Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 2.1
May 2007

Michaël Van Canneyt

Contents

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the FCL (Free Component Library).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures gave their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Free Component Library is a series of units that implemenent various classes and non-visual components for use with Free Pascal. They are building blocks for non-visual and visual programs, such as designed in Lazarus.

The `TDataset` descendents have been implemented in a way that makes them compatible to the Delphi implementation of these units. There are other units that have counterparts in Delphi, but most of them are unique to Free Pascal.

Chapter 1

Reference for unit 'base64'

1.1 Used units

Table 1.1: Used units by unit 'base64'

Name	Page
Classes	??
sysutils	??

1.2 Overview

base64 implements base64 encoding (as used for instance in MIME encoding) based on streams. it implements 2 streams which encode or decode anything written or read from it. The source or the destination of the encoded data is another stream. 2 classes are implemented for this: TBase64EncodingStream (22) for encoding, and TBase64DecodingStream (20) for decoding.

The streams are designed as plug-in streams, which can be placed between other streams, to provide base64 encoding and decoding on-the-fly...

1.3 Constants, types and variables

1.3.1 Types

```
TBase64DecodingMode = (bdmStrict,bdmMIME)
```

Table 1.2: Enumeration values for type TBase64DecodingMode

Value	Explanation
bdmMIME	MIME encoding
bdmStrict	Strict encoding

`TBase64DecodingMode` determines the decoding algorithm used by TBase64DecodingStream (20). There are 2 modes:

bdmStrict Strict mode, which follows RFC3548 and rejects any characters outside of base64 alphabet. In this mode only up to two '=' characters are accepted at the end. It requires the input to have a Size being a multiple of 4, otherwise an EBase64DecodingException (20) exception is raised.

bdmMime MIME mode, which follows RFC2045 and ignores any characters outside of base64 alphabet. In this mode any '=' is seen as the end of string, it handles apparently truncated input streams gracefully.

1.4 EBase64DecodingException

1.4.1 Description

`EBase64DecodeException` is raised when the stream contains errors against the encoding format. Whether or not this exception is raised depends on the mode in which the stream is decoded.

1.5 TBase64DecodingStream

1.5.1 Description

`TBase64DecodingStream` can be used to read data from a stream (the source stream) that contains Base64 encoded data. The data is read and decoded on-the-fly.

The decoding stream is read-only, and provides a limited forward-seek capability.

1.5.2 Method overview

Page	Property	Description
20	Create	Create a new instance of the `TBase64DecodingStream` class
21	Read	Read and decrypt data from the source stream
21	Reset	Reset the stream
22	Seek	Set stream position.
21	Write	Write data to the stream

1.5.3 Property overview

Page	Property	Access	Description
22	EOF	r	
22	Mode	rw	Decoding mode

1.5.4 TBase64DecodingStream.Create

Synopsis: Create a new instance of the `TBase64DecodingStream` class

Declaration:
```
constructor Create(AInputStream: TStream)
constructor Create(AInputStream: TStream;AMode: TBase64DecodingMode)
```

Visibility: public

Description: `Create` creates a new instance of the `TBase64DecodingStream` class. It stores the source stream `AInputStream` for reading the data from.

The optional `AMode` parameter determines the mode in which the decoding will be done. If omitted, `bdmMIME` is used.

See also: TBase64EncodingStream.Create (23), TBase64DecodingMode (19)

1.5.5 TBase64DecodingStream.Reset

Synopsis: Reset the stream

Declaration: `procedure Reset`

Visibility: public

Description: `Reset` resets the data as if it was again on the start of the decoding stream.

Errors: None.

See also: TBase64DecodingStream.EOF (22), TBase64DecodingStream.Read (21)

1.5.6 TBase64DecodingStream.Read

Synopsis: Read and decrypt data from the source stream

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` reads encrypted data from the source stream and stores this data in `Buffer`. At most `Count` bytes will be stored in the buffer, but more bytes will be read from the source stream: the encoding algorithm multiplies the number of bytes.

The function returns the number of bytes stored in the buffer.

Errors: If an error occurs during the read from the source stream, an exception may occur.

See also: TBase64DecodingStream.Write (21), TBase64DecodingStream.Seek (22), #rtl.classes.TStream.Read (**??**)

1.5.7 TBase64DecodingStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` always raises an `EStreamError` exception, because the decoding stream is read-only. To write to an encrypted stream, use a TBase64EncodingStream (22) instance.

Errors:

See also: TBase64DecodingStream.Read (21), TBase64DecodingStream.Seek (22), TBase64EncodingStream.Write (23), #rtl.classes.TStream.Write (**??**)

1.5.8 TBase64DecodingStream.Seek

Synopsis: Set stream position.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` sets the position of the stream. In the `TBase64DecodingStream` class, the `seek` operation is forward only, it does not support backward seeks. The forward seek is emulated by reading and discarding data till the desired position is reached.

For an explanation of the parameters, see TStream.Seek (**??**)

Errors: In case of an unsupported operation, an `EStreamError` exception is raised.

See also: TBase64DecodingStream.Read (21), TBase64DecodingStream.Write (21), TBase64EncodingStream.Seek (24), #rtl.classes.TStream.Seek (**??**)

1.5.9 TBase64DecodingStream.EOF

Synopsis:

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description:

1.5.10 TBase64DecodingStream.Mode

Synopsis: Decoding mode

Declaration: `Property Mode : TBase64DecodingMode`

Visibility: public

Access: Read,Write

Description: `Mode` is the mode in which the stream is read. It can be set when creating the stream or at any time afterwards.

See also: TBase64DecodingStream (20)

1.6 TBase64EncodingStream

1.6.1 Description

`TBase64EncodingStream` can be used to encode data using the base64 algorithm. At creation time, a destination stream is specified. Any data written to the `TBase64EncodingStream` instance will be base64 encoded, and subsequently written to the destination stream.

The `TBase64EncodingStream` stream is a write-only stream. Obviously it is also not seekable. It is meant to be included in a chain of streams.

1.6.2 Method overview

Page	Property	Description
23	Create	Create a new instance of the `TBase64EncodingStream` class.
23	Destroy	Remove a `TBase64EncodingStream` instannce from memory
23	Read	Read data from the stream
24	Seek	Position the stream
23	Write	Write data to the stream.

1.6.3 TBase64EncodingStream.Create

Synopsis: Create a new instance of the `TBase64EncodingStream` class.

Declaration:
```
constructor Create(AOutputStream: TStream)
```

Visibility: public

Description: `Create` instantiates a new `TBase64EncodingStream` class. The `AOutputStream` stream is stored and used to write the encoded data to.

See also: TBase64EncodingStream.Destroy (23), TBase64DecodingStream.Create (20)

1.6.4 TBase64EncodingStream.Destroy

Synopsis: Remove a `TBase64EncodingStream` instannce from memory

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes any remaining output and then removes the `TBase64EncodingStream` instance from memory by calling the inherited destructor.

Errors: An exception may be raised if the destination stream no longer exists or is closed.

See also: TBase64EncodingStream.Create (23)

1.6.5 TBase64EncodingStream.Read

Synopsis: Read data from the stream

Declaration:
```
function Read(var Buffer;Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Read` always raises an exception, because the encoding stream is write-only.

See also: TBase64EncodingStream.Write (23), TBase64EncodingStream.Seek (24), TBase64DecodingStream.Read (21), #rtl.classes.TStream.Read (**??**)

1.6.6 TBase64EncodingStream.Write

Synopsis: Write data to the stream.

Declaration:
```
function Write(const Buffer;Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: `Write` encodes `Count` bytes from `Buffer` using the Base64 mechanism, and then writes the encoded data to the destination stream. It returns the number of bytes from `Buffer` that were actually written. Note that this is not the number of bytes written to the destination stream: the base64 mechanism writes more bytes to the destination stream.

Errors: If there is an error writing to the destination stream, an error may occur.

See also: TBase64EncodingStream.Seek (24), TBase64EncodingStream.Read (23), TBase64DecodingStream.Write (21), #rtl.classes.TStream.Write (**??**)

1.6.7 TBase64EncodingStream.Seek

Synopsis: Position the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` always raises an `EStreamError` exception unless the arguments it received it don't change the current file pointer position. The encryption stream is not seekable.

Errors: An `EStreamError` error is raised.

See also: TBase64EncodingStream.Read (23), TBase64EncodingStream.Write (23), #rtl.classes.TStream.Seek (**??**)

Chapter 2

Reference for unit 'bufstream'

2.1 Used units

Table 2.1: Used units by unit 'bufstream'

Name	Page
Classes	**??**
sysutils	**??**

2.2 Overview

BufStream implements two one-way buffered streams: the streams store all data from (or for) the source stream in a memory buffer, and only flush the buffer when it's full (or refill it when it's empty). The buffer size can be specified at creation time. 2 streams are implemented: TReadBufStream (28) which is for reading only, and TWriteBufStream (29) which is for writing only.

Buffered streams can help in speeding up read or write operations, especially when a lot of small read/write operations are done: it avoids doing a lot of operating system calls.

2.3 Constants, types and variables

2.3.1 Constants

```
DefaultBufferCapacity : Integer = 16
```

If no buffer size is specified when the stream is created, then this size is used.

2.4 TBufStream

2.4.1 Description

`TBufStream` is the common ancestor for the TReadBufStream (28) and TWriteBufStream (29) streams. It completely handles the buffer memory management and position management. An in-

stance of TBufStream should never be created directly. It also keeps the instance of the source stream.

2.4.2 Method overview

Page	Property	Description
26	Create	Create a new TBufStream instance.
26	Destroy	Destroys the TBufStream instance

2.4.3 Property overview

Page	Property	Access	Description
26	Buffer	r	The current buffer
27	BufferPos	r	Current buffer position.
27	BufferSize	r	Amount of data in the buffer
27	Capacity	rw	Current buffer capacity

2.4.4 TBufStream.Create

Synopsis: Create a new TBufStream instance.

Declaration:
```
constructor Create(ASource: TStream;ACapacity: Integer)
constructor Create(ASource: TStream)
```

Visibility: public

Description: Create creates a new TBufStream instance. A buffer of size ACapacity is allocated, and the ASource source (or destination) stream is stored. If no capacity is specified, then DefaultBufferCapacity (25) is used as the capacity.

An instance of TBufStream should never be instantiated directly. Instead, an instance of TReadBufStream (28) or TWriteBufStream (29) should be created.

Errors: If not enough memory is available for the buffer, then an exception may be raised.

See also: TBufStream.Destroy (26), TReadBufStream (28), TWriteBufStream (29)

2.4.5 TBufStream.Destroy

Synopsis: Destroys the TBufStream instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: Destroy destroys the instance of TBufStream. It flushes the buffer, deallocates it, and then destroys the TBufStream instance.

See also: TBufStream.Create (26), TReadBufStream (28), TWriteBufStream (29)

2.4.6 TBufStream.Buffer

Synopsis: The current buffer

Declaration:
```
Property Buffer : Pointer
```

Visibility: public

Access: Read

Description: `Buffer` is a pointer to the actual buffer in use.

See also: TBufStream.Create (26), TBufStream.Capacity (27), TBufStream.BufferSize (27)

2.4.7 TBufStream.Capacity

Synopsis: Current buffer capacity

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the amount of memory the buffer occupies. To change the buffer size, the capacity can be set. Note that the capacity cannot be set to a value that is less than the current buffer size, i.e. the current amount of data in the buffer.

See also: TBufStream.Create (26), TBufStream.Buffer (26), TBufStream.BufferSize (27), TBufStream.BufferPos (27)

2.4.8 TBufStream.BufferPos

Synopsis: Current buffer position.

Declaration: `Property BufferPos : Integer`

Visibility: public

Access: Read

Description: `BufPos` is the current stream position in the buffer. Depending on whether the stream is used for reading or writing, data will be read from this position, or will be written at this position in the buffer.

See also: TBufStream.Create (26), TBufStream.Buffer (26), TBufStream.BufferSize (27), TBufStream.Capacity (27)

2.4.9 TBufStream.BufferSize

Synopsis: Amount of data in the buffer

Declaration: `Property BufferSize : Integer`

Visibility: public

Access: Read

Description: `BufferSize` is the actual amount of data in the buffer. This is always less than or equal to the Capacity (27).

See also: TBufStream.Create (26), TBufStream.Buffer (26), TBufStream.BufferPos (27), TBufStream.Capacity (27)

2.5 TReadBufStream

2.5.1 Description

`TReadBufStream` is a read-only buffered stream. It implements the needed methods to read data from the buffer and fill the buffer with additional data when needed.

The stream provides limited forward-seek possibilities.

2.5.2 Method overview

Page	Property	Description
28	Read	Reads data from the stream
28	Seek	Set location in the buffer
28	Write	Writes data to the stream

2.5.3 TReadBufStream.Seek

Synopsis: Set location in the buffer

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` sets the location in the buffer. Currently, only a forward seek is allowed. It is emulated by reading and discarding data. For an explanation of the parameters, see TStream.Seek" (**??**)

The seek method needs enhancement to enable it to do a full-featured seek. This may be implemented in a future release of Free Pascal.

Errors: In case an illegal seek operation is attempted, an exception is raised.

See also: TWriteBufStream.Seek (29), TReadBufStream.Read (28), TReadBufStream.Write (28)

2.5.4 TReadBufStream.Read

Synopsis: Reads data from the stream

Declaration: `function Read(var ABuffer;ACount: LongInt) : Integer;  Override`

Visibility: public

Description: `Read` reads at most `ACount` bytes from the stream and places them in `Buffer`. The number of actually read bytes is returned.

`TReadBufStream` first reads whatever data is still available in the buffer, and then refills the buffer, after which it continues to read data from the buffer. This is repeated untill `ACount` bytes are read, or no more data is available.

See also: TReadBufStream.Seek (28), TReadBufStream.Read (28)

2.5.5 TReadBufStream.Write

Synopsis: Writes data to the stream

Declaration: `function Write(const ABuffer;ACount: LongInt) : Integer;  Override`

Visibility: public

Description: `Write` always raises an `EStreamError` exception, because the stream is read-only. A TWriteBufStream (29) write stream must be used to write data in a buffered way.

See also: TReadBufStream.Seek (28), TReadBufStream.Read (28)

2.6 TWriteBufStream

2.6.1 Description

`TWriteBufStream` is a write-only buffered stream. It implements the needed methods to write data to the buffer and flush the buffer (i.e., write its contents to the source stream) when needed.

2.6.2 Method overview

Page	Property	Description
29	Destroy	Remove the `TWriteBufStream` instance from memory
30	Read	Read data from the stream
29	Seek	Set stream position.
30	Write	Write data to the stream

2.6.3 TWriteBufStream.Destroy

Synopsis: Remove the `TWriteBufStream` instance from memory

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes the buffer and then calls the inherited Destroy (26).

Errors: If an error occurs during flushing of the buffer, an exception may be raised.

See also: TBufStream.Create (26), TBufStream.Destroy (26)

2.6.4 TWriteBufStream.Seek

Synopsis: Set stream position.

Declaration:
```
function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override
```

Visibility: public

Description: `Seek` always raises an `EStreamError` exception, except when the seek operation would not alter the current position.

A later implementation may perform a proper seek operation by flushing the buffer and doing a seek on the source stream.

Errors:

See also: TWriteBufStream.Write (30), TWriteBufStream.Read (30), TReadBufStream.Seek (28)

2.6.5 TWriteBufStream.Read

Synopsis: Read data from the stream

Declaration: `function Read(var ABuffer;ACount: LongInt) : Integer;  Override`

Visibility: public

Description: `Read` always raises an `EStreamError` exception since `TWriteBufStream` is write-only. To read data in a buffered way, TReadBufStream (28) should be used.

See also: TWriteBufStream.Seek (29), TWriteBufStream.Write (30), TReadBufStream.Read (28)

2.6.6 TWriteBufStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const ABuffer;ACount: LongInt) : Integer;  Override`

Visibility: public

Description: `Write` writes at most `ACount` bytes from `ABuffer` to the stream. The data is written to the internal buffer first. As soon as the internal buffer is full, it is flushed to the destination stream, and the internal buffer is filled again. This process continues till all data is written (or an error occurs).

Errors: An exception may occur if the destination stream has problems writing.

See also: TWriteBufStream.Seek (29), TWriteBufStream.Read (30), TReadBufStream.Write (28)

Chapter 3

Reference for unit 'contnrs'

3.1 Used units

Table 3.1: Used units by unit 'contnrs'

Name	Page
Classes	**??**
sysutils	**??**

3.2 Overview

The contnrs implements various general-purpose classes:

Stacks Stack classes to push/pop pointers or objects

Object lists lists that manage objects instead of pointers, and which automatically dispose of the objects.

Component lists lists that manage components instead of pointers, and which automatically dispose the components.

Class lists lists that manage class pointers instead of pointers.

Stacks Stack classes to push/pop pointers or objects

Queues Classes to manage a FIFO list of pointers or objects

Hash lists General-purpose Hash lists.

3.3 Constants, types and variables

3.3.1 Constants

```
MaxHashListSize = Maxint div 16
```

MaxHashListSize is the maximum number of elements a hash list can contain.

```
MaxHashStrSize = Maxint
```

MaxHashStrSize is the maximum amount of data for the key string values. The key strings are kept in a continuous memory area. This constant determines the maximum size of this memory area.

```
MaxHashTableSize = Maxint div 4
```

MaxHashTableSize is the maximum number of elements in the hash.

```
MaxItemsPerHash = 3
```

MaxItemsPerHash is the threshold above which the hash is expanded. If the number of elements in a hash bucket becomes larger than this value, the hash size is increased.

3.3.2 Types

```
PHashItem = ^THashItem
```

PHashItem is a pointer type, pointing to the THashItem (33) record.

```
PHashItemList = ^THashItemList
```

PHashItemList is a pointer to the THashItemList (33). It's used in the TFPHashList (46) as a pointer to the memory area containing the hash item records.

```
PHashTable = ^THashTable
```

PHashTable is a pointer to the THashTable (33). It's used in the TFPHashList (46) as a pointer to the memory area containing the hash values.

```
TDataIteratorMethod = procedure(Item: Pointer;const Key: String;
                                var Continue: Boolean) of object
```

TDataIteratorMethod is a callback prototype for the TDataHashTable.Iterate (31) method. It is called for each data pointer in the hash list, passing the key (key) and data pointer (item) for each item in the list. If Continue is set to false, the iteration stops.

```
THashFunction = function(const S: String;const TableSize: LongWord)
                         : LongWord
```

THashFunction is the prototype for a hash calculation function. It should calculate a hash of string S, where the hash table size is TableSize. The return value should be the hash value.

```
THashItem = record
  HashValue : LongWord;
  StrIndex : Integer;
  NextIndex : Integer;
  Data : Pointer;
end
```

THashItem is used internally in the hash list. It should never be used directly.

```
THashItemList = Array[0..MaxHashListSize-1] of THashItem
```

THashItemList is an array type, primarily used to be able to define the PHashItemList (32) type. It's used in the TFPHashList (46) class.

```
THashTable = Array[0..MaxHashTableSize-1] of Integer
```

THashTable defines an array of integers, used to hold hash values. It's mainly used to define the PHashTable (32) class.

```
THTCustomNodeClass = Class of THTCustomNode
```

THTCustomNodeClass is used by THTCustomHashTable (31) to decide which class should be created for elements in the list.

```
THTNode = THTDataNode
```

THTNode is provided for backwards compatibility.

```
TIteratorMethod = TDataIteratorMethod
```

TIteratorMethod is used in an internal TFPHashTable (31) method.

```
TObjectIteratorMethod = procedure(Item: TObject;const Key: String;
                                  var Continue: Boolean) of object
```

TObjectIteratorMethod is the iterator callback prototype. It is used to iterate over all items in the hash table, and is called with each key value (Key) and associated object (Item). If Continue is set to false, the iteration stops.

```
TObjectListCallback = procedure(data: TObject;arg: pointer) of object
```

TObjectListCallback is used as the prototype for the TFPObjectList.ForEachCall (70) link call when a method should be called. The Data argument will contain each of the objects in the list in turn, and the Data argument will contain the data passed to the ForEachCall call.

```
TObjectListStaticCallback = procedure(data: TObject;arg: pointer)
```

TObjectListCallback is used as the prototype for the TFPObjectList.ForEachCall (70) link call when a plain procedure should be called. The Data argument will contain each of the objects in the list in turn, and the Data argument will contain the data passed to the ForEachCall call.

```
TStringIteratorMethod = procedure(Item: String;const Key: String;
                                  var Continue: Boolean) of object
```

TStringIteratorMethod is the callback prototype for the Iterate (40) method. It is called for each element in the hash table, with the string. If Continue is set to false, the iteration stops.

3.4 Procedures and functions

3.4.1 RSHash

Synopsis: Standard hash value calculating function.

Declaration: `function RSHash(const S: String;const TableSize: LongWord) : LongWord`

Visibility: default

Description: `RSHash` is the standard hash calculating function used in the TFPCustomHashTable (40) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: TFPCustomHashTable (40)

3.5 EDuplicate

3.5.1 Description

Exception raised when a key is stored twice in a hash table.

3.6 EKeyNotFound

3.6.1 Description

Exception raised when a key is not found.

3.7 TClassList

3.7.1 Description

`TClassList` is a Tlist (**??**) descendent which stores class references instead of pointers. It introduces no new behaviour other than ensuring all stored pointers are class pointers.

The `OwnsObjects` property as found in `TComponentList` and `TObjectList` is not implemented as there are no actual instances.

3.7.2 Method overview

Page	Property	Description
35	Add	Add a new class pointer to the list.
35	Extract	Extract a class pointer from the list.
36	First	Return first non-nil class pointer
36	IndexOf	Search for a class pointer in the list.
36	Insert	Insert a new class pointer in the list.
36	Last	Return last non-`Nil` class pointer
35	Remove	Remove a class pointer from the list.

3.7.3 Property overview

Page	Property	Access	Description
37	Items	rw	Index based access to class pointers.

3.7.4 TClassList.Add

Synopsis: Add a new class pointer to the list.

Declaration: `function Add(AClass: TClass) : Integer`

Visibility: public

Description: `Add` adds `AClass` to the list, and returns the position at which it was added. It simply overrides the TList (**??**) bevahiour, and introduces no new functionality.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TClassList.Extract (35), #rtl.classes.tlist.add (**??**)

3.7.5 TClassList.Extract

Synopsis: Extract a class pointer from the list.

Declaration: `function Extract(Item: TClass) : TClass`

Visibility: public

Description: `Extract` extracts a class pointer `Item` from the list, if it is present in the list. It returns the extracted class pointer, or `Nil` if the class pointer was not present in the list. It simply overrides the implementation in `TList` so it accepts a class pointer instead of a simple pointer. No new behaviour is introduced.

Errors: None.

See also: TClassList.Remove (35), #rtl.classes.Tlist.Extract (**??**)

3.7.6 TClassList.Remove

Synopsis: Remove a class pointer from the list.

Declaration: `function Remove(AClass: TClass) : Integer`

Visibility: public

Description: `Remove` removes a class pointer `Item` from the list, if it is present in the list. It returns the index of the removed class pointer, or `-1` if the class pointer was not present in the list. It simply overrides the implementation in `TList` so it accepts a class pointer instead of a simple pointer. No new behaviour is introduced.

Errors: None.

See also: TClassList.Extract (35), #rtl.classes.Tlist.Remove (**??**)

3.7.7 TClassList.IndexOf

Synopsis: Search for a class pointer in the list.

Declaration: `function IndexOf(AClass: TClass) : Integer`

Visibility: public

Description: `IndexOf` searches for `AClass` in the list, and returns it's position if it was found, or -1 if it was not found in the list.

Errors: None.

See also: #rtl.classes.tlist.indexof (**??**)

3.7.8 TClassList.First

Synopsis: Return first non-nil class pointer

Declaration: `function First : TClass`

Visibility: public

Description: `First` returns a reference to the first non-`Nil` class pointer in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TClassList.Last (36), TClassList.Pack (34)

3.7.9 TClassList.Last

Synopsis: Return last non-`Nil` class pointer

Declaration: `function Last : TClass`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` class pointer in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TClassList.First (36), TClassList.Pack (34)

3.7.10 TClassList.Insert

Synopsis: Insert a new class pointer in the list.

Declaration: `procedure Insert(Index: Integer;AClass: TClass)`

Visibility: public

Description: `Insert` inserts a class pointer in the list at position `Index`. It simply overrides the parent implementation so it only accepts class pointers. It introduces no new behaviour.

Errors: None.

See also: #rtl.classes.TList.Insert (**??**), TClassList.Add (35), TClassList.Remove (35)

3.7.11 TClassList.Items

Synopsis: Index based access to class pointers.

