A Panoply of TS

Alisdair Meredith, Library Working Group Chair

A Quick Tour of the Sausage Factory

- A Committee convened under ISO/IEC so multiple companies can co-operate and define the language
- Google ‘wg21’ to find the official site (usually top hit)
- Community site for publicizing activity, sharing the latest information, and encouraging wider involvement
  - isocpp.org

A Quick Tour of the Sausage Factory

- 1998 first standard (7 years, 21 meetings)
- 2003 TC1 (5 years, 10 meetings)
- 2011 second standard (8 years, 21 meetings)
- 2014 revised standard (3 years, 6 meetings)
- 2017 projected next standard (3 years, 6 meetings?)

Library Working Group

(Revised organization)

Library

- Review all standard wording
- Responsible for library consistency
- Decide when a paper is ready to advance

Evolution

- Review all new proposals
- Responsible for Library direction
- Decide when a library is feature complete

Study Groups

- SG1 Concurrency
- SG2 Modules
- SG3 Filesystem
- SG4 Networking
- SG5 Transactional memory
- SG6 Numerics
- SG7 Reflection
- SG8 Concepts
- SG9 Ranges
- SG10 Portability
- SG11 Database access
- SG12 Undefined behavior
- SG 13 Graphical I/O

Work carried over from C++14

- moving to a separate TS
  - array of runtime bounds — array extensions
  - `std::dynarray<T>`
  - `std::optional<T>` — library fundamentals

Study Groups

- SG1 Concurrency
- SG2 Modules
- SG3 Filesystem
- SG4 Networking
- SG5 Transactional memory
- SG6 Numerics
- SG7 Reflection
- SG8 Concepts
- SG9 Ranges
- SG10 Portability
- ~~SG11 Database access~~
- SG12 Undefined behavior
- SG 13 Graphical I/O

Active TS

(project filed with ISO)

Filesystem	Ballot resolution
Concepts Lite	Feature complete draft
Library Fundamentals	Feature complete draft
Array Extensions	New work item
Concurrency extensions	Project with working draft
Parallel algorithms	Project with working draft
Transactional Memory	New Work Item ballot
Networking	Active project

SG1 Concurrency

- Actively putting work into standards
  - shared locks in C++14
  - Ongoing clause 30 work for C++17
- Concurrency TS
  - Extensions for distinct tasks running concurrently
- Parallelism TS
  - Parallel algorithms to accelerate computation of a single result/effect

SG2 Modules

- Work is slow, not much visible yet
- Prototyping work happening in the Clang project
- Essentially a modern replacement for `#include`
  - fine-grained precompiled headers
  - Macro insulation
  - `export` done right?

SG3 Filesystem

- First filesystem TS in ballot resolution
  - Boost filesystem library
  - Revised by committee (minor)
- Plans for further TSes
  - Enterprise scale (clustering) filesystems
  - Secure APIs (e.g., man in the middle attacks)

SG4 Networking

- Communication layer to talk to the internet and beyond
- Reviewing from first principles, rather than adopt Boost asio
- First component ships in library fundamentals
  - Network byte order APIs
- Planning incremental TS, one per year
  - URI class
  - IP address class

SG5 Transactional Memory

- Outgrowth of SG1 (Concurrency) for specialist domain
- Transactional Memory TS New Work Item
  - Working draft is language complete
  - Library features imminent
- Simpler way of writing correct concurrent code without explicit locking
- *Potentially* expensive without hardware acceleration

SG6 Numerics

- Group of domain experts to manage numeric library proposals
- Imminent work on decimal floating point
  - Renew TS or move feature to C++17?
- Numeric TS imminent, or addition to Fundamentals 2
  - unbounded integers and rational numbers
  - fixed point binary
  - random number extensions

SG7 Reflection

- Incubator for work on both language and library features
- Both compile-time and runtime reflection in scope
- No concrete plans at the moment, but many interesting ideas
- Tricky balance with ‘do not pay for what you do not use’ principle

SG8 Concepts

- Concepts-lite TS (language features only)
  - Allows declarations with a simple constraints language
  - Much simpler/more efficient than SFINAE hacks
  - Extensible to checked bodies in the future?
- Concept-enabled library to follow
  - Enhance current library, or new (parallel) compatible-as-possible library?

