

# These Aren't the COM Objects You're Looking For

September, 2018



**Victor Ciura**

Technical Lead, Advanced Installer

[www.advancedinstaller.com](http://www.advancedinstaller.com)

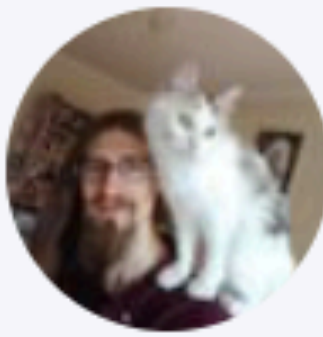
# *Abstract*

Windows COM is 25 years old. Yet it is relevant today more than ever, because Microsoft has bet its entire modern WinRT API on it (starting with Windows 8/10). But, if you're familiar with the "old" COM with its idioms and SDK helper classes, you're in for a treat. With the advent of modern C++ 17, using COM objects and new Windows APIs in your applications feels like a completely new experience.

In this session, we'll explore how using modern C++ features can radically transform the shape of your COM code. By eliminating a lot of boilerplate, your code will be much more readable and maintainable. Classic COM idioms around activation and `QueryInterface()` can feel totally different with modern C++ helpers. A beautiful example of modern COM usage is C++/WinRT (now part of Windows SDK). This is a standard C++ language projection for the new Windows Runtime API.

COM memory management, data marshalling, string handling can all feel quite mechanical in nature and very error prone, so a little help from modern C++ facilities would be more than welcomed. Error handling and debugging can be cumbersome for COM like APIs; we'll explore some tricks to improve this experience, as well.






Simon Brand @TartanLlama · Jul 6

So this happened

#CppCon







Thaaaaats



**Monday, September 24**

11:00

**Enough string\_view to Hang Ourselves**  
Victor Ciura

Manage

**Tuesday, September 25**

09:00

**Regular Types and Why Do I Care ?**  
Victor Ciura

Manage

**Wednesday, September 26**

09:00

**These Aren't the COM Objects You're Looking For**  
Victor Ciura

Manage

**You are here --->**

**Thursday, September 27**

09:00

**Better Tools in Your Clang Toolbox: Extending clang-tidy With Your Custom Checks**  
Victor Ciura

# These Aren't the COM Objects You're Looking For



Part 1 of N

Why COM ?

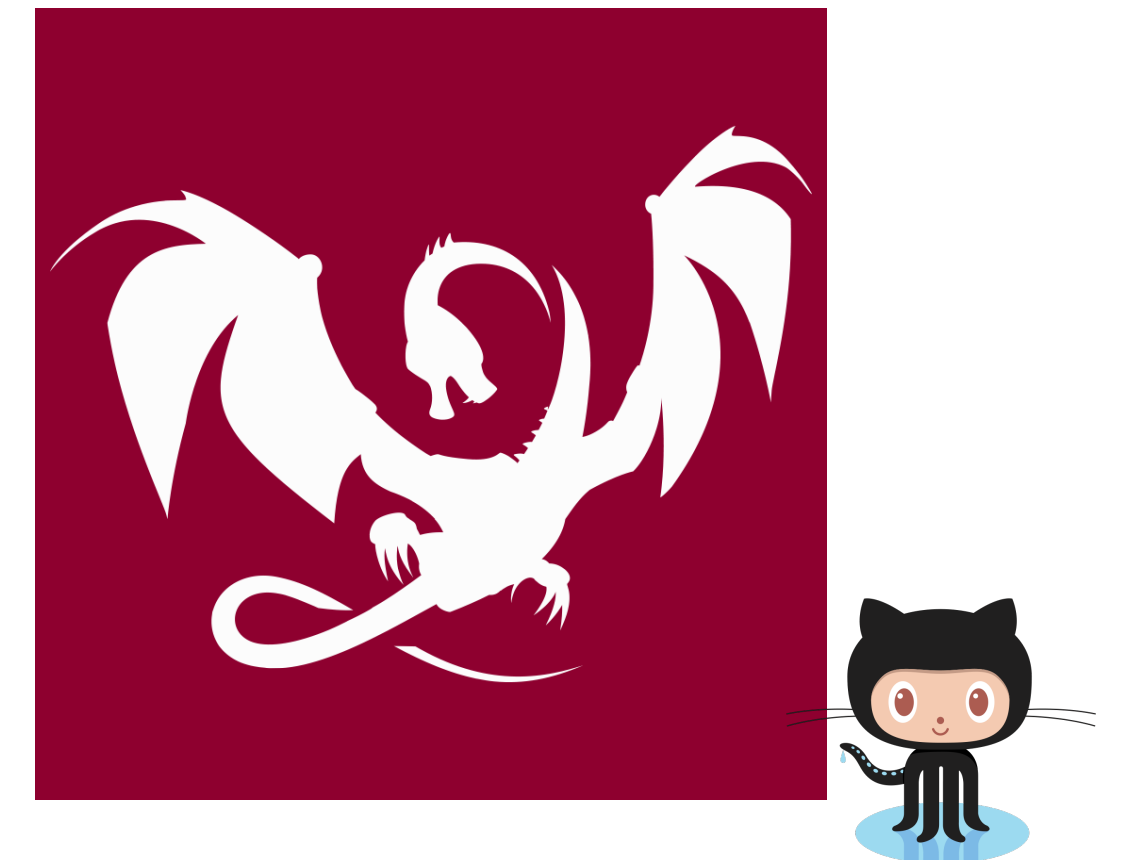
Why are we talking about this ?

Have we really exhausted all the cool C++ template<> topics 🤪 ?