Declaration: `Property Items[Index: Integer]: TClass; default`

Visibility: public

Access: Read,Write

Description: `Items` provides index-based access to the class pointers in the list. `TClassList` overrides the default `Items` implementation of `TList` so it returns class pointers instead of pointers.

See also: #rtl.classes.TList.Items (**??**), #rtl.classes.TList.Count (**??**)

3.8 TComponentList

3.8.1 Description

`TComponentList` is a TObjectList (76) descendent which has as the default array property TComponents (**??**) instead of objects. It overrides some methods so only components can be added.

In difference with TObjectList (76), `TComponentList` removes any `TComponent` from the list if the `TComponent` instance was freed externally. It uses the `FreeNotification` mechanism for this.

3.8.2 Method overview

Page	Property	Description
38	Add	Add a component to the list.
37	Destroy	Destroys the instance
38	Extract	Remove a component from the list without destroying it.
39	First	First non-nil instance in the list.
38	IndexOf	Search for an instance in the list
39	Insert	Insert a new component in the list
39	Last	Last non-nil instance in the list.
38	Remove	Remove a component from the list, possibly destroying it.

3.8.3 Property overview

Page	Property	Access	Description
40	Items	rw	Index-based access to the elements in the list.

3.8.4 TComponentList.Destroy

Synopsis: Destroys the instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` unhooks the free notification handler and then calls the inherited destroy to clean up the `TComponentList` instance.

Errors: None.

See also: TObjectList (76), #rtl.classes.TComponent (**??**)

3.8.5 TComponentList.Add

Synopsis: Add a component to the list.

Declaration: `function Add(AComponent: TComponent) : Integer`

Visibility: public

Description: `Add` overrides the `Add` operation of it's ancestors, so it only accepts `TComponent` instances. It introduces no new behaviour.

The function returns the index at which the component was added.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TObectList.Add (31)

3.8.6 TComponentList.Extract

Synopsis: Remove a component from the list without destroying it.

Declaration: `function Extract(Item: TComponent) : TComponent`

Visibility: public

Description: `Extract` removes a component (`Item`) from the list, without destroying it. It overrides the implementation of TObjectList (76) so only `TComponent` descendents can be extracted. It introduces no new behaviour.

`Extract` returns the instance that was extracted, or `Nil` if no instance was found.

See also: TComponentList.Remove (38), TObjectList.Extract (77)

3.8.7 TComponentList.Remove

Synopsis: Remove a component from the list, possibly destroying it.

Declaration: `function Remove(AComponent: TComponent) : Integer`

Visibility: public

Description: `Remove` removes `item` from the list, and if the list owns it's items, it also destroys it. It returns the index of the item that was removed, or -1 if no item was removed.

`Remove` simply overrides the implementation in TObjectList (76) so it only accepts `TComponent` descendents. It introduces no new behaviour.

Errors: None.

See also: TComponentList.Extract (38), TObjectList.Remove (77)

3.8.8 TComponentList.IndexOf

Synopsis: Search for an instance in the list

Declaration: `function IndexOf(AComponent: TComponent) : Integer`

Visibility: public

Description: `IndexOf` searches for an instance in the list and returns it's position in the list. The position is zero-based. If no instance is found, -1 is returned.

`IndexOf` just overrides the implementation of the parent class so it accepts only `TComponent` instances. It introduces no new behaviour.

Errors: None.

See also: TObjectList.IndexOf (78)

3.8.9 TComponentList.First

Synopsis: First non-nil instance in the list.

Declaration: `function First : TComponent`

Visibility: public

Description: `First` overrides the implementation of it's ancestors to return the first non-nil instance of `TComponent` in the list. If no non-nil instance is found, `Nil` is returned.

Errors: None.

See also: TComponentList.Last (39), TObjectList.First (78)

3.8.10 TComponentList.Last

Synopsis: Last non-nil instance in the list.

Declaration: `function Last : TComponent`

Visibility: public

Description: `Last` overrides the implementation of it's ancestors to return the last non-nil instance of `TComponent` in the list. If no non-nil instance is found, `Nil` is returned.

Errors: None.

See also: TComponentList.First (39), TObjectList.Last (79)

3.8.11 TComponentList.Insert

Synopsis: Insert a new component in the list

Declaration: `procedure Insert(Index: Integer;AComponent: TComponent)`

Visibility: public

Description: `Insert` inserts a `TComponent` instance (`AComponent`) in the list at position `Index`. It simply overrides the parent implementation so it only accepts `TComponent` instances. It introduces no new behaviour.

Errors: None.

See also: TObjectList.Insert (78), TComponentList.Add (38), TComponentList.Remove (38)

3.8.12 TComponentList.Items

Synopsis: Index-based access to the elements in the list.

Declaration: `Property Items[Index: Integer]: TComponent; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the components in the list using an index. It simply overrides the default property of the parent classes so it returns/accepts `TComponent` instances only. Note that the index is zero based.

See also: TObjectList.Items (79)

3.9 TFPCustomHashTable

3.9.1 Description

`TFPCustomHashTable` is a general-purpose hashing class. It can store string keys and pointers associated with these strings. The hash mechanism is configurable and can be optionally be specified when a new instance of the class is created; A default hash mechanism is implemented in RSHash (34).

A `TFPHasList` should be used when fast lookup of data based on some key is required. The other container objects only offer linear search methods, while the hash list offers faster search mechanisms.

3.9.2 Method overview

Page	Property	Description
41	ChangeTableSize	Change the table size of the hash table.
42	Clear	Clear the hash table.
41	Create	Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism
41	CreateWith	Instantiate a new `TFPCustomHashTable` instance with given algorithm and size
42	Delete	Delete a key from the hash list.
41	Destroy	Free the hash table.
42	Find	Search for an item with a certain key value.
42	IsEmpty	Check if the hash table is empty.

3.9.3 Property overview

Page	Property	Access	Description
44	AVGChainLen	r	Average chain length
43	Count	r	Number of items in the hash table.
45	Density	r	Number of filled slots
43	HashFunction	rw	Hash function currently in use
43	HashTable	r	Hash table instance
43	HashTableSize	rw	Size of the hash table
44	LoadFactor	r	Fraction of count versus size
44	MaxChainLength	r	Maximum chain length
45	NumberOfCollisions	r	Number of extra items
44	VoidSlots	r	Number of empty slots in the hash table.

3.9.4 TFPCustomHashTable.Create

Synopsis: Instantiate a new `TFPCustomHashTable` instance using the default hash mechanism

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TFPCustomHashTable` with hash size 196613 and hash algorithm RSHash (34)

Errors: If no memory is available, an exception may be raised.

See also: TFPCustomHashTable.CreateWith (41)

3.9.5 TFPCustomHashTable.CreateWith

Synopsis: Instantiate a new `TFPCustomHashTable` instance with given algorithm and size

Declaration:
```
constructor CreateWith(AHashTableSize: LongWord;
                       aHashFunc: THashFunction)
```

Visibility: public

Description: `CreateWith` creates a new instance of `TFPCustomHashTable` with hash size `AHashTableSize` and hash calculating algorithm `aHashFunc`.

Errors: If no memory is available, an exception may be raised.

See also: TFPCustomHashTable.Create (41)

3.9.6 TFPCustomHashTable.Destroy

Synopsis: Free the hash table.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` removes the hash table from memory. If any data was associated with the keys in the hash table, then this data is not freed. This must be done by the programmer.

Errors: None.

See also: TFPCustomHashTable.Destroy (41), TFPCustomHashTable.Create (41), TFPCustomHashTable.CreateWith (41), THTCustomNode.Data (73)

3.9.7 TFPCustomHashTable.ChangeTableSize

Synopsis: Change the table size of the hash table.

Declaration: `procedure ChangeTableSize(const ANewSize: LongWord);  Virtual`

Visibility: public

Description: `ChangeTableSize` changes the size of the hash table: it recomputes the hash value for all of the keys in the table, so this is an expensive operation.

Errors: If no memory is available, an exception may be raised.

See also: TFPCustomHashTable.HashTableSize (43)

3.9.8 TFPCustomHashTable.Clear

Synopsis: Clear the hash table.

Declaration: `procedure Clear;  Virtual`

Visibility: public

Description: `Clear` removes all keys and their associated data from the hash table. The data itself is not freed from memory, this should be done by the programmer.

Errors: None.

See also: TFPCustomHashTable.Destroy (41)

3.9.9 TFPCustomHashTable.Delete

Synopsis: Delete a key from the hash list.

Declaration: `procedure Delete(const aKey: String);  Virtual`

Visibility: public

Description: `Delete` deletes all keys with value `AKey` from the hash table. It does not free the data associated with key. If `AKey` is not in the list, nothing is removed.

Errors: None.

See also: TFPCustomHashTable.Find (42), TFPCustomHashTable.Add (40)

3.9.10 TFPCustomHashTable.Find

Synopsis: Search for an item with a certain key value.

Declaration: `function Find(const aKey: String) : THTCustomNode`

Visibility: public

Description: `Find` searches for the THTCustomNode (73) instance with key value equal to `Akey` and if it finds it, it returns the instance. If no matching value is found, `Nil` is returned.

Note that the instance returned by this function cannot be freed; If it should be removed from the hash table, the Delete (42) method should be used instead.

Errors: None.

See also: TFPCustomHashTable.Add (40), TFPCustomHashTable.Delete (42)

3.9.11 TFPCustomHashTable.IsEmpty

Synopsis: Check if the hash table is empty.

Declaration: `function IsEmpty : Boolean`

Visibility: public

Description: `IsEmpty` returns `True` if the hash table contains no elements, or `False` if there are still elements in the hash table.

Errors:

See also: TFPCustomHashTable.Count (43), TFPCustomHashTable.HashTableSize (43), TFPCustomHashTable.AVGChainLen (44), TFPCustomHashTable.MaxChainLength (44)

3.9.12 TFPCustomHashTable.HashFunction

Synopsis: Hash function currently in use

Declaration: `Property HashFunction : THashFunction`

Visibility: public

Access: Read,Write

Description: `HashFunction` is the hash function currently in use to calculate hash values from keys. The property can be set, this simply calls SetHashFunction (40). Note that setting the hash function does NOT the hash value of all keys to be recomputed, so changing the value while there are still keys in the table is not a good idea.

See also: TFPCustomHashTable.SetHashFunction (40), TFPCustomHashTable.HashTableSize (43)

3.9.13 TFPCustomHashTable.Count

Synopsis: Number of items in the hash table.

Declaration: `Property Count : LongWord`

Visibility: public

Access: Read

Description: `Count` is the number of items in the hash table.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.HashTableSize (43), TFPCustomHashTable.AVGChainLen (44), TFPCustomHashTable.MaxChainLength (44)

3.9.14 TFPCustomHashTable.HashTableSize

Synopsis: Size of the hash table

Declaration: `Property HashTableSize : LongWord`

Visibility: public

Access: Read,Write

Description: `HashTableSize` is the size of the hash table. It can be set, in which case it will be rounded to the nearest prime number suitable for RSHash.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.AVGChainLen (44), TFPCustomHashTable.MaxChainLength (44), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45)

3.9.15 TFPCustomHashTable.HashTable

Synopsis: Hash table instance

Declaration: `Property HashTable : TFPObjectList`

Visibility: public

Access: Read

Description: `TFPCustomHashTable` is the internal list object (TFPObjectList (64) used for the hash table. Each element in this table is again a TFPObjectList (64) instance or `Nil`.

3.9.16 TFPCustomHashTable.VoidSlots

Synopsis: Number of empty slots in the hash table.

Declaration: `Property VoidSlots : LongWord`

Visibility: public

Access: Read

Description: `VoidSlots` is the number of empty slots in the hash table. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.AVGChainLen (44), TFPCustomHashTable.MaxChainLength (44), TFPCustomHashTable.LoadFactor (44), TFPCustomHashTable.Density (45), TFPCustomHashTable.NumberOfCollisions (45)

3.9.17 TFPCustomHashTable.LoadFactor

Synopsis: Fraction of count versus size

Declaration: `Property LoadFactor : double`

Visibility: public

Access: Read

Description: `LoadFactor` is the ratio of elements in the table versus table size. Ideally, this should be as small as possible.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.AVGChainLen (44), TFPCustomHashTable.MaxChainLength (44), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45), TFPCustomHashTable.NumberOfCollisions (45)

3.9.18 TFPCustomHashTable.AVGChainLen

Synopsis: Average chain length

Declaration: `Property AVGChainLen : double`

Visibility: public

Access: Read

Description: `AVGChainLen` is the average chain length, i.e. the ratio of elements in the table versus the number of filled slots. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.LoadFactor (44), TFPCustomHashTable.MaxChainLength (44), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45), TFPCustomHashTable.NumberOfCollisions (45)

3.9.19 TFPCustomHashTable.MaxChainLength

Synopsis: Maximum chain length

Declaration: `Property MaxChainLength : LongWord`

Visibility: public

Access: Read

Description: `MaxChainLength` is the length of the longest chain in the hash table. Calculating this is an expensive operation.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.LoadFactor (44), TFPCustomHashTable.AvgChainLength (40), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45), TFPCustomHashTable.NumberOfCollisions (45)

3.9.20 TFPCustomHashTable.NumberOfCollisions

Synopsis: Number of extra items

Declaration: `Property NumberOfCollisions : LongWord`

Visibility: public

Access: Read

Description: `NumberOfCollisions` is the number of items which are not the first item in a chain. If this number is too big, the hash size may be too small.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.LoadFactor (44), TFPCustomHashTable.AvgChainLength (40), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45)

3.9.21 TFPCustomHashTable.Density

Synopsis: Number of filled slots

Declaration: `Property Density : LongWord`

Visibility: public

Access: Read

Description: `Density` is the number of filled slots in the hash table.

See also: TFPCustomHashTable.IsEmpty (42), TFPCustomHashTable.Count (43), TFPCustomHashTable.LoadFactor (44), TFPCustomHashTable.AvgChainLength (40), TFPCustomHashTable.VoidSlots (44), TFPCustomHashTable.Density (45)

3.10 TFPDataHashTable

3.10.1 Description

`TFPDataHashTable` is a TFPCustomHashTable (40) descendent which stores simple data pointers together with the keys. In case the data associated with the keys are objects, it's better to use TFPObjectHashTable (62), or for string data, TFPStringHashTable (72) is more suitable. The data pointers are exposed with their keys through the Items (46) property.

3.10.2 Method overview

Page	Property	Description
46	Add	Add a data pointer to the list.

3.10.3 Property overview

Page	Property	Access	Description
46	Items	rw	Key-based access to the items in the table

3.10.4 TFPDataHashTable.Add

Synopsis: Add a data pointer to the list.

Declaration: `procedure Add(const aKey: String;AItem: pointer);  Virtual`

Visibility: public

Description: `Add` adds a data pointer (`AItem`) to the list with key `AKey`.

Errors: If `AKey` already exists in the table, an exception is raised.

See also: TFPDataHashTable.Items (46)

3.10.5 TFPDataHashTable.Items

Synopsis: Key-based access to the items in the table

Declaration: `Property Items[index: String]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the items in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an `Nil` pointer.

See also: TFPStringHashTable.Add (72)

3.11 TFPHashList

3.11.1 Description

`TFPHashList` implements a fast hash class. The class is built for speed, therefore the key values can be shortstrings only, and the data can only be pointers.

if a base class for an own hash class is wanted, the TFPCustomHashTable (40) class can be used. If a hash class for objects is needed instead of pointers, the TFPHashObjectList (56) class can be used.

3.11.2 Method overview

Page	Property	Description
48	Add	Add a new key/data pair to the list
48	Clear	Clear the list
47	Create	Create a new instance of the hashlist
49	Delete	Delete an item from the list.
47	Destroy	Removes an instance of the hashlist from the heap
49	Error	Raise an error
49	Expand	Expand the list
49	Extract	Extract a pointer from the list
50	Find	Find data associated with key
50	FindIndexOf	Return index of named item.
50	FindWithHash	Find first element with given name and hash value
52	ForEachCall	Call a procedure for each element in the list
48	HashOfIndex	Return the hash valye of an item by index
50	IndexOf	Return the index of the data pointer
48	NameOfIndex	Returns the key name of an item by index
51	Pack	Remove nil pointers from the list
51	Remove	Remove first instance of a pointer
51	Rename	Rename a key
51	ShowStatistics	Return some statistics for the list.

3.11.3 Property overview

Page	Property	Access	Description
52	Capacity	rw	Capacity of the list.
52	Count	rw	Current number of elements in the list.
52	Items	rw	Indexed array with pointers
53	List	r	Low-level hash list
53	Strs	r	Low-level memory area with strings.

3.11.4 TFPHashList.Create

Synopsis: Create a new instance of the hashlist

Declaration: `constructor Create`

Visibility: public

Description: `Create` creates a new instance of `TFPHashList` on the heap and sets the hash capacity to 1.

See also: TFPHashList.Destroy (47)

3.11.5 TFPHashList.Destroy

Synopsis: Removes an instance of the hashlist from the heap

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the memory structures maintained by the hashlist and removes the `TFPHashList` instance from the heap.

`Destroy` should not be called directly, it's better to use `Free` or `FreeAndNil` instead.

See also: TFPHashList.Create (47), TFPHashList.Clear (48)

3.11.6 TFPHashList.Add

Synopsis: Add a new key/data pair to the list

Declaration: `function Add(const AName: shortstring;Item: Pointer) : Integer`

Visibility: public

Description: `Add` adds a new data pointer (`Item`) with key `AName` to the list. It returns the position of the item in the list.

Errors: If not enough memory is available to hold the key and data, an exception may be raised. If an item with this name already exists in the list, an exception is raised.

See also: TFPHashList.Extract (49), TFPHashList.Remove (51), TFPHashList.Delete (49)

3.11.7 TFPHashList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all items from the list. It does not free the data items themselves. It frees all memory needed to contain the items.

Errors: None.

See also: TFPHashList.Extract (49), TFPHashList.Remove (51), TFPHashList.Delete (49), TFPHashList.Add (48)

3.11.8 TFPHashList.NameOfIndex

Synopsis: Returns the key name of an item by index

Declaration: `function NameOfIndex(Index: Integer) : ShortString`

Visibility: public

Description: `NameOfIndex` returns the key name of the item at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashList.HashOfIndex (48), TFPHashList.Find (50), TFPHashList.FindIndexOf (50), TFPHashList.FindWithHash (50)

3.11.9 TFPHashList.HashOfIndex

Synopsis: Return the hash valye of an item by index

Declaration: `function HashOfIndex(Index: Integer) : LongWord`

Visibility: public

Description: `HashOfIndex` returns the hash value of the item at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashList.HashOfName (46), TFPHashList.Find (50), TFPHashList.FindIndexOf (50), TFPHashList.FindWithHash (50)

3.11.10 TFPHashList.Delete

Synopsis: Delete an item from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the item at position `Index`. The data to which it points is not freed from memory.

Errors: TFPHashList.Extract (49)TFPHashList.Remove (51)TFPHashList.Add (48)

3.11.11 TFPHashList.Error

Synopsis: Raise an error

Declaration: `procedure Error(const Msg: String;Data: PtrInt)`

Visibility: public

Description: `Error` raises an `EListError` exception, with message `Msg`. The `Data` pointer is used to format the message.

3.11.12 TFPHashList.Expand

Synopsis: Expand the list

Declaration: `function Expand : TFPHashList`

Visibility: public

Description: `Expand` enlarges the capacity of the list if the maximum capacity was reached. It returns itself.

Errors: If not enough memory is available, an exception may be raised.

See also: TFPHashList.Clear (48)

3.11.13 TFPHashList.Extract

Synopsis: Extract a pointer from the list

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: public

Description: `Extract` removes the data item from the list, if it is in the list. It returns the pointer if it was removed from the list, `Nil` otherwise.

`Extract` does a linear search, and is not very efficient.

See also: TFPHashList.Delete (49), TFPHashList.Remove (51), TFPHashList.Clear (48)

3.11.14 TFPHashList.IndexOf

Synopsis: Return the index of the data pointer

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` returns the index of the first occurrence of pointer `Item`. If the item is not in the list, -1 is returned.

The performed search is linear, and not very efficient.

See also: TFPHashList.HashOfIndex (48), TFPHashList.NameOfIndex (48), TFPHashList.Find (50), TFPHashList.FindIndexOf (50), TFPHashList.FindWithHash (50)

3.11.15 TFPHashList.Find

Synopsis: Find data associated with key

Declaration: `function Find(const AName: shortstring) : Pointer`

Visibility: public

Description: `Find` searches (using the hash) for the data item associated with item `AName` and returns the data pointer associated with it. If the item is not found, `Nil` is returned. It uses the hash value of the key to perform the search.

See also: TFPHashList.HashOfIndex (48), TFPHashList.NameOfIndex (48), TFPHashList.IndexOf (50), TFPHashList.FindIndexOf (50), TFPHashList.FindWithHash (50)

3.11.16 TFPHashList.FindIndexOf

Synopsis: Return index of named item.

Declaration: `function FindIndexOf(const AName: shortstring) : Integer`

Visibility: public

Description: `FindIndexOf` returns the index of the key `AName`, or -1 if the key does not exist in the list. It uses the hash value to search for the key.

See also: TFPHashList.HashOfIndex (48), TFPHashList.NameOfIndex (48), TFPHashList.IndexOf (50), TFPHashList.Find (50), TFPHashList.FindWithHash (50)

3.11.17 TFPHashList.FindWithHash

Synopsis: Find first element with given name and hash value

Declaration:
```
function FindWithHash(const AName: shortstring;AHash: LongWord)
                      : Pointer
```

Visibility: public

Description: `FindWithHash` searches for the item with key `AName`. It uses the provided hash value `AHash` to perform the search. If the item exists, the data pointer is returned, if not, the result is `Nil`.

See also: TFPHashList.HashOfIndex (48), TFPHashList.NameOfIndex (48), TFPHashList.IndexOf (50), TFPHashList.Find (50), TFPHashList.FindIndexOf (50)

3.11.18 TFPHashList.Rename

Synopsis: Rename a key

Declaration:
```
function Rename(const AOldName: shortstring;const ANewName: shortstring)
                      : Integer
```

Visibility: public

Description: `Rename` renames key `AOldname` to `ANewName`. The hash value is recomputed and the item is moved in the list to it's new position.

Errors: If an item with `ANewName` already exists, an exception will be raised.

3.11.19 TFPHashList.Remove

Synopsis: Remove first instance of a pointer

Declaration:
```
function Remove(Item: Pointer) : Integer
```

Visibility: public

Description: `Remove` removes the first occurence of the data pointer `Item` in the list, if it is present. The return value is the removed data pointer, or `Nil` if no data pointer was removed.

See also: TFPHashList.Delete (49), TFPHashList.Clear (48), TFPHashList.Extract (49)

3.11.20 TFPHashList.Pack

Synopsis: Remove nil pointers from the list

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `Nil` items from the list, and frees all unused memory.

See also: TFPHashList.Clear (48)

3.11.21 TFPHashList.ShowStatistics

Synopsis: Return some statistics for the list.

