

String to numeric value functions

Article • 10/19/2022 • 3 minutes to read

- [strtod, _strtod_l, wcstod, _wcstod_l](#)
- [strtol, wcstol, _strtol_l, _wcstol_l](#)
- [strtoul, _strtoul_l, wcstoul, _wcstoul_l](#)
- [_strtoi64, _wcstoi64, _strtoi64_l, _wcstoi64_l](#)
- [_strtoui64, _wcstoui64, _strtoui64_l, _wcstoui64_l](#)

Remarks

Each function in the **strto*** family converts a null-terminated string to a numeric value. The available functions are listed in the following table.

Function	Description
<code>strtod</code>	Convert string to double-precision floating point value
<code>strtol</code>	Convert string to long integer
<code>strtoul</code>	Convert string to unsigned long integer
<code>_strtoi64</code>	Convert string to 64-bit <code>__int64</code> integer
<code>_strtoui64</code>	Convert string to unsigned 64-bit <code>__int64</code> integer

`wcstod`, `wcstol`, `wcstoul`, and `_wcstoi64` are wide-character versions of `strtod`, `strtol`, `strtoul`, and `_strtoi64`, respectively. The string argument to each of these wide-character functions is a wide-character string; each function behaves identically to its single-byte-character counterpart otherwise.

The `strtod` function takes two arguments: the first is the input string, and the second a pointer to the character that ends the conversion process. `strtol`, `strtoul`, `_strtoi64` and `_strtoui64` take a third argument as the number base to use in the conversion process.

The input string is a sequence of characters that can be interpreted as a numerical value of the specified type. Each function stops reading the string at the first character it can't

recognize as part of a number. This character may be the terminating null character. For `strtol`, `strtoul`, `_strtoi64`, and `_strtoui64`, this terminating character can also be the first numeric character greater than or equal to the user-supplied number base.

If the user-supplied pointer to an end-of-conversion character isn't set to `NULL` or `nullptr` at call time, a pointer to the character that stopped the scan will be stored there instead. If no conversion can be performed (no valid digits were found or an invalid base was specified), the value of the string pointer is stored at that address.

`strtod` expects a string of the following form:

`[whitespace] [sign] [digits] [. digits] [{d | D | e | E}] [sign] digits]`

A *whitespace* may consist of space or tab characters, which are ignored; *sign* is either plus (+) or minus (-); and *digits* are one or more decimal digits. If no digits appear before the radix character, at least one must appear after the radix character. The decimal digits can be followed by an exponent, which consists of an introductory letter (*d*, *D*, *e*, or *E*) and an optionally signed integer. If no exponent part or radix character appears, a radix character is assumed to follow the last digit in the string. The first character that doesn't fit this form stops the scan.

The `strtol`, `strtoul`, `_strtoi64`, and `_strtoui64` functions expect a string of the following form:

`[whitespace] [{+ | -}] [0 [{ x | X }]] [digits]`

If the base argument is between 2 and 36, then it's used as the base of the number. If it's 0, the initial characters referenced to by the end-of-conversion pointer are used to determine the base. If the first character is 0 and the second character isn't 'x' or 'X', the string is interpreted as an octal integer; otherwise, it's interpreted as a decimal number. If the first character is '0' and the second character is 'x' or 'X', the string is interpreted as a hexadecimal integer. If the first character is '1' through '9', the string is interpreted as a decimal integer. The letters 'a' through 'z' (or 'A' through 'Z') are assigned the values 10 through 35; only letters whose assigned values are less than *base* are permitted. `strtoul` and `_strtoui64` allow a plus (+) or minus (-) sign prefix; a leading minus sign indicates that the return value is negated.

The output value is affected by the setting of the `LC_NUMERIC` category setting of the locale. For more information, see [setlocale](#). The versions of these functions without the `_1` suffix

use the current locale for this locale-dependent behavior; the versions with the `_l` suffix are identical except that they use the locale parameter passed in instead.

When the value returned by these functions would cause an overflow or underflow, or when conversion isn't possible, special case values are returned as shown:

Function	Condition	Value returned
<code>strtod</code>	Overflow	<code>+/- HUGE_VAL</code>
<code>strtod</code>	Underflow or no conversion	<code>0</code>
<code>strtol</code>	<code>+ Overflow</code>	<code>LONG_MAX</code>
<code>strtol</code>	<code>- Overflow</code>	<code>LONG_MIN</code>
<code>strtol</code>	Underflow or no conversion	<code>0</code>
<code>_strtoi64</code>	<code>+ Overflow</code>	<code>_I64_MAX</code>
<code>_strtoi64</code>	<code>- Overflow</code>	<code>_I64_MIN</code>
<code>_strtoi64</code>	No conversion	<code>0</code>
<code>_strtoui64</code>	Overflow	<code>_UI64_MAX</code>
<code>_strtoui64</code>	No conversion	<code>0</code>

`_I64_MAX`, `_I64_MIN`, and `_UI64_MAX` are defined in `<LIMITS.H>`.

`wcstod`, `wcstol`, `wcstoul`, `_wcstoi64`, and `_wcstoui64` are wide-character versions of `strtod`, `strtol`, `strtoul`, `_strtoi64`, and `_strtoui64`, respectively; the pointer to an end-of-conversion argument to each of these wide-character functions is a wide-character string. Otherwise, each of these wide-character functions behaves identically to its single-byte-character counterpart.

See also

- [Data conversion](#)
- [Locale](#)
- [Interpretation of multibyte-character sequences](#)
- [Math and floating-point support](#)
- [`atof`, `\_atof\_l`, `\_wtof`, `\_wtof\_l`](#)