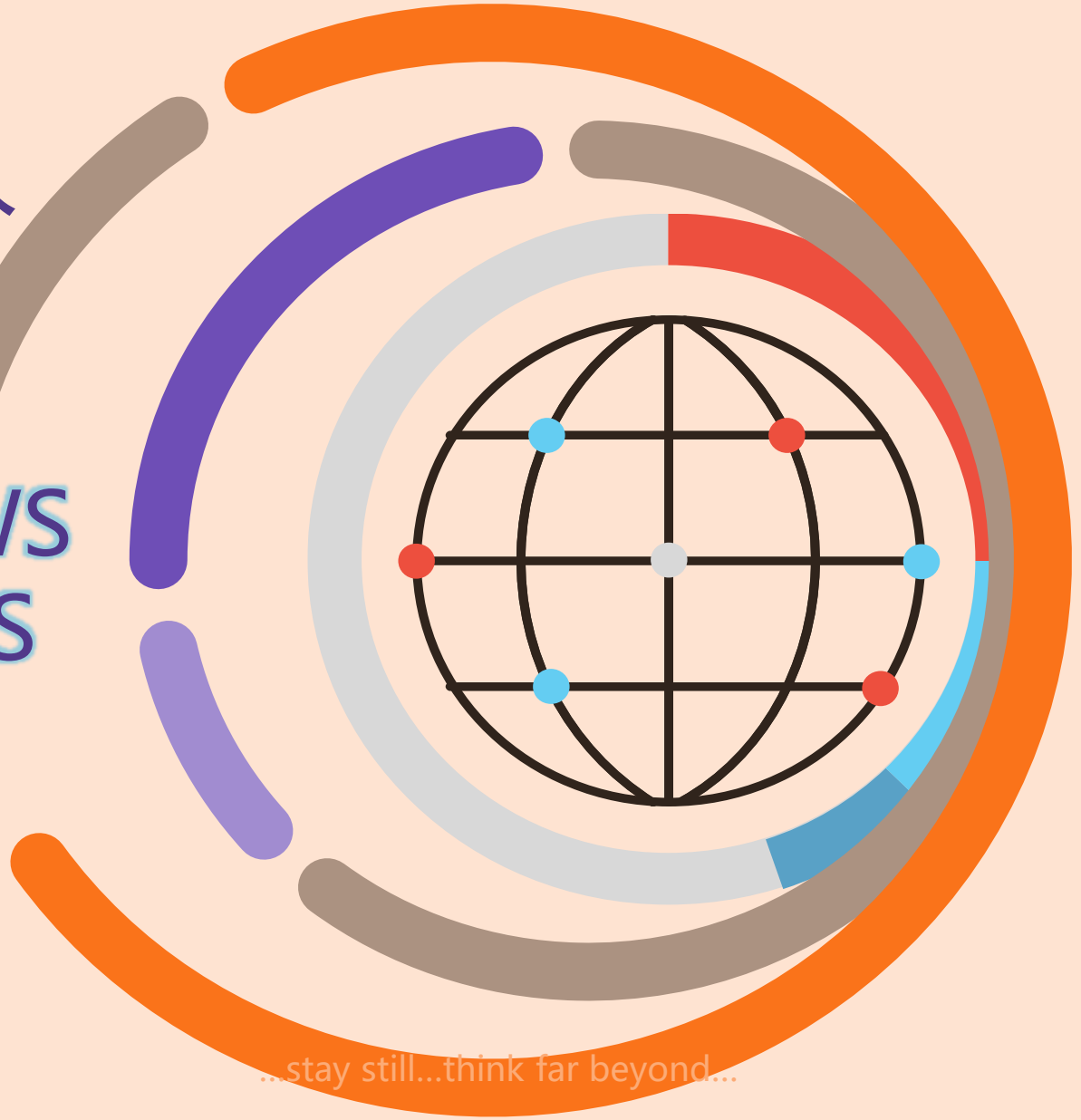
# WINDOWS编程实战进阶

## PRINCIPLE OF WINDOWS AND ITS APPLICATIONS




School of CS

Jicheng Hu

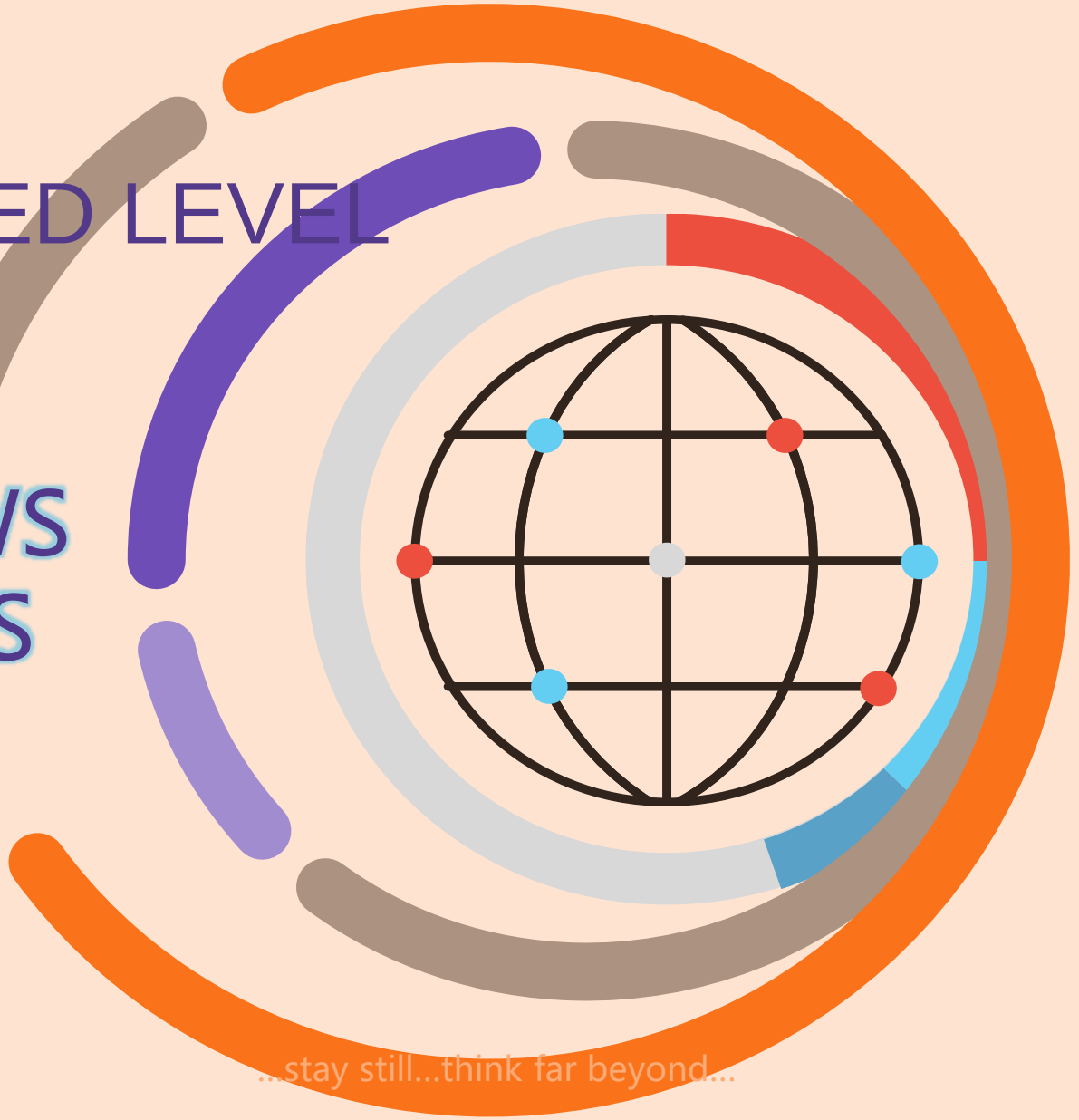
jicheng @ yahoo . com

https://gitee.com/wuhanuniversity/

# Windows Coding Skills

## APPROACHING TO ADVANCED LEVEL

# PRINCIPLE OF WINDOWS AND ITS APPLICATIONS



surf in the programming ocean

seek the endless technique waves

the shimmering spoondrift forms a coding life

...stay still...think far beyond...

# 内容提要

# x.1 RAII

## Resource Acquisition Is Initialization 资源获取即初始化

- 一种利用对象生命周期来管理资源的技术
  - ✓ 内存、文件句柄、网络连接、互斥量……
- 该技术使得资源的获取只需初始化
  - ✓ 对象构造时获取资源
  - ✓ 管理对资源的访问，使其在对象生命周期内保持有效
  - ✓ 在对象析构的时候负责释放资源

https://en.cppreference.com/w/cpp/language/raii

# x.1 RAII

## Resource Acquisition Is Initialization 资源获取即初始化

- 一种利用对象生命周期来管理资源的技术
  - ✓ 内存、文件句柄、网络连接、互斥量……
- 该技术使得资源的获取只需初始化
  - ✓ 对象构造时获取资源
  - ✓ 管理对资源的访问，使其在对象生命周期内保持有效
  - ✓ 在对象析构的时候负责释放资源

https://en.cppreference.com/w/cpp/language/raii

# RAII 例子

将所请求资源的生命周期绑定到一个对象的生存期

```
std::mutex m;

void bad()
{
    m.lock();                     // acquire the mutex
    f();                          // if f() throws an exception, the mutex is never released
    if(!everything_ok()) return; // early return, the mutex is never released
    m.unlock();                   // if bad() reaches this statement, the mutex is released
}

void good()
{
    std::lock_guard<std::mutex> lk(m); // RAII class: mutex acquisition is initialization
    f();                               // if f() throws an exception, the mutex is released
    if(!everything_ok()) return;       // early return, the mutex is released
}                                      // if good() returns normally, the mutex is released
```

