Charisma SDK Unity Code Quick Start

For basic information on how to quickly add new functionality to a Plug-n-Play project, check out the Charisma Unity SDK quick start videos here: https://charisma.ai/docs/plug-n-play/unity

This coding quickstart guide assumes that you have both the SDK and the Plug-n-Play packages installed.

Connectivity

- `Playthrough` is the root class that controls the communication with Charisma, through the `Colyseus` plugin, and Charisma `Api`
    - Colyseus is a 3rd party library used to manage multiplayer game connectivity with WebSockets. More information here: https://github.com/colyseus/colyseus. It is used to continuously receive playthrough events,
    - The `Api` class contains static methods used to communicate with Charisma in order to generate playthrough tokens used in Colyseus, as well as setting memories.
- In order to create a create a connection with Charisma, you will need to create a script that inherits from `PlaythroughInstanceBase`, which is a class that governs a `Playthrough`
- The inherited PlaythroughInstance can then be placed in the scene, and takes various parameters to initialize a Playthrough - `storyID, storyVersion, apiKey`, `startGraphReferenceId`
- The inherited PlaythroughInstance also has override functions **OnPlaythroughLoaded** and `OnPlaythroughLoaded` which you can use to extend playthrough behavior
- In order to start a playthrough, call the inherited function `PlaythroughInstanceBase.LoadPlaythrough` when it is most convenient. In the Plug-n-Play example script (PnPPlaythroughInstance), it is called on `override Start`

Messages

- Most Playthrough messages are sent and received via Colyseus, in the Playthrough class. You can find example usages of this by searching either `_room?.Send` and `_room.OnMessage` in the Playthrough class.
- Callbacks to room messages are assigned in `Playthrough.AssignRoomCallbacks`.
- Message types that are receiveable are defined in `Events`

- Message types that are sendable are defined in `ClientEvents` class
- When you subscribe to a Colyseus event through `_room.OnMessage<T>` - Colyseus will automatically handle the callback to subscribers if the message type matches
- Besides Events and ClientEvents, you can set memories by calling `Api.SetMemory`
- The most notable event is the `MessageEvent`, which is a general event from Charisma when you hit a new node in the graph. It included useful data such as the `message, metadata, emotions, memories`. The message field contains other critical data such as which character is speaking, metadata, and the generated speech.

Charisma Entities and Actors

- In the Plug-n-Play, Charisma links to and governs all Charisma entities in the scene. These entities are either inherited from `CharismaPlaythroughEntity` or `CharismaPlaythroughActor`
- `CharismaPlaythroughEntity` are simple entities that can be used in meta functions to help in performing certain tasks.
    - E.g. `CharismaMoveToEntity` is a waypoint in the scene and is used in the `MoveToFunction` to allow characters to move to specific locations, and it can be easily referenced in the graph through metadata
- `CharismaPlaythroughActor` are characters in the scene that are linked to a Charisma graph actor through the `_characterId`
    - E.g. The `CharismaHumanoidActor` is a full extension of the class which can receive messages, and control NPC behavior through `HumanoidNPCCharacterController`

Meta Functions

- In the Charisma graph, you can add KVP meta data to send to the client when the playthrough gets to a desired node
- This metadata received and included in `MessageEvent.message.metadata`
- In `PnPPlaythroughInstace.OnMessageReceived` this metadata is passed to the relevant meta function, by matching the key of the received metadata, to the meta function `MetadataID`
- `MetadataFunction` is the root scritptable object class representing an executable function during a playthrough.

- In order to create a new meta function, you can inherit this class, and override the `MetadataID`, and the `Execute` function
- `_metadataDependencies` are passed into the class through `PnPPlaythroughInstance.Start`. This provides each function with all the runtime Charisma entities in the scene that the function can then reference
    - Look into `PlayAnimationFunction` for an example usage of a metadata function
- After you create a new inherited meta function, you must create a scriptable object of the meta function.
    - You can create all the missing meta functions scriptable object with the **Charisma>Create Playthrough Metadata** toolbar menu
    - After the new SO is created, you must add the SO into your `PlaythroughInstance` object in the scene.

Animation

- NPCs are handled through `HumanoidNPCCharacterController`
- Animations for NPCs are handled through `HumanoidNPCAnimationController` and `HumanoidNPCAnimationConfig`
- Animations are either triggered automatically based on `AnimationFlags`, or requested directly by adding a new animation request (example in `PlayAnimationFunction` class)
- Animator animates the idle/walking/talking/listening/mannerisms states automatically based on flags and associated Charisma emotions
- E.g. in **GeraldineAnimationConfig** scriptable object, we have **Talking_1** and **Talking_2** animations with flags Talking, Standing - these will play when character is talking. If you attached an emotion to one of these, it would prioritize talking animation with the specified emotion
- Requested animations override the current playing animation, as soon as it's available to do so
- Request-only animations shouldn't be assigned any flags, as they shouldn't be governed by the anim system, and instead requested through the graph metadata (or whenever else you want to request them)
- Facial expressions are governed by the current Charisma emotion attached to the character. These are called by `HumanoidNPCCharacterController.ApplyEmotion` automatically, or requested with a meta function RequestFacialExpressionFunction
- Blendshapes for expression are in the Plug n Play samples, under Example>Animation>Data>Emotions

- Lipsync is handled by `LipsyncOVR` components, attached to the character prefabs, targeting character mouth blendshapes
    - For more information on how LipsyncOVR can be implemented seamlessly into the project, check out our **tutorial video**.

Animation Flags

Flag	**Usage**
None	Animations with no flags are to be triggered manually. In the Plug-n-Play, this is done through `PlayAnimationFunction`
Standing	Animation for when a character is not walking.
Walking	Animation for when a character is walking.
Idle	Default animation for when a character is not performing any actions.
Talking	Animation for when a character is talking.
Listening	Animation for when a character is listening to player speech.
Mannerism	Animation that is triggered randomly during in between idling.