SG9 Ranges

- No shortage of ideas
- No clear consensus on direction
  - Are ranges an atomic unit, or an iterable entity?
  - What is the result of `std::sort(range)`?
  - If iterable, can an input range be expressed better than an iterator pair?
- Clear belief this is important for a simpler library
- Tied to range concepts?

SG10 Portability

- Address issues to ease portable code
- Typical problems for users discovering and acting on implementation specific (potentially unspecified) behavior
- First output: feature detection standing document
  - feature-detection macros for new language features
  - Will not standardize a convention for non-conformance!

SG11 Database Access

- Broad interest outside the committee
- Minimal activity within the committee
  - lack of experts
  - too busy with other projects - spreading too thin
- Chair no longer able to attend, so disbanding the group
- Proposals still welcome under regular library working group process

SG12 Undefined Behavior

- Concern that there is too much un-necessary undefined behavior in the standard
- Concern that a strong attempt to eliminate all undefined behavior will eliminate important freedoms and optimization opportunities for compilers
- Early discussions, no clear product in short term
- Likely to visit every undefined behavior in Core clauses, and potentially address on case-by case basis with papers for C++17 and beyond

SG13 Graphics and I/O

- Newest study group, just forming and setting agenda
- Original idea was for basic graphics primitives
- Quickly extended to I/O such as mouse cursors
- Ambitious area, so trying to be conservative in scope to be sure to deliver something practical early to build on.

Active TS

(project filed with ISO)

Filesystem	Ballot resolution
Concepts Lite	Feature complete draft
Library Fundamentals	Feature complete draft
Array Extensions	New work item
Parallel algorithms	Project with working draft
Concurrency extensions	Project with working draft
Transactional Memory	New Work Item ballot
Networking	Active project

Active TS

(project filed with ISO)

- Filesystem N3940
- Concepts Lite N3929
- Library Fundamentals N3908
- Array Extensions N3820
- Parallel algorithms N3960
- Concurrency extensions N3875 N3857
- Transactional Memory N3919
- Networking (no working paper)

Estimated Publication

(my own private [optimistic!] guess)

Filesystem	Late 2014
Concepts Lite	Early 2015
Library Fundamentals	Early 2015
Array Extensions	Late 2015
Parallel algorithms	Late 2015
Concurrency extensions	Early 2016
Transactional Memory	Late 2015
Library Fundamentals 2	*Early 2016*

Basics of a TS

- Non-normative extensions to the standard
- Slightly faster ISO process intended for experimantal feature : one fewer ballot stage
- All headers will have the `<experimental/*>` prefix
- All new libraries live in namespace `std::experimental`
- All new libraries live in a further `inline` version namespace

Filesystem TS

- Library baked for many years as Boost::filesystem
- Standardization turned up many issues in the formal specification, not so much in the design
- non-template `filepath` class handles multiple character types
- Functions and iterators to navigate a filesystem
- Functions to create and manipulate files
- Specification based on POSIX standard semantics

Library Fundamentals TS

(new libraries)

- `optional`
- `any`
- `string_view`
- polymorphic allocators
- Boyer-Moore searching
- Network byte order

Library Fundamentals TS

(enhanced libraries)

- `shared_ptr` support for arrays
- standard containers support polymorphic allocators
- `sample` algorithm built on random number facility
- `apply` example becomes normative (forward a `tuple`)
- SFINAE friendly `iterator_traits`
- SFINAE friendly `common_type`

Library Fundamentals 2 TS

- enhanced assert facility
- more to follow...