参考网页 https://en.cppreference.com/w/cpp/language/raii

# 使用RAII的好处:

- 不需要显式地释放资源
- 对象所需的资源在其生命期内始终保持有效

RAII,java & C#: https://blog.csdn.net/u014053368/article/details/22595289

# RAII SUMMARIZATION:

- encapsulate each resource into a class, where
  - ✓ the constructor acquires the resource and establishes all class invariants or throws an exception if that cannot be done
  - ✓ the destructor releases the resource and never throws exceptions
- always use the resource via an instance of a RAII-class that either
  - ✓ has automatic storage duration or temporary lifetime itself, or
  - ✓ has lifetime that is bounded by the lifetime of an automatic or temporary object

Another name for such technique is Scope-Bound Resource Management (SBRM)

# VS2019 Community 使用微软的 RAII 实现 RAII resource wrappers

https://github.com/Microsoft/wil/wiki/RAII-resource-wrappers

- 打开项目的NuGet管理器
  - ✓ 右击需要安装的项目
  - ✓ 选择manage NuGet packages …
- 安装WIL
  - ✓ 在NuGet管理器中搜索Microsoft.Windows.ImplementationLibrary
  - ✓ 安装最新版本的WIL(1.0.200902.2, 9/3/2020)
- 在需要使用的地方添加头文件 #include <wil/resource.h>

Smart pointers and auto-releasing resource wrappers to let you manage Windows API HANDLEs, HWNDs, and other resources and resource handles with RAII semantics.

参考网页 https://en.cppreference.com/w/cpp/language/raii

# RAII resource wrappers

- The resource wrappers library is usable by any user-mode C++ code through relative inclusion of Resource.h

```
#include <wil/Resource.h>
```

- Note that Resource.h defines wrappers only for types that have been defined prior to the inclusion of Resource.h.

```
#include <WinINet.h>
#include <wil/Resource.h>
```

- It is safe to include Resource.h multiple times. Each time will define wrappers for any new types defined after the previous inclusion of Resource.h.

参考网页 https://github.com/Microsoft/wil/wiki/RAII-resource-wrappers

# RAII resource wrappers

```
// Construct a new pointer with a resource
wil::unique_handle ptr( handle );

// Retrieve the resource
auto resource = ptr.get( );

// Check validity of the resource
if ( ptr )
{
// resource is assigned
}

// Same as previous
if ( ptr.is_valid( ) )
{
// resource is assigned
}

// Free the resource
ptr.reset ( );
```

```
// Free and replace the resource
ptr.reset ( handle );

// Detach resource from the pointer without freeing
auto resource = ptr.release( );

// Return the address of the internal resource for out parameter use
// Note: Also frees any currently-held resource
WindowsApiCall ( &ptr );

// Same as previous
WindowsApiCall ( ptr.put( ) );

// Return the address of the internal resource for in-out parameter use
WindowsApiCall ( ptr.addressof ( ) );

// Swap resources between smart pointers
ptr.swap ( ptr2 );
```

参考网页 https://github.com/Microsoft/wil/wiki/RAII-resource-wrappers

# RAII resource wrappers

```
wil::unique_hwnd m_hMainWnd = nullptr;

HWND GetHandle ( ) const
{
     return m_hMainWnd.get ( );
}

static void OnNCCreate ( HWND const window  LPARAM const lparam ) noexcept
{
     auto cs = reinterpret_cast<CREATESTRUCT*>(lparam);
     auto that = static_cast<DesktopWindow*>(cs->lpCreateParams);
     WINRT_ASSERT ( that );
     WINRT_ASSERT ( !that->GetHandle ( ) );
     that->m_hMainWnd = wil::unique_hwnd ( window );
     SetWindowLongPtr ( window, GWLP_USERDATA,
          reinterpret_cast<LONG_PTR>(that) );
}
```

# x.2 PowerToys

## a set of utilities for power users to tune and streamline their Windows experience for greater productivity

- PowerToys是一组由微软首先在Windows 95中引入的实用型程序
- Windows XP发布后推出了PowerToys第二版，但自那之后便不再更新
- 17年之后，微软正在考虑向Windows 10用户推出PowerToys 3
  - ✓ PowerToys 3工具是开源的
  - ✓ 工具1：FancyZones
  - ✓ 工具2：Windows key shortcut guide

