Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 2.1
September 2006

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	??

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 (15) for encoding, and TBase64DecodingStream (13) 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 TBase64DecodingStream

1.3.1 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.

1.3.2 Method overview

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

1.3.3 Property overview

Page	Property	Access	Description
15	EOF	r	

1.3.4 TBase64DecodingStream.Create

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

Declaration: `constructor Create(AInputStream: TStream)`

Visibility: public

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

See also: TBase64EncodingStream.Create (16)

1.3.5 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 (15), TBase64DecodingStream.Read (14)

1.3.6 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 (14), TBase64DecodingStream.Seek (15), #rtl.classes.TStream.Read (**??**)

1.3.7 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 (15) instance.

Errors:

See also: TBase64DecodingStream.Read (14), TBase64DecodingStream.Seek (15), TBase64EncodingStream.Write (16), #rtl.classes.TStream.Write (**??**)

1.3.8 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 (14), TBase64DecodingStream.Write (14), TBase64EncodingStream.Seek (17), #rtl.classes.TStream.Seek (**??**)

1.3.9 TBase64DecodingStream.EOF

Synopsis:

Declaration: `Property EOF : Boolean`

Visibility: public

Access: Read

Description:

1.4 TBase64EncodingStream

1.4.1 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.

1.4.2 Method overview

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

1.4.3 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 (16), TBase64DecodingStream.Create (14)

1.4.4 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 (16)

1.4.5 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 (16), TBase64EncodingStream.Seek (17), TBase64DecodingStream.Read (14), #rtl.classes.TStream.Read (**??**)

1.4.6 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 (17), TBase64EncodingStream.Read (16), TBase64DecodingStream.Write (14), #rtl.classes.TStream.Write (**??**)

1.4.7 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 (16), TBase64EncodingStream.Write (16), #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 (21) which is for reading only, and TWriteBufStream (22) 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 (21) and TWriteBufStream (22) 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
19	Create	Create a new TBufStream instance.
19	Destroy	Destroys the TBufStream instance

2.4.3 Property overview

Page	Property	Access	Description
19	Buffer	r	The current buffer
20	BufferPos	r	Current buffer position.
20	BufferSize	r	Amount of data in the buffer
20	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 (18) is used as the capacity.

An instance of TBufStream should never be instantiated directly. Instead, an instance of TReadBufStream (21) or TWriteBufStream (22) should be created.

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

See also: TBufStream.Destroy (19), TReadBufStream (21), TWriteBufStream (22)

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 (19), TReadBufStream (21), TWriteBufStream (22)

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 (19), TBufStream.Capacity (20), TBufStream.BufferSize (20)

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 (19), TBufStream.Buffer (19), TBufStream.BufferSize (20), TBufStream.BufferPos (20)

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 (19), TBufStream.Buffer (19), TBufStream.BufferSize (20), TBufStream.Capacity (20)

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 (20).

See also: TBufStream.Create (19), TBufStream.Buffer (19), TBufStream.BufferPos (20), TBufStream.Capacity (20)

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
21	Read	Reads data from the stream
21	Seek	Set location in the buffer
21	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 (22), TReadBufStream.Read (21), TReadBufStream.Write (21)

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 (21), TReadBufStream.Read (21)

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 (22) write stream must be used to write data in a buffered way.

See also: TReadBufStream.Seek (21), TReadBufStream.Read (21)

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
22	Destroy	Remove the `TWriteBufStream` instance from memory
23	Read	Read data from the stream
22	Seek	Set stream position.
23	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 (19).

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

See also: TBufStream.Create (19), TBufStream.Destroy (19)

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 (23), TWriteBufStream.Read (23), TReadBufStream.Seek (21)

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 (21) should be used.

See also: TWriteBufStream.Seek (22), TWriteBufStream.Write (23), TReadBufStream.Read (21)

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 (22), TWriteBufStream.Read (23), TReadBufStream.Write (21)

