fTStringClass
Version 2.01

A PChar based string management object for Borland's DELPHI Windows development environment

---

ICFM Software,
London,UK
(Email: Compuserve 100010,1415)

ICFM SOFTWARE
LIBRARY DOCUMENTATION

Date: 5th November,1995

Re: **STRCLASS.PAS - The Pascal PChar based string object**

Contents

1. Introduction

2. Working with TStringClass objects

3. Character Positions

4. Parameters

5. Working with VCL text components

6. Pushing data directly into the class text buffer

7. Using the Global string class variable 'gSc'

8. ExceptionHandling

9. Limitations

10. Bug fixes, Ideas etc

11. Copyright Notice

12. TStringClass - Object Definition

12.1 Data Members

12.2 Private Methods

12.3 Protected Methods

12.4 Properties

12.5 Public Methods

12.5.1 Constructors/Destructors
12.5.2 Copy related
12.5.3 Clear type functions
12.5.3
12.5.4 Admin type functions
12.5.5 Assign related functions
12.5.6 Append related functions
12.5.7 Character element related functions
12.5.8 Other data type related functions
12.5.9 SysUtils unit compatible functions
12.5.10 Strings unit compatible functions
12.5.11 Comparison related functions
12.5.12 Insert/Delete/Trim related functions
12.5.12
12.5.13 Command Line related functions
12.5.14 Resource string related functions
12.5.15 INI file related functions

12.5.16 DOS path/filename related functions
12.5.17 Search related functions
12.5.18 Case related functions
12.5.19 Search & replace related functions
12.5.20 Parsing related functions

13. Container Objects

13.1 TBaseContainer
13.1.1 Public methods
13.1.2 Properties

13.2 TObjectContainer
13.2.1 Public methods
13.2.2 Properties

13.3 TRecordContainer
13.3.1 Public methods

13.4 TPCharContainer = CLASS(TObjectContainer)

13.5 TIntegerContainer = CLASS(TBaseContainer)
13.5.1 Public methods
13.5.2 Properties

13.6 TWordContainer = CLASS(TBaseContainer)
13.6.1 Public methods
13.6.2 Properties

13.7 TLongIntContainer = CLASS(TBaseContainer)
13.7.1 Public methods
13.7.2 Properties

13.8 TCustomTypeContainer = CLASS(TBaseContainer)
13.8.1 Public methods

14. Version Management

15. Index

1Introduction

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-intialised 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.

2Working 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 empoying any of these container related functions make sure the ‘ContainR’ unit is included in the ‘USES’ list as shown above.

3Character 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.

4Parameters

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 2ndt 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,wwww,,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 }
```

5Working 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.

6Pushing 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.

7Using the Global string class variable 'gSc'

Another tiresome issue underlying the use of objects is the need to 'create' or instantiate objects prior to their use. For purists this gives any code a measure of self containment which ensures its own security. However, on a more practical level having to include …

```
VAR
    SObj : TStringClass;
BEGIN
    …..
    …..
    SObj := TStringClass.Create
    …..
    …..
```

…. requires a lot more typing, and also requires that the 'SObj' be disposed of at the end of the process.

As a convenience for more pragmatic users of objects the 'StrClass' unit includes a global variable 'gSc' (**G**lobal **S**tring **C**lass) as an already instantiated string class variable.

This variable can be used a convenient vehicle for string class functions at any time. However, do not dispose of this variable. This is taken care of by the unit itself.

8ExceptionHandling

The TStringClass object makes extensive use of exception handling. It has it 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!)

9Limitations

The string class will only manage strings of up to 64K in size. If an object's string exceeds this size an exception error is raised. The container objects used within the parsing functions are limited to having no more than 2 billion items on a list.

10Bug fixes, Ideas etc

If you detect any bugs or would like to suggest ideas for improvements then please email them to my Compuserve account [100010,1415].

11Copyright 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.

12TStringClass - Object Definition

12.1Data 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 'TheLength' 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.

12.2Private Methods

Most of the private methods are concerned with maintaining the PChar string buffer, or converting other data types into PChar types. They are listed here for completeness:

```
FUNCTION AssignFromPChar(Source : PChar; Start : WORD) : PChar;
FUNCTION AssignMidPChar(Source : PChar; Start,Count : WORD) : PChar;
FUNCTION AssignLenPChar(Source : PChar; Len : WORD) : PChar;
FUNCTION AssignNLPChar(Source : PChar) : PChar;
FUNCTION AssignPadPChar(Source : PChar; Len : WORD; ACh : CHAR) : PChar;
FUNCTION AssignPChar(Source : PChar) : PChar;
FUNCTION AssignRightPChar(Source : PChar; Len : WORD) : PChar;
FUNCTION AssignString(CONST Source : STRING) : PChar;
FUNCTION AppendLenPChar(Source : PChar; Len : WORD) : PChar;
FUNCTION AppendMidPChar(Source : PChar; Start,Count : WORD) : PChar;
FUNCTION AppendNLPChar(Source : PChar) : PChar;
FUNCTION AppendPadPChar(Source : PChar; Len : WORD; ACh : CHAR) : PChar;
FUNCTION AppendPChar(Source : PChar) : PChar;
FUNCTION AppendRightPChar(Source : PChar; Len : WORD) : PChar;
FUNCTION AppendString(Source : STRING) : PChar;
FUNCTION AppendTClass(T : TClass) : PChar;
FUNCTION AppendTObject(T : TObject) : PChar;
FUNCTION AppendTrimPChar(Source : PChar) : PChar;
FUNCTION AppendWithTabPChar(Source : PChar) : PChar;
FUNCTION PrependPChar(Source : PChar) : PChar;
FUNCTION ComparePChar(Other : PChar) : INTEGER;
FUNCTION CompareIPChar(Other : PChar) : INTEGER;
FUNCTION CompareLPChar(Other : PChar; Len : WORD) : INTEGER;
FUNCTION CompareLIPChar(Other : PChar; Len : WORD) : INTEGER;
FUNCTION InsertPChar(Source : PChar; Index : WORD) : PChar;
FUNCTION CanCat(Source : PChar; VAR Extra : WORD) : BOOLEAN;
FUNCTION CanCopy(Source : PChar; VAR Extra : WORD) : BOOLEAN;
```

```
FUNCTION ChkSizeInc(E : WORD) : WORD;
FUNCTION CompConv(I : INTEGER) : INTEGER;
PROCEDURE DisposeStr;
FUNCTION ExpandBy(ExtraLen : WORD) : BOOLEAN;
FUNCTION GetCh(w : WORD) : CHAR;
FUNCTION GetLength : WORD;
FUNCTION GetMaxSize : WORD;
FUNCTION GetPChar : PChar;
FUNCTION GetSizeInc : WORD;
FUNCTION GetString : STRING;
PROCEDURE SetBufferLen(NewLen : WORD);
PROCEDURE SetSizeInc(ASize : WORD);
```

12.3Protected 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 'AppendTObject' 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 AppendTObject(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;
```

12.4Properties

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.

12.5Public Methods

12.5.1Constructors/Destructors

CONSTRUCTOR Create;
Instantiates the object with a NIL default PChar value and zero length internal buffer. Example :

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

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);
```

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]);
    ......
```

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.

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

StrType constant	String conversion(TRUE/FALSE)
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.

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;
```

12.5.2Copy related

Two functions for copying a string object's attributes.

FUNCTION Copy : POINTER; VIRTUAL;
This function instantiates and returns a copy of itself.

PROCEDURE CopyFrom(Source : TStringClass);
Copies the attributes and text string buffer of the parameter object 'Source'.

12.5.3Clear type functions

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

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

12.5.4Admin type functions

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.

12.5.5Assign related functions

Assign related functions replace the object's existing text string value with the values passed within the function parameters.

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' }
```

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).

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.

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).

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.

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’ }
```

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’ }
```

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.

12.5.6Append related functions

A set of functions for appending other text strings to the end or front of the object’s own text string.

FUNCTION Append(CONST Args : ARRAY OF CONST) : Pchar;
Appends the string represented by ‘Args’ to the end of the object’s own text string.

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.

'bt' constant	String conversion(FALSE/TRUE)
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.

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.

FUNCTION AppendCh(C : CHAR) : PChar;
Appends the CHAR ‘C’ to the end of the object’s existing text string value.

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.

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.

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.

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.

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.

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.

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.

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).

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.

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’.

FUNCTION AppendPtr(P : POINTER) : Pchar;
Converts the POINTER parameter ‘P’ to a hexidecimal string format and appends it to the end of the object’s existing text string value.

