# NVStrings Class

This class manages a list of strings stored in device memory. All methods run operations (in parallel) on all the strings it manages. The methods are modeled after the Pandas string operations and are meant to match directly to the python wrapper class, nvstrings.py, as much as possible. The managed strings are treated as immutable and all operations that would modify strings would create new strings. Even operations that may modify only some strings will create a new string list to be managed by the new instance independently.

## Instantiation

These constructors are private in order to control the memory allocation. The following methods will create new NVStrings given a list of character arrays. These methods will make a copy of strings in host/device memory so sufficient storage must be available for this to succeed.

```
static NVStrings* create_from_array(const char** hstrs, unsigned int
count);
static NVStrings* create_from_index(std::pair<const char*,size_t>* dstrs,
unsigned int count, bool devmem=true );
```

Parameters are host strings that are copied into device memory.

| Parameter | Description |
|---|---|
| hstrs | Array of null-terminated strings formatted as UTF-8 in host memory. |
| dstrs | Array of pairs each pointing to a string in device memory along with its byte length. |
| count | Number of strings in the array |
| devmem | True if the dstrs array itself is in device memory.<br>The dstrs can be in host memory but the pair values must point to device memory. |

```
std::vector<std::string> strings;  // bunch of strings in CPU memory
...
std::vector<const char*> ptrs(strings.size());
for( int i = 0; i < (int)strings.size(); i++ )
   ptrs[i] = strings[i].c_str();

NVStrings* d_strings = NVStrings::create_from_array(ptrs.data(), ptrs.
size());
...
```

You must use the destroy() method free any NVStrings instance created by methods in this class.

```
static void destroy(NVStrings* inst);
```

```
// free instance created in example above
NVStrings::destroy(d_strings);
```

## Attributes

These methods perform no operations but return data about the instance.

This method returns the number of strings managed by this instance.

```
unsigned int size();
```

This method sets the number of characters of each string into the given array. The array memory must be allocated by the caller. The return value is the number strings.

```
unsigned int len( int* lengths, bool devmem=true );
```

| Parameter | Description |
|---|---|
| lengths | Memory allocated by the caller to hold the lengths for each string. The memory size must be array of size() * sizeof(int). On return: a value of -1 indicates a null entry; an empty string will have a value of 0. |
| devmem | True if the lengths array is allocated in device memory. |

Example:

```
NVStrings* strs = ... // created with ["hello","","world"]
unsigned int count = strs->size();  // number of strings
int* lengths = new int[count];      // allocate memory for each length
strs->len(rtn,false);               // get the lengths
// print out the lengths
for(int idx=0; idx < (int)count; idx++)
        printf("%d length = %d\n", idx, lengths[idx]);
delete rtn; // done with this memory

output would be:
0 length = 5
1 length = 0
2 length = 5
```

These methods return True for strings where all the characters match the corresponding type.

```
unsigned int isalnum( bool* results, bool todevice=true );
unsigned int isalpha( bool* results, bool todevice=true );
unsigned int isdigit( bool* results, bool todevice=true );
unsigned int isspace( bool* results, bool todevice=true );
unsigned int isdecimal( bool* results, bool todevice=true );
unsigned int isnumeric( bool* results, bool todevice=true );
unsigned int islower( bool* results, bool todevice=true );
unsigned int isupper( bool* results, bool todevice=true );
```

| Parameter | Description |
|---|---|
| results | Memory allocated by the caller to hold the results for each string. |
| todevice | True if the results array is allocated in device memory. |

## Combining

These methods combine the strings with the argument strings and separators.

This method appends the given strings to this list of strings and returns as new strings.

```
NVStrings* cat( NVStrings* others, const char* separator, const char*
narep=0);
```

| Parameter | Description |
|---|---|
| others | Strings which are to be appended. The number of strings must match. |
| separator | If specified, this separator will be appended to each string before appending the others. |
| narep | This character will take the place of any null strings (not empty strings) in either list.<br>The default value is null – if either string is null, the result is null at the position. |

Example:

```
NVStrings* strs1 = ... // created with ["hello",null,"goodbye"]
NVStrings* strs2 = ... // created with ["world","globe",null]

NVStrings* strs = strs1->cat(strs2,":","_");

strs would now contain:
["hello:world","_:globe","goodbye:_"]
```

Concatenates all strings into one new string.

```
NVStrings* join( const char* separator, const char* narep=0 );
```

