String

Built-in string class.

Description

This is the built-in string class (and the one used by GDScript). It supports Unicode and provides all necessary means for string handling. Strings are reference-counted and use a copy-on-write approach, so passing them around is cheap in resources.

Tutorials

- GDScript format strings

Methods

String	String **(** bool from **)**
String	String **(** int from **)**
String	String **(** float from **)**
String	String **(** Vector2 from **)**
String	String **(** Rect2 from **)**
String	String **(** Vector3 from **)**
String	String **(** Transform2D from **)**
String	String **(** Plane from **)**
String	String **(** Quat from **)**
String	String **(** AABB from **)**
String	String **(** Basis from **)**
String	String **(** Transform from **)**
String	String **(** Color from **)**
String	String **(** NodePath from **)**

String	String **(** RID from **)**
String	String **(** Dictionary from **)**
String	String **(** Array from **)**
String	String **(** PoolByteArray from **)**
String	String **(** PoolIntArray from **)**
String	String **(** PoolRealArray from **)**
String	String **(** PoolStringArray from **)**
String	String **(** PoolVector2Array from **)**
String	String **(** PoolVector3Array from **)**
String	String **(** PoolColorArray from **)**
bool	begins_with **(** String text **)**
PoolStringArray	bigrams **( )**
String	c_escape **( )**
String	c_unescape **( )**
String	capitalize **( )**
int	casecmp_to **(** String to **)**
int	count **(** String what, int from=0, int to=0 **)**
int	countn **(** String what, int from=0, int to=0 **)**
String	dedent **( )**
bool	empty **( )**
bool	ends_with **(** String text **)**
void	erase **(** int position, int chars **)**
int	find **(** String what, int from=0 **)**
int	find_last **(** String what **)**
int	findn **(** String what, int from=0 **)**
String	format **(** Variant values, String placeholder="{_}" **)**
String	get_base_dir **( )**
String	get_basename **( )**
String	get_extension **( )**
String	get_file **( )**

int	hash ( )
int	hex_to_int ( )
String	http_escape ( )
String	http_unescape ( )
String	humanize_size (int size)
String	indent (String prefix)
String	insert (int position, String what)
bool	is_abs_path ( )
bool	is_rel_path ( )
bool	is_subsequence_of (String text)
bool	is_subsequence_ofi (String text)
bool	is_valid_filename ( )
bool	is_valid_float ( )
bool	is_valid_hex_number (bool with_prefix=false)
bool	is_valid_html_color ( )
bool	is_valid_identifier ( )
bool	is_valid_integer ( )
bool	is_valid_ip_address ( )
String	json_escape ( )
String	left (int position)
int	length ( )
String	lstrip (String chars)
bool	match (String expr)
bool	matchn (String expr)
PoolByteArray	md5_buffer ( )
String	md5_text ( )
int	naturalnocasecmp_to (String to)
int	nocasecmp_to (String to)
int	ord_at (int at)
String	pad_decimals (int digits)

String	pad_zeros (int digits)
String	percent_decode ( )
String	percent_encode ( )
String	plus_file (String file)
String	repeat (int count)
String	replace (String what, String forwhat)
String	replacen (String what, String forwhat)
int	rfind (String what, int from=-1)
int	rfindn (String what, int from=-1)
String	right (int position)
PoolStringArray	rsplit (String delimiter, bool allow_empty=true, int maxsplit=0)
String	rstrip (String chars)
PoolByteArray	sha1_buffer ( )
String	sha1_text ( )
PoolByteArray	sha256_buffer ( )
String	sha256_text ( )
float	similarity (String text)
String	simplify_path ( )
PoolStringArray	split (String delimiter, bool allow_empty=true, int maxsplit=0)
PoolRealArray	split_floats (String delimiter, bool allow_empty=true)
String	strip_edges (bool left=true, bool right=true)
String	strip_escapes ( )
String	substr (int from, int len=-1)
PoolByteArray	to_ascii ( )
float	to_float ( )
int	to_int ( )
String	to_lower ( )
String	to_upper ( )
PoolByteArray	to_utf8 ( )
PoolByteArray	to_wchar ( )

String	trim_prefix **(** String prefix **)**
String	trim_suffix **(** String suffix **)**
String	validate_node_name **( )**
String	xml_escape **( )**
String	xml_unescape **( )**

Method Descriptions

- String **String (** bool from **)**

Constructs a new String from the given bool.