Declaration:
```
procedure ShowStatistics
```

Visibility: public

Description: `ShowStatistics` prints some information about the hash list to standard output. It prints the following values:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

3.11.22 TFPHashList.ForEachCall

Synopsis: Call a procedure for each element in the list

Declaration:
```
procedure ForEachCall(proc2call: TListCallback;arg: pointer)
procedure ForEachCall(proc2call: TListStaticCallback;arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops over the items in the list and calls `proc2call`, passing it the item and `arg`.

3.11.23 TFPHashList.Capacity

Synopsis: Capacity of the list.

Declaration:
```
Property Capacity : Integer
```

Visibility: public

Access: Read,Write

Description: `Capacity` returns the current capacity of the list. The capacity is expanded as more elements are added to the list. If a good estimate of the number of elements that will be added to the list, the property can be set to a sufficiently large value to avoid reallocation of memory each time the list needs to grow.

See also: TFPHashList.Count (52), TFPHashList.Items (52)

3.11.24 TFPHashList.Count

Synopsis: Current number of elements in the list.

Declaration:
```
Property Count : Integer
```

Visibility: public

Access: Read,Write

Description: `Count` is the current number of elements in the list.

See also: TFPHashList.Capacity (52), TFPHashList.Items (52)

3.11.25 TFPHashList.Items

Synopsis: Indexed array with pointers

Declaration:
```
Property Items[Index: Integer]: Pointer; default
```

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the pointers, the index runs from 0 to Count-1 (52).

Errors: Specifying an invalid index will result in an exception.

See also: TFPHashList.Capacity (52), TFPHashList.Count (52)

3.11.26 TFPHashList.List

Synopsis: Low-level hash list

Declaration: `Property List : PHashItemList`

Visibility: public

Access: Read

Description: `List` exposes the low-level item list (33). It should not be used directly.

See also: TFPHashList.Strs (53), THashItemList (33)

3.11.27 TFPHashList.Strs

Synopsis: Low-level memory area with strings.

Declaration: `Property Strs : PChar`

Visibility: public

Access: Read

Description: `Strs` exposes the raw memory area with the strings.

See also: TFPHashList.List (53)

3.12 TFPHashObject

3.12.1 Description

`TFPHashObject` is a `TObject` descendent which is aware of the TFPHashObjectList (56) class. It has a name property and an owning list: if the name is changed, it will reposition itself in the list which owns it. It offers methods to change the owning list: the object will correctly remove itself from the list which currently owns it, and insert itself in the new list.

3.12.2 Method overview

Page	Property	Description
54	ChangeOwner	Change the list owning the object.
54	ChangeOwnerAndName	Simultaneously change the list owning the object and the name of the object.
54	Create	Create a named instance, and insert in a hash list.
54	CreateNotOwned	Create an instance not owned by any list.
55	Rename	Rename the object

3.12.3 Property overview

Page	Property	Access	Description
55	Hash	r	Hash value
55	Name	r	Current name of the object

3.12.4 TFPHashObject.CreateNotOwned

Synopsis: Create an instance not owned by any list.

Declaration:
```
constructor CreateNotOwned
```

Visibility: public

Description: `CreateNotOwned` creates an instance of `TFPHashObject` which is not owned by any TFPHashObjectList (56) hash list. It also has no name when created in this way.

See also: TFPHashObject.Name (55), TFPHashObject.ChangeOwner (54), TFPHashObject.ChangeOwnerAndName (54)

3.12.5 TFPHashObject.Create

Synopsis: Create a named instance, and insert in a hash list.

Declaration:
```
constructor Create(HashObjectList: TFPHashObjectList;
                   const s: shortstring)
```

Visibility: public

Description: `Create` creates an instance of `TFPHashObject`, gives it the name `S` and inserts it in the hash list HashObjectList (56).

See also: TFPHashObject.CreateNotOwned (54), TFPHashObject.ChangeOwner (54), TFPHashObject.Name (55)

3.12.6 TFPHashObject.ChangeOwner

Synopsis: Change the list owning the object.

Declaration:
```
procedure ChangeOwner(HashObjectList: TFPHashObjectList)
```

Visibility: public

Description: `ChangeOwner` can be used to move the object between hash lists: The object will be removed correctly from the hash list that currently owns it, and will be inserted in the list `HashObjectList`.

Errors: If an object with the same name already is present in the new hash list, an exception will be raised.

See also: TFPHashObject.ChangeOwnerAndName (54), TFPHashObject.Name (55)

3.12.7 TFPHashObject.ChangeOwnerAndName

Synopsis: Simultaneously change the list owning the object and the name of the object.

Declaration:
```
procedure ChangeOwnerAndName(HashObjectList: TFPHashObjectList;
                             const s: shortstring)
```

Visibility: public

Description: `ChangeOwnerAndName` can be used to move the object between hash lists: The object will be removed correctly from the hash list that currently owns it (using the current name), and will be inserted in the list `HashObjectList` with the new name `S`.

Errors: If the new name already is present in the new hash list, an exception will be raised.

See also: TFPHashObject.ChangeOwner (54), TFPHashObject.Name (55)

3.12.8 TFPHashObject.Rename

Synopsis: Rename the object

Declaration: `procedure Rename(const ANewName: shortstring)`

Visibility: public

Description: `Rename` changes the name of the object, and notifies the hash list of this change.

Errors: If the new name already is present in the hash list, an exception will be raised.

See also: TFPHashObject.ChangeOwner (54), TFPHashObject.ChangeOwnerAndName (54), TFPHashObject.Name (55)

3.12.9 TFPHashObject.Name

Synopsis: Current name of the object

Declaration: `Property Name : shortstring`

Visibility: public

Access: Read

Description: `Name` is the name of the object, it is stored in the hash list using this name as the key.

See also: TFPHashObject.Rename (55), TFPHashObject.ChangeOwnerAndName (54)

3.12.10 TFPHashObject.Hash

Synopsis: Hash value

Declaration: `Property Hash : LongWord`

Visibility: public

Access: Read

Description: `Hash` is the hash value of the object in the hash list that owns it.

See also: TFPHashObject.Name (55)

3.13 TFPHashObjectList

3.13.1 Method overview

Page	Property	Description
57	Add	Add a new key/data pair to the list
57	Clear	Clear the list
56	Create	Create a new instance of the hashlist
58	Delete	Delete an object from the list.
56	Destroy	Removes an instance of the hashlist from the heap
58	Expand	Expand the list
58	Extract	Extract a object instance from the list
59	Find	Find data associated with key
59	FindIndexOf	Return index of named object.
60	FindInstanceOf	Search an instance of a certain class
60	FindWithHash	Find first element with given name and hash value
61	ForEachCall	Call a procedure for each object in the list
58	HashOfIndex	Return the hash valye of an object by index
59	IndexOf	Return the index of the object instance
57	NameOfIndex	Returns the key name of an object by index
60	Pack	Remove nil object instances from the list
59	Remove	Remove first occurrence of a object instance
60	Rename	Rename a key
61	ShowStatistics	Return some statistics for the list.

3.13.2 Property overview

Page	Property	Access	Description
61	Capacity	rw	Capacity of the list.
61	Count	rw	Current number of elements in the list.
62	Items	rw	Indexed array with object instances
62	List	r	Low-level hash list
62	OwnsObjects	rw	Does the list own the objects it contains

3.13.3 TFPHashObjectList.Create

Synopsis: Create a new instance of the hashlist

Declaration: `constructor Create(FreeObjects: Boolean)`

Visibility: public

Description: `Create` creates a new instance of `TFPHashObjectList` on the heap and sets the hash capacity to 1.

If `FreeObjects` is `True` (the default), then the list owns the objects: when an object is removed from the list, it is destroyed (freed from memory). Clearing the list will free all objects in the list.

See also: TFPHashObjectList.Destroy (56), TFPHashObjectList.OwnsObjects (62)

3.13.4 TFPHashObjectList.Destroy

Synopsis: Removes an instance of the hashlist from the heap

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the memory structures maintained by the hashlist and removes the `TFPHashObjectList` instance from the heap. If the list owns its objects, they are freed from memory as well.

`Destroy` should not be called directly, it's better to use `Free` or `FreeAndNil` instead.

See also: TFPHashObjectList.Create (56), TFPHashObjectList.Clear (57)

3.13.5 TFPHashObjectList.Clear

Synopsis: Clear the list

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` removes all objects from the list. It does not free the objects themselves, unless OwnsObjects (62) is `True`. It always frees all memory needed to contain the objects.

Errors: None.

See also: TFPHashObjectList.Extract (58), TFPHashObjectList.Remove (59), TFPHashObjectList.Delete (58), TFPHashObjectList.Add (57)

3.13.6 TFPHashObjectList.Add

Synopsis: Add a new key/data pair to the list

Declaration: `function Add(const AName: shortstring;AObject: TObject) : Integer`

Visibility: public

Description: `Add` adds a new object instance (`AObject`) with key `AName` to the list. It returns the position of the object in the list.

Errors: If not enough memory is available to hold the key and data, an exception may be raised. If an object with this name already exists in the list, an exception is raised.

See also: TFPHashObjectList.Extract (58), TFPHashObjectList.Remove (59), TFPHashObjectList.Delete (58)

3.13.7 TFPHashObjectList.NameOfIndex

Synopsis: Returns the key name of an object by index

Declaration: `function NameOfIndex(Index: Integer) : ShortString`

Visibility: public

Description: `NameOfIndex` returns the key name of the object at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashObjectList.HashOfIndex (58), TFPHashObjectList.Find (59), TFPHashObjectList.FindIndexOf (59), TFPHashObjectList.FindWithHash (60)

3.13.8 TFPHashObjectList.HashOfIndex

Synopsis: Return the hash valye of an object by index

Declaration: `function HashOfIndex(Index: Integer) : LongWord`

Visibility: public

Description: `HashOfIndex` returns the hash value of the object at position `Index`.

Errors: If `Index` is out of the valid range, an exception is raised.

See also: TFPHashObjectList.HashOfName (56), TFPHashObjectList.Find (59), TFPHashObjectList.FindIndexOf (59), TFPHashObjectList.FindWithHash (60)

3.13.9 TFPHashObjectList.Delete

Synopsis: Delete an object from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` deletes the object at position `Index`. If OwnsObjects (62) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Extract (58), TFPHashObjectList.Remove (59), TFPHashObjectList.Add (57), TFPHashObjectList.OwnsObjects (62)

3.13.10 TFPHashObjectList.Expand

Synopsis: Expand the list

Declaration: `function Expand : TFPHashObjectList`

Visibility: public

Description: `Expand` enlarges the capacity of the list if the maximum capacity was reached. It returns itself.

Errors: If not enough memory is available, an exception may be raised.

See also: TFPHashObjectList.Clear (57)

3.13.11 TFPHashObjectList.Extract

Synopsis: Extract a object instance from the list

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes the data object from the list, if it is in the list. It returns the object instance if it was removed from the list, `Nil` otherwise. The object is *not* freed from memory, regardless of the value of OwnsObjects (62).

`Extract` does a linear search, and is not very efficient.

See also: TFPHashObjectList.Delete (58), TFPHashObjectList.Remove (59), TFPHashObjectList.Clear (57)

3.13.12 TFPHashObjectList.Remove

Synopsis: Remove first occurrence of a object instance

Declaration: `function Remove(AObject: TObject) : Integer`

Visibility: public

Description: `Remove` removes the first occurence of the object instance `Item` in the list, if it is present. The return value is the location of the removed object instance, or `-1` if no object instance was removed.

If OwnsObjects (62) is `True`, then the object itself is also freed from memory.

See also: TFPHashObjectList.Delete (58), TFPHashObjectList.Clear (57), TFPHashObjectList.Extract (58)

3.13.13 TFPHashObjectList.IndexOf

Synopsis: Return the index of the object instance

Declaration: `function IndexOf(AObject: TObject) : Integer`

Visibility: public

Description: `IndexOf` returns the index of the first occurrence of object instance `AObject`. If the object is not in the list, -1 is returned.

The performed search is linear, and not very efficient.

See also: TFPHashObjectList.HashOfIndex (58), TFPHashObjectList.NameOfIndex (57), TFPHashObjectList.Find (59), TFPHashObjectList.FindIndexOf (59), TFPHashObjectList.FindWithHash (60)

3.13.14 TFPHashObjectList.Find

Synopsis: Find data associated with key

Declaration: `function Find(const s: shortstring) : TObject`

Visibility: public

Description: `Find` searches (using the hash) for the data object associated with key `AName` and returns the data object instance associated with it. If the object is not found, `Nil` is returned. It uses the hash value of the key to perform the search.

See also: TFPHashObjectList.HashOfIndex (58), TFPHashObjectList.NameOfIndex (57), TFPHashObjectList.IndexOf (59), TFPHashObjectList.FindIndexOf (59), TFPHashObjectList.FindWithHash (60)

3.13.15 TFPHashObjectList.FindIndexOf

Synopsis: Return index of named object.

Declaration: `function FindIndexOf(const s: shortstring) : Integer`

Visibility: public

Description: `FindIndexOf` returns the index of the key `AName`, or -1 if the key does not exist in the list. It uses the hash value to search for the key.

See also: TFPHashObjectList.HashOfIndex (58), TFPHashObjectList.NameOfIndex (57), TFPHashObjectList.IndexOf (59), TFPHashObjectList.Find (59), TFPHashObjectList.FindWithHash (60)

3.13.16 TFPHashObjectList.FindWithHash

Synopsis: Find first element with given name and hash value

Declaration:
```
function FindWithHash(const AName: shortstring;AHash: LongWord)
                            : Pointer
```

Visibility: public

Description: `FindWithHash` searches for the object with key `AName`. It uses the provided hash value `AHash` to perform the search. If the object exists, the data object instance is returned, if not, the result is `Nil`.

See also: TFPHashObjectList.HashOfIndex (58), TFPHashObjectList.NameOfIndex (57), TFPHashObjectList.IndexOf (59), TFPHashObjectList.Find (59), TFPHashObjectList.FindIndexOf (59)

3.13.17 TFPHashObjectList.Rename

Synopsis: Rename a key

Declaration:
```
function Rename(const AOldName: shortstring;const ANewName: shortstring)
                      : Integer
```

Visibility: public

Description: `Rename` renames key `AOldname` to `ANewName`. The hash value is recomputed and the object is moved in the list to it's new position.

Errors: If an object with `ANewName` already exists, an exception will be raised.

3.13.18 TFPHashObjectList.FindInstanceOf

Synopsis: Search an instance of a certain class

Declaration:
```
function FindInstanceOf(AClass: TClass;AExact: Boolean;
                            AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` searches the list for an instance of class `AClass`. It starts searching at position `AStartAt`. If `AExact` is `True`, only instances of class `AClass` are considered. If `AExact` is `False`, then descendent classes of `AClass` are also taken into account when searching. If no instance is found, `Nil` is returned.

3.13.19 TFPHashObjectList.Pack

Synopsis: Remove nil object instances from the list

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `Nil` objects from the list, and frees all unused memory.

See also: TFPHashObjectList.Clear (57)

3.13.20 TFPHashObjectList.ShowStatistics

Synopsis: Return some statistics for the list.

Declaration: `procedure ShowStatistics`

Visibility: public

Description: `ShowStatistics` prints some information about the hash list to standard output. It prints the following values:

HashSizeSize of the hash table

HashMeanMean hash value

HashStdDevStandard deviation of hash values

ListSizeSize and capacity of the list

StringSizeSize and capacity of key strings

3.13.21 TFPHashObjectList.ForEachCall

Synopsis: Call a procedure for each object in the list

Declaration:
```
procedure ForEachCall(proc2call: TObjectListCallback;arg: pointer)
procedure ForEachCall(proc2call: TObjectListStaticCallback;arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops over the objects in the list and calls `proc2call`, passing it the object and `arg`.

3.13.22 TFPHashObjectList.Capacity

Synopsis: Capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` returns the current capacity of the list. The capacity is expanded as more elements are added to the list. If a good estimate of the number of elements that will be added to the list, the property can be set to a sufficiently large value to avoid reallocation of memory each time the list needs to grow.

See also: TFPHashObjectList.Count (61), TFPHashObjectList.Items (62)

3.13.23 TFPHashObjectList.Count

Synopsis: Current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of elements in the list.

See also: TFPHashObjectList.Capacity (61), TFPHashObjectList.Items (62)

3.13.24 TFPHashObjectList.OwnsObjects

Synopsis: Does the list own the objects it contains

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines what to do when an object is removed from the list: if it is `True` (the default), then the list owns the objects: when an object is removed from the list, it is destroyed (freed from memory). Clearing the list will free all objects in the list.

The value of `OwnsObjects` is set when the hash list is created, and cannot be changed during the lifetime of the hash list.

See also: TFPHashObjectList.Create (56)

3.13.25 TFPHashObjectList.Items

Synopsis: Indexed array with object instances

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the object instances, the index runs from 0 to Count-1 (61).

Errors: Specifying an invalid index will result in an exception.

See also: TFPHashObjectList.Capacity (61), TFPHashObjectList.Count (61)

3.13.26 TFPHashObjectList.List

Synopsis: Low-level hash list

Declaration: `Property List : TFPHashList`

Visibility: public

Access: Read

Description: `List` exposes the low-level hash list (46). It should not be used directly.

See also: TFPHashList (46)

3.14 TFPObjectHashTable

3.14.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (40) descendent which stores object instances together with the keys. In case the data associated with the keys are strings themselves, it's better to use TFPStringHashTable (72), or for arbitrary pointer data, TFPDataHashTable (45) is more suitable. The objects are exposed with their keys through the Items (64) property.

3.14.2 Method overview

Page	Property	Description
64	Add	Add a new object to the hash table
63	Create	Create a new instance of `TFPObjectHashTable`
63	CreateWith	Create a new hash table with given size and hash function

3.14.3 Property overview

Page	Property	Access	Description
64	Items	rw	Key-based access to the objects
64	OwnsObjects	rw	Does the hash table own the objects ?

3.14.4 TFPObjectHashTable.Create

Synopsis: Create a new instance of `TFPObjectHashTable`

Declaration:
```
constructor Create(AOwnsObjects: Boolean)
```

Visibility: public

Description: `Create` creates a new instance of `TFPObjectHashTable` on the heap. It sets the OwnsObjects (64) property to `AOwnsObjects`, and then calls the inherited `Create`. If `AOwnsObjects` is set to `True`, then the hash table owns the objects: whenever an object is removed from the list, it is automatically freed.

Errors: If not enough memory is available on the heap, an exception may be raised.

See also: TFPObjectHashTable.OwnsObjects (64), TFPObjectHashTable.CreateWith (63), TFPObjectHashTable.Items (64)

3.14.5 TFPObjectHashTable.CreateWith

Synopsis: Create a new hash table with given size and hash function

Declaration:
```
constructor CreateWith(AHashTableSize: LongWord;
                       aHashFunc: THashFunction;AOwnsObjects: Boolean)
```

Visibility: public

Description: `CreateWith` sets the OwnsObjects (64) property to `AOwnsObjects`, and then calls the inherited `CreateWith`. If `AOwnsObjects` is set to `True`, then the hash table owns the objects: whenever an object is removed from the list, it is automatically freed.

This constructor should be used when a table size and hash algorithm should be specified that differ from the default table size and hash algorithm.

Errors: If not enough memory is available on the heap, an exception may be raised.

See also: TFPObjectHashTable.OwnsObjects (64), TFPObjectHashTable.Create (63), TFPObjectHashTable.Items (64)

3.14.6 TFPObjectHashTable.Add

Synopsis: Add a new object to the hash table

Declaration: `procedure Add(const aKey: String;AItem: TObject);  Virtual`

Visibility: public

Description: `Add` adds the object `AItem` to the hash table, and associates it with key `aKey`.

Errors: If the key `aKey` is already in the hash table, an exception will be raised.

See also: TFPObjectHashTable.Items (64)

3.14.7 TFPObjectHashTable.Items

Synopsis: Key-based access to the objects

Declaration: `Property Items[index: String]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the objects in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an `Nil` instance.

See also: TFPObjectHashTable.Add (64)

3.14.8 TFPObjectHashTable.OwnsObjects

Synopsis: Does the hash table own the objects ?

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines what happens with objects which are removed from the hash table: if `True`, then removing an object from the hash list will free the object. If `False`, the object is not freed. Note that way in which the object is removed is not relevant: be it `Delete`, `Remove` or `Clear`.

See also: TFPObjectHashTable.Create (63), TFPObjectHashTable.Items (64)

3.15 TFPObjectList

3.15.1 Description

`TFPObjectList` is a TFPList (**??**) based list which has as the default array property TObjects (**??**) instead of pointers. By default it also manages the objects: when an object is deleted or removed from the list, it is automatically freed. This behaviour can be disabled when the list is created.

In difference with TObjectList (76), `TFPObjectList` offers no notification mechanism of list operations, allowing it to be faster than `TObjectList`. For the same reason, it is also not a descendent of `TFPList` (although it uses one internally).

3.15.2 Method overview

Page	Property	Description
66	Add	Add an object to the list.
69	Assign	Copy the contents of a list.
66	Clear	Clear all elements in the list.
65	Create	Create a new object list
66	Delete	Delete an element from the list.
65	Destroy	Clears the list and destroys the list instance
67	Exchange	Exchange the location of two objects
67	Expand	Expand the capacity of the list.
67	Extract	Extract an object from the list
68	FindInstanceOf	Search for an instance of a certain class
69	First	Return the first non-nil object in the list
70	ForEachCall	For each object in the list, call a method or procedure, passing it the object.
68	IndexOf	Search for an object in the list
68	Insert	Insert a new object in the list
69	Last	Return the last non-nil object in the list.
69	Move	Move an object to another location in the list.
70	Pack	Remove all `Nil` references from the list
67	Remove	Remove an item from the list.
70	Sort	Sort the list of objects

3.15.3 Property overview

Page	Property	Access	Description
71	Capacity	rw	Capacity of the list
71	Count	rw	Number of elements in the list.
71	Items	rw	Indexed access to the elements of the list.
72	List	r	Internal list used to keep the objects.
71	OwnsObjects	rw	Should the list free elements when they are removed.

3.15.4 TFPObjectList.Create

Synopsis: Create a new object list

Declaration:
```
constructor Create
constructor Create(FreeObjects: Boolean)
```

Visibility: public

Description: `Create` instantiates a new object list. The `FreeObjects` parameter determines whether objects that are removed from the list should also be freed from memory. By default this is `True`. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: TFPObjectList.Destroy (65), TFPObjectList.OwnsObjects (71), TObjectList (76)

3.15.5 TFPObjectList.Destroy

Synopsis: Clears the list and destroys the list instance

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` clears the list, freeing all objects in the list if OwnsObjects (71) is `True`.

See also: TFPObjectList.OwnsObjects (71), TObjectList.Create (76)

3.15.6 TFPObjectList.Clear

Synopsis: Clear all elements in the list.

Declaration: `procedure Clear`

Visibility: public

Description: Removes all objects from the list, freeing all objects in the list if OwnsObjects (71) is `True`.

See also: TObjectList.Destroy (76)

3.15.7 TFPObjectList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AObject: TObject) : Integer`

Visibility: public

Description: `Add` adds `AObject` to the list and returns the index of the object in the list.

Note that when OwnsObjects (71) is `True`, an object should not be added twice to the list: this will result in memory corruption when the object is freed (as it will be freed twice). The `Add` method does not check this, however.

Errors: None.

See also: TFPObjectList.OwnsObjects (71), TFPObjectList.Delete (66)

3.15.8 TFPObjectList.Delete

Synopsis: Delete an element from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the object at index `Index` from the list. When OwnsObjects (71) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (71) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TTFPObjectList.Remove (31), TFPObjectList.Extract (67), TFPObjectList.OwnsObjects (71), TTFPObjectList.Add (31), TTFPObjectList.Clear (31)

3.15.9 TFPObjectList.Exchange

Synopsis: Exchange the location of two objects

Declaration: `procedure Exchange(Index1: Integer;Index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the objects at indexes `Index1` and `Index2` in a direct operation (i.e. no delete/add is performed).

Errors: If either `Index1` or `Index2` is invalid, an exception will be raised.

See also: TTFPObjectList.Add (31), TTFPObjectList.Delete (31)

3.15.10 TFPObjectList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFPObjectList`

Visibility: public

Description: `Expand` increases the capacity of the list. It calls #rtl.classes.tfplist.expand (**??**) and then returns a reference to itself.

Errors: If there is not enough memory to expand the list, an exception will be raised.

See also: TFPObjectList.Pack (70), TFPObjectList.Clear (66), #rtl.classes.tfplist.expand (**??**)

3.15.11 TFPObjectList.Extract

Synopsis: Extract an object from the list

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes `Item` from the list, if it is present in the list. It returns `Item` if it was found, `Nil` if item was not present in the list.

Note that the object is not freed, and that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (70), TFPObjectList.Clear (66), TFPObjectList.Remove (67), TFPObjectList.Delete (66)

3.15.12 TFPObjectList.Remove

Synopsis: Remove an item from the list.

Declaration: `function Remove(AObject: TObject) : Integer`

Visibility: public

Description: `Remove` removes `Item` from the list, if it is present in the list. It frees `Item` if OwnsObjects (71) is `True`, and returns the index of the object that was found in the list, or -1 if the object was not found.

Note that only the first found object is removed from the list.

Errors: None.

See also: TFPObjectList.Pack (70), TFPObjectList.Clear (66), TFPObjectList.Delete (66), TFPObjectList.Extract (67)

3.15.13 TFPObjectList.IndexOf

Synopsis: Search for an object in the list

Declaration:
```
function IndexOf(AObject: TObject) : Integer
```

Visibility: public

Description: `IndexOf` searches for the presence of `AObject` in the list, and returns the location (index) in the list. The index is 0-based, and -1 is returned if `AObject` was not found in the list.

Errors: None.

See also: TFPObjectList.Items (71), TFPObjectList.Remove (67), TFPObjectList.Extract (67)

3.15.14 TFPObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class

Declaration:
```
function FindInstanceOf(AClass: TClass;AExact: Boolean;
                        AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` will look through the instances in the list and will return the first instance which is a descendent of class `AClass` if `AExact` is `False`. If `AExact` is true, then the instance should be of class `AClass`.

If no instance of the requested class is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.IndexOf (68)

3.15.15 TFPObjectList.Insert

Synopsis: Insert a new object in the list

Declaration:
```
procedure Insert(Index: Integer;AObject: TObject)
```

Visibility: public

Description: `Insert` inserts `AObject` at position `Index` in the list. All elements in the list after this position are shifted. The index is zero based, i.e. an insert at position 0 will insert an object at the first position of the list.