Appends each string with itself the specified number of times.

```
NVStrings* repeat(unsigned int count);
```

| Parameter | Description |
|---|---|
| count | The number of times each string should be repeated. Repeat count of 0 or 1 will just return copy of each string. |

## Splitting

Each string is split into one or more strings using the delimiter provided. These methods will return arrays of new NVStrings for each string.

```
int split( const char* delimiter, int maxsplit, std::vector<NVStrings*>&
results);
int rsplit( const char* delimiter, int maxsplit, std::vector<NVStrings*>&
results);
```

| Parameter | Description |
|---|---|
| delimiter | The character used to locate the split points of each string. |
| maxsplit | Maximum number of strings to return for each split. |
| results | List of new strings created by each split. |

Each string is split into two strings on the first delimiter found (rpartition searches from the end of the string). Three strings are returned for each string: beginning, delimiter, end.

```
int partition( const char* delimiter, std::vector<NVStrings*>& results);
int rpartition( const char* delimiter, std::vector<NVStrings*>& results);
```

| Parameter | Description |
|---|---|
| delimiter | The character used to locate the split point. |
| results | List of new strings created by each split. |

This method will create new NVStrings instances by splitting each string into new columns.

```
unsigned int split_column( const char* delimiter, int maxsplit, std::
vector<NVStrings*>& results);
unsigned int rsplit_column( const char* delimiter, int maxsplit, std::
vector<NVStrings*>& results);
```

A new set of columns (NVStrings) is created by splitting the strings vertically.

| Parameter | Description |
|---|---|
| delimiter | The character used to locate the split point. Default is space. |
| n | Maximum number of columns. Default (-1) is all tokens are split on the delimiter. |

Return value is the number of columns.

## Padding

Adding characters to the beginning or the end of strings to fill the width of characters specified.

```
NVStrings* pad( unsigned int width, padside side, const char* fillchar=0 );
NVStrings* ljust( unsigned int width, const char* fillchar=0 );
NVStrings* center( unsigned int width, const char* fillchar=0 );
NVStrings* rjust( unsigned int width, const char* fillchar=0 );
```

| Parameter | Description |
|---|---|
| width | The minimum width of characters of the new string. If the width is smaller than the existing string, no padding is performed. |
| fillchar | The character used to do the padding. Default is space character. Only the first character is used and the char array may contain UTF-8 encoded bytes. |
| side | Either one of "left", "right", "both". The default is "left"<br><br>"left" performs a padding on the left – same as rjust()<br>"right" performs a padding on the right – same as ljust()<br>"both" performs equal padding on left and right – sames as center() |

Pads the strings with leading zeros. It will handle prefix sign characters correctly for strings that are numbers.

```
NVStrings* zfill( unsigned int width );
```

| Parameter | Description |
|---|---|
| width | The minimum width of characters of the new string. If the width is smaller than the existing string, no padding is performed. |

Inserts new-line character into white-space in each string so new-line delimited sections in the string are not longer than the width of characters specified.

```
NVStrings* wrap( unsigned int width );
```

| Parameter | Description |
|---|---|
| width | The minimum width of characters of the lines in the string.<br>If an appropriate white-space is not found for a line, that line is neither truncated or split. |

## Stripping

Generally, these are used to remove whitespace from the beginning and the ends of each strings.

```
NVStrings* lstrip( const char* to_strip );
NVStrings* strip( const char* to_strip );
NVStrings* rstrip( const char* to_strip );
```

| Parameters | Description |
|---|---|
| to_strip | The character to strip from the beginning (lstrip), end (rstrip) or both (strip) |

## Substrings

Creating strings from sections of existing strings. The return is a new instance containing the strings between the specified positions.

```
NVStrings* slice( int start=0, int stop=-1, int step=1 );
```

| Parameter | Description |
|---|---|
| start | Beginning position of the string to extract. Default is 0 = beginning of the each string. |
| stop | Ending position of the string to extract. Default is -1 = end of each string. |
| step | Characters that are to be captured within the specified section. Default is 1 = every character. |

Extract a section of each string using individual character position ranges. The parameter arrays must be of length size() and must be in device memory. The return is a new instance containing the extracted strings.

```
NVStrings* slice_from( int* starts=0, int* stops );
```

| Parameter | Description |
|---|---|
| starts | Array of starting character positions. One per string. Default is null = beginning of each string. |
| stops | Array of ending character positions. Default is null = end of each string. |

