CFSoundSystem for Music and Sounds

Version History:
9/28/2023 First draft documentation of the new SDK7 system
10/3/2023 Brought it up to current SDK7 code

Table of Contents

Scope

This document describes a number of advanced model, view, and control modules for managing sound or especially music, in a Decentraland scene. A typical use is a DJConsole for streamed audio or music file playing, and/or a Jukebox setup.

Audience

Internal reference documentation for the developer of the system.
Possible developer-user subscribers or licensees.

Components at a glance

The components in this collection of classes can be assembled to produce a “CFSoundSystem”. More details in subsequent sections

- **CFSoundSystemModel** - the “Model” in this system.
  This class understands “Contexts”, “Catalogs”, and “States”, and it gets them from the cloud, based on settings in a CFSoundSystemConstants file. The cloud service and database are the same in this SDK7 version as they were in SDK6.

- **CFSoundSystemController** - the “Controller” in this system.
  This class orchestrates the relationship between one or more views (UX devices, sound players) and a model (e.g. the CFSound System.)

- **DJConsole** - This class presents a 3D UX “view” for the system

- **CFJukeBox** - This class presents a different 3D UX “view” for the system

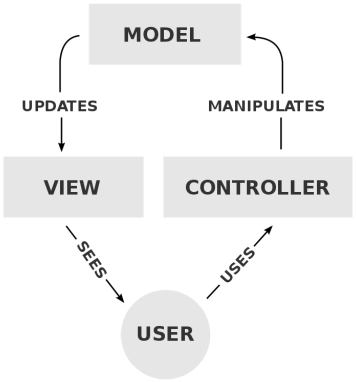
- **CFSoundPlayer** - This class manages the playing and stopping of sound from the location of a single Decentraland Entity. It can play streams from the cloud, or files deployed in the scene. There can be multiple CFSoundPlayers in a scene. They can be used standalone, or they can be children of a CFSoundSystem

- **DJBooth** - This class presents a DJBooth. A DJConsole is typically placed on top of a DJBooth. This DJBooth has several 3D Model “styles” that can be selected.

- **DJBoothSetupButton** - (lives in the DJBooth code file). A pair of these buttons is typically placed in some not-so-obvious place in a scene, by which a scene manager can, from in-world, change the booth appearance.

MVC Design Pattern

The CFSoundSystem is based on an MVC design pattern.



Communication to the Model and the Controller are by way of method calls which are abstracted by Interfaces. Communication between the Model's client code and the cloud service, which maintains the state of the system for various scenes, is by a REST service. Communications originating from the Model and Controller to the View(s) are by events (using the required EventManager module)

Model (ISoundSystemModel interface)
The Model represents the catalog of sounds and the state of sound in a scene and, if the system is in useService mode, is connected to a cloud service that persists the model.
It is sent commands from the Controller for state changes.
The model performs the state changes, and issues Events back to any listeners.
The Views typically listen to these events and adjust their appearance and sound generation as a result.
In this system, the **CFSoundSystemModel** class is the scene-side of the Model, implementing the ISoundSystemModel interface.

View(s)
There are both Visual and Audio "Views" in this system.

> **Visual**: the in-world UIs/UXs that show the status of sound or music that is selected, playing, etc, and which may provide UI controls, such as buttons, by which a user may provide input to the Controller
> Examples are the **DJConsole** and **CFJukeBox** classes.
>
> **Audio**: The **CFSoundPlayer**, which receives load/play/stop command calls, with sound specs provided.
> It is typically operated by commands from one View in the scene
> It sends out Events, to which that View typically listens.
> The event sent is when a fixed-duration audio sound has ended.

Controller (ISoundController interface)

The Controller orchestrates the system
It receives state-change-request calls from the UI Views as a result of user actions upon them
It forwards these change requests to the Model.
It receives change events from the Model
It issues events (but in the typical architecture they are ignored by the Views)
    These are forwarded to the View(s) as events
In this system, the **CFSoundSystemController** class is a Controller, implementing the ISoundSystemModel interface.

Component Details and Relationships

Model

CFSoundSystemModel

This class understands “Contexts”, “Catalogs”, and “States”..
It polls the cloud service at a regular interval, getting the relevant Context, Catalog and State, and processes that data, firing events to a controller/orchestrator (e.g. CFSoundSystemController) for it to handle changes that have occurred in the model.
If in useService mode,
  it gets the current state of the model from the cloud,
  based on settings in a CFSoundSystemConstants file
else
  It gets them from a local file initializer and maintains the model state in local client memory.
It also receives model state-change requests from the Controller, makes them to the model, and issues events for various types of changes.