Errors: None.

See also: TFPObjectList.Add (66), TFPObjectList.Delete (66)

3.15.16 TFPObjectList.First

Synopsis: Return the first non-nil object in the list

Declaration: `function First : TObject`

Visibility: public

Description: `First` returns a reference to the first non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.Last (69), TFPObjectList.Pack (70)

3.15.17 TFPObjectList.Last

Synopsis: Return the last non-nil object in the list.

Declaration: `function Last : TObject`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TFPObjectList.First (69), TFPObjectList.Pack (70)

3.15.18 TFPObjectList.Move

Synopsis: Move an object to another location in the list.

Declaration: `procedure Move(CurIndex: Integer;NewIndex: Integer)`

Visibility: public

Description: `Move` moves the object at current location `CurIndex` to location `NewIndex`. Note that the `NewIndex` is determined *after* the object was removed from location `CurIndex`, and can hence be shifted with 1 position if `CurIndex` is less than `NewIndex`.

Contrary to exchange (67), the move operation is done by extracting the object from it's current location and inserting it at the new location.

Errors: If either `CurIndex` or `NewIndex` is out of range, an exception may occur.

See also: TFPObjectList.Exchange (67), TFPObjectList.Delete (66), TFPObjectList.Insert (68)

3.15.19 TFPObjectList.Assign

Synopsis: Copy the contents of a list.

Declaration: `procedure Assign(Obj: TFPObjectList)`

Visibility: public

Description: `Assign` copies the contents of `Obj` if `Obj` is of type `TFPObjectList`

Errors: None.

3.15.20 TFPObjectList.Pack

Synopsis: Remove all `Nil` references from the list

Declaration:
```
procedure Pack
```

Visibility: public

Description: `Pack` removes all `Nil` elements from the list.

Errors: None.

See also: TFPObjectList.First (69), TFPObjectList.Last (69)

3.15.21 TFPObjectList.Sort

Synopsis: Sort the list of objects

Declaration:
```
procedure Sort(Compare: TListSortCompare)
```

Visibility: public

Description: `Sort` will perform a quick-sort on the list, using `Compare` as the compare algorithm. This function should accept 2 pointers and should return the following result:

less than 0 If the first pointer comes before the second.

equal to 0 If the pointers have the same value.

larger than 0 If the first pointer comes after the second.

The function should be able to deal with `Nil` values.

Errors: None.

See also: #rtl.classes.TList.Sort (**??**)

3.15.22 TFPObjectList.ForEachCall

Synopsis: For each object in the list, call a method or procedure, passing it the object.

Declaration:
```
procedure ForEachCall(proc2call: TObjectListCallback;arg: pointer)
procedure ForEachCall(proc2call: TObjectListStaticCallback;arg: pointer)
```

Visibility: public

Description: `ForEachCall` loops through all objects in the list, and calls `proc2call`, passing it the object in the list. Additionally, `arg` is also passed to the procedure. `Proc2call` can be a plain procedure or can be a method of a class.

Errors: None.

See also: TObjectListStaticCallback (33), TObjectListCallback (33)

3.15.23 TFPObjectList.Capacity

Synopsis: Capacity of the list

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of elements that the list can contain before it needs to expand itself, i.e., reserve more memory for pointers. It is always equal or larger than Count (71).

See also: TFPObjectList.Count (71)

3.15.24 TFPObjectList.Count

Synopsis: Number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the number of elements in the list. Note that this includes `Nil` elements.

See also: TFPObjectList.Capacity (71)

3.15.25 TFPObjectList.OwnsObjects

Synopsis: Should the list free elements when they are removed.

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines whether the objects in the list should be freed when they are removed (not extracted) from the list, or when the list is cleared. If the property is `True` then they are freed. If the property is `False` the elements are not freed.

The value is usually set in the constructor, and is seldom changed during the lifetime of the list. It defaults to `True`.

See also: TFPObjectList.Create (65), TFPObjectList.Delete (66), TFPObjectList.Remove (67), TFPObjectList.Clear (66)

3.15.26 TFPObjectList.Items

Synopsis: Indexed access to the elements of the list.

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` is the default property of the list. It provides indexed access to the elements in the list. The index `Index` is zero based, i.e., runs from 0 (zero) to `Count-1`.

See also: TFPObjectList.Count (71)

3.15.27 TFPObjectList.List

Synopsis: Internal list used to keep the objects.

Declaration: `Property List : TFPList`

Visibility: public

Access: Read

Description: `List` is a reference to the TFPList (**??**) instance used to manage the elements in the list.

See also: #rtl.classes.tfplist (**??**)

3.16 TFPStringHashTable

3.16.1 Description

`TFPStringHashTable` is a TFPCustomHashTable (40) descendent which stores simple strings together with the keys. In case the data associated with the keys are objects, it's better to use TFPObjectHashTable (62), or for arbitrary pointer data, TFPDataHashTable (45) is more suitable. The strings are exposed with their keys through the Items (72) property.

3.16.2 Method overview

Page	Property	Description
72	Add	Add a new string to the hash list

3.16.3 Property overview

Page	Property	Access	Description
72	Items	rw	Key based access to the strings in the hash table

3.16.4 TFPStringHashTable.Add

Synopsis: Add a new string to the hash list

Declaration: `procedure Add(const aKey: String;const aItem: String);  Virtual`

Visibility: public

Description: `Add` adds a new string `AItem` to the hash list with key `AKey`.

Errors: If a string with key `Akey` already exists in the hash table, an exception will be raised.

See also: TFPStringHashTable.Items (72)

3.16.5 TFPStringHashTable.Items

Synopsis: Key based access to the strings in the hash table

Declaration: `Property Items[index: String]: String; default`

Visibility: public

Access: Read,Write

Description: `Items` provides access to the strings in the hash table using their key: the array index `Index` is the key. A key which is not present will result in an empty string.

See also: TFPStringHashTable.Add (72)

3.17 THTCustomNode

3.17.1 Description

`THTCustomNode` is used by the TFPCustomHashTable (40) class to store the keys and associated values.

3.17.2 Method overview

Page	Property	Description
73	CreateWith	Create a new instance of `THTCustomNode`
73	HasKey	Check whether this node matches the given key.

3.17.3 Property overview

Page	Property	Access	Description
74	Key	r	Key value associated with this hash item.

3.17.4 THTCustomNode.CreateWith

Synopsis: Create a new instance of `THTCustomNode`

Declaration: `constructor CreateWith(const AString: String)`

Visibility: public

Description: `CreateWith` creates a new instance of `THTCustomNode` and stores the string `AString` in it. It should never be necessary to call this method directly, it will be called by the TFPHashTable (31) class when needed.

Errors: If no more memory is available, an exception may be raised.

See also: TFPHashTable (31)

3.17.5 THTCustomNode.HasKey

Synopsis: Check whether this node matches the given key.

Declaration: `function HasKey(const AKey: String) : Boolean`

Visibility: public

Description: `HasKey` checks whether this node matches the given key `AKey`, by comparing it with the stored key. It returns `True` if it does, `False` if not.

Errors: None.

See also: THTCustomNode.Key (74)

3.17.6 THTCustomNode.Key

Synopsis: Key value associated with this hash item.

Declaration: `Property Key : String`

Visibility: public

Access: Read

Description: `Key` is the key value associated with this hash item. It is stored when the item is created, and is read-only.

See also: THTCustomNode.CreateWith (73)

3.18 THTDataNode

3.18.1 Description

`THTDataNode` is used by TDataHashTable (31) to store the hash items in. It simply holds the data pointer.

It should not be necessary to use `THTDataNode` directly, it's only for inner use by `TFPDataHashTable`

3.18.2 Property overview

Page	Property	Access	Description
74	Data	rw	Data pointer

3.18.3 THTDataNode.Data

Synopsis: Data pointer

Declaration: `Property Data : pointer`

Visibility: public

Access: Read,Write

Description: Pointer containing the user data associated with the hash value.

3.19 THTObjectNode

3.19.1 Description

`THTObjectNode` is a THTCustomNode (73) descendent which holds the data in the TFPObjectHashTable (62) hash table. It exposes a data string.

It should not be necessary to use `THTObjectNode` directly, it's only for inner use by `TFPObjectHashTable`

3.19.2 Property overview

Page	Property	Access	Description
75	Data	rw	Object instance

3.19.3 THTObjectNode.Data

Synopsis: Object instance

Declaration: `Property Data : TObject`

Visibility: public

Access: Read,Write

Description: `Data` is the object instance associated with the key value. It is exposed in TFPObjectHashTable.Items (64)

See also: TFPObjectHashTable (62), TFPObjectHashTable.Items (64), THTOwnedObjectNode (75)

3.20 THTOwnedObjectNode

3.20.1 Description

`THTOwnedObjectNode` is used instead of THTObjectNode (74) in case TFPObjectHashTable (62) owns it's objects. When this object is destroyed, the associated data object is also destroyed.

3.20.2 Method overview

Page	Property	Description
75	Destroy	Destroys the node and the object.

3.20.3 THTOwnedObjectNode.Destroy

Synopsis: Destroys the node and the object.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first frees the data object, and then only frees itself.

See also: THTOwnedObjectNode (75), TFPObjectHashTable.OwnsObjects (64)

3.21 THTStringNode

3.21.1 Description

`THTStringNode` is a THTCustomNode (73) descendent which holds the data in the TFPString-HashTable (72) hash table. It exposes a data string.

It should not be necessary to use `THTStringNode` directly, it's only for inner use by `TFPStringHashTable`

3.21.2 Property overview

Page	Property	Access	Description
76	Data	rw	String data

3.21.3 THTStringNode.Data

Synopsis: String data

Declaration: `Property Data : String`

Visibility: public

Access: Read,Write

Description: `Data` is the data of this has node. The data is a string, associated with the key. It is also exposed in TFPStringHashTable.Items (72)

See also: TFPStringHashTable (72)

3.22 TObjectList

3.22.1 Description

`TObjectList` is a TList (**??**) descendent which has as the default array property TObjects (**??**) instead of pointers. By default it also manages the objects: when an object is deleted or removed from the list, it is automatically freed. This behaviour can be disabled when the list is created.

In difference with TFPObjectList (64), `TObjectList` offers a notification mechanism of list change operations: insert, delete. This slows down bulk operations, so if the notifications are not needed, `TObjectList` may be more appropriate.

3.22.2 Method overview

Page	Property	Description
77	Add	Add an object to the list.
76	create	Create a new object list.
77	Extract	Extract an object from the list.
78	FindInstanceOf	Search for an instance of a certain class
78	First	Return the first non-nil object in the list
78	IndexOf	Search for an object in the list
78	Insert	Insert an object in the list.
79	Last	Return the last non-nil object in the list.
77	Remove	Remove (and possibly free) an element from the list.

3.22.3 Property overview

Page	Property	Access	Description
79	Items	rw	Indexed access to the elements of the list.
79	OwnsObjects	rw	Should the list free elements when they are removed.

3.22.4 TObjectList.create

Synopsis: Create a new object list.

Declaration:
```
constructor create
constructor create(freeobjects: Boolean)
```

Visibility: public

Description: `Create` instantiates a new object list. The `FreeObjects` parameter determines whether objects that are removed from the list should also be freed from memory. By default this is `True`. This behaviour can be changed after the list was instantiated.

Errors: None.

See also: TObjectList.Destroy (76), TObjectList.OwnsObjects (79), TFPObjectList (64)

3.22.5 TObjectList.Add

Synopsis: Add an object to the list.

Declaration: `function Add(AObject: TObject) : Integer`

Visibility: public

Description: `Add` overrides the TList (**??**) implementation to accept objects (`AObject`) instead of pointers.

The function returns the index of the position where the object was added.

Errors: If the list must be expanded, and not enough memory is available, an exception may be raised.

See also: TObjectList.Insert (78), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (77), TObjectList.Remove (77)

3.22.6 TObjectList.Extract

Synopsis: Extract an object from the list.

Declaration: `function Extract(Item: TObject) : TObject`

Visibility: public

Description: `Extract` removes the object `Item` from the list if it is present in the list. Contrary to Remove (77), `Extract` does not free the extracted element if OwnsObjects (79) is `True`

The function returns a reference to the item which was removed from the list, or `Nil` if no element was removed.

Errors: None.

See also: TObjectList.Remove (77)

3.22.7 TObjectList.Remove

Synopsis: Remove (and possibly free) an element from the list.

Declaration: `function Remove(AObject: TObject) : Integer`

Visibility: public

Description: `Remove` removes `Item` from the list, if it is present in the list. It frees `Item` if OwnsObjects (79) is `True`, and returns the index of the object that was found in the list, or -1 if the object was not found.

Note that only the first found object is removed from the list.

Errors: None.

See also: TObjectList.Extract (77)

3.22.8 TObjectList.IndexOf

Synopsis: Search for an object in the list

Declaration:
```
function IndexOf(AObject: TObject) : Integer
```

Visibility: public

Description: `IndexOf` overrides the TList (**??**) implementation to accept an object instance instead of a pointer.

The function returns the index of the first match for `AObject` in the list, or -1 if no match was found.

Errors: None.

See also: TObjectList.FindInstanceOf (78)

3.22.9 TObjectList.FindInstanceOf

Synopsis: Search for an instance of a certain class

Declaration:
```
function FindInstanceOf(AClass: TClass;AExact: Boolean;
                        AStartAt: Integer) : Integer
```

Visibility: public

Description: `FindInstanceOf` will look through the instances in the list and will return the first instance which is a descendent of class `AClass` if `AExact` is `False`. If `AExact` is true, then the instance should be of class `AClass`.

If no instance of the requested class is found, `Nil` is returned.

Errors: None.

See also: TObjectList.IndexOf (78)

3.22.10 TObjectList.Insert

Synopsis: Insert an object in the list.

Declaration:
```
procedure Insert(Index: Integer;AObject: TObject)
```

Visibility: public

Description: `Insert` inserts `AObject` in the list at position `Index`. The index is zero-based. This method overrides the implementation in TList (**??**) to accept objects instead of pointers.

Errors: If an invalid `Index` is specified, an exception is raised.

See also: TObjectList.Add (77), TObjectList.Remove (77)

3.22.11 TObjectList.First

Synopsis: Return the first non-nil object in the list

Declaration:
```
function First : TObject
```

Visibility: public

Description: `First` returns a reference to the first non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TObjectList.Last (79), TObjectList.Pack (76)

3.22.12 TObjectList.Last

Synopsis: Return the last non-nil object in the list.

Declaration: `function Last : TObject`

Visibility: public

Description: `Last` returns a reference to the last non-`Nil` element in the list. If no non-`Nil` element is found, `Nil` is returned.

Errors: None.

See also: TObjectList.First (78), TObjectList.Pack (76)

3.22.13 TObjectList.OwnsObjects

Synopsis: Should the list free elements when they are removed.

Declaration: `Property OwnsObjects : Boolean`

Visibility: public

Access: Read,Write

Description: `OwnsObjects` determines whether the objects in the list should be freed when they are removed (not extracted) from the list, or when the list is cleared. If the property is `True` then they are freed. If the property is `False` the elements are not freed.

The value is usually set in the constructor, and is seldom changed during the lifetime of the list. It defaults to `True`.

See also: TObjectList.Create (76), TObjectList.Delete (76), TObjectList.Remove (77), TObjectList.Clear (76)

3.22.14 TObjectList.Items

Synopsis: Indexed access to the elements of the list.

Declaration: `Property Items[Index: Integer]: TObject; default`

Visibility: public

Access: Read,Write

Description: `Items` is the default property of the list. It provides indexed access to the elements in the list. The index `Index` is zero based, i.e., runs from 0 (zero) to `Count-1`.

See also: #rtl.classes.TList.Count (**??**)

3.23 TObjectQueue

3.23.1 Method overview

Page	Property	Description
80	Peek	Look at the first object in the queue.
80	Pop	Pop the first element off the queue
80	Push	Push an object on the queue

3.23.2 TObjectQueue.Push

Synopsis: Push an object on the queue

Declaration: `function Push(AObject: TObject) : TObject`

Visibility: public

Description: `Push` pushes another object on the queue. It overrides the `Push` method as implemented in `TQueue` so it accepts only objects as arguments.

Errors: If not enough memory is available to expand the queue, an exception may be raised.

See also: TObjectQueue.Pop (80), TObjectQueue.Peek (80)

3.23.3 TObjectQueue.Pop

Synopsis: Pop the first element off the queue

Declaration: `function Pop : TObject`

Visibility: public

Description: `Pop` removes the first element in the queue, and returns a reference to the instance. If the queue is empty, `Nil` is returned.

Errors: None.

See also: TObjectQueue.Push (80), TObjectQueue.Peek (80)

3.23.4 TObjectQueue.Peek

Synopsis: Look at the first object in the queue.

Declaration: `function Peek : TObject`

Visibility: public

Description: `Peek` returns the first object in the queue, without removing it from the queue. If there are no more objects in the queue, `Nil` is returned.

Errors: None

See also: TObjectQueue.Push (80), TObjectQueue.Pop (80)

3.24 TObjectStack

3.24.1 Description

`TObjectStack` is a stack implementation which manages pointers only.

`TObjectStack` introduces no new behaviour, it simply overrides some methods to accept and/or return `TObject` instances instead of pointers.

3.24.2 Method overview

Page	Property	Description
81	Peek	Look at the top object in the stack.
81	Pop	Pop the top object of the stack.
81	Push	Push an object on the stack.

3.24.3 TObjectStack.Push

Synopsis: Push an object on the stack.

Declaration: `function Push(AObject: TObject) : TObject`

Visibility: public

Description: `Push` pushes another object on the stack. It overrides the `Push` method as implemented in `TStack` so it accepts only objects as arguments.

Errors: If not enough memory is available to expand the stack, an exception may be raised.

See also: TObjectStack.Pop (81), TObjectStack.Peek (81)

3.24.4 TObjectStack.Pop

Synopsis: Pop the top object of the stack.

Declaration: `function Pop : TObject`

Visibility: public

Description: `Pop` pops the top object of the stack, and returns the object instance. If there are no more objects on the stack, `Nil` is returned.

Errors: None

See also: TObjectStack.Push (81), TObjectStack.Peek (81)

3.24.5 TObjectStack.Peek

Synopsis: Look at the top object in the stack.

Declaration: `function Peek : TObject`

Visibility: public

Description: `Peek` returns the top object of the stack, without removing it from the stack. If there are no more objects on the stack, `Nil` is returned.

Errors: None

See also: TObjectStack.Push (81), TObjectStack.Pop (81)

3.25 TOrderedList

3.25.1 Description

`TOrderedList` provides the base class for TQueue (84) and TStack (84). It provides an interface for pushing and popping elements on or off the list, and manages the internal list of pointers.

Note that `TOrderedList` does not manage objects on the stack, i.e. objects are not freed when the ordered list is destroyed.

3.25.2 Method overview

Page	Property	Description
83	AtLeast	Check whether the list contains a certain number of elements.
82	Count	Number of elements on the list.
82	Create	Create a new ordered list
82	Destroy	Free an ordered list
83	Peek	Return the next element to be popped from the list.
83	Pop	Remove an element from the list.
83	Push	Push another element on the list.

3.25.3 TOrderedList.Create

Synopsis: Create a new ordered list

Declaration: `constructor Create`

Visibility: public

Description: `Create` instantiates a new ordered list. It initializes the internal pointer list.

Errors: None.

See also: TOrderedList.Destroy (82)

3.25.4 TOrderedList.Destroy

Synopsis: Free an ordered list

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the internal pointer list, and removes the `TOrderedList` instance from memory.

Errors: None.

See also: TOrderedList.Create (82)

3.25.5 TOrderedList.Count

Synopsis: Number of elements on the list.

Declaration: `function Count : Integer`

Visibility: public

Description: `Count` is the number of pointers in the list.

Errors: None.

See also: TOrderedList.AtLeast (83)

3.25.6 TOrderedList.AtLeast

Synopsis: Check whether the list contains a certain number of elements.

Declaration: `function AtLeast(ACount: Integer) : Boolean`

Visibility: public

Description: `AtLeast` returns `True` if the number of elements in the list is equal to or bigger than `ACount`. It returns `False` otherwise.

Errors: None.

See also: TOrderedList.Count (82)

3.25.7 TOrderedList.Push

Synopsis: Push another element on the list.

Declaration: `function Push(AItem: Pointer) : Pointer`

Visibility: public

Description: `Push` adds `AItem` to the list, and returns `AItem`.

Errors: If not enough memory is available to expand the list, an exception may be raised.

See also: TOrderedList.Pop (83), TOrderedList.Peek (83)

3.25.8 TOrderedList.Pop

Synopsis: Remove an element from the list.

Declaration: `function Pop : Pointer`

Visibility: public

Description: `Pop` removes an element from the list, and returns the element that was removed from the list. If no element is on the list, `Nil` is returned.

Errors: None.

See also: TOrderedList.Peek (83), TOrderedList.Push (83)

3.25.9 TOrderedList.Peek

Synopsis: Return the next element to be popped from the list.

Declaration: `function Peek : Pointer`

Visibility: public

Description: `Peek` returns the element that will be popped from the list at the next call to Pop (83), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (83), TOrderedList.Push (83)

3.26 TQueue

3.26.1 Description

`TQueue` is a descendent of TOrderedList (81) which implements Push (83) and Pop (83) behaviour as a queue: what is first pushed on the queue, is popped of first (FIFO: First in, first out).

`TQueue` offers no new methods, it merely implements some abstract methods introduced by TOrderedList (81)

3.27 TStack

3.27.1 Description

`TStack` is a descendent of TOrderedList (81) which implements Push (83) and Pop (83) behaviour as a stack: what is last pushed on the stack, is popped of first (LIFO: Last in, first out).

`TStack` offers no new methods, it merely implements some abstract methods introduced by TOrderedList (81)

Chapter 4

Reference for unit 'dbugintf'

4.1 Writing a debug server

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (85) unit, and use the `DebugServerID` as `ServerID` identification. This constant, as well as the record containing the message which is sent between client and server is defined in the msgintf unit.

The dbugintf unit relies on the SimpleIPC (85) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to start the debug server if needed, so the process (85) unit should also be functional.

4.2 Overview

Use dbugintf to add debug messages to your application. The messages are not sent to standard output, but are sent to a debug server process which collects messages from various clients and displays them somehow on screen.

The unit is transparant in its use: it does not need initialization, it will start the debug server by itself if it can find it: the program should be called debugserver and should be in the `PATH`. When the first debug message is sent, the unit will initialize itself.

The FCL contains a sample debug server (dbugsvr) which can be started in advance, and which writes debug message to the console (both on Windows and Linux). The Lazarus project contains a visual application which displays the messages in a GUI.

The dbugintf unit relies on the SimpleIPC (85) mechanism to communicate with the debug server, hence it works on all platforms that have a functional version of that unit. It also uses `TProcess` to start the debug server if needed, so the process (85) unit should also be functional.

4.3 Constants, types and variables

4.3.1 Resource strings

```
SEntering = '> Entering '
```

String used when sending method enter message.

```
SExiting = '< Exiting '
```

String used when sending method exit message.

```
SProcessID = 'Process %s'
```

String used when sending identification message to the server.

```
SSeparator = '>-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-<'
```

String used when sending a separator line.

4.3.2 Constants

```
SendError : String = ''
```

Whenever a call encounteres an exception, the exception message is stored in this variable.

4.3.3 Types

```
TDebugLevel = (dlInformation,dlWarning,dlError)
```

Table 4.1: Enumeration values for type TDebugLevel

Value	Explanation
dlError	Error message
dlInformation	Informational message
dlWarning	Warning message

`TDebugLevel` indicates the severity level of the debug message to be sent. By default, an informational message is sent.

4.4 Procedures and functions

4.4.1 InitDebugClient

Synopsis: Initialize the debug client.

Declaration: `procedure InitDebugClient`

Visibility: default

Description: `InitDebugClient` starts the debug server and then performs all necessary initialization of the debug IPC communication channel.

Normally this function should not be called. The SendDebug (87) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (87), StartDebugServer (90)

4.4.2 SendBoolean

Synopsis: Send the value of a boolean variable

Declaration: `procedure SendBoolean(const Identifier: String;const Value: Boolean)`

Visibility: default

Description: `SendBoolean` is a simple wrapper around SendDebug (87) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (87), SendDateTime (87), SendInteger (88), SendPointer (89)

4.4.3 SendDateTime

Synopsis: Send the value of a `TDateTime` variable.

Declaration: `procedure SendDateTime(const Identifier: String;const Value: TDateTime)`

Visibility: default

Description: `SendDateTime` is a simple wrapper around SendDebug (87) which sends the name and value of an integer value as an informational message. The value is converted to a string using the DateTimeToStr (**??**) call.

Errors: None.