Returns the character specified in each string as a new string. The NVStrings returned contains a list of single character strings.

```
NVStrings* get(unsigned int pos);
```

| Parameter | Description |
|---|---|
| pos | The character position identifying the character in each string to return. |

Extract all strings using the specified regex pattern. The return is an array of NVStrings instances one for each string or for each column of strings.

```
int extract( const char* ptn, std::vector<NVStrings*>& results );
int extract_column( const char* ptn, std::vector<NVStrings*>& results );
```

| Parameter | Description |
|---|---|
| ptn | Regex pattern used to find strings within each string. |
| results | The vector contains the strings extracted by the pattern.<br>For extract(), there is an NVStrings per string in this instance.<br>For extract_column, there is NVStrings per number of most extracts for any string. |

## Modifying

Replace a section of each string with the specified repl string using the character position values provided.

```
NVStrings* slice_replace( const char* repl, int start=0, int stop=-1 );
```

| Parameter | Description |
|---|---|
| repl | String to insert into the specified section. |
| start | Beginning of section to replace. Default is 0 = beginning of each string. |
| stop | End of section to replace. Default is -1 = end of each string. |

Replace all occurrences of the specified string in each string with a new string. The _re method accepts a regex pattern to find strings to replace.

```
NVStrings* replace( const char* str, const char* repl, int maxrepl=-1 );
NVStrings* replace_re( const char* pat, const char* repl, int maxrepl=-1 );
```

| Parameter | Description |
|---|---|
| str | String to replace in each string. |
| pat | Regex pattern to use to find matching strings to replace. |
| repl | String to insert in place of the matched one. |
| maxrepl | Maximum number of strings to replace. Default is -1 = replace all. |

Replace individual characters with new characters.

```
NVStrings* translate( std::pair<unsigned,unsigned>* table, unsigned int
count );
```

| Parameter | Description |
|---|---|
| table | Table of individual characters mapped to their replacement character.<br>These are expected to be unicode characters.<br>If a character is mapped to 0, that character will be removed from the string.<br>Characters not included are left unchanged. |
| count | Number of entries in the table. |

## Change Case

These create new the strings and modify the case of each character as appropriate. Only the 0x0000-0xFFFF set of unicode code-points are used when changing case.

```
NVStrings* lower();
NVStrings* upper();
NVStrings* capitalize();
NVStrings* swapcase();
NVStrings* title();
```

Example

```
NVStrings* strs = // created with ["Abc","dEf","ghI jkl"]
NVStrings* lwr = strs.lower();       // -- ["abc","def","ghi jkl"]
NVStrings* upr = strs.upper();       // -- ["ABC","DEF","GHI JKL"]
NVStrings* cpt = strs.capitalize();  // -- ["Abc","Def","Ghi jkl"]
NVStrings* swp = strs.swapcase();    // -- ["aBC","DeF","GHi JKL"]
NVStrings* ttl = strs.title();       // -- ["Abc","Def","Ghi Jkl"]
```

## Find/Compare

Search for specified string in the strings managed by this instance. A value of -1 indicates the string was not found. Otherwise the return value is the character position where the string was found. The Python module, **nvstrings'** *index* and *rindex* methods call these two methods as well. The return value is the number of successful finds. If device memory is provided, a result value of -2 indicates a null string.

```
unsigned int find( const char* str, int start, int end, int* results, bool
todevice=true );
unsigned int rfind( const char* str, int start, int end, int* results,
bool todevice=true );
```

| Parameter | Description |
|---|---|
| str | Null-terminated string to search for in each string. |
| start | Character position to start search. |
| end | End character position to confine search.<br>Specify -1 to indicate the end of the string. |
| results | Array to hold operation results must be allocated by the caller.<br>Memory must be able to hold size() of int values. |
| todevice | True if results parameter is allocated in device memory. |

Compare each string to the given string. Returns value of 0 where string matches. Returns < 0 when first different character is lower than argument string or argument string is shorter. Returns > 0 when first different character is greater than the argument string or the argument string is longer.

```
int* compare( const char* str, int* results, bool todevice=true );
```

| Parameter | Description |
|---|---|
| str | Null-terminated string to search for in each string. |
| results | Array to hold operation results must be allocated by the caller.<br>Memory must be able to hold size() of int values. |
| todevice | True if results parameter is allocated in device memory. |

