

# Specifications for Managed Strings, Second Edition

Hal Burch (Software Engineering Institute)  
Fred Long (University of Wales, Aberystwyth)  
Raunak Rungta (Software Engineering Institute)  
Robert Seacord (Software Engineering Institute)  
David Svoboda (Software Engineering Institute)

**May 2010**

**TECHNICAL REPORT**  
CMU/SEI-2010-TR-018  
ESC-TR-2010-018

**CERT<sup>®</sup> Program**  
Unlimited distribution subject to the copyright.

<http://www.cert.org>



This report was prepared for the

SEI Administrative Agent  
ESC/XPK  
5 Eglin Street  
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2010 Carnegie Mellon University.

#### NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. This document may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

---

# Table of Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 String Manipulation Errors	1
1.2 Proposed Solution	1
1.3 The Managed String Library	2
1.4 Wide Character and Null-Terminated Byte Strings	2
<b>2 Library</b>	<b>5</b>
2.1 Use of <code>errno</code>	5
2.2 Runtime-Constraint Violations	5
2.3 Errors <code>&lt;errno.h&gt;</code>	5
2.4 Common Definitions <code>&lt;stddef.h&gt;</code>	6
2.5 Integer Types <code>&lt;stdint.h&gt;</code>	6
2.6 Managed String Type <code>&lt;string_m.h&gt;</code>	6
2.7 General Utilities <code>&lt;stdlib.h&gt;</code>	7
<b>3 Library Functions</b>	<b>9</b>
3.1 Utility Functions	9
3.1.1 The <code>isnull_m</code> Function	9
3.1.2 The <code>isempty_m</code> Function	9
3.1.3 Creating a Managed String	9
3.1.4 The <code>isntbs_m</code> Function	12
3.1.5 The <code>iswide_m</code> Function	12
3.1.6 The <code>strdelete_m</code> Function	13
3.1.7 The <code>strlen_m</code> Function	13
3.1.8 Extracting a Conventional String	13
3.1.9 The <code>strdup_m</code> Function	14
3.2 Copying Functions	15
3.2.1 Unbounded String Copy	15
3.2.2 The <code>strncpy_m</code> Function	16
3.3 Concatenation Functions	16
3.3.1 Unbounded Concatenation	16
3.3.2 Bounded Concatenation	17
3.4 Comparison Functions	19
3.4.1 Unbounded Comparison	19
3.4.2 Bounded String Comparison	20
3.5 Search Functions	22
3.5.1 The <code>strtok_m</code> Function	22
3.5.2 The <code>cstrchr_m</code> Function	22
3.5.3 The <code>wstrchr_m</code> Function	23
3.5.4 The <code>strspn_m</code> Function	23
3.5.5 The <code>cstrspn_m</code> Function	23
3.5.6 The <code>wstrspn_m</code> Function	24
3.5.7 The <code>strcspn_m</code> Function	24
3.5.8 The <code>cstrcspn_m</code> Function	25
3.5.9 The <code>wstrcspn_m</code> Function	25
3.6 Configuration Functions	26

3.6.1	The <code>setcharset_m</code> Function	26
3.6.2	The <code>setmaxlen_m</code> Function	26
3.7	Functions Derived from <code>printf</code>	26
3.7.1	The <code>sprintf_m</code> Function	27
3.7.2	The <code>vsprintf_m</code> Function	27
3.7.3	The <code>printf_m</code> Function	28
3.7.4	The <code>vprintf_m</code> Function	28
3.7.5	The <code>fprintf_m</code> Function	29
3.7.6	The <code>vfprintf_m</code> Function	29
3.8	Functions Derived from <code>scanf</code>	30
3.8.1	The <code>sscanf_m</code> Function	30
3.8.2	The <code>vsscanf_m</code> Function	30
3.8.3	The <code>scanf_m</code> Function	31
3.8.4	The <code>vscanf_m</code> Function	31
3.8.5	The <code>fscanf_m</code> Function	32
3.8.6	The <code>vfscanf_m</code> Function	32
3.9	String Slices	33
3.9.1	The <code>strslice_m</code> Function	33
3.9.2	The <code>strleft_m</code> Function	33
3.9.3	The <code>strright_m</code> Function	34
3.9.4	The <code>cchar_m</code> Function	34
3.9.5	The <code>wchar_m</code> Function	35

<b>References</b>	<b>37</b>
-------------------	-----------

---

## Acknowledgments

The authors want to thank David Keaton and Martin Sebor for their valuable contributions to this technical report.



---

## Abstract

This report describes a managed string library for the C programming language. Many software vulnerabilities in C programs result from the misuse of manipulation functions for standard C strings. Programming errors common to string-manipulation logic include buffer overflow, truncation errors, string termination errors, and improper data sanitization. The managed string library provides mechanisms to eliminate or mitigate these problems and improve system security. The CERT<sup>®</sup> Program, which is part of the Carnegie Mellon<sup>®</sup> Software Engineering Institute, provides a proof-of-concept implementation of the managed string library on its Secure Coding web pages.





---

# 1 Introduction

## 1.1 String Manipulation Errors

Many software vulnerabilities in C programs arise through the misuse of manipulation functions for standard C strings. String manipulation programming errors include truncation errors, termination errors, improper data sanitization, and buffer overflow through string copying.

Buffer overflow can easily occur during string copying if the fixed-length destination of the copy is not large enough to accommodate the source 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. However, this fixed-length array can still be exploited by a malicious user who supplies a carefully crafted string that overflows the array in a way that compromises the security of the system. This is the most common exploit in fielded C code today.

In attempting to overcome the buffer overflow problem, some programmers limit the number of characters that are copied. This can result in strings being improperly truncated, which in turn results in a loss of data that can 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, the resulting string may not be properly terminated. A subsequent access may run off the end of the string, corrupting data that should not have been touched.

