C++0x Standard Library

A Progress Report

Alisdair Meredith
Work List at a Glance

- Adopt TR1 components
- Embrace new language features
- Language support
- New library components
- Clean-up and maintenance
- Parallel development for TR2
Adopt TR1 components
Original Contents

- Utilities for general users
  - General purpose smart pointer
  - More containers: array and hashed containers
  - Function wrappers and binders
  - C99 Standard Library

- Tools for library writers
  - Tuples
  - Type traits

- Domain specific
  - Regular Expressions
  - Random number generator
  - Engineering/scientific math functions
Adopt TR1 components
Revised Contents

- Utilities for general users
  - General purpose smart pointer
  - More containers: array and hashed containers
  - Function wrappers and binders
  - C99 Standard Library

- Tools for library writers
  - Tuples
  - Type traits require compiler support

- Domain specific
  - Regular Expressions
  - Random number generator
  - Engineering/scientific math functions
Embrace new language features

- Concepts
- Rvalue references
  - Move semantics
  - ‘Perfect forwarding’
- Variadic templates
- Sequence constructors
- constexpr
- decltype
- long long and extended integer types
- Unicode character types
- New memory model / concurrency guarantees
Embrace new language features
(by example)

```cpp
#include <iostream>
#include <string>
#include <vector>

// 'strong' typedefs
struct user_name : std::string { using string::string; };
struct address_book : std::vector< user_name >
{ using vector::vector; };

address_book userlist() {
    address_book result = // sequence construction
    { "john smith", "jane doe", "a.n.other" };
    return result;        // return by move
}

int main() {
    for( auto& name : userlist() ) {
        std::cout << "user: " << name << std::endl;
    }
}
```
Language support

- **Initializer_list**
  - Required for sequence constructors

- **Concept ‘for’**
  - Required for new for loop syntax

- **Atomic types**
  - Library interface preferred to new keywords
New library components

- Consistent system/OS error reporting
  - API for retrieving and formatting system errors
  - Exception class for reporting system errors

- unique_ptr
  - Unique ownership of the pointer
  - Movable
  - Usable in standard containers
New library components

Threads

- Portable thread-launching API
  - Copyable vs. Movable
  - Cancellation
  - Propagating exceptions

- Basic synchronization primitives
  - mutex
  - condition variable
  - Upgradeable locks (TR2?)

- Futures (TR2?)
clean-up and maintenance
Update existing components

- Consistent use of std::string / const char *
- Consistent container interfaces
  - front()/back() for basic_string
  - data() for vector
  - at() for map
- Constant iterator functions cbegin()/cend()
  
  ```cpp
  for( auto it = container.cbegin();
    it != container.cend();
    ++it )
  { ... }
  ```

- Enhanced or missing algorithms
  - min/max/minmax/minmax_element/mean/variance
- Enhanced queries of numeric_limits for floating point display information
- Enhanced allocators support inplace resize
clean-up and maintenance

Deprecation

- auto_ptr
  - Replaced by unique_ptr
- C++98 function binders
  - Replaced by tr1 bind
- unary_function/binary_function
  - No longer necessary with tr1 bind / decltype
- vector<bool>
  - Underspecified premature optimisation
  - Replacement expected
clean-up and maintenance
Header cleanup

- Simplify ‘header dance’ with C headers
- Reduce un-necessary dependencies
e.g. `<exception>` should not require `<string>`
- Mandate useful dependencies

```cpp
#include <iostream>
int main()
{ std::cout << “hello C++09” << std::endl;
}
```
Parallel development for TR2

- Filesystem (accepted)
- Date and time support
- Network file support
- Range-types / container algorithms
- String algorithms
- Optional/Nullable values
- Type-safe 'any' class
- Range-checked numeric-casts
- 'Lexical' casts
- Interval arithmetic
- Unlimited precision integer type
Parallel development for TR2
Extensions made simple

- Problem updating std components to use TR types
- ‘inherting constructors’ allows compatible upgrades

```cpp
namespace std {
namespace tr2 {

    template<>
    class fstream {
        using fstream::fstream; // inherit all fstream ctors
        fstream( const basic_filepath<...> & path )
            : fstream( path.c_str() ) // delegate to existing ctor
        {
        }
    };
}
```
To be continued...

Updates can be found on wg21 web site

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