This is like find() above except that the find ranges vary per string. Individual sections are identified by the starts and ends arrays.

```
unsigned int find_from( const char* str, int* starts, int* ends, int*
results, bool todevice=true );
```

| Parameter | Description |
|---|---|
| str | Null-terminated string to search for in each string. |
| starts | Array of start positions. One value for each string.<br>Array must be device memory. |
| ends | Array of end positions. One value for each string.<br>Array must be device memory. |
| results | Array to hold operation results must be allocated by the caller.<br>Memory must be able to hold size() of int values. |
| todevice | True if results parameter is allocated in device memory. |

This is like find() above except that the it accepts a list of strings for which to search.

```
unsigned int find_multiple( NVStrings& strs, int* results );
```

| Parameter | Description |
|---|---|
| strs | List of strings to search for |
| results | Array to hold operation results must be allocated by the caller. Memory must be able to hold size() of int values. |

These methods all occurrences of strings matching the specified regex pattern.

```
int findall( const char* ptn, std::vector<NVStrings*>& results );
int findall_column( const char* ptn, std::vector<NVStrings*>& results );
```

| Parameter | Description |
|---|---|
| ptn | Regex pattern for searching. |
| results | Vector of strings matching the ptn regex pattern.<br>For findall(), an NVStrings per string instance.<br>For findall_column, an NVStrings per largest number of matches for any string. |

These methods search for given strings or regex patterns in return True when they are found and False otherwise.

```
int contains( const char* str, bool* results, bool todevice=true );
int contains_re( const char* ptn, bool* results, bool todevice=true );
int match( const char* ptn, bool* results, bool todevice=true );
unsigned int startswith( const char* str, bool* results, bool
todevice=true );
unsigned int endswith( const char* str, bool* results, bool todevice=true
);
```

| Parameter | Description |
|---|---|
| str | String to search for within each string. |
| ptn | Regex pattern used to find strings within each string. |
| results | Array to hold operation results must be allocated by the caller. Memory must be able to hold size() of bool values. |
| todevice | True if results parameter is allocated in device memory. |

Count the number of occurrences a specified pattern appears in each string.

```
int count_re( const char* ptn, int* results, bool todevice=true );
```

| Parameter | Description |
|---|---|

| | |
|---|---|
| ptn | Regex pattern used to find strings within each string. |
| results | Array to hold operation results must be allocated by the caller. Memory must be able to hold size() of int values. |

## Conversion

Converts each string to number if string contains decimal values. It honors a prefix sign and decimal point as appropriate.

```
unsigned int stoi(int* results, bool todevice=true );
unsigned stof(float* results, bool todevice=true );
```

| Parameter | Description |
|---|---|
| results | The conversion results are filled into this array.<br>The caller must allocate this array and the memory must hold at least size() values of the corresponding type. |
| todevice | True if the results array is allocated in device memory. |

## Array Methods

This method will create an index pointing to the internal string device memory pointers. The return value will be 0 if successful.

```
int create_index(std::pair<const char*,size_t>* strs, bool todevice=true );
```

| Parameter | Description |
|---|---|
| strs | Pair values of string pointer and string length in bytes. These are pointers to internal device memory managed by this instance.<br>The caller must allocate this array and the memory must be able to hold at least size() values of the pairs. |
| todevice | True if the strs array is allocated in device memory. |

This method will set a bit array identifying any null string entries in this instance. The return value is 0 if successful.

```
unsigned int set_null_bitarray(unsigned char* bitarray, bool
emptyIsNull=false, bool todevice=true );
```

| Parameter | Description |
|---|---|
| bitarray | Bit-array to hold the results. Bit value of 0 indicates a null string at that bit's position.<br>The caller must allocate this array and the memory must be able to hold at least size()/8 values. |
| emptyIsNull | Set to true if empty strings should be identified with a 0 bit value. |
| devmem | True if the strs array is allocated in device memory. |

Sorts the strings manage by this instance and returns a new instance.

```
enum sorttype { none=0, length=1, name=2 };
NVStrings* sort( sorttype& st, bool ascending=true );
```

| Parameter | Description |
|---|---|
| st | How to sort the strings. If both length and name are specified the sort order is length and then name. |
| ascending | Ascending or descending sort. Default is true = ascending. |

Create a new instance using or not using the specified strings from this instance.

```
NVStrings* sublist( unsigned int* pos, unsigned int count );
NVStrings* remove_strings( unsigned int* pos, unsigned int count, bool
devmem=true );
```

| Parameter | Description |
|---|---|
| pos | Array of 0-based indexes of strings to keep or remove. |
| count | The number of indexes provided. |
| devmem | True if the pos indexes are in device memory. |