Controller

A controller must implement the ISoundSystemController interface, and generate the required SoundControllerEvent events.

CFSoundSystemController

This is the MVC Controller..
It is set up by the scene, and is given the event handlers of Views in the scene.
This class receives change requests from the Views, and forwards them to the Model
It can provide events, but typically the Views get their events from the Model

Views

NOTES:
1. One of the Views (typically the DJConsole, but potentially a JukeBox) is configured to
a. operate the system's CFSoundPlayer, calling it to load, play and stop sounds, and
b. receive events from the CFSoundPlayer when sounds (of fixed duration) have ended.
2. One of the Views (typcially one JukeBox, but potentially the DJConsole) is set to handle an optional queued playlist of songs or sounds, advancing to the next sound in the playlist when the former one has ended.

DJConsole

This class presents a 3D UI to display information from the Model and to send user change requests to the Controller.-

JukeBox

This class presents a 3D UI to display information from the Model and to send user change requests to the Controller.

CFSoundPlayer

This class is basically a "View" that produces audio output, and that and signals end of sounds. This class manages the playing and stopping of sound from the location of a single Decentraland Entity. It uses an AudioStream to play streams from the cloud, or an AudioSource to play MP3 files that have been deployed in the scene.

It can be used standalone, or they can be a components in a CFSoundSystem, in which case they are typical managed by one View

Although there could be multiple CFSoundPlayers in a scene, only one will be heard at a time (the most recent one that has been told to Play.) As a result, the typical configuration of this system is to use one CFSoundPlayer, and to pass it to the constructor of *one and only one* of the Views (DJConsole or JukeBox), which will manage it for the entire system.

Note that only one AudioStream or AudioSource can play at a time in a scene: The latest AudioStream or AudioSource to play a stream or file silences any other ones that may have been playing, and the old ones don't resume automatically if the new one is stopped..

MP3TYPE sounds have a duration and the CFSoundPlayer keeps track of remaining time when they play.

API:

- Set up the player ID to use in Event-based callback
- Set the Event listener

- Remove and loaded sounds
- Loading a Sound File (type: MP3TYPE)
- Loading a Stream (type: STREAMTYPE)
    - Only one Sound File or Stream can be loaded at a time
- Play
- Stop
- Setting the duration that a sound should play
- Reducing the amount of time remaining to play (to be called from a system function)

Events Generated:

- An MP3TYPE sound has reached its play duration

Other Infrastructure Classes

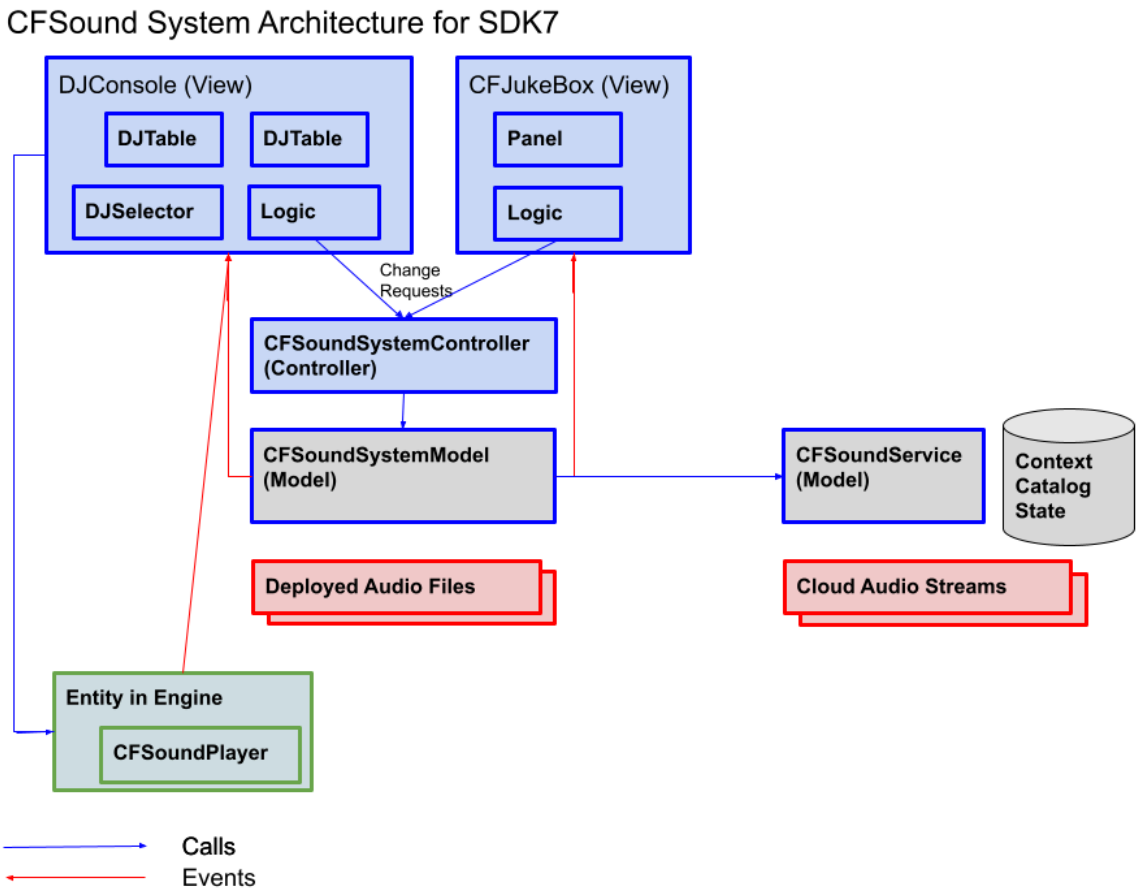
DJBooth – Physical infrastructure on which a DJConsole can be placed

