**ISO/IEC JTC1 SC22 WG14 N1132**

Date: 2005-08-19

Reference number of document: **ISO/IEC WDTR nnnnn**

Committee identification: ISO/IEC JTC1 SC22 WG14

SC22 Secretariat: ANSI

# Information Technology —

# Programming languages, their environments and system software

# interfaces —

# Specification for Managed Strings —

Dr. Fred Long
Department of Computer Science
University of Wales, Aberystwyth

Robert C. Seacord
CERT/CC
Carnegie Mellon University

# 1 Managed String handling <string_m.h>

## 1.1 Introduction

### 1.1.1 String manipulation errors

Many vulnerabilities in C programs arise through the use of the standard C string manipulating functions. String manipulation errors include buffer overflow through string copying, truncation errors, termination errors and improper data sanitization.

Buffer overflow can easily occur when copying strings if the fixed-length destination of the copy is not large enough to accommodate the source of the string. This is a particular problem when the source is user input, which is potentially unbounded. The usual programming practice is to allocate a character array that is generally large enough. The problem is that this can easily be exploited by malicious users who can supply a carefully crafted string that overflows the fixed length array in such a way that the security of the system is compromised. This is still the most common exploit in fielded code today.

In attempting to overcome the buffer overflow problem, some programmers try to limit the number of characters that are copied. This can result in strings being improperly truncated. This, in turn, results in a loss of data which may lead to a different type of software vulnerability.

A special case of truncation error is a termination error. Many of the standard C string functions rely on strings being null terminated. However, the length of a string does not include the null character. If just the non-null characters of a string are copied then the resulting string may become improperly terminated. A subsequent access may run off the end of the string and corrupt data that should not have been touched.

Finally, inadequate data sanitization can also lead to vulnerabilities. Many applications require data to be constrained not to contain certain characters. Very often, malicious users can be prevented from exploiting an application by ensuring that the illegal characters are not copied into the strings destined for the application.

### 1.1.2 Proposed solution

A secure string library should provide facilities to guard against the problems described above. Furthermore, it should satisfy the following requirements:

1. Operations should succeed or fail unequivocally. There should be no undefined behavior.
2. The facilities should be familiar to C programmers so that they can easily be adopted and existing code easily converted.
3. There should be no surprises in using the facilities. The new facilities should have similar semantics to the standard C string manipulating functions. Again, this will help with the conversion of legacy code.

Of course, some compromise is needed in order to meet these requirements. For example, it is not possible to completely preserve the existing semantics and provide protection against the problems described above.

Libraries that provide string manipulation functions can be categorized as static or dynamic. Static libraries rely on fixed-length arrays. A static approach cannot easily overcome the problems described. With a dynamic approach, strings are resized as necessary. This approach can more easily solve the problems, but a consequence is that memory can be exhausted if input is not limited. This can lead to denial-of-service attacks. Nevertheless, a dynamic managed string library is proposed since this leads to the most secure alternative.

### 1.1.3 The managed string library

This managed string library was developed in response to the need for a string library that can improve the quality and security of newly developed C language programs while eliminating obstacles to widespread adoption and possible standardization.

The managed string library is based on a dynamic approach in that memory is allocated and reallocated as required. This approach eliminates the possibility of unbounded copies, null-termination errors, and truncation by ensuring there is always adequate space available for the resulting string (including the terminating null character). The one exception is if memory is exhausted, which is treated as a constraint violation. In this way, the managed string library accomplishes the goal of succeeding or failing loudly.

The managed string library also provides a built in mechanism for dealing with data sanitization by (optionally) ensuring that all characters in a string belong to a predefined set of "safe" characters.

# 2 Terms, definitions, and symbols

For the purposes of this Technical Report, the following definitions apply. Other terms are defined where they appear in *italic* type. Terms explicitly defined in this Technical Reports are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not defined in this Technical Report are to be interpreted according to ISO/IEC 9899:1999 and ISO/IEC 2382−1. Mathematical symbols not defined in this Technical Report are to be interpreted according to ISO 31−11.

### 2.1 constraint

restriction, either syntactic or semantic, by which the exposition of language elements is to be interpreted; or, requirement on a program when calling a library function.

# 3 Library

## 3.1 Use of errno

An implementation may set **`errno`** for the functions defined in this technical report, but is not required to.

## 3.2 Constraint Violations

Most functions in this technical report include as part of their specification a list of constraints. These constraints are requirements on the program using the library.

Implementations shall check that the constraints specified for a function are met by the program. If a constraint is violated, the implementation shall call the currently registered constraint handler (see **set_constraint_handler** in **<stdlib.h>**). Multiple constraint violations in the same call to a library function result in only one call to the constraint handler. It is unspecified which one of the multiple constraint violations cause the handler to be called.

The constraint handler might not return. If the constraint handler does return, the library function whose constraint was violated should return some indication of failure as given in the function's specification.

Implementations are free to detect any case of undefined behavior and treat it as a constraint violation by calling the constraint handler. This license comes directly from the definition of undefined behavior.