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

3.3 Constants, types and variables

3.3.1 Types

```
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.

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

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

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

`TObjectListCallback` is used as the prototype for the TFPObjectList.ForEachCall (43) 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 (43) 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.

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 TFPHashTable (31) hash class. It's Robert Sedgwick's "Algorithms in C" hash function.

Errors: None.

See also: TFPHashTable (31)

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
26	Add	Add a new class pointer to the list.
26	Extract	Extract a class pointer from the list.
27	First	Return first non-nil class pointer
27	IndexOf	Search for a class pointer in the list.
28	Insert	Insert a new class pointer in the list.
27	Last	Return last non-`Nil` class pointer
27	Remove	Remove a class pointer from the list.

3.7.3 Property overview

Page	Property	Access	Description
28	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 (26), #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 (27), #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 (26), #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 (27), TClassList.Pack (25)

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 (27), TClassList.Pack (25)

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 (26), TClassList.Remove (27)

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 (46) 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 (46), `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
29	Add	Add a component to the list.
29	Destroy	Destroys the instance
29	Extract	Remove a component from the list without destroying it.
30	First	First non-nil instance in the list.
30	IndexOf	Search for an instance in the list
31	Insert	Insert a new component in the list
30	Last	Last non-nil instance in the list.
29	Remove	Remove a component from the list, possibly destroying it.

3.8.3 Property overview

Page	Property	Access	Description
31	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 (46), #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 (24)

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 (46) 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 (29), TObjectList.Extract (48)

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 (46) so it only accepts `TComponent` descendents. It introduces no new behaviour.

Errors: None.

See also: TComponentList.Extract (29), TObjectList.Remove (48)

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 (48)

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 (30), TObjectList.First (49)

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 (30), TObjectList.Last (49)

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 (49), TComponentList.Add (29), TComponentList.Remove (29)

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 (50)

3.9 TFPHashTable

3.9.1 Description

`TFPHashTable` 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 (25).

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
33	Add	Add a new key and its associated data to the hash.
33	ChangeTableSize	Change the table size of the hash table.
33	Clear	Clear the hash table.
32	Create	Instantiate a new `TFPHashTable` instance using the default hash mechanism
32	CreateWith	Instantiate a new `TFPHashTable` instance with given algorithm and size
33	Delete	Delete a key from the hash list.
32	Destroy	Free the hash table.
34	Find	Search for an item with a certain key value.
34	IsEmpty	Check if the hash table is empty.

3.9.3 Property overview

Page	Property	Access	Description
36	AVGChainLen	r	Average chain length
35	Count	r	Number of items in the hash table.
37	Density	r	Number of filled slots
34	HashFunction	rw	Hash function currently in use
35	HashTable	r	Hash table instance
35	HashTableSize	rw	Size of the hash table
35	Items	rw	Indexed access to the data pointer.
36	LoadFactor	r	Fraction of count versus size
36	MaxChainLength	r	Maximum chain length
37	NumberOfCollisions	r	Number of extra items
36	VoidSlots	r	Number of empty slots in the hash table.

3.9.4 TFPHashTable.Create

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

Declaration:
```
constructor Create
```

Visibility: public

Description: `Create` creates a new instance of `TFPHashTable` with hash size 196613 and hash algorithm RSHash (25)

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

See also: TFPHashTable.CreateWith (32)

3.9.5 TFPHashTable.CreateWith

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

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

Visibility: public

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

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

See also: TFPHashTable.Create (32)

3.9.6 TFPHashTable.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: TFPHashTable.Destroy (32), TFPHashTable.Create (32), TFPHashTable.CreateWith (32), THTNode.Data (46)

3.9.7 TFPHashTable.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: TFPHashTable.HashTableSize (35)

3.9.8 TFPHashTable.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: TFPHashTable.Destroy (32)

3.9.9 TFPHashTable.Add

Synopsis: Add a new key and its associated data to the hash.

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

Visibility: public

Description: `Add` calculates the hash value of `aKey` and adds key and it's associated data to the corresponding hash chain.

A given key can only be added once. It is an error to attempt to add the same key value twice.

Errors: If the key is already in the list, adding it a second time will raise an EDuplicate (25).

See also: TFPHashTable.Find (34), TFPHashTable.Delete (33)

3.9.10 TFPHashTable.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: TFPHashTable.Find (34), TFPHashTable.Add (33)

3.9.11 TFPHashTable.Find

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

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

Visibility: public

Description: `Find` searches for the THTNode (45) 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 (33) method should be used instead.

Errors: None.

See also: TFPHashTable.Add (33), TFPHashTable.Delete (33)

3.9.12 TFPHashTable.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: TFPHashTable.Count (35), TFPHashTable.HashTableSize (35), TFPHashTable.AVGChainLen (36), TFPHashTable.MaxChainLength (36)

3.9.13 TFPHashTable.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 (31). 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: TFPHashTable.SetHashFunction (31), TFPHashTable.HashTableSize (35)

3.9.14 TFPHashTable.Count

Synopsis: Number of items in the hash table.

Declaration: `Property Count : Int64`

Visibility: public

Access: Read

