

ICFM Software
TStringClass - Version 2.0

Contents

[Introduction](#)

[Copyright Notice](#)

[Shareware Registration](#)

[Working with TStringClass objects](#)

[Character Positions](#)

[Parameters](#)

[Working with VCL text components](#)

[Pushing data directly into the class text buffer](#)

[Using the Global string class variable 'gSc'](#)

[Exception Handling](#)

[Objects/Classes](#)

[TStringClass methods by category](#)

[TStringClass methods in Alphabetical order](#)

[Version Management](#)

[Technical Support](#)

TStringClass Version 2.0 Character Positions

The TStringClass object is based around a core PChar object data member. Like a PChar variable all character position referencing is normally zero (0) based. In that the first character in a PChar string is at position 0, as shown below ...

```
VAR
  APChar : Pchar;
  AStr : STRING;
  FirstCharacter : CHAR;
  SObj : TStringClass;
BEGIN
  ....
  ....
  GetMem(APChar,200);
  StrCopy(APChar,'Spring');
  AStr := 'String';
  FirstCharacter := APChar[0];
  FirstCharacter := AStr[1];
  SObj := TStringClass.Create;
  SObj.Assign([AStr]);
  FirstCharacter := SObj.Ch[0];
  ....
  ....
```

As a favour to those hard-core STRING users who are unable or unwilling to remember that PChar positioning is zero based, the StrClass library includes a new optional facility that allows all character position referencing to be one (1) based. The boolean typed constant 'StrClassBaseZero' is by default to TRUE. If this constant is reset to FALSE then the first character position is treated as '1' not '0'.

With 'StrClassBaseZero' set to FALSE the following is correct ..

```
VAR
  APChar : Pchar;
  AStr : STRING;
  FirstCharacter : CHAR;
  SObj : TStringClass;
BEGIN
  ....
  ....
  StrClassBaseZero := FALSE;
  ....
  ....
  GetMem(APChar,200);
  StrCopy(APChar,'Spring');
  AStr := 'String';
  FirstCharacter := APChar[0];
  FirstCharacter := AStr[1];
  SObj := TStringClass.Create;
  SObj.Assign([AStr]);
  FirstCharacter := SObj.Ch[1];
  ....
```

....
Note: this change to a base 1 first character affects all class methods that use or report a character position.

TStringClass Version 2.0

Working with TStringClass objects

To employ the TStringClass object in any of your units add the 'StrClass' name to the list of units found in the USES clause. For example

```
unit Testbed;
```

```
interface
```

```
uses
```

```
    SysUtils,  
    WinTypes,  
    WinProcs,  
    Messages,  
    Classes,  
    Graphics,  
    Controls,  
    Forms,  
    Dialogs,  
    StdCtrls,  
    ExtCtrls,  
    StrClass,  
    ContainR;
```

Some of the TStringClass object methods use a container object to hold lists of strings. This container object is located within the 'ContainR.Pas' unit. If you are employing any of these container related functions make sure the 'ContainR' unit is included in the 'USES' list as shown above.

TStringClass.AddFilterDesc

FUNCTION AddFilterDesc(CONST DescArgs,ExtArgs : ARRAY OF CONST) : PChar;

This method builds a string in the format required by the 'Filter' property of the 'TOpenDialog' VCL component. 'DescArgs' represents a description of the file type, whilst 'ExtArgs' holds the file extension. For example, the command

```
SObj.AddFilterDesc(['Pascal source'],['PAS'])
```

..... builds the string

```
Pascal Source (*.PAS)|*.PAS
```

The same method can be called multiple times, with each secondary call adding the new formatted text to the end of the object text string. For example ...

```
SObj.AddFilterDesc(['Pascal source'],['PAS'])  
SObj.AddFilterDesc(['C source'],['C'])
```

..... builds the string

```
Pascal Source (*.PAS)|*.PAS|C Source (*.C)|*.C
```

The 'ExtArgs' can include either or both of the '*' or '.' characters. If either or both are missing the method adds them to the resulting string.

TStringClass Version 2.0

Introduction

This **TStringClass** object is designed to manage large string variables by encapsulating a core PChar type text buffer within a controlled object wrapper.

Pascal 'STRING' type text variables are easy to use and manipulate, but suffer from a having a maximum size of 255 characters. If you are processing large arrays of STRING type variables they can waste valuable stack or heap space.

The alternative to using STRING types is to use the null terminated string (or 'PChar') type. For most programmers PChar's are a necessary evil.

This 'TStringClass' object was originally developed to solve some of the problems that are frequently encountered whilst using 'PChar' type variables, namely:

- * Using un-initialised or 'NIL' PChar variables with any of the STRINGS unit functions, and
- * Not declaring buffers of sufficient length, such that concatenating variables leads to internal memory overwrites.

The TStringClass object offers solutions to these and other related problems. It controls its own internal buffer for holding the PChar variable, and before performing any assignments or concatenations it always checks to see that sufficient room is available. If there is insufficient space it re-sizes the buffer to fit the required action.

This object was originally created to counteract problems with 'guess-timating' the PChar variable buffer length for complicated SQL expressions. Since then it has been expanded to cover almost anything that can be done with a string type variable.

The object has the following main categories of methods:

- Multiple constructors for initiating the object for different situations
- A set of high level methods for working with other TStringClass type variables.
- A range of different 'Assign' methods for moving different types of text variables into the object data value.
- A range of different 'Append' methods for adding different types of text variables onto the end of any existing object data value
- A set of character level methods for testing the object data string.
- A set of methods associated with converting other numeric data types into strings, or vice-versa
- A set of methods that mimic the functions found in the STRINGS unit
- A set of methods that mimic the text related functions found in the 'SysUtils' unit.
- Methods for string resource and INI file access
- A set of methods for inserting, deleting, padding and trimming the object's data string.
- A set of methods to manage comparisons with another PChar variable.
- A set of methods to provide DOS/Path name related processing and testing.
- Methods for searching characters and sub strings
- Methods for changing case
- Methods for replacing characters or sub strings.
- Methods for parsing the object's data value

As a rule the object is designed to handle the problems of passing NIL or zero length PChar parameters.

It can also manage the thorny old problem of passing parameters which are un-initialised or remain in use after they have been disposed. These are captured by exception handling.

TStringClass Version 2.0 Parameters

With Object Pascal there are several different types of variables that can be used to store text strings, as shown below

```
VAR
    MyArray :    ARRAY[0..200] OF CHAR;
    MyPChar :    PChar;
    MyStr :      STRING;
```

Rather than create a mass of different object methods to handle different parameter types, I decided to design TStringClass so that any type of parameter can be passed to the same function.

This is possible with the new DELPHI Object Pascal 'Open Array constructors' facility. This allows a function or procedure to be created that accepts an open ended set of parameters of different variable types. (See Chapter 8 page 82 of the Language Guide).

Most of the TStringClass object methods that have text type input parameters use this type of parameter. For instance

```
FUNCTION Assign(CONST Args : ARRAY OF CONST) : PChar;
```

The Assign method converts the 'Args' collection of parameters to a string and then assigns that string to the object, replacing any existing string value.

One condition to using 'ARRAY OF CONST' type parameters is that the entries must be enclosed within square brackets, as shown below

```
VAR
    Sobj : TStringClass;
    L : LONGINT;
    AStr : STRING;
BEGIN
    .....
    .....
    L := 5;
    AStr := 'I am ';
    SObj := TStringClass.Create;
    SObj.Assign(['There are ',L,' boats in the harbour']);
    SObj.Assign([AStr,L,' years old']);
    .....
    .....
```

The TStringClass object will accept and process all standard variables types, it will even accept other TStringClass objects as items in the parameter list.

```
VAR
    Sobj,Tobj : TStringClass;
    L : LONGINT;
    AStr : STRING;
BEGIN
    .....
```



```

....
L := 5;
AStr := 'I am ';
SObj := TStringClass.Create;
TObj := TStringClass.CreateString(['its my birthday']);
SObj.Assign([AStr,L,' years old and ',TObj]);
.....
.....

```

With Version 2 of TStringClass, the Args parameter list can now include text related VCL components, such as memo and edit box controls. For example :

```

VAR
  Sobj : TStringClass;
  TheMemo : TMemo;
BEGIN
  ....
  ....
  TheMemo.Lines.Add('This is the 1st line');
  TheMemo.Lines.Add('This is the 2nd line');
  SObj := TStringClass.Create;
  SObj.Assign([TheMemo]);
  .....
  .....

```

Where an ARRAY OF CONST type parameter is passed to a TStringClass object they will be processed in a left to right order. For a function like 'Assign', the sum of all parts within the 'ARRAY OF CONST' will be converted to a string - in a left to right order - and the resulting string will be assigned to the object's own text variable.

To help identification all 'ARRAY OF CONST' type parameters have an 'Args' part to their name.

Where an object's method returns strings or part strings as results, the function requires another instantiated TStringClass object as the target for that assignment.

Some of the parsing related functions use a container object to store lists of parsed sub strings. In such cases the function uses a 'TObjectContainer' object as the storage container. Refer to chapter 9 for more information of the container object hierarchy.

Certain of TStringClass's methods make use of a container method to hold the results of a parsing or string extracting process. For example the parsing function 'ParseDelimToList' will chop a delimited text into its parts and place each part into a string object which inturn is added to the parameter list object. For example

```

AList := TObjectContainer.Create;
AList.Capacity := 100;
TObj := TStringClass.CreateString(['xxx,yyy,zzzz,www,,qqqq,rrrr']);
TObj.ParseDelimToList(',',delim_None,AList);

```

The 'AList' parameter is a container object. It is an instance of the 'TObjectContainer' open array type container provided within this library.

With the example above 'AList' now holds a list of 6 items - each of which is an instance of TStringClass. To gain access to one of these string parts use the object container's 'Items' property, for example ..

```
AStrObj := TStringClass(AList.Items[0]); { first string part - xxx }  
AStrObj := TStringClass(AList.Items[AList.Count-1]); { last string part - rrrr }
```

TStringClass Version 2.0

Working with VCL text components

The TStringClass library now includes functions for moving data in and out of VCL text related controls. These functions are useful for situations where a control contains multi-line strings that exceed the 255 character limit.

Any method that includes an 'Args' ARRAY OF CONST type parameter can now include one or more VCL components in its list.

The TStringClass object will process any VCL component that drives from the parent class 'TWinControl'. This includes the TEdit and TMemo controls. In all cases it is using the 'GetTextBuf' and 'SetTextBuf' VCL methods to move text in and out of a control.

There are also separate methods for working with VCL components.

The 'FromComponent' method will extract the text from a single designated component control and assign it to the string class's own text buffer.

The 'ToComponent' moves a text string from the string class into the VCL component's own text buffer.

The 'FromComponentItem' method extracts part of the component's own string list and assigns it to the string class buffer. This method has a second 'Idx' INTEGER parameter that is used to determine which part. For list boxes and combo boxes this is equivalent to the list item number. For Memo controls it is equivalent to the text line number. Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

For memo controls only, if the 'Idx' parameter is set to the constant 'selected_text' then the method extracts the highlighted text.

TStringClass Version 2.0

Pushing data directly into the class text buffer

Strange thought his might seem there are situations where you might want to push data directly into the object's text buffer.

Although such actions are very un-OOP there can often occur. Typically they arise where a low level function requires a 'PChar' type parameter. In such a situation you would normally have to cope as follows

```
VAR
    AStr : PChar;
    SObj : TStringClass;
BEGIN
    ....
    ....
    GetMem(AStr,1000);
    SObj := TStringClass.Create;
    SomeLowLevelFunction(AStr,1000);
    SObj.Assign([AStr]);
    FreeMem(AStr,1000);
    ....
    ....
END
```

Having to create and dispose of the 'AStr' local variable is a pain. By contrast TStringClass offers a method of pushing data directly into its buffer, as shown below:

```
VAR
    AStr : PChar;
    SObj : TStringClass;
BEGIN
    ....
    ....
    SObj := TStringClass.CreateSize(1000);
    SomeLowLevelFunction(SObj.ZString,1000);
    SObj.RecalcLength;
    FreeMem(AStr,1000);
    ....
    ....
END
```

Fundamental to this approach is the use of the 'RecalcLength' method. This is used to re-calibrate the object's own internal data members.

This approach of pushing data indirectly into the object's text buffer is a short cut which ICFM Software cannot wholeheartedly recommend. However, not wishing to straight jacket our users it is issue which we felt was worth documenting.

TStringClass Version 2.0

Using the Global string class variable 'gSc'

The TStringClass object makes extensive use of exception handling. It has its own 'EStringClass' exception class.

```
EStringClass = class(Exception);
```

In reporting an exception error the TStringClass object will raise an exception with a description that includes the class name, function name and cause.

As some TStringClass object methods are called by other methods, the process of raising exceptions may actually cause several such messages to appear.

To minimise the space consumed by text error descriptions, all TStringClass exception reporting strings are held as a custom resource in the 'StrClass.Res' file. (Not a string table, a custom resource!)

Exception Handling

The TStringClass object makes extensive use of exception handling. It has its own 'EStringClass' exception class.

```
EStringClass = class(Exception);
```

In reporting an exception error the TStringClass object will raise an exception with a description that includes the class name, function name and cause.

As some TStringClass object methods are called by other methods, the process of raising exceptions may actually cause several such messages to appear.

To minimise the space consumed by text error descriptions, all TStringClass exception reporting strings are held as a custom resource in the 'StrClass.Res' file. (Not a string table, a custom resource!)

Objects/Classes

TStringClass

TBaseContainer

TObjectContainer

TRecordContainer

TPCharContainer

TIntegerContainer

TWordContainer

TLongIntContainer

TStringClass Version 2.0

Methods/Functions by category

Constructors/Destructors

Copy related

Clear type functions

Admin type functions

Assign related functions

Append related functions

Character element related functions

Other data type related functions

SysUtils unit compatible functions

Strings unit compatible functions

Comparison related functions

Insert/Delete/Trim related functions

Command Line related functions

Resource string related functions

INI file related functions

DOS path/filename related functions

Search related functions

Case related functions

Search & replace related functions

Parsing related functions

TStringClass Methods in Alphabetical order

AddBackSlash
AddDIC
AddFilterDesc
AddSIC
AnsiCompareStr
AnsiCompareText
AnsiLowerCase
AnsiUpperCase
Append
AppendBoolean
AppendByte
AppendCh
AppendDIC
AppendDouble
AppendDoubleTrim
AppendExt
AppendExtTrim
AppendLen
AppendLong
AppendMid
AppendNL
AppendPad
AppendPtr
AppendReal
AppendRight
AppendSIC
AppendStr
AppendStringRes
AppendTrim
AppendWithTab
Assign
AssignFrom
AssignLen
AssignMid
AssignNL
AssignPad
AssignRight
AssignTrim
BuildPathName
ChCount
Clear
Compare
CompareDouble
CompareExt
CompareI
CompareL
CompareLI
CompareLong
CompareStr
CompareText
Copy
CopyFrom

Create
CreateBoolean
CreateDirectory
CreateSize
CreateString
CreateStringNL
DefaultExtension
Delete
DeleteFrom
Destroy
DirectoryExists
DriveExists
Empty
ExpandFileName
FileExists
FileSplit
FindBetween2Ch
FindCmdLine
FindCmdLineAndParse
FindCmdLineParam
FindCurrentDir
FindFirst
FindFirstCh
FindIniSectionKeywords
FindLast
FindLastCh
FindNext
FindNextCh
FindPrev
FindPrevCh
FindRelPath
FirstCharToUpper
FirstNonSpaceCh
FirstParseDelim
FloatToStr
FloatToStrF
FmtLoadStr
ForceExtension
Format
FormatFloat
FromBoolean
FromByte
FromChar
FromComponent
FromComponentItem
FromDouble
FromDoubleTrim
FromExt
FromExtTrim
FromLong
FromPtr
FromReal
FromRealTrim
FromRGB
GetSystemDirectory
GetWindowsDirectory

Grep
HasBackSlash
HasCh
HasDirectory
HasDrive
HasExtension
HasFileName
HexFromByte
HexFromLong
HexFromPtr
HexFromWord
Includes
Insert
InsertL
IntToHex
IntToStr
IsAlphaNumeric
IsCh
IsFirstCh
IsLastCh
IsSame
IsSameI
IsSameL
IsSameLI
IsValidIdent
JustDirectory
JustExtension
JustFileName
JustName
LastNonSpaceCh
LoadStr
LoadStringRes
LowerCase
NextParseDelim
NL
NLAppend
PadCentre
PadEnd
PadFront
ParseDelimCount
ParseDelimToList
ParseMultiDelimToList
ParsePosToList
Prepend
ReadIniKeyword
RecalcLength
RemoveDIC
RemoveDirectory
RemoveLastCh;
RemoveSIC
ReplaceAll
ReplaceChAll
ReplaceChFirst
ReplaceChLast
ReplaceFirst
ReplaceLast

SameDirectory
SameDrive
SameExtension
SameFileName
SameName
SetCh
SetCurDir
StrCat
StrComp
StrCopy
StrECopy
StrEnd
StrIComp
StrLCat
StrLComp
StrLCopy
StrLen
StrLIComp
StrLower
StrMove
StrPas
StrPCopy
StrPos
StrRScan
StrScan
StrToFloat
StrToInt
StrToIntDef
StrUpper
SubStrCount
ToBoolean
ToByte
ToChar
ToComponent
ToDouble
ToExt
ToInteger
ToLong
ToLower
ToReal
ToRGB
ToUpper
ToWord
Trim
TrimEnd
TrimFront
TrimZero
UpperCase
Within
WriteIniKeyword

Version Management : enhancements and bug fixes by minor version

Changes for Version 1.1

Known Bugs & Bug fixes

JustDirectory

Fixed a bug where for parameters such as 'c:\windows' the process was extracting 'C:\' as the directory. The function now tests for the existence of the '.' dot and the number of '\' delimiters. If no dot is present and the number of '\' is less than 2 then the parameter string is assumed to be a directory and is simply assigned to the object.