- String **String (** int from **)**

Constructs a new String from the given int.

- String **String (** float from **)**

Constructs a new String from the given float.

- String **String (** Vector2 from **)**

Constructs a new String from the given Vector2.

- String **String (** Rect2 from **)**

Constructs a new String from the given Rect2.

- String **String (** Vector3 from **)**

Constructs a new String from the given Vector3.

- String **String (** Transform2D from **)**

Constructs a new String from the given Transform2D.

- String **String (** Plane from **)**

Constructs a new String from the given Plane.

- String **String (** Quat from **)**

Constructs a new String from the given Quat.

- String **String (** AABB from **)**

Constructs a new String from the given AABB.

- String **String (** Basis from **)**

Constructs a new String from the given Basis.

- String **String (** Transform from **)**

Constructs a new String from the given Transform.

- String **String (** Color from **)**

Constructs a new String from the given Color.

- String **String (** NodePath from **)**

Constructs a new String from the given NodePath.

- String **String (** RID from **)**

Constructs a new String from the given RID.

- String **String (** Dictionary from **)**

Constructs a new String from the given Dictionary.

- String **String (** Array from **)**

Constructs a new String from the given Array.

- String **String (** PoolByteArray from **)**

Constructs a new String from the given PoolByteArray.

- String **String (** PoolIntArray from **)**

Constructs a new String from the given PoolIntArray.

- String **String (** PoolRealArray from **)**

Constructs a new String from the given PoolRealArray.

- String **String (** PoolStringArray from **)**

Constructs a new String from the given PoolStringArray.

- String **String (** PoolVector2Array from **)**

Constructs a new String from the given PoolVector2Array.

- String **String (** PoolVector3Array from **)**

Constructs a new String from the given PoolVector3Array.

- String **String (** PoolColorArray from **)**

Constructs a new String from the given PoolColorArray.

- bool **begins_with (** String text **)**

Returns `true` if the string begins with the given string.

- PoolStringArray **bigrams ( )**

Returns an array containing the bigrams (pairs of consecutive letters) of this string.

```
print("Bigrams".bigrams()) # Prints "[Bi, ig, gr, ra, am, ms]"
```

- String **c_escape ( )**

Returns a copy of the string with special characters escaped using the C language standard.

- String **c_unescape ( )**

Returns a copy of the string with escaped characters replaced by their meanings. Supported escape sequences are `\'` , `\"` , `\?` , `\\` , `\a` , `\b` , `\f` , `\n` , `\r` , `\t` , `\v` .

Note: Unlike the GDScript parser, this method doesn't support the `\uXXXX` escape sequence.

- String **capitalize ( )**

Changes the case of some letters. Replaces underscores with spaces, adds spaces before in-word uppercase characters, converts all letters to lowercase, then capitalizes the first letter and every letter following a space character. For `capitalize camelCase mixed_with_underscores` , it will return `Capitalize Camel Case Mixed With Underscores` .

- int **casecmp_to (** String to **)**

Performs a case-sensitive comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "less than" or "greater than" are determined by the Unicode code points of each string, which roughly matches the alphabetical order.

Behavior with different string lengths: Returns `1` if the "base" string is longer than the `to` string or `-1` if the "base" string is shorter than the `to` string. Keep in mind this length is determined by the number of Unicode codepoints, *not* the actual visible characters.

Behavior with empty strings: Returns `-1` if the "base" string is empty, `1` if the `to` string is empty or `0` if both strings are empty.

To get a boolean result from a string comparison, use the `==` operator instead. See also nocasecmp_to.