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.

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.

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.

FUNCTION AppendTrim(CONST Args : ARRAY OF CONST) : Pchar;
The ‘Args’ parameter is converted to a string, trimmed of all leading and trailing spaces and then appended to the end of the object’s existing text string.

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.

FUNCTION NL : PChar;
Appends a ‘NewLine’ character combination to the end of the object text string.

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.

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.

12.5.7Character element related functions

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).

FUNCTION HasCh(Ch : CHAR; VAR Pos : WORD) : BOOLEAN;
Returns TRUE is 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).

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).

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.

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.

FUNCTION LastNonSpaceCh(VAR Ch : CHAR) : WORD;
Starting from the end of the object's string, this function returns the position (base 0) of the lright-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).

PROCEDURE RemoveLastCh;
This will remove the last character from the object string.

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).

12.5.8Other data type related functions

This set of functions provides a means of assigning or appending number type variables into the object string. In most cases the functions return a PChar pointer to the object's own string.

The ‘FromXXXX’ functions are provided for convenience, but the same result can be achieved in most cases by using the ‘Assign’ function with the required parameter type.

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.

'bt' constant	String conversion(FALSE/TRUE)
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.

FUNCTION FromByte(B : BYTE) : PChar;

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

FUNCTION FromChar(C : CHAR) : PChar;
Converts the CHAR 'Ch' to a string and assigns this to the object string.

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.

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.

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.

FUNCTION FromDoubleTrim(D : DOUBLE) : Pchar;
Converts the double number 'D' to a string with all leading white spaces and all trailing zeros removed.

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.

FUNCTION FromExtTrim(E : EXTENDED) : PChar;
Converts the extended number 'E' to a string with all leading white spaces and all trailing zeros removed.

FUNCTION FromLong(L : LONGINT) : PChar;
Converts the 'L' number to a string and assigns that string to the object string.

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.

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.

FUNCTION FromRealTrim(R : REAL) : PChar;
Converts the REAL number 'R' to a string with all leading white spaces and all trailing zeros removed.

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).

FUNCTION HexFromByte(B : BYTE) : PChar;
Converts the byte value 'B' into a hexidecimal string and assigns that string to the object.

FUNCTION HexFromLong(L : LONGINT) : PChar;
Converts the 'L' number into a hexidecimal string of type 'nnnn:nnnn' and assigns that string to the object.

FUNCTION HexFromPtr(P : POINTER) : PChar;
Converts the pointer 'P' into a hexidecimal string of type 'nnnn:nnnn' and assigns that string to the object.

FUNCTION HexFromWord(W : WORD) : PChar;
Converts the 'W' number into a hexidecimal string of type 'nnnn' and assigns that string to the object.

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.

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.

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'.

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.

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.

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.

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.

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.

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.

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).

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.

12.5.9SysUtils unit compatible functions

This section provides a set of object methods to mimic the string related functions found in the 'SysUtils' unit. In most cases the function result or target string has been omitted from the list of functions parameters.

These methods are quite useful if you are looking to quickly amend existing code with TStringClass based equivalent functions.

The use of any function like 'IsValidIndent that presumes a STRING type parameter will create an exception error if the object's text string is greater than 255 characters long.

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

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.

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.

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.

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.

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.

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.

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.

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.

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', '_'].

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.

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.

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.

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.

FUNCTION LoadStr(Ident: Word): Pchar;
LoadStr loads the string resource given by Ident from the application's executable file into the object's tetx buffer. If the string resource does not exist, an empty string is returned.

FUNCTION FmtLoadStr(Ident: Word; CONST Args: ARRAY OF CONST): Pchar;
FmtLoadStr loads a string from a program's resource string table and uses that sting, 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.

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.

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.

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.

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.

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.

12.5.10Strings unit compatible functions

This section provides a set of object methods to mimic the STRINGS unit functions. In most cases the 1st parameter of the equivalent STRINGS unit function has been omitted. Unlike the STRINGS unit functions these methods include error and parameter value testing for NIL and zero length strings. In most cases these methods return a PChar pointer to the object's string.

These methods are quite useful if you are looking to quickly amend existing PChar orientated source code with TStringClass based equivalent functions.