See also: SendDebug (87), SendBoolean (87), SendInteger (88), SendPointer (89)

4.4.4 SendDebug

Synopsis: Send a message to the debug server.

Declaration: `procedure SendDebug(const Msg: String)`

Visibility: default

Description: `SendDebug` sends the message `Msg` to the debug server as an informational message (debug level `dlInformation`). If no debug server is running, then an attempt will be made to start the server first.

The binary that is started is called debugserver and should be somewhere on the `PATH`. A sample binary which writes received messages to standard output is included in the FCL, it is called dbugsrv. This binary can be renamed to debugserver or can be started before the program is started.

Errors: Errors are silently ignored, any exception messages are stored in SendError (86).

See also: SendDebugEx (87), SendDebugFmt (88), SendDebugFmtEx (88)

4.4.5 SendDebugEx

Synopsis: Send debug message other than informational messages

Declaration: `procedure SendDebugEx(const Msg: String;MType: TDebugLevel)`

Visibility: default

Description: `SendDebugEx` allows to specify the debug level of the message to be sent in `MType`. By default, SendDebug (87) uses informational messages.

Other than that the function of `SendDebugEx` is equal to that of `SendDebug`

Errors: None.

See also: SendDebug (87), SendDebugFmt (88), SendDebugFmtEx (88)

4.4.6 SendDebugFmt

Synopsis: Format and send a debug message

Declaration:
```
procedure SendDebugFmt(const Msg: String;const Args: Array[] of const)
```

Visibility: default

Description: `SendDebugFmt` is a utility routine which formats a message by passing `Msg` and `Args` to Format (**??**) and sends the result to the debug server using SendDebug (87). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (87), SendDebugEx (87), SendDebugFmtEx (88), #rtl.sysutils.format (**??**)

4.4.7 SendDebugFmtEx

Synopsis: Format and send message with alternate type

Declaration:
```
procedure SendDebugFmtEx(const Msg: String;const Args: Array[] of const;
                         MType: TDebugLevel)
```

Visibility: default

Description: `SendDebugFmtEx` is a utility routine which formats a message by passing `Msg` and `Args` to Format (**??**) and sends the result to the debug server using SendDebugEx (87) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (87), SendDebugEx (87), SendDebugFmt (88), #rtl.sysutils.format (**??**)

4.4.8 SendInteger

Synopsis: Send the value of an integer variable.

Declaration:
```
procedure SendInteger(const Identifier: String;const Value: Integer;
                      HexNotation: Boolean)
```

Visibility: default

Description: `SendInteger` is a simple wrapper around SendDebug (87) which sends the name and value of an integer value as an informational message. If `HexNotation` is `True`, then the value will be displayed using hexadecimal notation.

Errors: None.

See also: SendDebug (87), SendBoolean (87), SendDateTime (87), SendPointer (89)

4.4.9 SendMethodEnter

Synopsis: Send method enter message

Declaration: `procedure SendMethodEnter(const MethodName: String)`

Visibility: default

Description: `SendMethodEnter` sends a "Entering `MethodName`" message to the debug server. After that it increases the message indentation (currently 2 characters). By sending a corresponding SendMethodExit (89), the indentation of messages can be decreased again.

By using the `SendMethodEnter` and `SendMethodExit` methods at the beginning and end of a procedure/method, it is possible to visually trace program execution.

Errors: None.

See also: SendDebug (87), SendMethodExit (89), SendSeparator (90)

4.4.10 SendMethodExit

Synopsis: Send method exit message

Declaration: `procedure SendMethodExit(const MethodName: String)`

Visibility: default

Description: `SendMethodExit` sends a "Exiting `MethodName`" message to the debug server. After that it decreases the message indentation (currently 2 characters). By sending a corresponding SendMethodEnter (89), the indentation of messages can be increased again.

By using the `SendMethodEnter` and `SendMethodExit` methods at the beginning and end of a procedure/method, it is possible to visually trace program execution.

Note that the indentation level will not be made negative.

Errors: None.

See also: SendDebug (87), SendMethodEnter (89), SendSeparator (90)

4.4.11 SendPointer

Synopsis: Send the value of a pointer variable.

Declaration: `procedure SendPointer(const Identifier: String;const Value: Pointer)`

Visibility: default

Description: `SendInteger` is a simple wrapper around SendDebug (87) which sends the name and value of a pointer value as an informational message. The pointer value is displayed using hexadecimal notation.

Errors: None.

See also: SendDebug (87), SendBoolean (87), SendDateTime (87), SendInteger (88)

4.4.12 SendSeparator

Synopsis: Send a separator message

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (87) which sends a short horizontal line to the debug server. It can be used to visually separate execution of blocks of code or blocks of values.

Errors: None.

See also: SendDebug (87), SendMethodEnter (89), SendMethodExit (89)

4.4.13 StartDebugServer

Synopsis: Start the debug server

Declaration: `function StartDebugServer : Integer`

Visibility: default

Description: `StartDebugServer` attempts to start the debug server. The process started is called debugserver and should be located in the `PATH`.

Normally this function should not be called. The SendDebug (87) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: SendDebug (87), InitDebugClient (86)

Chapter 5

Reference for unit 'dbugmsg'

5.1 Used units

Table 5.1: Used units by unit 'dbugmsg'

Name	Page
Classes	**??**

5.2 Overview

dbugmsg is an auxialiary unit used in the dbugintf (85) unit. It defines the message protocol used between the debug unit and the debug server.

5.3 Constants, types and variables

5.3.1 Constants

```
DebugServerID : String = 'fpcdebugserver'
```

`DebugServerID` is a string which is used when creating the message protocol, it is used when identifying the server in the (platform dependent) client-server protocol.

```
lctError = 2
```

`lctError` is the identification of error messages.

```
lctIdentify = 3
```

`lctIdentify` is sent by the client to a server when it first connects. It's the first message, and contains the name of client application.

```
lctInformation = 0
```

lctInformation is the identification of informational messages.

lctStop = -1

lctStop is sent by the client to a server when it disconnects.

lctWarning = 1

lctWarning is the identification of warning messages.

5.3.2 Types

```
TDebugMessage = record
  MsgType : Integer;
  MsgTimeStamp : TDateTime;
  Msg : String;
end
```

TDebugMessage is a record that describes the message passed from the client to the server. It should not be passed directly in shared memory, as the string containing the message is allocated on the heap. Instead, the WriteDebugMessageToStream (93) and ReadDebugMessageFromStream (92) can be used to read or write the message from/to a stream.

5.4 Procedures and functions

5.4.1 DebugMessageName

Synopsis: Return the name of the debug message

Declaration: `function DebugMessageName(msgType: Integer) : String`

Visibility: default

Description: DebugMessageName returns the name of the message type. It can be used to examine the MsgType field of a TDebugMessage (92) record, and if msgType contains a known type, it returns a string describing this type.

Errors: If MsgType contains an unknown type, 'Unknown' is returned.

5.4.2 ReadDebugMessageFromStream

Synopsis: Read a message from stream

Declaration:
```
procedure ReadDebugMessageFromStream(AStream: TStream;
                                     var Msg: TDebugMessage)
```

Visibility: default

Description: ReadDebugMessageFromStream reads a TDebugMessage (92) record (Msg) from the stream AStream.

The record is not read in a byte-ordering safe way, i.e. it cannot be exchanged between little- and big-endian systems.

Errors: If the stream contains not enough bytes or is malformed, then an exception may be raised.

See also: TDebugMessage (92), WriteDebugMessageToStream (93)

5.4.3 WriteDebugMessageToStream

Synopsis: Write a message to stream

Declaration:
```
procedure WriteDebugMessageToStream(AStream: TStream;
                                    const Msg: TDebugMessage)
```

Visibility: default

Description: `WriteDebugMessageFromStream` writes a TDebugMessage (92) record (`Msg`) to the stream `AStream`.

The record is not written in a byte-ordering safe way, i.e. it cannot be exchanged between little- and big-endian systems.

Errors: A stream write error may occur if the stream cannot be written to.

See also: TDebugMessage (92), ReadDebugMessageToStream (91)

Chapter 6

Reference for unit 'ezcgi'

6.1 Used units

Table 6.1: Used units by unit 'ezcgi'

Name	Page
Classes	??
strings	94
sysutils	??

6.2 Overview

ezcgi, written by Michael Hess, provides a single class which offers simple access to the CGI environment which a CGI program operates under. It supports both GET and POST methods. It's intended for simple CGI programs which do not need full-blown CGI support. File uploads are not supported by this component.

To use the unit, a descendent of the `TEZCGI` class should be created and the DoPost (97) or DoGet (97) methods should be overidden.

6.3 Constants, types and variables

6.3.1 Constants

```
hexTable = '0123456789ABCDEF'
```

String constant used to convert a number to a hexadecimal code or back.

6.4 ECGIException

6.4.1 Description

Exception raised by TEZcgi (95)

6.5 TEZcgi

6.5.1 Description

TEZcgi implements all functionality to analyze the CGI environment and query the variables present in it. It's main use is the exposed variables.

Programs wishing to use this class should make a descendent class of this class and override the DoPost (97) or DoGet (97) methods. To run the program, an instance of this class must be created, and it's Run (96) method should be invoked. This will analyze the environment and call the DoPost or DoGet method, depending on what HTTP method was used to invoke the program.

6.5.2 Method overview

Page	Property	Description
95	Create	Creates a new instance of the TEZCGI component
95	Destroy	Removes the TEZCGI component from memory
97	DoGet	Method to handle GET requests
97	DoPost	Method to handle POST requests
97	GetValue	Return the value of a request variable.
96	PutLine	Send a line of output to the web-client
96	Run	Run the CGI application.
96	WriteContent	Writes the content type to standard output

6.5.3 Property overview

Page	Property	Access	Description
99	Email	rw	Email of the server administrator
99	Name	rw	Name of the server administrator
98	Names	r	Indexed array with available variable names.
97	Values	r	Variables passed to the CGI script
99	VariableCount	r	Number of available variables.
98	Variables	r	Indexed array with variables as name=value pairs.

6.5.4 TEZcgi.Create

Synopsis: Creates a new instance of the TEZCGI component

Declaration:
```
constructor Create
```

Visibility: public

Description: Create initializes the CGI program's environment: it reads the environment variables passed to the CGI program and stores them in the Variable (94) property.

See also: TZECGI.Variables (94), TZECGI.Names (94), TZECGI.Values (94)

6.5.5 TEZcgi.Destroy

Synopsis: Removes the TEZCGI component from memory

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` removes all variables from memory and then calls the inherited destroy, removing the `TEZCGI` instance from memory.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TEZcgi.Create (95)

6.5.6 TEZcgi.Run

Synopsis: Run the CGI application.

Declaration: `procedure Run`

Visibility: public

Description: `Run` analyses the variables passed to the application, processes the request variables (it stores them in the Variables (94) property) and calls the DoPost (97) or DoGet (97) methods, depending on the method passed to the web server.

After creating the instance of `TEZCGI`, the `Run` method is the only method that should be called when using this component.

See also: TZECGI.Variables (94), TEZCGI.DoPost (97), TEZCGI.DoGet (97)

6.5.7 TEZcgi.WriteContent

Synopsis: Writes the content type to standard output

Declaration: `procedure WriteContent(ctype: String)`

Visibility: public

Description: `WriteContent` writes the content type `cType` to standard output, followed by an empty line. After this method was called, no more HTTP headers may be written to standard output. Any HTTP headers should be written before `WriteContent` is called. It should be called from the DoPost (97) or DoGet (97) methods.

See also: TEZCGI.DoPost (97), TEZCGI.DoGet (97), TEZcgi.PutLine (96)

6.5.8 TEZcgi.PutLine

Synopsis: Send a line of output to the web-client

Declaration: `procedure PutLine(sOut: String)`

Visibility: public

Description: `PutLine` writes a line of text (`sOut`) to the web client (currently, to standard output). It should be called only after WriteContent (96) was called with a content type of `text`. The sent text is not processed in any way, i.e. no HTML entities or so are inserted instead of special HTML characters. This should be done by the user.

Errors: No check is performed whether the content type is right.

See also: TEZcgi.WriteContent (96)

6.5.9 TEZcgi.GetValue

Synopsis: Return the value of a request variable.

Declaration: `function GetValue(Index: String;defaultValue: String) : String`

Visibility: public

Description: `GetValue` returns the value of the variable named `Index`, and returns `DefaultValue` if it is empty or does not exist.

See also: TEZCGI.Values (97)

6.5.10 TEZcgi.DoPost

Synopsis: Method to handle `POST` requests

Declaration: `procedure DoPost;  Virtual`

Visibility: public

Description: `DoPost` is called by the Run (96) method the `POST` method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: TEZcgi.Run (96), TEZcgi.DoGet (97)

6.5.11 TEZcgi.DoGet

Synopsis: Method to handle `GET` requests

Declaration: `procedure DoGet;  Virtual`

Visibility: public

Description: `DoGet` is called by the Run (96) method the `GET` method was used to invoke the CGI application. It should be overridden in descendents of `TEZcgi` to actually handle the request.

See also: TEZcgi.Run (96), TEZcgi.DoPost (97)

6.5.12 TEZcgi.Values

Synopsis: Variables passed to the CGI script

Declaration: `Property Values[Index: String]: String`

Visibility: public

Access: Read

Description: `Values` is a name-based array of variables that were passed to the script by the web server or the HTTP request. The `Index` variable is the name of the variable whose value should be retrieved. The following standard values are available:

AUTH_TYPE Authorization type

CONTENT_LENGTH Content length

CONTENT_TYPE Content type

GATEWAY_INTERFACE Used gateway interface

PATH_INFO Requested URL

PATH_TRANSLATED Transformed URL

QUERY_STRING Client query string

REMOTE_ADDR Address of remote client

REMOTE_HOSTDNS name of remote client

REMOTE_IDENT Remote identity.

REMOTE_USER Remote user

REQUEST_METHOD Request methods (POST or GET)

SCRIPT_NAME Script name

SERVER_NAME Server host name

SERVER_PORT Server port

SERVER_PROTOCOL Server protocol

SERVER_SOFTWARE Web server software

HTTP_ACCEPT Accepted responses

HTTP_ACCEPT_CHARSET Accepted character sets

HTTP_ACCEPT_ENCODING Accepted encodings

HTTP_IF_MODIFIED_SINCE Proxy information

HTTP_REFERER Referring page

HTTP_USER_AGENT Client software name

Other than the standard list, any variables that were passed by the web-client request, are also available. Note that the variables are case insensitive.

See also: TEZCGI.Variables (98), TEZCGI.Names (98), TEZCGI.GetValue (97), TEZcgi.VariableCount (99)

6.5.13 TEZcgi.Names

Synopsis: Indexed array with available variable names.

Declaration: `Property Names[Index: Integer]: String`

Visibility: public

Access: Read

Description: `Names` provides indexed access to the available variable names. The `Index` may run from 0 to VariableCount (99). Any other value will result in an exception being raised.

See also: TEZcgi.Variables (98), TEZcgi.Values (97), TEZcgi.GetValue (97), TEZcgi.VariableCount (99)

6.5.14 TEZcgi.Variables

Synopsis: Indexed array with variables as name=value pairs.

Declaration: `Property Variables[Index: Integer]: String`

Visibility: public

Access: Read

Description: `Variables` provides indexed access to the available variable names and values. The variables are returned as `Name=Value` pairs. The `Index` may run from 0 to VariableCount (99). Any other value will result in an exception being raised.

See also: TEZcgi.Names (98), TEZcgi.Values (97), TEZcgi.GetValue (97), TEZcgi.VariableCount (99)

6.5.15 TEZcgi.VariableCount

Synopsis: Number of available variables.

Declaration: `Property VariableCount : Integer`

Visibility: public

Access: Read

Description: `TEZcgi.VariableCount` returns the number of available CGI variables. This includes both the standard CGI environment variables and the request variables. The actual names and values can be retrieved with the Names (98) and Variables (98) properties.

See also: TEZcgi.Names (98), TEZcgi.Variables (98), TEZcgi.Values (97), TEZcgi.GetValue (97)

6.5.16 TEZcgi.Name

Synopsis: Name of the server administrator

Declaration: `Property Name : String`

Visibility: public

Access: Read,Write

Description: `Name` is used when displaying an error message to the user. This should set prior to calling the TEZcgi.Run (96) method.

See also: TEZcgi.Run (96), TEZcgi.Email (99)

6.5.17 TEZcgi.Email

Synopsis: Email of the server administrator

Declaration: `Property Email : String`

Visibility: public

Access: Read,Write

Description: `Email` is used when displaying an error message to the user. This should set prior to calling the TEZcgi.Run (96) method.

See also: TEZcgi.Run (96), TEZcgi.Name (99)

Chapter 7

Reference for unit 'gettext'

7.1 Used units

Table 7.1: Used units by unit 'gettext'

Name	Page
Classes	??
sysutils	??

7.2 Overview

The gettext unit can be used to hook into the resource string mechanism of Free Pascal to provide translations of the resource strings, based on the GNU gettext mechanism. The unit provides a class (TMOFile (102)) to read the .mo files with localizations for various languages. It also provides a couple of calls to translate all resource strings in an application based on the translations in a .mo file.

7.3 Constants, types and variables

7.3.1 Constants

```
MOFileHeaderMagic = $950412de
```

This constant is found as the first integer in a .mo

7.3.2 Types

```
PLongWordArray = ^TLongWordArray
```

Pointer to a TLongWordArray (101) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (101) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (101) array.

```
TLongWordArray = Array[0..(1shl30)divSizeOf(LongWord)] of LongWord
```

`TLongWordArray` is an array used to define the PLongWordArray (100) pointer. A variable of type `TLongWordArray` should never be directly declared, as it would occupy too much memory. The `PLongWordArray` type can be used to allocate a dynamic number of elements.

```
TMOFileHeader = packed record
  magic : LongWord;
  revision : LongWord;
  nstrings : LongWord;
  OrigTabOffset : LongWord;
  TransTabOffset : LongWord;
  HashTabSize : LongWord;
  HashTabOffset : LongWord;
end
```

This structure describes the structure of a .mo file with string localizations.

```
TMOStringInfo = packed record
  length : LongWord;
  offset : LongWord;
end
```

This record is one element in the string tables describing the original and translated strings. It describes the position and length of the string. The location of these tables is stored in the TMOFileHeader (101) record at the start of the file.

```
TMOStringTable = Array[0..(1shl30)divSizeOf(TMOStringInfo)] of TMOStringInfo
```

`TMOStringTable` is an array type containing TMOStringInfo (101) records. It should never be used directly, as it would occupy too much memory.

```
TPCharArray = Array[0..(1shl30)divSizeOf(PChar)] of PChar
```

`TLongWordArray` is an array used to define the PPCharArray (101) pointer. A variable of type `TPCharArray` should never be directly declared, as it would occupy too much memory. The `PPCharArray` type can be used to allocate a dynamic number of elements.

7.4 Procedures and functions

7.4.1 GetLanguageIDs

Synopsis: Return the current language IDs

Declaration: `procedure GetLanguageIDs(var Lang: String;var FallbackLang: String)`

Visibility: default

Description: `GetLanguageIDs` returns the current language IDs (an ISO string) as returned by the operating system. On windows, the `GetUserDefaultLCID` and `GetLocaleInfo` calls are used. On other operating systems, the `LC_ALL`, `LC_MESSAGES` or `LANG` environment variables are examined.

7.4.2 TranslateResourceStrings

Synopsis: Translate the resource strings of the application.

Declaration:
```
procedure TranslateResourceStrings(AFile: TMOFile)
procedure TranslateResourceStrings(const AFilename: String)
```

Visibility: default

Description: `TranslateResourceStrings` translates all the resource strings in the application based on the values in the .mo file `AFileName` or `AFile`. The procedure creates an TMOFile (102) instance to read the .mo file if a filename is given.

Errors: If the file does not exist or is an invalid .mo file.

See also: TranslateUnitResourceStrings (102), TMOFile (102)

7.4.3 TranslateUnitResourceStrings

Synopsis: Translate the resource strings of a unit.

Declaration:
```
procedure TranslateUnitResourceStrings(const AUnitName: String;
                                       AFile: TMOFile)
procedure TranslateUnitResourceStrings(const AUnitName: String;
                                       const AFilename: String)
```

Visibility: default

Description: `TranslateUnitResourceStrings` is identical in function to TranslateResourceStrings (102), but translates the strings of a single unit (`AUnitName`) which was used to compile the application. This can be more convenient, since the resource string files are created on a unit basis.

See also: TranslateResourceStrings (102), TMOFile (102)

7.5 EMOFileError

7.5.1 Description

`EMOFileError` is raised in case an TMOFile (102) instance is created with an invalid .mo.

7.6 TMOFile

7.6.1 Description

`TMOFile` is a class providing easy access to a .mo file. It can be used to translate any of the strings that reside in the .mo file. The internal structure of the .mo is completely hidden.

7.6.2 Method overview

Page	Property	Description
103	Create	Create a new instance of the `TMOFile` class.
103	Destroy	Removes the `TMOFile` instance from memory
103	Translate	Translate a string

7.6.3 TMOFile.Create

Synopsis: Create a new instance of the `TMOFile` class.

Declaration:
```
constructor Create(const AFilename: String)
constructor Create(AStream: TStream)
```

Visibility: public

Description: `Create` creates a new instance of the `MOFile` class. It opens the file `AFileName` or the stream `AStream`. If a stream is provided, it should be seekable.

The whole contents of the file is read into memory during the `Create` call. This means that the stream is no longer needed after the `Create` call.

Errors: If the named file does not exist, then an exception may be raised. If the file does not contain a valid TMOFileHeader (101) structure, then an EMOFileError (102) exception is raised.

See also: TMOFile.Destroy (103)

7.6.4 TMOFile.Destroy

Synopsis: Removes the `TMOFile` instance from memory

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` cleans the internal structures with the contents of the .mo. After this the `TMOFile` instance is removed from memory.

See also: TMOFile.Create (103)

7.6.5 TMOFile.Translate

Synopsis: Translate a string

Declaration:
```
function Translate(AOrig: PChar;ALen: Integer;AHash: LongWord) : String
function Translate(AOrig: String;AHash: LongWord) : String
function Translate(AOrig: String) : String
```

Visibility: public

Description: `Translate` translates the string `AOrig`. The string should be in the .mo file as-is. The string can be given as a plain string, as a `PChar` (with length `ALen`). If the hash value (`AHash`) of the string is not given, it is calculated.

If the string is in the `.mo` file, the translated string is returned. If the string is not in the file, an empty string is returned.

Errors: None.

Chapter 8

Reference for unit 'idea'

8.1 Used units

Table 8.1: Used units by unit 'idea'

Name	Page
Classes	**??**
sysutils	**??**

8.2 Overview

Besides some low level IDEA encryption routines, the IDEA unit also offers 2 streams which offer on-the-fly encryption or decryption: there are 2 stream objects: A write-only encryption stream which encrypts anything that is written to it, and a decription stream which decrypts anything that is read from it.

8.3 Constants, types and variables

8.3.1 Constants

```
IDEABLOCKSIZE = 8
```

IDEA block size

```
IDEAKEYSIZE = 16
```

IDEA Key size constant.

```
KEYLEN = (6 * ROUNDS + 4 )
```

Key length

```
ROUNDS = 8
```

Number of rounds to encrypt

8.3.2 Types

`IdeaCryptData = TIdeaCryptData`

Provided for backward functionality.

`IdeaCryptKey = TIdeaCryptKey`

Provided for backward functionality.

`IDEAkey = TIDEAKey`

Provided for backward functionality.

```
TIdeaCryptData = Array[0..3] of Word
```

`TIdeaCryptData` is an internal type, defined to hold data for encryption/decryption.

```
TIdeaCryptKey = Array[0..7] of Word
```

The actual encryption or decryption key for IDEA is 64-bit long. This type is used to hold such a key. It can be generated with the EnKeyIDEA (106) or DeKeyIDEA (105) algorithms depending on whether an encryption or decryption key is needed.

```
TIDEAKey = Array[0..keylen-1] of Word
```

The IDEA key should be filled by the user with some random data (say, a passphrase). This key is used to generate the actual encryption/decryption keys.

8.4 Procedures and functions

8.4.1 CipherIdea

Synopsis: Encrypt or decrypt a buffer.

Declaration:
```
procedure CipherIdea(Input: TIdeaCryptData;var outdata: TIdeaCryptData;
                     z: TIDEAKey)
```

Visibility: default

Description: `CipherIdea` encrypts or decrypts a buffer with data (`Input`) using key `z`. The resulting encrypted or decrypted data is returned in `Output`.

Errors: None.

See also: EnKeyIdea (106), DeKeyIdea (105), TIDEAEncryptStream (107), TIDEADecryptStream (106)

8.4.2 DeKeyIdea

Synopsis: Create a decryption key from an encryption key.