Array Extensions TS

- Cross-functional TS
  - Core and Library features
- Features removed from C++14, to ship as a unit
  - Fear that the library feature was not implementable
  - revisit the design
- Additional ideas in the area of arrays

Runtime Arrays

- language and library facilities for the same purpose
  - language facility guaranteed on stack
  - library optimistic, relies on unspecified compiler optimizations kicking in
- language facility subject to several unique restrictions, library facility is a regular class template
- only the library facility “knows” its size

Runtime Arrays : language

(array of runtime bound)

- Like an array with restrictions
- Cannot copy the array by value
- Cannot find the type, e.g., `declype` or `typeid`
- Cannot find the size of the object, e.g., `sizeof`
- Cannot bind a reference to such an array
- Cannot take the address of such an object
- Cannot be used as data member of a class

Runtime Arrays : library

(`std::dynarray<T>`)

- Like `std::array` with size supplied at construction
- Allocate on stack only if the `dynarray` is a local variable on the stack
- Class is neither copyable nor movable, otherwise use like a regular object
- Can be used as data member in other structures and captured by lambda expressions
- Supports `begin/end` and `for` loops

Array Extensions: new ideas

- `std::make_array`
  - Currenty returns `std::array`
  - Why not `std::dynarray`?
- `array_view`?

Concepts Lite TS

- Inspired by C++0x concepts
- Does *not* check function bodies against concepts
- Constraints are `constexpr` predicate functions
- Concepts are collections of constraints
  - callable as `constexpr` predicate function
- Concept matching is always inferred - no concept_map
- Concept-overloading is supported

Concepts Lite TS

- terse syntax might look like a regular function
  - `template` may be implicit
  - easiest way to constrain polymorphic lambda (C++14)

Concepts Lite TS

(example syntax)

```
template<typename T>
concept bool Constexpr_equality_comparable()
{
    requires (T a, T b) {
        { a == b } constexpr -> bool;
        { a != b } constexpr -> bool;
    };
}
```

Concepts Lite TS

(example syntax)

```
template<Object T, Allocator A>
class vector
{
    vector(vector&& x)
        requires Movable<T>();
    vector(const vector& x) // Copy constructor
        requires Copyable<T>();
    // Iterator range constructors
    template<Input_iterator I>
    vector(I first, I last);
    template<Forward_iterator I>
    vector(I first, I last);
};
```

Parallelism Extensions TS

- Adds an `execution_policy` overload to most functions in the `<algorithm>` header
  - (no parallel random numbers yet, maybe others)
- Adds a couple of algorithms popular in parallel computing
  - `exclusive_scan`
  - `inclusive_scan`

Concurrency TS

- Currently based on two proposals, may fill out more
- Many details still in flux
- N3875 Executors and Schedulers
- N3857 Enhancements built on `std::future`
  - Microsoft `then`

Transactional Memory TS

- A simpler model for writing threaded code
- Assume no data races, write code without locks
- If a race occurs, the ‘transaction’ rolls back
- higher abstraction pays a cost - hoped to be small
- hardware support more widely available
  - e.g., in the latest generation of Intel Haswell CPUs

Networking TS

- Determined ASIO was too big to swallow all at once
- Desire to revisit some design aspects, so not tied totally to perspective of a single library
- Ship a TS every year
- Initial concents:
  - Network byte order
  - IP address (v4, v6, union of both, all three?)
  - URI equivalent of `filepath`

Decimal Arithmetic

- Originally proposed by IBM who have hardware support
- Important for accuracy where decimal quantities are common e.g., finance
- TRs for C/C++ based on then-current standards
- Proposed based on C++14
  - literals, constexpr, etc.

C++17
(my predictions)

- Mostly defect fixes for now
- expect to adopt some features of the TS presented
- other work will land directly, mostly language features
- Fundamentals and array extensions strong candidates
- Concepts if the library is ready too
  - perhaps C++20 more likely
  - Concurrency may depend on vendor experience