On Support for TR-19769 and New Character Types

Nick Stoughton, Larry Dwyer

TR 19769 was developed by ISO/IEC JTC 1/SC 22/WG 14 and became a published type-2 technical report in 2004. It is at this point ready (in fact, overdue) for its three-year review under the JTC 1 procedures (see 16.4.1 of SC 22 N4305). The Working Group should at this point be making a recommendation to SC 22 whether the TR should be:

- Converted to an IS without change;
- Revised and published as an IS;
- Confirmed for continuation as a TR;
- Revised for publication as a revision to the TR;
- Withdrawn.

This paper argues for the third of those options; that the TR should be confirmed for continuation as a TR, and should not be considered for inclusion in the upcoming revision of IS 9899.

The API Problem

Several organizations have attempted to add support for 16 and 32 bit Unicode characters, and have run into the problem caused by the explosion in the number and complexity of library APIs required to handle the new types. One such discussion can be found in the gcc maintainers' discussions at http://gcc.gnu.org/ml/gcc/2000-08/msg00104.html. The application developer is left with a bewildering choice of interfaces, the library maintainer is faced with trying to fix the same bug in a multitude of variants of the same code, and the standards developer is faced with specifying the same interface over and over again, with similar maintenance problems (witness the problems already face by both WG 14 and the Austin Group in maintaining both narrow and wide character versions of all the string interfaces).

The ABI Problem

The Technical Report describes a problem and solution that is not an API problem, but an ABI problem. The ISO-C Standard defines the wide character type, wchar_t, which can be used to implement the UTF-16 or UTF-32 form of the ISO-10646 code set. The choice is left to the implementer.

The TR states that it is “sensible to give all the Unicode encoding forms appropriate data type support”. If it is sensible to support all Unicode types, then it must also be sensible to define, for example, time32_t and time64_t types for time representation. Those applications that are concerned with time in the present will work fine with a time32_t. Those applications that are concerned with time beyond the year 2038 can use the time64_t. To accommodate all of the uses of time_t, it is necessary to introduce all of the functions which manipulate time, for example, mktime32 and mktime64.

Combining "duplicate" functions for each ABI in a single ABI causes problems for application developers that provide their application in library form. Since they do not know which form the customer will prefer in the end user developed application, the ISV must provide all forms of their library. This applies to all third party libraries, since the end user is free to mix these libraries in a single program.

The system implementers determine whether and how to address any problems associated with types that may be limited by an ABI. In the case of time_t, they offer an ABI which supports 32-bit time and an ABI which supports 64-bit time. Not to limit the ABI to a single API will cause source portability problems from vendor to vendor. It is important to note that there is no exact bit-size requirement on a char16_t or char32_t (they are defined as “at least”), and some vendors may choose to implement these types with a 32 bit object, as synonyms for their existing wide character type.

Unicode Requirements

There is no requirement in either the Unicode specification or the ISO-10646 Standard that implementations must support all three of the translated forms; UTF-8, UTF-16 and UTF-32. Without reading the Unicode specification, one might get the incorrect impression that UTF-8, UTF-16 and UTF-32 are character definitions. In fact, UTF-8 and UTF-16 are multi-character translations (like multi-byte definitions) of ISO-10646. Each UTF-16 encoding might fit in a single character, or it might require two char16_t objects (called surrogate pairs). Unlike UTF-16, each UTF-32 encoding fits in a single object.

The debate in the Unicode Consortium about size versus ease of use and the marginalization of some languages versus the selection of preferred languages continues to this day. It is the reason that Unicode defines both UTF-16 and UTF-32. In order to arrive at a compromise the consortium chose to define both forms and leave it to the market to decide. By forcing all implementations to implement both forms, this TR effectively does an end run around the natural selection process and uses the ISO-C Working Group to trump the debate.

No matter which side of the fence an implementer is on, this forces all implementers to support both the UTF-16 and UTF-32 forms. It will require a complete, duplicate set of string manipulation functions, such as memset16 and memset32, These functions are not in lieu of the existing functions memset and wmemset. They must exist in addition to the functions already defined by various standards.

This duplication will require, by transitivity, a duplicate set of all character handling interfaces within libraries (or even duplicate libraries) provided by those system and application vendors that distribute their applications in the form of libraries. If the ISO-C Working Group is going to end the Unicode debate, they should redefine wchar_t to contain either UTF-16 or UTF-32 character definitions.

Conclusions

TR 19769 serves a purpose in the “natural selection” process described above.

Adding char16_t and char32_t support from TR-19769 would force ABI definitions into the ABI agnostic ISO-C Standard. The Working Group should not get involved in the problems of the implementation ABI's.

A char16_t storage unit cannot be used to store all codes defined in ISO-10646 because the number of codes defined in the Standard exceeds 65535. Therefore, the only way to represent all of ISO-10646 is to make the char16_t a multi-object representation akin to the multi-byte representation used for char. Hence, it is wrong to refer to it as a character class or to have a string literal definition for the mapping of UTF-16 characters into a string. The TR conveniently omits the fact that a 16 bit char16_t cannot hold all of the ISO-10646 codes.

The only complete and unambiguous implementation of char16_t is to make it an alias for char32_t, and char32_t in turn an alias for wchar_t. As such, these types do not serve any useful purpose.

The existing 9899 C standard contains all the required standardized APIs for multi-byte characters and wide characters, and does not need the additional types required by TR-19769.