FindCmdLineParam, FindCmdLine, FindCmdLineParam

If called as part of a DLL these functions might fail.

IsSameI

The function had an erroneous 'Len' 2nd parameter which has been removed.

New functions:

DeleteFrom
FindRelPath

Changes for Version 1.2

Known Bugs & Bug fixes

Container objects - using sorted object containers

The TObjectContainer class if pre-declared as 'Sorted' failed to sort the objects, even crashed in certain situations. There were multiple (coding error) reasons for this. The 'Compare' and 'Search' methods were faulty in accessing pointers on the list. (See also the new 'CompareFunc' function property).

Container Objects - Capacity property : un-initialised memory arrays

The 'SetCapacity' method failed to initialise local memory block memory arrays to zero. This led to a situation where a direct assignment of an item via the 'Items' property failed to increment the container item count. (This error only applied to small lists where the item pointer array was less than 64K in size).

ReadIniKeyword - buffer length

Within this method the internal PChar variable used as the target buffer for the initial INI file read was limited to 255 characters. Thus, if the INI file entry was longer than 255 characters it only read the first 255. This has been increased to 1000 (can anybody envisage an INI file entry exceeding 1000 characters??)

New functions

Container object - CompareFunc' function property.

Changes for Version 1.3

Known Bugs & Bug fixes

ReadIniKeyword - no clear if key word not found

If the 'ReadIniKeyword' function was used on a string class that already contained a text value, and the keyword does not exist or has a null value, the object text value was not being reset to null. This has been fixed so that in such situations the object's own text buffer is reset to null.

Changes for Version 1.4

Known Bugs & Bug fixes

ParseDelimToList - parse error if delimiter char is last char in source string.

The 'ParseDelimToList' function failed to parse a source string correctly where the delimiter was the last character in the source string.

Previously the source string 'A***' with a delimiter of '*' was being returned as a list of two string objects 'A' and null, when it should have been three: 'A', null and null. This has been fixed so that any source string with a trailing delimiter character will always return a list with an extra null sub string entry.

TBaseContainer.SortInPlace : memory leak

The quick-sort procedure used by this method was using two private unit level pointers which were not being released at the end of the process, thus losing 8 bytes of memory each time the process was called. This has been fixed.

Changes for Version 1.5

New TStringClass methods:

FUNCTION SameDrive(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameDirectory(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameExtension(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameFileName(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameName(CONST Args : ARRAY OF CONST) : BOOLEAN;

Changes for Version 1.6

Known bugs & bug fixes

ParseDelimToList : 'delim_IncSIC' and 'delim_IncDIC' - not implemented

The use of the 'delim_IncSIC' and 'delim_IncDIC' parameters was documented by NOT implemented. This has been fixed.

New TStringClass methods:

FUNCTION AddDIC : PChar;
FUNCTION AddSIC : PChar;

Changes for Version 1.7

Bug fixes:

TBaseContainer 'Move' method

This included a coding error that trashed the target pointer.

TBaseContainer 'SetCapacity' method

A similar error to the above, it only affected containers where a local memory to global memory move was enacted.

New TStringClass functionality:

The 'StrClassBaseZero' typed constant used for setting base 0 or base 1 character positioning.

New TStringClass method functions:

FUNCTION GetSystemDirectory : PChar;
FUNCTION GetWindowsDirectory : PChar;
FUNCTION NL : PChar

Changes for Version 2.0

New Functionality:

TStringClass

Args ARRAY OF CONST type parameters now except VCL text related components

'gSc' global string class variable

New 'delim_ExcNull' constant that can applied to 'ParseDelimToList'.

'AddFilterDesc'
'FromComponent' method
'FromComponentItem' method
'Grep' method
'ParseMultiDelimToList' method
'RecalcLength' method procedure
'RemoveDirectory' method function
'ToComponent' method

TObjectContainer: new methods :

FirstThatIdx
ForEachIdx
LastThatIdx

Bug fixes:

TStringClass Destroy not virtual

A major bug and cause of memory leaks! Sorry! Somehow the 'OVERRIDE' qualifier was left off the end of the Destroy interface declaration.

PadEnd/PadFront/Pad Centre - one character too long.

The Pad functions were adding one too many characters to the padding process.

Changes for Version 2.01

TBaseContainer object : 'AddItem method - duplicates bug

If a duplicate item was being added to a container and 'Duplicates' was set to 'duplgnore', this method was still reporting an exception error. This has been changed so that with 'duplgnore' active, any attempt to add a duplicate item is ignored with **no** exception error being created.

TPCharContainer : missing 'Compare' method

The TPCharContainer object was missing a 'Compare' method to enable sorted PChar lists. This has been added.

Technical Support

To raise questions, or report bugs and/or enhancement requests contact

ICFM Software
London,UK

Compuserve a/c [100010,1415]

TStringClass Object

Data Members

Constructors/destructors

Protected Methods

Public Methods

Properties

TStringClass Data Members

All the object's data members are deliberately set as 'PRIVATE' to ensure that the object's buffer management is left to internal procedures. (Note: all object data members start with the prefix 'F'. It is usually the case that all local variables and parameters start with a prefix of 'A').

FBuffer : PChar;

This holds the object's PChar data value.

FMaxSize : WORD;

This stores the size of the 'FBuffer' PChar variable memory buffer.

FLength : WORD;

This holds the length of the 'FBuffer' data member - excluding the null terminator. Thus a 'FBuffer' value of 'CAT' would equate to a 'FLength' value of 3.

FSizeInc : WORD;

This variable can be used to control by how much the internal memory buffer is increased. It can be set via the 'SizeInc' property.

TStringClass.CreateSize

CONSTRUCTOR CreateSize(ASize: WORD);

Instantiates the object with a NIL PChar value and an initial internal buffer size of 'ASize'. This constructor is useful for situations where lots of individual appends are anticipated, thus avoiding lots of individual buffer re-sizing.

Example :

```
AStrObj := TStringClass.CreateSize(2000);
```

TStringClass.Create

CONSTRUCTOR Create;

Instantiates the object with a NIL default PChar value and zero length internal buffer.

Example :

```
AStrObj := TStringClass.Create;
```

TStringClass Constructors/Destructors

```
CONSTRUCTOR Create;  
CONSTRUCTOR CreateSize(ASize: WORD);  
CONSTRUCTOR CreateString(CONST Args : ARRAY OF CONST);  
CONSTRUCTOR CreateStringNL(CONST Args : ARRAY OF CONST);  
CONSTRUCTOR CreateBoolean(B : BOOLEAN; StrType : INTEGER);  
  
DESTRUCTOR Destroy; OVERRIDE;
```

TStringClass.CreateString

CONSTRUCTOR CreateString(CONST Args : ARRAY OF CONST);

Instantiates the object with a starting PChar value equivalent to the 'Args' parameters. The Args array can include any mix of standard variable types including TStringClass and VCL text related objects.

Example:

```
VAR
  AString : STRING
  APChar : Pchar;
  L       : LONGINT;
  AStrObj : TStringClass;
BEGIN
  .....
  AString := 'There are ';
  L := 5;
  GetMem(APChar,256);
  StrCopy(APChar,' shopping months to Christmas');
  AStrObj := TStringClass.CreateString([AString,L,APChar]);
  .....
```

TStringClass.CreateStringNL

CONSTRUCTOR CreateNL(CONST Args : ARRAY OF CONST);

Instantiates the object with a starting string value (as with 'CreateString'), and appends a new line instruction (#13#10) to the end of the PChar data member.

TStringClass.CreateBoolean

CONSTRUCTOR CreateBoolean(B : BOOLEAN; StrType : INTEGER);

Instantiates the object with a starting PChar value converted from a BOOLEAN parameter. The 'StrType' parameter determines the type of string conversion, where

<u>StrType constant</u>	<u>String conversion(TRUE/FALSE)</u>
bt_NoYes	No / Yes
bt_01	0 / 1
bt_FalseTrue	False / True
bt_FT	F / T
bt_NY	N / Y
bt_OffOn	Off / On

If the 'StrType' is not one of the constant values shown above then no conversion or assignment is made.

TStringClass.Destroy

DESTRUCTOR Destroy;

This disposes of all data members and destroys the object. (Do not call this directly. Use the 'Free' method to dispose of a object)

```
VAR
    AStrObj : TStringClass;
BEGIN
    .....
    AStrObj := TStringClass.Create;
    .....
    .....
    AStrObj.Free;
```

TStringClass Copy related Methods

```
FUNCTION Copy : POINTER; VIRTUAL;  
PROCEDURE CopyFrom(Source : TStringClass);
```

TStringClass.Copy

FUNCTION Copy : POINTER; VIRTUAL;

This function instantiates and returns a copy of itself.

TStringClass.CopyFrom

PROCEDURE CopyFrom(Source : TStringClass);

Copies the attributes and text string buffer of the parameter object 'Source'.

TStringClass Clear type functions

PROCEDURE Clear;
PROCEDURE Empty;

TStringClass.Clear

PROCEDURE Clear;

Disposes of any object string, clears the internal buffer length to NIL and all other object attributes to 0.

TStringClass.Empty

PROCEDURE Empty

Clears the internal text string to '#0', but leaves the buffer size intact.

TStringClass Admin type functions

PROCEDURE RecalcLength:

TStringClass.RecalcLength;

PROCEDURE RecalcLength;

This method re-calculates the length of the text string found in the 'FBuffer' and assigns the result to the 'FLength' private variable. This method only ever need be called if a process pushes data directly into the objects text buffer.

TStringClass Assign related functions

FUNCTION Assign(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AssignFrom(CONST Args : ARRAY OF CONST; Start : WORD) : PChar;
FUNCTION AssignLen(CONST Args : ARRAY OF CONST ; Len : WORD) : PChar;
FUNCTION AssignMid(CONST Args : ARRAY OF CONST; Start,Count : WORD) : PChar;
FUNCTION AssignNL(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AssignPad(CONST Args : ARRAY OF CONST; Len : WORD; ACh : CHAR) : PChar;
FUNCTION AssignRight(CONST Args : ARRAY OF CONST; Len : WORD) : PChar;
FUNCTION AssignTrim(CONST Args : ARRAY OF CONST) : PChar;

TBaseContainer.Clear

PROCEDURE Clear; VIRTUAL;

Clears all items from the list, disposing of any records/objects on the list.

TStringClass.Assign

FUNCTION Assign(CONST Args : ARRAY OF CONST) : PChar;

Copies the 'Args' list of parameters into the object, replacing any existing object string value. If 'Args' contains more than one element, then the first entry is 'assigned' to the object, and all subsequent elements are appended. If all elements represent NIL or blank text parameters then NIL is assigned to the object text string.

Example:

```
VAR
    AString : STRING;
    Sobj : TStringClass;
BEGIN
    ....
    ....
    AString := 'Now is the winter of our discontent'
    Sobj := TStringClass.Create;
    SObj.Assign([AString]);    { 'Now is the winter of our discontent' }
```

TStringClass.AssignFrom

FUNCTION AssignFrom(CONST Args : ARRAY OF CONST; Start : WORD) : Pchar;

Converts the 'Args' parameter list into a text string and then assigns part of the string starting from position 'Start' (base 0) to the object text buffer.

Example:

```
VAR
    AString : STRING;
    Sobj : TStringClass;
BEGIN
    ....
    ....
    AString := 'Now is the winter of our discontent'
    Sobj := TStringClass.Create;
    SObj.AssignFrom([AString],7);    { 'the winter of our discontent' }
```

If the 'Args' generates a NIL string, or the 'Start' parameter is greater than or equal to the 'Args' string length then a NIL string value is assigned to the object text buffer.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.AssignLen

FUNCTION AssignLen(CONST Args : ARRAY OF CONST; Len : WORD) : PChar;

Converts the 'Args' parameter list into a text string and then assigns the first part of the parameter string up to 'Len' characters to the object text buffer.

Example:

```
VAR
    AString : STRING;
    Sobj : TStringClass;
BEGIN
    ....
    ....
    AString := 'Now is the winter of our discontent'
    Sobj := TStringClass.Create;
    SObj.AssignLen([AString],6);    { 'Now is' }
```

If 'Len' exceeds the 'Arg' text string length then the whole 'Args' string is assigned to the object text buffer.

TStringClass.AssignMid

FUNCTION AssignMid(CONST Args : ARRAY OF CONST; Start,Count : WORD) : PChar;

Converts the 'Args' parameter list into a text string and then assigns part of the parameter string from 'Start' (base 0) for 'Count' characters to the object text buffer.

Example:

```
VAR
    AString : STRING;
    Sobj : TStringClass;
BEGIN
    ....
    ....
    AString := 'Now is the winter of our discontent'
    Sobj := TStringClass.Create;
    SObj.AssignMid([AString],7,3);    { 'the' }
```

If 'Start' exceeds the parameter string length or the parameter string is Nil then a NIL string is assigned to the object.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.AssignNL

FUNCTION AssignNL(CONST Args : ARRAY OF CONST) : PChar;

As with 'Assign' this converts the 'Args' parameter to a text string and assigns it to the object text buffer. In addition this function appends a 'New Line' character combination (#13#10) to the end of the object string.

TStringClass.AssignPad

**FUNCTION AssignPad(CONST Args : ARRAY OF CONST;
 Len : WORD;
 ACh : CHAR) : PChar;**

As with 'Assign' this converts the 'Args' parameter to a text string and assigns it to the object text buffer. If the resultant text string is less than 'Len' characters long, the object string is right padded with multiple 'ACh' characters to bring it up to the length 'ALen'.

Example:

```
VAR
  AString : STRING;
  Sobj : TStringClass;
BEGIN
  ....
  ....
  AString := 'Now is"
  Sobj := TStringClass.Create;
  SObj.AssignPad([AString],8,'X');   { 'Now isXX' }
```

TStringClass.AssignRight

FUNCTION AssignRight(CONST Args : ARRAY OF CONST; Len : WORD) : PChar;

Converts the 'Args' parameter list into a text string and then assigns the rightmost 'Len' characters to the object text buffer.

Example:

```
VAR
    AString : STRING;
    Sobj : TStringClass;
BEGIN
    ....
    ....
    AString := 'Now is the winter of our discontent'
    Sobj := TStringClass.Create;
    SObj.AssignRight([AString],14);    { 'our discontent' }
```

TStringClass.AssignTrim

FUNCTION AssignTrim(CONST Args : ARRAY OF CONST) : PChar;

As with 'Assign' this converts the 'Args' parameter to a text string and assigns it to the object text buffer. The resultant text string is then trimmed to remove all leading and trailing space characters.