Declaration:
```
procedure DeKeyIdea(z: TIDEAKey;var dk: TIDEAKey)
```

Visibility: default

Description: `DeKeyIdea` creates a decryption key based on the encryption key `z`. The decryption key is returned in `dk`. Note that only a decryption key generated from the encryption key that was used to encrypt the data can be used to decrypt the data.

Errors: None.

See also: EnKeyIdea (106), CipherIdea (105)

8.4.3 EnKeyIdea

Synopsis: Create an IDEA encryption key from a user key.

Declaration: `procedure EnKeyIdea(UserKey: TIdeaCryptKey;var z: TIDEAKey)`

Visibility: default

Description: `EnKeyIdea` creates an IDEA encryption key from user-supplied data in `UserKey`. The Encryption key is stored in `z`.

Errors: None.

See also: DeKeyIdea (105), CipherIdea (105)

8.5 EIDEAError

8.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

8.6 TIDEADeCryptStream

8.6.1 Description

`TIDEADecryptStream` is a stream which decrypts anything that is read from it using the IDEA mechanism. It reads the encrypted data from a source stream and decrypts it using the CipherIDEA (105) algorithm. It is a read-only stream: it is not possible to write data to this stream.

When creating a `TIDEADecryptStream` instance, an IDEA decryption key should be passed to the constructor, as well as the stream from which encrypted data should be read written.

The encrypted data can be created with a TIDEAEncryptStream (107) encryption stream.

8.6.2 Method overview

Page	Property	Description
106	Read	Reads data from the stream, decrypting it as needed
107	Seek	Set position on the stream
107	Write	Write data to the stream

8.6.3 TIDEADeCryptStream.Read

Synopsis: Reads data from the stream, decrypting it as needed

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` attempts to read `Count` bytes from the stream, placing them in `Buffer` the bytes are read from the source stream and decrypted as they are read. (bytes are read from the source stream in blocks of 8 bytes. The function returns the number of bytes actually read.

Errors: If an error occurs when reading data from the source stream, an exception may be raised.

See also: TIDEADecryptStream.Write (107), TIDEADecryptStream.Seek (107), TIDEAEncryptStream (107)

8.6.4 TIDEADeCryptStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` always raises an EIDEAError (106) exception, because the decryption stream is read-only. To write to an encryption stream, use the Write (108) method of the TIDEAEncryptStream (107) decryption stream.

Errors: An EIDEAError (106) exception is raised when calling this method.

See also: TIDEADecryptStream.Read (106), TIDEAEncryptStream (107), TIDEAEncryptStream.Write (108)

8.6.5 TIDEADeCryptStream.Seek

Synopsis: Set position on the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` will only work on a forward seek. It emulates a forward seek by reading and discarding bytes from the input stream. The `TIDEADecryptStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them.

Errors: An EIDEAError (106) exception is raised if the stream does not allow the requested seek operation.

See also: TIDEADeCryptStream.Read (106)

8.7 TIDEAEncryptStream

8.7.1 Description

`TIDEAEncryptStream` is a stream which encrypts anything that is written to it using the IDEA mechanism, and then writes the encrypted data to the destination stream using the CipherIDEA (105) algorithm. It is a write-only stream: it is not possible to read data from this stream.

When creating a `TIDEAEncryptStream` instance, an IDEA encryption key should be passed to the constructor, as well as the stream to which encrypted data should be written.

The resulting encrypted data can be read again with a TIDEADecryptStream (106) decryption stream.

8.7.2 Method overview

Page	Property	Description
108	Destroy	Flush data buffers and free the stream instance.
109	Flush	Write remaining bytes from the stream
108	Read	Read data from the stream
109	Seek	Set stream position
108	Write	Write bytes to the stream to be encrypted

8.7.3 TIDEAEncryptStream.Destroy

Synopsis: Flush data buffers and free the stream instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` flushes any data still remaining in the internal encryption buffer, and then calls the inherited `Destroy`

By default, the destination stream is not freed when the encryption stream is freed.

Errors: None.

See also: TIDEAStream.Create (110)

8.7.4 TIDEAEncryptStream.Read

Synopsis: Read data from the stream

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` always raises an EIDEAError (106) exception, because the encryption stream is write-only. To read from an encrypted stream, use the Read (106) method of the TIDEADecryptStream (106) decryption stream.

Errors: An EIDEAError (106) exception is raised when calling this method.

See also: TIDEAEncryptStream.Write (108), TIDEADecryptStream (106), TIDEADecryptStream.Read (106)

8.7.5 TIDEAEncryptStream.Write

Synopsis: Write bytes to the stream to be encrypted

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream, encrypting the bytes as they are written (encryption in blocks of 8 bytes).

Errors: If an error occurs writing to the destination stream, an error may occur.

See also: TIDEADecryptStream.Read (106)

8.7.6 TIDEAEncryptStream.Seek

Synopsis: Set stream position

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` return the current position if called with `0` and `soFromCurrent` as arguments. With all other values, it will always raise an exception, since it is impossible to set the position on an encryption stream.

Errors: An EIDEAError (106) will be raised unless called with `0` and `soFromCurrent` as arguments.

See also: TIDEAEncryptStream.Write (108), EIDEAError (106)

8.7.7 TIDEAEncryptStream.Flush

Synopsis: Write remaining bytes from the stream

Declaration: `procedure Flush`

Visibility: public

Description: `Flush` writes the current encryption buffer to the stream. Encryption always happens in blocks of 8 bytes, so if the buffer is not completely filled at the end of the writing operations, it must be flushed. It should never be called directly, unless at the end of all writing operations. It is called automatically when the stream is destroyed.

Errors: None.

See also: TIDEAEncryptStream.Write (108)

8.8 TIDEAStream

8.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (107) and TIDEADeCryptStream (106), and simply provides some fields that these descendent classes use when encrypting/decrypting. One of these classes should be created, depending on whether one wishes to encrypt or to decrypt.

8.8.2 Method overview

Page	Property	Description
110	Create	Creates a new instance of the `TIDEAStream` class

8.8.3 Property overview

Page	Property	Access	Description
110	Key	r	Key used when encrypting/decrypting

8.8.4 TIDEAStream.Create

Synopsis: Creates a new instance of the `TIDEAStream` class

Declaration: `constructor Create(AKey: TIDEAKey;Dest: TStream)`

Visibility: public

Description: `Create` stores the encryption/decryption key and then calls the inherited `Create` to store the `Dest` stream.

Errors: None.

See also: TIDEAEncryptStream (107), TIDEADeCryptStream (106)

8.8.5 TIDEAStream.Key

Synopsis: Key used when encrypting/decrypting

Declaration: `Property Key : TIDEAKey`

Visibility: public

Access: Read

Description: `Key` is the key as it was passed to the constructor of the stream. It cannot be changed while data is read or written. It is the key as it is used when encrypting/decrypting.

See also: CipherIdea (105)

Chapter 9

Reference for unit 'iostream'

9.1 Used units

Table 9.1: Used units by unit 'iostream'

Name	Page
Classes	**??**

9.2 Overview

The iostream implements a descendent of THandleStream (**??**) streams that can be used to read from standard input and write to standard output and standard diagnostic output (`stderr`).

9.3 Constants, types and variables

9.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 9.2: Enumeration values for type TIOSType

Value	Explanation
iosError	The stream can be used to write to standard diagnostic output
iosInput	The stream can be used to read from standard input
iosOutPut	The stream can be used to write to standard output

`TIOSType` is passed to the Create (112) constructor of TIOStream (112), it determines what kind of stream is created.

9.4 EIOStreamError

9.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (112).

9.5 TIOStream

9.5.1 Description

`TIOStream` can be used to create a stream which reads from or writes to the standard input, output or stderr file descriptors. It is a descendent of `THandleStream`. The type of stream that is created is determined by the TIOSType (111) argument to the constructor. The handle of the standard input, output or stderr file descriptors is determined automatically.

The `TIOStream` keeps an internal `Position`, and attempts to provide minimal Seek (113) behaviour based on this position.

9.5.2 Method overview

Page	Property	Description
112	Create	Construct a new instance of TIOStream (112)
112	Read	Read data from the stream.
113	Seek	Set the stream position
113	SetSize	Set the size of the stream
113	Write	Write data to the stream

9.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (112)

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (112), which can subsequently be used

Errors: No checking is performed to see whether the requested file descriptor is actually open for reading/writing. In that case, subsequent calls to `Read` or `Write` or `seek` will fail.

See also: TIOStream.Read (112), TIOStream.Write (113)

9.5.4 TIOStream.Read

Synopsis: Read data from the stream.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` checks first whether the type of the stream allows reading (type is `iosInput`). If not, it raises a EIOStreamError (112) exception. If the stream can be read, it calls the inherited `Read` to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow reading.

See also: TIOSType (111), TIOStream.Write (113)

9.5.5 TIOStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` checks first whether the type of the stream allows writing (type is `iosOutput` or `iosError`). If not, it raises a EIOStreamError (112) exception. If the stream can be written to, it calls the inherited `Write` to actually read the data.

Errors: An `EIOStreamError` exception is raised if the stream does not allow writing.

See also: TIOSType (111), TIOStream.Read (112)

9.5.6 TIOStream.SetSize

Synopsis: Set the size of the stream

Declaration: `procedure SetSize(NewSize: LongInt);  Override`

Visibility: public

Description: `SetSize` overrides the standard `SetSize` implementation. It always raises an exception, because the standard input, output and stderr files have no size.

Errors: An `EIOStreamError` exception is raised when this method is called.

See also: EIOStreamError (112)

9.5.7 TIOStream.Seek

Synopsis: Set the stream position

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. Normally, standard input, output and stderr are not seekable. The `TIOStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

All other cases will result in a `EIOStreamError` exception.

Errors: An EIOStreamError (112) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (112)

Chapter 10

Reference for unit 'Pipes'

10.1 Used units

Table 10.1: Used units by unit 'Pipes'

Name	Page
Classes	??
sysutils	??

10.2 Overview

The Pipes unit implements streams that are wrappers around the OS's pipe functionality. It creates a pair of streams, and what is written to one stream can be read from another.

10.3 Constants, types and variables

10.3.1 Constants

```
ENoReadMSg = 'Cannot read from OuputPipeStream.'
```

Constant used in ENoReadPipe (115) exception.

```
ENoSeekMsg = 'Cannot seek on pipes'
```

Constant used in EPipeSeek (116) exception.

```
ENoWriteMsg = 'Cannot write to InputPipeStream.'
```

Constant used in ENoWritePipe (115) exception.

```
EPipeMsg = 'Failed to create pipe.'
```

Constant used in EPipeCreation (115) exception.

10.4 Procedures and functions

10.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles

Declaration:
```
function CreatePipeHandles(var Inhandle: THandle;var OutHandle: THandle)
                           : Boolean
```

Visibility: default

Description: `CreatePipeHandles` provides an OS-independent way to create a set of pipe filehandles. These handles are inheritable to child processes. The reading end of the pipe is returned in `InHandle`, the writing end in `OutHandle`.

Errors: On error, `False` is returned.

See also: CreatePipeStreams (115)

10.4.2 CreatePipeStreams

Synopsis: Create a pair of pipe stream.

Declaration:
```
procedure CreatePipeStreams(var InPipe: TInputPipeStream;
                            var OutPipe: TOutputPipeStream)
```

Visibility: default

Description: `CreatePipeStreams` creates a set of pipe file descriptors with CreatePipeHandles (115), and if that call is succesfull, a pair of streams is created: `InPipe` and `OutPipe`.

Errors: If no pipe handles could be created, an EPipeCreation (115) exception is raised.

See also: CreatePipeHandles (115), TInputPipeStream (116), TOutputPipeStream (117)

10.5 ENoReadPipe

10.5.1 Description

Exception raised when a write operation is attempted on a write-only pipe.

10.6 ENoWritePipe

10.6.1 Description

Exception raised when a read operation is attempted on a read-only pipe.

10.7 EPipeCreation

10.7.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

10.8 EPipeError

10.8.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

10.9 EPipeSeek

10.9.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

10.10 TInputPipeStream

10.10.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (115) call to represent the reading end of a pipe. It is a TStream (**??**) descendent which does not allow writing, and which mimics the seek operation.

10.10.2 Method overview

Page	Property	Description
117	Read	Read data from the stream to a buffer.
116	Seek	Set the current position of the stream
116	Write	Write data to the stream.

10.10.3 Property overview

Page	Property	Access	Description
117	NumBytesAvailable	r	Number of bytes available for reading.

10.10.4 TInputPipeStream.Write

Synopsis: Write data to the stream.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` overrides the parent implementation of `Write`. On a `TInputPipeStream` will always raise an exception, as the pipe is read-only.

Errors: An ENoWritePipe (115) exception is raised when this function is called.

See also: TInputPipeStream.Read (117), TInputPipeStream.Seek (116)

10.10.5 TInputPipeStream.Seek

Synopsis: Set the current position of the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. Normally, pipe streams stderr are not seekable. The `TInputPipeStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

All other cases will result in a `EPipeSeek` exception.

Errors: An EPipeSeek (116) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (116), #rtl.classes.tstream.seek (**??**)

10.10.6 TInputPipeStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` calls the inherited read and adjusts the internal position pointer of the stream.

Errors: None.

See also: TInputPipeStream.Write (116), TInputPipeStream.Seek (116)

10.10.7 TInputPipeStream.NumBytesAvailable

Synopsis: Number of bytes available for reading.

Declaration: `Property NumBytesAvailable : DWord`

Visibility: public

Access: Read

Description: `NumBytesAvailable` is the number of bytes available for reading. This is the number of bytes in the OS buffer for the pipe. It is not a number of bytes in an internal buffer.

If this number is nonzero, then reading `NumBytesAvailable` bytes from the stream will not block the process. Reading more than `NumBytesAvailable` bytes will block the process, while it waits for the requested number of bytes to become available.

See also: TInputPipeStream.Read (117)

10.11 TOutputPipeStream

10.11.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (115) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

10.11.2 Method overview

Page	Property	Description
118	Read	Read data from the stream.
118	Seek	Sets the position in the stream

10.11.3 TOutputPipeStream.Seek

Synopsis: Sets the position in the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` is overridden in `TOutputPipeStream`. Calling this method will always raise an exception: an output pipe is not seekable.

Errors: An EPipeSeek (116) exception is raised if this method is called.

10.11.4 TOutputPipeStream.Read

Synopsis: Read data from the stream.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` overrides the parent `Read` implementation. It always raises an exception, because a output pipe is write-only.

Errors: An ENoReadPipe (115) exception is raised when this function is called.

See also: TOutputPipeStream.Seek (118)

Chapter 11

Reference for unit 'pooledmm'

11.1 Used units

Table 11.1: Used units by unit 'pooledmm'

Name	Page
Classes	??

11.2 Overview

pooledmm is a memory manager class which uses pools of blocks. Since it is a higher-level implementation of a memory manager which works on top of the FPC memory manager, It also offers more debugging and analysis tools. It is used mainly in the LCL and Lazarus IDE.

11.3 Constants, types and variables

11.3.1 Types

```
PPooledMemManagerItem = ^TPooledMemManagerItem
```

`PPooledMemManagerItem` is a pointer type, pointing to a TPooledMemManagerItem (120) item, used in a linked list.

```
TEnumItemsMethod = procedure(Item: Pointer) of object
```

`TEnumItemsMethod` is a prototype for the callback used in the TNonFreePooledMemManager.EnumerateItems (121) call. The parameter `Item` will be set to each of the pointers in the item list of TNonFreePooledMemManager (120).

```
TPooledMemManagerItem = record
  Next : PPooledMemManagerItem;
end
```

TPooledMemManagerItem is used internally by the TPooledMemManager (122) class to maintain the free list block. It simply points to the next free block.

11.4 TNonFreePooledMemManager

11.4.1 Description

TNonFreePooledMemManager keeps a list of fixed-size memory blocks in memory. Each block has the same size, making it suitable for storing a lot of records of the same type. It does not free the items stored in it, except when the list is cleared as a whole.

It allocates memory for the blocks in a exponential way, i.e. each time a new block of memory must be allocated, it's size is the double of the last block. The first block will contain 8 items.

11.4.2 Method overview

Page	Property	Description
120	Clear	Clears the memory
120	Create	Creates a new instance of TNonFreePooledMemManager
121	Destroy	Removes the TNonFreePooledMemManager instance from memory
121	EnumerateItems	Enumerate all items in the list
121	NewItem	Return a pointer to a new memory block

11.4.3 Property overview

Page	Property	Access	Description
121	ItemSize	r	Size of an item in the list

11.4.4 TNonFreePooledMemManager.Clear

Synopsis: Clears the memory

Declaration: `procedure Clear`

Visibility: public

Description: Clear clears all blocks from memory, freeing the allocated memory blocks. None of the pointers returned by NewItem (121) is valid after a call to Clear

See also: TNonFreePooledMemManager.NewItem (121)

11.4.5 TNonFreePooledMemManager.Create

Synopsis: Creates a new instance of TNonFreePooledMemManager

Declaration: `constructor Create(TheItemSize: Integer)`

Visibility: public

Description: Create creates a new instance of TNonFreePooledMemManager and sets the item size to TheItemSize.

Errors: If not enough memory is available, an exception may be raised.

See also: TNonFreePooledMemManager.ItemSize (121)

11.4.6 TNonFreePooledMemManager.Destroy

Synopsis: Removes the `TNonFreePooledMemManager` instance from memory

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the list, clears the internal structures, and then calls the inherited `Destroy`.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TNonFreePooledMemManager.Create (120), TNonFreePooledMemManager.Clear (120)

11.4.7 TNonFreePooledMemManager.NewItem

Synopsis: Return a pointer to a new memory block

Declaration: `function NewItem : Pointer`

Visibility: public

Description: `NewItem` returns a pointer to an unused memory block of size ItemSize (121). It will allocate new memory on the heap if necessary.

Note that there is no way to mark the memory block as free, except by clearing the whole list.

Errors: If no more memory is available, an exception may be raised.

See also: TNonFreePooledMemManager.Clear (120)

11.4.8 TNonFreePooledMemManager.EnumerateItems

Synopsis: Enumerate all items in the list

Declaration: `procedure EnumerateItems(const Method: TEnumItemsMethod)`

Visibility: public

Description: `EnumerateItems` will enumerate over all items in the list, passing the items to `Method`. This can be used to execute certain operations on all items in the list. (for example, simply list them)

11.4.9 TNonFreePooledMemManager.ItemSize

Synopsis: Size of an item in the list

Declaration: `Property ItemSize : Integer`

Visibility: public

Access: Read

Description: `ItemSize` is the size of a single block in the list. It's a fixed size determined when the list is created.

See also: TNonFreePooledMemManager.Create (120)

11.5 TPooledMemManager

11.5.1 Description

`TPooledMemManager` is a class which maintains a linked list of blocks, represented by the TPooledMemManagerItem (120) record. It should not be used directly, but should be descended from and the descendent should implement the actual memory manager.

11.5.2 Method overview

Page	Property	Description
122	Clear	Clears the list
122	Create	Creates a new instance of the `TPooledMemManager` class
122	Destroy	Removes an instance of `TPooledMemManager` class from memory

11.5.3 Property overview

Page	Property	Access	Description
124	AllocatedCount	r	Total number of allocated items in the list
123	Count	r	Number of items in the list
124	FreeCount	r	Number of free items in the list
124	FreedCount	r	Total number of freed items in the list.
123	MaximumFreeCountRatio	rw	Maximum ratio of free items over total items
123	MinimumFreeCount	rw	Minimum count of free items in the list

11.5.4 TPooledMemManager.Clear

Synopsis: Clears the list

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` clears the list, it disposes all items in the list.

See also: TPooledMemManager.FreedCount (124)

11.5.5 TPooledMemManager.Create

Synopsis: Creates a new instance of the `TPooledMemManager` class

Declaration: `constructor Create`

Visibility: public

Description: `Create` initializes all necessary properties and then calls the inherited create.

See also: TPooledMemManager.Destroy (122)

11.5.6 TPooledMemManager.Destroy

Synopsis: Removes an instance of `TPooledMemManager` class from memory

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` calls Clear (122) and then calls the inherited destroy.

`Destroy` should never be called directly. Instead `Free` should be used, or `FreeAndNil`

See also: TPooledMemManager.Create (122)

11.5.7 TPooledMemManager.MinimumFreeCount

Synopsis: Minimum count of free items in the list

Declaration: `Property MinimumFreeCount : Integer`

Visibility: public

Access: Read,Write

Description: `MinimumFreeCount` is the minimum number of free items in the linked list. When disposing an item in the list, the number of items is checked, and only if the required number of free items is present, the item is actually freed.

The default value is 100000

See also: TPooledMemManager.MaximumFreeCountRatio (123)

11.5.8 TPooledMemManager.MaximumFreeCountRatio

Synopsis: Maximum ratio of free items over total items

Declaration: `Property MaximumFreeCountRatio : Integer`

Visibility: public

Access: Read,Write

Description: `MaximumFreeCountRatio` is the maximum ratio (divided by 8) of free elements over the total amount of elements: When disposing an item in the list, if the number of free items is higher than this ratio, the item is freed.

The default value is 8.

See also: TPooledMemManager.MinimumFreeCount (123)

11.5.9 TPooledMemManager.Count

Synopsis: Number of items in the list

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the total number of items allocated from the list.

See also: TPooledMemManager.FreeCount (124), TPooledMemManager.AllocatedCount (124), TPooledMemManager.FreedCount (124)

11.5.10 TPooledMemManager.FreeCount

Synopsis: Number of free items in the list

Declaration: `Property FreeCount : Integer`

Visibility: public

Access: Read

Description: `FreeCount` is the current total number of free items in the list.

See also: TPooledMemManager.Count (123), TPooledMemManager.AllocatedCount (124), TPooledMemManager.FreedCount (124)

11.5.11 TPooledMemManager.AllocatedCount

Synopsis: Total number of allocated items in the list

Declaration: `Property AllocatedCount : Int64`

Visibility: public

Access: Read

Description: `AllocatedCount` is the total number of newly allocated items on the list.

See also: TPooledMemManager.Count (123), TPooledMemManager.FreeCount (124), TPooledMemManager.FreedCount (124)

11.5.12 TPooledMemManager.FreedCount

Synopsis: Total number of freed items in the list.

Declaration: `Property FreedCount : Int64`

Visibility: public

Access: Read

Description: `FreedCount` is the total number of elements actually freed in the list.

See also: TPooledMemManager.Count (123), TPooledMemManager.FreeCount (124), TPooledMemManager.AllocatedCount (124)

Chapter 12

Reference for unit 'process'

12.1 Used units

Table 12.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	114
sysutils	??

12.2 Overview

The Process unit contains the code for the TProcess (127) component, a cross-platform component to start and control other programs, offering also access to standard input and output for these programs.

`TProcess` does not handle wildcard expansion, does not support complex pipelines as in Unix. If this behaviour is desired, the shell can be executed with the pipeline as the command it should execute.

12.3 Constants, types and variables

12.3.1 Types

```
TProcessOption = (poRunSuspended,poWaitOnExit,poUsePipes,
                  poStderrToOutPut,poNoConsole,poNewConsole,
                  poDefaultErrorMode,poNewProcessGroup,poDebugProcess,
                  poDebugOnlyThisProcess)
```

When a new process is started using TProcess.Execute (129), these options control the way the process is started. Note that not all options are supported on all platforms.

```
TProcessOptions= Set of (poDebugOnlyThisProcess,poDebugProcess,
                         poDefaultErrorMode,poNewConsole,
```

Table 12.2: Enumeration values for type TProcessOption

Value	Explanation
poDebugOnlyThisProcess	Do not follow processes started by this process (Win32 only)
poDebugProcess	Allow debugging of the process (Win32 only)
poDefaultErrorMode	Use default error handling.
poNewConsole	Start a new console window for the process (Win32 only)
poNewProcessGroup	Start the process in a new process group (Win32 only)
poNoConsole	Do not allow access to the console window for the process (Win32 only)
poRunSuspended	Start the process in suspended state.
poStderrToOutPut	Redirect standard error to the standard output stream.
poUsePipes	Use pipes to redirect standard input and output.
poWaitOnExit	Wait for the process to terminate before returning.

```
                                poNewProcessGroup,poNoConsole,poRunSuspended,
                                poStderrToOutPut,poUsePipes,poWaitOnExit)
```

Set of TProcessOption (125).

```
TProcessPriority = (ppHigh,ppIdle,ppNormal,ppRealTime)
```

Table 12.3: Enumeration values for type TProcessPriority

Value	Explanation
ppHigh	The process runs at higher than normal priority.
ppIdle	The process only runs when the system is idle (i.e. has nothing else to do)
ppNormal	The process runs at normal priority.
ppRealTime	The process runs at real-time priority.

