Introduction:

The following document describes the organization and conventions used by the source code of the pmod SDK.

SDK organization

The SDK is divided in two main modules:

1) Foundation/Model support

This module is responsible of the API to support the following features:

* Type registration
* Type inspection
* Activation factory registration
* Adapter factory API and registration
* Dictionary implementation
* Object type info services
* Asynchronous operation implementation
* Enumeration type implementation
* Reflection API for Object Dispatch type objects
* Projection runtime support for the following languages (beyond C++)
  + Objective-C
  + Java
  + .NET
  + Node JS

Consumers of this layer should be able to get basic functionality of reflection and projection support for all supported platforms and languages. When using advanced mechanism such as Observable collections and Observable objects, the library does not provide an implementation and it let the consumer to provide it. Most of the code generation can be used as a starting point in conjunction of the pmod template library support, which define classes to help on this matter.

2) Foundation/Model library

This modules define a runtime library that implement most of the interfaces defined by the SDK using in most of the cases COM aggregation as the primary mechanism of extension.

The library offers an implementation for the following interfaces:

* Observable Collections
* Observable Objects
* Commands
* Async operations
* Binding engine with expression support
* Basic View Model support

Also the library define numerous sub modules to help authoring powerful portable components such as:

* JSON parser
* Logger framework
* Mailbox dispatcher
* Expression library support based on reflection

SDK principles

One of the key principles of the SDK API is the fact that both the foundation and library runtimes are based on the following rules:

* No STL C++ runtime struct or types. That means no vector<T> or map<K, V> types.This allow mixing different C++ runtimes and completely isolation on the compiler support and STL library.
* Export symbols based on ‘C’ naming conventions (no mangled C++ is allowed)
* No direct export of any C++ class type but using a raw interface model (which only allow vtable method definition)
* No throw exceptions in any API definition. Same as before the runtime should abstract from any specific compiler implementation
* HRESULT type for most of the method calls. Since the SDK API does not allow exception outside the boundary of the runtime the return tytpe is based on Microsoft COM HRESULT type
* Reference count based on COM IInspectable interfaces. Most of the interfaces are defined derived from IInspectable to better support the Windows runtime architecture
* IEnumerable and IDictionary interfaces based on similar windows runtime equivalent
* COM aggregation to extend classes. Since we don’t allow to directly export C++ classes, the library allow creation of objects that users can create as ‘aggregable’ instances and override they default behavior

SDK Coding conventions

The following conventions are applied to most of the source code:

* Type definition will use the Pascal case convention
* Interface definition will start with an uppercase ‘I’ to highlight the fact that is a pure vtable definition
* Most the interfaces will define a unique GUID that wil be used later to cast the class implementation
* All of the interface definition will be defined in a separate file
* File name convention:
  + Public headers will use the lower case with underscore like this:
    - <json\_utils.h>
    - <enumerable.h>
  + Implementation files will use the Pascal camel convention like:
    - “ObservableObject.h”
    - “ObservableCollection.cpp”

Public Header organization

Consuming the library typically requires accessing the public SDK headers and binary builds of the dynamic (or static) library.

The hierarchy of files is the following:

* public/
  + foundation/ (foundation namespace definition)
    - interfaces/
    - library/
      * interfaces/
  + pmod/ (pmod namespace definition)
    - interfaces/
    - library/
      * interfaces/
  + pal /(PAL specific libararies)
  + winrt/ (Windows runtime specific headers)

Implementation hierarchy organization

* src/
  + foundation/: Foundation classes implementation
  + model/: Model classes implementation
  + inc/: shared common include files for the implementation
  + src/: shared common source files
  + foundation\_library/: foundation library classes
  + model\_library/: model library classes
  + foundation\_pal/: common PAL classes
  + pal/: PAL implementation classes for the different platforms supported
  + winrt/: projection support for windows runtime
  + objective-c/: projection support for Objective-C language
  + coreclr/: projection support for .NET language interop
  + java/: projection support for Java/JNI
  + nodejs/: projection support for Node JS platform