C++ Media Management

by Michael LeGore

Procedure

In order to allow for simple addition of components and features, I am designing a system to be modular and completely extendable. My creating an interface for C++ plugins and/or ruby scripts, I plan on allowing for post-compile time additions to the core of the program. For example, if a user wanted to be able to interface with a specific soundboard, that user could write a ruby script that implements certain methods (like play, pause, stop, etc.) and specifies that it is a driver for a sound source, the program would be able to import that resource and use it to play back sound over the sound system.

Extensibility

The main goal of this project is to experiment with the extensibilty of a program through the use of Ruby scripting, dynamically loaded extensions, and well thought out design descisions. For example, by generalizing all media in to classes such as a Resource class of an Event class, I can extend with different forms of media, without needing to revise the core of the program, and without hardcoding anything.
An example of the preliminary Event class with basic functions that allows the timeline to interface with it:

```
#define EVENT_INSTANT 0
#define EVENT_DURATION 1
#define EVENT_UPDATING 2
//enum event_type {EVENT_INSTANT,EVENT_DURATION,EVENT_UPDATE};
class Event
{
        long long length;
        event_type type;
        virtual void play() = 0;
        virtual void pause() = 0;
        virtual void stop() = 0;
        virtual void seek(long long ms) = 0;
        virtual void forward(long long ms) = 0;
        virtual void reverse(long long ms) = 0;
        void advanceTime(long long ms);
};
```

Possible Features

Hardware Interface

In order to interface with many different types of hardware, I will be creating a simple protocol that allows for hardware (ie. Microcontrollers) to expose methods and variables to the Media application. By declaring what methods a piece of hardware has at time of serial handshake, the user could then add those method calls to the timeline of events.

```
//Spec for the preformance hardware serial interface psudocode
// » means read from serial « means write to serial
// all commands end with ;
Establishing contact:
while » != contact
«polling
end
//Identify the device
»devicetype=[deviceType];numproperties=[number of properties];[property1=,prop-
erty2=...]>;][numactions=;][action1([param1,param2,param3]):[retvalue,retvalue,ret-
value];action2...]
//Run routine action1 with params 4 5 6
»action1(4,5,6);
```