TStringClass Append related functions

FUNCTION Append(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AppendBoolean(B : BOOLEAN; bt : INTEGER) : PChar;
FUNCTION AppendByte(B : BYTE) : PChar;
FUNCTION AppendCh(C : CHAR) : PChar;
FUNCTION AppendDIC(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AppendDouble(D : DOUBLE; Width,Places : BYTE) : PChar;
FUNCTION AppendDoubleTrim(D : DOUBLE) : PChar;
FUNCTION AppendExt(E : EXTENDED; Width,Places : BYTE) : PChar;
FUNCTION AppendExtTrim(E : EXTENDED) : PChar;
FUNCTION AppendLen(CONST Args : ARRAY OF CONST; Len : WORD) : PChar;
FUNCTION AppendLong(L : LONGINT) : PChar;
FUNCTION AppendMid(CONST Args : ARRAY OF CONST; Start,Count : WORD) : PChar;
FUNCTION AppendNL(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AppendPad(CONST Args : ARRAY OF CONST; Len : WORD; ACh : CHAR) : PChar;
FUNCTION AppendPtr(P : POINTER) : PChar;
FUNCTION AppendReal(R : REAL; Width,Places : BYTE) : PChar;
FUNCTION AppendRight(CONST Args : ARRAY OF CONST; Len : WORD) : PChar;
FUNCTION AppendSIC(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AppendTrim(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION AppendWithTab(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION NL : PChar;
FUNCTION NLAppend(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION Prepend(CONST Args : ARRAY OF CONST) : PChar;

TStringClass.Append

FUNCTION Append(CONST Args : ARRAY OF CONST) : Pchar;

Appends the string represented by 'Args' to the end of the object's own text string.

TStringClass.AppendBoolean

FUNCTION AppendBoolean(B : BOOLEAN; bt : INTEGER) : Pchar;

Converts the boolean value 'B' to a string of type 'bt' and then appends that string to the end of the object's existing text string. The parameter 'bt' can be any of the following values.

<u>'bt' constant</u>	<u>String conversion(FALSE/TRUE)</u>
bt_NoYes	No / Yes
bt_01	0 / 1
bt_FalseTrue	False / True
bt_FT	F / T
bt_NY	N / Y
bt_OffOn	Off / On

If the 'bt' is not one of the constant values shown above then no conversion or append is made.

TStringClass.AppendByte

FUNCTION AppendByte(B : BYTE) : Pchar;

Converts the BYTE parameter 'B' to a string and appends that string to the end of the object's existing text string value.

TStringClass.AppendCh

FUNCTION AppendCh(C : CHAR) : PChar;

Appends the CHAR 'C' to the end of the object's existing text string value.

TStringClass.AppendDIC

FUNCTION AppendDIC(CONST Args : ARRAY OF CONST) : Pchar;

The 'Args' parameter is converted to a string. That string is then enclosed by double inverted comma characters ("), and the enclosed string is then appended to the end of the object's own text string.

TStringClass.AppendDouble

FUNCTION AppendDouble(D : DOUBLE; Width,Places : BYTE) : Pchar;

Converts the DOUBLE parameter 'D' to a right justified string of 'Width' characters and 'Places' decimal precision. The resulting string is then appended that string to the end of the object's existing text string value.

TStringClass.AppendDoubleTrim

FUNCTION AppendDoubleTrim(D : DOUBLE) : PChar;

Converts the DOUBLE parameter 'D' to a decimal string with all leading spaces and trailing zero's removed. The resulting string is then appended to the end of the object's existing text string value.

TStringClass.AppendExt

FUNCTION AppendExt(E : EXTENDED; Width,Places : BYTE) : PChar;

Converts the EXTENDED parameter 'E' to a right justified string of 'Width' characters and 'Places' decimal precision. The resulting string is then appended to the end of the object's existing text string value.

TStringClass.AppendExtTrim

FUNCTION AppendExtTrim(E : EXTENDED) : Pchar;

Converts the EXTENDED parameter 'E' to a decimal string with all leading spaces and trailing zero's removed. The resulting string is then appended to the end of the object's existing text string value.

TStringClass.AppendLen

FUNCTION AppendLen(CONST Args : ARRAY OF CONST; Len : WORD) : Pchar;

The 'Args' parameter is converted to a string and the first 'Len' characters are appended to the end of the object's existing text string.

TStringClass.AppendLong

FUNCTION AppendLong(L : LONGINT) : PChar;

Converts the LONG parameter 'L' to string and appends it to the end of the object's existing text string value.

TStringClass.AppendMid

FUNCTION AppendMid(CONST Args : ARRAY OF CONST; Start, Count : WORD) : Pchar;

The 'Arg's parameter is converted to a string, and starting from the 'Start' character (base 0), 'Count' characters of the 'Args' string are appended to the end of the object's existing text string.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.AppendNL

FUNCTION AppendNL(CONST Args : ARRAY OF CONST) : Pchar;

Appends the string represented by 'Args' to the end of the object's own text string and then appends a further 'NewLine' character combination (#13#10) to the end of the object text string.

TStringClass.AppendPad

**FUNCTION AppendPad(CONST Args : ARRAY OF CONST;
Len : WORD;
ACh : CHAR) : Pchar;**

The 'Arg's parameter is converted to a string, and appended to the end of the object's existing text string. If the resulting object string is less than 'Len' characters long, then multiple 'ACh' characters are appended to the end of the object text string to create a string of length 'Len'.

TStringClass.AppendPtr

FUNCTION AppendPtr(P : POINTER) : Pchar;

Converts the POINTER parameter 'P' to a hexadecimal string format and appends it to the end of the object's existing text string value.

TStringClass.AppendReal

FUNCTION AppendReal(R : REAL; Width,Places : BYTE) : PChar;

Converts the REAL parameter 'R' to a right justified string of 'Width' characters and 'Places' decimal precision. The resulting string is then appended to the end of the object's existing text string value.

TStringClass.AppendRight

FUNCTION AppendRight(CONST Args : ARRAY OF CONST; Len : WORD) : Pchar;

The 'Args' parameter is converted to a string and the right most 'Len' characters are appended to the end of the object's existing text string.

TStringClass.AppendSIC

FUNCTION AppendSIC(CONST Args : ARRAY OF CONST) : Pchar;

The 'Args' parameter is converted to a string. That string is then enclosed by single inverted comma characters ('), and the enclosed string is then appended to the end of the object's own text string.

TStringClass.AppendTrim

FUNCTION AppendSIC(CONST Args : ARRAY OF CONST) : Pchar;

The 'Args' parameter is converted to a string. That string is then enclosed by single inverted comma characters ('), and the enclosed string is then appended to the end of the object's own text string.

TStringClass.AppendWithTab

FUNCTION AppendWithTab(CONST Args : ARRAY OF CONST) : PChar;

The 'Args' parameter is converted to a string and appended to the end of the object's own text string. A tab character (#9) is then appended to the end of the object text string.

TStringClass.NL

FUNCTION NL : PChar;

Appends a 'NewLine' character combination to the end of the object text string.

TStringClass.NLAppend

FUNCTION NLAppend(CONST Args : ARRAY OF CONST) : PChar;

Appends a 'NewLine' character combination to the object text string BEFORE appending the 'Args' string to the end of the object text string.

TStringClass.Prepend

FUNCTION Prepend(CONST Args : ARRAY OF CONST) : PChar;

The 'Args' parameter is converted to a string, and then inserted in front of any existing object text string.

TStringClass Character element related functions

```
FUNCTION FirstNonSpaceCh(VAR ACh : CHAR) : WORD;  
FUNCTION HasCh(ACh : CHAR; VAR Pos : WORD) : BOOLEAN;  
FUNCTION IsCh(W : WORD; ACh : CHAR) : BOOLEAN;  
FUNCTION IsFirstCh(ACh : CHAR) : BOOLEAN;  
FUNCTION IsLastCh(ACh : CHAR) : BOOLEAN;  
FUNCTION LastNonSpaceCh(VAR ACh : CHAR) : WORD;  
PROCEDURE RemoveLastCh;  
PROCEDURE SetCh(W : WORD; ACh : CHAR);
```

TStringClass.FirstNonSpaceCh

FUNCTION *FirstNonSpaceCh*(VAR Ch : CHAR) : WORD;

Returns the position (base 0) of the first non space character in the string. It also sets the 'Ch' parameter with the CHAR value at that location. If no non-space characters are found or the string is NIL the function returns a 'WORDRESERROR' result (equivalent to \$FFFF).

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.HasCh

FUNCTION HasCh(Ch : CHAR; VAR Pos : WORD) : BOOLEAN;

Returns TRUE if the object's string includes the 'Ch' character. It returns the position of the instance of that character in variable 'Pos' (base 0).

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.IsCh

FUNCTION IsCh(W : WORD; Ch : CHAR) : BOOLEAN;

Returns TRUE if the character in the object's string at position 'W' (base 0) is 'Ch'. The function returns FALSE if the string is NIL or 'W' exceeds the string length.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.IsFirstCh

FUNCTION IsFirstCh(Ch : CHAR) : BOOLEAN;

Returns TRUE if the first character in the object's string is 'Ch'. The function returns FALSE if the string is NIL.

TStringClass.IsLastCh

FUNCTION IsLastCh(Ch : CHAR) : BOOLEAN;

Returns TRUE if the last character in the object's string is 'Ch'. The function returns FALSE if the string is NIL.

TStringClass.LastNonSpaceCh

FUNCTION LastNonSpaceCh(VAR Ch : CHAR) : WORD;

Starting from the end of the object's string, this function returns the position (base 0) of the right-most non space character in the string. It also sets the 'Ch' parameter with the CHAR value at that location. If no non-space characters are found or the string is NIL the function returns a 'WORDRESERROR' result (equivalent to \$FFFF).

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.RemoveLastCh

PROCEDURE RemoveLastCh;

This will remove the last character from the object string.

TStringClass.SetCh

PROCEDURE *SetCh*(*W* : *WORD*; *Ch* : *CHAR*);

This will set the character at position 'W' (base 0) to 'Ch'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass : Other data type related functions

FUNCTION FromBoolean(B : BOOLEAN; bt : INTEGER) : PChar;
FUNCTION FromByte(B : BYTE) : PChar;
FUNCTION FromChar(C : CHAR) : PChar;
FUNCTION FromComponent(AComp : TWinControl) : PChar;
FUNCTION FromComponentItem(AComp : TWinControl; Idx : INTEGER) : PChar;
FUNCTION FromDouble(D : DOUBLE; Width,Places : BYTE) : PChar;
FUNCTION FromDoubleTrim(D : DOUBLE) : PChar;
FUNCTION FromExt(E : EXTENDED; Width,Places : BYTE) : PChar;
FUNCTION FromExtTrim(E : EXTENDED) : PChar;
FUNCTION FromLong(L : LONGINT) : PChar;
FUNCTION FromPtr(P : POINTER) : PChar;
FUNCTION FromReal(R : REAL; Width,Places : BYTE) : PChar;
FUNCTION FromRealTrim(R : REAL) : PChar;
FUNCTION FromRGB(C : TColorRef) : PChar;
FUNCTION HexFromByte(B : BYTE) : PChar;
FUNCTION HexFromLong(L : LONGINT) : PChar;
FUNCTION HexFromPtr(P : POINTER) : PChar;
FUNCTION HexFromWord(W : WORD) : PChar;
FUNCTION ToBoolean(VAR B : BOOLEAN) : BOOLEAN;
FUNCTION ToByte(VAR B : BYTE) : BOOLEAN;
FUNCTION ToChar(VAR C : CHAR) : BOOLEAN;
FUNCTION ToComponent(AComp : TWinControl) : BOOLEAN;
FUNCTION ToDouble(VAR D : DOUBLE) : BOOLEAN;
FUNCTION ToExt(VAR E : EXTENDED) : BOOLEAN;
FUNCTION ToInteger(VAR I : INTEGER) : BOOLEAN;
FUNCTION ToLong(VAR L : LONGINT) : BOOLEAN;
FUNCTION ToReal(VAR R : REAL) : BOOLEAN;
FUNCTION ToRGB(VAR C : TColorRef) : BOOLEAN;
FUNCTION ToWord(VAR W : WORD) : BOOLEAN;

TStringClass.FromBoolean

FUNCTION FromBoolean(B : BOOLEAN; bt : INTEGER) : PChar;

Converts the boolean value 'B' to a string of type 'bt' and then assigns that value to the object string. The parameter 'bt' can be any of the following values.

<u>'bt' constant</u>	<u>String conversion(FALSE/TRUE)</u>
bt_NoYes	No / Yes
bt_01	0 / 1
bt_FalseTrue	False / True
bt_FT	F / T
bt_NY	N / Y
bt_OffOn	Off / On

If the 'bt' is not one of the constant values shown above then no conversion or assignment is made.

TStringClass.FromByte

FUNCTION FromByte(B : BYTE) : PChar;

Converts the byte number 'B' to a string and assigns that string to the object string.

TStringClass.FromChar

FUNCTION FromChar(C : CHAR) : PChar;

Converts the CHAR 'Ch' to a string and assigns this to the object string.

TStringClass.FromComponent

FUNCTION FromComponent(AComp : TWinControl) : PChar;

Assigns to the object's own text buffer the text string associated with VCL component. This function will accept any VCL object that derives from 'TWinControl' and responds to the 'GetTextBuf' method.

TStringClass.FromComponentItem

FUNCTION FromComponentItem(AComp : TWinControl; Idx : INTEGER) : PChar;

Assigns to the object's own text buffer a text string taken from a line or item of the VCL component. For list boxes and combo boxes the 'Idx' parameter is equivalent to the item number. For memo controls it is equivalent to the line number.

Two special cases arise for the 'Idx' parameter.

```
All_Items      = -1;  
Selected_text = -2;
```

Use of the 'All_Items' parameter is equivalent to calling 'FromComponent' where the whole component text string is retrieved. Use of the 'selected_text' parameter only retrieves highlighted/selected text.

TStringClass.FromDouble

FUNCTION FromDouble(D : DOUBLE; Width,Places : BYTE) : PChar;

Converts the double number 'D' to a string of the designated 'Width' in characters and number of decimal places.

TStringClass.FromDoubleTrim

FUNCTION FromDoubleTrim(D : DOUBLE) : Pchar;

Converts the double number 'D' to a string with all leading white spaces and all trailing zeros removed.

TStringClass.FromExt

FUNCTION FromExt(E : EXTENDED; Width,Places : BYTE) : PChar;

Converts the 'E' number to a string of the declared 'Width' and 'Places' decimal places, and then assigns that string to object string.

TStringClass.FromExtTrim

FUNCTION FromExtTrim(E : EXTENDED) : PChar;

Converts the extended number 'E' to a string with all leading white spaces and all trailing zeros removed.

TStringClass.FromLong

FUNCTION FromLong(L : LONGINT) : PChar;

Converts the 'L' number to a string and assigns that string to the object string.

TStringClass.FromPtr

FUNCTION FromPtr(P : POINTER) : PChar;

Converts the pointer value 'P' to a string of type 'SEGMENT:OFFSET' and then assigns that string to the object.

TStringClass.FromReal

FUNCTION FromReal(R : REAL; Width,Places : BYTE) : PChar;

Converts the 'R' number to a string of the declared 'Width' and 'Places' decimal places, and then assigns that string to the object string.

TStringClass.FromRealTrim

FUNCTION FromRealTrim(R : REAL) : PChar;

Converts the REAL number 'R' to a string with all leading white spaces and all trailing zeros removed.

TStringClass.FromRGB

FUNCTION FromRGB(C : TColorRef) : PChar;

Converts the RGB colour value into a comma delimited string of type 'RED, GREEN, BLUE', and then assigns that string to the object. (This is useful for writing values back to an INI file).

TStringClass.HexFromByte

FUNCTION HexFromByte(B : BYTE) : PChar;

Converts the byte value 'B' into a hexadecimal string and assigns that string to the object.

TStringClass.HexFromLong

FUNCTION HexFromLong(L : LONGINT) : PChar;

Converts the 'L' number into a hexadecimal string of type 'nnnn:nnnn' and assigns that string to the object.

TStringClass.HexFromPtr

FUNCTION HexFromPtr(P : POINTER) : PChar;

Converts the pointer 'P' into a hexadecimal string of type 'nnnn:nnnn' and assigns that string to the object.

TStringClass.HexFromWord

FUNCTION HexFromWord(W : WORD) : PChar;

Converts the 'W' number into a hexadecimal string of type 'nnnn' and assigns that string to the object.

TStringClass.ToBoolean

FUNCTION ToBoolean(VAR B : BOOLEAN) : BOOLEAN;

Returns TRUE if the object string can be converted into a BOOLEAN type. The variable parameter 'B' is set with the converted value. The conversion is tested against the range of boolean to string conversions permitted by the 'bt_xxx' constants.

