Subject: Re: Architectural and C++/Upp questions
Posted by Klugier on Mon, 28 Sep 2020 20:08:53 GMT
View Forum Message <> Reply to Message

Hello Xemuth,

I am glad you ask :) I like threads like this!

---
1. You could create API similar to Unity. You will need two following classes:
- SceneManager (Context probably not very approrpiate name)
- Scene

In context of SceneManager (context) you could use String as in Unity to identify specific scene. The next think I would change is the use of reference. I would opt for pointer, because right now you can not detect errors in your API. Internally you could store it on heap and only in return situation just & for pointer passing. So, the code would look like this:

Scene* SceneManager::CreateScene(const String& id); // In case of error nullptr is returned - you could use other constructs here as well (optional or upp concepts - read Core tutorial for tips). Fine to return Scene here.
bool SceneManager::RemoveScene(const String& id); // bool is fine for error handling
bool SceneManager::HasScene(const String& id); // nice addition to remove
Scene* GetScene(const String& id); // in context of error - nullptr is returned

---

2.

```
class Service{
 public:
  Service(); // <- For one varialbe not need (I do not see the code, so I may not understand
  Service(Service&& service); // <- Not need (break the rule of 5)
  Service(const Service& service); // <- Not need (breaks the rule of 5)
  virtual ~Service(); // <- fine not need for implementation - just mark it = default;

		// Do you plan to support concrete message set then replacing message with enum make sense here...
		virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args = Vector<Value>());
```

Backing to static_cast proble. Why not use template method instead and do the cast here - it will be hidden for the user:

```
auto* service = ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
```

// Pointer for error handling - nullptr in case of error. I suggest using dynamic_cast here. However, you can not distinguish error type here - whenever it fail on dynamic_cast or fail on finding service name... You could add HasService method that will return bool...

Reference:
- https://en.cppreference.com/w/cpp/language/rule_of_three

---

3. Vector<Value> - seems like task for optional not vector. In c++17 std::optional should do the job. In the world of U++ you could return One<Value>. For more information please read - Core tutorial (3.11 One).

Anyway in the example you show - bool should be enough:

```
// override - nice to add this and consider channing the name of the method to OnMessageReceive
// in c++11 = Vector<Value>() could be simplify to {}
virtual bool OnMessageReceive(const String& message,const Vector<Value>& args = {}) override{
 if(message.IsEqual("AddQueue") && args.GetCount() >= 3){
  try{
     AddDrawToQueue(args[0].Get<String>(),args[1].Get<String>(), args[2].Get<String>());
     return true;
  }catch(Exc& exception){ // Don't like exceptions here :)
     Cout() << exception << EOL;
  }
 }
       return false;
}
```

Generally speaking I do not like exceptions in C++. I enjoy c++17 approach that allows to unpack tuple in one line:

```
auto [service, error] = ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
if (error) {
   // Log error... etc...
   return;
}

// Make further processing with service...
```

This is exactly the same error handling available in golang. The difference is that it is the only option there :) In your case simple *service should be enough. Alternatively you could use exceptions...

Reference:
- https://en.cppreference.com/w/cpp/language/structured_bindin g

Klugier