This enumerated type determines the priority of the newly started process. It translates to default platform specific constants. If finer control is needed, then platform-dependent mechanism need to be used to set the priority.

```
TShowWindowOptions = (swoNone,swoHIDE,swoMaximize,swoMinimize,
                      swoRestore,swoShow,swoShowDefault,
                      swoShowMaximized,swoShowMinimized,
                      swoshowMinNOActive,swoShowNA,swoShowNoActivate,
                      swoShowNormal)
```

This type describes what the new process' main window should look like. Most of these have only effect on Windows. They are ignored on other systems.

```
TStartupOption = (suoUseShowWindow,suoUseSize,suoUsePosition,
                  suoUseCountChars,suoUseFillAttribute)
```

These options are mainly for Win32, and determine what should be done with the application once it's started.

Table 12.4: Enumeration values for type TShowWindowOptions

Value	Explanation
swoHIDE	The main window is hidden.
swoMaximize	The main window is maximized.
swoMinimize	The main window is minimized.
swoNone	Allow system to position the window.
swoRestore	Restore the previous position.
swoShow	Show the main window.
swoShowDefault	When showing Show the main window on
swoShowMaximized	The main window is shown maximized
swoShowMinimized	The main window is shown minimized
swoshowMinNOActive	The main window is shown minimized but not activated
swoShowNA	The main window is shown but not activated
swoShowNoActivate	The main window is shown but not activated
swoShowNormal	The main window is shown normally

Table 12.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (127).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (127).
suoUsePosition	Use the window sizes as specified in TProcess (127).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (126)
suoUseSize	Use the window sizes as specified in TProcess (127)

```
TStartupOptions= Set of (suoUseCountChars,suoUseFillAttribute,
                         suoUsePosition,suoUseShowWindow,suoUseSize)
```

Set of TStartUpOption (126).

12.4 EProcess

12.4.1 Description

Exception raised when an error occurs in a TProcess routine.

12.5 TProcess

12.5.1 Description

`TProcess` is a component that can be used to start and control other processes (programs/binaries). It contains a lot of options that control how the process is started. Many of these are Win32 specific, and have no effect on other platforms, so they should be used with care.

The simplest way to use this component is to create an instance, set the CommandLine (135) property to the full pathname of the program that should be executed, and call Execute (129). To determine whether the process is still running (i.e. has not stopped executing), the Running (139) property can be checked.

More advanced techniques can be used with the Options (137) settings.

12.5.2 Method overview

Page	Property	Description
130	CloseInput	Close the input stream of the process
130	CloseOutput	Close the output stream of the process
130	CloseStderr	Close the error stream of the process
129	Create	Create a new instance of the `TProcess` class.
129	Destroy	Destroy this instance of `TProcess`
129	Execute	Execute the program with the given options
130	Resume	Resume execution of a suspended process
131	Suspend	Suspend a running process
131	Terminate	Terminate a running process
131	WaitOnExit	Wait for the program to stop executing.

12.5.3 Property overview

Page	Property	Access	Description
135	Active	rw	Start or stop the process.
135	ApplicationName	rw	Name of the application to start
135	CommandLine	rw	Command-line to execute
136	ConsoleTitle	rw	Title of the console window
136	CurrentDirectory	rw	Working directory of the process.
136	Desktop	rw	Desktop on which to start the process.
137	Environment	rw	Environment variables for the new process
134	ExitStatus	r	Exit status of the process.
142	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
132	Handle	r	Handle of the process
135	InheritHandles	rw	Should the created process inherit the open handles of the current process.
133	Input	r	Stream connected to standard input of the process.
137	Options	rw	Options to be used when starting the process.
134	Output	r	Stream connected to standard output of the process.
138	Priority	rw	Priority at which the process is running.
132	ProcessHandle	r	Alias for Handle (132)
133	ProcessID	r	ID of the process.
139	Running	r	Determines wheter the process is still running.
139	ShowWindow	rw	Determines how the process main window is shown (Windows only)
138	StartupOptions	rw	Additional (Windows) startup options
134	Stderr	r	Stream connected to standard diagnostic output of the process.
132	ThreadHandle	r	Main process thread handle
133	ThreadID	r	ID of the main process thread
140	WindowColumns	rw	Number of columns in console window (windows only)
140	WindowHeight	rw	Height of the process main window
140	WindowLeft	rw	X-coordinate of the initial window (Windows only)
132	WindowRect	rw	Positions for the main program window.
141	WindowRows	rw	Number of rows in console window (Windows only)
141	WindowTop	rw	Y-coordinate of the initial window (Windows only)
141	WindowWidth	rw	Height of the process main window (Windows only)

12.5.4 TProcess.Create

Synopsis: Create a new instance of the `TProcess` class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` creates a new instance of the `TProcess` class. After calling the inherited constructor, it simply sets some default values.

12.5.5 TProcess.Destroy

Synopsis: Destroy this instance of `TProcess`

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` cleans up this instance of `TProcess`. Prior to calling the inherited destructor, it cleans up any streams that may have been created. If a process was started and is still executed, it is *not* stopped, but the standard input/output/stderr streams are no longer available, because they have been destroyed.

Errors: None.

See also: TProcess.Create (129)

12.5.6 TProcess.Execute

Synopsis: Execute the program with the given options

Declaration: `procedure Execute; Virtual`

Visibility: public

Description: `Execute` actually executes the program as specified in CommandLine (135), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (137), then the call will only return when the program has finished executing (or if an error occured). If this option is not given, the call returns immediatly, but the WaitOnExit (131) call can be used to wait for it to close, or the Running (139) call can be used to check whether it is still running.

The TProcess.Terminate (131) call can be used to terminate the program if it is still running, or the Suspend (131) call can be used to temporarily stop the program's execution.

The ExitStatus (134) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (127) exception is raised.

See also: TProcess.Running (139), TProcess.WaitOnExit (131), TProcess.Terminate (131), TProcess.Suspend (131), TProcess.Resume (130), TProcess.ExitStatus (134)

12.5.7 TProcess.CloseInput

Synopsis: Close the input stream of the process

Declaration: `procedure CloseInput; Virtual`

Visibility: public

Description: `CloseInput` closes the input file descriptor of the process, that is, it closes the handle of the pipe to standard input of the process.

See also: TProcess.Input (133), TProcess.StdErr (134), TProcess.Output (134), TProcess.CloseOutput (130), TProcess.CloseStdErr (130)

12.5.8 TProcess.CloseOutput

Synopsis: Close the output stream of the process

Declaration: `procedure CloseOutput; Virtual`

Visibility: public

Description: `CloseOutput` closes the output file descriptor of the process, that is, it closes the handle of the pipe to standard output of the process.

See also: TProcess.Output (134), TProcess.Input (133), TProcess.StdErr (134), TProcess.CloseInput (130), TProcess.CloseStdErr (130)

12.5.9 TProcess.CloseStderr

Synopsis: Close the error stream of the process

Declaration: `procedure CloseStderr; Virtual`

Visibility: public

Description: `CloseStdErr` closes the standard error file descriptor of the process, that is, it closes the handle of the pipe to standard error output of the process.

See also: TProcess.Output (134), TProcess.Input (133), TProcess.StdErr (134), TProcess.CloseInput (130), TProcess.CloseStdErr (130)

12.5.10 TProcess.Resume

Synopsis: Resume execution of a suspended process

Declaration: `function Resume : Integer; Virtual`

Visibility: public

Description: `Resume` should be used to let a suspended process resume it's execution. It should be called in particular when the `poRunSuspended` flag is set in Options (137).

Errors: None.

See also: TProcess.Suspend (131), TProcess.Options (137), TProcess.Execute (129), TProcess.Terminate (131)

12.5.11 TProcess.Suspend

Synopsis: Suspend a running process

Declaration: `function Suspend : Integer;  Virtual`

Visibility: public

Description: `Suspend` suspends a running process. If the call is successful, the process is suspended: it stops running, but can be made to execute again using the Resume (130) call.

`Suspend` is fundamentally different from TProcess.Terminate (131) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (137), TProcess.Resume (130), TProcess.Terminate (131), TProcess.Execute (129)

12.5.12 TProcess.Terminate

Synopsis: Terminate a running process

Declaration: `function Terminate(AExitCode: Integer) : Boolean;  Virtual`

Visibility: public

Description: `Terminate` stops the execution of the running program. It effectively stops the program.

On Windows, the program will report an exit code of `AExitCode`, on other systems, this value is ignored.

Errors: On error, a nonzero value is returned.

See also: TProcess.ExitStatus (134), TProcess.Suspend (131), TProcess.Execute (129), TProcess.WaitOnExit (131)

12.5.13 TProcess.WaitOnExit

Synopsis: Wait for the program to stop executing.

Declaration: `function WaitOnExit : Boolean`

Visibility: public

Description: `WaitOnExit` waits for the running program to exit. It returns `True` if the wait was succesful, or `False` if there was some error waiting for the program to exit.

Note that the return value of this function has changed. The old return value was a DWord with a platform dependent error code. To make things consistent and cross-platform, a boolean return type was used.

Errors: On error, `False` is returned. No extended error information is available, as it is highly system dependent.

See also: TProcess.ExitStatus (134), TProcess.Terminate (131), TProcess.Running (139)

12.5.14 TProcess.WindowRect

Synopsis: Positions for the main program window.

Declaration: `Property WindowRect : Trect`

Visibility: public

Access: Read,Write

Description: `WindowRect` can be used to specify the position of

12.5.15 TProcess.Handle

Synopsis: Handle of the process

Declaration: `Property Handle : THandle`

Visibility: public

Access: Read

Description: `Handle` identifies the process. In Unix systems, this is the process ID. On windows, this is the process handle. It can be used to signal the process.

The handle is only valid after TProcess.Execute (129) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (132), TProcess.ProcessID (133), TProcess.ThreadID (133)

12.5.16 TProcess.ProcessHandle

Synopsis: Alias for Handle (132)

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (132) and is provided for completeness only.

See also: TProcess.Handle (132), TProcess.ThreadHandle (132), TProcess.ProcessID (133), TProcess.ThreadID (133)

12.5.17 TProcess.ThreadHandle

Synopsis: Main process thread handle

Declaration: `Property ThreadHandle : THandle`

Visibility: public

Access: Read

Description: `ThreadHandle` is the main process thread handle. On Unix, this is the same as the process ID, on Windows, this may be a different handle than the process handle.

The handle is only valid after TProcess.Execute (129) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (132), TProcess.ProcessID (133), TProcess.ThreadID (133)

12.5.18 TProcess.ProcessID

Synopsis: ID of the process.

Declaration: `Property ProcessID : Integer`

Visibility: public

Access: Read

Description: `ProcessID` is the ID of the process. It is the same as the handle of the process on Unix systems, but on Windows it is different from the process Handle.

The ID is only valid after TProcess.Execute (129) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (132), TProcess.ThreadHandle (132), TProcess.ThreadID (133)

12.5.19 TProcess.ThreadID

Synopsis: ID of the main process thread

Declaration: `Property ThreadID : Integer`

Visibility: public

Access: Read

Description: `ProcessID` is the ID of the main process thread. It is the same as the handle of the main proces thread (or the process itself) on Unix systems, but on Windows it is different from the thread Handle.

The ID is only valid after TProcess.Execute (129) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (133), TProcess.Handle (132), TProcess.ThreadHandle (132)

12.5.20 TProcess.Input

Synopsis: Stream connected to standard input of the process.

Declaration: `Property Input : TOutputPipeStream`

Visibility: public

Access: Read

Description: `Input` is a stream which is connected to the process' standard input file handle. Anything written to this stream can be read by the process.

The `Input` stream is only instantiated when the `poUsePipes` flag is used in Options (137).

Note that writing to the stream may cause the calling process to be suspended when the created process is not reading from it's input, or to cause errors when the process has terminated.

See also: TProcess.OutPut (134), TProcess.StdErr (134), TProcess.Options (137), TProcessOption (125)

12.5.21 TProcess.Output

Synopsis: Stream connected to standard output of the process.

Declaration: `Property Output : TInputPipeStream`

Visibility: public

Access: Read

Description: `Output` is a stream which is connected to the process' standard output file handle. Anything written to standard output by the created process can be read from this stream.

The `Output` stream is only instantiated when the `poUsePipes` flag is used in Options (137).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (137).

Note that reading from the stream may cause the calling process to be suspended when the created process is not writing anything to standard output, or to cause errors when the process has terminated.

See also: TProcess.InPut (133), TProcess.StdErr (134), TProcess.Options (137), TProcessOption (125)

12.5.22 TProcess.Stderr

Synopsis: Stream connected to standard diagnostic output of the process.

Declaration: `Property Stderr : TInputPipeStream`

Visibility: public

Access: Read

Description: `StdErr` is a stream which is connected to the process' standard diagnostic output file handle (`StdErr`). Anything written to standard diagnostic output by the created process can be read from this stream.

The `StdErr` stream is only instantiated when the `poUsePipes` flag is used in Options (137).

The `Output` stream equals the Output (134) when the `poStdErrToOutPut` flag is used in Options (137).

Note that reading from the stream may cause the calling process to be suspended when the created process is not writing anything to standard output, or to cause errors when the process has terminated.

See also: TProcess.InPut (133), TProcess.Output (134), TProcess.Options (137), TProcessOption (125)

12.5.23 TProcess.ExitStatus

Synopsis: Exit status of the process.

Declaration: `Property ExitStatus : Integer`

Visibility: public

Access: Read

Description: `ExitStatus` contains the exit status as reported by the process when it stopped executing. The value of this property is only meaningful when the process is no longer running. If it is not running then the value is zero.

See also: TProcess.Running (139), TProcess.Terminate (131)

12.5.24 TProcess.InheritHandles

Synopsis: Should the created process inherit the open handles of the current process.

Declaration: `Property InheritHandles : Boolean`

Visibility: public

Access: Read,Write

Description: `InheritHandles` determines whether the created process inherits the open handles of the current process (value `True`) or not (`False`).

On Unix, setting this variable has no effect.

See also: TProcess.InPut (133), TProcess.Output (134), TProcess.StdErr (134)

12.5.25 TProcess.Active

Synopsis: Start or stop the process.

Declaration: `Property Active : Boolean`

Visibility: published

Access: Read,Write

Description: `Active` starts the process if it is set to `True`, or terminates the process if set to `False`. It's mostly intended for use in an IDE.

See also: TProcess.Execute (129), TProcess.Terminate (131)

12.5.26 TProcess.ApplicationName

Synopsis: Name of the application to start

Declaration: `Property ApplicationName : String`

Visibility: published

Access: Read,Write

Description: `ApplicationName` is an alias for TProcess.CommandLine (135). It's mostly foruse in the Windows `CreateProcess` call. If `CommandLine` is not set, then `ApplicationName` will be used instead.

Note that either `CommandLine` or `ApplicationName` must be set prior to calling `Execute`.

See also: TProcess.CommandLine (135)

12.5.27 TProcess.CommandLine

Synopsis: Command-line to execute

Declaration: `Property CommandLine : String`

Visibility: published

Access: Read,Write

Description: `CommandLine` is the command-line to be executed: this is the name of the program to be executed, followed by any options it should be passed.

If the command to be executed or any of the arguments contains whitespace (space, tab character, linefeed character) it should be enclosed in single or double quotes.

If no absolute pathname is given for the command to be executed, it is searched for in the `PATH` environment variable. On Windows, the current directory always will be searched first. On other platforms, this is not so.

Note that either `CommandLine` or `ApplicationName` must be set prior to calling `Execute`.

See also: TProcess.ApplicationName (135)

12.5.28 TProcess.ConsoleTitle

Synopsis: Title of the console window

Declaration: `Property ConsoleTitle : String`

Visibility: published

Access: Read,Write

Description: `ConsoleTitle` is used on Windows when executing a console application: it specifies the title caption of the console window. On other platforms, this property is currently ignored.

Changing this property after the process was started has no effect.

See also: TProcess.WindowColumns (140), TProcess.WindowRows (141)

12.5.29 TProcess.CurrentDirectory

Synopsis: Working directory of the process.

Declaration: `Property CurrentDirectory : String`

Visibility: published

Access: Read,Write

Description: `CurrentDirectory` specifies the working directory of the newly started process.

Changing this property after the process was started has no effect.

See also: TProcess.Environment (137)

12.5.30 TProcess.Desktop

Synopsis: Desktop on which to start the process.

Declaration: `Property Desktop : String`

Visibility: published

Access: Read,Write

Description: `DeskTop` is used on Windows to determine on which desktop the process' main window should be shown. Leaving this empty means the process is started on the same desktop as the currently running process.

Changing this property after the process was started has no effect.

On unix, this parameter is ignored.

See also: TProcess.Input (133), TProcess.Output (134), TProcess.StdErr (134)

12.5.31 TProcess.Environment

Synopsis: Environment variables for the new process

Declaration: `Property Environment : TStrings`

Visibility: published

Access: Read,Write

Description: `Environment` contains the environment for the new process; it's a list of `Name=Value` pairs, one per line.

If it is empty, the environment of the current process is passed on to the new process.

See also: TProcess.Options (137)

12.5.32 TProcess.Options

Synopsis: Options to be used when starting the process.

Declaration: `Property Options : TProcessOptions`

Visibility: published

Access: Read,Write

Description: `Options` determine how the process is started. They should be set before the Execute (129) call is made.

Table 12.6:

option	Meaning
`poRunSuspended`	Start the process in suspended state.
`poWaitOnExit`	Wait for the process to terminate before returning.
`poUsePipes`	Use pipes to redirect standard input and output.
`poStderrToOutPut`	Redirect standard error to the standard output stream.
`poNoConsole`	Do not allow access to the console window for the process (Win32 only)
`poNewConsole`	Start a new console window for the process (Win32 only)
`poDefaultErrorMode`	Use default error handling.
`poNewProcessGroup`	Start the process in a new process group (Win32 only)
`poDebugProcess`	Allow debugging of the process (Win32 only)
`poDebugOnlyThisProcess`	Do not follow processes started by this process (Win32 only)

See also: TProcessOption (125), TProcessOptions (126), TProcess.Priority (138), TProcess.StartUpOptions (138)

12.5.33 TProcess.Priority

Synopsis: Priority at which the process is running.

Declaration: `Property Priority : TProcessPriority`

Visibility: published

Access: Read,Write

Description: `Priority` determines the priority at which the process is running.

Table 12.7:

Priority	Meaning
`ppHigh`	The process runs at higher than normal priority.
`ppIdle`	The process only runs when the system is idle (i.e. has nothing else to do)
`ppNormal`	The process runs at normal priority.
`ppRealTime`	The process runs at real-time priority.

Note that not all priorities can be set by any user. Usually, only users witha dministrative rights (the root user on Unix) can set a higher process priority.

On unix, the process priority is mapped on `Nice` values as follows:

Table 12.8:

Priority	Nice value
`ppHigh`	20
`ppIdle`	20
`ppNormal`	0
`ppRealTime`	-20

See also: TProcessPriority (126)

12.5.34 TProcess.StartupOptions

Synopsis: Additional (Windows) startup options

Declaration: `Property StartupOptions : TStartupOptions`

Visibility: published

Access: Read,Write

Description: `StartUpOptions` contains additional startup options, used mostly on Windows system. They determine which other window layout properties are taken into account when starting the new process.

See also: TProcess.ShowWindow (139), TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.FillAttribute (142)

Table 12.9:

Priority	Meaning
`suoUseShowWindow`	Use the Show Window options specified in ShowWindow (139)
`suoUseSize`	Use the specified window sizes
`suoUsePosition`	Use the specified window sizes.
`suoUseCountChars`	Use the specified console character width.
`suoUseFillAttribute`	Use the console fill attribute specified in FillAttribute (142).

12.5.35 TProcess.Running

Synopsis: Determines wheter the process is still running.

Declaration: `Property Running : Boolean`

Visibility: published

Access: Read

Description: `Running` can be read to determine whether the process is still running.

See also: TProcess.Terminate (131), TProcess.Active (135), TProcess.ExitStatus (134)

12.5.36 TProcess.ShowWindow

Synopsis: Determines how the process main window is shown (Windows only)

Declaration: `Property ShowWindow : TShowWindowOptions`

Visibility: published

Access: Read,Write

Description: `ShowWindow` determines how the process' main window is shown. It is useful only on Windows.

Table 12.10:

Option	Meaning
`swoNone`	Allow system to position the window.
`swoHIDE`	The main window is hidden.
`swoMaximize`	The main window is maximized.
`swoMinimize`	The main window is minimized.
`swoRestore`	Restore the previous position.
`swoShow`	Show the main window.
`swoShowDefault`	When showing Show the main window on a default position
`swoShowMaximized`	The main window is shown maximized
`swoShowMinimized`	The main window is shown minimized
`swoshowMinNOActive`	The main window is shown minimized but not activated
`swoShowNA`	The main window is shown but not activated
`swoShowNoActivate`	The main window is shown but not activated
`swoShowNormal`	The main window is shown normally

12.5.37 TProcess.WindowColumns

Synopsis: Number of columns in console window (windows only)

Declaration: `Property WindowColumns : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowColumns` is the number of columns in the console window, used to run the command in. This property is only effective if `suoUseCountChars` is specified in StartupOptions (138)

See also: TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowRows (141), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.38 TProcess.WindowHeight

Synopsis: Height of the process main window

Declaration: `Property WindowHeight : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowHeight` is the initial height (in pixels) of the process' main window. This property is only effective if `suoUseSize` is specified in StartupOptions (138)

See also: TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.39 TProcess.WindowLeft

Synopsis: X-coordinate of the initial window (Windows only)

Declaration: `Property WindowLeft : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowLeft` is the initial X coordinate (in pixels) of the process' main window, relative to the left border of the desktop. This property is only effective if `suoUsePosition` is specified in StartupOptions (138)

See also: TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.40 TProcess.WindowRows

Synopsis: Number of rows in console window (Windows only)

Declaration: `Property WindowRows : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowRows` is the number of rows in the console window, used to run the command in. This property is only effective if `suoUseCountChars` is specified in StartupOptions (138)

See also: TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.41 TProcess.WindowTop

Synopsis: Y-coordinate of the initial window (Windows only)

Declaration: `Property WindowTop : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowTop` is the initial Y coordinate (in pixels) of the process' main window, relative to the top border of the desktop. This property is only effective if `suoUsePosition` is specified in StartupOptions (138)

See also: TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.42 TProcess.WindowWidth

Synopsis: Height of the process main window (Windows only)

Declaration: `Property WindowWidth : Cardinal`

Visibility: published

Access: Read,Write

Description: `WindowWidth` is the initial width (in pixels) of the process' main window. This property is only effective if `suoUseSize` is specified in StartupOptions (138)

See also: TProcess.WindowHeight (140), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.FillAttribute (142), TProcess.StartupOptions (138)

12.5.43 TProcess.FillAttribute

Synopsis: Color attributes of the characters in the console window (Windows only)

Declaration: `Property FillAttribute : Cardinal`

Visibility: published

Access: Read,Write

Description: `FillAttribute` is a WORD value which specifies the background and foreground colors of the console window.

See also: TProcess.WindowHeight (140), TProcess.WindowWidth (141), TProcess.WindowLeft (140), TProcess.WindowTop (141), TProcess.WindowColumns (140), TProcess.WindowRows (141), TProcess.StartupOptions (138)

Chapter 13

Reference for unit 'streamcoll'

13.1 Used units

Table 13.1: Used units by unit 'streamcoll'

Name	Page
Classes	**??**
sysutils	**??**

13.2 Overview

The streamcoll unit contains the implentation of a collection (and corresponding collection item) which implements routines for saving or loading the collection to/from a stream. The collection item should implement 2 routines to implement the streaming; the streaming itself is not performed by the TStreamCollection (146) collection item.

The streaming performed here is not compatible with the streaming implemented in the `Classes` unit for components. It is independent of the latter and can be used without a component to hold the collection.

The collection item introduces mostly protected methods, and the unit contains a lot of auxiliary routines which aid in streaming.

13.3 Procedures and functions

13.3.1 ColReadBoolean

Synopsis: Read a boolean value from a stream

Declaration: `function ColReadBoolean(S: TStream) : Boolean`

Visibility: default

Description: `ColReadBoolean` reads a boolean from the stream `S` as it was written by ColWriteBoolean (145) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (144), ColWriteBoolean (145), ColReadString (145), ColReadInteger (144), ColReadFloat (144), ColReadCurrency (144)

13.3.2 ColReadCurrency

Synopsis: Read a currency value from the stream

Declaration: `function ColReadCurrency(S: TStream) : Currency`

Visibility: default

Description: `ColReadCurrency` reads a currency value from the stream `S` as it was written by ColWriteCurrency (145) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (144), ColReadBoolean (143), ColReadString (145), ColReadInteger (144), ColReadFloat (144), ColWriteCurrency (145)

13.3.3 ColReadDateTime

Synopsis: Read a `TDateTime` value from a stream

Declaration: `function ColReadDateTime(S: TStream) : TDateTime`

Visibility: default

Description: `ColReadDateTime` reads a currency value from the stream `S` as it was written by ColWriteDateTime (145) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColWriteDateTime (145), ColReadBoolean (143), ColReadString (145), ColReadInteger (144), ColReadFloat (144), ColReadCurrency (144)

13.3.4 ColReadFloat

Synopsis: Read a floating point value from a stream

Declaration: `function ColReadFloat(S: TStream) : Double`

Visibility: default

Description: `ColReadFloat` reads a double value from the stream `S` as it was written by ColWriteFloat (146) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (144), ColReadBoolean (143), ColReadString (145), ColReadInteger (144), ColWriteFloat (146), ColReadCurrency (144)

13.3.5 ColReadInteger

Synopsis: Read a 32-bit integer from a stream.

