

# Experience Base Class

Reflections Project

by Frank Cohen, [fcohen@starlingwatch.com](mailto:fcohen@starlingwatch.com)

Distributed under a GPL v3 open-source license.

<https://github.com/frankcohen/ReflectionsOS>

*My prompt to ChatGPT 4:* I am writing c++ code in Arduino IDE 2.3.2 for ESP32-S3. I have a central dispatch object called Inveigle. It is implemented as a .cpp and .h files. It randomly picks 1 of 5 experiences. Each experience has a setup, run, and teardown part state, with the conclusion of one state starting the next, and when the teardown part finishes, the Inveigle randomly picks another experience. Each experience can be overridden at any time in the setup and run parts, in which case the teardown part runs. I will add additional experiences over time. I need the code to have an abstract base class definition to implement each experience.

## ChatGPT response:

To create a robust and extensible system for handling experiences in your Inveigle object, you can use an abstract base class for the experiences. This will ensure that each experience has a consistent interface for setup, run, and teardown operations. Here's a possible implementation of the system:

## Experience Base Class (Abstract)

First, define an abstract base class `Experience` in `Experience.h`. This class will declare the virtual methods for setup, run, and teardown, which must be implemented by any derived class.

```
cpp
// Experience.h
#ifndef EXPERIENCE_H
#define EXPERIENCE_H

class Experience {
public:
    virtual ~Experience() {}

    virtual void setup() = 0;
    virtual void run() = 0;
    virtual void teardown() = 0;
};
```

```
#endif // EXPERIENCE_H
```

## Example Derived Experience Class

Next, create a concrete implementation of the `Experience` class. For example, `Experience1`.

cpp

```
// Experience1.h
#ifndef EXPERIENCE1_H
#define EXPERIENCE1_H

#include "Experience.h"

class Experience1 : public Experience {
public:
    void setup() override;
    void run() override;
    void teardown() override;
};

#endif // EXPERIENCE1_H

// Experience1.cpp
#include "Experience1.h"
#include <Arduino.h>

void Experience1::setup() {
    // Setup code for Experience1
    Serial.println("Experience1 setup");
}

void Experience1::run() {
    // Run code for Experience1
    Serial.println("Experience1 run");
}
```

```

void Experience1::teardown() {
    // Teardown code for Experience1
    Serial.println("Experience1 teardown");
}

```

## Central Dispatch Object (Inveigle)

Now, create the `Inveigle` class that manages the experiences. It will randomly pick an experience, handle state transitions, and allow for experiences to be overridden.

```

cpp
// Inveigle.h
#ifndef INVEIGLE_H
#define INVEIGLE_H

#include "Experience.h"
#include <vector>

class Inveigle {
public:
    Inveigle();
    void setup();
    void run();
    void overrideExperience(Experience* newExperience);

private:
    void pickRandomExperience();
    void transitionToNextState();

    std::vector<Experience*> experiences;
    Experience* currentExperience;
    enum State { SETUP, RUN, TEARDOWN } currentState;
};

#endif // INVEIGLE_H

// Inveigle.cpp
#include "Inveigle.h"
#include "Experience1.h"

```

```

// Include other experiences as needed
#include <Arduino.h>

Inveigle::Inveigle() : currentExperience(nullptr),
currentState(SETUP) {
    // Add instances of each experience to the vector
    experiences.push_back(new Experience1());
    // Add other experiences here
}

void Inveigle::setup() {
    pickRandomExperience();
    currentExperience->setup();
    currentState = RUN;
}

void Inveigle::run() {
    if (currentState == RUN) {
        currentExperience->run();
        // Check for override condition here
    } else if (currentState == TEARDOWN) {
        currentExperience->teardown();
        pickRandomExperience();
        currentExperience->setup();
        currentState = RUN;
    }
}

void Inveigle::overrideExperience(Experience*
newExperience) {
    currentExperience->teardown();
    delete currentExperience;
    currentExperience = newExperience;
    currentExperience->setup();
    currentState = RUN;
}

void Inveigle::pickRandomExperience() {
    if (currentExperience) {
        delete currentExperience;
    }
}

```

```

    }
    int randomIndex = random(experiences.size());
    currentExperience = experiences[randomIndex];
}

void Inveigle::transitionToNextState() {
    switch (currentState) {
        case SETUP:
            currentExperience->setup();
            currentState = RUN;
            break;
        case RUN:
            currentExperience->run();
            currentState = TEARDOWN;
            break;
        case TEARDOWN:
            currentExperience->teardown();
            pickRandomExperience();
            currentExperience->setup();
            currentState = RUN;
            break;
    }
}

```

## Usage in the Arduino Sketch

Finally, use the `Inveigle` object in your Arduino sketch.

```

cpp
// sketch.ino
#include <Arduino.h>
#include "Inveigle.h"

Inveigle inveigle;

void setup() {
    Serial.begin(115200);
    inveigle.setup();
}

```

```

}

void loop() {
    inveigle.run();
    // Add delay or other logic as needed
    delay(1000);
}

```

This setup ensures that each experience can be managed uniformly through the **Experience** base class, and the **Inveigle** class handles the lifecycle and random selection of experiences. You can add additional experiences by creating new derived classes from **Experience** and adding them to the **Inveigle**'s experience list.