TStringClass.ToByte

FUNCTION ToByte(VAR B : BYTE) : BOOLEAN;

Returns TRUE if the object string can be converted into a BYTE number type. The variable parameter 'B' is set with the converted value.

TStringClass.ToChar

FUNCTION ToChar(VAR C : CHAR) : BOOLEAN;

Returns TRUE if the string has at least one character, and assigns the first character of the string into the parameter 'C'.

TStringClass.ToComponent

FUNCTION ToComponent(AComp : TWinControl) : BOOLEAN;

This assigns the object's own text string to the VCL component. The process will return TRUE if the assignment is completed. This method will work with any descendant of the 'TWinControl' class that implements a 'SetTextBuf' method.

TStringClass.ToDouble

FUNCTION ToDouble(VAR D : DOUBLE) : BOOLEAN;

Returns TRUE if the object string can be converted into a DOUBLE number type. The variable parameter 'D' is set with the converted value.

TStringClass.ToExt

FUNCTION ToExt(VAR E : EXTENDED) : BOOLEAN;

Returns TRUE if the object string can be converted into an EXTENDED number type. The variable parameter 'E' is set with the converted value.

TStringClass.ToInteger

FUNCTION ToInteger(VAR I : INTEGER) : BOOLEAN;

Returns TRUE if the object string can be converted into an INTEGER number type. The variable parameter 'I' is set with the converted value. If the object string includes a decimal place dot the conversion will fail.

TStringClass.ToLong

FUNCTION ToLong(VAR L : LONGINT) : BOOLEAN;

Returns TRUE if the object string can be converted into an LONGINT number type. The variable parameter 'L' is set with the converted value. If the object string includes a decimal place dot the conversion will fail.

TStringClass.ToReal

FUNCTION ToReal(VAR R : REAL) : BOOLEAN;

Returns TRUE if the object string can be converted into an REAL number type. The variable parameter 'R' is set with the converted value.

TStringClass.ToRGB

FUNCTION ToRGB(VAR C : TColorRef) : BOOLEAN;

Returns TRUE if the object string can be converted from a 'RED, GREEN, BLUE' delimited string type into a valid TColorRef value. This is useful for reading INI file values. (Refer to the 'FromRGB' function).

TStringClass.ToWord

FUNCTION ToWord(VAR W : WORD) : BOOLEAN;

Returns TRUE if the object string can be converted from a text string type into a valid WORD value. The variable parameter 'W' is set with the converted value.

TStringClass SysUtils unit compatible functions

FUNCTION AnsiCompareStr(CONST S2: STRING): Integer;
FUNCTION AnsiCompareText(CONST S2: STRING): Integer;
FUNCTION AnsiLowerCase(CONST S : STRING) : PChar;
FUNCTION AnsiUpperCase(CONST S : STRING) : PChar;
FUNCTION AppendStr(CONST S: string) : PChar;
FUNCTION CompareStr(CONST S2: STRING): Integer;
FUNCTION CompareText(CONST S2: STRING): Integer;
FUNCTION FloatToStr(Value: Extended): PChar;
FUNCTION FloatToStrF(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): PChar;
FUNCTION FmtLoadStr(Ident: Word; CONST Args: ARRAY OF CONST): PChar;
FUNCTION Format(CONST Format: STRING; CONST Args: ARRAY OF CONST): PChar;
FUNCTION FormatFloat(const Format: STRING; Value: Extended): PChar;
FUNCTION IntToHex(Value: Longint; Digits: Integer): PChar;
FUNCTION IntToStr(Value: Longint): PChar;
FUNCTION IsValidIdent: Boolean;
FUNCTION LoadStr(Ident: Word): PChar;
FUNCTION LowerCase(const S: string): PChar;
FUNCTION StrToFloat : Extended;
FUNCTION StrToInt : Longint;
FUNCTION StrToIntDef(Default: Longint): Longint;
FUNCTION UpperCase(CONST S: string) : PChar;

TStringClass.AppendStr

FUNCTION AppendStr(CONST S: string) : Pchar;
Appends the 'S' string to the end of the object's own text buffer.

TStringClass.UpperCase

FUNCTION UpperCase(CONST S: string) : Pchar;

Converts the 'S' parameter to upper case and assigns to the object text buffer, replacing any existing text string value.

TStringClass.LowerCase

FUNCTION LowerCase(const S: string): PChar;

Converts the 'S' parameter to lower case and assigns to the object text buffer, replacing any existing text string value.

TStringClass.CompareStr

FUNCTION CompareStr(CONST S2: STRING): Integer;

A case sensitive comparison between the object's own text string and the parameter 'S2' string. This is equivalent to the 'Compare' function.

TStringClass.CompareText

FUNCTION CompareText(CONST S2: STRING): Integer;

A comparison between the object's own text string and the parameter 'S2' string, which ignores case. This is equivalent to the 'CompareI' function.

TStringClass.AnsiUpperCase

FUNCTION AnsiUpperCase(CONST S : STRING) : Pchar;

AnsiUpperCase converts all characters in the given string 'S' to upper case and assigns the result to the object text buffer. The conversion uses the currently installed language driver.

TStringClass.AnsiLowerCase

FUNCTION AnsiLowerCase(CONST S : STRING) : PChar;

AnsiUpperCase converts all characters in the given string 'S' to lower case and assigns the result to the object text buffer. The conversion uses the currently installed language driver.

TStringClass.AnsiCompareStr

FUNCTION AnsiCompareStr(CONST S2: STRING): Integer;

AnsiCompareStr compares the object's own text string to S2, with case-sensitivity. The compare operation is controlled by the currently installed language driver. The return value is the same as for CompareStr.

TStringClass.AnsiCompareText

FUNCTION AnsiCompareText(CONST S2: STRING): Integer;

AnsiCompareText compares the object's text string to S2, without case-sensitivity. The compare operation is controlled by the currently installed language driver. The return value is the same as for CompareText.

TStringClass.IsValidIdent

FUNCTION IsValidIdent: Boolean;

IsValidIdent returns true if the object string is a valid identifier. An identifier is defined as a character from the set ['A'..'Z', 'a'..'z', '_'] followed by zero or more characters from the set ['A'..'Z', 'a'..'z', '0'..'9', '_'].

TStringClass.IntToStr

FUNCTION IntToStr(Value: Longint): Pchar;

Converts the LONGINT 'Value' into a text string and assigns it to the object text buffer, replacing any existing text value.

TStringClass.IntToHex

FUNCTION IntToHex(Value: Longint; Digits: Integer): Pchar;

The IntToHex function converts a number into a string containing the number's hexadecimal (base 16) representation with a specific number of digits; and assigns the resultant string to the object text buffer.

TStringClass.StrToInt

FUNCTION StrToInt : Longint;

The StrToInt function converts the object text string representing an integer-type number in either decimal or hexadecimal notation into a number. If the string does not represent a valid number, StrToInt raises an EConvertError exception. Use of the alternative 'ToLong' function is recommended.

TStringClass.StrToIntDef

FUNCTION StrToIntDef(Default: Longint): Longint;

The StrToIntDef function converts the object text string into a number. If S does not represent a valid number, StrToIntDef returns the number passed in Default.

TStringClass.LoadStr

FUNCTION LoadStr(Ident: Word): Pchar;

LoadStr loads the string resource given by Ident from the application's executable file into the object's text buffer. If the string resource does not exist, an empty string is returned.

TStringClass.FmtLoadStr

FUNCTION FmtLoadStr(Ident: Word; CONST Args: ARRAY OF CONST): Pchar;

FmtLoadStr loads a string from a program's resource string table and uses that string, plus the args array, as a parameter to Format. Ident is the string resource ID of the desired format string. Result is the output of Format. The resulting formatted text string is assigned to the object text buffer.

TStringClass.Format

FUNCTION Format(CONST Format: STRING; CONST Args: ARRAY OF CONST): Pchar;

This function formats the series of arguments in the open array 'Args'. Formatting is controlled by the 'Format' parameter; the results are assigned to the object's text buffer.

TStringClass.FloatToStr

FUNCTION FloatToStr(Value: Extended): Pchar;

FloatToStr converts the floating-point value given by Value to its string representation, and assigns that string to the object text buffer. The conversion uses general number format with 15 significant digits.

TStringClass.FloatToStrF

FUNCTION *FloatToStrF*(*Value: Extended;*
Format: TFloatFormat;
Precision, Digits: Integer): Pchar;

FloatToStrF converts the floating-point value given by Value to its string representation, and assigns that string to the object buffer.

TStringClass.FormatFloat

FUNCTION FormatFloat(const Format: STRING; Value: Extended): Pchar;

FormatFloat formats the floating-point value given by Value using the format string given by Format, and assigns the resulting string to the object text buffer.

TStringClass.StrToFloat

FUNCTION StrToFloat : Extended;

StrToFloat converts the object text string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The DecimalSeparator global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is raised.

TStringClass : Strings unit compatible functions

FUNCTION StrCat(Source : PChar) : PChar;
FUNCTION StrComp(Str2 : PChar) : INTEGER;
FUNCTION StrCopy(Source : PChar) : PChar;
FUNCTION StrECopy(Source : PChar) : PChar;
FUNCTION StrEnd : PChar;
FUNCTION StrIComp(Str2 : PChar) : INTEGER;
FUNCTION StrLCat(Source : PChar; MaxLen : WORD) : PChar;
FUNCTION StrLIComp(Str2 : PChar; MaxLen : WORD) : INTEGER;
FUNCTION StrLComp(Str2 : PChar; MaxLen : WORD) : INTEGER;
FUNCTION StrLCopy(Str2 : PChar; MaxLen : WORD) : INTEGER;
FUNCTION StrLen : WORD;
FUNCTION StrLower : PChar;
FUNCTION StrMove(Source : PChar; Count : WORD) : PChar;
FUNCTION StrPas : STRING;
FUNCTION StrPCopy(Source : STRING) : PChar;
FUNCTION StrPos(Str2 : PChar) : PChar;
FUNCTION StrRScan(Chr : CHAR) : PChar;
FUNCTION StrScan(Chr : CHAR) : PChar;
FUNCTION StrUpper : PChar;

TStringClass.StrCat

FUNCTION StrCat(Source : PChar) : PChar;

Appends the string 'Source; to the end of the object's own string (equivalent to the 'Append' method).

TStringClass.StrComp

FUNCTION StrComp(Str2 : PChar) : INTEGER;

Compares the object's string with another string 'Str2'. The comparison is case sensitive. (Equivalent to the 'Compare' method).

TStringClass.StrCopy

FUNCTION StrCopy(Source : PChar) : PChar;

Copies the 'Source' string into the object string (equivalent to the 'Assign' method).

TStringClass.StrECopy

FUNCTION StrECopy(Source : PChar) : PChar;

Copies the 'Source' string into the object string and returns a PChar pointer to the end of the resulting string.

TStringClass.StrEnd

FUNCTION StrEnd : PChar;

Returns a PChar pointer to the end of the object string (i.e. the \0 NULL terminator character position).

TStringClass.StrlComp

FUNCTION StrlComp(Str2 : PChar) : INTEGER;

Compares the object string with the parameter string 'Str2', ignoring any case differences. It returns a '-1', '0' or '1' result as defined in the 'Compare_xxx' constants. (Equivalent to the 'CompareI' method).

TStringClass.StrLCat

FUNCTION StrLCat(Source : PChar; MaxLen : WORD) : PChar;

Appends the first 'MaxLen' characters of parameter 'Source' to the end of the object's own string (equivalent to the 'AppendLen' method).

TStringClass.StrLIComp

FUNCTION StrLIComp(Str2 : PChar; MaxLen : WORD) : INTEGER;

Compares the first 'Maxlen' characters of the object string with the string 'Str2', ignoring case. The function returns one of the 'Compare_xxx' constants. (Equivalent to the 'CompareLI' method).

TStringClass.StrLComp

FUNCTION StrLComp(Str2 : PChar; MaxLen : WORD) : INTEGER;

Compares the first 'Maxlen' characters of the object string with the string 'Str2', including case. The function returns one of the 'Compare_xxx' constants. (Equivalent to the 'CompareL' method).

TStringClass.StrLCopy

FUNCTION StrLCopy(Str2 : PChar; MaxLen : WORD) : INTEGER;

Copies the first 'MaxLen' characters of 'Str2' into the object string (equivalent to the 'AssignLen' method).

TStringClass.StrLen

FUNCTION StrLen : WORD;

Returns the length of the object string (equivalent to the 'GetLength' method).

TStringClass.StrLower

FUNCTION StrLower : PChar;

Converts the object string to lower case (equivalent to the 'ToLower' method).

TStringClass.StrMove

FUNCTION StrMove(Source : PChar; Count : WORD) : PChar;
Copies the first 'Count' characters of 'Source' into the string object.

TStringClass.StrPas

FUNCTION StrPas : STRING;

Returns the object PChar as a Pascal STRING variable.

TStringClass.StrPCopy

FUNCTION StrPCopy(Source : STRING) : PChar;
Copies a Pascal STRING type into the object PChar string.

TStringClass.StrPos

FUNCTION StrPos(Str2 : PChar) : PChar;

Returns the PChar position of sub string 'Str2' within the object's own string. The function returns NIL if no substring is found. (refer to the 'FindFirst' method for similar functionality).

TStringClass.StrRScan

FUNCTION StrRScan(Chr : CHAR) : PChar;

Returns a PChar pointer to the right most location of the 'Chr' CHAR in the object string.

TStringClass.StrScan

FUNCTION StrScan(Chr : CHAR) : PChar;

Returns a PChar pointer to the first location (from the left side) of the 'Chr' CHAR in the object string.

TStringClass.StrUpper

FUNCTION StrUpper : PChar;

Converts the object string to upper case (equivalent to the 'ToUpper' method).

TStringClass : Comparison related functions

FUNCTION Compare(CONST Args : ARRAY OF CONST) : INTEGER;
FUNCTION CompareI(CONST Args : ARRAY OF CONST) : INTEGER;
FUNCTION CompareL(CONST Args : ARRAY OF CONST; Len : WORD) : INTEGER;
FUNCTION CompareLI(CONST Args : ARRAY OF CONST; Len : WORD) : INTEGER;
FUNCTION CompareLong(L : LONGINT) : INTEGER;
FUNCTION CompareDouble(D : DOUBLE) : INTEGER;
FUNCTION CompareExt(E : EXTENDED) : INTEGER;
FUNCTION IsSame(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION IsSameI(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION IsSameL(CONST Args : ARRAY OF CONST; Len : WORD) : BOOLEAN;
FUNCTION IsSameLI(CONST Args : ARRAY OF CONST; Len : WORD) : BOOLEAN;
FUNCTION Includes(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION Within(CONST Args : ARRAY OF CONST) : BOOLEAN;

TStringClass.Compare

FUNCTION Compare(CONST Args : ARRAY OF CONST) : INTEGER;

Compares the object string to the other string defined in 'Args' and returns an integer comparison result of type 'Compare_xx'. The test is case sensitive.

TStringClass.CompareI

FUNCTION CompareI(CONST Args : ARRAY OF CONST) : INTEGER;

Compares the object string to the other string defined in 'Args' and returns an integer comparison result of type 'Compare_xx'. The test ignores case.

TStringClass.CompareL

FUNCTION CompareL(CONST Args : ARRAY OF CONST; Len : WORD) : INTEGER;

Compares the first 'Len' characters of the object string with the same set of characters from the 'Args' string. The test is case sensitive.

TStringClass.CompareLI

FUNCTION CompareLI(CONST Args : ARRAY OF CONST; Len : WORD) : INTEGER;

Compares the first 'Len' characters of the object string with the same set of characters from the 'Args' string. The test ignores case.

TStringClass.CompareLong

FUNCTION CompareLong(L : LONGINT) : INTEGER;

The function attempts to convert the object string to a LONGINT number type and then compares the converted number to the 'L' number, returning one of the 'Compare_xx' type results. If the string cannot be converted to a LONGINT the function returns a 'Compare_LT' result.

TStringClass.CompareDouble

FUNCTION CompareDouble(D : DOUBLE) : INTEGER;

The function attempts to convert the object string to a DOUBLE number type and then compares the converted number to the 'D' number, returning one of the 'Compare_xx' type results. If the string cannot be converted to a double the function returns a 'Compare_LT' result.

TStringClass.CompareExt

FUNCTION CompareExt(E : EXTENDED) : INTEGER;

The function attempts to convert the object string to a EXTENDED number type and then compares the converted number to the 'E' number, returning one of the 'Compare_xx' type results. If the string cannot be converted to a double the function returns a 'Compare_LT' result.

