWeakObjectPtr<const UAudioLinkSettingsAbstract> Settings;
7 };
8
9 // Create a Submix Audio Link.
10 // @param InCreateArgs Arguments used to create the AudioLink instance
11 // @return The newly created Link instance (if successful).
12 virtual TUniquePtr<IAudioLink> CreateSubmixAudioLink(const
    FAudioLinkSubmixCreateArgs& InCreateArgs) = 0;
```

 Copy full snippet

```
1  /**
2   * Parameters use when creating a Submix Audio Link
3   */
4   struct FAudioLinkSubmixCreateArgs
5   {
6   TWeakObjectPtr<const USoundSubmix> Submix;
7   FAudioDevice* Device = nullptr;
8   TWeakObjectPtr<const UAudioLinkSettingsAbstract> Settings;
9   };
10
11  /**
12   * Create a Submix Audio Link.
13   * @param InCreateArgs Arguments used to create the AudioLink instance
14   * @return The newly created Link instance (if successful).
15   */
16  virtual TUniquePtr<IAudioLink> CreateSubmixAudioLink(const
    FAudioLinkSubmixCreateArgs& InCreateArgs) = 0;
```

 Copy full snippet

Example: Factory Implementation

```
1  class FAudioLinkExampleFactory : public IAudioLinkFactory
2  {
3  public:
4      FAudioLinkExampleFactory() = default;
5      virtual ~FAudioLinkExampleFactory() = default;
6
7      static FName GetFactoryNameStatic();
8
9  protected:
10     /** Begin IAudioLinkFactory */
11     FName GetFactoryName() const override;
12     TSubclassOf<UAudioLinkSettingsAbstract> GetSettingsClass() const override;
13     TUniquePtr<IAudioLink> CreateSubmixAudioLink(const
        FAudioLinkSubmixCreateArgs&) override;
```

```

14 TSharedPtr<IAudioLink> CreateSourceAudioLink(const
    FAudioLinkSourceCreateArgs&) override;
15 FAudioLinkSourcePushedSharedPtr CreateSourcePushedAudioLink(const
    FAudioLinkSourcePushedCreateArgs&) override;
16 FAudioLinkSynchronizerSharedPtr CreateSynchronizerAudioLink() override;
17 /** End IAudioLinkFactory */
18 };

```

 Copy full snippet

UAudioLinkSettingsAbstract

Every AudioLink implementation needs a settings class derived from

`UAudioLinkSettingsAbstract` to create links with associated data, like buffer sizes, handshakes, and UAsset references.

Example: Settings Implementation

```

1 UCLASS(config = Engine, defaultconfig)
2 class AUDIOLINKEXAMPLERUNTIME_API UAudioLinkSettingsExample : public
    UAudioLinkSettingsAbstract
3 {
4     GENERATED_BODY()
5
6     UPROPERTY(Config, EditAnywhere, Category = "Example|AudioLink")
7     float MyBufferSize = 1.0f;
8 };
9
10 class FAudioLinkSettingsProxyExample : public IAudioLinkSettingsProxy
11 {
12 public:
13     FAudioLinkSettingsProxyExample(const UAudioLinkSettingsExample&);
14     virtual ~FAudioLinkSettingsProxyExample() = default;
15
16     float GetMyBufferSize() const { return MyBufferSize; }
17
18 private:
19 #if WITH_EDITOR

```

```

20 void RefreshFromSettings(UAudioLinkSettingsAbstract* InSettings,
    FPropertyChangedEvent& InPropertyChangedEvent) override
21 {
22     MyBufferSize = CastChecked<UAudioLinkSettingsExample>(InSettings)-
        >MyBufferSize;
23 }
24 #endif //WITH_EDITOR
25
26 float MyBufferSize = 0;
27 };

```

 Copy full snippet

IAudioLinkSettingsProxy

AudioLink Settings follow a proxy design pattern, so they can safely exist outside the game thread where audio typically executes. A thread-safe copy of the settings is created and attached as a shared pointer to the setting's `UObject`.

This approach has two primary benefits.

- It provides garbage collection protection, as any threads that still have the shared pointer can continue to operate safely.
- `PostEditChangedProperty` automatically ferries data to the proxy when changes are made in the editor so that settings can behave like a standard asset.

All AudioLink Settings assets must implement a proxy of themselves and handle refreshing when the owning Asset changes via the `RefreshFromSettings` function.

Additionally, you should implement default settings for your properties, which serializes in the `DefaultEngine.ini`. In the Example: Settings Implementation, this is achieved via the `defaultconfig` markup on the settings object.



The settings default is used in cases where the property has not been set, so it is unnecessary to pass it every time you create an AudioLink.

UFactory Implementation

Unreal Engine serializes AudioLink Settings as an asset. As a result, you are responsible for implementing an asset factory for the settings class so you can create the asset.

Example: Standard Settings Factory Implementation