Finally, inadequate data sanitization can also lead to software vulnerabilities. To function properly, many applications require that data does not contain certain characters. Ensuring that the strings used by the application do not include illegal characters can often prevent malicious users from exploiting an application.

## 1.2 Proposed Solution

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

- Operations should succeed or fail unequivocally.
- The facilities should be familiar to C programmers to facilitate both their adoption and the conversion of existing code.
- Using the facilities should not involve any surprises. The new facilities should have semantics similar to the manipulation functions for standard C strings. Again, this will help with the conversion of legacy code.

Of course, some compromises are needed to meet these requirements. For example, it is not possible to completely preserve the existing semantics and provide protection against the programming errors 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 overcome the errors described above as easily as a dynamic approach. With a dynamic approach, strings are resized as necessary, but a consequence is that memory can be exhausted if input is not limited. To mitigate this problem, the managed string library allows for the specification of a per-string maximum length.

### 1.3 The Managed String Library

The CERT<sup>®</sup> Program, which is part of the Carnegie Mellon<sup>®</sup> Software Engineering Institute, has developed a proof-of-concept implementation of the managed string library in response to the need for a string library that could improve the quality and security of newly developed C language programs while eliminating obstacles to widespread adoption and possible standardization [CERT 2009]. The managed string library is available on the CERT Secure Coding website, <http://www.cert.org/secure-coding/managedstring.html>.

The managed string library is based on a dynamic approach where memory is allocated and reallocated as required. This approach eliminates the possibility of unbounded copies, null-termination errors, and truncation by ensuring adequate space is always available for the resulting string (including the terminating null character).

A runtime-constraint violation occurs when memory cannot be allocated. In this way, the managed string library accomplishes the goal of succeeding or failing unequivocally.

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

### 1.4 Wide Character and Null-Terminated Byte Strings

A number of managed string functions

- accept either a null-terminated byte string or a wide character string as input
- provide one of those string types as a return value

The managed string library works equally well with either type of string. For example, it is possible to create a managed string from a wide character string and then extract a null-terminated byte string (or vice versa). It is also possible to copy a null-terminated byte string and then concatenate a wide character string. Managed string functions will handle conversions implicitly when possible. If a conversion cannot be performed, the operation is halted and a runtime-constraint error is reported.

Strings are maintained in the format in which they are initially provided, until such a time that a conversion is necessary. String promotions are relatively simple: performing an operation on two null-terminated byte strings results in a null-terminated byte string, an operation on a null-terminated byte string and a wide character string results in a wide character string, and operations on two wide character strings result in a wide character string. Conversions are performed as necessary in the locale defined at the time the conversion occurs.

---

<sup>®</sup> Carnegie Mellon and CERT are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Managed strings also support the definition of a restricted character set that identifies the set of allowable characters for the string. When an operation requires that a null-terminated byte string be converted to a wide character string, the restricted character set is also converted as part of the operation.



---

## 2 Library

### 2.1 Use of `errno`

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

### 2.2 Runtime-Constraint Violations

Most functions in this technical report include as part of their specifications a list of runtime-constraints, which are requirements on the program using the library. Despite its name, a runtime-constraint is not a kind of constraint. Implementations shall verify that the runtime-constraint for a library function are not violated by the program.

Implementations shall check that the runtime-constraints specified for a function are met by the program. If a runtime-constraint is violated, the implementation shall call the currently registered constraint handler (see `set_constraint_handler` in Section 2.7). Multiple runtime-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 runtime-constraint violations cause the handler to be called.

Sometimes the runtime-constraints section for a function states an action to be performed if a runtime-constraint violation occurs. Such actions are performed before calling the runtime-constraint handler. Sometimes the runtime-constraints section lists actions that are prohibited if a runtime-constraint violation occurs. Such actions are prohibited to the function both before the handler is called and after the handler returns.

The runtime-constraint handler may not return. If it does, the library function whose runtime-constraint was violated shall return some indication of failure as given by the returns section in the function's specification.

Although runtime-constraints replace many cases of undefined behavior from ISO/IEC 9899:1999 [ISO/IEC 1999], undefined behavior can still occur. Implementations are free to detect any case of undefined behavior and treat it as a runtime-constraint violation by calling the runtime-constraint handler. This license comes directly from the definition of undefined behavior.

### 2.3 Errors `<errno.h>`

The header `<errno.h>` defines the following type, which is `int`:

```
errno_t
```

## 2.4 Common Definitions <stddef.h>

The <stddef.h> header defines the following type, which is `size_t`:<sup>1</sup>

```
    rsize_t
```

## 2.5 Integer Types <stdint.h>

The <stdint.h> header defines the following macro, which expands to a value of type `size_t`.<sup>2</sup>

```
    RSIZE_MAX
```

Functions that have parameters of type `rsize_t` consider it a runtime-constraint violation if the values of those parameters are greater than `RSIZE_MAX`.

### Recommended Practice

Extremely large object sizes are frequently a sign that an object's size was calculated incorrectly. For example, negative numbers appear as very large positive numbers when converted to an unsigned type such as `size_t`. Also, some implementations do not support objects as large as the maximum value that can be represented by type `size_t`.

For those reasons, it is sometimes beneficial to restrict the range of object sizes to detect programming errors. For implementations targeting machines with large address spaces, `RSIZE_MAX` should be defined as the smaller of the size of the largest object supported or  $(\text{SIZE\_MAX} \gg 1)$ , even if this limit is smaller than the size of some legitimate, but very large, objects. Implementations targeting machines with small address spaces may wish to define `RSIZE_MAX` as `SIZE_MAX`, which means that no object size is considered a runtime-constraint violation.