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

FUNCTION StrComp(Str2 : PChar) : INTEGER;
Compares the object's string with another string 'Str2'. The comparison is case sensitive. (Equivalent to the 'Compare' method).

FUNCTION StrCopy(Source : PChar) : PChar;
Copies the 'Source' string into the object string (equivalent to the 'Assign' method).

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.

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

FUNCTION StrIComp(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).

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).

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).

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).

FUNCTION StrLCopy(Str2 : PChar; MaxLen : WORD) : INTEGER;
Copies the first 'MaxLen' characters of 'Str2' into the object string (equivalent to the 'AssignLen' method).

FUNCTION StrLen : WORD;
Returns the length of the object string (equivalent to the 'GetLength' method).

FUNCTION StrLower : PChar;
Converts the object string to lower case (equivalent to the 'ToLower' method).

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

FUNCTION StrPas : STRING;
Returns the object PChar as a Pascal STRING variable.

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

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).

FUNCTION StrRScan(Chr : CHAR) : PChar;
Returns a PChar pointer to the right most location of the 'Chr' CHAR in the object string.

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.

FUNCTION StrUpper : PChar;
Converts the object string to upper case (equivalent to the 'ToUpper' method).

12.5.11Comparison related functions

A set of methods for comparing the object string with another string. For functions that return INTEGER results the values are interpreted as:

Result type	Result	Constant Id
Object string less than other string	-1	Compare_LT
Object string equal to other string	0	Compare_EQ
Object string greater than other string	1	Compare_GT

The StrClass.Pas unit includes three 'Compare_XX' constants that map to the -1/0/1 results to aide source code legibility.

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.

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.

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.

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.

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.

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.

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.

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);
```

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);
```

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);
```

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);
```

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.

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.

12.5.12Insert/Delete/Trim related functions

Most of these functions return a PChar pointer to the object's own string.

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.

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.

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).

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).

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).

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).

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.

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.

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.

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.

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.

FUNCTION Trim : PChar;
Removes all leading and trailing spaces from the object string.

FUNCTION TrimEnd : PChar;
Removes all trailing spaces from the object string.

FUNCTION TrimFront : PChar;
Removes all leading spaces from the object string.

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.

12.5.13Command Line related functions

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.

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.

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.

12.5.14Resource string related functions

FUNCTION AppendStringRes(Instance : THandle; Id : WORD) : Pchar;
Loads the string resource 'Id' and appends it to the end of the existing object text string.

FUNCTION LoadStringRes(Instance : THandle; Id : WORD) : Pchar;
Assigns to the object the string loaded from the task resources that has an identifier of 'Id'.

12.5.15INI file related functions

A small set of methods to help with loading string resources and INI file settings.

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.

FUNCTION WriteIniKeyword(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.

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.

12.5.16DOS path/filename related functions

A set of methods for helping with DOS path and file name string processing. Most functions return a PChar pointer to the object's own string.

FUNCTION AddBackSlash : PChar;
Adds a '\' back slash to the end of the object string.

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.

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

Builds a validated path name from the three components.

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.

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.

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.

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.

FUNCTION ExpandFileName : Pchar;
Converts a DOS path file into a full drive, path and file name, removing any '..' type re-directions.

FUNCTION FileExists : BOOLEAN;
Returns TRUE if the object string representing a DOS path name exists as a DOS file.

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;
```

FUNCTION FindCurrentDir : PChar;
Copies the current DOS directory path name into the object text string.

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' }
```

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

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

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

FUNCTION HasBackSlash : BOOLEAN;
Returns TRUE if the last character of the object string is a '\' back slash.

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

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.

FUNCTION HasFileName : BOOLEAN;
Returns TRUE if the object string has a file name component.

FUNCTION HasDirectory : BOOLEAN;
Returns TRUE iof the object string has a directory component.

FUNCTION JustDirectory(CONST Args : ARRAY OF CONST) : Pchar;
Extracts from the 'Args' parameter the directory component and assigns this to the object string.

FUNCTION JustExtension(CONST Args : ARRAY OF CONST) : PChar;
Extracts from the 'Args' parameter the extension component and assigns this to the object string.

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.

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.

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.

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:\')

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 }
```

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.

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.

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 }
```

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.

12.5.17Search related functions

A set of methods that search the object string for occurrences of sub strings or characters.

