Subject: Re: Architectural and C++/Upp questions
Posted by mirek on Tue, 29 Sep 2020 07:07:35 GMT
View Forum Message <> Reply to Message

Klugier wrote on Mon, 28 September 2020 22:08Hello Xemuth,
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.

Never use pointer if you can use reference. Pointers in general produce heap bugs.

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...

Do not repeat yourself:

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

That said, Service concept itself feels bit like overengineering. Do you really need that level of abstraction? Cannot you just have RenderOpenGL class or something?

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

Exceptions in C++ are superior when used correctly.

Anyway, in this context, you should always ask: Is the error handling even worth it? I mean, can the application meaningfully continue after error? Can it do something reasonable with the fact? If not, then just LOG the error and either end the application with message, or fix it so that it can continue like nothing happened. Set resetable error flag. Maybe add Event<> WhenError so that client code can throw the exception if it needs to.

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

```
   return;
}
```

This is terrible idea (yes I suppose Rust and Golang are going this direction but they have additional language constructs to deal with it. Still makes me doubtful about them). This approach will infest your code with tons of meaningless errorhandling code.

I guess it is the time I have started working on that "U++ design philosophy" article....

Mirek
