Interactive audio plug-in development using the Wwise SDK

Joel Robichaud
ADC 2018

Let’s start with some terminology

“**Interactive multimedia**, any computer-delivered electronic system that allows the user to control, combine, and manipulate different types of media, such as text, sound, video, computer graphics, and animation.

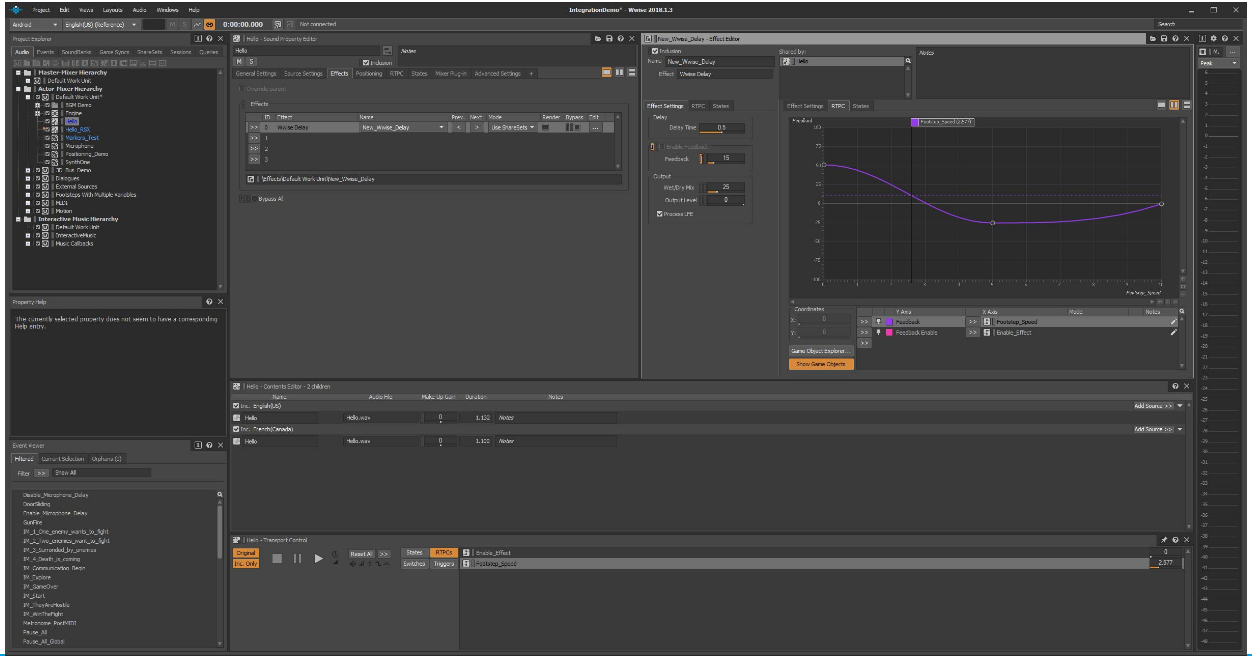
[…]

Interactive multimedia shift the user’s role from observer to participant and are considered the next generation of electronic information systems.”

— *Encyclopaedia Britannica*

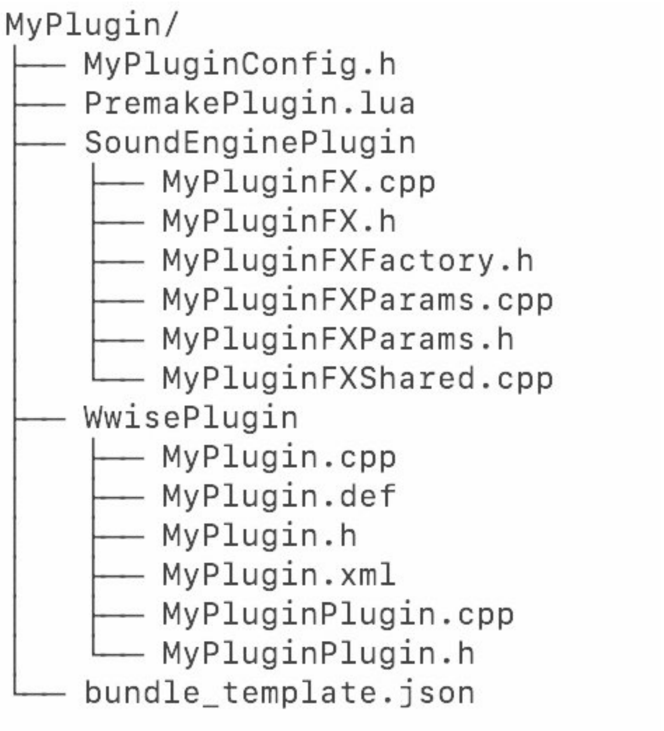
How Wwise fits in the picture

- Wwise is an interactive audio solution with a focus towards video games
- It supports audio plug-ins, but not quite in the same way that DAWs typically do



Anatomy of a Wwise plug-in

- Consists of two parts (and some config files)
- One part is executed in the Sound Engine to produce or modify the input sound based on settings defined by the user in Wwise
- Another part runs in Wwise and contains the UI that the user can use to modify the plug-in's properties

