Cross platform GUID association with types

CppCon 2014, Lightning talks

Vladimir Morozov

Microsoft

9/12/2014

Why do we need GUIDs for types?

- To implement IUnknown::QueryInterface
  - In Office we do not use RTTI
  - IUnknown::QueryInterface is used for dynamic cast
- Used by a lot of code in Office and COM in general
- Visual C++ has native support for GUIDs
  - To specify GUID: `__declspec(uuid(“38a24b6a-91d3-499e-9e4a-5cc6fc647331”))`
  - To get GUID for a type: `__uuidof(IWidget)`
- No support in C++ Standard

Typical cross platform GUID association

```
struct __declspec(uuid("4D675322-F6F5-4E85-94EF-2927DFAA1409"))
IWorkerCallback : IUnknown
{
    virtual void Invoke(IWorkerObject* pObj) = 0;
};

#ifdef __clang__
// cannot specialize template in a different namespace
}} // namespace Mso::Async

guid_of<Mso::Async::IWorkerCallback>::value =
    { 0x4D675322, 0xF6F5, 0x4E85, { 0x94, 0xEF, 0x29, 0x27, 0xDF, 0xAA, 0x14, 0x09 } };

namespace Mso { namespace Async {
#endif
```

```
#define __uuidof(type) guid_of<type>::value
```

Image source: http://commons.wikimedia.org/wiki/File:The_Puffer_Fish.jpg by Uploader1977

Can we do better?

Would be ideal solution

```
[uuid("4D675322-F6F5-4E85-94EF-2927DFAA1409“)]
struct IWorkerCallback : IUnknown
{
    virtual void Invoke(IWorkerObject* pObj) = 0;
};
```

But…

- Not a Standard C++
- Works only in Visual C++
- Increases instance size by a pointer size
- Do not use it!

Macro to the rescue!

```
STRUCT_GUID(IWorkerCallback, "4D675322-F6F5-4E85-94EF-2927DFAA1409")
struct IWorkerCallback : IUnknown
{
    virtual void Invoke(IWorkerObject* pObj) = 0;
};
```

- Keep struct/class keyword outside of macro for tooling support. E.g. Visual Studio, Sublime, etc.
- struct/class keyword inside of macro need to match the type’s struct/class keyword
  - Otherwise Visual C++ gives Level 1 warning
  - Important for Visual C++ ABI

Image source: http://commons.wikimedia.org/wiki/File:Clown.fish.arp.750pix.jpg by Adrian Pingstone

How can we implement the STRUCT_GUID macro?

Implementation for VC++

```
#define STRUCT_GUID(type, guidString) \
    struct __declspec(uuid(guidString)) type;
```

- Implementation is trivial
- __declspec(uuid) can be applied anywhere: type declaration, forward declaration, or redeclaration

Implementation for Clang

STRUCT_GUID is expanded to get_guid () function definition

```
#define STRUCT_GUID(type, guidString) \
    struct type; \
    extern "C++" \
    constexpr GUID get_guid(type*) noexcept { return str_to_guid(guidString); }
```

guid_of<T>::value is initialized with get_guid()

```
template <typename T> struct guid_of {
    static constexpr GUID value = get_guid(static_cast<T*>(nullptr)); }
```

__uuidof() returns guid_of<T>::value

```
#define __uuidof(type) guid_of<type>::value
```

Default get_guid() does a static_assert. To ensure that GUID is type specific.

```
template <typename T> constexpr GUID get_guid(T*) { static_assert(/*error*/); }
```

Implementation for Clang (cont.)

```
/// To represent a GUID string without curly braces: "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
typedef char GuidString[37];

/// Converts a hexadecimal ASCII character to an unsigned char.
const unsigned char H2U[256] = {
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0, 0, 0, 0, 0,
0, 10, 11, 12, 13, 14, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 10, 11, 12, 13, 14, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
};

/// Converts string to a GUID at compile time. Expected format: "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX“
constexpr GUID str_to_guid(const GuidString& g) noexcept
{
    return { static_cast<unsigned long>((H2U[g[0]] << 28) | (H2U[g[1]] << 24) | (H2U[g[2]] << 20) | (H2U[g[3]] << 16)
                                      | (H2U[g[4]] << 12) | (H2U[g[5]] << 8) | (H2U[g[6]] << 4) | H2U[g[7]]),
        static_cast<unsigned short>((H2U[g[9]] << 12) | (H2U[g[10]] << 8) | (H2U[g[11]] << 4) | H2U[g[12]]),
        static_cast<unsigned short>((H2U[g[14]] << 12) | (H2U[g[15]] << 8) | (H2U[g[16]] << 4) | H2U[g[17]]),
        {
            static_cast<unsigned char>((H2U[g[19]] << 4) | H2U[g[20]]),
            static_cast<unsigned char>((H2U[g[21]] << 4) | H2U[g[22]]),
            static_cast<unsigned char>((H2U[g[24]] << 4) | H2U[g[25]]),
            static_cast<unsigned char>((H2U[g[26]] << 4) | H2U[g[27]]),
            static_cast<unsigned char>((H2U[g[28]] << 4) | H2U[g[29]]),
            static_cast<unsigned char>((H2U[g[30]] << 4) | H2U[g[31]]),
            static_cast<unsigned char>((H2U[g[32]] << 4) | H2U[g[33]]),
            static_cast<unsigned char>((H2U[g[34]] << 4) | H2U[g[35]])
        }
    };
}
```

Implementation for Clang (cont.)

- str_to_guid() is a constexpr function evaluated at compile time
- get_guid() is called using ADL (Argument-dependent name lookup)
  - It must be defined in the same namespace as the type
  - If associated GUID is not found then you see a compilation error.
- Known issues:
  - Works only for C++11.
  - NDK linker error when __uuidof() is used as a template parameter for types:

    ```
    template< typename C, const IID* piid = &__uuidof( C ) > class QIPtr
    ```

    must be replaced with

    ```
    template< typename C, const IID* piid = nullptr > class QIPtr
    ```

    In the code instead of using piid we use `resolve_guid_ptr< C, piid >::guid`
    It uses two different specializations which either use piid or __uuidof(C) for the guid field initialization

Conclusion

- STRUCT_GUID allows to associate string based GUIDs with types for multiple platforms – VC++, Clang
- Backward compatibility with existing code – continue to use __uuidof()
- This technique can be used for any other custom type attributes
  - Custom attribute macro is expanded to a constexpr function
  - Have a special class to access the attribute value
  - Use the constexpr function to initialize the attribute value
  - Have a default template based function to be used when custom attribute is not defined.