### 3.3 Errors <errno.h>

The header **<errno.h>** defines a type.

The type is
**errno_t**
which is type **int**.

# 4 Library functions

## 4.1 Utility functions

### 4.1.1 The isnullstr_m function

**Synopsis**

```
#include <string_m.h>
errno_t isnullstr_m(const string_m s, int *nullstr);
```

**Constraints**

**s** shall not be a null pointer. **s** shall reference a valid managed string. **nullstr** shall not be a null pointer.

**Description**

The **isnullstr_m** function tests whether the string represented by **s** is null and delivers this result in the parameter referenced by **nullstr**, given the managed string **s**.

**Returns**

The **isnullstr_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.2 The `isemptystr_m` function

**Synopsis**

```
#include <string_m.h>
errno_t isemptystr_m(const string_m s, int *emptystr);
```

**Constraints**

**`s`** shall not be a null pointer. **`s`** shall reference a valid managed string. **`nullstr`** shall not be a null pointer.

**Description**

The **`isemptystr_m`** function tests whether the string represented by **`s`** is empty and delivers this result in the parameter referenced by **`emptystr`**, given the managed string **`s`**.

**Returns**

The **`isemptystr_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.3 The `strcreate_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strcreate_m(string_m *s, const char *cstr);
```

**Constraints**

There shall be sufficient memory available to create the managed string **`s`**. **`cstr`** shall contain only valid characters.

**Description**

The **`strcreate_m`** function creates a managed string, referenced by **`s`**, given a conventional string **`cstr`** (which may be null or empty).

**Returns**

The **`strcreate_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.4 The `strdelete_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strdelete_m(string_m *s);
```

**Constraints**

**`s`** shall not be a null pointer.

**Description**
The **strdelete_m** function deletes the managed string referenced by **s** (which may be null or empty).

**Returns**
The **strdelete_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.5 The strlen_m function

**Synopsis**

```
#include <string_m.h>
errno_t strlen_m(const string_m s, size_t *size);
```

**Constraints**
**s** shall not be a null pointer. **s** shall not represent a null string. **size** shall not be a null pointer.

**Description**
The **strlen_m** function computes the length of the string represented by the managed string **s** and stores the result into the variable referenced by **size**.

**Returns**
The **strlen_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.6 The getstr_m function

**Synopsis**

```
#include <string_m.h>
errno_t getstr_m(const string_m s, char **string);
```

**Constraints**
**s** shall not be a null pointer. **s** shall reference a valid managed string. **string** shall not be a null pointer. If there is a constraint violation, then the array (if any) pointed to by **string** is not modified.

**Description**
The **getstr_m** function delivers a conventional string into the variable referenced by **string**, given the managed string **s**.

**Returns**
The **getstr_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.1.7 The `strdup_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strdup_m(string_m *s1, const string_m s2);
```

**Constraints**

There shall be sufficient memory available to create the managed string **`s1`**. **`s2`** shall not be null and shall reference a valid managed string.

**Description**

The **`strdup_m`** function creates a duplicate of the managed string **`s2`** (including the terminating null character) in the managed string **`s1`**.

**Returns**

The **`strdup_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.2 Copying functions

### 4.2.1 The `strcpy_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strcpy_m(string_m s1, const string_m s2);
```

**Constraints**

Neither **`s1`** nor **`s2`** shall be null. There shall be sufficient memory available to create the managed string **`s1`**. **`s2`** shall reference a valid managed string. **`s2`** shall contain only valid characters.

**Description**

The **`strcpy_m`** function copies the string represented by the managed string **`s2`** (including the terminating null character) into the string represented by the managed string **`s1`**.

**Returns**

The **`strcpy_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.2.2 The `strncpy_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strncpy_m (string_m s1,
           const string_m s2,
           size_t n);
```

**Constraints**

Neither **s1** nor **s2** shall be null. There shall be sufficient memory available to create the managed string **s1**. **s2** shall reference a valid managed string. **s2** shall contain only valid characters.

**Description**

The **strncpy_m** function copies not more than **n** characters (characters that follow a null character are not copied) from the string represented by the managed string **s2** to the string represented by the managed string **s1**. If the string represented by **s2** is a string that is shorter than **n** characters, null characters are appended to the copy in the array pointed to by **s1**, until **n** characters in all have been written.

**Returns**

The **strncpy_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.3 Concatenation functions

### 4.3.1 The `strcat_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strcat_m(string_m s1, const string_m s2);
```

**Constraints**

Neither **s1** nor **s2** shall be null. Neither **s1** nor **s2** shall represent the null string. There shall be sufficient memory available to create the managed string **s1**. **s2** shall reference a valid managed string. **s2** shall contain only valid characters.

**Description**

The **strcat_m** function concatenates the string represented by the managed string **s2** (including the terminating null character) onto the string represented by the managed string **s1**.