## 2.6 Managed String Type <string\_m.h>

The <string\_m.h> header defines an abstract data type:

```
    typedef struct string_mx string_mx;
```

The structure referenced by this data type is private and implementation defined. All managed strings of this type have a maximum string length that is determined when the string is created. For functions that have parameters of type pointer to `string_mx`, it is a runtime-constraint violation if the maximum length of a managed string is exceeded.

Managed strings can also have a defined set of valid characters that can be used in the string. For functions that have parameters of type pointer to `string_mx`, it is a runtime-constraint violation if a managed string contains invalid characters. For functions that have parameters of type pointer

---

<sup>1</sup> See the description of the `RSIZE_MAX` macro in <stdint.h>.

<sup>2</sup> The `RSIZE_MAX` macro does not have to expand to a constant expression.

to `string_mx`, it is a runtime-constraint violation if the request requires allocating more memory than is available.<sup>3</sup>

Managed strings support both null and empty strings. An empty string is one that has zero characters. A null string is an uninitialized string or a string that has been explicitly set to null.

For computing the length of a string to determine if the maximum length is exceeded, the length of a null-terminated byte string is the number of bytes, and the length of a wide character string is the number of characters. Thus, promoting a multi-byte, null-terminated byte string may change its length. Constants strings can be created by defining the structure `string_mx` to be constant and then calling the `const_strcreate_m` function.

## 2.7 General Utilities <stdlib.h>

The header <stdlib.h> defines six types:

- `errno_t`, which is type `int`
- `rsize_t`, which is type `size_t`
- `constraint_handler_t`, which has the definition

```
typedef void (*constraint_handler_t)(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error);
```
- `malloc_handler_t`, which has the definition

```
typedef void * (*malloc_handler_t)(
    size_t size);
```
- `realloc_handler_t`, which has the definition

```
typedef void * (*realloc_handler_t)(
    void * ptr, size_t size);
```
- `free_handler_t`, which has the definition

```
typedef void (*free_handler_t)(void *ptr);
```

---

<sup>3</sup> The library depends on `malloc()` and `realloc()` returning a null pointer to signify insufficient memory. On some systems, particularly systems using optimistic memory allocation schemes, `malloc()` may return a non-null pointer even when there is insufficient memory. On systems where there is no such mechanism to detect out-of-memory conditions, the library will not be able to properly validate this condition.





---

## 3 Library Functions

### 3.1 Utility Functions

#### 3.1.1 The `isnull_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t isnull_m(const string_mx * s, _Bool *nullstr);
```

##### Runtime-Constraints

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

##### Description

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

##### Returns

The `isnull_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.1.2 The `isempty_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t isempty_m(const string_mx * s, _Bool *emptystr);
```

##### Runtime-Constraints

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

##### Description

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

##### Returns

The `isempty_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.1.3 Creating a Managed String

##### 3.1.3.1 The `strcreate_m` Function

##### Synopsis

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

### Runtime-Constraints

`s` shall not be a null pointer. `charset` shall not be an empty string (denoted by ""). Invalid characters are not present in the C string passed to the function.

### Description

The `strcreate_m` function creates a managed string, referenced by `s`, given a conventional string `cstr` (which may be null or empty). `maxsize` specifies the maximum length of the string in characters. If `maxsize` is 0, the system-defined maximum size is used. `charset` restricts the set of allowable characters to those in the null-terminated byte string `cstr` (which may be empty). If `charset` is a null pointer, no restricted character set is defined. If specified, duplicated characters in a `charset` are ignored. Characters in the `charset` may be provided in any order. The `\0` character cannot be specified as part of `charset`.

### Returns

The `strcreate_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.1.3.2 The `wstrcreate_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t wstrcreate_m(string_mx **s,
                    const wchar_t *cstr,
                    const size_t maxsize,
                    const wchar_t *charset);
```

### Runtime-Constraints

`s` shall not be a null pointer. `charset` shall not be an empty string (denoted by L ""). Invalid characters are not present in the C string passed to the function.

### Description

The `wstrcreate_m` function creates a managed string, referenced by `s`, given a wide character string `cstr` (which may be null or empty). `maxsize` specifies the maximum size of the string in characters. If `maxsize` is zero, the system-defined maximum length is used. `charset` restricts the set of allowable characters to those in the wide character string `cstr` (which may be empty). If `charset` is a null pointer, no restricted character set is defined. Characters in the `charset` may be provided in any order. The `\0` character cannot be specified as part of `charset`.

### Returns

The `wstrcreate_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.1.3.3 The `const_strcreate_m` function

##### Synopsis

```
#include <string_m.h>
```

```

errno_t const_strcreate_m(const string_mx **str,
                          const char *cstr,
                          const size_t maxsize,
                          const char *charset);

```

### Runtime-Constraints

`str` shall not be a null pointer. `charset` shall not be an empty string (denoted by ""). Memory allocation for the string should succeed. Invalid characters are not present in the C string passed to the function.

### Description

The `const_strcreate_m` function creates a constant managed string, given a conventional constant C string `cstr` (which may be null or empty). `maxsize` specifies the maximum size of the string in characters. If `maxsize` is 0, the system-defined maximum length is used. `charset` restricts the set of allowable characters to those in the null-terminated byte string `cstr` (which may be empty). If `charset` is a null pointer, no restricted character set is defined. If specified, duplicated characters in a `charset` are ignored. Characters in the `charset` may be provided in any order. The `\0` character cannot be specified as part of `charset`. The pointer to a constant string structure is returned to the caller by storing it in the parameter passed to the function.

This function acts as a wrapper function to the `strcreate_m` function. It passes all the arguments to the `strcreate_m` function to create a managed string. The pointer of that managed string is returned to the user as a pointer to the constant managed string.

### Returns

The `const_strcreate_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.1.3.4 The `const_wstrcreate_m` function

#### Synopsis