TStringClass.IsSame

FUNCTION IsSame(CONST Args : ARRAY OF CONST) : BOOLEAN;

Returns TRUE if the object string is the same as the 'Args' string. The test is case sensitive. This is equivalent to the command:

```
IsSame := (Compare(Other) = Compare_EQ);
```

TStringClass.IsSameI

FUNCTION IsSameI(Other : PChar) : BOOLEAN;

Returns TRUE if the object string is the same as the 'Args' string. The test ignores case. This is equivalent to the command:

```
IsSameI := (CompareI(Other) = Compare_EQ);
```

TStringClass.IsSameL

FUNCTION IsSameL(CONST Args : ARRAY OF CONST; Len : WORD) : BOOLEAN;

Returns TRUE if the first 'Len' characters of the object string are the same as the 'Args' string. The test is case sensitive. This is equivalent to the command:

IsSameL := (CompareL(Other,Len) = Compare_EQ);

TStringClass.IsSameLI

FUNCTION IsSameLI(CONST Args : ARRAY OF CONST; Len : WORD) : BOOLEAN;

Returns TRUE if the first 'Len' characters of the object string are the same as the 'Args' string. The test ignores case. This is equivalent to the command:

```
IsSameLI := (CompareLI(Other,Len) = Compare_EQ);
```

TStringClass.Includes

FUNCTION Includes(CONST Args : ARRAY OF CONST) : BOOLEAN;

Returns TRUE if the 'Args' text string is to be found within the object's own text string. The comparison is case sensitive.

TStringClass.Within

FUNCTION Within(CONST Args : ARRAY OF CONST) : BOOLEAN;

Returns TRUE if the object's own text string is found within the 'Args' text string. The comparison is case sensitive.

TStringClass : Insert/Delete/Trim related functions

FUNCTION AddDIC : PChar;
FUNCTION AddSIC : PChar;
FUNCTION Delete(Index,Count : WORD) : PChar;
FUNCTION DeleteFrom(Index : WORD) : PChar;
FUNCTION Insert(CONST Args : ARRAY OF CONST; Index : WORD) : PChar;
FUNCTION InsertL(CONST Args : ARRAY OF CONST; Len,Index : WORD) : PChar;
FUNCTION PadCentre(NewLen : WORD; ACh : CHAR) : PChar;
FUNCTION PadEnd(NewLen : WORD; ACh : CHAR) : PChar;
FUNCTION PadFront(NewLen : WORD; ACh : CHAR) : PChar;
FUNCTION RemoveDIC : PChar;
FUNCTION RemoveSIC : PChar;
FUNCTION Trim : PChar;
FUNCTION TrimEnd : PChar;
FUNCTION TrimFront : PChar;
FUNCTION TrimZero : PChar;

TStringClass.AddDIC

FUNCTION AddDIC : PChar;

Inserts a double inverted comma (") character at the start and end of the object string. The insert at either end is only made where the character is not already there.

TStringClass.AddSIC

FUNCTION AddDIC : PChar;

Inserts a single inverted comma (') character at the start and end of the object string. The insert at either end is only made where the character is not already there.

TStringClass.Delete

FUNCTION Delete(Index, Count : WORD) : PChar;

Deletes from the object string characters starting from position 'Index' (base 0) for 'Count' number of characters. Example:

```
Str1.Assign('This is a test')
Str1.Delete(5,5);
WriteLn(Str1.ZString);    {This test}
```

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.DeleteFrom

FUNCTION DeleteFrom(Index : WORD) : PChar;

Deletes from the object string all characters starting from position 'Index' (base 0) for. Example:

```
Str1.Assign('This is a test')
Str1.DeleteFrom(3);
WriteLn(Str1.ZString);    { Thi }
```

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.Insert

FUNCTION *Insert*(**CONST** *Args* : **ARRAY OF CONST**; *Index* : **WORD**) : **PChar**;

Inserts string 'Args' into the object string at position 'Index' (base 0).

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.InsertL

FUNCTION *InsertL*(**CONST** *Args* : **ARRAY OF CONST**; *Len,Index* : **WORD**) : **PChar**;

Inserts 'Len' characters from string 'Args' into the object string at position 'Index' (base 0).

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.PadCentre

FUNCTION PadCentre(NewLen : WORD; Ch : CHAR) : PChar;

Expands the object string to a new length of 'NewLen' characters. If the old object string was less than 'NewLen' long the string is padded equally at both front and end with the 'Ch' character to create a string of the required overall length.

TStringClass.PadEnd

FUNCTION PadEnd(NewLen : WORD; Ch : CHAR) : PChar;

Expands the object string to a new length of 'NewLen' characters. If the old object string was less than 'NewLen' long the string is padded at its end with the 'Ch' character to create a string of the required overall length.

TStringClass.PadFront

FUNCTION PadFront(NewLen : WORD; Ch : CHAR) : PChar;

Expands the object string to a new length of 'NewLen' characters. If the old object string was less than 'NewLen' long the string is padded at its front with the 'Ch' character to create a string of the required overall length.

TStringClass.RemoveDIC

FUNCTION RemoveDIC : PChar;

If the object string has double inverted commas (") at the start and end of its string these are removed from both ends.

TStringClass.RemoveSIC

FUNCTION RemoveSIC : PChar;

If the object string has single inverted commas (') at the start and end of its string these are removed from both ends.

TStringClass.Trim

FUNCTION Trim : PChar;

Removes all leading and trailing spaces from the object string.

TStringClass.TrimEnd

FUNCTION TrimEnd : PChar;

Removes all trailing spaces from the object string.

TStringClass.TrimFront

FUNCTION TrimFront : PChar;

Removes all leading spaces from the object string.

TStringClass.TrimZero

FUNCTION TrimZero : PChar;

Removes all trailing '0' zero characters from the object string. (Useful for removing redundant zeros from real number strings). If the last character in the just trimmed string is '.' (decimal dot) then that is also removed.

TStringClass : Command Line related functions

FUNCTION FindCmdLine : PChar;

FUNCTION FindCmdLineAndParse(IncExeParam : BOOLEAN;
VAR AList : TObjectContainer) : PChar;

FUNCTION FindCmdLineParam(Idx : INTEGER) : PChar;

TStringClass.FindCmdLine

FUNCTION FindCmdLine : Pchar;

Recreates the current application's command line (both EXE and all parameters) as a string and assigns it to the object text buffer.

TStringClass.FindCmdLineAndParse

**FUNCTION FindCmdLineAndParse(IncExeParam : BOOLEAN;
VAR AList : TObjectContainer) : Pchar;**

Recreates the current application's command line (both EXE and all parameters) as a string and assigns it to the object text buffer. The component parts of the command line are then assigned to new string objects which are added to the 'AList' container object (which owns those new string objects). If 'IncExeParam' is TRUE then the index 0 command line parameter is included on the container list.

TStringClass.FindCmdLineParam

FUNCTION FindCmdLineParam(Idx : **INTEGER**) : **Pchar**;

Retrieves a component part of an application's command line parameter and assigns it to the object text buffer. An Idx of '0' retrieves the application EXE path.

TStringClass : Resource string related functions

FUNCTION AppendStringRes(Instance : THandle; Id : WORD) : PChar;
FUNCTION LoadStringRes(Instance : THandle; Id : WORD) : PChar;

TStringClass.AppendStringRes

FUNCTION AppendStringRes(Instance : THandle; Id : WORD) : Pchar;

Loads the string resource 'Id' and appends it to the end of the existing object text string.

TStringClass.LoadStringRes

FUNCTION LoadStringRes(Instance : THandle; Id : WORD) : Pchar;

Assigns to the object the string loaded from the task resources that has an identifier of 'Id'.

TStringClass : INI file related functions

```
FUNCTION FindIniSectionKeywords(CONST IniFileArgs,  
                                SectionArgs : ARRAY OF CONST;  
                                VAR  AList : TObjectContainer) : WORD;  
FUNCTION ReadIniKeyword(CONST IniFileArgs : ARRAY OF CONST;  
                          CONST SectionArgs : ARRAY OF CONST;  
                          CONST KeyWordArgs : ARRAY OF CONST) : WORD;  
FUNCTION WriteIniKeyword(CONST IniFileArgs : ARRAY OF CONST;  
                           CONST SectionArgs : ARRAY OF CONST;  
                           CONST KeyWordArgs : ARRAY OF CONST) : WORD;
```

TStringClass.FindIniSectionKeywords

**FUNCTION FindIniSectionKeywords(CONST IniFileArgs, SectionArgs : ARRAY OF CONST;
VAR AList : TObjectContainer) : WORD;**

Loads into the object's own buffer the text string equivalent to all keywords found in the 'SectionArgs' section of the 'IniFileArgs' INI file. The keywords are then parsed and placed as individual TStringClass instances on the container object 'Alist'. The list owns the newly created string objects. The functions returns the number of keywords found.

TStringClass.ReadIniKeyword

***FUNCTION ReadIniKeyword(CONST IniFileArgs : ARRAY OF CONST;
CONST SectionArgs : ARRAY OF CONST;
CONST KeyWordArgs : ARRAY OF CONST) : WORD;***

Assigns to the object string the INI file keyword value associated with the keyword, section and INI file name 'xxxxArgs' parameters.

TStringClass.WritelniKeyword

***FUNCTION WritelniKeyword(CONST IniFileArgs : ARRAY OF CONST;
CONST SectionArgs : ARRAY OF CONST;
CONST KeyWordArgs : ARRAY OF CONST) : WORD;***

Writes to the INI file the object string associated with the keyword, section and INI file name 'xxxArgs' parameters.

TStringClass : DOS path/filename related functions

FUNCTION AddBackSlash : PChar;
FUNCTION AddFilterDesc(CONST DescArgs,ExtArgs : ARRAY OF CONST) : PChar;
FUNCTION BuildPathName(CONST DirArgs,
 FileNameArgs,
 ExtArgs : ARRAY OF CONST) : PChar;
FUNCTION CreateDirectory : BOOLEAN;
FUNCTION DefaultExtension(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION DirectoryExists : BOOLEAN;
FUNCTION DriveExists : BOOLEAN;
FUNCTION ExpandFileName : PChar;
FUNCTION FileExists : BOOLEAN;
FUNCTION FileSplit(VAR ADirObj,ANameObj,AnExtObj : TStringClass) : WORD;
FUNCTION FindCurrentDir : PChar;
FUNCTION FindRelPath(CONST StartDirArgs,EndDirArgs : ARRAY OF CONST) :
PChar;
FUNCTION ForceExtension(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION GetSystemDirectory : PChar;
FUNCTION GetWindowsDirectory : PChar;
FUNCTION HasBackSlash : BOOLEAN;
FUNCTION HasDrive : BOOLEAN;
FUNCTION HasExtension(VAR DotPos : WORD) : BOOLEAN;
FUNCTION HasFileName : BOOLEAN;
FUNCTION HasDirectory : BOOLEAN;
FUNCTION JustDirectory(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION JustExtension(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION JustFileName(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION JustName(CONST Args : ARRAY OF CONST) : PChar;
FUNCTION RemoveDirectory : BOOLEAN;
FUNCTION SameDrive(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameDirectory(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameExtension(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameFileName(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SameName(CONST Args : ARRAY OF CONST) : BOOLEAN;
FUNCTION SetCurDir : BOOLEAN;

TStringClass.AddBackSlash

FUNCTION AddBackSlash : PChar;

Adds a '\' back slash to the end of the object string.

TStringClass.BuildPathName

***FUNCTION BuildPathName(CONST DirArgs,
 FileNameArgs,
 ExtArgs : ARRAY OF CONST) : PChar;***

Builds a validated path name from the three components.

TStringClass.CreateDirectory

FUNCTION CreateDirectory : BOOLEAN;

This function will attempt to create the DOS directory of the directory part of the object's own text string. The text string can hold a full file name (i.e. 'C:\test\test.txt'). The process will extract the related directory path (without affecting the object string) and try to create that directory. The function returns TRUE if the directory was successfully created.

TStringClass.DefaultExtension

FUNCTION DefaultExtension(CONST Args : ARRAY OF CONST) : Pchar;

If the object string has no file name extension the extension 'Args' is appended to the end of the object string. The 'Args' parameter can have a '.' prefix. The method will check for duplicate '.' entries.

TStringClass.DirectoryExists

FUNCTION DirectoryExists : BOOLEAN;

Returns TRUE if the directory component of the object string value exists. The object string can be a full path/file name. This method will extract the directory component (without affecting the object string) and test for the directory's existence.

TStringClass.DriveExists

FUNCTION DriveExists : BOOLEAN;

Returns TRUE if the drive letter component of the object string value exists. The object string can be a full path/file name. This method will extract the drive letter component (without affecting the object string) and test for the drive's existence.

TStringClass.ExpandFileName

FUNCTION ExpandFileName : Pchar;

Converts a DOS path file into a full drive, path and file name, removing any '..' type re-directions.

TStringClass.FileExists

FUNCTION FileExists : BOOLEAN;

Returns TRUE if the object string representing a DOS path name exists as a DOS file.

TStringClass.FindCurrentDir

FUNCTION FindCurrentDir : PChar;

Copies the current DOS directory path name into the object text string.

TStringClass.FileSplit

FUNCTION FileSplit(VAR ADirObj, ANameObj, AnExtObj : TStringClass) : WORD;

Takes the object's own text string and splits it into the three DOS parts of directory/path, filename (excluding extension) and extension (excluding the '.' dot). The three resulting sub strings are assigned to the three TStringClass parameters. These TStringClass objects must be instantiated prior to calling this function. The function returns a WORD value that indicates the presence of each sub string. This WORD value is composed from the 'fs_xxx' constants :

```
fs_Directory = 1;  
fs_Name      = 2;  
fs_extension = 4;
```

TStringClass.FindRelPath

FUNCTION FindRelPath(CONST StartDirArgs,EndDirArgs : ARRAY OF CONST) : PChar;

Builds a DOS path string that allows the relative movement from the 'StartDirArgs' directory to the end 'EndDirArgs' directory. If both args are the same or the drives are different a NULL string is assigned to the object. For example :

```
TObj := TStringClass.Create;  
TObj.FindRelPath(['c:\windows'],['c:\test\data']); { '..\test\data' }
```

TStringClass.ForceExtension

FUNCTION ForceExtension(CONST Args : ARRAY OF CONST) : Pchar;
Force the object string to have the file name extension 'Args'.

TStringClass.GetSystemDirectory

FUNCTION GetSystemDirectory : PChar;

Assigns the Windows system directory path to the object text string.

TStringClass.GetWindowsDirectory

FUNCTION GetWindowsDirectory : PChar;
Assigns the Windows directory path to the object text string.

TStringClass.HasBackSlash

FUNCTION HasBackSlash : BOOLEAN;

Returns TRUE if the last character of the object string is a '\' back slash.

TStringClass.HasDrive

FUNCTION HasDrive : BOOLEAN;

Returns TRUE if the object string is at least three characters long and the first letter is between 'A' and 'Z'.

TStringClass.HasExtension

FUNCTION HasExtension(VAR DotPos : WORD) : BOOLEAN;

Returns TRUE if the object string includes an extension, setting the 'DotPos' variable parameter with the base 0 position of the '.' delimiter.

TStringClass.HasFileName

FUNCTION HasFileName : BOOLEAN;

Returns TRUE if the object string has a file name component.

TStringClass.HasDirectory

FUNCTION HasDirectory : BOOLEAN;

Returns TRUE if the object string has a directory component.

TStringClass.JustExtension

FUNCTION JustExtension(CONST Args : ARRAY OF CONST) : PChar;

Extracts from the 'Args' parameter the extension component and assigns this to the object string.

TStringClass.JustFileName

FUNCTION JustFileName(CONST Args : ARRAY OF CONST) : Pchar;

Extracts from the 'Args' parameter the DOS '8.3' style file name component and assigns this to the object string.

TStringClass.JustName

FUNCTION JustName(CONST Args : ARRAY OF CONST) : Pchar;

Extracts from the 'Args' parameter the file name component (excluding any '.' and extension) and assigns this to the object string.

TStringClass.JustDirectory

FUNCTION JustDirectory(CONST Args : ARRAY OF CONST) : Pchar;

Extracts from the 'Args' parameter the directory component and assigns this to the object string.

