Project: Programming Language C++, Library Evolution Working Group
Document number: P0123R0
Date: 2015-09-25
Reply-to: Neil MacIntosh neilmac@microsoft.com

Unifying the interfaces of string_view and array_view

Contents

Introduction

This paper presents a design for *basic_string_view* (similar to that proposed in N3762 [1]) that would have an interface consistent with the *array_view* type described in P0122 [2]. Doing so improves the generality of the *basic_string_view* type and allows it to offer bounds-safety guarantees like *array_view*.

It is worth noting that the *basic_string_view* type presented here directly depends upon the *array_view* proposed in P0122.

Motivation and Scope

basic_string_view is a "vocabulary type" that is proposed for inclusion in the standard library. It can be widely used in C++ programs, as a replacement for passing const *basic_string* objects or zero-terminated character arrays. This *basic_string_view* design supports high performance and bounds-safe access to contiguous sequences of characters. This type would also improve modularity, composability, and reuse by decoupling accesses to string data from the specific container types used to store that data.

It is desirable that the interface offered by *basic_string_view* is harmonized with *array_view*, given the similarity between the purposes and functionality of the two types. This has the positive benefit of also reducing the number of interfaces that need to be learned by C++ programmers who want to perform bounds-safe, high-performance access to sequences – whether they are sequences of characters, or objects.

Impact on the Standard

basic_string_view is a pure library extension. It does not require any changes to standard classes, functions or headers. Nor does it require any changes or extensions to the core language.

However, it can be imagined that some standard library functions and classes might benefit from adopting overloads for this new types, if it is adopted.

basic_string_view as presented here has been implemented in standard C++ and successfully used within at least one commercial codebase. An open source reference implementation is available at https://github.com/Microsoft/GSL [3].

Design Decisions

basic_string_view as a type alias

A string is simply a contiguous sequence of characters. An *array_view* is a vocabulary type that encapsulates access to a contiguous sequence of objects. For simplicity, *basic_string_view* should simply be a type alias for *array_view*. Specifically of the form:

```
template<class CharT, size_t Extent = dynamic_range>
using basic_basic_string_view = array_view<CharT, Extent>;
```

This design decision allows code that deals with contiguous sequences to look and behave uniformly (whether they are of elements or characters). It also reduces the “API” surface that a C++ programmer must learn and remembered for users of *basic_string_view* and *array_view.*

The design keeps the interface of *basic_string_view* interface as simple as possible. This, in turn, makes the requirement on containers that it can be a view over as simple as possible. The proposed type-alias form of *basic_string_view* can be used over a wide variety of string containers - such as *CString*, *const char**, *BSTR*, *QString* or any of the other myriad of string types that are commonly used in C++ today – with minimal adaptation effort. That capacity – to decouple functions from the details of the string type being used – is a significant benefit that *basic_string_view* can bring to C++ programmers.

Removing string-specific member functions
The approach of using a type alias to *array_view* removes the opportunity for *basic_string_view* to have a range of string-specific functions. Instead, string-specific functionality can be offered as free functions. This design follows the general approach of the standard library, which is to separate algorithms such as *find_first()* from the containers or views they operate over, by making them free functions. The lack of member functions also makes it clearer to users of *string_view* parameters or variables that they cannot assume they are operating over a *basic_string*. *basic_string_view* is a type that decouples users from the details of underlying string container types.

Zero termination
basic_string_view is completely agnostic of zero-termination requirements/promises in the string data it contains. Again, this allows maximum flexibility for usage.

A free function is provided for creating a *string_view* with a measured length from zero-terminated strings: *ensure_z()*. Using this function for initialization from string literals is zero-overhead, but makes the intent of a programmer clearer in the source code. For other variables, it will have the same runtime overhead as a *strlen()* (or equivalent operation).

Character traits
Although this proposal does not include character traits support in the proposed definition of *basic_string_view*, it is not prejudiced against such inclusion. It would certainly be possible to add an additional template parameter to the type alias if free functions that wanted to operate over *basic_string_view* would find a character traits template type argument helpful.

Static and dynamic lengths
By adopting the type-alias design, *basic_string_view* objects are capable of being declared as either having a static-size (fixed at compile-time) or dynamic-size (provided at runtime). Conversions between the two varieties are allowed with limitations to ensure bounds-safety is always preserved. Fixed-size *basic_string_view* can be implemented with no size overhead when compared to passing a single pointer.

Supporting both mutable and const forms
basic_string_view as a type-alias can also support either read-only or mutable access to the sequence it encapsulates. To access read-only data, the user can declare a *basic_string_view<const char>* (for example), and access to mutable data would use a *basic_string_view<char>*. While it is acknowledged

that the majority of *basic_string_view* usage would tend to be for read-only access, it is clearly useful to have mutable access to an existing string (particularly one of fixed-size).

Convenience aliases

There are a number of "convenience" aliases for the various combinations of character types and constness that are commonly useful with *basic_string_view*:

```
template<size_t Extent = dynamic_range>
using string_view = basic_string_view<char, Extent>;
```

```
template<size_t Extent = dynamic_range>
using cstring_view = basic_string_view<const char, Extent>;

template<size_t Extent = dynamic_range>
using wstring_view = basic_string_view<wchar_t, Extent>;

template<size_t Extent = dynamic_range>
using cwstring_view = basic_string_view<const wchar_t, Extent>;
```

Specification

```
template<class CharT, size_t Extent = dynamic_range>
using basic_basic_string_view = array_view<CharT, Extent>;

template<size_t Extent = dynamic_range>
using string_view = basic_string_view<char, Extent>;

template<size_t Extent = dynamic_range>
using cstring_view = basic_string_view<const char, Extent>;

template<size_t Extent = dynamic_range>
using wstring_view = basic_string_view<wchar_t, Extent>;

template<size_t Extent = dynamic_range>
using cwstring_view = basic_string_view<const wchar_t, Extent>;

//
// ensure_z - creates a string_view for a zero-terminated character array.
// Will fail fast if a null-terminator cannot be found before
// the limit of size_type.
//
template<class T>
basic_string_view<T, dynamic_range> ensure_z(T* const & sz, size_t max =
std::numeric_limits<size_t>::max());
```

```
template<class T, size_t N>
basic_string_view<T, dynamic_range> ensure_z(T(&sz)[N]);

template<class Cont>
basic_string_view<typename std::remove_pointer<typename Cont::pointer>::type,
dynamic_range> ensure_z(Cont& cont);

//
// to_string() allows explicit conversions from string_view to string
//
template<class CharT, size_t Extent>
std::basic_string<typename std::remove_const<CharT>::type> to_string(const
basic_string_view<CharT, Extent>& view);
```

Acknowledgements

This version of *string_view* was designed to support the C++ Core Coding Guidelines [4] and as such, the current version reflects the input of Herb Sutter, Jim Springfield, Gabriel Dos Reis, Chris Hawblitzel, Gor Nishanov, and Dave Sielaff. Łukasz Mendakiewicz, Bjarne Stroustrup, Eric Niebler and Artur Laksberg provided helpful review of this version of *string_view* during its development.

Many thanks to Gabriel Dos Reis for review of this document during its initial development.

References

[1] J. Yasskin, "string_view: a non-owning reference to a string, revision 5” 09 January 2013. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3762.html

[2] N. MacIntosh, "array_view: bounds-safe views for sequences of Objects” 25 September 2015.

[3] string_view reference implementation: https://github.com/Microsoft/GSL