- int **count (** String what, int from=0, int to=0 **)**

Returns the number of occurrences of substring `what` between `from` and `to` positions. If `from` and `to` equals 0 the whole string will be used. If only `to` equals 0 the remained substring will be used.

- int **countn (** String what, int from=0, int to=0 **)**

Returns the number of occurrences of substring `what` (ignoring case) between `from` and `to` positions. If `from` and `to` equals 0 the whole string will be used. If only `to` equals 0 the remained substring will be used.

- String **dedent ( )**

Returns a copy of the string with indentation (leading tabs and spaces) removed. See also indent to add indentation.

- bool **empty ( )**

Returns `true` if the length of the string equals `0`.

- bool **ends_with (** String text **)**

Returns `true` if the string ends with the given string.

- void **erase (** int position, int chars **)**

Erases `chars` characters from the string starting from `position`.

- int **find (** String what, int from=0 **)**

Finds the first occurrence of a substring. Returns the starting position of the substring or `-1` if not found. Optionally, the initial search index can be passed.

Note: If you just want to know whether a string contains a substring, use the `in` operator as follows:

```
# Will evaluate to `false`.
if "i" in "team":
    pass
```

- int **find_last (** String what **)**

Finds the last occurrence of a substring. Returns the starting position of the substring or `-1` if not found.

- int **findn (** String what, int from=0 **)**

Finds the first occurrence of a substring, ignoring case. Returns the starting position of the substring or `-1` if not found. Optionally, the initial search index can be passed.

- String **format (** Variant values, String placeholder="{_}" **)**

Formats the string by replacing all occurrences of `placeholder` with `values`.

- String **get_base_dir ( )**

If the string is a valid file path, returns the base directory name.

- String **get_basename ( )**

If the string is a valid file path, returns the full file path without the extension.

- String **get_extension ( )**

Returns the extension without the leading period character (`.`) if the string is a valid file name or path. If the string does not contain an extension, returns an empty string instead.

```
print("/path/to/file.txt".get_extension())  # "txt"
print("file.txt".get_extension())  # "txt"
print("file.sample.txt".get_extension())  # "txt"
print(".txt".get_extension())  # "txt"
print("file.txt.".get_extension())  # "" (empty string)
print("file.txt..".get_extension())  # "" (empty string)
print("txt".get_extension())  # "" (empty string)
print("".get_extension())  # "" (empty string)
```

- String **get_file ( )**

If the string is a valid file path, returns the filename.

- int **hash ( )**

Returns the 32-bit hash value representing the string's contents.

Note: `String` s with equal content will always produce identical hash values. However, the reverse is not true. Returning identical hash values does *not* imply the strings are equal, because different strings can have identical hash values due to hash collisions.

- int **hex_to_int ( )**

Converts a string containing a hexadecimal number into an integer. Hexadecimal strings are expected to be prefixed with "`0x`" otherwise `0` is returned.

```
print("0xff".hex_to_int()) # Print "255"
```

- String **http_escape ( )**

Escapes (encodes) a string to URL friendly format. Also referred to as 'URL encode'.

```
print("https://example.org/?escaped=" + "Godot Engine:'docs'".http_escape())
```

- String **http_unescape ( )**

Unescapes (decodes) a string in URL encoded format. Also referred to as 'URL decode'.

```
print("https://example.org/?escaped=" +
"Godot%20Engine%3A%27docs%27".http_unescape())
```

- String **humanize_size (** int size **)**

Converts `size` represented as number of bytes to human-readable format using internationalized set of data size units, namely: B, KiB, MiB, GiB, TiB, PiB, EiB. Note that the next smallest unit is picked automatically to hold at most 1024 units.

```
var bytes = 133790307
var size = String.humanize_size(bytes)
print(size) # prints "127.5 MiB"
```

- String **indent (** String prefix **)**

Returns a copy of the string with lines indented with `prefix`.

For example, the string can be indented with two tabs using `"\t\t"`, or four spaces using `"    "`. The prefix can be any string so it can also be used to comment out strings with e.g. `"# "`. See also dedent to remove indentation.

