boost::out_ptr – a scalable output pointer abstraction for smart pointers

IMPLEMENTING C++20'S P1132 – TARGETING BOOST 1.71

ThePhD

phdofthehouse@gmail.com; @thephantomderp

https://thephd.github.io/vendor/future_cxx/papers/d1132.html

https://github.com/ThePhD/out_ptr

PRELIMINARY CODE

Shared Code

From libavformat

```
#include <memory>
#include <avformat.h>

struct AVFormatContextDeleter {
                void operator() (AVFormatContext* c) noexcept {
                        avformat_close_input(&c);
                        avformat_free_context(c);
                }
};
typedef std::unique_ptr<AVFormatContext, AVFormatContextDeleter> AVFormatContext;
// Signature from libavformat:
// int avformat_open_input(AVFormatContext **ps, const char *url, AVInputFormat *fmt, AVDictionary **options);
```

BEFORE/AFTER

Current Code

```cpp
int main (int, char* argv[]) {
        AVFormatContext context(avformat_alloc_context());
        // ...
        // used, need to reopen
        AVFormatContext* raw_context = context.release();
        if (avformat_open_input(&raw_context,
                argv[0], nullptr, nullptr) != 0) {
                std::stringstream ss;
                ss << "ffmpeg_image_loader could not open file '"
                        << path << "'";
                throw FFmpegInputException(ss.str().c_str());
        }
        context.reset(raw_context);

        // ... off to the races !

        return 0;
}
```

With Proposal

```cpp
int main (int, char* argv[]) {
        AVFormatContext context(avformat_alloc_context());
        // ...
        // used, need to reopen

        if (avformat_open_input(std::inout_ptr(context),
                argv[0], nullptr, nullptr) != 0) {
                std::stringstream ss;
                ss << "ffmpeg_image_loader could not open file '"
                        << argv[0] << "'";
                throw FFmpegInputException(ss.str().c_str());
        }


        // ... off to the races!

        return 0;
}
```

MOTIVATION I – STANDARDIZE EXISTING PRACTICE

- The Most Existing-iest Practice there has ever Been
  - Microsoft: CComPtr, circa before I knew what a computer was
  - Fortune 100 Companies, small hobby developers, game studios, etc. have this abstraction (including VMWare, co-authors)
  - `std::retain_ptr` (http://wg21.link/p0468) was going to overload operator& - this paper solves their problem
  - Microsoft: got a better idea and have WRL::ComPtrRef instead of overloading unary & which behaves exactly like `std::out_ptr`/`std::inout_ptr`

MOTIVATION II

- Remove the destructive urge people have to overload unary operator& on smart pointers
  - Let them not fall as Microsoft has in ye olden days: `CComPtr`
  - https://groups.google.com/a/isocpp.org/forum/#!topic/std-proposals/8MQhnL9rXBI

- Remove the only reason anyone wants to overload unary operator&
  - Pave the way for `std::addressof` to become obsolete
  - Prevent defensive programming by library programmers

DESIGN

-
  ```
  namespace std {
  	template <class Pointer, class Smart, class... Args>
  	out_ptr_t<Smart, Pointer, Args...>
  	out_ptr(Smart& s, Args&&... args) noexcept;

  	template <class Smart, class... Args>
  	out_ptr_t<Smart, POINTER_OF(Smart), Args...>
  	out_ptr(Smart& s, Args&&... args) noexcept;

  	template <class Pointer, class Smart, class... Args>
  	inout_ptr_t<Smart, Pointer, Args...>
  	inout_ptr(Smart& s, Args&&... args) noexcept;

  	template <class Smart, class... Args>
  	inout_ptr_t<Smart, POINTER_OF(Smart), Args...>
  	inout_ptr(Smart& s, Args&&... args) noexcept;
  }
  ```

DESIGN: WELL-DEFINED AS ARGUMENT

```
std::unique_ptr<int, resource_deleter> resource(nullptr);

error_num err = c_api_create_handle(24, std::out_ptr(resource));
if (err == C_API_ERROR_CONDITION) {
     // handle errors
}

// resource.get() the out-value from the C API function
```

DESIGN: SHARED_PTR SAFETY

- When used with std::shared_ptr, it **requires** that additional arguments are passed so it can reset the deleter too:
-
```
std::shared_ptr<int> resource(nullptr);

error_num err = c_api_create_handle(42, std::out_ptr(resource));
// ERROR: deleter was changed
// to an equivalent of std::default_delete!!
```

