Using Exceptions

Philosophy

Introducing exceptions to an existing non-exception-based codebase can be perilous. The console was originally written in C at a time when C++ was relatively unused in the Windows operating system. As part of our project to modernize the Windows console, we converted to use C++, but still had an aversion to using exception-based error handling in our code for fear that it might introduce unexpected failures. However, the STL and other libraries like it are so useful that sometimes it's significantly simpler to use them. Given that, we have a set of rules that we follow when considering exception use.

Rules

1. **DO NOT** allow exceptions to leak out of new code into old code
2. **DO** use `NTSTATUS` or `HRESULT` as return values as appropriate (`HRESULT` is preferred)
3. **DO** encapsulate all exception behaviors within implementing classes
4. **DO NOT** introduce modern exception throwing code into old code. Instead, refactor as needed to allow encapsulation or use non-exception based code
5. **DO** use WIL as an alternative for non-throwing modern facilities (e.g. `wil::unique_ptr<>`)

Examples

Encapsulating exception behaviors in a class

```
class ExceptionsDoNotLeak
{
    public:
    HRESULT SomePublicFunction();
    int iPublic;

    private:
    void _SomePrivateFunction();
    int _iPrivate;
};
```

Using WIL for non-throwing modern facilities

TODO