参考网页 https://github.com/microsoft/PowerToys

# x.2 PowerToys

## a set of utilities for power users to tune and streamline their Windows experience for greater productivity

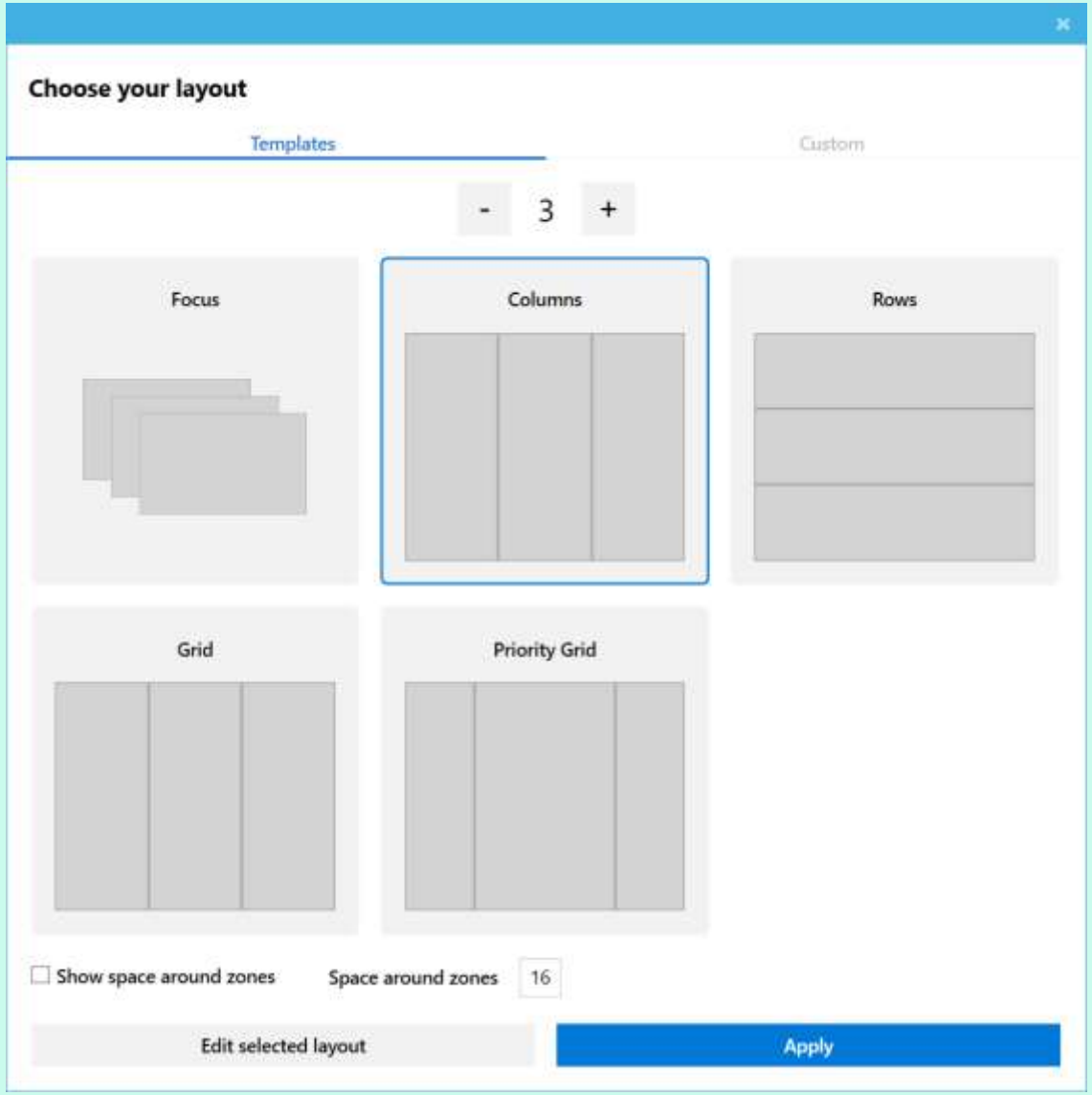
- PowerToys是一组由微软首先在Windows 95中引入的实用型程序
- Windows XP发布后推出了PowerToys第二版，但自那之后便不再更新
- 17年之后，微软正在考虑向Windows 10用户推出PowerToys 3
  - ✓ PowerToys 3工具是开源的
  - ✓ 工具1：FancyZones
  - ✓ 工具2：Windows key shortcut guide

Prerequisites to Build the Installer

- Install the WiX Toolset Visual Studio 2019 Extension.
- Install the WiX Toolset build tools from https://wixtoolset.org/releases/