This class presents a DJBooth. A DJConsole is typically placed on top of a DJBooth. This DJBooth has several 3D Model “styles” that can be selected.

DJBoothSetupButton - Changes appearance of the DJBooth

This class lives in the DJBooth code file, presents a 3D Button, and a pair of these is typically used select the next or previous style of a DJBooth. .

CFSoundSystem Architecture



Typical Setup Sequence

[ ] Review this against the final coding

1. Instantiate a CFSoundSystemModel

a. Pass in context, realm, local data, refresh interval (0.5), useSystem (true), debugLevel
b. Example code:

```
c.    let musicModel:ISoundSystemModelAPI = new CFSoundSystemModel(
d.        (this.context && this.context !=null && this.context != 
    "")?this.context:CFSoundSystemConstants.context,
e.        (this.realm && this.realm !=null && this.realm != 
    "")?this.realm:"all",
f.        localSoundSystemContext, // initial values,
g.        localSoundSystemState,   // unless service gets new ones
h.        localSoundSystemCatalog,
i.        0.5, // polling interval, in seconds.
j.             // 0.3 works with some ignored intervals
k.        true,// useService.
l.             // If false, it will keep the provided defaults above
m.        LOG_VERBOSITY // debug level
n.    )
```

2. Instantiate a CFSoundSystemController

a. Pass into constructor the Model, and debug level
b. Example Code:

```
c.     let musicController:ISoundSystemController = new 
    CFSoundSystemController(
d.        musicModel,   // the controller will internally attach to the 
    model's events
e.        LOG_VERBOSITY // debug level
f.     )
```

3. (Optional) Set up a DJBooth

a. Example code:

```
    let djBooth = new DJBooth(
       {
         parent:scene,
         position: Vector3.create(29,1.2,16),
         rotation: Quaternion.fromEulerDegrees(0,0,0),
         scale:Vector3.create(1,1,1)
       },
       1,
       showDjConsole // callback function(boolean) that will show/hide the View UI
     )
```

l.

4. (Optional) Set up two DJBoothSetupButtons

To call the show/hide method of the DJBooth

5. Set up a CFSoundPlayer

This will produce that actual sound in the scene. It will be the sound player for both the DJConsole and the Jukebox(es)

a. Example code:

```
     let cfSoundPlayer = new CFSoundPlayer(
       0, // playerId // yyy might be obsolete, though it is sent out with events
       djBooth.rootEntity, // must tie it to some entity in the scene
       DEFAULT_FILE_LOG_VOLUME,
       DEFAULT_STREAM_LOG_VOLUME,
       LOG_VERBOSITY
     )
```

