TStringClass

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: 16th August,1995

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

Contents

1. Introduction

4
2. Working with StringClass objects

5
3. Parameters

5
4. ExceptionHandling

7
5. Limitations

7
6. Bug fixes, Ideas etc

7
7. Copyright Notice

7
8. TStringClass - Object Definition

7
8.1 Data Members

8
8.2 Private Methods

8
8.3 Properties

9
8.4 Public Methods

9
8.4.1 Constructors/Destructors
9
8.4.2 Copy related
11
8.4.3 Clear type functions

11
8.4.4 Assign related functions
11
8.4.5 Append related functions
13
8.4.6 Character element related functions
15
8.4.7 Other data type related functions
15
8.4.8 SysUtils unit compatible functions
18
8.4.9 Strings unit compatible functions
19
8.4.10 Comparison related functions
21
8.4.11 Insert/Delete/Trim related functions
22
8.4.12 Command Line related functions
23
8.4.13 Resource string related functions
24
8.4.14 INI file related functions
24
8.4.15 DOS path/filename related functions
24
8.4.16 Search related functions
26
8.4.17 Case related functions
27
8.4.18 Search & replace related functions
27
8.4.19 Parsing related functions
28
9. Container Objects

29
9.1 TBaseContainer

29
9.1.1 Public methods
29
9.1.2 Properties
30
9.2 TObjectContainer

31
9.2.1 Public methods
31
9.2.2 Properties
32
9.3 TRecordContainer

32
9.3.1 Public methods
32
9.4 TPCharContainer = CLASS(TObjectContainer)

33
9.5 TIntegerContainer = CLASS(TBaseContainer)

33
9.5.1 Public methods
33
9.5.2 Properties
34
9.6 TWordContainer = CLASS(TBaseContainer)

34
9.6.1 Public methods
34
9.6.2 Properties
34
9.7 TLongIntContainer = CLASS(TBaseContainer)

34
9.7.1 Public methods
34
9.7.2 Properties
34
9.8 TCustomTypeContainer = CLASS(TBaseContainer)

34
9.8.1 Public methods
35

1.

Introduction

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

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

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

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

* Using un-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 asignments or concatenations it always checks to see that sufficient room is available. If there is insufficient space it resizes 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 TStringObject 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.

2. Working with TStringClass objects

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

```
unit Testbed;

interface

uses

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

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

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

```
VAR
      APChar : Pchar;
      AStr : STRING;
BEGIN
      ....
      ....
      GetMem(APChar,200);
      StrCopy(APChar,’Spring’);
      AStr := ‘String’;
      FirstCharacter :=  APChar[0];
      FirstCharacter := AStr[1];
```

3. Parameters

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

```
VAR

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

Rather than create a mass of different object methods to handle different parameter types, I decided to design TStringClass so that any type of parameter can be passed to a 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]);
    .......
    .......
```

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 hava 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 chapetr 9 for more information of the container object hierarchy.

4. ExceptionHandling

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!)

5. Limitations

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 there lists.

6. Bug fixes, Ideas etc

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

7. Copyright Notice

8.

TStringClass - Object Definition

8.1 Data Members

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

FBuffer : PChar;
This holds the object's PChar data value.

FMaxSize : WORD;
This stores the size of the 'FBuffer' PChar variable memory buffer.

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

8.2 Private Methods

Most of the private methods are concerned with maintaining the PChar string buffer, or converting other data types into Pchar types. They 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 InitDataMembers;
PROCEDURE SetBufferLen(NewLen : WORD);
PROCEDURE SetSizeInc(ASize : WORD);
```

8.3 Properties

All but one of the properties are read only types.

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

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 lenbgth 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.

8.4 Public Methods

8.4.1 Constructors/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 resizing.
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 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;
```

8.4.2 Copy 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'.

8.4.3 Clear type functions
PROCEDURE Clear;
Clears the internal buffer length to NIL and all other attributes to 0.

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

8.4.4 Assign 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.

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.

FUNCTION AssignNL(CONST Args : ARRAY OF CONST) : PChar;
As with 'Assign' this converts the 'Args' pararameter 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' pararameter 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' pararameter to a text string and assigns it to the object text buffer. If the resultant text string is then trimed to remove all leading and trailing space characters.

8.4.5 Append 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 converte 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.

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 NLAppend(CONST Args : ARRAY OF CONST) : Pchar;
Appends a ‘NewLine’ character comination 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.

8.4.6 Character 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).

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).

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.

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).

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'.

8.4.7 Other 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 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 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 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.

8.4.8 SysUtils 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 ommited 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 tetx 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.

8.4.9 Strings 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 ommitted. 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).

8.4.10 Comparison 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 senstiive.

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.

8.4.11 Insert/Delete/Trim related functions

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

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^.GetPChar);  {This test}
```

FUNCTION Insert(CONST Args : ARRAY OF CONST; Index : WORD) : PChar;
Inserts string 'Args' into the object string at position 'Index' (base 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).

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.

8.4.12 Command 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.

8.4.13 Resource 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 an identifier of 'Id'.

8.4.14 INI 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.

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

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

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 bettwen '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 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 existance.

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 existance.

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

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

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.

8.4.16 Search 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 retruned 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'.

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'.

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'.

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'.

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'.

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'.

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'.

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'.

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'.

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

8.4.17 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;
Writeln(Str1^.GetPChar);          { 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.

8.4.18 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 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'.

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'.

8.4.19 Parsing 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.

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.

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 }
```

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. 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.

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.

For each entry in the PosArray the method will create a bew 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 trhe end of the string the string object created fgor that entry is assignred 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.

9. Container 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 unfamilar 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 adapyted for you 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.

9.1 TBaseContainer

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

9.1.1 Public 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.

9.1.2 Properties

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 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;
If set to TRUE the list may store duplicates list entries (Only applies to sorted containers).

PROPERTY GrowAsRequired : BOOLEAN READ FGrowAsRequired WRITE FGrowAsRequired;
Using the 'Items' property found in descendant container objects a list item can be assigned to a specfic 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 grwo 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 contaimer requires that 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;
```

9.2 TObjectContainer

An open ended array type container for holding object pointers.

9.2.1 Public 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.

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

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

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

9.2.2 Properties

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.

9.3 TRecordContainer

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

9.3.1 Public 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));
```

9.4 TPCharContainer = CLASS(TObjectContainer)

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

9.5 TIntegerContainer = CLASS(TBaseContainer)

A container for holding lists of integers.

9.5.1 Public 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 lst 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.

9.5.2 Properties

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.

9.6 TWordContainer = CLASS(TBaseContainer)

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

9.6.1 Public 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;

9.6.2 Properties

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

9.7 TLongIntContainer = 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 LONGINTtype item parameters.

9.7.1 Public 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;

9.7.2 Properties

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

9.8 TCustomTypeContainer = 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.

9.8.1 Public 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.