Declaration: `function ColReadInteger(S: TStream) : Integer`

Visibility: default

Description: `ColReadInteger` reads a 32-bit integer from the stream `S` as it was written by ColWriteInteger (146) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (144), ColReadBoolean (143), ColReadString (145), ColWriteInteger (146), ColReadFloat (144), ColReadCurrency (144)

13.3.6 ColReadString

Synopsis: Read a string from a stream

Declaration: `function ColReadString(S: TStream) : String`

Visibility: default

Description: `ColReadStream` reads a string value from the stream `S` as it was written by ColWriteString (146) and returns the read value. The value cannot be read and written across systems that have different endian values.

See also: ColReadDateTime (144), ColReadBoolean (143), ColWriteString (146), ColReadInteger (144), ColReadFloat (144), ColReadCurrency (144)

13.3.7 ColWriteBoolean

Synopsis: Write a boolean to a stream

Declaration: `procedure ColWriteBoolean(S: TStream;AValue: Boolean)`

Visibility: default

Description: `ColWriteBoolean` writes the boolean `AValue` to the stream. `S`.

See also: ColReadBoolean (143), ColWriteString (146), ColWriteInteger (146), ColWriteCurrency (145), ColWriteDateTime (145), ColWriteFloat (146)

13.3.8 ColWriteCurrency

Synopsis: Write a currency value to stream

Declaration: `procedure ColWriteCurrency(S: TStream;AValue: Currency)`

Visibility: default

Description: `ColWriteCurrency` writes the currency `AValue` to the stream `S`.

See also: ColWriteBoolean (145), ColWriteString (146), ColWriteInteger (146), ColWriteDateTime (145), ColWriteFloat (146), ColReadCurrency (144)

13.3.9 ColWriteDateTime

Synopsis: Write a `TDateTime` value to stream

Declaration: `procedure ColWriteDateTime(S: TStream;AValue: TDateTime)`

Visibility: default

Description: `ColWriteDateTime` writes the `TDateTimeAValue` to the stream `S`.

See also: ColReadDateTime (144), ColWriteBoolean (145), ColWriteString (146), ColWriteInteger (146), ColWriteFloat (146), ColWriteCurrency (145)

13.3.10 ColWriteFloat

Synopsis: Write floating point value to stream

Declaration: `procedure ColWriteFloat(S: TStream;AValue: Double)`

Visibility: default

Description: `ColWriteFloat` writes the double `AValue` to the stream `S`.

See also: ColWriteDateTime (145), ColWriteBoolean (145), ColWriteString (146), ColWriteInteger (146), ColReadFloat (144), ColWriteCurrency (145)

13.3.11 ColWriteInteger

Synopsis: Write a 32-bit integer to a stream

Declaration: `procedure ColWriteInteger(S: TStream;AValue: Integer)`

Visibility: default

Description: `ColWriteInteger` writes the 32-bit integer `AValue` to the stream `S`. No endianness is observed.

See also: ColWriteBoolean (145), ColWriteString (146), ColReadInteger (144), ColWriteCurrency (145), ColWriteDateTime (145)

13.3.12 ColWriteString

Synopsis: Write a string value to the stream

Declaration: `procedure ColWriteString(S: TStream;AValue: String)`

Visibility: default

Description: `ColWriteString` writes the string value `AValue` to the stream `S`.

See also: ColWriteBoolean (145), ColReadString (145), ColWriteInteger (146), ColWriteCurrency (145), ColWriteDateTime (145), ColWriteFloat (146)

13.4 EStreamColl

13.4.1 Description

Exception raised when an error occurs when streaming the collection.

13.5 TStreamCollection

13.5.1 Description

`TStreamCollection` is a TCollection (**??**) descendent which implements 2 calls LoadFromStream (147) and SaveToStream (147) which load and save the contents of the collection to a stream.

The collection items must be descendents of the TStreamCollectionItem (148) class for the streaming to work correctly.

Note that the stream must be used to load collections of the same type.

13.5.2 Method overview

Page	Property	Description
147	LoadFromStream	Load the collection from a stream
147	SaveToStream	Load the collection from the stream.

13.5.3 Property overview

Page	Property	Access	Description
147	Streaming	r	Indicates whether the collection is currently being written to stream

13.5.4 TStreamCollection.LoadFromStream

Synopsis: Load the collection from a stream

Declaration: `procedure LoadFromStream(S: TStream)`

Visibility: public

Description: `LoadFromStream` loads the collection from the stream `S`, if the collection was saved using SaveToStream (147). It reads the number of items in the collection, and then creates and loads the items one by one from the stream.

Errors: An exception may be raised if the stream contains invalid data.

See also: TStreamCollection.SaveToStream (147)

13.5.5 TStreamCollection.SaveToStream

Synopsis: Load the collection from the stream.

Declaration: `procedure SaveToStream(S: TStream)`

Visibility: public

Description: `SaveToStream` saves the collection to the stream `S` so it can be read from the stream with LoadFromStream (147). It does this by writing the number of collection items to the stream, and then streaming all items in the collection by calling their `SaveToStream` method.

Errors: None.

See also: TStreamCollection.LoadFromStream (147)

13.5.6 TStreamCollection.Streaming

Synopsis: Indicates whether the collection is currently being written to stream

Declaration: `Property Streaming : Boolean`

Visibility: public

Access: Read

Description: `Streaming` is set to `True` if the collection is written to or loaded from stream, and is set again to `False` if the streaming process is finished.

See also: TStreamCollection.LoadFromStream (147), TStreamCollection.SaveToStream (147)

13.6 TStreamCollectionItem

13.6.1 Description

`TStreamCollectionItem` is a TCollectionItem (**??**) descendent which implements 2 abstract routines: `LoadFromStream` and `SaveToStream` which must be overridden in a descendent class.

These 2 routines will be called by the TStreamCollection (146) to save or load the item from the stream.

Chapter 14

Reference for unit 'streamex'

14.1 Used units

Table 14.1: Used units by unit 'streamex'

Name	Page
Classes	**??**

14.2 Overview

streamex implements some extensions to be used together with streams from the classes unit.

14.3 TBidirBinaryObjectReader

14.3.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectReader (**??**), which implements the necessary support for BiDi data: the position in the stream (not available in the standard streaming) is emulated.

14.3.2 Property overview

Page	Property	Access	Description
149	Position	rw	Position in the stream

14.3.3 TBidirBinaryObjectReader.Position

Synopsis: Position in the stream

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: `Position` exposes the position of the stream in the reader for use in the TDelphiReader (150) class.

See also: TDelphiReader (150)

14.4 TBidirBinaryObjectWriter

14.4.1 Description

`TBidirBinaryObjectReader` is a class descendent from TBinaryObjectWriter (**??**), which implements the necessary support for BiDi data.

14.4.2 Property overview

Page	Property	Access	Description
150	Position	rw	Position in the stream

14.4.3 TBidirBinaryObjectWriter.Position

Synopsis: Position in the stream

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: `Position` exposes the position of the stream in the writer for use in the TDelphiWriter (151) class.

See also: TDelphiWriter (151)

14.5 TDelphiReader

14.5.1 Description

`TDelphiReader` is a descendent of `TReader` which has support for BiDi Streaming. It overrides the stream reading methods for strings, and makes sure the stream can be positioned in the case of strings. For this purpose, it makes use of the TBidirBinaryObjectReader (149) driver class.

14.5.2 Method overview

Page	Property	Description
151	GetDriver	Return the driver class as a TBidirBinaryObjectReader (149) class
151	Read	Read data from stream
151	ReadStr	Overrides the standard `ReadStr` method

14.5.3 Property overview

Page	Property	Access	Description
151	Position	rw	Position in the stream

14.5.4 TDelphiReader.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectReader (149) class

Declaration: `function GetDriver : TBidirBinaryObjectReader`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectReader (149) class.

See also: TBidirBinaryObjectReader (149)

14.5.5 TDelphiReader.ReadStr

Synopsis: Overrides the standard `ReadStr` method

Declaration: `function ReadStr : String`

Visibility: public

Description: `ReadStr` makes sure the TBidirBinaryObjectReader (149) methods are used, to store additional information about the stream position when reading the strings.

See also: TBidirBinaryObjectReader (149)

14.5.6 TDelphiReader.Read

Synopsis: Read data from stream

Declaration: `procedure Read(var Buf;Count: LongInt);  Override`

Visibility: public

Description: `Read` reads raw data from the stream. It reads `Count` bytes from the stream and places them in `Buf`. It forces the use of the TBidirBinaryObjectReader (149) class when reading.

See also: TBidirBinaryObjectReader (149), TDelphiReader.Position (151)

14.5.7 TDelphiReader.Position

Synopsis: Position in the stream

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: Position in the stream.

See also: TDelphiReader.Read (151)

14.6 TDelphiWriter

14.6.1 Description

`TDelphiWriter` is a descendent of `TWriter` which has support for BiDi Streaming. It overrides the stream writing methods for strings, and makes sure the stream can be positioned in the case of strings. For this purpose, it makes use of the TBidirBinaryObjectWriter (150) driver class.

14.6.2 Method overview

Page	Property	Description
152	FlushBuffer	Flushes the stream buffer
152	GetDriver	Return the driver class as a TBidirBinaryObjectWriter (150) class
152	Write	Write raw data to the stream
152	WriteStr	Write a string to the stream
153	WriteValue	Write value type

14.6.3 Property overview

Page	Property	Access	Description
153	Position	rw	Position in the stream

14.6.4 TDelphiWriter.GetDriver

Synopsis: Return the driver class as a TBidirBinaryObjectWriter (150) class

Declaration: `function GetDriver : TBidirBinaryObjectWriter`

Visibility: public

Description: `GetDriver` simply returns the used driver and typecasts it as TBidirBinaryObjectWriter (150) class.

See also: TBidirBinaryObjectWriter (150)

14.6.5 TDelphiWriter.FlushBuffer

Synopsis: Flushes the stream buffer

Declaration: `procedure FlushBuffer`

Visibility: public

Description: `FlushBuffer` flushes the internal buffer of the writer. It simply calls the `FlushBuffer` method of the driver class.

14.6.6 TDelphiWriter.Write

Synopsis: Write raw data to the stream

Declaration: `procedure Write(const Buf;Count: LongInt);  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buf` to the buffer, updating the position as needed.

14.6.7 TDelphiWriter.WriteStr

Synopsis: Write a string to the stream

Declaration: `procedure WriteStr(const Value: String)`

Visibility: public

Description: `WriteStr` writes a string to the stream, forcing the use of the TBidirBinaryObjectWriter (150) class methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (150)

14.6.8 TDelphiWriter.WriteValue

Synopsis: Write value type

Declaration: `procedure WriteValue(Value: TValueType)`

Visibility: public

Description: `WriteValue` overrides the same method in `TWriter` to force the use of the TBidirBinaryObjectWriter (150) methods, which update the position of the stream.

See also: TBidirBinaryObjectWriter (150)

14.6.9 TDelphiWriter.Position

Synopsis: Position in the stream

Declaration: `Property Position : LongInt`

Visibility: public

Access: Read,Write

Description: `Position` exposes the position in the stream as exposed by the TBidirBinaryObjectWriter (150) instance used when streaming.

See also: TBidirBinaryObjectWriter (150)

Chapter 15

Reference for unit 'StreamIO'

15.1 Used units

Table 15.1: Used units by unit 'StreamIO'

Name	Page
Classes	**??**
sysutils	**??**

15.2 Overview

The StreamIO unit implements a call to reroute the input or output of a text file to a descendents of TStream (**??**).

This allows to use the standard pascal Read (**??**) and Write (**??**) functions (with all their possibilities), on streams.

15.3 Procedures and functions

15.3.1 AssignStream

Synopsis: Assign a text file to a stream.

Declaration: `procedure AssignStream(var F: Textfile;Stream: TStream)`

Visibility: default

Description: `AssignStream` assigns the stream `Stream` to file `F`. The file can subsequently be used to write to the stream, using the standard Write (**??**) calls.

Before writing, call Rewrite (**??**) on the stream. Before reading, call Reset (**??**).

Errors: if `Stream` is `Nil`, an exception will be raised.

See also: #rtl.classes.TStream (**??**), GetStream (155)

15.3.2 GetStream

Synopsis: Return the stream, associated with a file.

Declaration: `function GetStream(var F: TTextRec) : TStream`

Visibility: default

Description: `GetStream` returns the instance of the stream that was associated with the file `F` using AssignStream (154).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (154), #rtl.classes.TStream (**??**)

Chapter 16

Reference for unit 'zstream'

16.1 Used units

Table 16.1: Used units by unit 'zstream'

Name	Page
Classes	**??**
paszlib	156
sysutils	**??**
zbase	156

16.2 Overview

The ZStream unit implements a TStream (**??**) descendent (TCompressionStream (157)) which uses the deflate algorithm to compress everything that is written to it. The compressed data is written to the output stream, which is specified when the compressor class is created.

Likewise, a `TStream` descendent is implemented which reads data from an input stream (TDecompressionStream (160)) and decompresses it with the inflate algorithm.

16.3 Constants, types and variables

16.3.1 Types

```
TCompressionLevel = (clNone,clFastest,clDefault,clMax)
```

Compression level for the deflate algorithm

```
TGZOpenMode = (gzOpenRead,gzOpenWrite)
```

Open mode for gzip file.

Table 16.2: Enumeration values for type TCompressionLevel

Value	Explanation
clDefault	Use default compression
clFastest	Use fast (but less) compression.
clMax	Use maximum compression
clNone	Do not use compression, just copy data.

Table 16.3: Enumeration values for type TGZOpenMode

Value	Explanation
gzOpenRead	Open file for reading
gzOpenWrite	Open file for writing

16.4 ECompressionError

16.4.1 Description

`ECompressionError` is the exception class used by the TCompressionStream (157) class.

16.5 EDecompressionError

16.5.1 Description

`EDecompressionError` is the exception class used by the TDeCompressionStream (160) class.

16.6 EZlibError

16.6.1 Description

Errors which occur in the zstream unit are signaled by raising an `EZLibError` exception descendent.

16.7 TCompressionStream

16.7.1 Description

`TCompressionStream`

16.7.2 Method overview

Page	Property	Description
158	Create	Create a new instance of the compression stream.
158	Destroy	Flushe data to the output stream and destroys the compression stream.
158	Read	Overridden to raise an exception.
159	Seek	Overrides seek to raise an exception.
159	Write	Write data to the stream

16.7.3 Property overview

Page	Property	Access	Description
159	CompressionRate	r	Running compression rate of compression stream
159	OnProgress		Progress handler

16.7.4 TCompressionStream.Create

Synopsis: Create a new instance of the compression stream.

Declaration:
```
constructor Create(CompressionLevel: TCompressionLevel;Dest: TStream;
                   ASkipHeader: Boolean)
```

Visibility: public

Description: `Create` creates a new instance of the compression stream. It merely calls the inherited constructor with the destination stream `Dest` and stores the compression level.

If `ASkipHeader` is set to `True`, the method will not write the block header to the stream. This is required for deflated data in a zip file.

Note that the compressed data is only completely written after the compression stream is destroyed.

See also: TCompressionStream.Destroy (158)

16.7.5 TCompressionStream.Destroy

Synopsis: Flushe data to the output stream and destroys the compression stream.

Declaration:
```
destructor Destroy;  Override
```

Visibility: public

Description: `Destroy` flushes the output stream: any compressed data not yet written to the output stream are written, and the deflate structures are cleaned up.

Errors: None.

See also: TCompressionStream.Create (158)

16.7.6 TCompressionStream.Read

Synopsis: Overridden to raise an exception.

Declaration:
```
function Read(var Buffer;Count: LongInt) : LongInt;  Override
```

Visibility: public

Description: The `Read` method of `TStream` is overridden, and always raises an exception, because `TCompressionStream` is write-only.

Errors: An ECompressionError (157) exception is raised.

See also: ECompressionError (157), TCompressionStream.Write (159)

16.7.7 TCompressionStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` takes `Count` bytes from `Buffer` and comresseses (deflates) them. The compressed result is written to the output stream.

Errors: If an error occurs, an ECompressionError (157) exception is raised.

See also: TCompressionStream.Read (158), TCompressionStream.Seek (159)

16.7.8 TCompressionStream.Seek

Synopsis: Overrides seek to raise an exception.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: The `Seek` method of `TStream` is overridden, and always raises an exception, because `TCompressionStream` is write-only, and cannot seek.

Errors: An ECompressionError (157) exception is raised.

See also: ECompressionError (157), TCompressionStream.Read (158), TCompressionStream.Write (159)

16.7.9 TCompressionStream.CompressionRate

Synopsis: Running compression rate of compression stream

Declaration: `Property CompressionRate : extended`

Visibility: public

Access: Read

Description: The `Compressionrate` is updated as more data is written to the stream and represents the ratio of outputted data versus written data.

See also: TCompressionStream.Write (159)

16.7.10 TCompressionStream.OnProgress

Synopsis: Progress handler

Declaration: `Property OnProgress :`

Visibility: public

Access:

Description: `OnProgress` is called whenever output data is written to the output stream. It can be used to update a progress bar or so. The `Sender` argument to the progress handler is the compression stream instance.

16.8 TCustomZlibStream

16.8.1 Description

`TCustomZlibStream` serves as the ancestor class for the TCompressionStream (157) and TDecompressionStream (160) classes.

It introduces support for a progess handler, and stores the input or output stream.

16.8.2 Method overview

Page	Property	Description
160	Create	Create a new instance of `TCustomZlibStream`

16.8.3 TCustomZlibStream.Create

Synopsis: Create a new instance of `TCustomZlibStream`

Declaration: `constructor Create(Strm: TStream)`

Visibility: public

Description: `Create` creates a new instance of `TCustomZlibStream`. It stores a reference to the input/output stream, and initializes the deflate compression mechanism so they can be used by the descendents.

See also: TCompressionStream (157), TDecompressionStream (160)

16.9 TDecompressionStream

16.9.1 Description

`TDecompressionStream` performs the inverse operation of TCompressionStream (157). A read operation reads data from an input stream and decompresses (inflates) the data it as it goes along.

The decompression stream reads it's compressed data from a stream with deflated data. This data can be created e.g. with a TCompressionStream (157) compression stream.

16.9.2 Method overview

Page	Property	Description
161	Create	Creates a new instance of the `TDecompressionStream` stream
161	Destroy	Destroys the `TDecompressionStream` instance
161	Read	Read data from the compressed stream
162	Seek	Move stream position to a certain location in the stream.
161	Write	Write data to the stream

16.9.3 Property overview

Page	Property	Access	Description
162	OnProgress		Progress handler

16.9.4 TDecompressionStream.Create

Synopsis: Creates a new instance of the `TDecompressionStream` stream

Declaration: `constructor Create(ASource: TStream;ASkipHeader: Boolean)`

Visibility: public

Description: `Create` creates and initializes a new instance of the `TDecompressionStream` class. It calls the inherited `Create` and passes it the `Source` stream. The source stream is the stream from which the compressed (deflated) data is read.

If `ASkipHeader` is true, then the gzip data header is skipped, allowing `TDecompressionStream` to read deflated data in a .zip file. (this data does not have the gzip header record prepended to it).

Note that the source stream is by default not owned by the decompression stream, and is not freed when the decompression stream is destroyed.

See also: TDecompressionStream.Destroy (161)

16.9.5 TDecompressionStream.Destroy

Synopsis: Destroys the `TDecompressionStream` instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the inflate structure, and then simply calls the inherited destroy.

By default the source stream is not freed when calling `Destroy`.

See also: TDecompressionStream.Create (161)

16.9.6 TDecompressionStream.Read

Synopsis: Read data from the compressed stream

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` will read data from the compressed stream until the decompressed data size is `Count` or there is no more compressed data available. The decompressed data is written in `Buffer`. The function returns the number of bytes written in the buffer.

Errors: If an error occurs, an EDeCompressionError (157) exception is raised.

See also: TCompressionStream.Write (159)

16.9.7 TDecompressionStream.Write

Synopsis: Write data to the stream

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` will raise a EDeCompressionError (157) exception, because the `TDecompressionStream` class is read-only.

Errors: An EDeCompressionError (157) exception is always raised.

See also: TDeCompressionStream.Read (161), EDeCompressionError (157)

16.9.8 TDecompressionStream.Seek

Synopsis: Move stream position to a certain location in the stream.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` overrides the standard `Seek` implementation. Normally, pipe streams stderr are not seekable. The `TDecompressionStream` stream tries to provide seek capabilities for the following limited number of cases:

Origin=soFromBeginning If `Offset` is larger than the current position, then the remaining bytes are skipped by reading them from the stream and discarding them.

Origin=soFromCurrent If `Offset` is zero, the current position is returned. If it is positive, then `Offset` bytes are skipped by reading them from the stream and discarding them, if the stream is of type `iosInput`.

All other cases will result in a `EPipeSeek` exception.

Errors: An EDecompressionError (157) exception is raised if the stream does not allow the requested seek operation.

See also: TDecompressionStream.Read (161)

16.9.9 TDecompressionStream.OnProgress

Synopsis: Progress handler

Declaration: `Property OnProgress :`

Visibility: public

Access:

Description: `OnProgress` is called whenever input data is read from the source stream. It can be used to update a progress bar or so. The `Sender` argument to the progress handler is the decompression stream instance.

16.10 TGZFileStream

16.10.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

16.10.2 Method overview

Page	Property	Description
163	Create	Create a new instance of `TGZFileStream`
163	Destroy	Removes `TGZFileStream` instance
163	Read	Read data from the compressed file
164	Seek	Set the position in the compressed stream.
164	Write	Write data to be compressed

16.10.3 TGZFileStream.Create

Synopsis: Create a new instance of `TGZFileStream`

Declaration: `constructor Create(FileName: String;FileMode: TGZOpenMode)`

Visibility: public

Description: `Create` creates a new instance of the `TGZFileStream` class. It opens `FileName` for reading or writing, depending on the `FileMode` parameter. It is not possible to open the file read-write. If the file is opened for reading, it must exist.

If the file is opened for reading, the TGZFileStream.Read (163) method can be used for reading the data in uncompressed form.

If the file is opened for writing, any data written using the TGZFileStream.Write (164) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (157) exception is raised.

See also: TGZFileStream.Destroy (163), TGZOpenMode (156)

16.10.4 TGZFileStream.Destroy

Synopsis: Removes `TGZFileStream` instance

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` closes the file and releases the `TGZFileStream` instance from memory.

See also: TGZFileStream.Create (163)

16.10.5 TGZFileStream.Read

Synopsis: Read data from the compressed file

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Read` overrides the `Read` method of `TStream` to read the data from the compressed file. The `Buffer` parameter indicates where the read data should be stored. The `Count` parameter specifies the number of bytes (*uncompressed*) that should be read from the compressed file. Note that it is not possible to read from the stream if it was opened in write mode.

The function returns the number of uncompressed bytes actually read.

Errors: If `Buffer` points to an invalid location, or does not have enough room for `Count` bytes, an exception will be raised.

See also: TGZFileStream.Create (163), TGZFileStream.Write (164), TGZFileStream.Seek (164)

16.10.6 TGZFileStream.Write

Synopsis: Write data to be compressed

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt;  Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the compressed file. The data is compressed as it is written, so ideally, less than `Count` bytes end up in the compressed file. Note that it is not possible to write to the stream if it was opened in read mode.

The function returns the number of (uncompressed) bytes that were actually written.

Errors: In case of an error, an EZlibError (157) exception is raised.

See also: TGZFileStream.Create (163), TGZFileStream.Read (163), TGZFileStream.Seek (164)

16.10.7 TGZFileStream.Seek

Synopsis: Set the position in the compressed stream.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt;  Override`

Visibility: public

Description: `Seek` sets the position to `Offset` bytes, starting from `Origin`. Not all combinations are possible, see TDecompressionStream.Seek (162) for a list of possibilities.

Errors: In case an impossible combination is asked, an EZlibError (157) exception is raised.

See also: TDecompressionStream.Seek (162)