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

See also: TFPHashTable.IsEmpty (34), TFPHashTable.HashTableSize (35), TFPHashTable.AVGChainLen (36), TFPHashTable.MaxChainLength (36)

3.9.15 TFPHashTable.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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.AVGChainLen (36), TFPHashTable.MaxChainLength (36), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37)

3.9.16 TFPHashTable.Items

Synopsis: Indexed access to the data pointer.

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

Visibility: public

Access: Read,Write

Description: `Item` allows indexed access to the data pointers. When reading the property, if `Index` exists, then the associated data pointer is returned. If it does not exist, `Nil` is returned. When writing the property, if `Index` does not exist, a new item is added with the associated data pointer. If it existed, then the associated data pointer is overwritten with the new value.

See also: TFPHashTable.Find (34), TFPHashTable.Add (33)

3.9.17 TFPHashTable.HashTable

Synopsis: Hash table instance

Declaration: `Property HashTable : TFPObjectList`

Visibility: public

Access: Read

Description: `TFPHashTable` is the internal list object (TFPObjectList (37) used for the hash table. Each element in this table is again a TFPObjectList (37) instance or `Nil`.

3.9.18 TFPHashTable.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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.AVGChainLen (36), TFPHashTable.MaxChainLength (36), TFPHashTable.LoadFactor (36), TFPHashTable.Density (37), TFPHashTable.NumberOfCollisions (37)

3.9.19 TFPHashTable.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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.AVGChainLen (36), TFPHashTable.MaxChainLength (36), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37), TFPHashTable.NumberOfCollisions (37)

3.9.20 TFPHashTable.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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.LoadFactor (36), TFPHashTable.MaxChainLength (36), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37), TFPHashTable.NumberOfCollisions (37)

3.9.21 TFPHashTable.MaxChainLength

Synopsis: Maximum chain length

Declaration: `Property MaxChainLength : Int64`

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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.LoadFactor (36), TFPHashTable.AvgChainLength (31), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37), TFPHashTable.NumberOfCollisions (37)

3.9.22 TFPHashTable.NumberOfCollisions

Synopsis: Number of extra items

Declaration: `Property NumberOfCollisions : Int64`

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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.LoadFactor (36), TFPHashTable.AvgChainLength (31), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37)

3.9.23 TFPHashTable.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: TFPHashTable.IsEmpty (34), TFPHashTable.Count (35), TFPHashTable.LoadFactor (36), TFPHashTable.AvgChainLength (31), TFPHashTable.VoidSlots (36), TFPHashTable.Density (37)

3.10 TFPObjectList

3.10.1 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 (46), `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).

3.10.2 Method overview

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

3.10.3 Property overview

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

3.10.4 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 (38), TFPObjectList.OwnsObjects (44), TObjectList (46)

3.10.5 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 (44) is `True`.

See also: TFPObjectList.OwnsObjects (44), TObjectList.Create (47)

3.10.6 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 (44) is `True`.

See also: TObjectList.Destroy (46)

3.10.7 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 (44) 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 (44), TFPObjectList.Delete (39)

3.10.8 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 (44) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (44) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TTFPObjectList.Remove (24), TFPObjectList.Extract (40), TFPObjectList.OwnsObjects (44), TTFPObjectList.Add (24), TTFPObjectList.Clear (24)

3.10.9 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 (24), TTFPObjectList.Delete (24)

3.10.10 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 (43), TFPObjectList.Clear (39), #rtl.classes.tfplist.expand (**??**)

3.10.11 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 (43), TFPObjectList.Clear (39), TFPObjectList.Remove (40), TFPObjectList.Delete (39)

3.10.12 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 (44) 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 (43), TFPObjectList.Clear (39), TFPObjectList.Delete (39), TFPObjectList.Extract (40)

3.10.13 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 (44), TFPObjectList.Remove (40), TFPObjectList.Extract (40)

3.10.14 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 (41)

3.10.15 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 (39), TFPObjectList.Delete (39)

3.10.16 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 (42), TFPObjectList.Pack (43)

3.10.17 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 (42), TFPObjectList.Pack (43)

3.10.18 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 (40), 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 (40), TFPObjectList.Delete (39), TFPObjectList.Insert (41)

3.10.19 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.

3.10.20 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 (42), TFPObjectList.Last (42)

3.10.21 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 (**??**)

3.10.22 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 (25), TObjectListCallback (25)

3.10.23 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 (44).

See also: TFPObjectList.Count (44)

3.10.24 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 (44)

3.10.25 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 (38), TFPObjectList.Delete (39), TFPObjectList.Remove (40), TFPObjectList.Clear (39)

3.10.26 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 (44)

3.10.27 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 (**??**)

3.11 THTNode

3.11.1 Description

`THTNode` is used by the TFPHashTable (31) class to store the keys and associated values.

3.11.2 Method overview

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

3.11.3 Property overview

Page	Property	Access	Description
46	Data	rw	Data associated with this hash value.
46	Key	r	Key value associated with this hash item.

3.11.4 THTNode.CreateWith

Synopsis: Create a new instance of `THTNode`

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

Visibility: public

Description: `CreateWith` creates a new instance of `THTNode` 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)