# Who Am I?



**Advanced Installer**



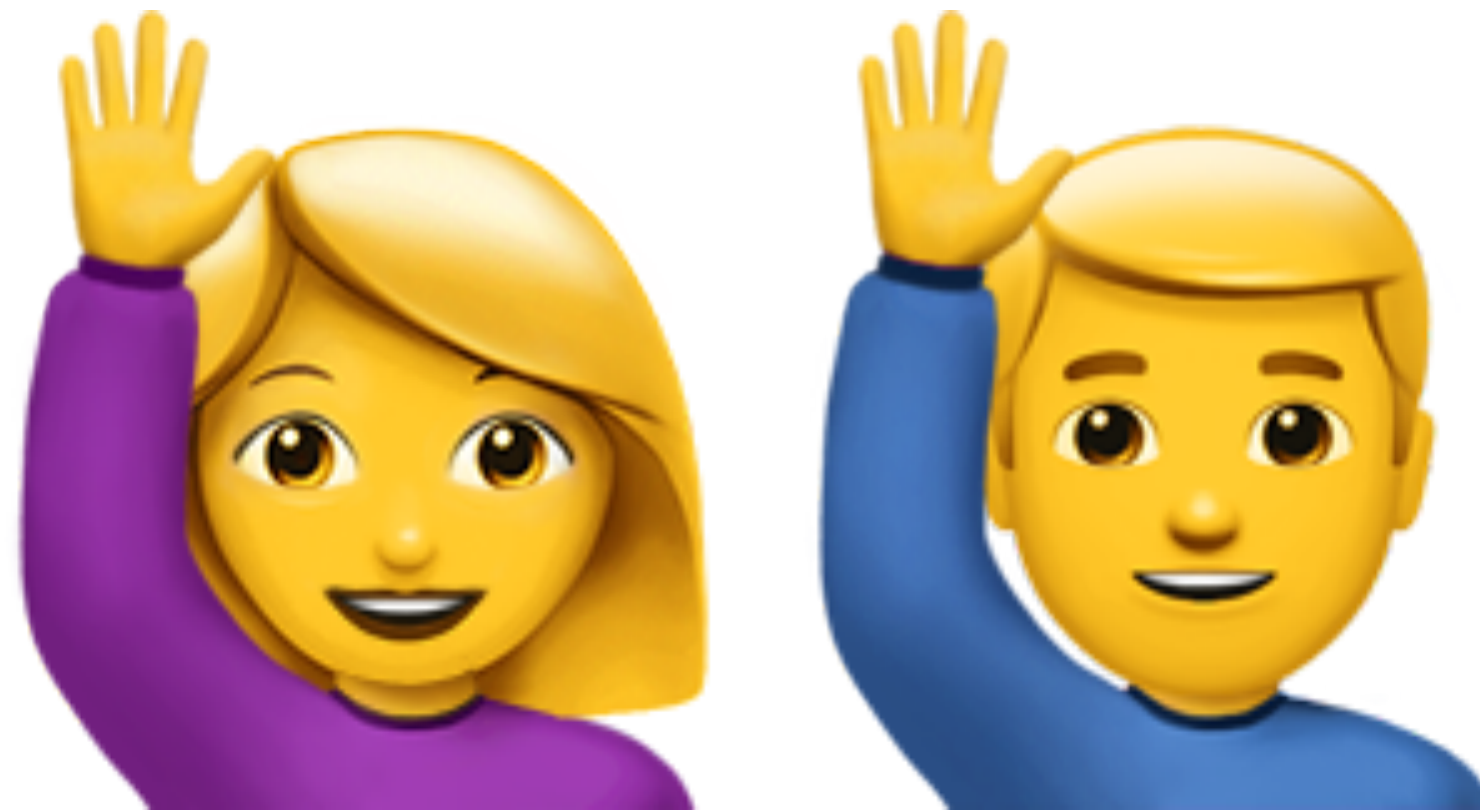
**Clang Power Tools**



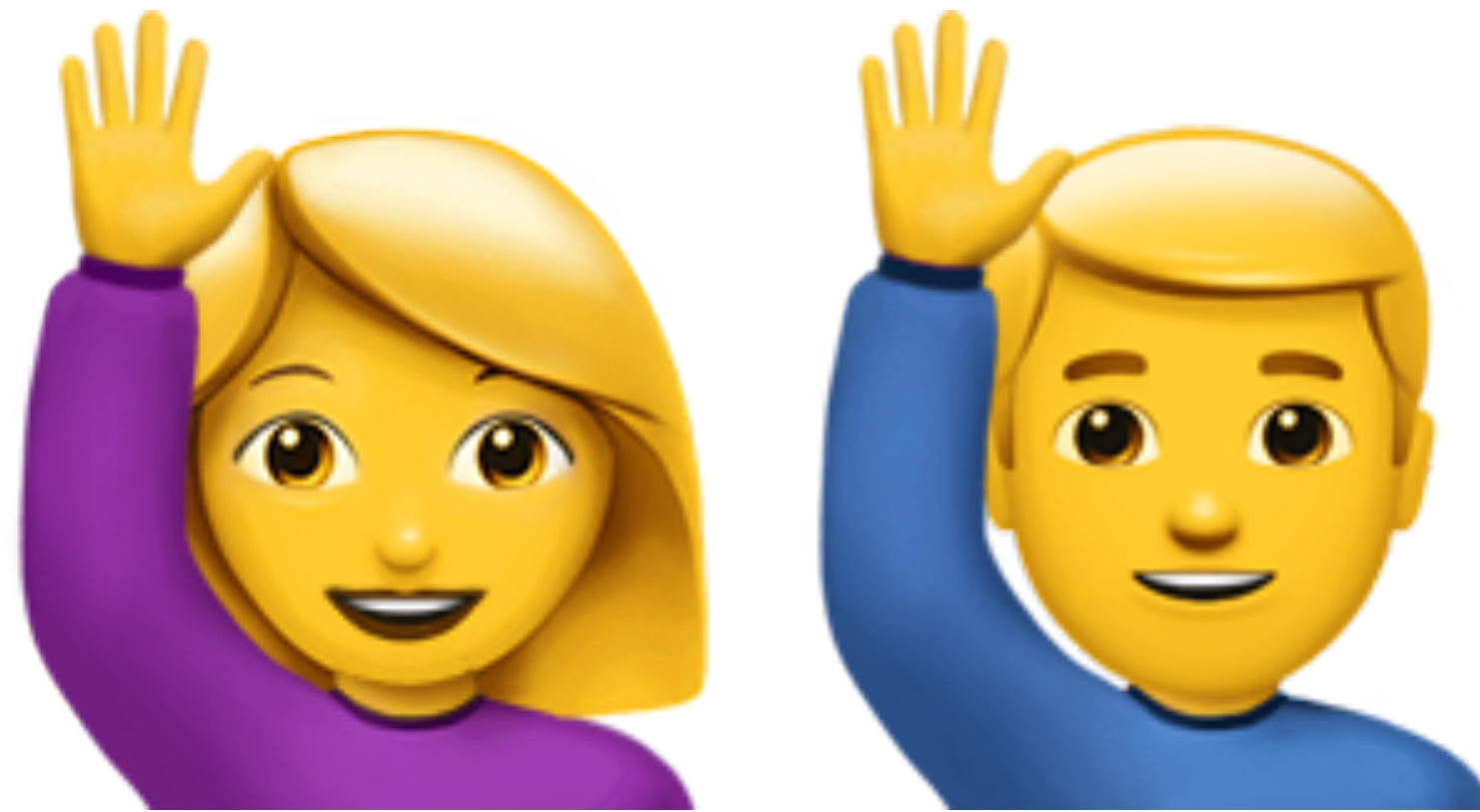
**@ciura\_victor**



**How many of you  
have done any COM programming  
in the last 20 years ?**



**How many of you  
are doing COM programming  
*currently* (this year) ?**





**Let's start using COM...**

CoInitializeEx()