Note: Empty lines are kept empty.

- String **insert (** int position, String what **)**

Returns a copy of the string with the substring `what` inserted at the given position.

- bool **is_abs_path ( )**

If the string is a path to a file or directory, returns `true` if the path is absolute.

- bool **is_rel_path ( )**

If the string is a path to a file or directory, returns `true` if the path is relative.

- bool **is_subsequence_of (** String text **)**

Returns `true` if this string is a subsequence of the given string.

- bool **is_subsequence_ofi (** String text **)**

Returns `true` if this string is a subsequence of the given string, without considering case.

- bool **is_valid_filename ( )**

Returns `true` if this string is free from characters that aren't allowed in file names, those being:

`: / \ ? * " | % < >`

- bool **is_valid_float ( )**

Returns `true` if this string contains a valid float. This is inclusive of integers, and also supports exponents:

```
print("1.7".is_valid_float()) # Prints "True"
print("24".is_valid_float()) # Prints "True"
print("7e3".is_valid_float()) # Prints "True"
print("Hello".is_valid_float()) # Prints "False"
```

- bool **is_valid_hex_number (** bool with_prefix=false **)**

Returns `true` if this string contains a valid hexadecimal number. If `with_prefix` is `true`, then a validity of the hexadecimal number is determined by `0x` prefix, for instance: `0xDEADC0DE`.

- bool **is_valid_html_color ( )**

Returns `true` if this string contains a valid color in hexadecimal HTML notation. Other HTML notations such as named colors or `hsl()` colors aren't considered valid by this method and will return `false`.

- bool **is_valid_identifier ( )**

Returns `true` if this string is a valid identifier. A valid identifier may contain only letters, digits and underscores (`_`) and the first character may not be a digit.

```
print("good_ident_1".is_valid_identifier()) # Prints "True"
print("1st_bad_ident".is_valid_identifier()) # Prints "False"
print("bad_ident_#2".is_valid_identifier()) # Prints "False"
```

- bool **is_valid_integer ( )**

Returns `true` if this string contains a valid integer.

```
print("7".is_valid_int()) # Prints "True"
print("14.6".is_valid_int()) # Prints "False"
print("L".is_valid_int()) # Prints "False"
print("+3".is_valid_int()) # Prints "True"
print("-12".is_valid_int()) # Prints "True"
```

- bool **is_valid_ip_address ( )**

Returns `true` if this string contains only a well-formatted IPv4 or IPv6 address. This method considers reserved IP addresses such as `0.0.0.0` as valid.

- String **json_escape ( )**

Returns a copy of the string with special characters escaped using the JSON standard.

- String **left (** int position **)**

Returns a number of characters from the left of the string.

- int **length ( )**

Returns the string's amount of characters.

- String **lstrip (** String chars **)**

Returns a copy of the string with characters removed from the left. The `chars` argument is a string specifying the set of characters to be removed.

Note: The `chars` is not a prefix. See trim_prefix method that will remove a single prefix string rather than a set of characters.

- bool **match (** String expr **)**

Does a simple case-sensitive expression match, where `"*"` matches zero or more arbitrary characters and `"?"` matches any single character except a period (`"."`). An empty string or empty expression always evaluates to `false`.

- bool **matchn (** String expr **)**

Does a simple case-insensitive expression match, where `"*"` matches zero or more arbitrary characters and `"?"` matches any single character except a period (`"."`). An empty string or empty expression always evaluates to `false`.

- PoolByteArray **md5_buffer ( )**

Returns the MD5 hash of the string as an array of bytes.

- String **md5_text ( )**

Returns the MD5 hash of the string as a string.

- int **naturalnocasecmp_to (** String to **)**

Performs a case-insensitive *natural order* comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "less than" or "greater than" are determined by the Unicode code points of each string, which roughly matches the alphabetical order. Internally, lowercase characters will be converted to uppercase during the comparison.

