Doc No: WG21 **P0392**
Date: 2016-06-23
Reply to: Nicolai Josuttis (nico@josuttis.de)
Subgroup: LWG
Prev. Version: ----

Adapting string_view by filesystem paths

Motivation

In C++17 std::string_view was added, which is especially useful to use when dealing with character sequences where there is no need to have them zero-terminated or to pass/use them to/as a sink.

To support that, (library) function taking strings for that case should provide a way to accept a string view directly instead of taking a string only. The benefit is to avoid the need for explicit and possibly expensive conversions to strings when passing string views.

One place, where this would be a drawback are paths in the filesystem library (class std::filesystem::path), where at a couple of places currently you can only pass strings.

The current situation means that

a) you simply can't pass a string_view as argument to create a path:

```
string_view sv = ...;
path p(sv);          // Ill-formed
```

b) and an explicit conversion such as

```
string_view sv = ...;
path p(string(sv)); // OK, but unnecessarily expensive
```

is unnecessarily expensive because the passed string_view objects first would is converted to a string before it is internally processed/copied.

This paper fixes this by providing parallel string_view support wherever currently only a string can be passed.

Note that it is the intention still to internally use and return a string because there path is the owner of the value.

Note that the fix does not replace string operations by string_view. Instead it adds the string_view overloads.

Note also that issue 2707, which is tentatively ready proposes to add overloads for move semantics when passing strings:

```
namespace std::filesystem {
  class path {
    path(string_type&& source);
    …
    path& operator=(string_type&& source);
    path& assign(string_type&& source);
    …
};
```

Note also that there is issue 2711, which is affected by this paper. **As requested by LWG, to ensure that the proposed wording is taking that into account and that we don't have a "race" between the paper and the issue, this paper also adopts the resolution of issue 2711 with the necessary modifications.**

Wording of Proposed Changes

(all against N4594)

[Drafting note due to adoption of issue 2711: Functions taking EcharT or Source parameter types often provide additional overloads with the same arity and concrete types. In order to allow conversions to these concrete types in the interface we need to explicitly disable the EcharT and Source overloads.]

Due to the adoption of issue 2711 **change 27.10.5 [fs.req] as indicated**:

> -2- ~~Template parameters named EcharT shall be~~ Functions with template parameters named EcharT shall not participate in overload resolution unless EcharT is one of the encoded character types.

In **27.10.8 Class path [class.path] in class std::filesystem::path**

after:

```
path& operator+=(const string_type& x);
```

add:

```
path& operator+=(basic_string_view<value_type> x);
```

and after:

```
int compare(const string_type& s) const;
```

add:

```
int compare(basic_string_view<value_type> s) const;
```

In **27.10.8.3 path requirements [path.req]**

after 1.1:

> (1.1) — basic_string<EcharT, traits, Allocator>. A function argument const Source& source shall have an effective range [source.begin(), source.end()).

add:

> (1.x) — basic_string_view<EcharT, traits>. A function argument const Source& source shall have an effective range [source.begin(), source.end()).

Due to adoption of Issue 2711 with necessary fixes for string_view **insert a new paragraph between 27.10.8.3 [path.req] p1 and p2 as indicated** (green are only the changes due to string_view support):

> Functions taking template parameters named `Source` shall not participate in overload resolution unless either `Source` is a specialization of `basic_string` **or `basic_string_view`** or the qualified-id `iterator_traits<decay_t<Source>>::value_type` is valid and denotes a possibly `const` encoded character type (14.8.2 [temp.deduct]).

In **27.10.8.4.4 path concatenation [path.concat]**

after**:**

```
path& operator+=(const string_type& x);
```

add:

```
path& operator+=(basic_string_view<value_type> x);
```

in **27.10.8.4.8 path compare [path.compare]**

Replace**:**

```
int compare(const string_type& s) const
    Returns: compare(path(s)).
```

by:

```
int compare(const string_type& s) const
int compare(basic_string_view<value_type> s) const
    Returns: compare(path(s)).
```

Acknowledgements

Thanks to Daniel and Titus for their reviews and feedback.