Test document for mpark/wg21

Visual inspection of various features of the framework

Document #:	D0000R1
Date:	2025-01-09
Project:	Programming Language C++
Audience:	Library Evolution Library
Revises:	D0000R0
Reply-to:	Author 0 <author0@gmail.com> Author 1 <author1@gmail.com>

Contents

1 Introduction

This framework provides support for various common elements for C++ papers. This document is intended to test various features implemented in `mpark/wg21`.

2 Title

The title is specified by YAML metadata block.

```yaml
---
title: Example Title
subtitle: Example Subtitle
document: DxxxxRn
date: 2019-06-13
audience:
  - Library Evolution
  - Library
author:
  - name: Author One
    email: <one@author.com>
  - name: Author Two
    email: <two@author.com>
toc: false    # default: `true`
toc-depth: 4  # default: `3`
---
```

`date: today` will generate today's date in `YYYY-MM-DD` (ISO 8601) format.

3 Markdown

Pandoc Markdown is the Markdown flavor used for this framework.

3.1 Inline Formatting

Inline formatting such as **bold**, *italics*, and `verbatim` work as you would expect. There are also useful extensions such as ~~strikeout~~, su$_{\text{b}}$script, su$^{\text{per}}$script, and highlighted code: `constexpr`.

Various compositions in compact list:

— ~~`A<B<T>>`~~
— `compare_3way`
— `3WAY<R>`
— `operator@`
— `operator+`
— `x @ y`
— ~~`x & y`~~
— **foo `constexpr` bar**
— *foo `constexpr` bar*
— ~~foo `constexpr` bar~~
— `hello world`
— ~~*`hello world`*~~
— ~~**`hello world`**~~
— `namespace` *`unspecified`* `{ struct sender_base {}; }`
— `namespace` *`unspecified`* `{ struct sender_base {}; }`

Loose list:

— $\texttt{x}_i$ `<=>` $\texttt{y}_i$

— ~~foo `hello world` bar~~

3.2 Inline Code in Headers: `int`, `x & y`

3.3 Code Block

3.3.1 No Syntax Highlighting

```
#include <iostream>
#include "foo.h"

__FILE__;

int x = 42'234'234;
const int x = 42ul;
const int x = 0B01011;

bool b = true;

struct process {
  hello ~~constexpr~~ detail::foo::template foo;

  [[using CC: opt(1), debug]] x;

  template <typename I>
  [[nodiscard]] auto operator()(I i) -> O<I> { /* ... */ };
};

namespace unspecified { struct sender_base {}; }
using unspecified::sender_base;

template<class, class> struct as-receiver; // exposition only
template<class, class> struct as-invocable; // exposition only
```

3.3.2 C++ Syntax Highlighting

```
#include <iostream>
#include "foo.h"

__FILE__;

int x = 42'234'234;
const int x = 42ul;
const int x = 0B01011;

bool b = true;

struct process {
  hello constexpr detail::foo::template foo;

  [[using CC: opt(1), debug]] x;

  template <typename I>
  [[nodiscard]] auto operator()(I i) -> O<I> { /* ... */ };

  x_i <=> y_i;
};

if (x) {
  return ""sv;
  return 5ms;
}

std::printf("%d", x);

std::variant<I1, I2> input = 'h';
std::variant<I1, I2> input = "h";
std::variant<I1, I2> input = "hello";

// mapping from a `variant` of inputs to a `variant` of results:
auto output = std::visit<std::variant<O<I1>, O<I2>>>(process{}, input);

// coercing different results to a common type:
auto result = std::visit<std::common_type_t<O<I1>, O<I2>>>(process{}, input);

// visiting a `variant` for the side-effects, discarding results:
std::visit<void>(process{}, input);

namespace unspecified { struct sender_base {}; }
using unspecified::sender_base;
```

3.3.3 diff Syntax Highlighting

```
some things just don't change.

// 20.3.4 tuple-like access to pair:
- constexpr typename tuple_element<I, std::pair<T1, T2> >::type&
+ constexpr tuple_element_t<I, pair<T1, T2> >&
-   get(std::pair<T1, T2>&) noexcept;
+   get(pair<T1, T2>&) noexcept;

unspecified detail::foo::template foo;
+ unspecified detail::foo::template foo;
- unspecified detail::foo::template foo;
```

3.3.4 rust Syntax Highlighting

```
enum Result<T, E> {
  Ok(T),
  Err(E),
}

match parse(some_input) {
  Ok(v) => // use `v`
  Err(err) => // use `err`
}
```

4 Comparison Tables

Table 1: Put your caption here

Before

```
switch (x) {
  case 0: std::cout << "got zero"; break;
  case 1: std::cout << "got one"; break;
  default: std::cout << "don't care";
}
```

After

```
inspect (x) {
  0: std::cout << "got zero";
  1: std::cout << "got one";
  _: std::cout << "don't care";
}
```

Before

```
switch (x) {
  case 0: std::cout << "got zero"; break;
  case 1: std::cout << "got one"; break;
  default: std::cout << "don't care";
}
```

After

```
inspect (x) {
  0: std::cout << "got zero";
  1: std::cout << "got one";
  _: std::cout << "don't care";
}
```

Before

```
if (s == "foo") {
  std::cout << "got foo";
} else if (s == "bar") {
  std::cout << "got bar";
} else {
  std::cout << "don't care";
}
```

After

```
inspect (s) {
  "foo": std::cout << "got foo";
  "bar": std::cout << "got bar";
  _: std::cout << "don't care";
}
```

5 Proposed Wording

5.1 Paragraph Numbers

2 An expression is *potentially evaluated* unless it is an unevaluated operand (7.2) or a subexpression thereof. The set of *potential results* of an expression `e` is defined as follows:

(2.1) — If `e` is an *id-expression* (7.5.4), the set contains only `e`.