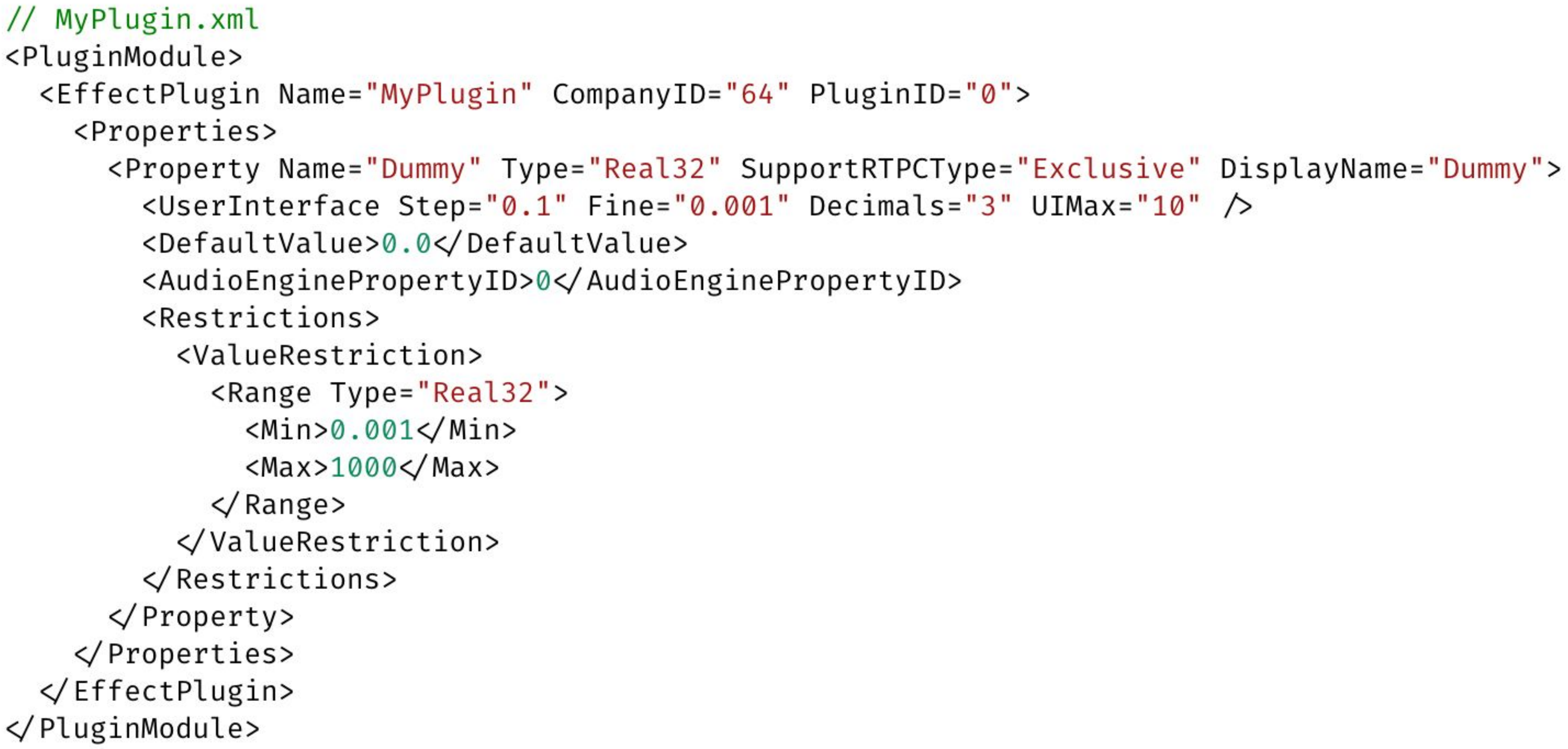


```
MyPlugin/
├── MyPluginConfig.h
├── PremakePlugin.lua
├── SoundEnginePlugin
│   ├── MyPluginFX.cpp
│   ├── MyPluginFX.h
│   ├── MyPluginFXFactory.h
│   ├── MyPluginFXParams.cpp
│   ├── MyPluginFXParams.h
│   └── MyPluginFXShared.cpp
├── WwisePlugin
│   ├── MyPlugin.cpp
│   ├── MyPlugin.def
│   ├── MyPlugin.h
│   ├── MyPlugin.xml
│   ├── MyPluginPlugin.cpp
│   └── MyPluginPlugin.h
└── bundle_template.json
```

Authoring plug-ins

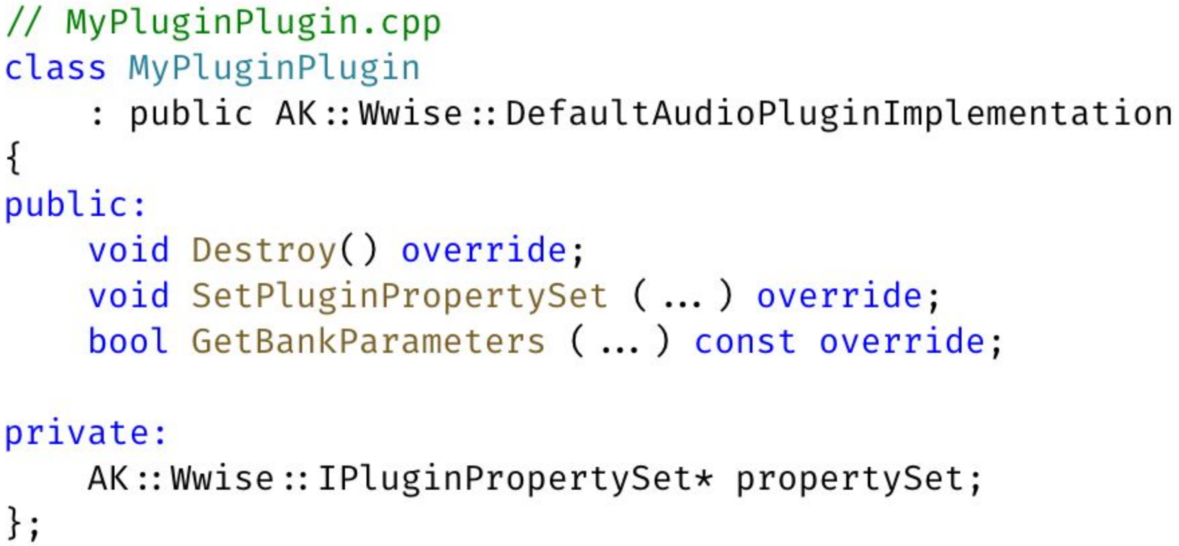
- Once built, they consist of an XML descriptor file and a DLL file built for Windows
- They statically link with the Sound Engine plug-in to be able to provide real-time editing capabilities
- Common editing operations such as persistence and undo / redo support are automatically handled

Authoring plug-ins (cont.)