FUNCTION ChCount(ACh : CHAR) : WORD;
Returns the number of occurrences of CHAR 'Ch' in the object string.

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 netx 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).

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).

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).

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).

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).

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).

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).

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).

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).

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 asterix ('*') 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.

FUNCTION SubStrCount(CONST SubArgs : ARRAY OF CONST) : WORD;
Returns the total number of occurrences that sub string 'SubArgs' is located within the object string.

12.5.18 Case related functions

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;
WriteIn(Str1.ZString);              { This Is A Test }
```

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’).

FUNCTION ToLower : PChar;
Converts the whole object string to lower case.

FUNCTION ToUpper : Pchar;
Converts the whole object string to upper case.

12.5.19 Search & replace related functions

Most of these methods return a PChar pointer to the object string.

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’.

FUNCTION ReplaceChAll(OldCh,NewCh : CHAR) : PChar;
Replaces every occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

FUNCTION ReplaceChFirst(OldCh,NewCh : CHAR) : PChar;
Replaces the first occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

FUNCTION ReplaceChLast(OldCh,NewCh : CHAR) : Pchar;
Replaces the last occurrence of the CHAR 'OldCh' with the replacement CHAR 'NewCh'.

FUNCTION ReplaceFirst(CONST OldArgs,NewArgs : ARRAY OF CONST) : PChar;
Replaces the first 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 'OldArgs'.

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'.

12.5.20Parsing related functions

This set of functions provides a variety of methods for parsing or splitting text strings into a number of parts. Parsing operations place parsed sub strings into their own string class. A container class ‘TObjectContainer’ is used to hold multiple sub strings. (Refer to section 9 for an explanation of container objects).

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).

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).

FUNCTION ParseDelimCount(DelimCh : CHAR) : WORD;

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

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.

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’.

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' psarameter musty 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 extracetd 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.

13Container Objects

For certain of the parsing related methods the TStringClass object uses container objects to hold lists of parsed sub strings.

These container objects are of the type ‘TObjectContainer’.

For those who used Borland Pascal for Windows the hierarchy of container objects defined in ‘ContainR.Pas’ will have a familiar ring. They are based largely upon the TCollection object used in the OWL application framework.

For those unfamiliar with OWL, a TCollection object was a type of open ended array into which items could be added, inserted or deleted.

I’ve taken the basic TCollection concept and adapted this for use in the DELPHI world. The differences include:

- The container objects can manage open ended arrays of up to 2,147,483,647 items.
- The container object hierarchy includes objects for holding basic data types i.e. large arrays of integers, objects, and records.
- It makes extensive use of the Property aspect of object definition.

13.1TBaseContainer

All container objects derive from the abstract class ‘TBaseContainer’ .

13.1.1Public methods

CONSTRUCTOR Create; VIRTUAL;
Creates a new object.

DESTRUCTOR Destroy; OVERRIDE;
Destroys the open array list and all its list items

PROCEDURE Clear; VIRTUAL;
Clears all items from the list, disposing of any records/objects on the list.

PROCEDURE Delete(Idx : LONGINT); VIRTUAL;
Deletes the Idx item (base 0) from the list. The item itself is not disposed of.

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.

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.

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

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

PROCEDURE Move(CurIdx, NewIdx: LONGINT); VIRTUAL;

Moves the item ‘CurIdx’ to the new position ‘NewIdx’.

PROCEDURE Pack; VIRTUAL;
Scans the list to remove any null item pointers.

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

13.1.2Properties

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-organisiing.

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.

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

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.

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 there 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.

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 ono-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.

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 overriden 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;
```

13.2TObjectContainer

An open ended array type container for holding object pointers.

13.2.1Public methods

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

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 positioned based upon the objects ‘Compare’ result.

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

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

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.

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);
.....
.....
```

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. The 'Action' procedure must be nested procedure 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.

This method is useful for counting items data elements of data objects on a list, for example :

```
VAR
        AList : TObjectContainer;
        TotalAge : LONGINT;

        { ++++++++++++++++++++++++++++++++++++++++++++++++++++ }

        PROCEDURE DoAgeCount(ThisDataObj : TDataObject; Idx : LONGINT); FAR;
        BEGIN
          INC(TotalAge,ThisDataObj.Age);
        END;

        { ++++++++++++++++++++++++++++++++++++++++++++++++++++ }

BEGIN
  ….
  …..