(2.2) — If `e` is a subscripting operation (7.6.1.1) with an array operand, the set contains the potential results of that operand.

5.2 Wording Changes

Large changes are `::: add` for additions, `::: rm` for removals.

Modify section 28.5.5 [format.functions]:

```
template<class... Args>
  string format(const locale& loc, string_view fmt, const Args&... args);
```

Returns: `vformat(loc, fmt, make_format_args(args...))`.

Small, inline changes are done with `[new text]{.add}` or `[old text]{.rm}`.

Specifier	Replacement
`%a`	The locale's abbreviated weekday name. If the value does not contain a valid weekday, ~~`setstate(ios::failbit)` is called~~ `format_error` is thrown.
`%A`	The locale's full weekday name. If the value does not contain a valid weekday, ~~`setstate(ios::failbit)` is called~~ `format_error` is thrown.

5.3 Grammar Changes

selection-statement:
  `if constexpr`$_{opt}$ `(` *init-statement*$_{opt}$ *condition* `)` *statement*
  `if constexpr`$_{opt}$ `(` *init-statement*$_{opt}$ *condition* `)` *statement* `else` *statement*
  `switch (` *init-statement*$_{opt}$ *condition* `)` *statement*
  `inspect constexpr`$_{opt}$ `(` *init-statement*$_{opt}$ *condition* `) {` *inspect-case-seq* `}`

inspect-case-seq:
  inspect-case
  inspect-case-seq inspect-case

inspect-case:
  attribute-specifier-seq$_{opt}$ *inspect-pattern inspect-guard*$_{opt}$ `:` *statement*

inspect-pattern:
  wildcard-pattern
  identifier-pattern
  constant-pattern
  structured-binding-pattern
  alternative-pattern
  binding-pattern
  extractor-pattern

inspect-guard:
  `if (` *expression* `)`

6 Stable Names

Stable names are written as `[basic.life]{.sref}`, and renders as 6.7.4 [basic.life]. You can also add a class `-` or `.unnumbered` to omit the section number.

It uses https://timsong-cpp.github.io/cppwp/annex-f as the underlying database.

Examples:

- `[basic.life]{.sref}` → 6.7.4 [basic.life]
- `[basic.life]{- .sref}` → [basic.life]
- `[basic.life]{.unnumbered .sref}` → [basic.life]

7 Notes

There are three supported styles of note:

- Use the `note` class for notes that are expected to appear in the specification wording

```
[Notes will look like this]{.note}
```

  [*Note:* Notes will look like this — *end note*]

- Use the `ednote` for editorial notes, these will be formatted as

```
[Editorial notes are important]{.ednote}
```

[Editor's note: Editorial notes are important]

— Use `draftnote` to include text that is intended as questions or information for reviews and working groups.

```
[Drafting notes can be used to provide comments for reviewers that are explicitly not to be
 included in the specification.]{.draftnote}

 [It is also possible to indicate the a note is for
 a specific `audience` via this optional attribute.]{.draftnote audience="the reader"}
```

[Drafting note: Drafting notes can be used to provide comments for reviewers that are explicitly not to be included in the specification.]

[Drafting note for the reader: It is also possible to indicate the a note is for a specific `audience` via this optional attribute.]

8 Citation

Automatic references are written as `[@N4762]` and renders as [N4762]. Anything in https://wg21.link/index.yaml are linked automatically.

- N Papers (e.g., `[@N3887]` → [N3887])
- P Papers (e.g., `[@P1371R1]` → [P1371R1])
- CWG Issues (e.g., `[@CWG1234]` → [CWG1234])
- LWG Issues (e.g., `[@LWG1234]` → [LWG1234])
- Github Edits (e.g, `[@EDIT1234]` → [EDIT1234])
- Standing Documents (e.g., `[@SD6]` → [SD6])

You may also write `[@P2996R8]{.title}` to include the title of the paper, and renders as: [P2996R8] (Reflection for C++26).

9 Automatic Header Links

Automatic header links are written as `[](#auto-header-links)`, and renders as Automatic Header Links.

10 References

[CWG1234] Johannes Schaub. 2011-01-18. abstract-declarator does not permit ... after ptr-operator.
https://wg21.link/cwg1234

[EDIT1234] [intro.races] Remove duplicated index entry for "data race."
https://wg21.link/edit1234

[LWG1234] Matt Austern. "Do the right thing" and NULL.
https://wg21.link/lwg1234

[N3887] Michael Park. 2013-12-26. Consistent Metafunction Aliases.
https://wg21.link/n3887

[N4762] Richard Smith. 2018-07-07. Working Draft, Standard for Programming Language C++.
https://wg21.link/n4762

[P1371R1] Sergei Murzin, Michael Park, David Sankel, Dan Sarginson. 2019-06-17. Pattern Matching.
https://wg21.link/p1371r1

[P2996R8] Barry Revzin, Wyatt Childers, Peter Dimov, Andrew Sutton, Faisal Vali, Daveed Vandevoorde, Dan Katz. 2024-12-17. Reflection for C++26.
https://wg21.link/p2996r8

[SD6] SG10 Feature Test Recommendations.
https://wg21.link/sd6