6. Set up some “View(s”)

One of them should be the CFSoundPlayer handler
And one of them should be a queue handler if any of the Views can build playlist queues. Examples would be:

a. (Optional) Set up a DJConsole

// e.g. on top of the DJBooth
DJConsole instantiates two CFSoundPlayers
Example code:

```
b.      let djConsole = new DJConsole(
c.        {
d.          parent:djBooth.rootEntity,
e.          position: Vector3.create(-0.24,0.12,0),
f.          rotation: Quaternion.fromEulerDegrees(-40,-90,0),
g.          scale: Vector3.create(1,1,1)
h.        },
i.        musicModel,
j.        musicController,
k.        cfSoundPlayer,
l.        true, // handleQueueAdvancment
m.        LOG_VERBOSITY // debugLevel
n.      )
o.
p.      // Function to allow the visible part of the DJConsole to be
    shown/hidden by
q.      // the DJBooth selector -- when DJBooth name is "" the booth
    (and console) should be hidden
r.      function showDjConsole(show:boolean) {
s.        console.log("/n*** DJConsole.show("+show+") still needs to be
    implemented for hiding Console if booth name = ''")
t.        djConsole.show(show)
u.      }
v.
w.      showDjConsole(true)
x.
y.
```

b. (Optional) And/or, set up one or more JukeBox

Example code:

```
    let jukeBox = new CFJukeBox(
      {
        parent:scene,
        position:Vector3.create(29,0,11),
        rotation:Quaternion.fromEulerDegrees(0,0,0),
        scale:Vector3.create(0.63,0.63,0.63)
      },
      musicModel,
      musicController,
      null, // In this scene, we'll let the DJConsole manage the CFSoundPlayer
      false,
      LOG_VERBOSITY // debugLevel
    )
```

Additional Notes

Locus, PlayerId, and Table

The term “locusNum” (or “loci” array in the plural) refers to an abstraction of a UI location. It is represented in the Model as locusNum:number (or an array of them as “loci”. The database stores them in the “loci” array field in State documents.

Currently States support exactly two loci. (0 and 1)
A DJConsole has two “DJTables” – the small upper panels that each display a loaded song, and manage play, stop and volume. These two tables are mapped to loci 0 and 1 in the model and database
A jukebox has one song showing/controllable, and is mapped to locusNum 0 (loci[0]).

Each locus can optionally have a queue, e.g for a song playlist managed by jukebox(es).

In the SDK6 version of the code, and perhaps still a remnant in SDK7 code in development, there were separate CFSoundPlayers for each locus. In SDK7 there is one CFSoundPlayer for the entire system, and references to playerID should be changed to locusNum

Mode

Stored as a top level field in Context schema
Values are:

> “**CONSOLE**” - Only DJs/Admins can operate the system
> “**USER**” - users can also operate the system using something like a Jukebox, and there should be separate states for each

Scenes will use this depending on the control/realm setup desired
E.g. if there is only an administrative View, probably this is set to CONSOLE
but if there is the ability for any player to control the system for everyone in the same realm/island, this would be USER.
In scenes with both a DJConsole and a CFJukebox, the console can switch modes.

data.djBoothIndex:

Stored as a member of custom data field in Context schema
(not all scenes will use this)
Used by the DJConsole to persist, across all realms, the current index of the DJConsole style

Use Cases

- The Unity Cafe in the Decentraland Conference Center has a complete setup, with both a DJConsole and a CFJukeBox.'
- The on top of the tower in the DXTower scene in District X
- An SDK7 demo scene at 5,117

Known Issues

- The code conflates "playerId" and "locus
- As of 10/2/2023 the SDK7 version of this system, which is a major rewrite, does not exist in the Unity Cafe, but is demonstrated at Carl's SDK7 demo/test scene, at 5,117. The Unity Cafe s of this date still uses SDK6 version

- CFJukeBox is not fully implemented for SDK7 as of 10/3/2023

- Playlists (Queuing of sounds/songs) has only a partial implementation for SDK7 in the Model, as of 10/3/2023, no Views can yet control or display them

- In Preview mode, you must touch the scene before already-playing audio will be heard. This is not needed in deployed scenes.

- The latest version of the SDK7 scene code is currently stored in the scene-base repository, not in a separate repo of its own.

Updates needed:

- ☐ Review/update the role of each component
- ☐ Update the data flow, even types, etc.
- ☐ CFSoundPlayer should not be described as a View.
    - ☐ And it is created by Controller.initialize(lociEntities)
- ☐ Document: that a scene can only hear one AudioStream at a time, even if there are multiple players/entities. AudioSources can be multiple, though each needs its own entity