```
1 class FAssetTypeActions_AudioLinkExampleSettings : public
  FAssetTypeActions_Base
2 {
3 public:
4 virtual FText GetName() const override;
5 virtual FColor GetTypeColor() const override;
6 virtual const TArray<FText>& GetSubMenus() const override;
7 virtual UClass* GetSupportedClass() const override;
8 virtual uint32 GetCategories() override;
9 };
10
11 UCLASS(hidecategories = Object, MinimalAPI)
12 class UAudioLinkExampleSettingsFactory : public UFactory
13 {
14 GENERATED_UCLASS_BODY()
15
16 virtual UObject* FactoryCreateNew(UClass* Class, UObject* InParent, FName
  Name, EObjectFlags Flags, UObject* Context, FFeedbackContext* Warn)
  override;
17
18 virtual uint32 GetMenuCategories() const override;
19 };
```

 Copy full snippet


IAudioLink

`IAudioLink` is the primary abstraction used by the `IAudioLinkFactory` and is often returned by a `TUniquePtr`. It's an opaque type designed to contain the hidden details of plugin-specific implementation and in a thread-safe shared pointer to both a consumer and

producer. The consumer object maintains a weak reference to the producer and safely terminates the connection when the producer is deleted, typically when the link ends due to elapsing lifetime.

Example: Instance Implementation

```
1  /* AudioLink Instance, a container holding shared pointers for lifetime  
   management. */  
2  struct FExampleLink : IAudioLink  
3  {  
4      // Circular buffer (submix/source) that listening for new buffers  
5      FSharedBufferedOutputPtr ProducerSP;  
6  
7      // Example client  
8      FSharedExampleAudioInputClientPtr ConsumerSP;  
9  
10     // ...  
11 };
```

 Copy full snippet

IAudioMixerPlatformInterface

When you register an AudioLink Factory with UE, an AudioLink Platform Mixer is instantiated and hooks into AudioLink instead of the platform's hardware. Traditionally, third-party libraries have had to modify `.ini` files to prevent this, but AudioLink makes this less troublesome.

IBufferedAudioOutput

Most AudioLink instances use a class derived from `IBufferedAudioOutput` to create a producer object. Internally, they function as a circular buffer of pulse-code modulation (PCM) data waiting to be drained by a consumer object.



A circular buffer is a first in, first out (FIFO) data structure that buffers PCM data across UE boundaries. For AudioLink, these are atomic but not lockless.

Implementations of this interface are already defined in the source code

(`FBufferedSourceListener`) and (`FBufferedSubmixListener`), so it's likely unnecessary to build your own derived versions of these types.

Often, the link is created before the format of the incoming audio is recognized. As a result, an (`OnFormatKnown`) delegate fires once the format is known. You should set the delegate and start playback when it occurs.



Give careful consideration to how much space you allocate for these objects. A size too large can lead to latency, but too small can lead to buffer underruns. Typically, you should use a size at a 2:1 or higher ratio of the consumer's bitrate.

Lifetime of a Link Instance

The lifetime of a link instance depends on its type.

- A source link generally lives for the lifetime of a source's playback. Once over, the source is deleted, severing the link. This severance is mainly due to the format of the source potentially being different for every new source.
- A submix link is open while the submix is running, which is usually the application's runtime. This can lead to issues caused by instance lifetime, especially in the editor. See [Troubleshooting](#) for more information.

IAudioLinkSynchronizer

Every AudioLink implementation needs a synchronizer class derived from

(`IAudioLinkSynchronizer`), which contains registration and removal calls for setting various callbacks. This class synchronizes to another clock source, so UE and the external application can stay in sync.

You must provide thread-safe delegates for each of these callbacks:

- `Suspend`
- `Resume`
- `OpenStream`
- `CloseStream`
- `BeginRender`
- `EndRender`

Use `OpenStream` to notify UE of the format of an external AudioLink Factory. This notification is most useful for matching block / sample rates and channel counts.

Use `BeginRender` and `EndRender` to synchronize UE with the external renderer.

Example: Synchronizer Implementation

```

1 struct FExampleSynchronizerAudioLink : IAudioLinkSynchronizer
2 {
3     IAudioLinkSynchronizer::FOnSuspend OnSuspend;
4     IAudioLinkSynchronizer::FOnResume OnResume;
5     IAudioLinkSynchronizer::FOnOpenStream OnOpenStream;
6     IAudioLinkSynchronizer::FOnCloseStream OnCloseStream;
7     IAudioLinkSynchronizer::FOnBeginRender OnBeginRender;
8     IAudioLinkSynchronizer::FOnEndRender OnEndRender;
9
10    FRWLock RwLock;
11
12    // ...
13
14    #define MAKE_DELEGATE_FUNC(X)\
15    FDelegateHandle Register##X##Delegate(const FOn##X::FDelegate& InDelegate)\
16    override\
17    {\
18        FWriteScopeLock WriteLock(RwLock);\
19        return On##X.Add(InDelegate);\
20    }\
21    bool Remove##X##Delegate(const FDelegateHandle& InHandle) override\
22    {\
23        FWriteScopeLock WriteLock(RwLock);\
24        return On##X.Remove(InHandle);\

```

```
24 }  
25  
26 MAKE_DELEGATE_FUNC(Suspend)  
27 MAKE_DELEGATE_FUNC(Resume)  
28 MAKE_DELEGATE_FUNC(OpenStream)  
29 MAKE_DELEGATE_FUNC(CloseStream)  
30 MAKE_DELEGATE_FUNC(BeginRender)  
31 MAKE_DELEGATE_FUNC(EndRender)  
32  
33 #undef MAKE_DELEGATE_FUNC  
34 };
```

 Copy full snippet