When used for sorting, natural order comparison will order suites of numbers as expected by most people. If you sort the numbers from 1 to 10 using natural order, you will get `[1, 2, 3, ...]` instead of `[1, 10, 2, 3, ...]`.

Behavior with different string lengths: Returns `1` if the "base" string is longer than the `to` string or `-1` if the "base" string is shorter than the `to` string. Keep in mind this length is determined by the number of Unicode codepoints, *not* the actual visible characters.

Behavior with empty strings: Returns `-1` if the "base" string is empty, `1` if the `to` string is empty or `0` if both strings are empty.

To get a boolean result from a string comparison, use the `==` operator instead. See also nocasecmp_to and casecmp_to.

- int **nocasecmp_to (** String to **)**

Performs a case-insensitive comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "less than" or "greater than" are determined by the Unicode code points of each string, which roughly matches the alphabetical order. Internally, lowercase characters will be converted to uppercase during the comparison.

Behavior with different string lengths: Returns `1` if the "base" string is longer than the `to` string or `-1` if the "base" string is shorter than the `to` string. Keep in mind this length is determined by the number of Unicode codepoints, *not* the actual visible characters.

Behavior with empty strings: Returns `-1` if the "base" string is empty, `1` if the `to` string is empty or `0` if both strings are empty.

To get a boolean result from a string comparison, use the `==` operator instead. See also casecmp_to.

- int **ord_at (** int at **)**

Returns the character code at position `at`.

- String **pad_decimals (** int digits **)**

Formats a number to have an exact number of `digits` after the decimal point.

- String **pad_zeros (** int digits **)**

Formats a number to have an exact number of `digits` before the decimal point.

- String **percent_decode ( )**

Decode a percent-encoded string. See percent_encode.

- String **percent_encode ( )**

Percent-encodes a string. Encodes parameters in a URL when sending a HTTP GET request (and bodies of form-urlencoded POST requests).

- String **plus_file (** String file **)**

If the string is a path, this concatenates `file` at the end of the string as a subpath. E.g. `"this/is".plus_file("path") == "this/is/path"`.

- String **repeat (** int count **)**

Returns original string repeated a number of times. The number of repetitions is given by the argument.

- String **replace (** String what, String forwhat **)**

Replaces occurrences of a case-sensitive substring with the given one inside the string.

- String **replacen (** String what, String forwhat **)**

Replaces occurrences of a case-insensitive substring with the given one inside the string.

- int **rfind (** String what, int from=-1 **)**

Performs a case-sensitive search for a substring, but starts from the end of the string instead of the beginning.

- int **rfindn (** String what, int from=-1 **)**

Performs a case-insensitive search for a substring, but starts from the end of the string instead of the beginning.

- String **right (** int position **)**

Returns the right side of the string from a given position.

- PoolStringArray **rsplit (** String delimiter, bool allow_empty=true, int maxsplit=0 **)**

Splits the string by a `delimiter` string and returns an array of the substrings, starting from right.

The splits in the returned array are sorted in the same order as the original string, from left to right.

If `maxsplit` is specified, it defines the number of splits to do from the right up to `maxsplit`. The default value of 0 means that all items are split, thus giving the same result as split.

Example:

```
var some_string = "One,Two,Three,Four"
var some_array = some_string.rsplit(",", true, 1)
print(some_array.size()) # Prints 2
print(some_array[0]) # Prints "One,Two,Three"
print(some_array[1]) # Prints "Four"
```

- String **rstrip (** String chars **)**

Returns a copy of the string with characters removed from the right. The `chars` argument is a string specifying the set of characters to be removed.

Note: The `chars` is not a suffix. See trim_suffix method that will remove a single suffix string rather than a set of characters.

- PoolByteArray **sha1_buffer ( )**

Returns the SHA-1 hash of the string as an array of bytes.

- String **sha1_text ( )**

Returns the SHA-1 hash of the string as a string.

- PoolByteArray **sha256_buffer ( )**

Returns the SHA-256 hash of the string as an array of bytes.

- String **sha256_text ( )**

Returns the SHA-256 hash of the string as a string.