**Returns**

The **strcat_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.3.2 The `strncat_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strncat_m (string_m s1,
           const string_m s2,
           size_t n);
```

**Constraints**

Neither **`s1`** nor **`s2`** shall be null. Neither **`s1`** nor **`s2`** shall represent the null string. There shall be sufficient memory available to create the managed string **`s1`**. **`s2`** shall reference a valid managed string. **`s2`** shall contain only valid characters.

**Description**

The **`strncat_m`** function appends not more than **`n`** characters (a null character and characters that follow it are not appended) from the string represented by the managed string **`s2`** to the end of the string represented by the managed string **`s1`**. The initial character of the string represented by **`s2`** overwrites the null character at the end of the string represented by **`s1`**. A terminating null character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

**Returns**

The **`strncat_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.4 Comparison functions

The sign of a nonzero value delivered by the comparison functions **`strcmp_m`**, and **`strncmp_m`** is determined by the sign of the difference between the values of the first pair of characters (both interpreted as **`unsigned char`**) that differ in the objects being compared.

### 4.4.1 The `strcmp_m` function

**Synopsis**

```
#include <string_m.h>
errno_t strcmp_m (const string_m s1,
           const string_m s2
           int *cmp);
```

**Constraints**

**`cmp`** shall not be null. Neither **`s1`** nor **`s2`** shall be null. Neither **`s1`** nor **`s2`** shall represent the null string.

**Description**
The **strcmp_m** function compares the string represented by the managed string **s1** to the string represented by the managed string **s2**, and delivers the result in the variable **cmp**.

**Returns**
The **strcmp_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.4.2 The strcoll_m function

**Synopsis**

```
#include <string_m.h>
errno_t strcoll_m (const string_m s1,
            const string_m s2
            int *cmp);
```

**Constraints**
**cmp** shall not be null. Neither **s1** nor **s2** shall be null. Neither **s1** nor **s2** shall represent the null string.

**Description**
The **strcoll_m** function compares the string represented by the managed string **s1** to the string represented by the managed string **s2**, both interpreted as appropriate to the **LC_COLLATE** category of the current locale, and delivers the result in the variable **cmp**.

**Returns**
The **strcoll_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.4.3 The strncmp_m function

**Synopsis**

```
#include <string_m.h>
errno_t strncmp_m (const string_m s1,
            const string_m s2,
            size_t n
            int *cmp);
```

**Constraints**
**cmp** shall not be null. Neither **s1** nor **s2** shall be null. Neither **s1** nor **s2** shall represent the null string.

**Description**
The **strncmp_m** function compares not more than **n** characters (characters that follow a null character are not compared) from the string represented by the managed string **s1** to the string represented by the managed string **s2**.

**Returns**
The **strncmp_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.5 Search functions

### 4.5.1 The strtok_m function

**Synopsis**

```
#include <string_m.h>
errno_t strtok_m(string_m token,
                 string_m str,
                 const string_m delim,
                 string_m ptr);
```

**Constraints**
None of **token**, **str**, **delim**, **ptr** shall be null. There shall be sufficient memory available to create the managed string **token**.

**Description**
The **strtok_m** function scans the string represented by **str**. The substring of **str** up to but not including the first occurrence of any of the characters contained in the string represented by **delim** is returned as the string represented by **token**. The remainder of the string represented by **str** (after but not including the first character found from **delim**) is returned as the string represented by **ptr**.

**Returns**

The **strtok_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.6 Sanitization functions

### 4.6.1 The setcharset function

**Synopsis**

```
#include <string_m.h>
errno_t setcharset(const string_m s);
```

**Constraints**
**s** shall not be null. There shall be sufficient memory available to create the managed string **s**. **s** shall not represent the null string. **s** shall reference a valid managed string.

**Description**
The **setcharset** function sets the set of allowable characters to be those in the string represented by **s** (which may be empty).

**Returns**
The **setcharset** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

## 4.7 Printing to string functions

### 4.7.1 The sprintf_m function

**Synopsis**

```
#include <string_m.h>
errno_t sprintf_m(string_m buf, const string_m fmt,
            ...);
```

**Constraints**
Neither **buf** nor **fmt** shall be null. **fmt** shall reference a valid managed string. The represented by **fmt** shall represent a valid format compatible with the arguments after **fmt**. There shall be sufficient memory available to create the managed string **buf**.

**Description**
The **sprint_m** function formats its parameters after the second parameter into a string according to the format contained in the string represented by the managed string **fmt** and delivers the result in the string represented by the managed string **buf**.

**Returns**
The **sprintf_m** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.

### 4.7.2 The vsprintf_m function

**Synopsis**

```
#include <string_m.h>
errno_t vsprintf_m(string_m buf,
            const string_m fmt,
            va_list args);
```

**Constraints**
Neither **buf** nor **fmt** shall be null. **fmt** shall reference a valid managed string. The string represented by **fmt** shall represent a valid format compatible with the arguments **args**. There shall be sufficient memory available to create the managed string **buf**.

**Description**
The **vsprintf_m** function formats its parameters **args** into a string according to the format contained in the string represented by the managed string **fmt** and delivers the result in the string represented by the managed string **buf**.

**Returns**

The **`vsprintf_m`** function returns zero if there was no constraint violation. Otherwise, a non-zero value is returned.