  AList := TObjectContainer.Create;
  AList.Capacity := 100;
  { params - initials, surname, age and salary }
  ADataObj := TDataObject.CreateInit('J','Smith',34,10500.45);
  AList.Add(ADataObj);
  ADataObj := TDataObject.CreateInit('F','Brown',40,17453.89);
  AList.Add(ADataObj);
  ADataObj := TDataObject.CreateInit('A','Jones',23,12765.34);
  AList.Add(ADataObj);
  ADataObj := TDataObject.CreateInit('W','Bloggs',19,23456.21);
  AList.Add(ADataObj);
  TotalAge := 0;
  { count total ages }
  AList.ForEach(@DoAgeCount);
```

The 'DoAgeCount' local nested procedure is called for each data object on he list, thus allowing 'TotalAge' to accumalate a total.

PROCEDURE ForEachIdx(Action: Pointer): Pointer;
This is a variation on the 'ForEach' method described above. 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);
        .....
        .....
```

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'

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

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.

FUNCTION Last: POINTER;
Returns a pointer to the last object on the list.

FUNCTION LastThat(Test: Pointer): Pointer;
Similar to the 'FirstThat' function this iterates across the list from last to first.

FUNCTION LastThatIdx(Test: Pointer): Pointer;
This is a variation on the 'LastThat' method described above. 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);
        .....
        .....
```

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

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

13.2.2Properties

PROPERTY Items[Idx : 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 Idx.

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.

13.3TRecordContainer

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.

13.3.1Public methods

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

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

13.4TPCharContainer = CLASS(TObjectContainer)

A container object for holding lists of ‘PChar’ variable types.

13.5TIntegerContainer = CLASS(TBaseContainer)

A container for holding lists of integers.

13.5.1Public methods

CONSTRUCTOR Create;
Creates a container object to hold integer items.

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.

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

FUNCTION First: INTEGER; VIRTUAL;
Returns the integer value of the first item on the list.

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.

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

FUNCTION Last: INTEGER;
Returns the integer value of the last item on the list.

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

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

13.5.2Properties

PROPERTY Items[Idx : 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.

13.6TWordContainer = CLASS(TBaseContainer)

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.

13.6.1Public methods

CONSTRUCTOR Create;
FUNCTION Add(Item: WORD): LONGINT; VIRTUAL;
FUNCTION Append(Item : WORD) : LONGINT; VIRTUAL;
FUNCTION First: WORD; VIRTUAL;

FUNCTION Includes(Item : WORD; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION Insert(Idx: LONGINT; Item: WORD) : LONGINT; VIRTUAL;
FUNCTION Last: WORD;
FUNCTION Prepend(Item : WORD) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: WORD): LONGINT; VIRTUAL;

13.6.2Properties

PROPERTY Items[Idx : LONGINT]: WORD READ GetWord WRITE PutWord;