TStringClass.RemoveDirectory

FUNCTION RemoveDirectory : BOOLEAN;

This function will attempt to delete a DOS directory of the directory part of the object's own text string. The text string can hold a full file name (i.e. 'C:\test\test.txt'). The process will extract the related directory path (without affecting the object string) and try to delete that directory. The function returns TRUE if the directory was successfully deleted.

TStringClass.SameDrive

FUNCTION SameDrive(CONST Args : ARRAY OF CONST) : BOOLEAN;

Compares the object's own string to the string in 'Arg's and returns TRUE if they share the same disk drive letter. This test process presumes that each entry is at least three characters long (i.e. 'C:\')

TStringClass.SameDirectory

FUNCTION SameDirectory(CONST Args : ARRAY OF CONST) : BOOLEAN;

Compares the object's own string to the string in 'Arg's and returns TRUE if they share the same DOS directory. Example:

```
AStrObj := TStringClass.CreateString(['c:\test\company.dat']);  
BStrObj := TStringClass.CreateString(['c:\test\employee.dat']);  
AStrObj.SameDirectory([BStrObj]); { returns TRUE }
```

TStringClass.SameExtension

FUNCTION SameExtension(CONST Args : ARRAY OF CONST) : BOOLEAN;

Compares the object's own string to the string in 'Arg's and returns TRUE if they share the same DOS file name extension.

TStringClass.SameFileName

FUNCTION SameFileName(CONST Args : ARRAY OF CONST) : BOOLEAN;

Compares the object's own string to the string in 'Arg's and returns TRUE if they share the same DOS '8.3' file name.

TStringClass.SameName

FUNCTION SameName(CONST Args : ARRAY OF CONST) : BOOLEAN;

Compares the object's own string to the string in 'Arg's and returns TRUE if they share the same DOS '8' file name. Example :

```
AStrObj := TStringClass.CreateString(['c:\data\company.dat']);  
BStrObj := TStringClass.CreateString(['c:\test\company.idx']);  
AStrObj.SameName([BStrObj]); { returns TRUE }
```

TStringClass.SetCurDir

FUNCTION SetCurDir : BOOLEAN;

Returns TRUE if the object string includes a directory component that can be made the current working directory. The object string can include a full DOS path/file name. This process will extract the directory component, and attempt to change the current directory to that component. The current object text string is NOT affected.

TStringClass : Search related functions

```
FUNCTION ChCount(ACh : CHAR) : WORD;
FUNCTION FindBetween2Ch(FirstCh,
                        SecondCh      : CHAR;
                        StartFrom      : WORD;
                        VAR SubStrStart : WORD;
                        VAR SubStrLen  : WORD;
                        CutSubStr,
                        IncDelims      : BOOLEAN;
                        VAR ASubStr    : TStringClass) : BOOLEAN;
FUNCTION FindFirst(CONST SubArgs : ARRAY OF CONST; VAR P : WORD) : BOOLEAN;
FUNCTION FindFirstCh(ACh : CHAR; VAR P : WORD) : BOOLEAN;
FUNCTION FindLast(CONST SubArgs : ARRAY OF CONST; VAR P : WORD) : BOOLEAN;
FUNCTION FindLastCh(ACh : CHAR; VAR P : WORD) : BOOLEAN;
FUNCTION FindNext(CONST SubArgs : ARRAY OF CONST;
                  StartPos : WORD;
                  VAR NextPos : WORD) : BOOLEAN;
FUNCTION FindNextCh(ACh : CHAR; StartPos : WORD; VAR NextPos : WORD) : BOOLEAN;
FUNCTION FindPrev(CONST SubArgs : ARRAY OF CONST;
                  StartPos : WORD; VAR PrevPos : WORD) : BOOLEAN;
FUNCTION FindPrevCh(ACh : CHAR; StartPos : WORD; VAR PrevPos : WORD) : BOOLEAN;
FUNCTION Grep(CONST SearchArgs : ARRAY OF CONST; VAR AList : TWordContainer) : WORD;
FUNCTION SubStrCount(CONST SubArgs : ARRAY OF CONST) : WORD;
```

TStringClass.ChCount

FUNCTION ChCount(ACh : CHAR) : WORD;

Returns the number of occurrences of CHAR 'Ch' in the object string.

TStringClass.FindBetween2Ch

FUNCTION FindBetween2Ch(FirstCh,

SecondCh : CHAR;

StartFrom : WORD;

VAR SubStrStart : WORD;

VAR SubStrLen : WORD;

CutSubStr,

IncDelims : BOOLEAN;

VAR ASubStr : TStringClass) : BOOLEAN;

Retrieves a sub string from the object's own text string starting from the first instance of character 'FirstCh' to the next instance of character 'SecondCh'. The substring is assigned to the 'ASubStr' string object. The 'ASubStr' object must be an already instantiated TStringClass object. The function returns TRUE if a sub string is found. The base 0 relative start position and length is returned in 'SubStrStart' and 'SubStrLen'. If 'CutSubStr' is TRUE the located sub string is deleted from the object's own text string. If 'IncDelims' is TRUE the FirstCh/SecondCh characters are included in the string assigned to 'ASubStr'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindFirst

FUNCTION FindFirst(CONST SubArgs : ARRAY OF CONST; VAR P : WORD) : BOOLEAN;

Searches for the first occurrence of sub string 'SubArgs' in the object string, and returns TRUE if the sub string is found. The position of the sub string (base 0) is returned in variable parameter 'P'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindFirstCh

FUNCTION FindFirstCh(ACH : CHAR; VAR P : WORD) : BOOLEAN;

Searches for the first occurrence of CHAR 'Ch' in the object string, and returns TRUE if the CHAR is found. The position of the CHAR (base 0) is returned in variable parameter 'P'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindLast

FUNCTION FindLast(CONST SubArgs : ARRAY OF CONST; VAR P : WORD) : BOOLEAN;

Searches for the last occurrence of sub string 'SubArgs' in the object string (starting from the end of the string), and returns TRUE if the sub string is found. The position of the sub string (base 0) is returned in variable parameter 'P'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindLastCh

FUNCTION FindLastCh(ACh : CHAR; VAR P : WORD) : BOOLEAN;

Searches for the last occurrence of CHAR 'Ch' in the object string, and returns TRUE if the CHAR is found. The position of the CHAR (base 0) is returned in variable parameter 'P'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindNext

```
FUNCTION FindNext(CONST SubArgs : ARRAY OF CONST;  
                  StartPos : WORD;  
                  VAR NextPos : WORD) : BOOLEAN;
```

Searches for the next occurrence of sub string 'SubArgs' in the object string starting from character position 'StartPos' (base 0), and returns TRUE if the sub string is found. The position of the sub string (base 0) is returned in variable parameter 'NextPos'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindNextCh

FUNCTION FindNextCh(ACh : CHAR;
StartPos : WORD; VAR NextPos : WORD) : BOOLEAN;

Searches for the next occurrence of CHAR 'Ch' in the object string starting from character position 'StartPos' (base 0), and returns TRUE if the CHAR is found. The position of the CHAR (base 0) is returned in variable parameter 'NextPos'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindPrev

**FUNCTION FindPrev(CONST SubArgs : ARRAY OF CONST;
StartPos : WORD; VAR PrevPos : WORD) : BOOLEAN;**

Searches for the previous occurrence of sub string 'SubArgs' in the object string starting from character position 'StartPos' (base 0) and searching in a right to left direction. The function returns TRUE if the sub string is found. The position of the new sub string (base 0) is returned in variable parameter 'PrevPos'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.FindPrevCh

FUNCTION FindPrevCh(ACH : CHAR; StartPos : WORD; VAR PrevPos : WORD) : BOOLEAN;

Searches for the previous occurrence of CHAR 'Ch' in the object string starting from character position 'StartPos' (base 0) and searching in a right to left direction. The function returns TRUE if the CHAR is found. The position of the new CHAR (base 0) is returned in variable parameter 'PrevPos'.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.Grep

**FUNCTION Grep(CONST SearchArgs : ARRAY OF CONST;
VAR AList : TWordContainer) : WORD;**

This method will search the object's own string for a pattern of characters defined in the 'SearchArgs' parameter, and will report the position of each such occurrence as a WORD entry in the word list array 'AList'. The method function returns the number of occurrences found.

For example

```
VAR
    MObj : TStringClass;
    AList : TWordContainer;
BEGIN
    .....
    .....
    MObj := TStringClass.CreateString(['WIN.INI']);
    AList := TWordContainer.Create
    MObj.Grep(['IN'],AList)
    .....
    .....
```

The 'Grep' function will return a result of '2' will word array entries of '1' and '4'. **Note:** if the typed constant 'StrClassBaseZero' is set to FALSE then the character positions returned in the word array are base one (1).

The search pattern is case sensitive.

The 'SearchArgs' parameter can take a number of special characters:

- ? This can be any character. Thus, a search pattern of 'A?' will find both 'AB' and 'AZ'.
- \ This is a special delimiter to indicate that the next character in the parameter list should be taken as a literal. Thus, if you wanted to search for the '?' character you should use '\?' as part of the search pattern.
- [] Any set of characters within square brackets allows an 'OR' type test against each of the characters within the square brackets. For example a search pattern of 'A[BC]' will find both 'AB' and 'AC'.
- [!] Placing a '!' character as the first within the square brackets means that the search will succeed for any character other than those listed within the square brackets. For example., a search pattern of 'A![BC]' will find 'AD' but not find 'AB'.
- :a This will match any alphabetic character ('A' to 'Z' or 'a' to 'z'), irrespective of case
- :d This will match any digit ('0' to '9')
- :n This will match any alpha-numeric ('A' to 'Z', or 'a' to 'z', or '0' to '9')
- A minus sign placed after a character makes that character's presence optional For example, a search pattern of 'SM-X' will find both 'SMX' and 'SX'.
- * An asterisk (*) placed after a character will set matches to occur if optional extensions of the same character appear. Thus, 'FR*' will find both 'FR' and 'FRR' and 'FRRR'.
- + A plus placed after a character will set matches to occur of further examples of the same character are found. Thus, 'FR+' will find both 'FRR' and 'FRRR', but not find 'FR'.

The StrClass' unit includes a number of constants to assist with the preparation of grep related search patterns ...

```
grep_any      = '?';
grep_nextLiteral = '\';
grep_optionStart = '[';
grep_optionEnd   = ']';
  grep_class     = '.';
grep_Alpha      = 'A';
grep_numeric    = 'D';
grep_alphaNumeric = 'N';
grep_Chcont0    = '*';
grep_Chcont1    = '+';
grep_ChOption   = '-';
grep_Not        = '!';
```

The example project 'GrepTest.Dpr' can be used to demonstrate the various grep related search options.

If the 'SearchArgs' parameter includes an invalid sequence of characters an exception error is raised.

TStringClass.SubStrCount

FUNCTION SubStrCount(CONST SubArgs : ARRAY OF CONST) : WORD;

Returns the total number of occurrences that sub string 'SubArgs' is located within the object string.

TStringClass : Case related functions

```
FUNCTION FirstCharToUpper : PChar;  
FUNCTION IsAlphaNumeric : BOOLEAN;  
FUNCTION ToLower : PChar;  
FUNCTION ToUpper : PChar;
```

TStringClass.FirstCharToUpper

FUNCTION FirstCharToUpper : PChar;

Converts the first character of each separate word in the object string to upper case. For example:

```
Str1.Assign('This is a test')
Str1.FirstCharToUpper;
WriteLn(Str1.ZString);           { This Is A Test }
```

TStringClass.IsAlphaNumeric

FUNCTION IsAlphaNumeric : BOOLEAN;

Returns TRUE if all characters in the text string are alpha numeric (between 'A' to 'Z' , 'a' to 'z', or '0' to '9').

TStringClass.ToLower

FUNCTION ToLower : PChar;

Converts the whole object string to lower case.

TStringClass.ToUpper

FUNCTION ToUpper : Pchar;

Converts the whole object string to upper case.

TStringClass : Search & replace related functions

FUNCTION ReplaceAll(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;
FUNCTION ReplaceChAll(OldCh,NewCh : CHAR) : PChar;
FUNCTION ReplaceChFirst(OldCh,NewCh : CHAR) : PChar;
FUNCTION ReplaceChLast(OldCh,NewCh : CHAR) : PChar;
FUNCTION ReplaceFirst(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;
FUNCTION ReplaceLast(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;

TStringClass.ReplaceAll

FUNCTION ReplaceAll(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;

Replaces every occurrence of the sub string 'OldArgs' in the object string with the replacement sub string 'NewArgs'. If 'NewArgs' is NIL then the process will just remove all instances of 'OldArgs'.

TStringClass.ReplaceChAll

FUNCTION ReplaceChAll(OldCh,NewCh : CHAR) : PChar;

Replaces every occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

TStringClass.ReplaceChFirst

FUNCTION ReplaceChFirst(OldCh,NewCh : CHAR) : PChar;

Replaces the first occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

TStringClass.ReplaceChLast

FUNCTION ReplaceChLast(OldCh,NewCh : CHAR) : Pchar;

Replaces the last occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

TStringClass.ReplaceFirst

FUNCTION ReplaceChLast(OldCh,NewCh : CHAR) : Pchar;

Replaces the last occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

TStringClass.ReplaceLast

FUNCTION ReplaceLast(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;

Replaces the last occurrence of the sub string 'OldArgs' in the object string with the replacement sub string 'NewArgs'. If 'NewArgs' is NIL then process will just remove 'OldArgs'.

TStringClass.FirstParseDelim

```
FUNCTION FirstParseDelim(CONST Args : ARRAY OF CONST;  
DelimCh      : CHAR;  
VAR DelimPos : WORD) : BOOLEAN;
```

The 'Args' string is presumed to be a string holding multiple values delimited by the character 'DelimCh'. This method will extract the first item from the source string and assign it to this object string. In the process it sets the variable parameter 'DelimPos' with the index position (base 0) of the next delimited item. The function will return FALSE if the 'Args' string is NIL or of zero length.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.NextParseDelim

```
FUNCTION NextParseDelim(CONST Args : ARRAY OF CONST;  
                        DelimCh   : CHAR;  
                        StartPos   : WORD;  
                        VAR NextDelimPos : WORD) : BOOLEAN;
```

Used in association with 'FirstParseDelim', this method returns the next item to be parsed from the 'Args' string. The search starts at position 'StartPos' and assigns the parsed string into the object string. The method updates the variable parameter 'NextDelimPos' with the starting position of the next delimited entry. The method returns FALSE if there are no further string items to parse.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

TStringClass.ParseDelimCount

FUNCTION ParseDelimCount(DelimCh : CHAR) : WORD;

Returns the number of items in a string delimited by the 'DelimCh' character.

TStringClass.ParseDelimToList

**FUNCTION ParseDelimToList(DelimCh : CHAR;
Special : INTEGER;
VAR AList : TObjectContainer) : WORD;**

A high level method of parsing the object's own string. 'DelimCh' specifies the character used to delimiter string entries. The method adds a new 'TStringClass' to the 'AList' container class for each item it parses from its own string (the object string is unaffected by this process). The container class owns the newly created string class objects. The 'Special' parameter is used to determine how double inserted commas or single inverted commas should be treated. The options are:

```
delim_None    = 0;    { do nothing }
delim_IncDIC  = 1;    { add double inverted commas to the start/end of each item }
delim_IncSIC  = 2;    { add single inverted commas to the start/end of each item }
delim_ExcDIC  = 4;    { remove any double inverted commas from the start/end of each item }
delim_ExcSIC  = 8;    { remove any single inverted commas from the start/end of each item }

delim_ExcNull = 16   { exclude any NULL strings from the list }
```

If there is no text between the location of two delimiters the String Object created for that item is set to have a NIL string. However, if the 'Special' parameter includes 'delim_ExcNull' such null part strings are **not** added to this list.

If the 'AList' parameter is passed as a NIL value this method will create the container object. The function returns the number of items parsed from the object's own string.

Examples

```
VAR
  AList : TObjectContainer;
  SObj  : TStringClass;

  { ++++++ }

PROCEDURE AddToListBox(S : TStringClass); FAR;
BEGIN
  AListBox.Items.Add(S.Text);
END;

  { ++++++ }

BEGIN
  ...
  ...
  AList := TObjectContainer.Create;
  SObj := TStringClass.CreateString(['aaa,bbb,ccc']);
  SObj.ParseDelimToList(',',delim_none,AList);
  AList.ForEach(@AddToListBox);
  AList.Free;
  SObj.Free;
  ...
  ...
END;
```

With the above you must not forget to make the nested procedure a 'FAR', and the 'action' procedure ('AddToListBox') can only be nested within the same procedure block.