# CoInitializeEx()

- initializes the COM library for use by the calling **thread**
- sets the thread's concurrency model (COINIT\_APARTMENTTHREADED, COINIT\_MULTITHREADED)
- and creates a new *apartment* for the thread (if one is required)

Need to initialize the COM on a *thread* before any COM API calls,  
otherwise the function will return **CO\_E\_NOTINITIALIZED**



# CoInitializeEx()

CoInitializeEx() must be called **once** for each **thread** that uses COM

Multiple calls to CoInitializeEx() by the same thread are *allowed*,  
but subsequent valid calls return **S\_FALSE**

**Close** COM *gracefully* on a thread: each successful call to CoInitializeEx(), must be balanced by a corresponding call to **CoUninitialize()**

**“Do. Or do not. There is no try.”**  
**– Yoda**



# Any Trekkies in the room ?





# Any Trekkies in the room ?

**I'm sorry...**



**Some of this material may be unpleasant for you.**

**So, how can we fix COM init ?**

CoInitializeEx()



RAII

```
class ComInitializerEx
{
public:
    ComInitializerEx(DWORD aComFlags) {
        HRESULT hRes = ::CoInitializeEx(nullptr, aComFlags);
        mClose = (hRes == S_OK);

        if ((hRes != S_OK) && (hRes != S_FALSE))
            ATLASSERT(SUCCEEDED(hRes));
    }

    ~ComInitializerEx() {
        if (mClose)
            ::CoUninitialize();
    }

private:
    bool mClose; // true if COM was initialized by this instance on this thread
};
```

```
void FuncRequiringCom()
{
    ComInitializerEx com(COINIT_MULTITHREADED);
    ...
    // use of COM APIs
}
```

👉 you can **nest** these objects  
as needed

```
class ClassRequiringCom
{
public:
    MethodsRequiringCom();
    ...
private:
    ComInitializerEx com_{ COINIT_MULTITHREADED };
};
```

**Use it anywhere...**



# CoInitializerEx / RAII

- You don't care about the **thread** you're on
- You don't care if you're already **initialized** on this thread
- You can **nest** these objects as needed
- You don't worry about **closing**/unloading COM when you're done

# Let's Talk About **Strings**

**What is the COM string type ?**

**BSTR**



# BSTR

```
typedef WCHAR OLECHAR;  
typedef OLECHAR * BSTR;
```

A BSTR is a *composite* data type that consists of:

Length prefix	A four-byte integer that contains the number of bytes in the following data string. It appears immediately before the first character of the data string. This value does not include the terminator.
Data string	A string of Unicode characters. May contain multiple <i>embedded null</i> characters.
Terminator	A NULL (0x0000) WCHAR

# BSTR

```
typedef WCHAR OLECHAR;  
typedef OLECHAR * BSTR;
```

BSTR is a **pointer**



Length prefix	A four-byte integer that contains the number of bytes in the following data string. It appears immediately before the first character of the data string. This value does not include the terminator.
Data string	A string of Unicode characters. May contain multiple <i>embedded null</i> characters.
Terminator	A NULL (0x0000) WCHAR

# BSTR

```
BSTR str = L"My first COM string";
```

This code:

- **compiles**
- **links**
- **is incorrect (not working)**

If (*by accident*) you pass a *string literal* as an argument to a COM function that is expecting a BSTR, the COM function behaves unexpectedly. 🔥

# BSTR

## Allocating / Releasing Memory for a BSTR

```
BSTR str = ::SysAllocString(L"My first COM string");  
  
if (str)  
{  
    UseTheString(str);  
    ...  
    ::SysFreeString(str);  
}
```





**“Somebody has to save our skins.”**  
**– Leia Organa**

RAII

FTW !

# Visual C++ Compiler COM Support

```
#include <comutil.h>
```

Compiler RAI Support Classes:

`_bstr_t`

`_com_ptr_t`

`_variant_t`

`_com_error`  
(HRESULT wrapper)

# \_bstr\_t

- encapsulates the **BSTR** data type
- manages resource allocation through **SysAllocString** / **SysFreeString** and other APIs
- uses *reference counting* to avoid excessive overhead (**CoW**)
- provides various conversion *constructors*: **const char\***, **const wchar\_t\***, **BSTR**, etc.
- provides various *operators*: **=**, **+**, **==**, **!=**, **+=**, **<**, **>**, **char\***, **wchar\_t\***, etc.



# \_bstr\_t

```
_bstr_t str(L"My first COM string");  
ComApiWithBstrParam(str.GetBSTR());  
std::wstring my_std_str = str;
```

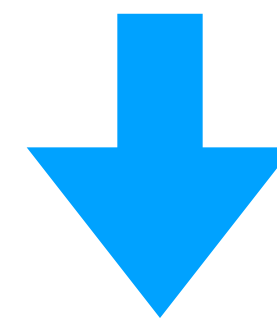
Also:

```
_bstr_t other_str(my_std_str.c_str());  
  
if (str == other_str) // lexicographical compare  
{ ... }
```