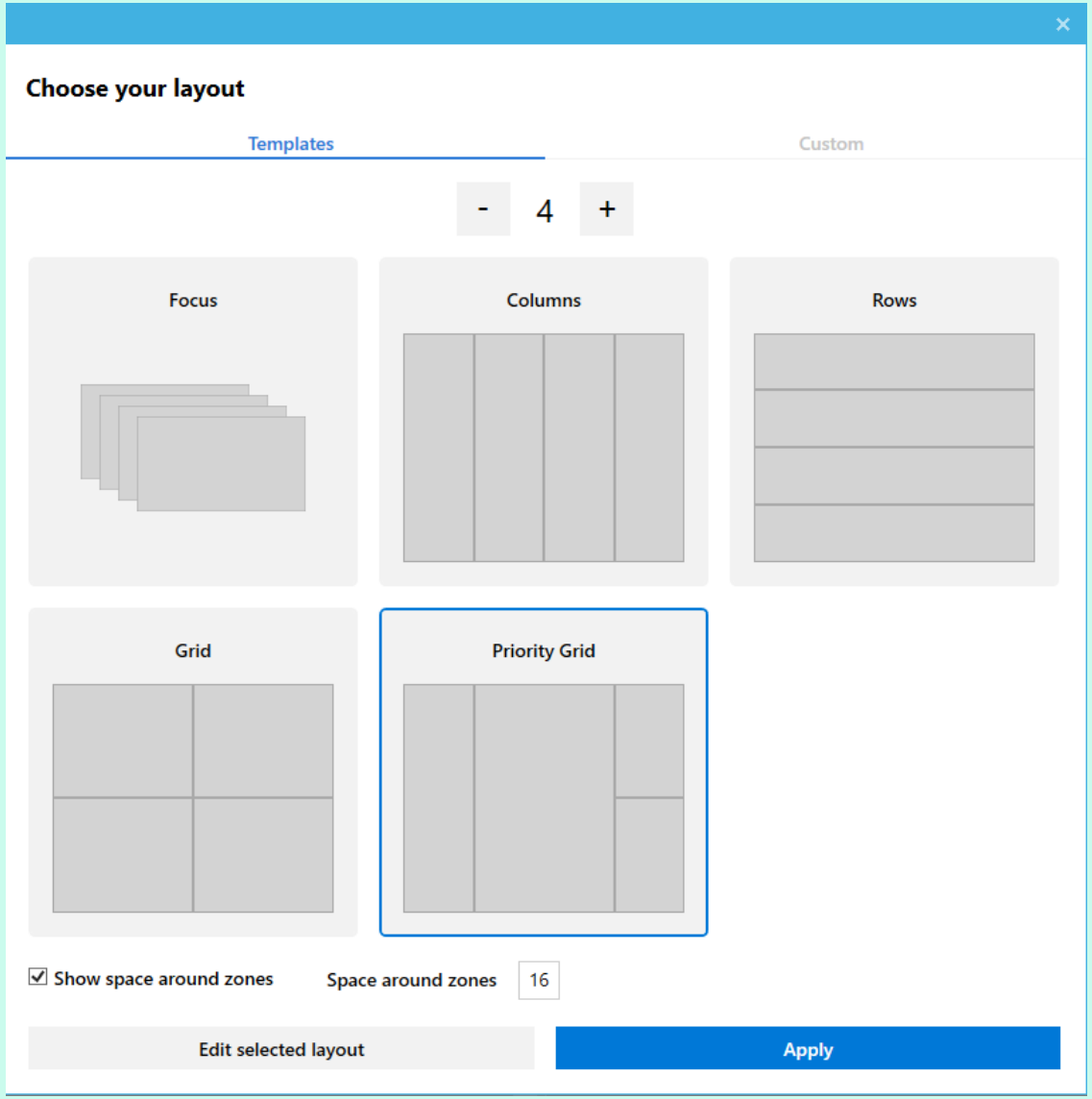
参考网页 https://github.com/microsoft/PowerToys