...oOo...

Or, secondly

```
VAR
  AList : TObjectContainer;
  PartObj,SObj : TStringClass;
  E : LONGINT;
BEGIN
  ...
  ...
  AList := TObjectContainer.Create;
  SObj := TStringClass.CreateString(['aaa,bbb,ccc']);
  SObj.ParseDelimToList(',',delim_none,AList);
  FOR E := 0 TO AList.Count-1 DO
  BEGIN
    PartObj := AList.Items[E];
    AListBox.Items.Add(PartObj.Text);
  END;
  AList.Free;
  SObj.Free;
  ...
  ...
END;
```

Where the source string ends with a delimiter character, the parsing process adds a null string object to the target list. For example

```
AStrObj.Assign(['AA*BB*CC*']);
AStrObj.ParseDelimToList('*',delim_none,AList)
```

.... returns a result of 5 sub strings: 'AA', 'BB', 'CC', null and null.

TStringClass.ParseMultiDelimToList

```
FUNCTION ParseMultiDelimToList(CONST DelimArgs : ARRAY OF CONST;  
                               Special          : INTEGER;  
                               VAR AList      : TObjectContainer) : WORD;
```

This method is similar to the previous 'ParseDelimToList'. It differs in that the first parameter can contain multiple delimiters and that these delimiters can be of any variable type accepted as part of an 'ARRAY OF CONST'. Thus a delimiter may be either a single character or multi-character string. The parsing process checks each item on the 'DelimArgs' list to see what delimiter appears first. The order of items within the 'DelimArgs' parameter is not relevant.

For example

```
TObj := TStringClass.CreateString(['>QFADS 14:30:30 - 14:45:00 95/11/01']);  
TObj.ParseMultiDelimToList([' - ','>QFADS',';','/'],delim_ExcNull,AList);
```

.... will parse down to ...

```
14  
30  
30  
14  
45  
00  
95  
11  
01
```

The method uses the same set of 'Special' parameters as outlined under 'ParseDelimToList'.

TStringClass.ParsePosToList

**FUNCTION ParsePosToList(VAR PosArray;
PosCt : WORD;
VAR AList : TObjectContainer) : WORD;**

A high level method of parsing the object's own string based around an array of start and length settings passed to it in the 'PosArray' parameter. The 'PosArray' parameter must be a two dimension INTEGER array that holds 'PosCt' number of entries. Each array entry must have as its first element the starting position of the sub string to be extracted (base zero) and as its second element the number of characters to extracted.

Note: if the typed constant 'StrClassBaseZero' is set to FALSE then the first character position starts from one (1) not zero (0).

For each entry in the PosArray the method will create a new string object, assign it the sub string extracted from the main string and add it to the 'AList' collection (the collection owns all string objects created in this manner). If the PosArray entries are located at a position beyond the end of the string the string object created for that entry is assigned a NIL string. If the 'AList' parameter is passed as a NIL value this method will create the collection object. The function returns the number of items parsed and added to the 'AList' collection.

TBaseContainer

All container objects derive from the abstract class 'TBaseContainer' .

Properties

Public Methods

TBaseContainer : Public Methods

```
CONSTRUCTOR Create; VIRTUAL;  
DESTRUCTOR Destroy; OVERRIDE;  
PROCEDURE Clear; VIRTUAL;  
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;  
PROCEDURE DeleteAll; VIRTUAL;  
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);  
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;  
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;  
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;  
PROCEDURE Pack; VIRTUAL;  
PROCEDURE RemoveAll;
```

TBaseContainer Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TBaseContainer.Create

CONSTRUCTOR *Create*; **VIRTUAL**;
Creates a new object.

TBaseContainer.Destroy

DESTRUCTOR Destroy; OVERRIDE;

Destroys the open array list and all its list items

TBaseContainer.Delete

PROCEDURE Delete(Idx : LONGINT); ***VIRTUAL***;

Deletes the Idx item (base 0) from the list. The item itself is not disposed of.

TBaseContainer.DeleteAll

PROCEDURE DeleteAll; VIRTUAL;

Deletes all items from the list, resetting the count to zero. None of the items previously on the list are disposed of.

TBaseContainer.DeleteBlock

PROCEDURE DeleteBlock(Sidx, Eidx : LONGINT);

Deletes a block of items starting from item 'Sidx' to 'Eidx' inclusive (base 0), without disposing of any objects/records.

TBaseContainer.Exchange

PROCEDURE Exchange(*Idx1,Idx2: LONGINT*); ***VIRTUAL***;
Exchanges the position of two items on the list.

TBaseContainer.InsertBlock

FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; ***VIRTUAL;***
Inserts 'Number' empty (null) item pointers starting from position Idx.

TBaseContainer.Move

PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;
Moves the item 'CurIdx' to the new position 'NewIdx'.

TBaseContainer.Pack

PROCEDURE Pack; VIRTUAL;

Scans the list to remove any null item pointers.

TBaseContainer.RemoveAll

PROCEDURE RemoveAll;

Removes and disposes of all objects/records on the list.

TBaseContainer.Capacity

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;

Each list can be assigned a capacity. Items are added to list until the list count reaches the capacity. Once this is reached the list is re-organised to provide extra capacity. By defining a large capacity in advance the process can avoid wasteful list re-organising.

TBaseContainer.Count

PROPERTY Count: LONGINT READ Fcount;
Returns the number of items on the list.

TBaseContainer.CompareFunc

PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;

For containers where items are to be sorted the object can use a global function located in another unit to determine the comparison result between items on the list. This is achieved by assigning the address of that global function to this property. The global function must be of the type

```
TCompareFunc = FUNCTION(VAR Ptr1,Ptr2) : INTEGER;
```

The two parameters 'Ptr1' and 'Ptr2' are of the same type as the item being stored on the list, i.e. if the list holds objects each parameter will be an object pointer (TObject), or INTEGER types if the list holds integers. The compare function assigned to this property must be declared in the interface section of a unit. This function must return an INTEGER result. If the related sort process requires that 'Ptr1' be located before 'Ptr2' then the function should return a '-1' result. If the sort process determines that both parameters are equal the function should return a '0' result. Or if 'Ptr1' is to be placed after 'Ptr2' then the function must return a '1' result.

For example ...

```
INTERFACE
```

```
TYPE
```

```
    TDataClass = CLASS(TObject)
        PRIVATE
            Name : STRING;
        PUBLIC
            CONSTRUCTOR CreateName(CONST AName : STRING);
            FUNCTION GetName : STRING;
        END;

    FUNCTION NameCompare(VAR Ptr1,Ptr2) : INTEGER;
```

```
IMPLEMENTATION
```

```
.....
```

```
FUNCTION NameCompare(VAR Ptr1,Ptr2) : INTEGER;
BEGIN
    Result := CompareText(TDataClass(Ptr1).GetName,TDataClass(Ptr2).GetName);
END;
```

```
.....
```

```
AList := TObjectContainer.Create;
AList.Sort := sortAscending;
AList.CompareFunc := NameCompare;
```

If for some reason a container object uses both this property and overrides the 'Compare' function method, then the sort process will always use the virtual Compare method.

TBaseContainer.Delta

PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;

Sets or returns the size by which the capacity will grow each time the count reaches the capacity limit.

TBaseContainer.Duplicates

PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE Fduplicates;

This property determines how to handle the addition of duplicate items to a sorted container, and only applies to containers which are sorting their contents. By default this property is set to '**dupIgnore**'. With sorted collections this means that an attempt to add a duplicate item will simply be ignored, with no exception error.

```
TDuplicates = (dupIgnore, dupAccept, dupError);
```

If this property is set to '**dupAccept**' then the container will process and accept duplicate list entries. However, if this property is set to '**dupError**', then any attempt to add a duplicate item will create a run time exception error.

TBaseContainer.GrowAsRequired

PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;

Using the 'Items' property found in descendant container objects a list item can be assigned to a specific list position. This does not have to be at the last used position. This allows a list to hold non-contiguous entries. The GrowAsRequired property is used to control whether such 'Items' assignments can be made past the capacity limit. If set to TRUE (the default) then the capacity will grow to meet any use of the 'Items' property. If set to FALSE then any attempt to use the 'Items' property outside of the list capacity will create an exception error.

TBaseContainer.Sort

PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

Used to set whether a list should be sorted into a pre-determined order. The attribute should be passed a 'TContainerSortType' value.

```
TContainerSortType = (sortNone,sortAscending,sortDescending);
```

A sorted container requires that either:

- a new class be derived from the required class type, and the protected method 'Compare' be overridden with a new method.

```
FUNCTION Compare(Ptr1,Ptr2 : POINTER) : INTEGER; VIRTUAL;
```

- or that the 'CompareFunc' function property be set. This is a function variable type uses the following prototype format ..

```
TCompareFunc = FUNCTION(VAR Ptr1,Ptr2) : INTEGER;
```

The two parameters 'Ptr1' and 'Ptr2' are of the same type as the item being stored on the list, i.e. if the list holds objects each parameter will be an object pointer (TObject), or INTEGER types if the list holds integers. The compare function assigned to this property must be declared in the interface section of a unit. This function must return an INTEGER result. If the related sort process requires that 'Ptr1' be located before 'Ptr2' then the function should return a '-1' result. If the sort process determines that both parameters are equal the function should return a '0' result. Or if 'Ptr1' is to be placed after 'Ptr2' then the function must return a '1' result.

For example ...

```
INTERFACE
```

```
TYPE
```

```
TDataClass = CLASS(TObject)
  PRIVATE
    Name : STRING;
  PUBLIC
    CONSTRUCTOR CreateName(CONST AName : STRING);
    FUNCTION GetName : STRING;
END;

FUNCTION NameCompare(VAR Ptr1,Ptr2) : INTEGER;
```

```
IMPLEMENTATION
```

```
.....
```

```
FUNCTION NameCompare(VAR Ptr1,Ptr2) : INTEGER;
BEGIN
  Result := CompareText(TDataClass(Ptr1).GetName,TDataClass(Ptr2).GetName);
END;
```

.....

```
AList := TObjectContainer.Create;  
AList.Sort := sortAscending;  
AList.CompareFunc := NameCompare;
```

TObjectContainer

An open ended array type container for holding object pointers.

Public Methods

Properties

TObjectContainer : Public Methods

CONSTRUCTOR Create; OVERRIDE;
DESTRUCTOR Destroy; OVERRIDE;
FUNCTION Add(Item: POINTER): LONGINT; VIRTUAL;
FUNCTION Append(Item : POINTER) : LONGINT; VIRTUAL;
PROCEDURE Clear; VIRTUAL;
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;
PROCEDURE DeleteAll; VIRTUAL;
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;
FUNCTION First: POINTER; VIRTUAL;
FUNCTION FirstThat(Test: Pointer): Pointer;
FUNCTION FirstThatIdx(Test: Pointer): Pointer;
PROCEDURE ForEach(Action: Pointer);
PROCEDURE ForEachIdx(Action: Pointer);
FUNCTION Includes(Item : POINTER; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION IndexOf(Item: POINTER): LONGINT; VIRTUAL;
FUNCTION Insert(Idx: LONGINT; Item: POINTER) : LONGINT; VIRTUAL;
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Last: POINTER;
FUNCTION LastThat(Test: Pointer): Pointer;
FUNCTION LastThatIdx(Test: Pointer): Pointer;
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;
PROCEDURE Pack; VIRTUAL;
FUNCTION Prepend(Item : POINTER) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: POINTER): LONGINT; VIRTUAL;
PROCEDURE RemoveAll;

TObjectContainer.Add

FUNCTION Add(Item: POINTER): LONGINT; VIRTUAL;

Adds an instantiated object to the list. If sorting is not active then the object instance is added to the end of the list. If sorting is active then the object will be placed in a position based upon the objects 'Compare' result.

TObjectContainer.Create

CONSTRUCTOR Create; OVERRIDE;
Creates an instance of the new object type.

TObjectContainer.Append

FUNCTION Append(Item : POINTER) : LONGINT; VIRTUAL;
Adds the object to the end of the list.

TObjectContainer.First

FUNCTION First: POINTER; VIRTUAL;
Returns a pointer to the first object on the list.

TObjectContainer.FirstThat

FUNCTION FirstThat(Test: Pointer): Pointer;

Iterates across the objects on the list calling the local nested function 'Test' with the current list object as the parameter. The process continues until such time that the test function returns TRUE. The function returns a pointer to the object that was active when the TRUE condition prevailed. If no TRUE return is returned the 'FirstThat' function returns NIL.

Example ...

```
VAR
  Alist : TObjectContainer;

  { ++++++ }

  FUNCTION DoTest(AnObject : TMyObject) : BOOLEAN; FAR;
  BEGIN

  END;

  { ++++++ }

BEGIN
  Alist := TObjectContainer.Create;
  .....
  .....
  Alist.FirstThat(@DoTest);
  .....
  .....
```

The 'Test' function must be nested function and must be declared as a 'FAR' function. IT can have only one parameter - an object of whatever type is being stored in the container.

This function is useful for reviewing objects on the list and returning a pointer to some object that meets a test.

TObjectContainer.ForEach

PROCEDURE ForEach(Action: Pointer);

Similar to 'FirstThat' this function iterates across all objects on the list, running the procedure 'Action' for each object. The iteration cannot be stopped. As with 'FirstThat', the 'Action' procedure must be nested function and must be declared as a 'FAR' type. It can have only one parameter - an object of whatever type is being stored in the container. This function is useful for reviewing objects on the list and accumulating list wide totals.

TObjectContainer.Includes

FUNCTION Includes(Item : POINTER; VAR Idx : LONGINT) : BOOLEAN;

Returns TRUE if the object 'Item' is found on the list. The base 0 zero list position is returned in 'Idx'

TObjectContainer.IndexOf

FUNCTION IndexOf(Item: POINTER): LONGINT; VIRTUAL;
Returns the base 0 index position of an object on the list.

TObjectContainer.Insert

FUNCTION Insert(Idx: LONGINT; Item: POINTER) : LONGINT; VIRTUAL;

Inserts an object 'Item' at the list position 'Idx'. This was create an exception error if sorting is active.

TObjectContainer.Last

FUNCTION Last: POINTER;

Returns a pointer to the last object on the list.

ObjectContainer.LastThat

FUNCTION LastThat(Test: Pointer): Pointer;

Similar to the 'FirstThat' function this iterates across the list from last to first.

TObjectContainer.Prend

FUNCTION Prepend(Item : POINTER) : LONGINT; VIRTUAL;
Insert the object 'Item' at the front of the list.

TObjectContainer.Remove

FUNCTION Remove(Item: POINTER): LONGINT; VIRTUAL;
Removes the object 'Item' from the list and disposes it.

TObjectContainer.Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: POINTER READ GetPtr WRITE PutPtr;
PROPERTY Own : BOOLEAN READ Fown WRITE Fown;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TObjectContainer.Items

PROPERTY Items[Index : LONGINT]: POINTER READ GetPtr WRITE PutPtr;

Allows an object to be assigned to a zero based position in the list, or returns the object found at position Index.

TObjectContainer.Own

PROPERTY Own : BOOLEAN READ Fown WRITE Fown;

If set to TRUE the list is deemed to own the objects and will dispose of them once they are removed from the list. If set to FALSE the object will NOT dispose of the object if it is removed from the list.

TRecordContainer

A container object to used to store lists of record pointers. The 'Items' property should have assigned to it a pointer to the record structure.