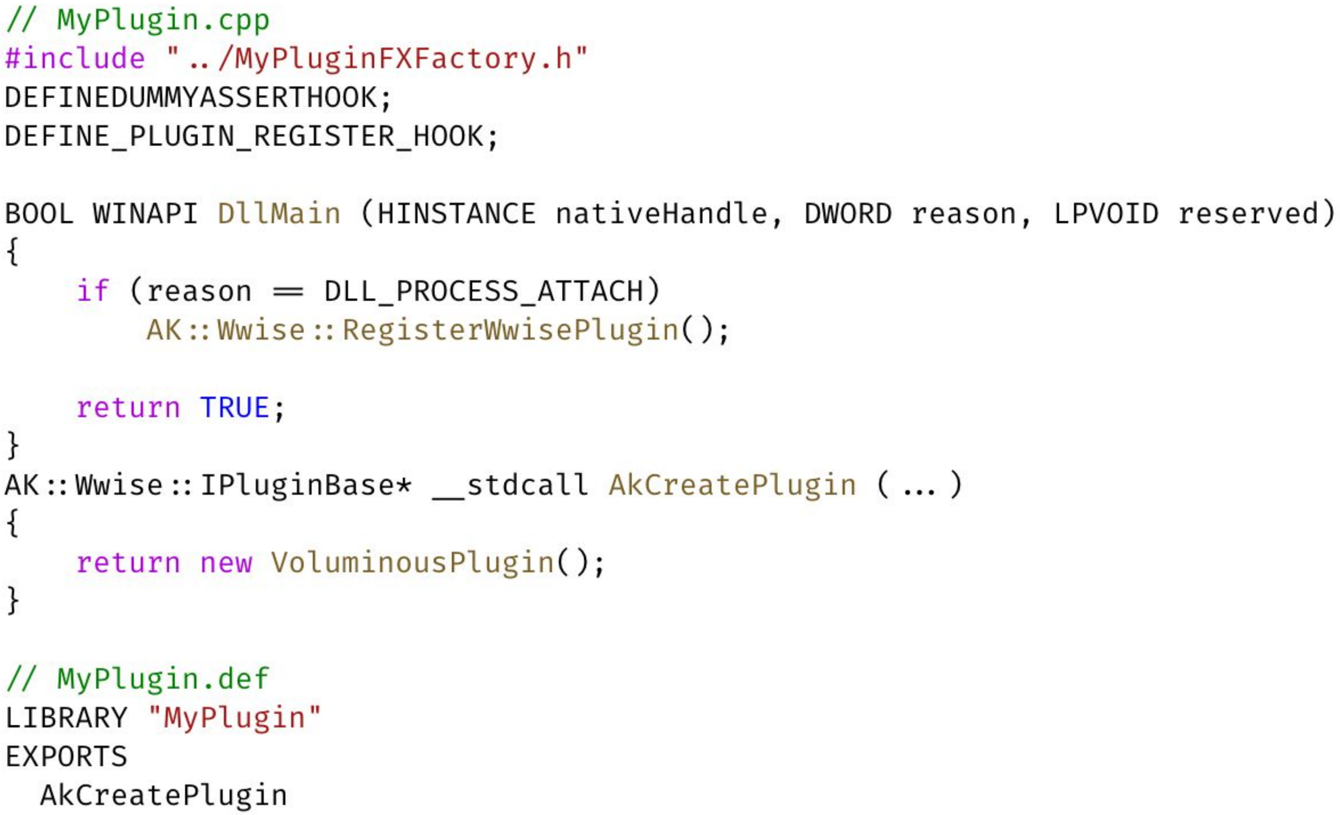
```
// MyPlugin.xml
<PluginModule>
  <EffectPlugin Name="MyPlugin" CompanyID="64" PluginID="0">
    <Properties>
      <Property Name="Dummy" Type="Real32" SupportRTPCType="Exclusive" DisplayName="Dummy">
        <UserInterface Step="0.1" Fine="0.001" Decimals="3" UIMax="10" />
        <DefaultValue>0.0</DefaultValue>
        <AudioEnginePropertyID>0</AudioEnginePropertyID>
        <Restrictions>
          <ValueRestriction>
            <Range Type="Real32">
              <Min>0.001</Min>
              <Max>1000</Max>
            </Range>
          </ValueRestriction>
        </Restrictions>
      </Property>
    </Properties>
  </EffectPlugin>
</PluginModule>
```

Authoring plug-ins (cont.)



```cpp
// MyPluginPlugin.cpp
class MyPluginPlugin
    : public AK::Wwise::DefaultAudioPluginImplementation
{
public:
    void Destroy() override;
    void SetPluginPropertySet ( … ) override;
    bool GetBankParameters ( … ) const override;

private:
    AK::Wwise::IPluginPropertySet* propertySet;
};
```

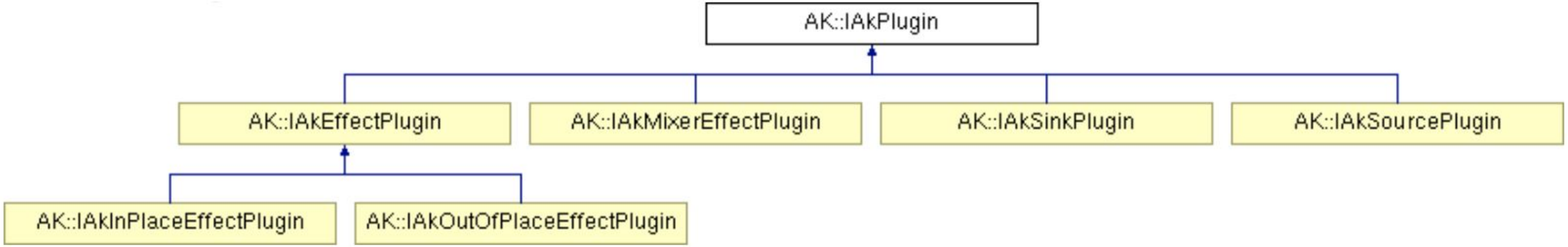
Authoring plug-ins (cont.)



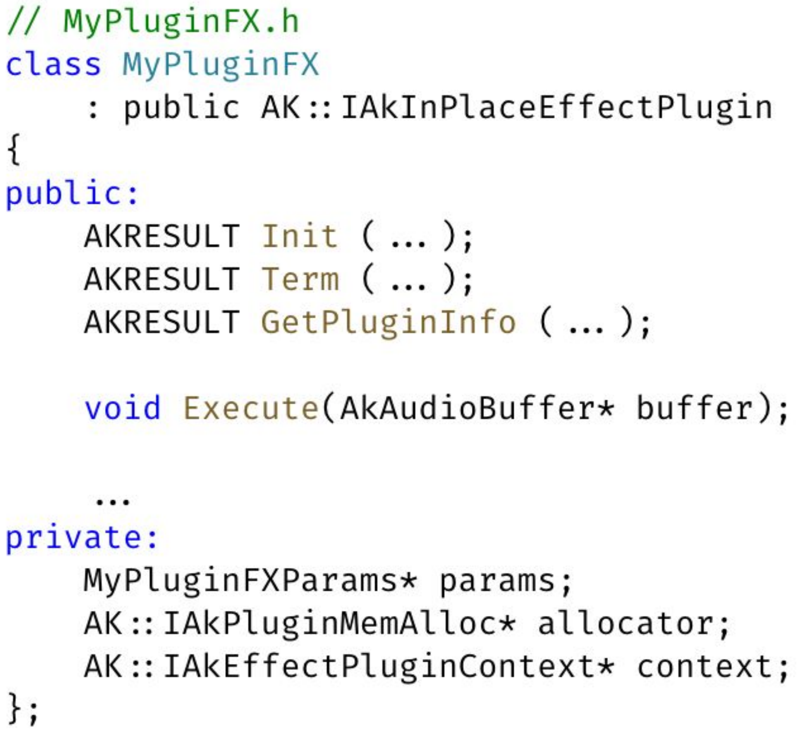
```
// MyPlugin.cpp
#include "../MyPluginFXFactory.h"
DEFINEDUMMYASSERTHOOK;
DEFINE_PLUGIN_REGISTER_HOOK;

BOOL WINAPI DllMain (HINSTANCE nativeHandle, DWORD reason, LPVOID reserved)
{
    if (reason == DLL_PROCESS_ATTACH)
        AK::Wwise::RegisterWwisePlugin();

    return TRUE;
}
AK::Wwise::IPluginBase* __stdcall AkCreatePlugin ( ... )
{
    return new VoluminousPlugin();
}

// MyPlugin.def
LIBRARY "MyPlugin"
EXPORTS
  AkCreatePlugin
```