# FancyZones的功能



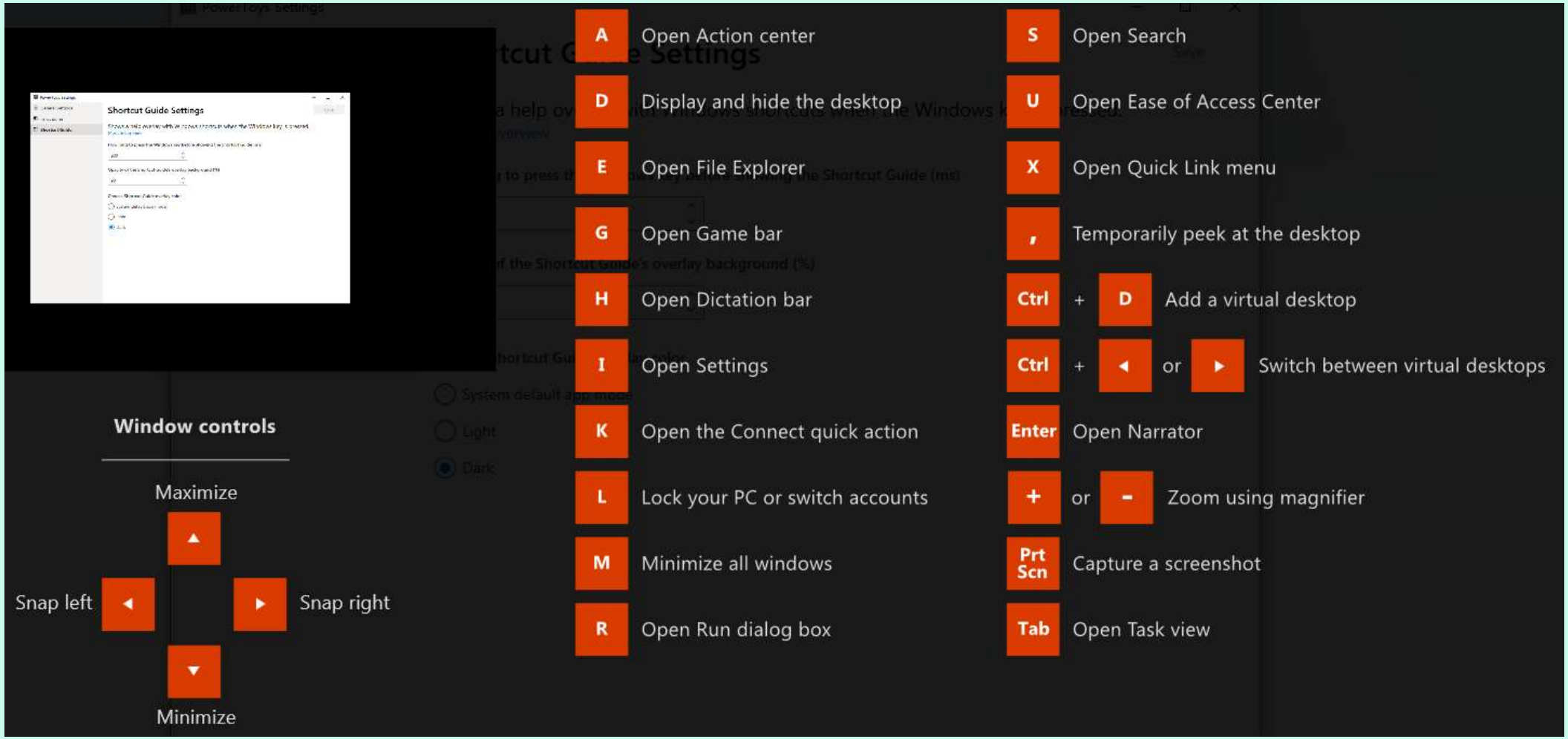

- 在显示桌面上用户自定义布局
- 将应用程序窗口对齐到定义好的布局
  - 拖动窗口时使用shift键

# FancyZones的功能




- 在显示桌面上用户自定义布局
- 将应用程序窗口对齐到定义好的布局
  - 拖动窗口时使用shift键

# Windows key shortcut guide的功能




- Windows键长按900ms

# x.3 coding style

You are defined by your character

Your character is defined by your coding style

- Not best-practices or requirements
  - ✓ deleting arrays with delete[]……
- typography

```
TextFileProcessor ::
TextFileProcessor( class ConstStringFinder& theConstStringFinder )

    : TextFileProcessor_Base( theConstStringFinder )

    , m_ThreadHandle  ( NULL )
    , m_startNLSearch (    0 )
    , m_endNLSearch   (    0 )
    , m_LineEndGetIdx (    0 )
    , m_LineEndPutIdx (    0 )
    , m_LineEnds      ( new const void*[ sc_LineEndSize ] )
{
    ;
}
```

# x.3 coding style

You are defined by your character

Your character is defined by your coding style

- Not best-practices or requirements
  - ✓ deleting arrays with delete[]……
- typography

```
TextFileProcessor ::
TextFileProcessor( class ConstStringFinder& theConstStringFinder )

    : TextFileProcessor_Base( theConstStringFinder )

    , m_ThreadHandle  ( NULL )
    , m_startNLSearch (    0 )
    , m_endNLSearch   (    0 )
    , m_LineEndGetIdx (    0 )
    , m_LineEndPutIdx (    0 )
    , m_LineEnds      ( new const void*[ sc_LineEndSize ] )
{
    ;
}
```

# Codes formatted in columns

seems to be a lot easier for us to read...

```
int myVar         = 1;    // comment 1
int myLongerVar   = 200;  // comment 2

MyStruct arrayOfMyStruct[] =
{
    // Name,                timeout,   valid
    {"A string",            1000,      true    },   // Comment 1
    {"Another string",      2000,      false   },   // Comment 2
    {"Yet another string",  11111000,  false   },   // Comment 3
    {NULL,                  5,         true    },   // Comment 4
};
```

## enumeration

```
// obsolete in C++11
namespace EntityType {
    enum Enum {
        Ground = 0,
        Human,
        Aerial,
        Total
    };
}

void foo(EntityType::Enum entityType)
{
    if (entityType == EntityType::Ground) {
        /*code*/
    }
}
```

## scoped enumeration

```
// Scoped enumeration (declared with enum class or enum struct)
enum class EntityType
{
    Ground = 0,
    Human,
    Aerial,
    Total
};

void foo ( EntityType entityType )
{
    if (entityType == EntityType::Ground )
    {
        /*code*/
    }
}
```