- float **similarity (** String text **)**

Returns the similarity index (Sorensen-Dice coefficient) of this string compared to another. A result of 1.0 means totally similar, while 0.0 means totally dissimilar.

```
print("ABC123".similarity("ABC123")) # Prints "1"
print("ABC123".similarity("XYZ456")) # Prints "0"
print("ABC123".similarity("123ABC")) # Prints "0.8"
print("ABC123".similarity("abc123")) # Prints "0.4"
```

- String **simplify_path ( )**

Returns a simplified canonical path.

- PoolStringArray **split (** String delimiter, bool allow_empty=true, int maxsplit=0 **)**

Splits the string by a `delimiter` string and returns an array of the substrings. The `delimiter` can be of any length.

If `maxsplit` is specified, it defines the number of splits to do from the left up to `maxsplit`. The default value of `0` means that all items are split.

Example:

```
var some_string = "One,Two,Three,Four"
var some_array = some_string.split(",", true, 1)
print(some_array.size()) # Prints 2
print(some_array[0]) # Prints "One"
print(some_array[1]) # Prints "Two,Three,Four"
```

If you need to split strings with more complex rules, use the RegEx class instead.

- PoolRealArray **split_floats (** String delimiter, bool allow_empty=true **)**

Splits the string in floats by using a delimiter string and returns an array of the substrings.

For example, `"1,2.5,3"` will return `[1,2.5,3]` if split by `","`.

- String **strip_edges (** bool left=true, bool right=true **)**

Returns a copy of the string stripped of any non-printable character (including tabulations, spaces and line breaks) at the beginning and the end. The optional arguments are used to toggle stripping on the left and right edges respectively.

- String **strip_escapes ( )**

Returns a copy of the string stripped of any escape character. These include all non-printable control characters of the first page of the ASCII table (< 32), such as tabulation (`\t` in C) and newline (`\n` and `\r`) characters, but not spaces.

- String **substr (** int from, int len=-1 **)**

Returns part of the string from the position `from` with length `len`. Argument `len` is optional and using `-1` will return remaining characters from given position.

- PoolByteArray **to_ascii ( )**

Converts the String (which is a character array) to PoolByteArray (which is an array of bytes). The conversion is faster compared to to_utf8, as this method assumes that all the characters in the String are ASCII characters.

- float **to_float ( )**

Converts a string containing a decimal number into a `float`. The method will stop on the first non-number character except the first `.` (decimal point), and `e` which is used for exponential.

```
print("12.3".to_float()) # 12.3
print("1.2.3".to_float()) # 1.2
print("12ab3".to_float()) # 12
print("1e3".to_float()) # 1000
```

- int **to_int ( )**

Converts a string containing an integer number into an `int`. The method will remove any non-number character and stop if it encounters a `.`.

```
print("123".to_int()) # 123
print("a1b2c3".to_int()) # 123
print("1.2.3".to_int()) # 1
```

- String **to_lower ( )**

Returns the string converted to lowercase.

- String **to_upper ( )**

Returns the string converted to uppercase.

- PoolByteArray **to_utf8 ( )**

Converts the String (which is an array of characters) to PoolByteArray (which is an array of bytes). The conversion is a bit slower than to_ascii, but supports all UTF-8 characters. Therefore, you should prefer this function over to_ascii.

- PoolByteArray **to_wchar ( )**

Converts the String (which is an array of characters) to PoolByteArray (which is an array of bytes).

- String **trim_prefix (** String prefix **)**

Removes a given string from the start if it starts with it or leaves the string unchanged.

- String **trim_suffix (** String suffix **)**

Removes a given string from the end if it ends with it or leaves the string unchanged.

- String **validate_node_name ( )**

Removes any characters from the string that are prohibited in Node names (`.` `:` `@` `/` `"`).

- String **xml_escape ( )**

Returns a copy of the string with special characters escaped using the XML standard.

- String **xml_unescape ( )**

Returns a copy of the string with escaped characters replaced by their meanings according to the XML standard.