```

#include <string_m.h>
errno_t const_wstrcreate_m(string_mx **str,
                          const wchar_t *wcstr,
                          const size_t maxsize,
                          const wchar_t *charset);

```

### Runtime-Constraints

`str` shall not be a null pointer. `charset` shall not be an empty string (denoted by L ""). Memory allocation for the string should succeed. Invalid characters are not present in the C string passed to the function.

### Description

The `const_wstrcreate_m` function creates a constant managed string, referenced by `str`, given a wide character string `wcstr` (which may be null or empty). `maxsize` specifies the maximum size of the string in characters. If `maxsize` is zero, the system-defined maximum

length is used. `charset` restricts the set of allowable characters to those in the wide character string `wcstr` (which may be empty). If `charset` is a null pointer, no restricted character set is defined. Characters in the `charset` may be provided in any order. The `\0` character cannot be specified as part of `charset`.

#### Returns

The `const_wstrcreate_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.1.4 The `isntbs_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t isntbs_m(const string_mx * s,
                 _Bool *ntbstr);
```

#### Runtime-Constraints

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

#### Description

The `isntbs_m` function tests whether the managed string `s` is a null-terminated byte string and delivers this result in the parameter referenced by `ntbstr`, given the managed string `s`.

#### Returns

The `isntbs_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.1.5 The `iswide_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t iswide_m(const string_mx * s,
                 _Bool *widestr);
```

#### Runtime-Constraints

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

#### Description

The `iswide_m` function tests whether the managed string `s` is a wide character string and delivers this result in the parameter referenced by `widestr`, given the managed string `s`.

#### Returns

The `iswide_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.1.6 The `strdelete_m` Function

#### Synopsis

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

#### Runtime-Constraints

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

#### Description

The `strdelete_m` function deletes the managed string referenced by `**s` (which may be null or empty). `s` is set to a null pointer.

#### Returns

The `strdelete_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.1.7 The `strlen_m` Function

#### Synopsis

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

#### Runtime-Constraints

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

#### Description

The `strlen_m` function computes the length of the constant managed string `s` and stores the result into the variable referenced by `size`. If the managed string is either null or empty, the length is computed as 0. For a null-terminated byte string, the length is the number of bytes. For a wide character string, the length is the number of characters.

#### Returns

The `strlen_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.1.8 Extracting a Conventional String

#### 3.1.8.1 The `cgetstr_m` Function

#### Synopsis

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

#### Runtime-Constraints

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

## Description

The `cgetstr_m` function allocates storage for, and returns a pointer to, a null-terminated byte string represented by the managed string `s` and referenced by `string`. The caller is responsible for freeing `*string` when the null-terminated byte string is no longer required.

## Example

```
if (retValue = cgetstr_m(str1, &cstr)) {
    fprintf(stderr, "error %d from cgetstr_m.\n", retValue);
} else {
    printf("(%s)\n", cstr);
    free(cstr); // free duplicate string
}
```

## Returns

The `cgetstr_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned. If there is a runtime-constraint violation, `*string` is set to a null pointer.

### 3.1.8.2 The `wgetstr_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t wgetstr_m( const string_mx * s,
                  const wchar_t **wctr);
```

#### Runtime-Constraints

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

#### Description

The `wgetstr_m` function delivers a wide character string into the variable referenced by `wctr`, given the managed string `s`. The caller is responsible for freeing `*wctr` when the wide character string is no longer required.

#### Returns

The `wgetstr_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned. If there is a runtime-constraint violation, `*wctr` is set to a null pointer.

### 3.1.9 The `strdup_m` Function

#### Synopsis

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

#### Runtime-Constraints

`s1` shall not be a null pointer. `s2` shall reference a valid managed string.

#### Description

The `strdup_m` function creates a duplicate of the managed string `s2` and stores it in `s1`. The duplicate shall have the same set of valid characters and maximum length.

## Returns

The `strdup_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

## 3.2 Copying Functions

### 3.2.1 Unbounded String Copy

#### 3.2.1.1 The `strcpy_m` Function

##### Synopsis

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

##### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings.

##### Description

The `strcpy_m` function copies the managed string `s2` into the managed string `s1`. Note that the set of valid characters and maximum length are not copied as these are attributes of `s1`.<sup>4</sup>

##### Returns

The `strcpy_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.2.1.2 The `cstrcpy_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t cstrcpy_m(string_mx *s1, const char *cstr);
```

##### Runtime-Constraints

`s1` shall reference a valid managed string.

##### Description

The `cstrcpy_m` function copies the string `cstr` into the managed string `s1`.

##### Returns

The `cstrcpy_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.2.1.3 The `wstrcpy_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t wstrcpy_m(string_mx *s1, const wchar_t *wcstr);
```

---

<sup>4</sup> If `s2` contains characters that are not in the set of valid characters or exceeds the maximum length defined for `s1`, a runtime-constraint violation occurs as described in Section 2.6.

### Runtime-Constraints

`s1` shall reference a valid managed string.

### Description

The `wstrncpy_m` function copies the string `wcstr` into the managed string `s1`.

### Returns

The `wstrncpy_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

## 3.2.2 The `strncpy_m` Function

### Synopsis

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

### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings.

### Description

The `strncpy_m` function copies not more than `n` characters from the managed string `s2` to the managed string `s1`. If `s2` does not contain `n` characters, the entire string is copied. If `s2` contains more than `n` characters, `s1` is set to the string containing the first `n` characters.

### Returns

The `strncpy_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

## 3.3 Concatenation Functions

### 3.3.1 Unbounded Concatenation

#### 3.3.1.1 The `strcat_m` Function

### Synopsis

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

### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings.

### Description

The `strcat_m` function concatenates the managed string `s2` onto the end of the managed string `s1`.



## Returns

The `strcat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.3.1.2 The `cstrcat_m` Function

#### Synopsis

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

#### Runtime-Constraints

`s` shall reference a valid managed string.

