pmod *library SDK*

C++ cross platform SDK to create an object model hierarchy and project to multiple different languages including Windows runtime, .NET , Java, Objective C, Node.

A typical application project workflow will include the following steps:

* Describe your model schema in a neutral syntax (today is C#)
* Run code generation tools
* Author your code in portable C++ using pmod SDK library
* Package your code for different platforms including the code generated projection for each of the supported language
* Use the object model from the different language and platforms supported by pmod

# Overview

Developing a cross platform application today is a challenge if you want to target the most popular phone devices such as iOS, Android and Windows Phone. This is true also for desktop development where there is a variety combination of platforms, languages and OS’s.

.NET Core is emerging as a good alternative but still lack of full support for certain platforms make it at hard choice for devs. On the other hand, C++ has been for decades a language/platform supported in most of the devices and OS’s, and the progress of the language (> C ++ 11) make it a very attractive alternative to develop a cross platform solution.

So what is preventing devs today to select C++ as an alternative?

The answer is probably the lack of a good and portable native (C++) UI platform for each of the target devices. Also developers prefer the best UI framework available for each platform and they will select probably Cocoa UI for iOS, Java for Android, and C#/XAML for Windows Phone. The pmod SDK library will give them the natural ‘projection’ object model for each of this platforms without the developer to write any ‘glue’ code. The combination of generated code and the pmod runtime library will make this to happen.

## Goals

The pmod SDK has the following goals:

* Support building of cross-platform applications using C++ and combine it to your favorite UI framework
* Support of the most useful types that allow the construction of powerful app which includes:
  + Definition of properties with notifications
  + Enumerable & observable collections
  + Enumeration values
  + Structs
  + Dictionary of <string,object>
  + Generic event notification mechanism
  + Method invocation
  + Commands with enable/disable state with notifications
  + Full asynchronous pattern support
  + Full reflection type support
  + JSON serialization
* Foundation types and concepts based on the windows runtime types for smooth integration with Microsoft technology
* Primary target for phone devices to tackle memory consumption and perf from the ground up
* Runtime projection for the following languages:
  + Java (both desktop and Android devices)
  + Objective-C (for iOS & MacOSX)
  + Windows runtime (.winmd support)
  + .NET (for .Desktop & Xamarin)
  + Node
* Portable code generation tools (currently runs in MacOSX, Windows, Linux)
* No additional dependencies (no boost or any other OSS component is required)
* Full integration with Visual Studio 15 and NuGet packages for easy adoption and development
* Integration with XCode projects

# Pmod layers

The following diagram shows the layers of the pmod library with the user integration code.



* The blue components will be part of the SDK and the application will include in its deploy mechanism.
* Green components are generated during the build phase.
* User code is the cross platform layer build by the user

# Case study:

A sample that ships with the SDK is building an application intended to access the movies available on theater and capabilities to see the poster image, rating votes and a short description.

Step 1: Describing the model schema

public interface IMoviesApiService

{

[MethodAttribute(IsAsync = true)]

IPageMovies GetNowPlaying(int page, string language);

}

Step 2: Implement the code in C++ using pmod & Microsoft Cpp REST SDK:

class CMoviesApiService :

public ::\_FastMoviesApiServiceBase 🡸= generated base class

{

protected:

HRESULT GetNowPlayingAsync(INT32 page, HSTRING language, const foundation::AsyncOperationClassPtr& asyncOperationClassPtr) override

{

// cpp rest SDK specific code …

asyncOperationClassPtr.SetCompleted(movies);

return S\_OK;

}

Java Android integration:

**public** AsyncOperation<PageMovies> getNowPlayingAsync()  
{  
 **final** AsyncOperation<PageMovies> pageMoviesResult = *apiService*.getNowPlaying(1, **""**);  
 pageMoviesResult.addCompletedListener(**new** CompletedEventListener() { *//this is fine* @Override  
 **public void** onCompleted(CompletedEvent eventArgs) {  
 **nowPlayingMovies** = pageMoviesResult.getResult();  
 }  
 });  
 **return** pageMoviesResult;  
}

C#/XAML integration:

private async void MainPage\_Loaded(object sender, RoutedEventArgs e)

{

var result = await App.MoviesApiService.GetNowPlaying(1, "");

// when we return from await it will be the UI dispatch thread

int count = result.Movies.Count;

MoviesPageListView.ItemsSource = result.Movies;

}

Note there is a natural integration with the each of the supported platforms taking advantage of the advanced feature available which in this case is using ‘await’ keyword for the .NET & windows runtime platform.

Alternatives to this approach

Without the pmod SDK what would have been the alternatives to create such application, considering the different targets desired? which include iOS, Android, Windows Phone.

Let’s say you have already invested heavily in native code that already resolve your business layer, what it takes from here to target your application in multiple platforms?

Android:

To fully take advantage of the platform you want to use java for the UI development. You would need to write a JNI layer to access your native code API. This is a tough task considering all the knowledge required to resolve including string encoding, thread affinity, event dispatching, etc.. You would end up writing thousands of lines of code both in Java & additional C++.

iOS:

Objective-C has a natural way to consume C++ and in this case it may be possible to not require any layer at all. But the reality is that UI devs would prefer a natural Objective-C to use instead of mixing C++ and Objective-C. Also you lose the advantage of providing a full KVO integration

Windows Phone (UWP):

Writing a Universal Windows app would typically require creating a wrapper layer to produce a .winmd file to consume from C#. The task would require writing an .idl file and another C++ layer to adapt the existing C++ API. How about supporting async patterns and observable patterns? You would end up writing thousands of lines of code.