Editors Notes

There have been substantial revisions to this document since the Mt. Tremblant meeting. The most significant of these changes are:

1. Support for "literal" functions that will operate on a C-style or wide character literal string instead of requiring a managed string to be created. These functions have been added for usability of the API.
2. The managed string library now provides equivalent support for wide character strings and multibyte character strings (for each function that accepts a char * there is a corresponding function that accepts wchar_t *). The internal representation of the string is not specified. Extracting a string in a different format from the one from which the string was created implies an implicit conversion.
3. Modified support for valid characters subsets to be on a per string basis.
4. Added for system defined and per string maximum length support. Exceeding these length restrictions now results in a runtime-constraint violation.
5. Additional functions have been added to provide managed versions of a larger subset of C99 string functions.

ISO/IEC JTC1 SC22 WG14 N1158

Date: 2005-08-19

Reference number of document: **ISO/IEC WDTR N1158**

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

Hal Burch
CERT/CC
Carnegie Mellon University

Table of Contents

Introduction

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 C 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.

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.
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. To mitigate against this issue, the managed string library supports an implementation defined maximum string length. Additionally, the string creation function allows for the specification of a per string maximum length.

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 runtime-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.

1 Scope

This Technical Report specifies a series of extensions of the programming language C, specified by International Standard ISO/IEC 9899:1999.

International Standard ISO/IEC 9899:1999 provides important context and specification for this Technical Report. Clauses 3 and 4 of this Technical Report should be read as if they were merged into Clauses 3 and 4 of ISO/IEC 9899:1999. Clause 5 of this Technical Report should be read as if it were merged into Subclause 6.10.8 of ISO/IEC 9899:1999. Clause 6 of this Technical Report should be read as if it were merged into the parallel structure of named Subclauses of Clause 7 of ISO/IEC 9899:1999. Statements made in ISO/IEC 9899:1999, whether about the language or library, apply to this technical report unless a corresponding section of this technical report states otherwise. In particular, Subclause 7.1.4 ("Use of library functions") of ISO/IEC 9899:1999 applies to this technical report.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this Technical Report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9899:1999, *Information technology — Programming languages, their environments and system software interfaces — Programming Language C.*

ISO/IEC 9899:1999/Cor 1:2001, *Information technology — Programming languages, their environments and system software interfaces — Programming Language C — Technical Corrigendum 1 .*

ISO/IEC 9899:1999/Cor 2:2004, *Information technology — Programming languages, their environments and system software interfaces — Programming Language C — Technical Corrigendum 2 .*

ISO 31−11:1992, *Quantities and units — Part 11: Mathematical signs and symbols for use in the physical sciences and technology*.

ISO/IEC 646, *Information technology — ISO 7-bit coded character set for information interchange*.

ISO/IEC 2382−1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*.

ISO 4217, *Codes for the representation of currencies and funds*.

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*.

ISO/IEC 10646 (all parts), *Information technology — Universal Multiple-Octet Coded Character Set (UCS).*

1IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems* (previously designated IEC 559:1989).

3 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.

3.1 Runtime-constraint

requirement on a program when calling a library function

NOTE 1 Despite the similar terms, a runtime-constraint is not a kind of constraint.

NOTE 2 Implementations shall verify that the runtime-constraints for a library function are not violated by the program.

4 Conformance

If a ‘‘shall’’ or ‘‘shall not’’ requirement that appears outside of a constraint or runtime constraint is violated, the behavior is undefined.

5 Predefined macro names

The following macro name is conditionally defined by the implementation:

`__STDC_LIB_EXT2__` The integer constant **`200603L`**, intended to indicate conformance to this technical report.[1]

6 Library

6.1 Use of errno

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

6.2 Runtime-Constraint Violations

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

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 **`<stdlib.h>`**). 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 calling the handler and after the handler returns.

The runtime-constraint handler might not return. If the handler does return, 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 International Standard ISO/IEC 9899:1999, undefined behavior still exists in this technical report. 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.

[1] The intention is that this will remain an integer constant of type long int that is increased with each revision of this technical report.

6.3 Errors `<errno.h>`

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

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

6.4 Managed String Type `<string_m.h>`

The header **`<string_m.h>`** defines a type an abstract data type:

```
typedef void *string_m;
```

The structure referenced by this 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. Functions that have parameters of type `string_m` consider it a runtime-constraint violation if the maximum length of a managed string is exceeded.

Managed strings may also have a defined set of valid characters that can be used in the string. Functions that have parameters of type `string_m` consider it a runtime-constraint violation if a managed string would contain invalid characters.

In addition, functions consider it a runtime-constraint violation if the request would require allocating more memory than available.

