

# Text Formatting

## P0645R1

Victor Zverovich ([victor.zverovich@gmail.com](mailto:victor.zverovich@gmail.com))

# Overview

- Python-like format string syntax:

```
string s = fmt::format("{} ", 42);  
fmt::format("{:10x}", 42); // compare to sprintf("%10x", 42)
```

- Formatting is extensible for user-defined types
- Natural function call API using variadic templates:

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

- Positional arguments:

```
string s = fmt::format("{1}{0}", "foo", "bar");
```

- Control over the use of locales:

```
fmt::format("{} ", 1.2); // not using locale  
fmt::format("{:n}", 1.2); // using locale
```

# Changes

## Changes since [R0](#)

- Add section [Compile-time processing of format strings](#).
- Separate parsing and formatting in the extension API replacing `format_value` function template with class template `formatter` to allow compile-time processing of format strings.
- Change return type of `format_to` and `vformat_to` to `OutputIterator` in synopsis.
- Remove sections Null-terminated string view and Format string, and replace `basic_cstring_view` with `basic_string_view`.
- Add a link to the implementation in [Introduction](#).
- Add a note regarding time formatting and compatibility with D0355 "Extending <chrono> to Calendars and Time Zones" [\[16\]](#) to section [Extensibility](#).
- Rename `basic_args` to `basic_format_args`.
- Rename `is_numeric` to `is_arithmetic`.
- Add the `count` function that counts the number of characters and use it to define output ranges.
- Remove `basic_buffer` and section Formatting buffer and replace buffers with output iterators.
- Add [Appendix A: Benchmarks](#).
- Explain the purpose of the type-erased API in more details in the [Binary footprint](#) section.
- Add [Appendix B: Binary code comparison](#).
- Add formatting function overloads for `wchar_t` strings.

# Extensibility

Replacement field syntax

```
replacement-field ::= '{' [arg-id] [':' format-spec] '}'
```

where `format-spec` is predefined for built-in types, but can be customized for user-defined types, e.g. `put_time`-like formatting for `tm`:

```
time_t t = time(nullptr);  
string date = fmt::format("The date is {0:%Y-%m-%d}.",  
                           *localtime(&t));
```

by providing a specialization of `formatter` for `tm`:

```
template <>  
struct formatter<tm> {  
    constexpr fmt::parse_context::iterator  
        parse(fmt::parse_context& ctx); // constexpr-ready  
  
    template <class FormatContext>  
    typename FormatContext::iterator  
        format(const tm& tm, FormatContext& ctx);  
};
```

# Compile-time format strings

Extension API is constexpr-ready: parsing can be done at compile time.

Possible API (not proposed):

```
template <class String, class... Args>
string format(String format_str,
              const Args&... args);
```

If P0424R1 "Reconsidering literal operator templates for strings" goes in it will be possible to do:

```
string s = format("{0}"_fmt, 42);
```

Demonstrated to work with all of the features of the current proposal in the reference implementation (compile-time strings emulated with macros).

Another option: P0732R0 "Class Types in Non-Type Template Parameters"

Runtime format strings still need to be supported, compile-time can be added later.

# Output iterators

"Look at using or explain why not to use an output iterator." - LEWG

Removed the buffer API which was in R0.

Changed `format_to` to the following:

```
template <class OutputIterator, class... Args>
OutputIterator format_to(
    OutputIterator out,
    string_view format_str,
    const Args&... args);
```

# Output size

Added a function to compute output size:

```
template <class... Args>
size_t count(
    string_view format_str,
    const Args&... args);
```

*Effects:* The function writes to the range `[out, out + fmt::count(format_str, args...)]` the format string `format_str` with each replacement field substituted with the character representation of the argument it refers to, formatted according to the specification given in the field.

*Returns:* The end of the output range.

# Benchmarks

Format 1000 random integers.

Run on (4 X 3100 MHz CPU s)

2018-01-27 07:12:00

Benchmark	Time		CPU Iterations		
-----					
sprintf	882311	ns	881076	ns	781
ostringstream	2892035	ns	2888975	ns	242
to_string	1167422	ns	1166831	ns	610
format	675636	ns	674382	ns	1045
format_to	499376	ns	498996	ns	1263

Compiled with clang (Apple LLVM version 9.0.0 clang-900.0.39.2) with -O3 -DNDEBUG and run on a macOS system.

sprintf and format\_to use a stack-allocated array, the rest use std::string.



# Type-erased API

Type-erased version of `format` to avoid code bloat:

```
string vformat(  
    string_view format_str, format_args args);
```

`format` can be an inline wrapper that only creates type-erased format argument list `format_args`:

```
template <class... Args>  
inline string format(  
    string_view format_str,  
    const Args&... args) {  
    return vformat(format_str, make_args(args...));  
}
```

# Binary code comparison

```
void consume(const char*);
```

```
// 84 bytes
```

```
void sprintf_test() {  
    char buffer[100];  
    sprintf(buffer, "The answer is %d.", 42);  
    consume(buffer);  
}
```

```
// 127 bytes
```

```
void format_test() {  
    consume(format("The answer is {}. ", 42).c_str());  
}
```

```
// 607 bytes
```

```
void ostreamstream_test() {  
    std::ostreamstream ss;  
    ss << "The answer is " << 42 << ".";  
    consume(ss.str().c_str());  
}
```

C++ source #1 x

A Save/Load + Add new... C++

```
1 #include <fmt/core.h>
2
3 int main() {
4     std::string s =
5     fmt::format("The answer is {}.", 42);
6 }
```

x86-64 clang 4.0.0 (Editor #1, Compiler #1) C++ x

x86-64 clang 4.0.0 -std=c++14 -O2

A 11010 .LX0: .text // \s+ Intel Demangle

Libraries + Add new...

```
1 main: # @main
2     sub rsp, 56
3     mov qword ptr [rsp], 42
4     lea rdi, [rsp + 24]
5     mov r8, rsp
6     mov esi, .L.str
7     mov edx, 17
8     mov ecx, 2
9     call fmt::vformat[abi:cxx11](fmt::
10     mov rdi, qword ptr [rsp + 24]
11     lea rax, [rsp + 40]
12     cmp rdi, rax
13     je .LBB0_2
14     call operator delete(void*)
15 .LBB0_2:
16     xor eax, eax
17     add rsp, 56
18     ret
19
20 .L.str:
21     .asciz "The answer is {}."
```

# Thanks

- Beman Dawes
- Bengt Gustafsson
- Eric Niebler
- Jason McKesson
- Jeffrey Yasskin
- Joël Lamotte
- Howard Hinnant
- Lee Howes
- Louis Dionne
- Michael Park
- Thiago Macieira
- Zach Laine
- LEWG