# \_com\_ptr\_t

**COM interface smart pointer**

```
_COM_SMARTPTR_TYPEDEF(IMyInterface, __uuidof(IMyInterface));
```



declares the `_com_ptr_t` specialization: `IMyInterfacePtr`

# \_com\_ptr\_t

```
_COM_SMARTPTR_TYPEDEF(ITaskbarList3, __uuidof(ITaskbarList3));
```

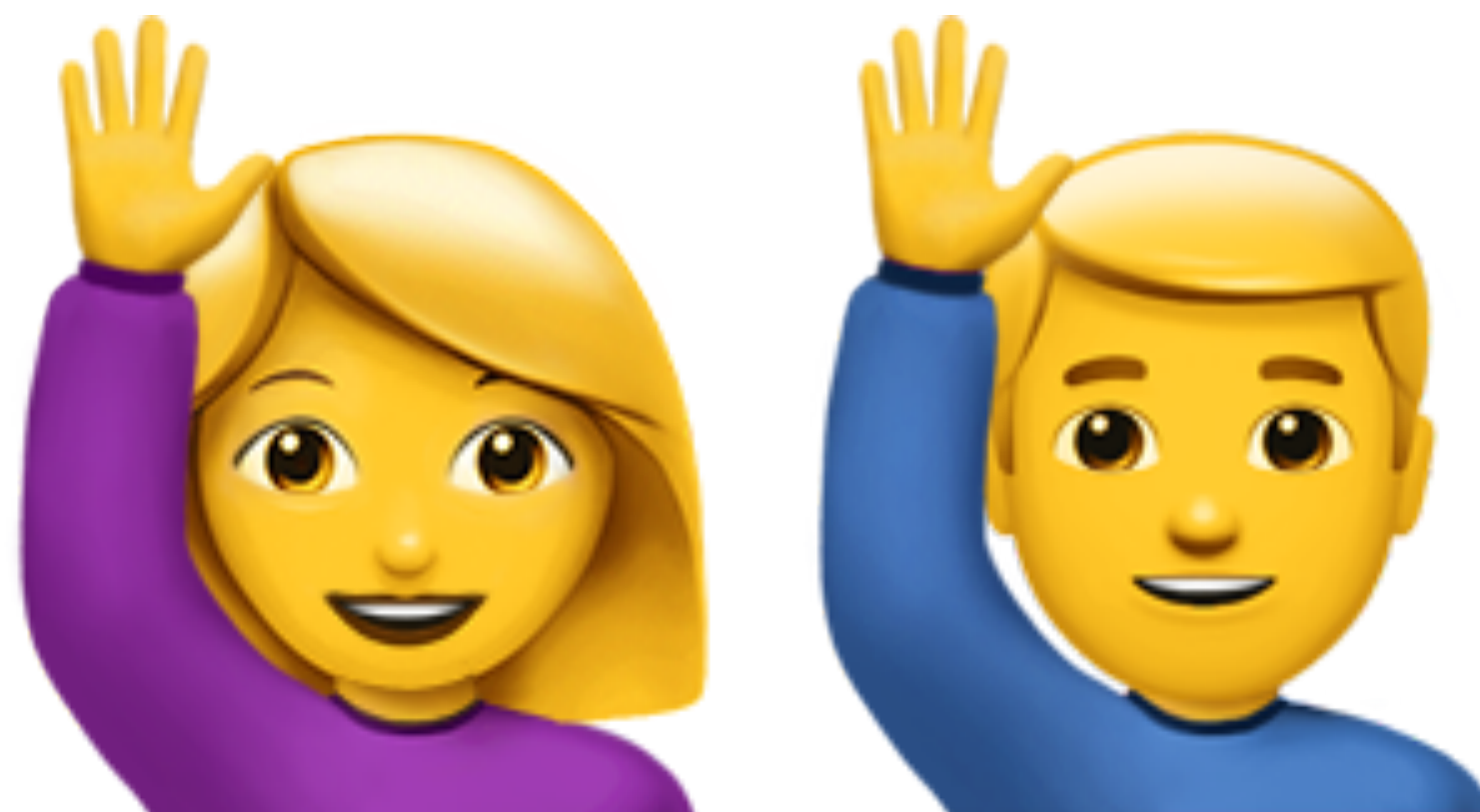
```
ITaskbarList3Ptr taskBar;
```

```
HRESULT hr = ::CoCreateInstance(CLSID_TaskbarList, nullptr,  
                                CLSCTX_INPROC_SERVER,  
                                IID_PPV_ARGS(&taskBar));
```

```
ATLASSERT(SUCCEEDED(hr));
```

```
if (taskBar)  
    taskBar->SetProgressState(wnd, TBPF_INDETERMINATE);
```