**My ask of ChatGPT:**

*Please give me an example of creating Experience 6. In this experience the Setup makes a call to video.cpp to start running a mpeg video. You do not need to show me video.cpp. The interface is video::startVideo(String videoName). The Teardown makes a call to video::stopVideo();*

ChatGPT response

Certainly! Below is an example of how you can create **Experience6**, which will utilize the **video::startVideo** and **video::stopVideo** functions as described.

## Experience6 Header

Create a new header file **Experience6.h**:

```

cpp
// Experience6.h
#ifndef EXPERIENCE6_H
#define EXPERIENCE6_H

#include "Experience.h"
#include <Arduino.h>

class Experience6 : public Experience {
public:
    void setup() override;
    void run() override;
    void teardown() override;
};

```

```
#endif // EXPERIENCE6_H
```

## Experience6 Implementation

Create the corresponding implementation file `Experience6.cpp`:

```
cpp
```

```
// Experience6.cpp
#include "Experience6.h"
#include "video.h" // Assuming video.h is where
video::startVideo and video::stopVideo are declared

void Experience6::setup() {
    // Setup code for Experience6
    Serial.println("Experience6 setup");
    video::startVideo("example.mpeg"); // Replace
    "example.mpeg" with your video name
}

void Experience6::run() {
    // Run code for Experience6
    Serial.println("Experience6 run");
    // Add any additional run logic here
}

void Experience6::teardown() {
    // Teardown code for Experience6
    Serial.println("Experience6 teardown");
    video::stopVideo();
}
```

## Including Experience6 in Inveigle

Modify the `Inveigle` class to include `Experience6`.

### Update the Header

Update `Inveigle.h` to forward-declare `Experience6`:

```

cpp
// Inveigle.h
#ifndef INVEIGLE_H
#define INVEIGLE_H

#include "Experience.h"
#include <vector>

class Inveigle {
public:
    Inveigle();
    void setup();
    void run();
    void overrideExperience(Experience* newExperience);

private:
    void pickRandomExperience();
    void transitionToNextState();

    std::vector<Experience*> experiences;
    Experience* currentExperience;
    enum State { SETUP, RUN, TEARDOWN } currentState;
};

#endif // INVEIGLE_H

```

## Update the Implementation

Update `Inveigle.cpp` to include and initialize `Experience6`:

```

cpp
// Inveigle.cpp
#include "Inveigle.h"
#include "Experience1.h"
#include "Experience6.h"
// Include other experiences as needed
#include <Arduino.h>

```



```

Inveigle::Inveigle() : currentExperience(nullptr),
currentState(SETUP) {
    // Add instances of each experience to the vector
    experiences.push_back(new Experience1());
    experiences.push_back(new Experience6());
    // Add other experiences here
}

void Inveigle::setup() {
    pickRandomExperience();
    currentExperience->setup();
    currentState = RUN;
}

void Inveigle::run() {
    if (currentState == RUN) {
        currentExperience->run();
        // Check for override condition here
    } else if (currentState == TEARDOWN) {
        currentExperience->teardown();
        pickRandomExperience();
        currentExperience->setup();
        currentState = RUN;
    }
}

void Inveigle::overrideExperience(Experience*
newExperience) {
    currentExperience->teardown();
    delete currentExperience;
    currentExperience = newExperience;
    currentExperience->setup();
    currentState = RUN;
}

void Inveigle::pickRandomExperience() {
    if (currentExperience) {
        delete currentExperience;
    }
    int randomIndex = random(experiences.size());

```

```

        currentExperience = experiences[randomIndex];
    }

void Inveigle::transitionToNextState() {
    switch (currentState) {
        case SETUP:
            currentExperience->setup();
            currentState = RUN;
            break;
        case RUN:
            currentExperience->run();
            currentState = TEARDOWN;
            break;
        case TEARDOWN:
            currentExperience->teardown();
            pickRandomExperience();
            currentExperience->setup();
            currentState = RUN;
            break;
    }
}

```

## Updating the Arduino Sketch

Ensure that the sketch includes the `Inveigle` header and properly initializes the system:

```

cpp
// sketch.ino
#include <Arduino.h>
#include "Inveigle.h"

Inveigle inveigle;

void setup() {
    Serial.begin(115200);
    inveigle.setup();
}

```

```
void loop() {  
    inveigle.run();  
    // Add delay or other logic as needed  
    delay(1000);  
}
```

With these changes, **Experience6** will start playing a video when it is set up, and stop the video during teardown. You can continue adding more experiences by following the same pattern.