Extending the `noexcept` Policy, Rev0

The way, we currently use `noexcept` in the C++ standard library following the rules of N3279:

- No library destructor should throw. They shall use the implicitly supplied (nonthrowing) exception specification.
- Each library function having a `wide` contract, that the LWG agree cannot throw, should be marked as unconditionally `noexcept`.
- If a library swap function, move-constructor, or move-assignment operator is conditionally-wide (i.e. can be proven to not throw by applying the `noexcept` operator) then it should be marked as conditionally `noexcept`. *No other function should use a conditional `noexcept` specification.*
- Library functions designed for compatibility with “C” code (such as the atomics facility), may be marked as unconditionally `noexcept`.

However, more than once we now had the case that we had types wrapped or extended in a way that the original behavior of the type should only changed as small as possible to make the wrapping/extension as transparent as possible.

Examples are:
- `std::atomic<>` (see P0883)
- `std::function_ref` (proposed in P0792)

So, I propose the following new rules:

a) No library destructor should throw. They shall use the implicitly supplied (nonthrowing) exception specification.

b) Each library function having a `wide` contract (i.e., does not specify undefined behavior due to a precondition) that the LWG agree cannot throw, should be marked as unconditionally `noexcept`.

c) If a library swap function, move-constructor, or move-assignment operator is conditionally-wide (i.e. can be proven to not throw by applying the `noexcept` operator) then it should be marked as conditionally `noexcept`.

d) If a library type has wrapping semantics to transparently provide the same behavior as the underlying type, then default constructor, copy constructor, and copy-assignment operator should be marked as conditionally `noexcept` the underlying exception specification still holds.

e) No other function should use a conditional `noexcept` specification.

f) Library functions designed for compatibility with “C” code (such as the atomics facility), may be marked as unconditionally `noexcept`.

Acknowledgements

Thanks to a lot of people who discussed the issue, proposed information and possible wording.

Feature Test Macro

This is a pure design guideline for the C++ library and needs no feature macro.