#### Description

The `cstrcat_m` function concatenates the null-terminated byte string `cstr` onto the end of the managed string `s`. If `cstr` is a null pointer, this function returns without modifying `s`.

## Returns

The `cstrcat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.3.1.3 The `wstrcat_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t wstrcat_m(string_mx *s, const wchar_t *wcstr);
```

#### Runtime-Constraints

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

#### Description

The `wstrcat_m` function concatenates the wide character string `wcstr` onto the end of the managed string `s`. If `wcstr` is a null pointer, this function returns without modifying `s`.

## Returns

The `wstrcat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

## 3.3.2 Bounded Concatenation

### 3.3.2.1 The `strncat_m` Function

#### Synopsis

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

### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings.

### Description

The `strncat_m` function appends not more than `n` characters from the managed string `s2` to the end of the managed string `s1`. If `s2` is a null pointer, `strncat_m` returns without modifying `s1`.

### Returns

The `strncat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.3.2.2 The `cstrncat_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t cstrncat_m (string_mx *s,
                  const char *cstr,
                  rsize_t n);
```

### Runtime-Constraints

`s` shall reference a valid managed string.

### Description

The `cstrncat_m` function appends not more than `n` bytes from the null-terminated byte string `cstr` to the end of the managed string `s`. If `cstr` is null, `cstrncat_m` returns without modifying `s`. The `cstrncat_m` function guarantees that the resulting string `s` is properly terminated.

### Returns

The `cstrncat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.3.2.3 The `wstrncat_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t wstrncat_m (string_mx *s,
                  const wchar_t *wcstr,
                  rsize_t n);
```

### Runtime-Constraints

`s` shall reference a valid managed string.

### Description

The `wstrncat_m` function appends not more than `n` characters from the wide character string `wcstr` to the end of the managed string `s`. If `wcstr` is a null pointer, the `wstrncat_m` func-

tion returns without modifying *s*. The `wstrncat_m` function guarantees that the resulting string *s* is properly terminated.

#### Returns

The `wstrncat_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.4 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` but promoted to `int`) that differ in the objects being compared.

For the purpose of comparison, a null string is less than any other string, including an empty string. Null strings are equal, and empty strings are equal.

The set of valid characters defined for each string is not a factor in the evaluation; however, it is held as an invariant that each managed string contains only characters identified as valid for that string.

#### 3.4.1 Unbounded Comparison

##### 3.4.1.1 The `strcmp_m` Function

###### Synopsis

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

###### Runtime-Constraints

*s1* and *s2* shall reference valid managed strings. *cmp* shall not be a null pointer.

###### Description

The `strcmp_m` function compares the constant managed string *s1* to the constant managed string *s2* and sets *cmp* to an integer value greater than, equal to, or less than 0 accordingly as *s1* is greater than, equal to, or less than *s2*.

###### Returns

The `strcmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

##### 3.4.1.2 The `cstrcmp_m` Function

###### Synopsis

```
#include <string_m.h>
errno_t cstrcmp_m ( const string_mx * s1,
                   const char *cstr,
                   int *cmp);
```

### Runtime-Constraints

`s1` shall reference a valid managed string. `cmp` shall not be a null pointer.

### Description

The `cstrcmp_m` function compares the managed string `s1` to the null-terminated byte string `cstr` and sets `cmp` to an integer value greater than, equal to, or less than 0 accordingly as `s1` is greater than, equal to, or less than `cstr`.

### Returns

The `cstrcmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.4.1.3 The `wstrcmp_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t wstrcmp_m ( const string_mx * s1,
                   const wchar_t *wstr,
                   int *cmp);
```

### Runtime-Constraints

`s1` shall reference a valid managed string. `cmp` shall not be a null pointer.

### Description

The `wstrcmp_m` function compares the managed string `s1` to the wide character string `wstr` and sets `cmp` to an integer value greater than, equal to, or less than 0 accordingly as `s1` is greater than, equal to, or less than `wstr`.

### Returns

The `wstrcmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.4.2 Bounded String Comparison

##### 3.4.2.1 The `strncmp_m` Function

##### Synopsis

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

### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings. `cmp` shall not be a null pointer.

### Description

The `strncmp_m` function compares not more than `n` characters (characters that follow a null character are not compared) from the managed string `s1` to the managed string `s2` and sets `cmp`

to an integer value greater than, equal to, or less than 0 accordingly as `s1` is greater than, equal to, or less than `s2`. If `n` is equal to 0, the `strncmp_m` function sets `cmp` to the integer value 0, regardless of the contents of the string.

#### Returns

The `strncmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

#### 3.4.2.2 The `cstrncmp_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t cstrncmp_m ( const string_mx * s1,
                    const char *cstr, rsize_t n,
                    int *cmp);
```

##### Runtime-Constraints

`s1` shall reference a valid managed string. `cmp` shall not be a null pointer.

##### Description

The `cstrncmp_m` function compares not more than `n` bytes (bytes that follow a null character are not compared) from the managed string `s1` to the null-terminated byte string `cstr` and sets `cmp` to an integer value greater than, equal to, or less than 0 accordingly as `s1` is greater than, equal to, or less than `cstr`. If `n` is equal to 0, the `cstrncmp_m` function sets `cmp` to the integer value 0, regardless of the contents of the string.

#### Returns

The `cstrncmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

#### 3.4.2.3 The `wstrncmp_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t wstrncmp_m ( const string_mx * s1,
                    const wchar_t *wstr, rsize_t n,
                    int *cmp);
```

##### Runtime-Constraints

`s1` shall reference a valid managed string. `cmp` shall not be a null pointer.

##### Description

The `wstrncmp_m` function compares not more than `n` characters (characters that follow a null character are not compared) from managed string `s1` to the wide character string `wstr` and sets `cmp` to an integer value greater than, equal to, or less than 0 accordingly as `s1` is greater than, equal to, or less than `wstr`. If `n` is equal to 0, the `wstrncmp_m` function sets `cmp` to the integer value 0 regardless of the contents of the string.