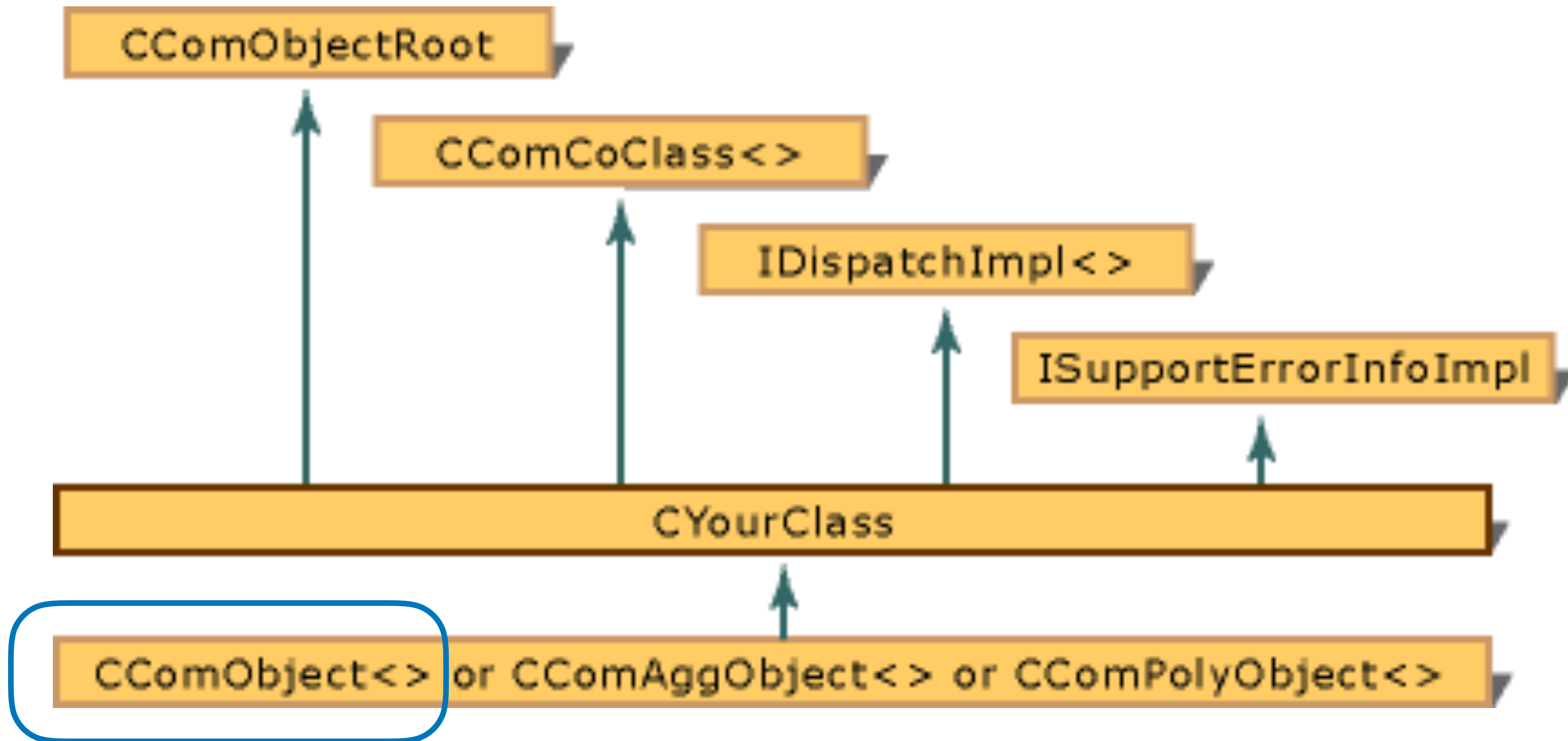
# Active Template Library ATL





# ATL

A set of template-based C++ classes  
intended to **simplify** COM programming 🤔



# CComObject<>

```
template<class Base>  
class CComObject : public Base
```

```
CComObject<CMYCircle> * pCircle = nullptr;  
HRESULT hRes = CComObject<CMYCircle>::CreateInstance(&pCircle);  
ATLASSERT(SUCCEEDED(hRes));
```

```
pCircle->AddRef();  
pCircle->SetRadius(5.2);
```

# CComObject<>

```
class ATL_NO_VTABLE CMyCircle :  
    public CComObjectRootEx<CComSingleThreadModel>,  
    public CComCoClass<CMyCircle, &CLSID_CMyCircle>,  
    public IDispatchImpl<IMyCircle, &IID_IMyCircle, &LIBID_NVC_ATL_COMLib, 1, 0>  
{  
public:
```

```
    DECLARE_REGISTRY_RESOURCEID(IDR_MYCIRCLE)
```

```
    DECLARE_NOT_AGGREGATABLE(CMyCircle)
```

```
    BEGIN_COM_MAP(CMyCircle)
```

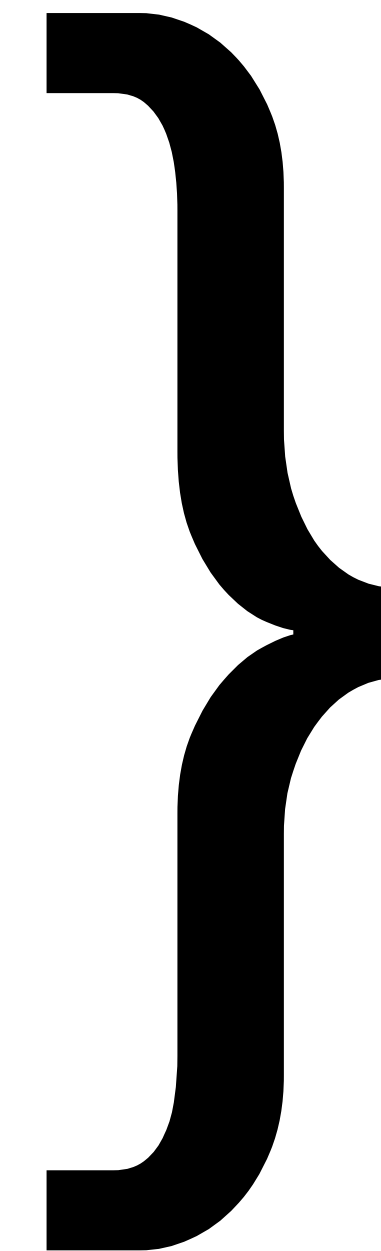
```
        COM_INTERFACE_ENTRY(IMyCircle)
```

```
        COM_INTERFACE_ENTRY(IDispatch)
```

```
    END_COM_MAP()
```

```
    DECLARE_PROTECT_FINAL_CONSTRUCT()
```

```
    ...
```



**BOILERPLATE**

# CComObject<>

```
class ATL_NO_VTABLE CMyCircle :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CMyCircle, &CLSID_CMyCircle>,
    public IDispatchImpl<IMyCircle, &IID_IMyCircle, &LIBID_NVC_ATL_COMLib, 1, 0>
{
    ...

    HRESULT FinalConstruct() { return S_OK; }
    void FinalRelease() { }

public:
    CMyCircle() { }
    STDMETHOD(SetRadius)(double val);
    ...
};
```

} **Actual code we care about**



**“Your focus determines your reality.”**  
**– Qui-Gon Jinn**

# CComObject<>

```
template<class Base>  
class CComObject : public Base
```

**Do you recognize this pattern ?**

**CRTP**

# CRTP

```
template<class Base>  
class CComObject : public Base
```

- achieves a similar effect to the use of *virtual functions*
- without the costs of dynamic polymorphism (no **VTABLE**)
- binding done at *compile time*

This pattern is used extensively in the Windows **ATL** and **WTL** libraries.

**“Power! Unlimited power!”**  
**— Darth Sidious**

# ATL

<code>&lt;comutil.h&gt;</code>	ATL
<code>_com_ptr_t</code>	<code>CComPtr&lt;T&gt;</code>
<code>_bstr_t</code>	<code>CComBSTR</code>
<code>_variant_t</code>	<code>CComVariant</code>



# CComBSTR

- ATL wrapper for the `BSTR` data type
- manages resource allocation through `SysAllocString` / `SysFreeString` and other APIs
- supports *move semantics* `&&`
- provides various conversion *constructors*: `const char*`, `const wchar_t*`, `BSTR`, etc.
- provides various *operators*: `=`, `+`, `==`, `!=`, `+=`, `<`, `>`, `char*`, `wchar_t*`, `BSTR`, etc.

# CComBSTR

```
CComBSTR str(L"My first COM string");  
ComApiWithBstrParam(str); // operator BSTR()  
std::wstring my_std_str = str;
```

Also:

```
CComBSTR other_str(my_std_str.c_str());  
  
if (str == other_str) // lexicographical compare  
{ ... }
```

**“There’s always a bigger fish.”**  
**— Qui-Gon Jinn**

# ATL CString

```
typedef CStringT<TCHAR, StrTraitATL<TCHAR, ChTraitsCRT<TCHAR>>> CAatlString;
```

- supports both `char` and `wchar_t` through `StringTraits` policy
- manages resource allocation & deallocation
- uses *reference counting* to avoid excessive copy overhead (**CoW**)
- 50+ methods and operators

# ATL CString

```
typedef CStringT<TCHAR, StrTraitATL<TCHAR, ChTraitsCRT<TCHAR>>> CAatlString;
```

- provides various conversion *constructors*: `const char*`, `const wchar_t*`, `VARIANT`, etc.
- provides various *operators*: `=`, `+`, `==`, `!=`, `+=`, `<`, `>`, `char*`, `wchar_t*`, etc.
- provides tons of ***utility methods*** and ***string algorithms***

Eg.

Find, FindOneOf, Format, MakeLower, MakeReverse, Left, Mid, Right, Replace, ReverseFind, Tokenize, Trim, etc.