7 Library functions

7.1 Utility functions

7.1.1 The `isnull_m` function

Synopsis

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

Runtime-constraints
`s` shall not be a null pointer. **`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 zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.1.2 The `isempty_m` function

Synopsis
```
     #include <string_m.h>
     errno_t isempty_m(const string_m 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 zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.1.3 Creating a string_m

7.1.3.1 The `strcreate_m` function

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

Runtime-constraints
`s` shall not be a null pointer.

Description
The `strcreate_m` function creates a managed string, referenced by `s`, given a conventional string `cstr` (which may be null or empty). `maxlen` specifies the maximum length of the string in characters. If `maxlen` is 0 the system defined maximum length is used. `charset` restricts the set of allowable characters to be those in the c-style string `cstr` (which may be empty). If `charset` is NULL no restricted character set is defined.

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

7.1.3.2 The `wstrcreate_m` function

Synopsis
```
     #include <string_m.h>
     errno_t wstrcreate_m(string_m *s, const wchar_t
               *wcstr,
              const size_t maxlen,
              const wchar_t *charset);
```

Runtime-constraints
`s` shall not be a null pointer.

Description
The `wstrcreate_m` function creates a managed string, referenced by `s`, given a wide character string `wcstr` (which may be null or empty). `maxlen` specifies the maximum length of the string in characters. If `maxlen` is 0 the system defined maximum length is used. `charset` restricts the set of allowable characters to be those in the wide character string `wcstr` (which may be empty). If `charset` is NULL, no restricted character set is defined.

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

7.1.4 The `strdelete_m` function

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

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

7.1.5 The `strlen_m` function

Synopsis
```
     #include <string_m.h>
     errno_t strlen_m(const string_m s, size_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 managed string **s** and stores the result into the variable referenced by **size**.

Returns
The **strlen_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.1.6 Extracting a conventional string

7.1.6.1 The cgetstr_m function

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

Runtime-constraints
s shall reference a valid managed string. **string** shall not be a null pointer.

Description
The **cgetstr_m** function delivers a conventional string into the variable referenced by **string**, given the managed string **s**. The caller is responsible for freeing ***string.**

Returns
The **cgetstr_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned. If there is a runtime-constraint violation, ***string** is set to null.

7.1.6.2 The wgetstr_m function

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

Runtime-constraints
s shall reference a valid managed string. **wcstr** shall not be a null pointer.

Description
The **wgetstr_m** function delivers a wide character string into the variable referenced by **wcstr**, given the managed string **s**. The caller is responsible for freeing ***wcstr.**

Returns
The **wgetstr_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned. If there is a runtime-constraint violation ***wcstr** is set to null.

7.1.7 The `strdup_m` function

Synopsis

```
     #include <string_m.h>
     errno_t strdup_m(string_m *s1, const string_m 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` in the managed string `s1`. The duplicate shall have the same set of valid characters and maximum length.

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

7.2 Copying functions

7.2.1 Unbounded string copy

7.2.1.1 The `strcpy_m` function

Synopsis

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

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

7.2.1.2 The `cstrcpy_m` function

Synopsis

```
     #include <string_m.h>
     errno_t cstrcpy_m(string_m 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 zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.2.1.3 The `wstrcpy_m` function

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

Runtime-constraints
`s1` shall reference a valid managed strings.

Description
The `wstrcpy_m` function copies the string `wcstr` into the managed string `s1`.

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

7.2.2 The `strncpy_m` function

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

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 zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.3 Concatenation functions

7.3.1 Unbounded concatenation

7.3.1.1 The `strcat_m` function

Synopsis

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

Runtime-constraints

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

Description

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

Returns

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

7.3.1.2 The `cstrcat_m` function

Synopsis

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

Runtime-constraints

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

Description

The c`strcat_m` function concatenates the c-style string `cstr` onto the managed string `s`.

Returns

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

7.3.1.3 The `wstrcat_m` function

Synopsis

```
#include <string_m.h>
errno_t wstrcat_m(string_m 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 managed string `s`.

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

7.3.2 Bounded concatenation

7.3.2.1 The `strncat_m` function

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

Runtime-constraints
`s1` and `s2` shall reference valid managed strings. `s2` shall not be null.

Description
The `strncat_m` function appends not more than `n` characters from the managed string `s2` to the end of the managed string `s1`.

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

7.3.2.2 The `cstrncat_m` function

Synopsis
```
     #include <string_m.h>
     errno_t strncat_m (string_m s1,
                const char *cstr,
                size_t n);
```

Runtime-constraints
`s` shall reference a valid managed string. `cstr` shall not be a null pointer.

Description
The `cstrncat_m` function appends not more than `n` characters from C-style string `cstr` to the end of the managed string `s`.

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

7.3.2.3 The `wstrncat_m` function

Synopsis