Sound Engine plug-ins

- Once built, they consist of a static and (optionally) a shared library
- There are different kinds of Sound Engine plug-ins you can create, depending on what you want them to do (effect, source, sink or mixer)

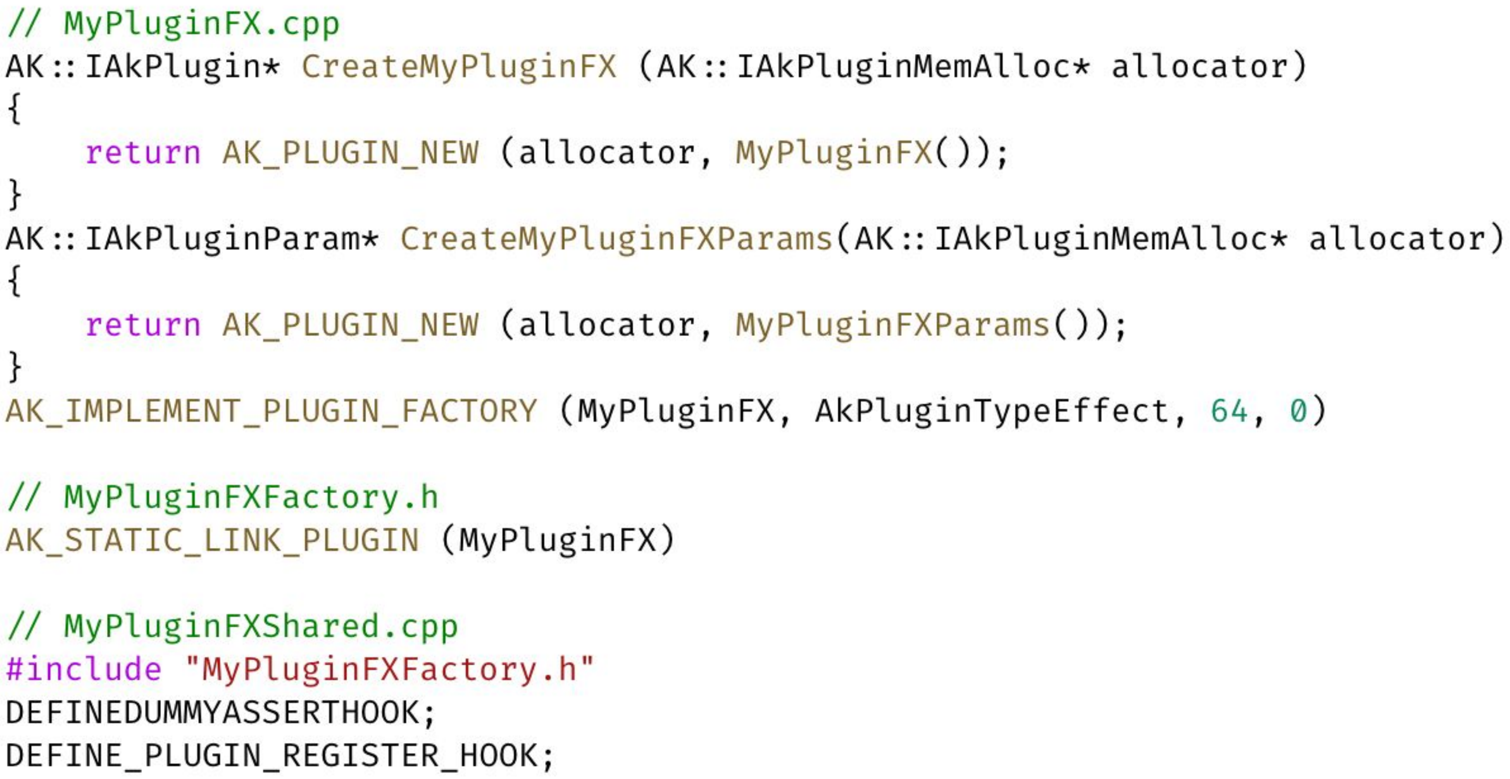


Sound Engine plug-ins (cont.)



```
// MyPluginFX.h
class MyPluginFX
    : public AK::IAkInPlaceEffectPlugin
{
public:
    AKRESULT Init ( ... );
    AKRESULT Term ( ... );
    AKRESULT GetPluginInfo ( ... );

    void Execute(AkAudioBuffer* buffer);

    ...
private:
    MyPluginFXParams* params;
    AK::IAkPluginMemAlloc* allocator;
    AK::IAkEffectPluginContext* context;
};
```

Sound Engine plug-ins (cont.)



```
// MyPluginFX.cpp
AK::IAkPlugin* CreateMyPluginFX (AK::IAkPluginMemAlloc* allocator)
{
    return AK_PLUGIN_NEW (allocator, MyPluginFX());
}
AK::IAkPluginParam* CreateMyPluginFXParams(AK::IAkPluginMemAlloc* allocator)
{
    return AK_PLUGIN_NEW (allocator, MyPluginFXParams());
}
AK_IMPLEMENT_PLUGIN_FACTORY (MyPluginFX, AkPluginTypeEffect, 64, 0)

// MyPluginFXFactory.h
AK_STATIC_LINK_PLUGIN (MyPluginFX)

// MyPluginFXShared.cpp
#include "MyPluginFXFactory.h"
DEFINEDUMMYASSERTHOOK;
DEFINE_PLUGIN_REGISTER_HOOK;
```

