An opinionated comparison of C++ frameworks for consuming and implementing Windows Runtime types

devblogs.microsoft.com/oldnewthing/20220606-00

June 6, 2022

Raymond Chen

There are three leading C++ frameworks for consuming and implementing Windows Runtime types. The current recommendation (as of this writing) is C++/WinRT.

	WRL	C++/CX	C++/WinRT
Error handling	`HRESULT` -based	Exception-based	Exception-based
Interop with C++ standard library	Poor	Middling	Good
Code verbosity	Very high	Low	Low
Code generation	Small	Explosively large	Small
Compile time	Low	Low	High[1]
IDL file	Manually authored	Automatically generated[2]	Manually authored
Static class constructor[3]	Supported	Not supported	Supported
COM static lifetime for factories[4]	Can implement manually	Cannot implement	Built-in
Default threading model	It’s complicated[5]	Free	Free
Can choose nondefault threading model	Yes	No	Yes
Language	Standard C++	Nonstandard extension	Standard C++

Static analysis tools	Supported	Not supported	Supported
Language standard required	C++11 and higher	C++14 or C++17 with `/await`	C++17 and higher
Forward compatibility	Compatible	Incompatible with C++20	Compatible
XAML compiler support	No	Yes	Yes
Coroutine support	No	Yes via PPL[6]	Yes
License/source code	Ships in SDK	Closed source	Open source
Support	Maintenance	None	Active

Notes

[1] C++/WinRT contains a large number of types and template specializations, which slows down the compiler. The precompiled header file easily exceeds 1GB in size. You can define `WINRT_LEAN_AND_MEAN` to remove rarely-used features and improve compile times.

[2] Automatic generation of the IDL file is a two-edged sword. Although it saves a lot of effort, it can also get in the way: If you need to make a runtime class object marshallable, you need to register a marshaller for the autogenerated interface, which will have an ugly autogenerated name, and whose UUID may not be stable. Autogeneration also conflicts with versioning, makes it harder to interop with other languages, and it can result in puzzling behavior if you don't understand how the autogeneration works. Furthermore, the autogenerated interface names do not follow Windows Runtime naming conventions.

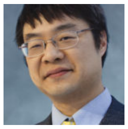
[3] Static class constructors allow class statics to be delay-initialized. This is significant because running constructors at `DLL_PROCESS_ATTACH` creates the risk of deadlocks and other unfortunate behaviors. C++/CX clients must work around this by having a static `InitializeStatics()` method which initializes the statics (e.g., dependency properties) and calling it at an opportune moment.

[4] COM static lifetime allows you to register an object in the COM static lifetime store, which allows you to (1) obtain it later, and (2) destruct it automatically when COM is uninitialized. The former provides a persistent-lifetime object for things like global event sources. The latter permits the object's destructors to run while COM is still initialized.

[5] Default is normally free-threaded, but if `BUILD_WINDOWS` is set, then default is single-threaded.

[6] PPL coroutine support is very large.

Raymond Chen



Follow