# Curiously Recurring Template Pattern

```
// pass a class as a template parameter to its base class
template<class Derived>
struct BaseCRTP { };

struct Example : BaseCRTP<Example> { };
```

```
// Within the base class, it can get ahold of the derived instance,
//   complete with the derived type, simply by casting
//   (either static_cast or dynamic_cast work)
template<class Derived>
struct BaseCRTP {
  void call_foo() {
    Derived& self = *static_cast<Derived*>(this);
    self.foo();
  }
};

struct Example : BaseCRTP<Example> {
  void foo() { cout << "foo()\n"; }
};
```

In effect, call_foo has been injected into the derived class with full access to the derived class's members.

# Pointer-to-Implementation



The implementation of foo is decoupled from its public interface, so that

- it can use members and types from other headers without requiring these dependencies to be present when the class is used, and
- the implementation can be modified without forcing a recompile of the code that uses the class.

```
// a forward declaration a pointer may be used
class private_foo;

// foo.h
class foo {
public:
  foo();
  ~foo();
  void bar();
private:
  private_foo* pImpl;
};
```

Users of the class simply include the header, which contains nothing specific about the implementation of the class. All implementation details are contained inside foo.cpp.

Someone called it "Handle Body"

```
// foo.cpp
#include whichever header defines the types T and U

// define the private implementation class
class private_foo {
public:
  void bar() { /*...*/ }

private:
  T member1;
  U member2;
};

// fill in the public interface function definitions:
foo::foo() : pImpl(new private_foo()) {}
foo::~foo() { delete pImpl; }
void foo::bar() { pImpl->bar(); }
```

# Compile-time polymorphism



contrast to runtime polymorphism

The implementation of foo is decoupled from its public interface, so that

- it can use members and types from other headers without requiring these dependencies to be present when the class is used, and
- the implementation can be modified without forcing a recompile of the code that uses the class.

Users of the class simply include the header, which contains nothing specific about the implementation of the class. All implementation details are contained inside foo.cpp.

Someone called it "Handle Body"

```
// a forward declaration a pointer may be used
class private_foo;

// foo.h
class foo {
public:
  foo();
  ~foo();
  void bar();
private:
  private_foo* pImpl;
};
```

```
// foo.cpp
#include whichever header defines the types T and U

// define the private implementation class
class private_foo {
public:
  void bar() { /*...*/ }

private:
  T member1;
  U member2;
};

// fill in the public interface function definitions:
foo::foo() : pImpl(new private_foo()) {}
foo::~foo() { delete pImpl; }
void foo::bar() { pImpl->bar(); }
```

# x.4 XAML Islands　　UWP XAML hosting API

Starting in Windows 10, version 1903, non-UWP desktop apps (including C++ Win32, WPF, and Windows Forms apps) can use the UWP XAML hosting API to host UWP controls in any UI element that is associated with a window handle (HWND)

- enables non-UWP desktop apps to use the UI features that are only available via UWP controls
  - ✓host UWP controls that use the Fluent Design System
  - ✓support Windows Ink

参考网页 https://docs.microsoft.com/en-us/windows/apps/desktop/modernize/using-the-xaml-hosting-api

# x.4 XAML Islands UWP XAML hosting API

Starting in Windows 10, version 1903, non-UWP desktop apps (including C++ Win32, WPF, and Windows Forms apps) can use the UWP XAML hosting API to host UWP controls in any UI element that is associated with a window handle (HWND)

- enables non-UWP desktop apps to use the UI features that are only available via UWP controls
  - ✓host UWP controls that use the Fluent Design System
  - ✓support Windows Ink

参考网页 https://docs.microsoft.com/en-us/windows/apps/desktop/modernize/using-the-xaml-hosting-api

