StringList Library
Version 0.1.1

Richard D. Clark

Clark Productions Software

rickclark58 [at] gmail [dot] com

Table of Contents

Introduction....3
  License....3
StringList Public Interface....4
  Constructor, Destructor....4
  Properties....4
  Functions....4
  Subroutines....4
Constructor....6
  Namespace....6
  Creating a New StringList....6
Destructor....6
Property Version (Read Only)....6
Property Count (Read Only)....7
Property Sorted (Read Only)....7
Property Strings (Read Write)....7
Property Text (Read Write)....7
Property CaseSensitive (Read Write)....8
Property SortAscending (Read Write)....8
Function LoadFromFile....8
Function SaveToFile....9
Function Add....9
Function InsertItem....9
Function DeleteItem....10
Function Move....10
Function Find....10
Function IsEqual....11
Sub ClearList ()....11
Sub SwapItems....11
Sub Sort ()....11
Sub Copy 12

Introduction

The StringList library is a FreeBasic object that creates and manages a list of strings. The file stringlist.bi contains the code for the StringList object and and can be used in your program by including the file in your project.

```
'Include the stringlist.
#Include [Once] "stringlist.bi"
```

License

StringList Public Interface

The following lists the public interface elements of the StringList followed by an explanation of each element.

True/False

False is defined as 0.

True is defined as (Not False).

Constructor, Destructor

Constructor ()

Destructor ()

Properties

Version

Count

Sorted

Strings

Text

CaseSensitive

SortAscending

Functions

LoadFromFile

SaveToFile

Add

InsertItem

DeleteItem

Move

Find

IsEqual

Subroutines

ClearList

SwapItems

Sort ()

Copy

Constructor

```
Declare Constructor ()
```

The Constructor sets the default values of the StringList:

Sorted = False

List is set to NULL.

Count = 0

Case Sensitive = False

Sort Ascending = True

Version = Current Version

Namespace

The StringList is contained within the Namespace **strlist**. To access the object interface either preface the command with strlist or use the FreeBasic Using command.

```
'Set the namespace.
Using strlist
```

Creating a New StringList

To create a new StringList use the following code:

```
Dim myList As stringlist
```

Destructor

The destructor is called whenever the object goes out of scope. The destructor deletes all strings from the StringList and releases the memory used by the list. The destructor calls the private subroutine _DestroyList.

Property Version (Read Only)

```
Declare Property Version () As String
```

Returns the current version of the StringList as a string.

Example:

```
Print "Current version: "; myList.Version
```

Property Count (Read Only)

```
Declare Property Count () As Integer
```

Returns the number of strings in the StringList. The indexes of the strings range from 0 to Count – 1.

Example:

```
Print myList.Count
```

Property Sorted (Read Only)

```
Declare Property Sorted () As Integer
```

Returns the current sort state of the StringList. True indicates that the list is sorted, False that the list is not sorted. Various methods such as Add, Swap and Move reset the sorted status to False.

Property Strings (Read Write)

```
Declare Property Strings (index As Integer) As String
Declare Property Strings (index As Integer, item As String)
```

Returns or sets a string based on *index*. Index must be within the range 0 to Count – 1. The Strings property supports empty strings. The string must not contain the NULL character (character 0) since the StringList uses Zstrings internally to store the strings.

Example:

```
Print myList.Strings(myList.Count – 1)
myList.Strings(myList.Count - 1) = "This is a new string."
```

Property Text (Read Write)

```
Declare Property Text (delimiter As String) As String
Declare Property Text (delimiter As String, txt As String)
```

Returns or sets the list of strings based on the passed *delimiter* value. The returned value is a string containing all the strings in the StringList with each string delimited by the passed delimiter string. This property can also be used to replace the current list of strings with a set of strings in *txt* that are delimited by delimiter.

Example:

```
txt = myList.Text(Chr(13, 10))
myList.Text(Chr(13, 10)) = txt
```

Property CaseSensitive (Read Write)

```
Declare Property CaseSensitive () As Integer
Declare Property CaseSensitive (scase As Integer)
```

Returns or sets the current case sensitivity flag. If False, sorting and comparison functions are not case sensitive. If True, sorting and comparison functions are case sensitive.

Example:

```
cs = myList.CaseSensitive
myList.CaseSensitive = TRUE
```

Property SortAscending (Read Write)

```
Declare Property SortAscending () As Integer
Declare Property SortAscending (sortdir As Integer)
```

Returns the current sort flag. If True, the list will be sorted in ascending order. If False the list will be sorted in descending order.

Example:

```
so = myList.SortAscending
myList.SortAscending = TRUE
```

Function LoadFromFile

```
Declare Function LoadFromFile (fname As String) As Integer
```

This function will attempt to load a list of strings from the file passed as fname. Each string in the file must be delimited by the standard end-of-line character produced by the FreeBasic Print # command. The function will return True if the file could be loaded or False if an error occurs. This function will set the Sorted flag to False.