DESIGN: CASTING SUPPORT I

- Consider the following function:

-
  ```
  HRESULT EnumAdapterByGpuPreference(UINT Adapter,  DXGI_GPU_PREFERENCE GpuPreference,
        REFIID riid, void** ppvAdapter);
  ```
- The desired type is `IDXGIAdapter`, but it takes `void**`
  - Very common in "polymorphic" C APIs with single stable allocation function and an integer/enumeration that switches on allocation type

DESIGN: CASTING SUPPORT II

- Converts implicitly to `void**`

-
  ```
  HRESULT result = dxgi_factory.EnumAdapterByGpuPreference(0,
        DXGI_GPU_PREFERENCE_MINIMUM_POWER,  IID_IDXGIAdapter,
        std::out_ptr<void*>(adapter));

  if (FAILED(result)) {
        // handle errors
  }
  // adapter.get() contains strongly-typed pointer
  ```

DESIGN: CASTING SUPPORT III

- Also can static_cast to the proper type if you provide an explicit casting parameter

-
  ```
  std::unique_ptr<int, fd_deleter> my_unique_fd;
  auto err = fopen_s( std::out_ptr<FILE*>(my_unique_fd), "prod.csv", "rb" );
  // check err, then work with raw fd
  ```

- Full example available here: https://github.com/ThePhD/out_ptr/blob/master/examples/source/std.custom_unique_ptr.cpp

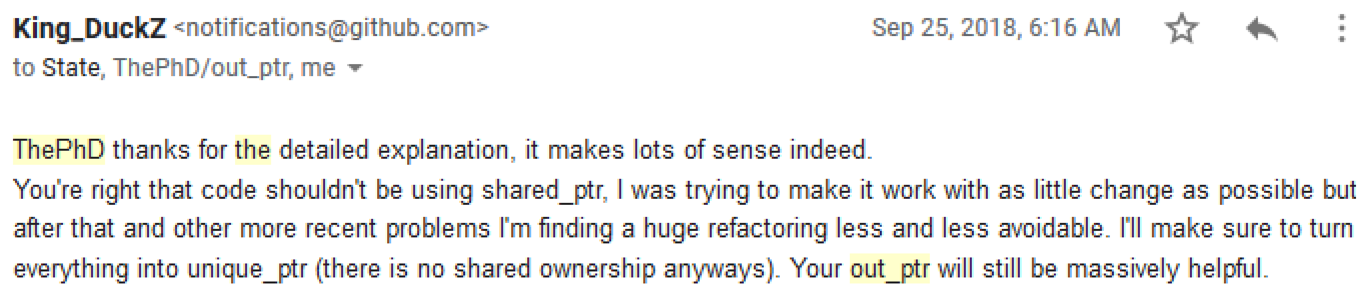
DESIGN: OVERRIDABLE AND SCALABLE

- Works with non-standard pointers just as much as standards pointers through well-defined customization points
  - `class std::out_ptr_t<Smart, Pointer,  Args…>`
  - `class std::inout_ptr_t<Smart, Pointer,  Args…>`

IMPLEMENTATION EXPERIENCE

- Lots of it
  - VMWare for a long time
  - My own code for a long time
  - Others for a long time: https://groups.google.com/a/isocpp.org/forum/#!topic/std-proposals/8MQhnL9rXBI

- Public implementation: https://github.com/ThePhD/out_ptr
  - Already in use from others



King_DuckZ <notifications@github.com>
Sep 25, 2018, 6:16 AM
to State, ThePhD/out_ptr, me

ThePhD thanks for the detailed explanation, it makes lots of sense indeed.
You're right that code shouldn't be using shared_ptr, I was trying to make it work with as little change as possible but after that and other more recent problems I'm finding a huge refactoring less and less avoidable. I'll make sure to turn everything into unique_ptr (there is no shared ownership anyways). Your out_ptr will still be massively helpful.

FUN FOR THE WEEK

- Polishing…
  - implementation; C++11 only
  - examples – simple and complex
- Writing….
  - formal documentation
  - colloquial documentation
- At: https://github.com/ThePhD/out_ptr/issues

THANK YOU!

- For listening,
  - And for (eventual) contributions!

- Find me at
  - the #include<c++> Discord, C++Now 2019 Channel (*cppnow2019*): https://discord.gg/ZPErMGW
  - the CppLang Slack, Library in a Week Channel (*cppnow-liaw*): https://cpplang.slack.com/archives/CAJQY2YBS/p1557203114009100