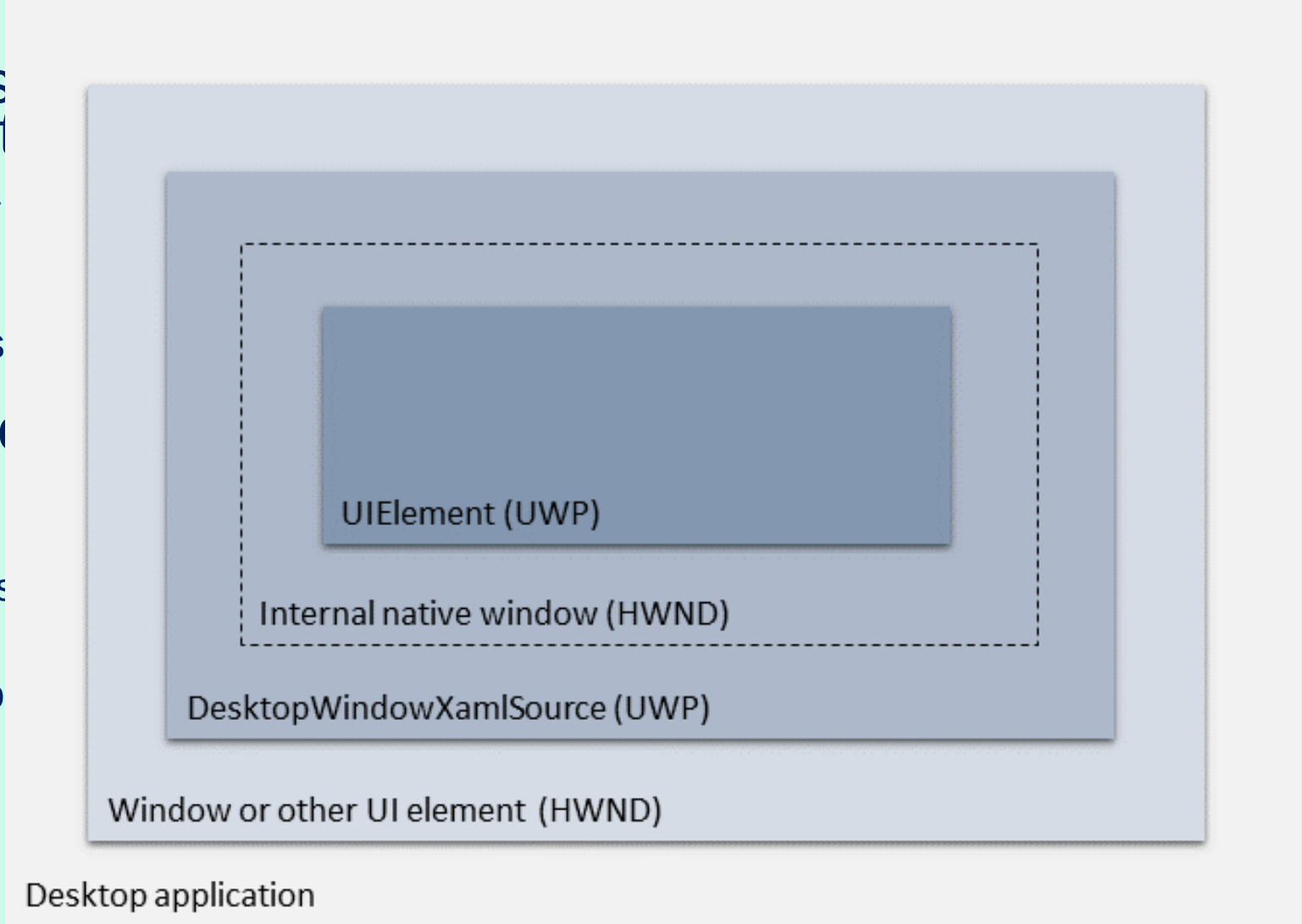
# 1. Prerequisites

XAML Islands require Windows 10, version 1903 (or later) and the corresponding build of the Windows SDK. To use XAML Islands in your C++ Win32 app, you must first set up your project.

1) Support for C++/WinRT
   - ✓ install the Microsoft.Windows.CppWinRT NuGet package in the project
2) Configure your project for app deployment
   - ✓ Install the Microsoft.Toolkit.Win32.UI.SDK package
3) Additional requirements for custom UWP controls
   - ✓ for a custom UWP control, additional instructions will be discussed later

# 2. Architecture of the API

1) At the bas
you want t
   - ✓ must have a
2) At the 2nd
   - ✓ provides the
   - ✓ Your code is
     UI element
3) the Deskt
creates a
control
   - ✓ you can acces
4) at the top
   - ✓ any UWP co
     controls



# 3. Host a standard UWP control

1) Windows 10, version 1903 SDK (version 10.0.18362) or a later

2) Retarget solution, select the 10.0.18362.0 or later

3) Install the Microsoft.Windows.CppWinRT NuGet package

4) Install the Microsoft.Toolkit.Win32.UI.SDK NuGet package
   - ✓ In the NuGet Package Manager window, make sure that Include prerelease is selected
   - ✓ install version v6.0.0-preview7 (or later)

# 4. Host a custom UWP control

what you need:

1) configure the project to meet the prerequisites for hosting XAML Islands

2) add reference to the project of custom control

3) access to an instance of the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class

# 4. Host a custom UWP control

general steps:

1) add a Blank UWP project
   - ✓ in the solution that contains C++ Win32 desktop project
2) add the project that contains the source code for the custom UWP XAML control
   - ✓ typically a UWP class library project
3) In the UWP app project, add a reference to the UWP class library project
4) In your C++ Win32 project, add a reference to the UWP app project and the UWP class library project
5) Assign an instance of the custom control to host to the Content property of the DesktopWindowXamlSource object in your code

https://github.com/marb2000/XamlIslands/tree/master/1903_Samples/CppWinRT_Win32_App

# THANK YOU !