13.7TLongIntContainer = CLASS(TBaseContainer)

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.

13.7.1Public methods

CONSTRUCTOR Create;
FUNCTION Add(Item: LONGINT): LONGINT; VIRTUAL;
FUNCTION Append(Item : LONGINT) : LONGINT; VIRTUAL;
FUNCTION First: LONGINT; VIRTUAL;
FUNCTION Includes(Item : LONGINT; VAR Idx : LONGINT) : BOOLEAN;
FUNCTION Insert(Idx: LONGINT; Item: LONGINT) : LONGINT; VIRTUAL;
FUNCTION Last: LONGINT;
FUNCTION Prepend(Item : LONGINT) : LONGINT; VIRTUAL;
FUNCTION Remove(Item: LONGINT): LONGINT; VIRTUAL;

13.7.2Properties

PROPERTY Items[Idx : LONGINT]: LONGINT READ GetLongInt WRITE PutLongInt;

13.8TCustomTypeContainer = CLASS(TBaseContainer)

A container for holding items of user defined length. Note that for this container an item is passed as undefined variable type. This presumes that a valid record/variable is passed, NOT a pointer to the record or variable.

13.8.1Public methods

CONSTRUCTOR Create(CustomTypeLen : WORD);
Creates a container object to hold items of length 'CustomTypeLen' bytes

FUNCTION Add(VAR Item): LONGINT; VIRTUAL;
Adds the custom type to the list.

FUNCTION Append(VAR Item) : LONGINT; VIRTUAL;
Adds the custom item to the end of the list.

FUNCTION Insert(Idx: LONGINT; VAR Item) : LONGINT; VIRTUAL;
Inserts the custom item at position Idx.

FUNCTION Prepend(VAR Item) : LONGINT; VIRTUAL;
Asdds the custom item to the front of the list.

FUNCTION Retrieve(Idx : LONGINT; VAR Item) : BOOLEAN;
Retrieves the custom item at position Idx into the variable Item.

14Version Management

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 lead 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 returns 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

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.

TObjectContainer: new methods :

FirstThatIdx
ForEachIdx
LastThatIdx

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 'dupIgnore', this method was still reporting an exception error. This has been changed so that with 'dupIgnore' 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.

15Index

A

Add, 7, 29, 31, 39, 40, 45, 50, 51, 54
AddBackSlash, 31
AddDIC, 29
AddFilterDesc, 31
All_Items, 22
AnsiCompareStr, 25
AnsiCompareText, 25
AnsiLowerCase, 24
AnsiUpperCase, 24
Append, 18, 45, 50, 51
AppendBoolean, 18
AppendByte, 19
AppendCh, 19
AppendDIC, 19
AppendDouble, 19
AppendDoubleTrim, 19
AppendExt, 19
AppendExtTrim, 19
AppendLen, 19
AppendLong, 19
AppendMid, 19
AppendNL, 20
AppendPad, 20
AppendPtr, 20
AppendReal, 20
AppendRight, 20
AppendSIC, 20
AppendStr, 24
AppendStringRes, 30
AppendTObject, 13
AppendTrim, 20
AppendWithTab, 20
Assign, 16
AssignFrom, 17
AssignLen, 17
AssignMid, 17
AssignNL, 18
AssignPad, 18
AssignRight, 18
AssignTrim, 18

B

BuildPathName, 32

C

Capacity, 43, 53
Ch, 14
ChCount, 34
Clear, 16, 42
Compare, 28
Compare_EQ, 27, 28
Compare_GT, 27
Compare_LT, 27, 28
CompareDouble, 28
CompareExt, 28
CompareFunc, 43, 53
CompareI, 28
CompareL, 28
CompareLI, 28
CompareLong, 28
CompareStr, 24
CompareText, 24
ContainR, 5
Copy, 16
CopyFrom, 16
Count, 44
Create, 5, 6, 7, 8, 9, 10, 15, 16, 17, 18, 32, 34, 36, 39, 40, 42, 43, 44, 45, 46, 48, 49, 50, 51
CreateBoolean, 15
CreateDirectory, 32
CreateNL, 15
CreateSize, 15
CreateString, 15

D

DefaultExtension, 32
Delete, 13, 29, 42, 53
DeleteAll, 42
DeleteBlock, 42
DeleteFrom, 29
delim_ExcDIC, 38
delim_ExcNull, 39, 40, 55
delim_ExcSIC, 39
delim_IncDIC, 38, 54