```
     #include <string_m.h>
     errno_t wstrncat_m (string_m s1,
                const wchar_t *wcstr,
                size_t n);
```

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

Description
The `wstrncat_m` function appends not more than `n` characters from C-style string `wcstr` to the end of the managed string `s`.

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

7.4 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.

For the purpose of comparison, a null string is less than any other string.

7.4.1 Unbounded comparison

7.4.1.1 The `strcmp_m` function

Synopsis

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

Runtime-constraints
`s1` and `s2` shall reference valid managed strings. `cmp` shall not be null.

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

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

7.4.1.2 The `cstrcmp_m` function

Synopsis

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

Runtime-constraints
`s1` shall reference valid a managed string. `cmp` shall not be null.

Description
The `cstrcmp_m` function compares the managed string `s1` to the C-style string `cstr` and sets `cmp` to an integer value greater than, equal to, or less than zero, accordingly as `s1`is greater than, equal to, or less than `cstr.`

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

7.4.1.3 The `wstrcmp_m` function

Synopsis

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

Runtime-constraints
`s1` shall reference valid a managed string. `cmp` shall not be null.

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 zero, accordingly as `s1` is greater than, equal to, or less than `wstr.`

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

7.4.2 The `strcoll_m` function

Synopsis

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

Runtime-constraints
`s1` and `s2` shall reference valid managed strings. `cmp` shall not be null.

Description
The `strcoll_m` function compares the managed string `s1` to the managed string `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the current locale, and sets `cmp` to an integer value greater than, equal to, or less than zero, accordingly as `s1` is greater than, equal to, or less than `s2` when both are interpreted as appropriate to the current locale.

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

7.4.3 Bounded string comparison

7.4.3.1 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);
```

Runtime-constraints
`s1` and `s2` shall reference valid managed strings. `cmp` shall not be null.

Description
The `strncmp_m` function compares not more than `n` characters from the managed string `s1` to the managed string `s2` and sets `cmp` to an integer value greater than, equal to, or less than zero, accordingly as `s1`is greater than, equal to, or less than `s2.`

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

7.4.3.2 The `cstrncmp_m` function

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

Runtime-constraints
`s1` shall reference valid managed strings. `cmp` shall not be null.

Description
The `cstrncmp_m` function compares not more than `n` characters (characters that follow a null character are not compared) from the managed string `s1` to the C-style string `cstr` and sets `cmp` to an integer value greater than, equal to, or less than zero, accordingly as `s1`is greater than, equal to, or less than `cstr.`

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

7.4.3.3 The `wstrncmp_m` function

Synopsis
```
     #include <string_m.h>
     errno_t wstrncmp_m (const string_m s1,
                 const wchar_t *cstr, size_t n,
                 int *cmp);
```

Runtime-constraints
`s1` shall reference valid managed strings. `cmp` shall not be null.

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 zero, accordingly as `s1`is greater than, equal to, or less than `wstr.`

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

7.5 Search functions

7.5.1 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);
```

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 zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

7.5.2 The `cstrchr_m` function

Synopsis
```
     #include <string_m.h>
     errno_t cstrchr_m(string_m out, const string_m str,
                char c);
```

Runtime-constraints
`out` and **`str`** shall reference valid managed strings.

Description
The **`cstrchr_m`** function scans the managed string **`str`** for the first occurrence of **`c`**. **`out`** is set to the string containing and following the first occurrence of **`c`**. If **`str`** does not contain **`c`**, **`out`** is set to the null string.

Returns

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

7.5.3 The `wstrchr_m` function

Synopsis
```
     #include <string_m.h>
     errno_t wstrchr_m(string_m out, const string_m str,
                wchar_t wc);
```

Runtime-constraints
`out` and **`str`** shall reference valid managed strings.

Description

The `wstrchr_m` function scans the managed string `str` for the first occurrence of `wc`. `out` is set to the string containing and following the first occurrence of `c`. If `str` does not contain `wc`, `out` is set to the null string.

Returns

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

7.5.4 The `strspn_m` function

Synopsis

```
#include <string_m.h>
errno_t strspn_m(string_m str, string_m accept,
           size_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 number of characters in the maximum initial segment of the managed string `str` which consists entirely of characters from the managed string `accept`. It sets `*len` to this length.

Returns

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

7.5.5 The `strcspn_m` function

Synopsis

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

Runtime-constraints

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

Description

The `strcspn_m` function computes the number of characters in the maximum initial segment of the managed string `str` which consists entirely of characters *not* from the managed string `reject`. It sets `*len` to this length.

Returns

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

7.6 Configuration functions

7.6.1 The `setcharset_m` function

Synopsis

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

Runtime-constraints
`s` and `charset` shall reference valid managed strings.

Description
The `setcharset` function sets the subset of allowable characters to be those in the managed string `s` (which may be empty). If `mcharset` is set to null, a restricted subset of valid characters is not enforced.

