C2X Liaison: Rebuttal to N2526

Title: **On the use of “const” for data that shall not be modified**
Author: **Nick Stoughton / The Austin Group**
Date: **2020-08-10**

Introduction

N2526 proposes to change the return type of four functions, **getenv**, **localeconv**, **setlocale**, and **strerror** to add the “`const`” qualifier, so they return “`const char *`” instead of a plain “`char *`”.

This proposal was accepted at the August 2020 WG 14 meeting, but met resistance from the liaison to the POSIX committee. This paper attempts to explain some of the difficulties that such a change would bring about.

The Austin Group, a Joint Working Group between ISO/IEC JTC 1/SC 22, the IEEE and The Open Group, is responsible for the development and maintenance of the POSIX standard (ISO/IEC Standard 9945).

The POSIX Standard incorporates all the C library APIs (chapter 7 of the C standard), and defers to C explicitly where such an interface is shared. For example, the introduction to the “**`getenv`**” description states

> The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The Austin Group is committed to tracking developments in the C standard, and is undergoing revision at this time. The next version will align with either C17 or C2X, if C2X is published before the POSIX revision. Therefore, the changes proposed in N2526 could force the change of those interfaces in POSIX-2X.

Existing Practice Review

It is the opinion of the Austin Group that N2526 does not adhere to the C2X charter as described in WG 14 N 2086 (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2086.htm) in that it violates guiding principle #1:

> **Existing code is important, existing implementations are not.** A large body of C code exists of considerable commercial value. Every attempt has been made to ensure that the bulk of this code will be acceptable to any implementation conforming to the

> Standard. The Committee did not want to *force* most programmers to modify their C programs just to have them accepted by a conforming translator.

Assigning a "`const char *`" value to a non-const variable (as much existing code does) will provoke at least warnings from current conforming compilers. Users who wish to treat warnings as errors (as is common in safety and security applications) or who adhere to a "no warnings" coding standard will need to modify their source code (which in the case of third-party libraries may not be possible).

N2526 claims to have searched for problematic uses of "**`strerror`**" in some sources published on github. Such a search is problematic in several ways:

- Of the four interfaces discussed, **`strerror`** is possibly the least problematic and most likely to be used in a constant form. You can only search public repositories in github. Many thousands of private repositories on github were not searched, nor were any repositories on other similar sites.
- N2526 only examined the "first few pages" of results. This is inadequate as a basis for justifying such a radical change.
- The OpenSolaris source code base contains many dozens of cases that would lead to failures in compilation.
- The GNU/Linux source code base for a simple embedded device contains over 100 cases for getenv alone, and a test compilation changing **getenv** only, failed long before completing (building "bash", in a basically alphabetical list of packages).

Of the functions proposed in N2526, getenv is the most widespread and problematic.

An Improved Search Strategy

Searching for a function name and analysing the code in the results visually is very time consuming. However, for a specific type of function use, a shortcut can be employed: search for variable initializations that use the function, then search for variations that include "const". Subtract to find the number that do not use "const".

Using the regular expression search facility on codesearch.debian.net produces the following results:

RE	**Count**
char .*= *getenv	9482
const *char .*= *getenv	4281
char *const.*= *getenv	219
struct *lconv.*= *localeconv	469
const *struct *lconv.*= *localeconv	45
struct *lconv *const.*= *localeconv	15
char .*= *setlocale	638
const *char .*= *setlocale	154
char *const.*= *setlocale	81
char .*= *strerror	1098
const *char .*= *strerror	440
char *const.*= *strerror	42

From which can be derived the following percentages:

Function	With const	Without const
getenv	47.5	52.5
localeconv	12.8	87.2
setlocale	36.8	63.2
strerror	43.9	56.1

Although these figures are just for calls that occur in variable initializations, the percentages for calls where the variable definition and assignment are separate are likely to be in the same ballpark.

Thus it can be seen that N2526's claim that "such programs are apparently uncommon" is an incorrect conclusion.

POSIX Additional APIs

For each of the 4 interfaces discussed in N2526, POSIX contains a normative requirement (as does C17) that the returned string shall not be modified by the application. However, POSIX extends this requirement in the case of getenv():

- The application shall ensure that it does not modify the string pointed to by the getenv() function, unless it is part of a string previously added to the environment using putenv().

That is to say, it is legal to add a string to the environment with one API, and then to modify the string returned from getenv() in the knowledge that it is held in modifiable storage.

Potential Solutions

The Austin Group would strongly encourage the addition of new functions that add the "const" qualifier. For example, "const char *cgetenv(const char *name)".

The alternative would be for POSIX and C to diverge over these interfaces, with POSIX replacing its deference to C with a statement to the effect that conforming POSIX applications

might not conform to C2X. This could in turn lead to fewer implementations of C2X being released.

The Austin Group would support the deprecation / obsolescence of the original interfaces, but this paper does not propose to do that.

Proposed Wording

Add new sections (renumbering subsequent sections as needed):

7.11.1.2 The csetlocale function

Synopsis

```
#include <locale.h>
const char * csetlocale(int category, const char *locale);
```

Description

The **`csetlocale`** function is identical to the **`setlocale`** function, except that the return type is `const` qualified.

7.11.2.2 The clocaleconv function

Synopsis

```
#include <locale.h>
const struct lconv * clocaleconv(void);
```

Description

The **`clocaleconv`** function is identical to the **`localeconv`** function, except that the return type is `const` qualified.

7.22.4.7 The cgetenv function

Synopsis

```
#include <stdlib.h>
const char * cgetenv(const char *name);
```

Description

The **cgetenv** function is identical to the **getenv** function, except that the return type is `const` qualified.

7.24.4.7 The cstrerror function

Synopsis

```
#include <string.h>
const char * cstrerror(int errnum);
```

Description

The **cstrerror** function is identical to the **strerror** function, except that the return type is `const` qualified.

Acknowledgements

I would like to thank the core Austin Group for reviewing this, and especially to Geoff Clare, Don Cragun, and Jörg Schilling.