Communication is key

- All the information that is required at runtime is packaged into SoundBanks, which contain audio sources, audio object information, events, prefetch data for streaming, and streaming references
- Audio sources are transcoded to optimized formats (Vorbis, AAC, ADPCM, PCM) depending on which type of compression is required and are decoded at runtime
- Data is tunneled directly to the Sound Engine when doing real-time editing in Wwise

Memory allocation will fail

- Handling memory allocation failure in Sound Engine plug-ins is mandatory since it cannot be assumed that memory will be readily available, especially on game consoles
- Use the allocators provided by the Sound Engine, do not use new or C++ exceptions
- Don’t allocate more memory than you need to and be very conscious of the footprint of your plug-in — the same mindset should be applied to the CPU usage of your plug-in (use SIMD whenever possible)

Portability doesn’t even begin to describe it

- There are quite a lot of platforms to support depending on which version of the Wwise SDK you want to target
- You don’t *have* to support all of them (some of them require a license), but keep in mind that it may impact the popularity of your plug-in if you intend to make it public
- Making an Authoring-only plug-in may be a good alternative since it can then be used to like a VST to pre-render effects

androidtv
XBOX 360
WiiU
NINTENDO SWITCH
magic leap
3DS
PS3
XBOX ONE
tvOS
iOS
X
PS4
PSVITA

A pipeline for building Wwise plug-ins

- With so many target platforms, we needed to provide better development tools to plug-in developers
- We decided to rewrite a subset of our internal build pipeline in python and give it to plug-in developers (along with our custom build of premake and the scripts that go with it)
- While we were at it, we also decided to write a plug-in generator to accelerate the process of creating new plug-ins

A pipeline for building Wwise plug-ins (cont.)

Generate

A project base is generated for a given plug-in type (effet, source, sink or mixer)

Premake

The solutions required to build a given platform are generated using Premake from a PremakePlugin.lua file located at the root of the project

Build

The premade solutions are built using the development tools for that platform

Package

The resulting build artifacts are then packaged into tar.xz archives (one archive per platform)

Bundle

A JSON metadata file is generated for use with the Wwise Launcher

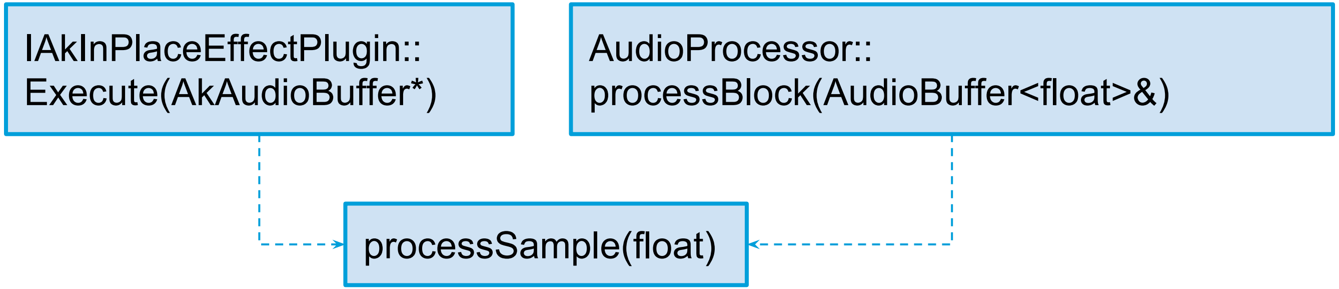
Live demo - plug-in development tools

Bridging the gap

- How can we adapt an existing JUCE plug-in to also work with Wwise?
- We mostly care about reusing the user interface code with as little modifications as possible to make an Authoring plug-in
- Thankfully, building JUCE along with Wwise is as simple as omitting to build the plug-in clients library code

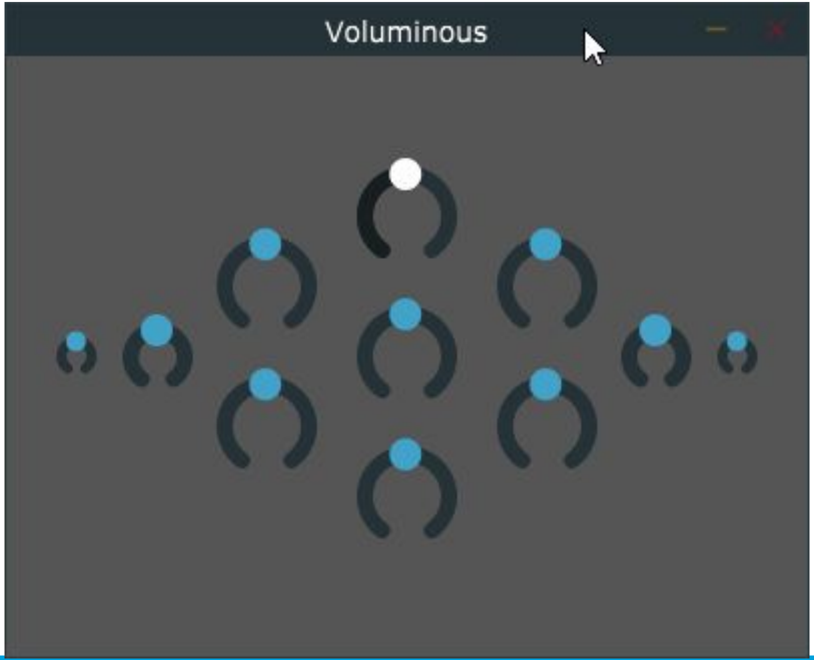
Bridging the gap (cont.)