Example:

```
chk = myList.LoadFromFile(flname)
```

Function SaveToFile

```
Declare Function SaveToFile (fname As String) As Integer
```

This function will attempt to save the string list to a text file. Each string in the file will be delimited by the standard end-of-file character produced by the FreeBasic Print # command. The function will return True if the file was created successfully or False if an error occurs.

Example:

```
chk = myList.SaveToFile(flname)
```

Function Add

```
Declare Function Add (item As String) As Integer
```

This function will attempt to add string *item* to the StringList. If successful, the string's index is returned. If an error occurs, -1 is returned. This function will set the Sorted flag to False.

Example:

```
idx = myList.Add("Item" & I)
```

Function InsertItem

```
Declare Function InsertItem (index As Integer, item As String) As Integer
```

This function will attempt to insert string *item* at the location *index*. Index must be in the range 0 to Count. If index is Count, the string is added to the end of the list. Each string is moved down one index value to accommodate the new string.

The function will return True if the string was successfully inserted or False if an error occurs. This function will set the Sorted flag to False.

Example:

```
ret = myList.InsertItem(0, "This string was inserted.")
ret = myList.InsertItem(myList.Count, "This string was also inserted.")
```

Function DeleteItem

```
Declare Function DeleteItem (index As Integer) As Integer
```

This function will attempt to delete a string at *index*. Index must be in the range of 0 to Count – 1. The remaining strings will be moved down one index. The function returns True if the string was deleted or False if an error occurs. This function will set the Sorted flag to False.

Example:

```
ret = myList.DeleteItem(myList.Count - 1)
```

Function Move

```
Declare Function Move (fromindex As Integer, toindex As Integer) As Integer
```

This function will attempt to move string from *fromindex* to *toindex*. Fromindex must be in the range 0 to Count – 1. Toindex must be in the range 0 to Count. If fromindex and toindex are the same, no action occurs. Move performs an InsertItem at toindex followed by a DeleteItem at fromindex. The function returns True if the string was moved successfully or False if an error occurs. This function will set the Sorted flag to False.

Example:

```
ret = myList.Move(0, myList.Count - 1)
```

Function Find

```
Declare Function Find (fkey As String, ByRef index As Integer) As Integer
```

This function will search the list for *fkey*, returning the index value, if found, in *index*. If the key is found, index will contain the string's index and the function will return True. If the key is not found, index will contain -1 and the function will return False.

The comparison used is based on the current CaseSensitive setting. If the list is sorted in ascending order, the function will search the list using a binary search. If the list is unsorted, or sorted in descending order, a linear search is used.

Example:

```
ret = myList.Find("item17", idx)
```

Function IsEqual

```
Declare Function IsEqual(sl As stringlist) As Integer
```

This function will compare the passed StringList *sl* to the current StringList to see if they are equal. The two StringLists must have the same number of strings, and each string must be equal. The function uses the CaseSensitive setting to compare the individual strings. The function returns True if equal, False if not. Two empty StringLists are considered equal.

Example:

```
ret = myList.IsEqual(myList1)
```

Sub ClearList ()

```
Declare Sub ClearList ()
```

This subroutine will delete all strings and deallocate the memory used by the StringList. The subroutine calls the private subroutine _DestroyList. After clearing the list, the Count will be 0 and the Sorted flag will be False.

Example:

```
myList.ClearList
```

Sub SwapItems

```
Declare Sub SwapItems (index1 As Integer, index2 As Integer)
```

This subroutine will swap the string at *index1* with the string at *index2*. Index1 and index2 must be in the range of 0 to Count – 1. If index1 equals index2 then no swap occurs. This function will set the Sorted flag to False.

Example:

```
myList.SwapItems 0, myList.Count - 1
```

Sub Sort ()

```
Declare Sub Sort ()
```

This subroutine sorts the list based on the SortAscending flag and the current CaseSensitive setting. It uses the CRT qsort procedure which calls the private function _Qcompare. After sorting, the Sorted flag is set to True.

Example:

```
myList.Sort
```

Sub Copy

```
Declare Sub Copy (ByRef sl As stringlist)
```

This subroutine will copy the current StringList to *sl*. All the strings and settings, except the Sorted flag, are copied as well.

Note: Do not use the assignment operator = to make a copy of the StringList. = will only copy a reference to the original StringList and modifications to the copy will affect the original, including clearing the StringList when the copy goes out of scope. Using = will generate undefined behavior and could crash the program. Always use Copy to create a copy of the StringList.

Example:

```
myList.Copy myList1
```

Change Log

Version 0.1.1

- Deleted sbool Enumeration and replaced the True/False definitions with defines. All methods and variables that used sbool have been changed to Integer. If you are using sbool in your program, change the references to Integer.

Version 0.1

- Released working version of code.