3.11.5 THTNode.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: THTNode.Key (46)

3.11.6 THTNode.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: THTNode.CreateWith (45)

3.11.7 THTNode.Data

Synopsis: Data associated with this hash value.

Declaration: `Property Data : pointer`

Visibility: public

Access: Read,Write

Description: `Data` is the (optional) data associated with this hash value. It will be set by the TFPHashTable.Add (33) method.

See also: TFPHashTable.Add (33)

3.12 TObjectList

3.12.1 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 (37), `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.

3.12.2 Method overview

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

3.12.3 Property overview

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

3.12.4 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 (46), TObjectList.OwnsObjects (50), TFPObjectList (37)

3.12.5 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 (49), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (48), TObjectList.Remove (48)

3.12.6 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 (48), `Extract` does not free the extracted element if OwnsObjects (50) 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 (48)

3.12.7 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 (50) 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 (48)

3.12.8 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 (48)

3.12.9 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 (48)

3.12.10 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 (47), TObjectList.Remove (48)

3.12.11 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 (49), TObjectList.Pack (46)

3.12.12 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 (49), TObjectList.Pack (46)

3.12.13 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 (47), TObjectList.Delete (46), TObjectList.Remove (48), TObjectList.Clear (46)

3.12.14 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 (**??**)

3.13 TObjectQueue

3.13.1 Method overview

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

3.13.2 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 (51), TObjectQueue.Peek (51)

3.13.3 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 (50), TObjectQueue.Peek (51)

3.13.4 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 (50), TObjectQueue.Pop (51)

3.14 TObjectStack

3.14.1 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.

3.14.2 Method overview

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

3.14.3 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 (52), TObjectStack.Peek (52)

3.14.4 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 (51), TObjectStack.Peek (52)

3.14.5 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 (51), TObjectStack.Pop (52)

3.15 TOrderedList

3.15.1 Description

`TOrderedList` provides the base class for TQueue (54) and TStack (55). 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.

3.15.2 Method overview

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

3.15.3 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 (53)

3.15.4 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 (52)

3.15.5 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 (53)

3.15.6 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 (53)

3.15.7 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 (54), TOrderedList.Peek (54)

3.15.8 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 (54), TOrderedList.Push (54)

3.15.9 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 (54), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (54), TOrderedList.Push (54)

3.16 TQueue

3.16.1 Description

`TQueue` is a descendent of TOrderedList (52) which implements Push (54) and Pop (54) 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 (52)

3.17 TStack

3.17.1 Description

`TStack` is a descendent of TOrderedList (52) which implements Push (54) and Pop (54) 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 (52)

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 (56) 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 (56) 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 (56) 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 (56) 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 (56) 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 (58) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (58), StartDebugServer (61)

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 (58) which sends the name and value of a boolean value as an informational message.

Errors: None.

See also: SendDebug (58), SendDateTime (58), SendInteger (59), SendPointer (60)

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 (58) 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 (58), SendBoolean (58), SendInteger (59), SendPointer (60)

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 (57).

See also: SendDebugEx (58), SendDebugFmt (59), SendDebugFmtEx (59)

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 (58) uses informational messages.

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

Errors: None.

See also: SendDebug (58), SendDebugFmt (59), SendDebugFmtEx (59)

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 (58). It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (58), SendDebugEx (58), SendDebugFmtEx (59), #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 (58) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (58), SendDebugEx (58), SendDebugFmt (59), #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 (58) 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 (58), SendBoolean (58), SendDateTime (58), SendPointer (60)

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 (60), 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 (58), SendMethodExit (60), SendSeparator (61)

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 (60), 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 (58), SendMethodEnter (60), SendSeparator (61)

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 (58) 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 (58), SendBoolean (58), SendDateTime (58), SendInteger (59)

4.4.12 SendSeparator

Synopsis: Send a separator message

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (58) 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 (58), SendMethodEnter (60), SendMethodExit (60)

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 (58) call will attempt to start the server by itself if it is not yet running.

Errors: On error, `False` is returned.

See also: SendDebug (58), InitDebugClient (57)

Chapter 5

Reference for unit 'gettext'

5.1 Used units

Table 5.1: Used units by unit 'gettext'

Name	Page
Classes	??
sysutils	??

5.2 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 (64)) 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.

5.3 Constants, types and variables

5.3.1 Constants