- The DSP code most likely won’t be reusable since not all of the Sound Engine platforms support the C++ feature-set required to build JUCE
- You could theoretically use the same DSP code for both types of plug-in if it was low-level enough (left as an exercise to the reader)



Case study

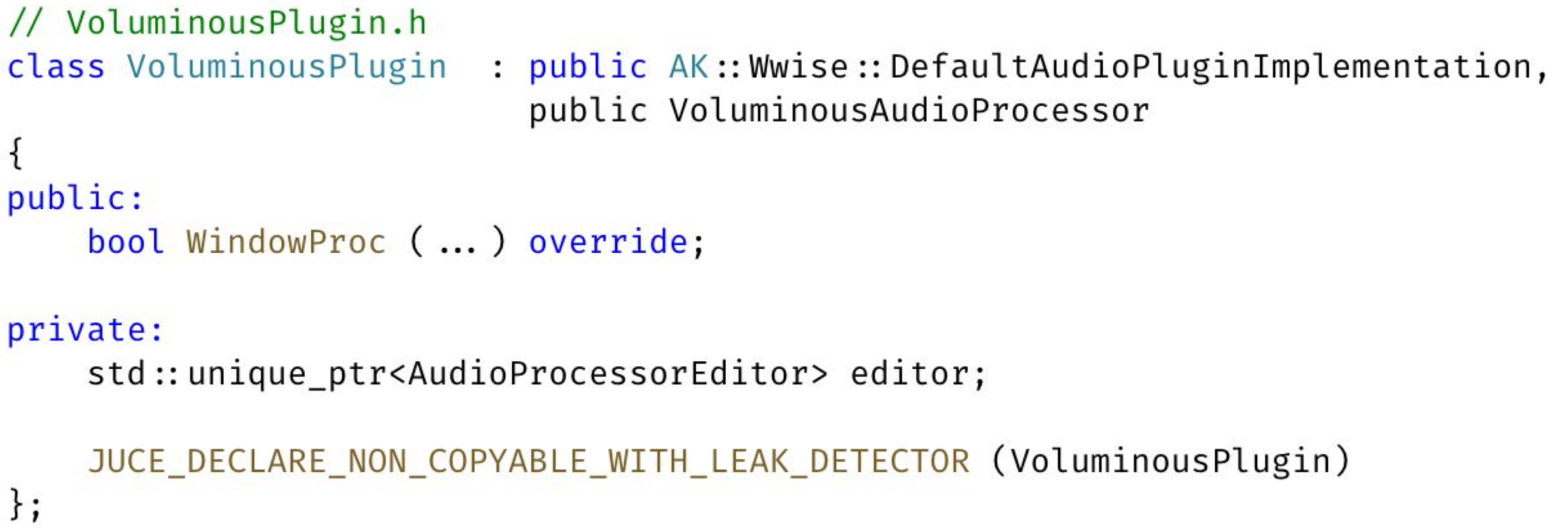
- **Voluminous**, an unnecessarily large volume control plug-in made with JUCE



You’re the host (and the AudioProcessor)

- The Authoring plug-in should manage the lifetime of the AudioProcessorEditor and should do so in the WindowProc when handling the WM_INITDIALOG and WM_DESTROY messages
- It should also inherit from your AudioProcessor implementation to facilitate the creation of the AudioProcessorEditor
- One downside with this approach is that it will most likely lead to dead code due to most of the functions of the AudioProcessor never being called

You’re the host (cont.)



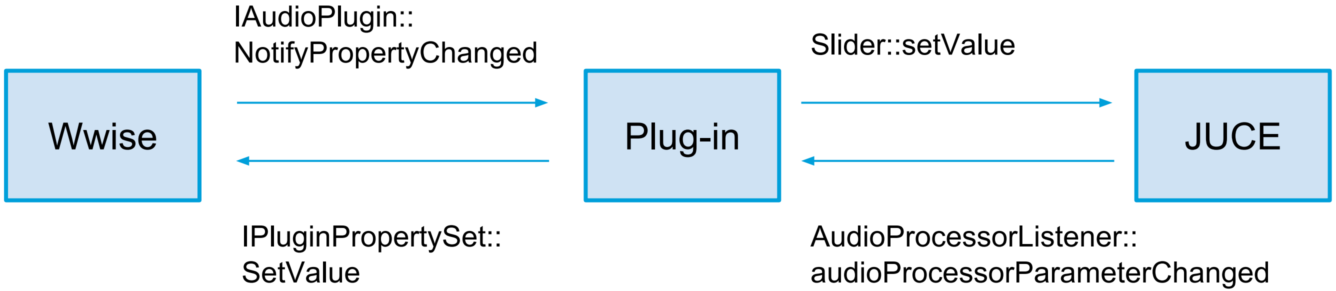
```
// VoluminousPlugin.h
class VoluminousPlugin  : public AK::Wwise::DefaultAudioPluginImplementation,
                          public VoluminousAudioProcessor
{
public:
    bool WindowProc ( ... ) override;

private:
    std::unique_ptr<AudioProcessorEditor> editor;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (VoluminousPlugin)
};
```

You’re the host (cont.)

```cpp
// VoluminousPlugin.cpp
bool VoluminousPlugin::WindowProc (AK::Wwise::IAudioPlugin::eDialog dialog,
                                    HWND nativeHandle, UINT message, WPARAM wparam,
                                    LPARAM lparam, LRESULT& result)
{
    switch (message)
    {
    case WM_INITDIALOG:
        editor.reset (createEditorIfNeeded());
        editor->setOpaque (true);
        editor->setVisible (true);
        editor->addToDesktop (0, nativeHandle);
    break;
    case WM_DESTROY:
        editor->removeFromDesktop();
        editorBeingDeleted (editor.get());
        editor = nullptr;
    break;
    }
     ...
}
```