## Returns

The `wstrncmp_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

## 3.5 Search Functions

### 3.5.1 The `strtok_m` Function

#### Synopsis

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

#### Runtime-Constraints

`token`, `str`, `delim`, and `ptr` shall reference valid managed strings.

#### Description

The `strtok_m` function scans the managed string `str`. The substring of `str`, up to but not including the first occurrence of any of the characters contained in the managed string `delim`, is returned as the managed string `token`. The remainder of the managed string `str`, after but not including the first character found from `delim`, is returned as the managed string `ptr`. If `str` does not contain any characters in `delim` (or if `delim` is either empty or null), `token` shall be set to `str`, and `ptr` will be set to the null string.

#### Returns

The `strtok_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.2 The `cstrchr_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t cstrchr_m(const string_mx * str,
                  char c,
                  rsize_t *index);
```

#### Runtime-Constraints

`str` shall reference valid managed strings.

#### Description

The `cstrchr_m` function scans the managed string `str` for the first occurrence of `c`. The parameter `index` is set to the first occurrence of character `c` in the string `str`. If `c` is not found in `str`, the index references to `~0`.

## Returns

The `cstrchr_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.3 The `wstrchr_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t wstrchr_m( const string_mx * str,
                  wchar_t wc,
                  rsize_t *index);
```

#### Runtime-Constraints

`str` shall reference valid managed strings.

#### Description

The `wstrchr_m` function scans the managed string `str` for the first occurrence of `wc`. The parameter `index` is set to the first occurrence of wide character `c` in the string `str`. If `c` is not found in `str`, the `index` references to `-0`.

## Returns

The `wstrchr_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.4 The `strspn_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t strspn_m(string_mx *str, string_mx *accept,
                 rsize_t *len);
```

#### Runtime-Constraints

`str` and `accept` shall reference a valid managed string. `len` shall not be a null pointer.

#### Description

The `strspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters from the managed string `accept`. The `strspn_m` function sets `*len` to this length. If the managed string `str` is null or empty, `*len` is set to 0.

## Returns

The `strspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.5 The `cstrspn_m` Function

#### Synopsis

```
#include <string_m.h>
```

```
    errno_t cstrspn_m(string_mx *str, const char *accept,
                    rsize_t *len);
```

#### Runtime-Constraints

`str` and `accept` shall reference a valid managed string. `len` shall not be a null pointer.

#### Description

The `cstrspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters from the string `accept`. The `cstrspn_m` function sets `*len` to this length. If the managed string `str` is null or empty, `*len` is set to 0.

#### Returns

The `cstrspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.6 The `wstrspn_m` Function

#### Synopsis

```
#include <string_m.h>
    errno_t wstrspn_m(string_mx *str, const wchar_t *accept,
                    rsize_t *len);
```

#### Runtime-Constraints

`str` and `accept` shall reference a valid managed string. `len` shall not be a null pointer.

#### Description

The `wstrspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters from the string `accept`. The `wstrspn_m` function sets `*len` to this length. If the managed string `str` is null or empty, `*len` is set to 0.

#### Returns

The `wstrspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.7 The `strcspn_m` Function

#### Synopsis

```
#include <string_m.h>
    errno_t strcspn_m(string_mx *str, string_mx *reject,
                    rsize_t *len);
```

#### Runtime-Constraints

`str` and `reject` shall reference valid managed strings. `len` shall not be a null pointer.

#### Description

The `strcspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters *not* from the managed string `reject`. The `strcspn_m` function sets `*len` to this length. If the managed string `str` is null or empty,



\*len is set to 0. If the managed string `reject` is null or empty, \*len is set to the length of `str`.

#### Returns

The `strcspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.8 The `cstrcspn_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t cstrcspn_m(string_mx *str, const char *reject,
                  rsize_t *len);
```

#### Runtime-Constraints

`str` and `reject` shall reference valid managed strings. `len` shall not be a null pointer.

#### Description

The `cstrcspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters *not* from the managed string `reject`. The `cstrcspn_m` function sets \*len to this length. If the managed string `str` is null or empty, \*len is set to 0. If the managed string `reject` is null or empty, \*len is set to the length of `str`.

#### Returns

The `cstrcspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

### 3.5.9 The `wstrcspn_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t wstrcspn_m(string_mx *str, const wchar_t *reject,
                  rsize_t *len);
```

#### Runtime-Constraints

`str` and `reject` shall reference valid managed strings. `len` shall not be a null pointer.

#### Description

The `wstrcspn_m` function computes the length of the maximum initial segment of the managed string `str`, which consists entirely of characters *not* from the managed string `reject`. The `wstrcspn_m` function sets \*len to this length. If the managed string `str` is null or empty, \*len is set to 0. If the managed string `reject` is null or empty, \*len is set to the length of `str`.

## Returns

The `wstrcspn_m` function returns 0 if there was no runtime-constraint violation. Otherwise, a nonzero value is returned.

## 3.6 Configuration Functions

### 3.6.1 The `setcharset_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t setcharset_m(string_mx *s,
                    const string_mx * charset);
```

#### Runtime-Constraints

`s` shall reference a valid managed string.

#### Description

The `setcharset_m` function sets the subset of allowable characters to those in the managed string `charset`, which may be null or empty. If `charset` is a null pointer or the managed string represented by `charset` is null, a restricted subset of valid characters is not enforced. If the managed string `charset` is empty, then only empty or null strings can be created.