Public Methods

Properties

TRecordContainer : Public Methods

```
CONSTRUCTOR Create(ARecSize : WORD);  
DESTRUCTOR Destroy; OVERRIDE;  
FUNCTION Add(Item: POINTER): LONGINT; VIRTUAL;  
FUNCTION Append(Item : POINTER) : LONGINT; VIRTUAL;  
PROCEDURE Clear; VIRTUAL;  
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;  
PROCEDURE DeleteAll; VIRTUAL;  
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);  
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;  
FUNCTION First: POINTER; VIRTUAL;  
FUNCTION FirstThat(Test: Pointer): Pointer;  
PROCEDURE ForEach(Action: Pointer);  
FUNCTION Includes(Item : POINTER; VAR Idx : LONGINT) : BOOLEAN;  
FUNCTION IndexOf(Item: POINTER): LONGINT; VIRTUAL;  
FUNCTION Insert(Idx: LONGINT; Item: POINTER) : LONGINT; VIRTUAL;  
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;  
FUNCTION Last: POINTER;  
FUNCTION LastThat(Test: Pointer): Pointer;  
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;  
PROCEDURE Pack; VIRTUAL;  
FUNCTION Prepend(Item : POINTER) : LONGINT; VIRTUAL;  
FUNCTION Remove(Item: POINTER): LONGINT; VIRTUAL;  
PROCEDURE RemoveAll;
```


TPCharContainer : Public Methods

CONSTRUCTOR Create; OVERRIDE;
DESTRUCTOR Destroy; OVERRIDE;
FUNCTION Add(Item: POINTER): LONGINT; VIRTUAL;
FUNCTION Append(Item : POINTER) : LONGINT; VIRTUAL;
PROCEDURE Clear; VIRTUAL;
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;
PROCEDURE DeleteAll; VIRTUAL;
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;
FUNCTION First: POINTER; VIRTUAL;
FUNCTION FirstThat(Test: Pointer): Pointer;
PROCEDURE ForEach(Action: Pointer);
FUNCTION Includes(Item : POINTER; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION IndexOf(Item: POINTER): LONGINT; VIRTUAL;
FUNCTION Insert(Idx: LONGINT; Item: POINTER) : LONGINT; VIRTUAL;
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Last: POINTER;
FUNCTION LastThat(Test: Pointer): Pointer;
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;
PROCEDURE Pack; VIRTUAL;
FUNCTION Prepend(Item : POINTER) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: POINTER): LONGINT; VIRTUAL;
PROCEDURE RemoveAll;

TIntegerContainer.Create

CONSTRUCTOR Create;

Creates a container object to hold integer items.

TRecordContainer : Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: POINTER READ GetPtr WRITE PutPtr;
PROPERTY Own : BOOLEAN READ Fown WRITE Fown;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TRecordContainer.Create

CONSTRUCTOR Create(ARecSize : WORD);
Used to create the record container. Example ...

```
TYPE
  PMyRec = ^TMyRec;
  TMyRec = RECORD
    Name : STRING;
    Age   : LONGINT;
  END;
VAR
  Alist : TRecordContainer
  Arc   : PMyRec;
BEGIN
  .....
  .....
  Alist := TRecordContainer.Create(SIZEOF(TMyRec));
```

TPCharContainer

A container object for holding lists of 'PChar' variable types.

Public Methods

Properties

TIntegerContainer

A container for holding lists of integers.

Public Methods

Properties

TPCharContainer : Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: POINTER READ GetPtr WRITE PutPtr;
PROPERTY Own : BOOLEAN READ Fown WRITE Fown;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TIntegerContainer.Add

FUNCTION Add(Item: INTEGER): LONGINT; VIRTUAL;

Adds the new integer value to the list. If sorting is not active it is appended to the end of the list.

TIntegerContainer : Public Methods

```
CONSTRUCTOR Create;  
DESTRUCTOR Destroy; OVERRIDE;  
FUNCTION Add(Item: INTEGER): LONGINT; VIRTUAL;  
FUNCTION Append(Item : INTEGER) : LONGINT; VIRTUAL;  
PROCEDURE Clear; VIRTUAL;  
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;  
PROCEDURE DeleteAll; VIRTUAL;  
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);  
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;  
FUNCTION First: INTEGER; VIRTUAL;  
FUNCTION Includes(Item : INTEGER; VAR Idx : LONGINT) : BOOLEAN;  
FUNCTION Insert(Idx: LONGINT; Item: INTEGER) : LONGINT; VIRTUAL;  
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;  
FUNCTION Last: INTEGER;  
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;  
PROCEDURE Pack; VIRTUAL;  
FUNCTION Prepend(Item : INTEGER) : LONGINT; VIRTUAL;  
FUNCTION Remove(Item: INTEGER): LONGINT; VIRTUAL;  
PROCEDURE RemoveAll;
```

TIntegerContainer.Append

FUNCTION Append(Item : INTEGER) : LONGINT; VIRTUAL;
Appends the new integer item to the end of the list.

TIntegerContainer.First

FUNCTION First: INTEGER; VIRTUAL;

Returns the integer value of the first item on the list.

TIntegerContainer.Includes

FUNCTION Includes(Item : INTEGER; VAR Idx : LONGINT) : BOOLEAN;

Returns TRUE if the 'Item' integer is found on the list. 'Idx' is returned with the base zero list position.

TIntegerContainer.Insert

FUNCTION Insert(Idx: LONGINT; Item: INTEGER) : LONGINT; VIRTUAL;
Inserts the new integer entry 'Item' into the list at position 'Idx' (base 0)

TIntegerContainer.Last

FUNCTION Last: INTEGER;

Returns the integer value of the last item on the list.

TIntegerContainer.Prend

FUNCTION Prepend(Item : INTEGER) : LONGINT; VIRTUAL;
Inserts the Item value into the start of the list.

TIntegerContainer.Remove

FUNCTION Remove(Item: INTEGER): LONGINT; VIRTUAL;
Removes the integer item from the list.

TIntegerContainer : Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: INTEGER READ GetInteger WRITE PutInteger;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TIntegerContainer.Items

PROPERTY Items[Index : LONGINT]: INTEGER READ GetInteger WRITE PutInteger;

Used to assign integer values to specific positions on the list, or to return the integer value found at position 'Idx' on the list.

TWordContainer

A container for holding lists of WORD type values. The methods and properties have the exact same purpose as for the TIntegerContainer, except that they use WORD type item parameters.

Public Methods

Properties

TLongIntContainer

A container for holding lists of LONGINT type values. The methods and properties have the exact same purpose as for the [TIntegerContainer](#), except that they use LONGINT type item parameters.

[Public Methods](#)

[Properties](#)

TLongIntContainer : Public Methods

CONSTRUCTOR Create;
DESTRUCTOR Destroy; OVERRIDE;
FUNCTION Add(Item: LONGINT): LONGINT; VIRTUAL;
FUNCTION Append(Item : LONGINT) : LONGINT; VIRTUAL;
PROCEDURE Clear; VIRTUAL;
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;
PROCEDURE DeleteAll; VIRTUAL;
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;
FUNCTION First: LONGINT; VIRTUAL;
FUNCTION Includes(Item : LONGINT; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION Insert(Idx: LONGINT; Item: LONGINT) : LONGINT; VIRTUAL;
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Last: LONGINT;
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;
PROCEDURE Pack; VIRTUAL;
FUNCTION Prepend(Item : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: LONGINT): LONGINT; VIRTUAL;
PROCEDURE RemoveAll;

TWordContainer : Public Methods

CONSTRUCTOR Create;
DESTRUCTOR Destroy; OVERRIDE;
FUNCTION Add(Item: WORD): LONGINT; VIRTUAL;
FUNCTION Append(Item : WORD) : LONGINT; VIRTUAL;
PROCEDURE Clear; VIRTUAL;
PROCEDURE Delete(Idx : LONGINT); VIRTUAL;
PROCEDURE DeleteAll; VIRTUAL;
PROCEDURE DeleteBlock(SIdx,EIdx : LONGINT);
PROCEDURE Exchange(Idx1,Idx2: LONGINT); VIRTUAL;
FUNCTION First: WORD; VIRTUAL;
FUNCTION Includes(Item : WORD; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION Insert(Idx: LONGINT; Item: WORD) : LONGINT; VIRTUAL;
FUNCTION InsertBlock(Idx,Number : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Last: WORD;
PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;
PROCEDURE Pack; VIRTUAL;
FUNCTION Prepend(Item : WORD) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: WORD): LONGINT; VIRTUAL;
PROCEDURE RemoveAll;

TLongIntContainer : Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: LONGINT READ GetLongInt WRITE PutLongInt;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TWordContainer : Properties

PROPERTY Capacity: LONGINT READ FCapacity WRITE SetCapacity;
PROPERTY CompareFunc : TCompareFunc READ FCompareFunc WRITE FCompareFunc;
PROPERTY Count: LONGINT READ FCount;
PROPERTY Delta : LONGINT READ FDelta WRITE FDelta;
PROPERTY Duplicates : TDuplicates READ FDuplicates WRITE FDuplicates;
PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
PROPERTY Items[Idx : LONGINT]: WORD READ GetWord WRITE PutWord;
PROPERTY Sort : TContainerSortType READ FSort WRITE SetSort;

TObjectContainer.FirstThatIdx

FUNCTION FirstThatIdx(Test: Pointer): Pointer;

This is a variation on the 'FirstThat' method described above. With 'FirstThatIdx' the nested local function requires two parameters. The first being an object pointer of whatever type is being stored on the list. The second must be a LONGINT parameter. This is passed by the FirstThat process and represents the base zero (0) index position of current data object within the list (i.e. somewhere between '0' to 'Count-1').

```
VAR
  AList : TObjectContainer;
  ATotal : LONGINT

  { ++++++ }

  FUNCTION DoTest(AnObject : TMyObject; Idx : LONGINT) : BOOLEAN; FAR;
  BEGIN
    ...
  END;

  { ++++++ }

BEGIN
  AList := TObjectContainer.Create;
  .....
  .....
  ATotal := 0;
  AList.FirstThatIdx(@DoTest);
  .....
  .....
```

TObjectContainer.ForEachIdx

PROCEDURE ForEachIdx(Action: Pointer): Pointer;

This is a variation on the 'ForEach' method. With 'ForEachIdx' the nested local procedure requires two parameters. The first being an object pointer of whatever type is being stored on the list. The second must be a LONGINT parameter. This is passed by the ForEach process and represents the base zero (0) index position of current data object within the list (i.e. somewhere between '0' to 'Count-1').

```
VAR
    AList : TObjectContainer;
    ATotal : LONGINT

    { ++++++ }

    PROCEDURE DoTest(AnObject : TMyObject; Idx : LONGINT); FAR;
    BEGIN
        ...
    END;

    { ++++++ }

BEGIN
    AList := TObjectContainer.Create;
    .....
    .....
    ATotal := 0;
    AList.ForEachIdx(@DoTest);
    .....
    .....
```

TObjectContainer.LastThatIdx

FUNCTION LastThatIdx(Test: Pointer): Pointer;

This is a variation on the 'LastThat' method. With 'LastThatIdx' the nested local function requires two parameters. The first being an object pointer of whatever type is being stored on the list. The second must be a LONGINT parameter. This is passed by the LastThat process and represents the base zero (0) index position of current data object within the list (i.e. somewhere between '0' to 'Count-1'). For example ...

```
VAR
  AList : TObjectContainer;
  ATotal : LONGINT

  { ++++++ }

  FUNCTION DoTest(AnObject : TMyObject; Idx : LONGINT) : BOOLEAN; FAR;
  BEGIN
    ...

  END;

  { ++++++ }

BEGIN
  AList := TObjectContainer.Create;
  .....
  .....
  ATotal := 0;
  AList.LastThatIdx(@DoTest);
  .....
  .....
```

TStringClass Protected Methods

The object includes a number of protected methods. All the 'DoXXXXX' type methods are those that relate to processing strings based upon a character position. These have been introduced to allow for the safe processing of the 'StrClassBaseZero' related optional first character facility. Each of these 'DoXXXX' methods work using a base character position of zero (0). If you intend creating your own objects derived from TStringClass and having your own new methods that involve character position related processing, then to ensure correct processing use the 'DoXXX' method and set all calculations around a base zero first character.

Similarly, the 'InitDataMembers' function has been made virtual and moved to the protected section to allow for the initialisation of new data fields defined in descendant objects.

The 'AppendToObject' method has been virtual to allow for situations where a developer wants to allow the passing of the text associated with custom VCL classes into the string class object.

```
FUNCTION AppendToObject(T : TObject) : PChar; VIRTUAL;
PROCEDURE InitDataMembers; VIRTUAL;
FUNCTION DoAssignFrom(CONST Args : ARRAY OF CONST; Start : WORD) : PChar;
FUNCTION DoAssignMid(CONST Args : ARRAY OF CONST;
                    Start,Count : WORD) : PChar;
FUNCTION DoAppendMid(CONST Args : ARRAY OF CONST;
                    Start,Count : WORD) : PChar;
FUNCTION DoFirstNonSpaceCh(VAR ACh : CHAR) : WORD;
FUNCTION DoHasCh(ACh : CHAR; VAR Pos : WORD) : BOOLEAN;
FUNCTION DoIsCh(W : WORD; ACh : CHAR) : BOOLEAN;
FUNCTION DoLastNonSpaceCh(VAR ACh : CHAR) : WORD;
PROCEDURE DoSetCh(W : WORD; ACh : CHAR);
FUNCTION DoDelete(Index,Count : WORD) : PChar;
FUNCTION DoDeleteFrom(Index : WORD) : PChar;
FUNCTION DoInsert(CONST Args : ARRAY OF CONST; Index : WORD) : PChar;
FUNCTION DoInsertL(CONST Args : ARRAY OF CONST; Len,Index : WORD) : PChar;
FUNCTION DoFindBetween2Ch(FirstCh,
                          SecondCh      : CHAR;
                          StartFrom     : WORD;
                          VAR SubStrStart : WORD;
                          VAR SubStrLen  : WORD;
                          CutSubStr,
                          IncDelims     : BOOLEAN;
                          VAR ASubStr   : TStringClass) :
BOOLEAN;
FUNCTION DoFindFirst(CONST SubArgs : ARRAY OF CONST;
                    VAR P : WORD) : BOOLEAN;
FUNCTION DoFindFirstCh(ACh : CHAR; VAR P : WORD) : BOOLEAN;
FUNCTION DoFindLast(CONST SubArgs : ARRAY OF CONST;
                    VAR P : WORD) : BOOLEAN;
FUNCTION DoFindLastCh(ACh : CHAR; VAR P : WORD) : BOOLEAN;
FUNCTION DoFindNext(CONST SubArgs : ARRAY OF CONST;
                    StartPos : WORD;
                    VAR NextPos : WORD) : BOOLEAN;
FUNCTION DoFindNextCh(ACh : CHAR; StartPos : WORD;
                    VAR NextPos : WORD) : BOOLEAN;
FUNCTION DoFindPrev(CONST SubArgs : ARRAY OF CONST;
```

```
                StartPos : WORD; VAR PrevPos : WORD) : BOOLEAN;
FUNCTION DoFindPrevCh(ACh : CHAR; StartPos : WORD;
                VAR PrevPos : WORD) : BOOLEAN;
FUNCTION DoFirstParseDelim(CONST Args : ARRAY OF CONST;
                DelimCh      : CHAR;
                VAR DelimPos : WORD) : BOOLEAN;
FUNCTION DoNextParseDelim(CONST Args : ARRAY OF CONST;
                DelimCh      : CHAR;
                StartPos     : WORD;
                VAR NextDelimPos : WORD) : BOOLEAN;
```

TStringClass : Properties

All but one of the properties are read only types.

PROPERTY Ch[w : WORD] : CHAR READ GetCh;

Returns the CHAR type at position 'w' within the object's text string. (Remember 'w' should be calculated from base 0. e.g. the 2nd character would be position 1). If the 'w' value exceeds the length of the object text string an exception error is raised.

Note: If the typed constant 'StrClassBaseZero' is set to FALSE then character position starts from one (1) not zero (0).

PROPERTY Length : WORD READ GetLength;

Returns the current object text string length (excluding the null terminator). Thus, a string of 'ABC' would have a length of 3.

PROPERTY MaxSize : WORD READ GetMaxSize;

Returns the current buffer size. This may be larger than the object text string length.

PROPERTY SizeInc : WORD READ GetSizeInc WRITE SetSizeInc;

Used to set (or read) the minimum amount in bytes by which the buffer size will increase the next time it is required to expand. If a string object anticipates receiving multiple 'appends' then it is worth using the 'SizeInc' property to set a large incremental jump in the internal buffer size.

PROPERTY Text : STRING READ GetString;

Returns the object text string as a STRING type variable. If the object text string variable length exceeds 255 characters then an exception error is raised.

PROPERTY ZString : PChar READ GetPChar;

Returns a 'PChar' type to the object's own text string buffer.

Copyright Notice

No liability can be accepted for use of this source code.

This source code remains the copyright of ICFM Software. You may use it in your own applications, but cannot employ it as part of another software library without the express permission of ICFM Software.

Shareware Registration

Please use the Compuserve Shareware Registration service to register your use of this source code library. Whilst using Compuserve '**GO SWREG**'. Search for '**STRING**' to find this shareware offering.

Shareware Registration costs just **US\$35** and gains full access to all the source code used within the TStringClass and object container classes. It also includes six months free updates.

For further details please contact ICFM Software via our Compuserve account : [**100010,1415**].