# ATL CString

```
CAtlString str(L"My first COM string");
```

```
ComApiWithBstrParam(str.AllocSysString()); // allocates an OLE BSTR copy  
std::wstring my_std_str = str.GetString(); // get null-term C string
```

Also:

```
CAtlString other_str(my_std_str.c_str());
```

```
if (str == other_str) // lexicographical compare  
{ ... }
```

**What about this **modern** COM  
I keep hearing about ?**

Surely things must have improved in the last 25 years...

# Windows Runtime (WinRT)

**Windows 8 / 10**

**“This is a new day, a new beginning.”**  
**– Ahsoka Tano**

# What is the Windows Runtime ?

Modern class-based, object-oriented Windows API

Metadata about the classes/members

Language projections for natural / familiar use (C++, C#, JavaScript, etc.)



**How do I access  
the Windows Runtime from C++ ?**

# Windows Runtime C++ Template Library

## WRL

- enables you to more easily **implement** and **consume** COM components
- adds little abstraction over the Windows Runtime ABI (very thin wrapper)
- gives you the ability to control the underlying code (low-level access)
- error handling based on **HRESULT**
- design inspired by **ATL** => can be mixed with existing older COM code
- uses ISO standard C++
- uses smart pointers & RAII
- rather verbose (boilerplate)
- supports UWP apps

# Windows Runtime C++ Template Library

## C++/CX

- uses **non**-standard C++ language extensions
- terse syntax
- learning curve
- high-level abstraction
- represents **HRESULT** values as **exceptions**
- automates housekeeping tasks
- discontinued...

Let's start using **Windows Runtime...**

```
#include <Windows.Foundation.h>  
#include <wrl/wrappers/corewrappers.h>  
#include <wrl/client.h>  
  
using namespace ABI::Windows::Foundation;  
using namespace Microsoft::WRL;  
using namespace Microsoft::WRL::Wrappers;
```



# WRL

## Init...

Microsoft::WRL::Wrappers::RoInitializeWrapper

## RAII

```
RoInitializeWrapper init(RO_INIT_MULTITHREADED);  
if (FAILED(init))  
{  
    return PrintError(__LINE__, init);  
}
```

ABI::Windows::Foundation::IUriRuntimeClassFactory

```
// get the activation factory for IUriRuntimeClass interface
ComPtr<IUriRuntimeClassFactory> uriFactory;
HRESULT hr = GetActivationFactory(
    HStringReference(RuntimeClass_Windows_Foundation_Uri).Get(),
    &uriFactory);
if (FAILED(hr))
{
    return PrintError(__LINE__, hr);
}
```

What is the **Windows Runtime** string type ?

HSTRING

# HSTRING

represents **immutable** string in the Windows Runtime  
(handle)

Usage:

WindowsCreateString()

WindowsDuplicateString()

WindowsDeleteString()

WindowsConcatString()

<https://docs.microsoft.com/en-us/windows/desktop/WinRT/hstring>

# HSTRING

```
HRESULT WindowsCreateString(  
    PCNZWCH sourceString,  
    UINT32 length,  
    HSTRING *string  
);
```

```
HRESULT WindowsDeleteString(  
    HSTRING string  
);
```



```
HRESULT WindowsDuplicateString(  
    HSTRING string,  
    HSTRING *newString  
);
```

```
HRESULT WindowsConcatString(  
    HSTRING string1,  
    HSTRING string2,  
    HSTRING *newString  
);
```

<https://docs.microsoft.com/en-us/windows/desktop/WinRT/hstring>

# HSTRING

`Microsoft::WRL::Wrappers::HString` is the HSTRING wrapper from **WRL**

```
HString str;  
hr = str.Set(L"Hello");  
if (FAILED(hr))  
{  
    return PrintError(__LINE__, hr);  
}
```



# HSTRING



`Platform::String` is the language projection for `C++/CX`

Usage:

```
Platform::String ^ s = L"Hello";
```

```
bool String::operator+ (String ^ str1, String ^ str2);
```

```
bool String::operator== (String ^ str1, String ^ str2);
```

**“You can’t stop the change, any more than  
you can stop the suns from setting.”  
— Shmi Skywalker**

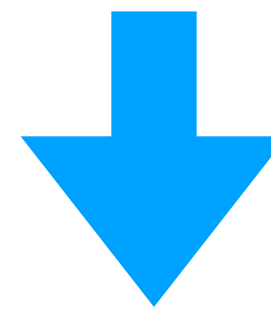
# What is C++/WinRT ?

- an ISO standard **C++17** language projection for the **Windows Runtime**
- header-only library
- C++ class wrappers for WinRT APIs
- you can *author* and *consume* Windows Runtime APIs
- supersedes **WRL** and **C++/CX**

<http://aka.ms/cppwinrt>

# What is C++/WinRT ?

<https://github.com/Microsoft/cppwinrt>



**Windows SDK 10.0.17134.0**  
(Windows 10, version 1803)

<http://aka.ms/cppwinrt>

# What is C++/WinRT ?

Project was started a few years back by **Kenny Kerr**.

If you want to learn more about the project & history,  
checkout the links below:

<https://moderncpp.com>

<http://cppcast.com/2015/05/kenny-kerr/>

<http://cppcast.com/2016/10/kenny-kerr/>



<https://kennykerr.ca/about/>



# What is C++/WinRT ?

CppCon 2017: Scott Jones & Kenny Kerr "C++/WinRT and the Future of C++ on Windows" **cppcon** | 2017  
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

```
C:\> cppwinrt.exe -in local
```

```
C:\winrt\base.h  
C:\winrt\Windows.Media.h  
C:\winrt\Windows.UI.Composition.h  
...
```

```
#include "winrt\Windows.Media.h"  
using namespace Windows::Media;  
int main()  
{ ... }
```

```
C:\> cl.exe app.cpp /std:c++17 ...
```

From `cppwinrt.exe` to `cl.exe`

SCOTT JONES  
KENNY KERR

C++/WinRT  
and the Future of  
C++ on Windows

CppCon.org

[https://www.youtube.com/watch?v=7TdpWB\\_vRZM](https://www.youtube.com/watch?v=7TdpWB_vRZM)



# What is C++/WinRT ?

Effective C++/WinRT for UWP and Win32 - *Brent Rector, Kenny Kerr*

Effective C++/WinRT for UWP and Win32

### What is C++/WinRT?

- An ISO standard C++17 language projection for the Windows Runtime
- Header-only library
- C++ class wrappers for WinRT APIs
- How do you get it?
  - Visual Studio 2017 15.7
    - Select C++ workload