#### Returns

The `setcharset_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.6.2 The `setmaxlen_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t setmaxlen_m(string_mx *s, rsize_t maxlen);
```

#### Runtime-Constraints

`s` shall reference a valid managed string.

#### Description

The `setmaxlen_m` function sets the maximum length of the managed string `s`. If `maxlen` is 0, the system-defined maximum length is used.

#### Returns

The `setmaxlen_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned..

## 3.7 Functions Derived from `printf`

These functions are the managed string equivalents to the `printf`-derived functions in C.

The `%s` format specification refers to a managed string, rather than a null-terminated byte string or wide character string. The format specification `%ls` indicates that the managed string should be output as a wide character string. The format specification `%hs` indicates that the managed string should be output as a null-terminated byte string. All `printf`-derived functions will output a null-terminated byte string if (1) any specified output stream is byte oriented and (2) the format string and all argument strings are null-terminated byte strings; otherwise the output will be a wide character string.

Applying a byte output function to a wide-oriented stream or applying a wide character output function to a byte-oriented stream will result in a runtime-constraint error.

The `%n` format specification is not recognized.

### 3.7.1 The `sprintf_m` Function

#### Synopsis

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

#### Runtime-Constraints

`buf` and `fmt` shall reference valid managed strings. The managed string `fmt` shall be a valid format compatible with the arguments after `fmt`.

#### Description

The `sprintf_m` function formats its parameters after the third parameter into a string according to the format contained in the managed string `fmt` and stores the result in the managed string `buf`.

If not a null pointer, `*count` is set to the number of characters written in `buf`, not including the terminating null character.

#### Returns

The `sprintf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.7.2 The `vsprintf_m` Function

#### Synopsis

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

#### Runtime-Constraints

`buf` and `fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments `args`.

## Description

The `vsprintf_m` function formats its parameters `args` into a string according to the format contained in the managed string `fmt` and stores the result in the managed string `buf`.

If not a null pointer, `*count` is set to the number of characters written in `buf`, not including the terminating null character.

## Returns

The `vsprintf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.7.3 The `printf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t printf_m(const string_mx * fmt, int *count, ...);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `fmt`.

## Description

The `printf_m` function formats its parameters after the second parameter into a string according to the format contained in the managed string `fmt` and outputs the result to standard output.

If not a null pointer, `*count` is set to the number of characters transmitted.

## Returns

The `printf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.7.4 The `vprintf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t vprintf_m(const string_mx * fmt, int *count,
                 va_list args);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments `args`.

## Description

The `vprintf_m` function formats its parameters `args` into a string according to the format contained in the managed string `fmt` and outputs the result to standard output.

If not a null pointer, `*count` is set to the number of characters transmitted.

## Returns

The `vprintf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.7.5 The `fprintf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t fprintf_m(FILE *file, const string_mx *fmt, int
                 *count, ...);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `fmt`. `file` shall not be a null pointer.

If not a null pointer, `*count` is set to the number of characters transmitted.

#### Description

The `fprintf_m` function formats its parameters after the third parameter into a string according to the format contained in the managed string `fmt` and outputs the result to `file`.

#### Returns

The `fprintf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.7.6 The `vfprintf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t vfprintf_m(FILE *file, const string_mx *fmt,
                  int *count, va_list args);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments `args`. `file` shall not be a null pointer.

#### Description

The `vfprintf_m` function formats its parameters `args` into a string according to the format contained in the managed string `fmt` and outputs the result to `file`.

If not a null pointer, `*count` is set to the number of characters transmitted.

#### Returns

The `vfprintf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.8 Functions Derived from `scanf`

These functions are the managed string equivalents to the `scanf`-derived functions in C. Managed string format strings differ from standard C format strings primarily in that they are represented as managed strings. The `%s` specification refers to a managed string rather than a null-terminated byte string or wide character string. The use of `char*` or `wchar_t*` pointers in the `varargs` list will result in a runtime-constraint if detected. The managed string read by `%s` is created as a null-terminated byte string if the input string is a null-terminated byte string or the input stream has byte orientation; otherwise a wide character string is created. The format specification `%ls` indicates that the managed string should be created as a wide character string. The format specification `%hs` indicates that the managed string should be created as a null-terminated byte string.

Applying a byte input function to a wide-oriented stream or applying a wide character input function to a byte-oriented stream will result in a runtime-constraint error.

#### 3.8.1 The `sscanf_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t sscanf_m(string_m buf, const string_mx * fmt,
                 int *count, ...);
```

##### Runtime-Constraints

`buf` and `fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `fmt`.

##### Description

The `sscanf_m` function processes the managed string `buf` according to the format contained in the managed string `fmt` and stores the results using the arguments after `count`.

If not a null pointer, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

##### Returns

The `sscanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

#### 3.8.2 The `vsscanf_m` Function

##### Synopsis

```
#include <string_m.h>
errno_t vsscanf_m(string_mx *buf,
                  const string_mx * fmt,
                  int *count,
                  va_list args);
```

### Runtime-Constraints

`buf` and `fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments `args`.

### Description

The `vsscanf_m` function processes the managed string `buf` according to the format contained in the managed string `fmt` and stores the results using the arguments in `args`.

If not a null pointer, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

### Returns

The `vsscanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.8.3 The `scanf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t scanf_m( const string_mx * fmt, int *count, ...);
```

### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `count`.

### Description

The `scanf_m` function processes input from standard input according to the format contained in the managed string `fmt` and stores the results using the arguments after `count`.

If not null, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

### Returns

The `scanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.8.4 The `vscanf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t vscanf_m( const string_mx * fmt, int *count,
                 va_list args);
```

### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments `args`.

## Description

The `vsscanf_m` function processes input from standard input according to the format contained in the managed string `fmt` and stores the results using the arguments in `args`.

If not null, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Returns

The `vsscanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.8.5 The `fscanf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t fscanf_m(FILE *file,    const string_mx *fmt,
                  int *count, ...);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `count`. `file` shall not be a null pointer.