delim_IncSIC, 38, 54
delim_None, 8, 38
Delta, 44
Destroy, 16, 42, 55
DirectoryExists, 32
DriveExists, 32
Duplicates, 44

E

EConvertError, 25
Empty, 16
EStringClass, 10
Exchange, 42
ExpandFileName, 32

F

FBuffer, 12, 16
FileExists, 32
FileSplit, 32
FindBetween2Ch, 34
FindCmdLine, 30, 53
FindCmdLineAndParse, 30
FindCmdLineParam, 30, 53

FindCurrentDir, 32
FindFirst, 34
FindFirstCh, 35
FindIniSectionKeywords, 31
FindLast, 35
FindLastCh, 35
FindNext, 35
FindNextCh, 35
FindPrev, 35
FindPrevCh, 35
FindRelPath, 32, 53
First, 45, 50, 51
FirstCharToUpper, 37
FirstNonSpaceCh, 20
FirstParseDelim, 38
FirstThat, 46
FirstThatIdx, 46
FLength, 12, 16
FloatToStr, 25
FloatToStrF, 25
FMaxSize, 12
FmtLoadStr, 25
ForceExtension, 33
ForEach, 47
ForEachIdx, 47
Format, 25
FormatFloat, 26
FromBoolean, 21
FromByte, 22
FromChar, 22
FromComponent, 22
  Working with VCL, 8
FromComponent', 22
FromComponentItem, 22
  Working with VCL, 9
FromDouble, 22
FromDoubleTrim, 22
FromExt, 22
FromExtTrim, 22
FromLong, 22
FromPtr, 22
FromReal, 22
FromRealTrim, 23
FromRGB, 23
fs_Directory, 32
fs_extension, 32
fs_Name, 32
FSizeInc, 12

G

GetSystemDirectory, 33
GetWindowsDirectory, 33
Grep, 36
GrowAsRequired, 44
gSc, 10

H

HasBackSlash, 33
HasCh, 21
HasDirectory, 33
HasDrive, 33
HasExtension, 33
HasFileName, 33
HexFromByte, 23
HexFromLong, 23
HexFromPtr, 23
HexFromWord, 23

I

Includes, 29, 48, 50, 51
IndexOf, 48
InitDataMembers, 13
Insert, 29, 48, 50, 51
InsertBlock, 42
InsertL, 29
IntToHex, 25
IntToStr, 25
IsAlphaNumeric, 37
IsCh, 21
IsFirstCh, 21
IsLastCh, 21
IsSame, 28
IsSameI, 28, 53
IsSameL, 28
IsSameLI, 28
IsValidIdent, 25
Items, 49, 50, 51

J

JustDirectory, 33, 53
JustExtension, 33
JustFileName, 33
JustName, 33

L

Last, 48, 50, 51
LastNonSpaceCh, 21
LastThat, 48
LastThatIdx, 48
Length, 14
LoadStr, 25
LoadStringRes, 31
LowerCase, 24

M

MaxSize, 14
Move, 42, 54

N

NextParseDelim, 38
NL, 20
NLAppend, 20

O

Own, 49

P

Pack, 43
PadCentre, 29
PadEnd, 30
PadFront, 30
ParseDelimCount, 38
ParseDelimToList, 38, 54
ParseMultiDelimToList, 40
ParsePosToList, 40
Prepend, 20, 49, 50, 51, 52

R

ReadIniKeyword, 31, 53
RecalcLength, 9, 10, 16, 55
Remove, 49, 50, 51
RemoveAll, 43
RemoveDIC, 30
RemoveDirectory, 33
RemoveLastCh, 21
RemoveSIC, 30
ReplaceAll, 37
ReplaceChAll, 37
ReplaceChFirst, 37
ReplaceChLast, 37
ReplaceFirst, 37
ReplaceLast, 38
Retrieve, 52

S

SameDirectory, 33
SameDrive, 33
SameExtension, 34
SameFileName, 34
SameName, 34
Selected_text, 22
SetCapacity, 54
SetCh, 21
SetCurDir, 34
SizeInc, 14
Sort, 44
SortInPlace, 54
StrCat, 26
StrClass, 5
StrClass.Res, 10
StrClassBaseZero, 6, 9, 13, 14, 17, 18, 19, 20, 21, 29, 34, 35, 36, 38, 40, 55
  Working with VCL, 9
StrClassBaseZero', 6
StrComp, 26
StrCopy, 26
StrECopy, 26
StrEnd, 26
StrIComp, 26
STRINGS, 26
StrLCat, 26
StrLComp, 27
StrLCopy, 27
StrLen, 27
StrLIComp, 26
StrLower, 27
StrMove, 27
StrPas, 27
StrPCopy, 27
StrPos, 27
StrRScan, 27
StrScan, 27
StrToFloat, 26
StrToInt, 25
StrToIntDef, 25
StrUpper, 27
SubStrCount, 37
SysUtils, 4, 24

T

TCollection, 42
TCompareFunc, 43, 44
TEdit, 8
Text, 14
TMemo, 8
TObjectContainer, 8
ToBoolean, 23
ToByte, 23
ToChar, 23
ToComponent, 23
  Working with VCL, 9
ToDouble, 23
ToExt, 23
ToLong, 23
ToLower, 37
ToReal, 24
ToRGB, 24
ToUpper, 37
ToWord, 24
Trim, 30
TrimEnd, 30
TrimFront, 30
TrimZero, 30
TWinControl, 8, 23

U

UpperCase, 24
USES, 5

W

Within, 29
WriteIniKeyword, 31

Z

zero based, 5
ZString, 9, 14