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

7.6.2 The `setmaxlen_m` function

Synopsis

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

Runtime-constraints
`s` shall reference a valid managed string.

Description
The `setcharset_m` function sets the maximum length of the managed string `s`. If `maxlen` is 0 the system defined maximum length is used.

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

7.7 `printf`-derived functions

These functions are the managed string equivalents to the printf-derived functions in C. Managed string format strings differ from standard C format strings primarily in that they are represented as managed strings. In addition, the ‘%s’ specification refers to a managed string, rather than a C-style or wide character string and the `%n’ specification is not recognized.

7.7.1 The `sprintf_m` function

Synopsis

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

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 second parameter into a string according to the format contained in the managed string **`fmt`** and stores the result in the managed string **`buf`**.

Returns

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

7.7.2 The `vsprintf_m` function

Synopsis

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

Runtime-constraints

`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be null. 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`**.

Returns

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

7.7.3 The `snprintf_m` function

Synopsis

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

Runtime-constraints
`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be null. The managed string **`fmt`** shall by a valid format compatible with the arguments after **`fmt`**.

Description
The **`sprintf_m`** function formats its parameters after the second parameter into a string according to the format contained in the managed string **`fmt`** and stores the result in the managed string **`buf`**. If the resulting string contains more than **`max`** characters, it is truncated.

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

7.7.4 The `vsnprintf_m` function

Synopsis
```
#include <string_m.h>
errno_t vsnprintf_m(string_m buf, int max,
          const string_m fmt,
          va_list args);
```

Runtime-constraints
`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be null. The managed string **`fmt`** shall by 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 the resulting string contains more than **`max`** characters, it is truncated.

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

7.7.5 The `printf_m` function

Synopsis
```
#include <string_m.h>
errno_t printf_m(const string_m fmt, ...);
```

Runtime-constraints
`fmt` shall reference a valid managed string. **`fmt`** shall not be null. The managed string **`fmt`** shall by 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.

Returns

The **printf_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.7.6 The `vprintf_m` function

Synopsis

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

Runtime-constraints

fmt shall reference a valid managed string. **fmt** shall not be null. The managed string **fmt** shall by 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

Returns

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

7.7.7 The `fprintf_m` function

Synopsis

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

Runtime-constraints

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

Description

The **fprintf_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 **file**.

Returns

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

7.7.8 The `vfprintf_m` function

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

Runtime-constraints
`fmt` shall reference a valid managed string. **`fmt`** shall not be null. The managed string **`fmt`** shall by 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`**.

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

7.8 `scanf`-derived functions

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 and the '%s' specification refers to a managed string, rather than a C-style or wide character string.

7.8.1 The `sscanf_m` function

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

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

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

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

7.8.2 The `vsscanf_m` function

Synopsis

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

Runtime-constraints

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

Description

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

Returns

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

7.8.3 The `scanf_m` function

Synopsis

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

Runtime-constraints

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

Description

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

Returns

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

7.8.4 The `vscanf_m` function

Synopsis

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

Runtime-constraints

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

Description
The **`vscanf_m`** function process input from standard input according to the format contained in the managed string **`fmt`** and stores the results using the arguments in **`args`**.

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

7.8.5 The `fscanf_m` function

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

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

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

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

7.8.6 The `vfprintf_m` function

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

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

Description
The **`vfprintf_m`** function process input from **`file`** according to the format contained in the managed string **`fmt`** and stores the results using the arguments in **`args`**.

Returns

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

7.9 String slices

7.9.1 The `strslice_m` function

Synopsis

```
#include <string_m.h>
errno_t strslice_m(string_m s1,
           const string_m s2,
           size_t offset, size_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 zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

7.9.2 The `strleft_m` function

Synopsis

```
#include <string_m.h>
errno_t strleft_m(string_m s1,
           const string_m s2,
           size_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 null, **s1** is set to the null string.

Returns

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

7.9.3 The `strright_m` function

Synopsis

```
     #include <string_m.h>
     errno_t strleft_m(string_m s1,
                const string_m s2,
                size_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 null, **`s1`** is set to the null string.

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

7.9.4 The `strchar_m` function

Synopsis

```
     #include <string_m.h>
     errno_t strchar_m(const string_m s,
                size_t offset,
                char *c);
```

Runtime-constraints
`s1` shall reference a valid managed string. **`c`** shall not be null. **`offset`** shall be less than the length of the managed string **`s1`**. The character to be returned in **`c`** shall be representable as a **`char`**.

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

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

7.9.5 The wstrchar_m function

Synopsis

```
     #include <string_m.h>
     errno_t wstrchar_m(const string_m s,
                size_t offset,
                wchar_t *wc);
```

Runtime-constraints

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

Description

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

Returns

The **wstrchar_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.