#### Description

The `fscanf_m` function processes input from `file` according to the format contained in the managed string `fmt` and stores the results using the arguments after `count`.

If not a null pointer, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

#### Returns

The `fscanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.8.6 The `vfscanf_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t vfscanf_m(FILE *file,    const string_mx *fmt,
                  int *count, va_list args);
```

#### Runtime-Constraints

`fmt` shall reference a valid managed string. `fmt` shall not be a null pointer. The managed string `fmt` shall be a valid format compatible with the arguments after `count`. `file` shall not be a null pointer.



## Description

The `vfscanf_m` function processes input from `file` according to the format contained in the managed string `fmt` and stores the results using the arguments after `count`.

If not a null pointer, `*count` is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Returns

The `vfscanf_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

## 3.9 String Slices

### 3.9.1 The `strslice_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t strslice_m(string_m s1,
                  const string_mx * s2,
                  rsize_t offset, rsize_t len);
```

#### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings. There shall be sufficient memory to store the result.

#### Description

The `strslice_m` function takes up to `len` characters from `s2`, starting at the `offset` character in the string, and stores the result in `s1`. If there are insufficient characters to copy `len` characters, all available characters are copied. If `offset` is greater than the number of characters in `s2`, `s1` is set to the null string. If `offset` is equal to the number of characters in `s2` or `len` is 0, `s1` is set to the empty string.

#### Returns

The `strslice_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.9.2 The `strleft_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t strleft_m(string_mx *s1,
                 const string_mx * s2,
                 rsize_t len);
```

#### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings. There shall be sufficient memory to store the result.

## Description

The `strleft_m` function copies up to `len` characters from the start of the managed string `s2` to the managed string `s1`. If `s2` does not have `len` characters, the entire string is copied. If `s2` is a null string, `s1` is set to the null string.

## Returns

The `strleft_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.9.3 The `strright_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t strright_m(string_mx *s1,
                  const string_mx * s2,
                  rsize_t len);
```

#### Runtime-Constraints

`s1` and `s2` shall reference valid managed strings. There shall be sufficient memory to store the result.

## Description

The `strright_m` function copies up to the last `len` characters from the managed string `s2` to the managed string `s1`. If `s2` does not have `len` characters, the entire string is copied. If `s2` is a null string, `s1` is set to the null string.

## Returns

The `strright_m` function returns 0 if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

### 3.9.4 The `cchar_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t cchar_m(const string_mx * s,
               rsize_t offset,
               char *c);
```

#### Runtime-Constraints

`s` shall reference a valid managed string. `c` shall not be a null pointer. `offset` shall be less than the length of the managed string `s`. The character to be returned in `c` shall be representable as a `char`.

## Description

The `cchar_m` function sets `c` to the `offset` character (the first character having an `offset` of 0) in the managed string `s`.

## Returns

The `cchar_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.

### 3.9.5 The `wchar_m` Function

#### Synopsis

```
#include <string_m.h>
errno_t wchar_m( const string_mx * s,
                 rsize_t offset,
                 wchar_t *wc);
```

#### Runtime-Constraints

`s1` shall reference a valid managed string. `wc` shall not be a null pointer. `offset` shall be less than the length of the managed string `s1`.

#### Description

The `wchar_m` function sets `wc` to the `offset` character (the first character having an `offset` of 0) in the managed string `s`.

#### Returns

The `wchar_m` function returns 0 if no runtime-constraints were violated. Otherwise, a nonzero value is returned.



---

## References

*URLs are valid as of the publication date of this document.*

### **[CERT 2009]**

CERT. *Managed String Library*. <http://www.cert.org/secure-coding/managedstring.html> (2009).

### **[ISO/IEC 1999]**

International Organization for Standardization, International Electrotechnical Commission.  
ISO/IEC 9899:1999, *Programming Languages—C*. <http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1124.pdf> (May 6, 2005).



<b>REPORT DOCUMENTATION PAGE</b>			<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. <b>AGENCY USE ONLY</b> (Leave Blank)	2. <b>REPORT DATE</b> May 2010	3. <b>REPORT TYPE AND DATES COVERED</b> Final		
4. <b>TITLE AND SUBTITLE</b> Specifications for Managed Strings, Second Edition		5. <b>FUNDING NUMBERS</b> FA8721-05-C-0003		
6. <b>AUTHOR(S)</b> Hal Burch, Fred Long, Raunak Rungta, Robert Seacord, David Svoboda				
7. <b>PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. <b>PERFORMING ORGANIZATION REPORT NUMBER</b> CMU/SEI-2010-TR-018	
9. <b>SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116			10. <b>SPONSORING/MONITORING AGENCY REPORT NUMBER</b> ESC-TR-2010-018	
11. <b>SUPPLEMENTARY NOTES</b>				
12A <b>DISTRIBUTION/AVAILABILITY STATEMENT</b> Unclassified/Unlimited, DTIC, NTIS			12B <b>DISTRIBUTION CODE</b>	
13. <b>ABSTRACT (MAXIMUM 200 WORDS)</b> This report describes a managed string library for the C programming language. Many software vulnerabilities in C programs result from the misuse of manipulation functions for standard C strings. Programming errors common to string-manipulation logic include buffer overflow, truncation errors, string termination errors, and improper data sanitization. The managed string library provides mechanisms to eliminate or mitigate these problems and improve system security. The CERT® Program, which is part of the Carnegie Mellon® Software Engineering Institute, provides a proof-of-concept implementation of the managed string library on its Secure Coding web pages.				
14. <b>SUBJECT TERMS</b> string library, software security, C programming, runtime-constraint handling			15. <b>NUMBER OF PAGES</b> 46	
16. <b>PRICE CODE</b>				
17. <b>SECURITY CLASSIFICATION OF REPORT</b> Unclassified	18. <b>SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	19. <b>SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	20. <b>LIMITATION OF ABSTRACT</b> UL	