Microsoft Build  
May 7-9, 2018 // Seattle, WA

Microsoft BUILD  
2018

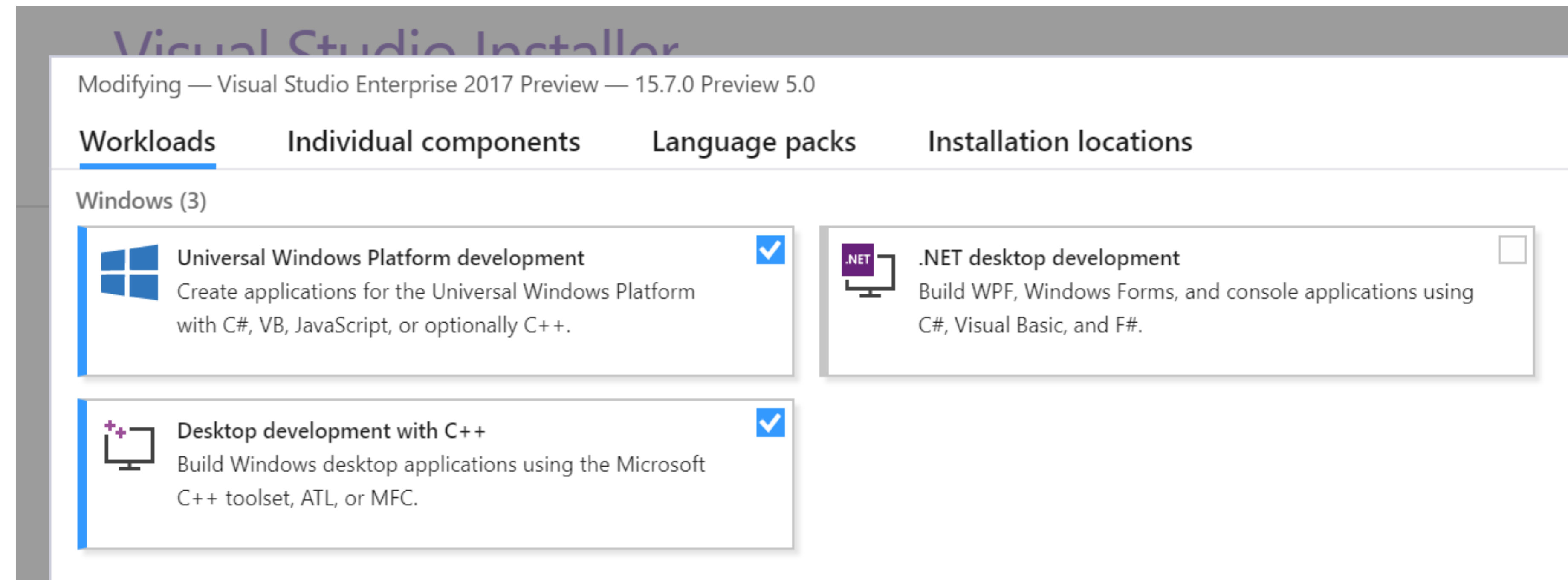
<https://channel9.msdn.com/Events/Build/2018/BRK2425>

# What is C++/WinRT ?

## How to get it

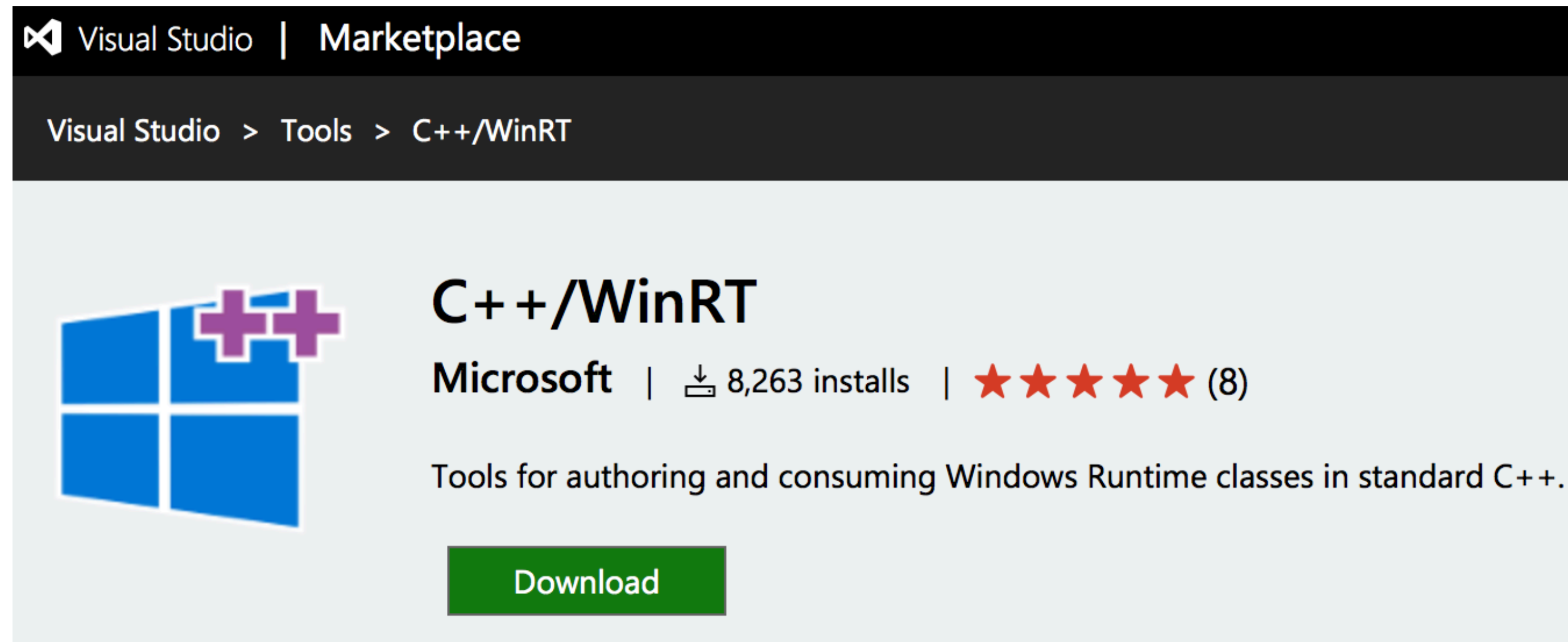
Comes with Visual Studio 2017 (starting with v15.7)

✓ Select C++ workload



<https://docs.microsoft.com/en-us/windows/uwp/cpp-and-winrt-apis/intro-to-using-cpp-with-winrt>