Parameter proxying

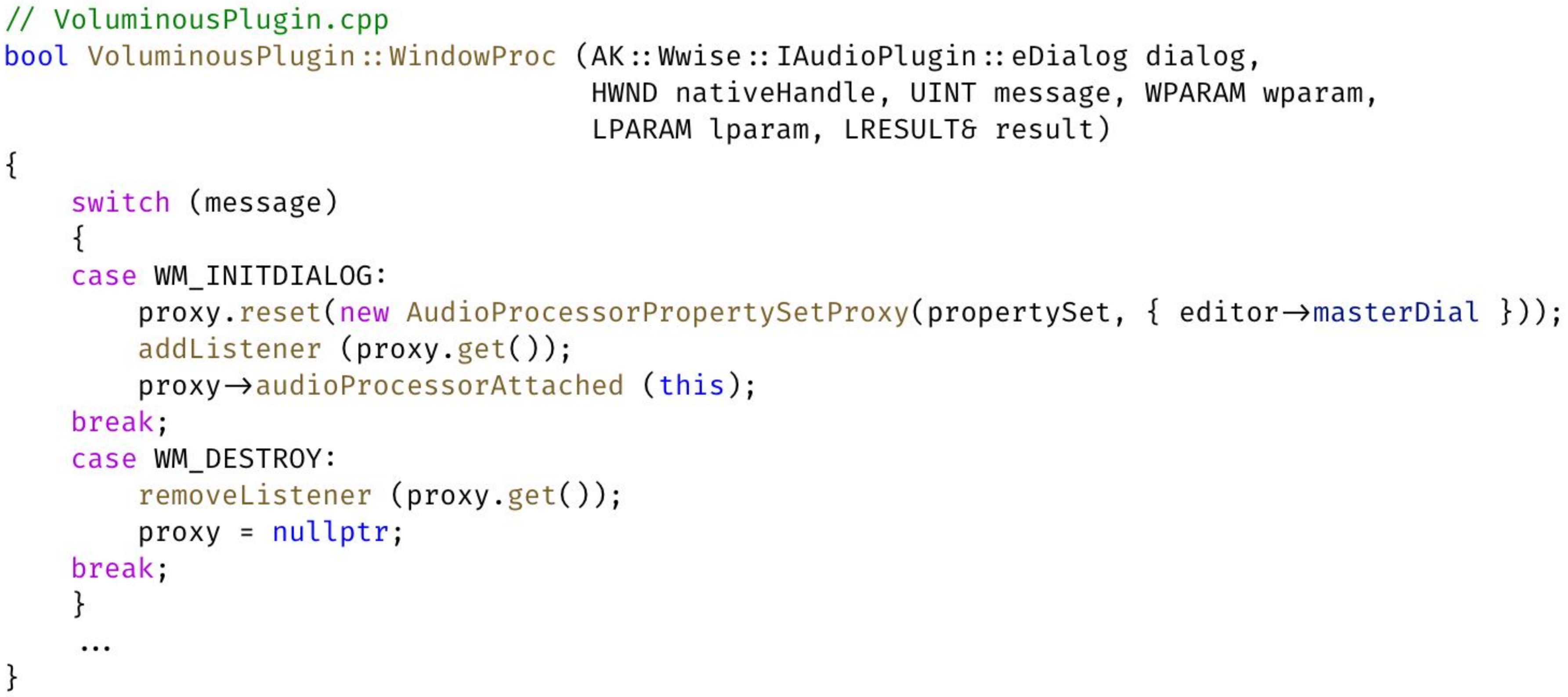
- We have two data models that we need to keep in sync
  - Subclass AudioProcessorListener to listen to parameter changes in the AudioProcessor and update the corresponding property in the PluginPropertySet
  - Update the JUCE parameters when NotifyPropertyChanged is called (mostly used for undo / redo support)
- Stack overflows due to notification ping-pongs are avoided since both Wwise and JUCE do not send notifications when values haven't changed



Parameter proxying (cont.)

```
// VoluminousPlugin.h
class VoluminousPlugin  : public AK::Wwise::DefaultAudioPluginImplementation,
                          public VoluminousAudioProcessor
{
public:
    void SetPluginPropertySet (AK::Wwise::IPluginPropertySet* propertySet) override { ... }
    void NotifyPropertyChanged ( ... ) override;

private:
    AK::Wwise::IPluginPropertySet* propertySet;
    std::unique_ptr<AudioProcessorPropertySetProxy> proxy;
};
```

Parameter proxying (cont.)

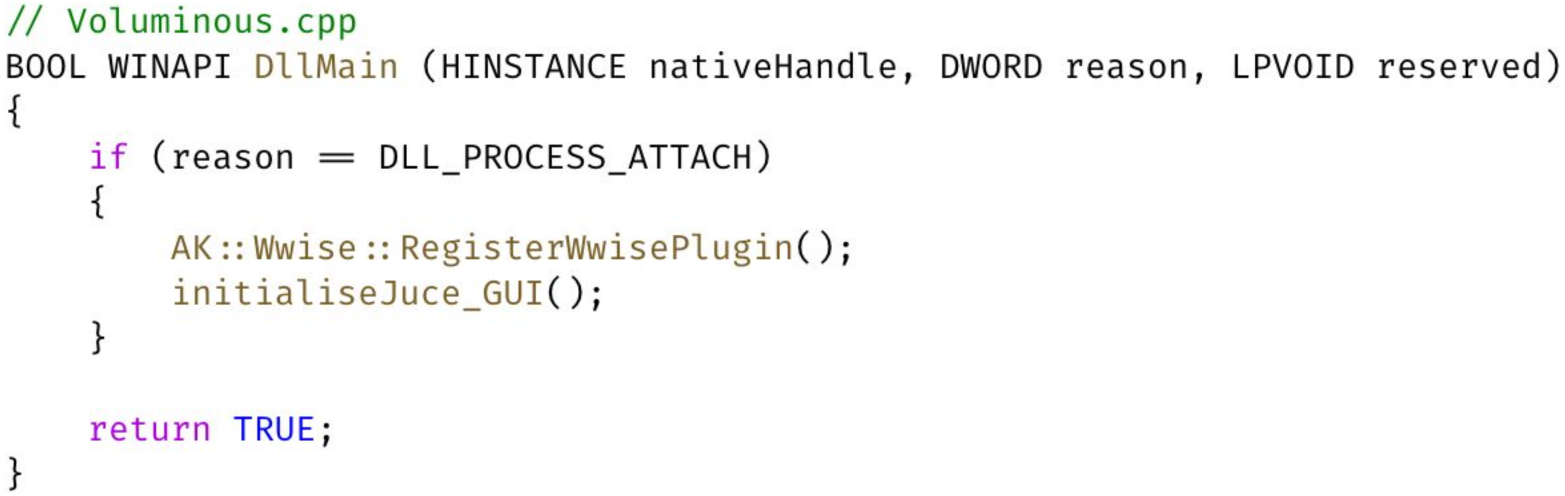


```cpp
// VoluminousPlugin.cpp
bool VoluminousPlugin::WindowProc (AK::Wwise::IAudioPlugin::eDialog dialog,
                                   HWND nativeHandle, UINT message, WPARAM wparam,
                                   LPARAM lparam, LRESULT& result)
{
    switch (message)
    {
    case WM_INITDIALOG:
        proxy.reset(new AudioProcessorPropertySetProxy(propertySet, { editor->masterDial }));
        addListener (proxy.get());
        proxy->audioProcessorAttached (this);
    break;
    case WM_DESTROY:
        removeListener (proxy.get());
        proxy = nullptr;
    break;
    }
    ...
}
```

