|  |  |
| --- | --- |
| Document number: | *Nnnnn=yy-nnnn* |
| Date: | 2017-09-04 |
| Project: | Programming Language C++, Library Working Group |
| Reply-to: | Ben Craig <ben dot craig at gmail dot com> |

**I. Introduction**

C++ defines a “freestanding” implementation that is intended to be used on systems with limited operating system support. The standard has a small set of headers and functionality that are required to be present in a freestanding implementation.

Some features are problematic in system programming. Exceptions, RTTI, heap management, global state, thread-local storage, and floating point support are all problematic.

This proposal seeks to expand the list of headers and functionality to include all parts of the hosted C++ implementation that don’t use problematic features, while simultaneously attempting to remove headers and functionality from the freestanding requirements that do require the problematic features.

This is intended as an early paper to seek general direction and opinions.

**II. Motivation**

Systems programmers want to sort things. They want to use move semantics. They may even want to bundle the arguments of a variadic template function into a tuple. These are all reasonable things to do on a system with no operating system and kilobytes of storage. The C++ standard even has reasonable specifications for these operations in the context of a tiny, OS-less system. However, systems programmers must currently rely on either hand-rolled code or implementer extensions in order to get these facilities.

Systems programmers don’t have a guide as to what C++ library facilities will work without trying them. The standard says atomic\_load will work; memcpy will probably work; but will stable\_sort? Standardizing the subset of implementable C++ that is usable in a freestanding environment would provide clarity here, and help to educate systems programmers.

Why is this important? What kinds of problems does it address? What is the intended user community? What level of programmers (novice, experienced, expert) is it intended to support? What existing practice is it based on? How widespread is its use? How long has it been in use? Is there a reference implementation and test suite available for inspection?

**III. Scope**

**IV. Impact On the Standard**

Rather than list all of the facilities available to a freestanding implementation in one place, as is currently done in [compliance], the standard would tag each header, class, or function that is available in a freestanding implementation. I expect this to be a large number of small edits, but the edits wouldn’t have difficult to understand ramifications throughout the standard.

**V. Design Decisions**

Why did you choose the specific design that you did? What alternatives did you consider, and what are the tradeoffs? What are the consequences of your choice, for users and implementers? What decisions are left up to implementers? If there are any similar libraries in use, how do their design decisions compare to yours?

**VI. Technical Specifications**

<cstddef>

<cstdlib>

-atexit

-at\_quick\_exit

-exit

-quick\_exit

**VII. Acknowledgements**

Similar work was done in the C++11 timeframe by Lawrence Crowl and Alberto Ganesh Barbati in N3256.

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3256.html