**std::function move operations should be noexcept**

Document number: P0771R0
Date: 2017-10-16
Project: Programming Language C++, Library Working Group
Reply-to: Nevin “☺” Liber, nliber@ocient.com or mailto:nevin@cplusplusguy.com

## Table of Contents

- Introduction ...................................................................................................................... 1
- Motivation and Scope ...................................................................................................... 1
- Impact on the Standard ................................................................................................. 2
- Design Decisions ............................................................................................................ 2
- Technical Specifications ............................................................................................... 2
- Acknowledgements ....................................................................................................... 3
- References ....................................................................................................................... 3

## Introduction

The move constructor and move assignment operator for `std::function` should be noexcept.

## Motivation and Scope

It is highly desirable to have noexcept move operations, especially when it does not impose an undue burden on implementers or a high cost for users.

The other type-erased standard libraries `any` and `shared_ptr` already require this. `function` is very similar to `any` in that both encourage the small object optimization.

It appears that `function` is required to use the small object optimization, at least to hold a `reference_wrapper` object or function pointer [func.wrap.func.con#4], and this proposal is compatible with that.

Both libstdc++ and libc++ already implement this.
Impact on the Standard

Impact on the standard is minor. The declarations for the move constructor and move assignment operator for `function` have to have `noexcept` added, and the throws clause for the move constructor has to be deleted.

Design Decisions

A possible implementation technique: if the object either is too big to fit inside the small object optimization space inside `function` or the object has a `noexcept(false)` move constructor or `noexcept(false)` assignment operator, then store it in the heap; otherwise, store it in the small object optimization space.

Because default construction and `swap` are already `noexcept`, it is very likely that a currently conforming implementation of `function` already does something like this under the covers, even if they don’t declare their move constructor and move assignment operator as `noexcept`.

Technical Specifications

Changes relative to `n4687`:

```cpp
[func.wrap.func]

function() noexcept;
function(nullptr_t) noexcept;
function(const function&);
function(function&&) noexcept;
template<class F> function(F);

function& operator=(const function&);
function& operator=(function&&) noexcept;
function& operator=(nullptr_t) noexcept;
template<class F> function& operator=(F&&);
template<class F> function& operator=(reference_wrapper<F>) noexcept;

[func.wrap.func.con]

function(function&& f) noexcept;
Postconditions: If `f` has no target; otherwise, the target of `*this` is equivalent to the target of `f` before the construction, and `f` is in a valid state with an unspecified value.
Throws: Shall not throw exceptions if `f`'s target is a specialization of `reference_wrapper` or a function pointer. Otherwise, may throw `bad_alloc` or any exception thrown by the copy or move constructor of the stored callable object. [ Note: Implementations should avoid the use of dynamically allocated memory for small callable objects, for example, where `f`'s target is an object holding only a pointer or reference to an object and a member function pointer. —end note ]

function& operator=(function&& f) noexcept;
Effects: Replaces the target of `*this` with the target of `f`.
Returns: `*this`.
```
Acknowledgements
Special thanks to Ion Gaztañaga, Gabriel Dos Reis, Pete Becker, Bjarne Stroustrup, Jonathan Wakely, and Stephan T. Lavavej for the discussion on this way back when; Howard Hinnant for that as well as answering a theoretical design question on function, Billy O’Neal for pointing out on an LEWG thread that the small object optimization is required (as well as Stephan and Billy informing me how their version of function is implemented), and Geoffrey Romer for recently implicitly reminding me that no one had actually submitted a paper on this yet. Thank them or blame me for the content of this paper.

References
n4687 - Working Draft, Standard for Programming Language C++, Richard Smith
std_function.h, libstdc++ (gcc)
functional – libc++ (clang)