Graceful initialization and shutdown

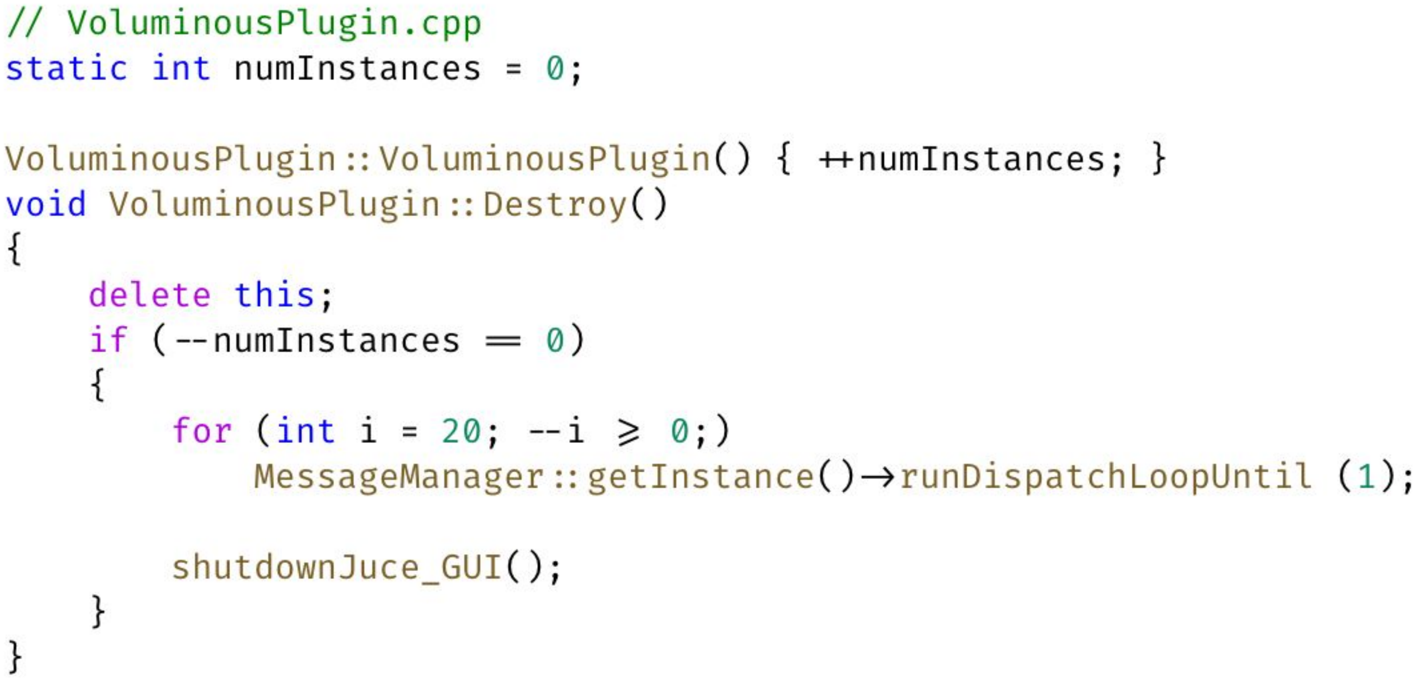
- The GUI services provided by JUCE need to be properly initialized and shut down using initialiseJuce_GUI and shutdownJuce_GUI to avoid triggering the memory leak detectors
- Even when doing so, you may still leak messages if you don’t empty the message queue before exiting
- None of this actually matters since the resources will be reclaimed by the operating system immediately after

Graceful initialization and shutdown (cont.)



```
// Voluminous.cpp
BOOL WINAPI DllMain (HINSTANCE nativeHandle, DWORD reason, LPVOID reserved)
{
    if (reason == DLL_PROCESS_ATTACH)
    {
        AK::Wwise::RegisterWwisePlugin();
        initialiseJuce_GUI();
    }

    return TRUE;
}
```

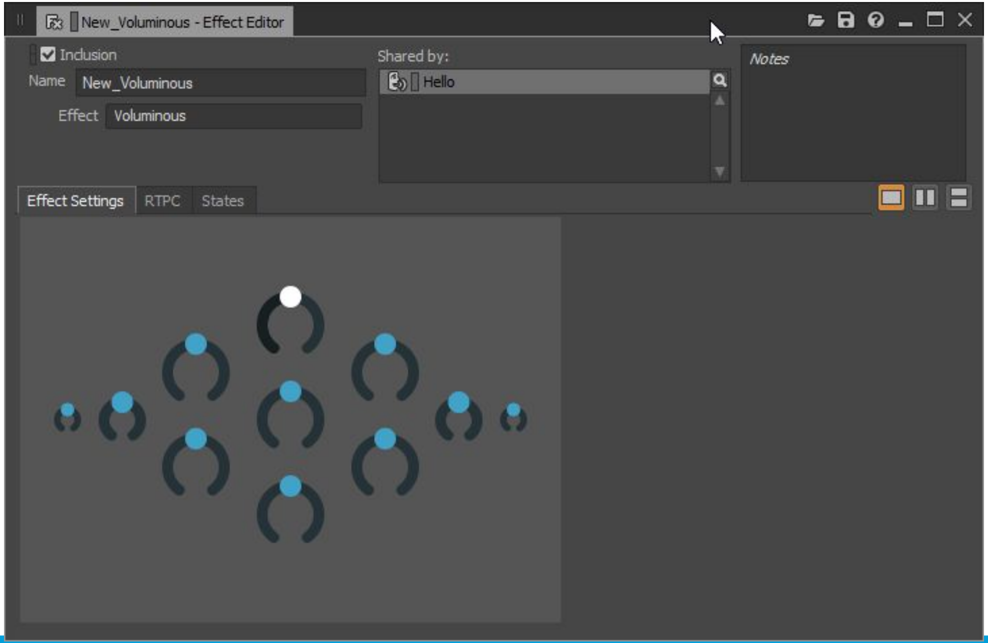
Graceful initialization and shutdown (cont.)



```cpp
// VoluminousPlugin.cpp
static int numInstances = 0;

VoluminousPlugin::VoluminousPlugin() { ++numInstances; }
void VoluminousPlugin::Destroy()
{
    delete this;
    if (--numInstances == 0)
    {
        for (int i = 20; --i >= 0;)
            MessageManager::getInstance()->runDispatchLoopUntil (1);

        shutdownJuce_GUI();
    }
}
```

Where next

- An actual JUCE client for Wwise plug-ins with limited platform support?
- VST plug-in support in Wwise for pre-rendering effects?



New_Voluminous - Effect Editor

☑ Inclusion

Name: New_Voluminous

Effect: Voluminous

Shared by: Hello

Q & A

contact: jrobichaud@audiokinetic.com

sample: github.com/joelrobichaud/Voluminous