# C++/WinRT Visual Studio extension



- **Debug visualization** of C++/WinRT projected types (similar to C# debugging)
- **Project templates** for getting started a C++/WinRT application
- **MSBuild** support for generating C++/WinRT projection headers and component skeletons

<https://marketplace.visualstudio.com/items?itemName=CppWinRTTeam.cppwinrt101804264>

**Let's start using Windows Runtime...  
(again)**

```
#pragma comment(lib, "windowsapp")  
  
#include <winrt/base.h>  
#include <winrt/Windows.Foundation.h>  
  
using namespace winrt;  
using namespace Windows::Foundation;  
  
winrt::init_apartment();
```

```
cl /std:c++17 /EHsc /W4 /WX /I"%WindowsSdkDir%Include%\%UCRTVersion%\cppwinrt"
```

# C++/WinRT

- performs better and produces smaller binaries than any other language projection
- outperforms handwritten code using the ABI interfaces directly (eg. **WRL**)
- the underlying abstractions use modern C++ idioms, that the Visual C++ compiler is designed to optimize (magic statics, empty base classes, `strlen()` elision)



**What is the C++/WinRT string type ?**

`winrt::hstring`

# winrt::hstring

Represents an **immutable** string consistent with the underlying **HSTRING**

```
winrt::hstring str(L"Hello!");
```

# winrt::hstring

**Encapsulates HSTRING behind an interface similar to that of `std::wstring`**

```
hstring() noexcept;  
hstring(hstring const & h);  
explicit hstring(std::wstring_view const & v);  
hstring(wchar_t const * c);  
hstring(wchar_t const * c, uint32_t s);
```

```

void Test(hstring const & theHstring,
         wstring_view const & theWstringView,
         wchar_t const * wideLiteral,
         wstring const & wideString)
{
    hstring fromDefault{};

    hstring fromHstring{ theHstring };

    hstring fromWstringView{ theWstringView };

    hstring fromWideLiteral{ wideLiteral };
    hstring fromWideString{ wideString.c_str() };

    hstring fromWideLiteralWithSize{ wideLiteral, 256 };
    hstring fromWideStringWithSize{ wideString.c_str(), 256 };
}

```

# winrt::hstring

```
operator std::wstring_view() const noexcept;
```

```
Uri uri{ L"http://example.com" };
```

```
// uses hstring's conversion operator to std::wstring_view  
std::wstring domain { uri.Domain() };
```

```
hstring Uri::Domain() const;
```

# winrt::hstring

An `hstring` is a **range**, so you can use it with range-based `for`, or with `std::for_each`

```
hstring theHstring;  
for (const auto & element : theHstring)  
{  
    std::wcout << element;  
}
```



# winrt::hstring

`hstring` is UTF16,  
but it plays nice with UTF-8 text

```
winrt::hstring w{ L"Hello!" };
```

```
std::string c = winrt::to_string(w);  
WINRT_ASSERT(c == "Hello!");
```

```
w = winrt::to_hstring(c);  
WINRT_ASSERT(w == L"Hello!");
```

ATL CString

BSTR

const wchar\_t\*

winrt::hstring

CCoMBSR

Platform::String

const char\*

\_bstr\_t

XString

wxString

std::string

WTF::String

WTF::CString


QString

folly::fbstring

MFC CString

**“I’m just a simple man  
trying to make my way in the universe.”  
— Jango Fett**

**So we ended up with something like this...**

**XString** ↔  `const char*` ↔ `std::string`  
`const char[N]`

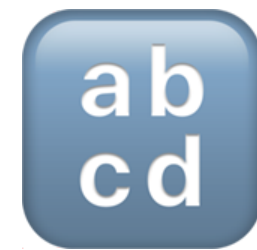
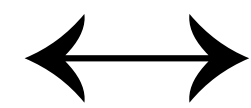
**+ glue code**





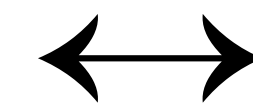
# String Algorithms

XString



const char\*

const char[N]



std::string



**I have a whole talk just on C++17 `std::string_view`**

# Enough `string_view` to hang ourselves

**CppCon 2018**

<http://sched.co/FnL6>



Is C++17 `string_view`  
the answer to all our string problems ?

## `std::string_view`

A *lightweight* string-like view into an array of characters.

It can be constructed from a `const char*` (null terminated) or from a *pointer and a length*.

Intended to be used as *glue code*, to avoid large ***overload sets***.



## `std::string_view`

A `string_view` does not manage the **storage** that it refers to.

Lifetime management is up to the user (caller).

# Convenience Conversions (and Gotchas)

- `const char *` automatically converts to `std::string` via constructor (*not explicit*)
- `const char *` automatically converts to `std::string_view` via constructor (*not explicit*)
- `std::string` automatically converts to `std::string_view` via *conversion operator*
- can construct a `std::string` from a `std::string_view` via constructor (*explicit*)

## `std::string_view`

Design goal: avoid temporary `std::string` objects.

`std::string_view` was designed to interoperate with `std::string` 😈

Caveat: comes with some *usage complexity* (gotchas).

**“It’s a trap!”**

**– Admiral Ackbar**

## On COM...

Windows COM is **25** years old... and it shows this in many corners.

Yet it is **relevant** today more than ever,  
because Microsoft has bet its entire modern WinRT API on it.

With the advent of **C++17**,  
using COM objects and new WinRT APIs feels like a completely **new experience**.



**“The dark side of the COM  
is a pathway to many abilities some  
consider to be unnatural.”**

**– Chancellor Palpatine**

**“Great, kid. Don’t get cocky.”**  
**– Han Solo**

# C++ Slack is your friend



<https://cpplang.slack.com>

**CppLang Slack auto-invite:**

<https://cpplang.now.sh/>



**Cpplang**

cpplang.slack.com



# CppCast

```
auto CppCast = pod_cast<C++>("http://cppcast.com");
```



**Rob Irving**

**@robwirving**

**Jason Turner**

**@lefticus**

# <http://cpp.chat>

<https://www.youtube.com/channel/UCsefcSZGxO9ITBqFbsV3sJg/>

<https://overcast.fm/itunes1378325120/cpp-chat>



**Jon Kalb**

**@\_JonKalb**

**Phil Nash**

**@phil\_nash**

# These Aren't the COM Objects You're Looking For

September, 2018



[@ciura\\_victor](https://twitter.com/ciura_victor)

**Victor Ciura**

Technical Lead, Advanced Installer

[www.advancedinstaller.com](http://www.advancedinstaller.com)

# Questions



[@ciura\\_victor](https://twitter.com/ciura_victor)