```
MOFileHeaderMagic = $950412de
```

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

5.3.2 Types

```
PLongWordArray = ^TLongWordArray
```

Pointer to a TLongWordArray (63) array.

```
PMOStringTable = ^TMOStringTable
```

Pointer to a TMOStringTable (63) array.

```
PPCharArray = ^TPCharArray
```

Pointer to a TPCharArray (63) array.

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

TLongWordArray is an array used to define the PLongWordArray (62) 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 (63) record at the start of the file.

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

TMOStringTable is an array type containing TMOStringInfo (63) 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 (63) 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.

5.4 Procedures and functions

5.4.1 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.

5.4.2 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 (64) 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 (62), TMOFile (64)

5.5 EMOFileError

5.5.1 Description

`EMOFileError` is raised in case an TMOFile (64) instance is created with an invalid .mo.

5.6 TMOFile

5.6.1 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.

5.6.2 Method overview

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

5.6.3 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 (63) structure, then an EMOFileError (64) exception is raised.

See also: TMOFile.Destroy (65)

5.6.4 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 (64)

5.6.5 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 6

Reference for unit 'idea'

6.1 Used units

Table 6.1: Used units by unit 'idea'

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

6.2 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.

6.3 Constants, types and variables

6.3.1 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

6.3.2 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 (68) or DeKeyIDEA (67) 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.

6.4 Procedures and functions

6.4.1 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 (68), DeKeyIdea (67), TIDEAEncryptStream (69), TIDEADecryptStream (68)

6.4.2 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 (68), CipherIdea (67)

6.4.3 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 (67), CipherIdea (67)

6.5 EIDEAError

6.5.1 Description

`EIDEAError` is used to signal errors in the IDEA encryption decryption streams.

6.6 TIDEADeCryptStream

6.6.1 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 (67) 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 (69) encryption stream.

6.6.2 Method overview

Page	Property	Description
68	Read	Reads data from the stream, decrypting it as needed
69	Seek	Set position on the stream
69	Write	Write data to the stream

6.6.3 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 (69), TIDEADecryptStream.Seek (69), TIDEAEncryptStream (69)

6.6.4 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 (68) exception, because the decryption stream is read-only. To write to an encryption stream, use the Write (70) method of the TIDEAEncryptStream (69) decryption stream.

Errors: An EIDEAError (68) exception is raised when calling this method.

See also: TIDEADecryptStream.Read (68), TIDEAEncryptStream (69), TIDEAEncryptStream.Write (70)

6.6.5 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 (68) exception is raised if the stream does not allow the requested seek operation.

See also: TIDEADeCryptStream.Read (68)

6.7 TIDEAEncryptStream

6.7.1 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 (67) 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 (68) decryption stream.

6.7.2 Method overview

Page	Property	Description
70	Destroy	Flush data buffers and free the stream instance.
71	Flush	Write remaining bytes from the stream
70	Read	Read data from the stream
71	Seek	Set stream position
70	Write	Write bytes to the stream to be encrypted

6.7.3 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 (72)

6.7.4 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 (68) exception, because the encryption stream is write-only. To read from an encrypted stream, use the Read (68) method of the TIDEADecryptStream (68) decryption stream.

Errors: An EIDEAError (68) exception is raised when calling this method.

See also: TIDEAEncryptStream.Write (70), TIDEADecryptStream (68), TIDEADecryptStream.Read (68)

6.7.5 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 (68)

6.7.6 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 (68) will be raised unless called with `0` and `soFromCurrent` as arguments.

See also: TIDEAEncryptStream.Write (70), EIDEAError (68)

6.7.7 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 (70)

6.8 TIDEAStream

6.8.1 Description

Do not create instances of `TIDEAStream` directly. It implements no useful functionality: it serves as a common ancestor of the TIDEAEncryptStream (69) and TIDEADeCryptStream (68), 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.

6.8.2 Method overview

Page	Property	Description
72	Create	Creates a new instance of the `TIDEAStream` class

6.8.3 Property overview

Page	Property	Access	Description
72	Key	r	Key used when encrypting/decrypting

6.8.4 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 (69), TIDEADeCryptStream (68)

6.8.5 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 (67)

Chapter 7

Reference for unit 'iostream'

7.1 Used units

Table 7.1: Used units by unit 'iostream'

Name	Page
Classes	**??**

7.2 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`).

7.3 Constants, types and variables

7.3.1 Types

```
TIOSType = (iosInput,iosOutPut,iosError)
```

Table 7.2: 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 (74) constructor of TIOStream (74), it determines what kind of stream is created.

7.4 EIOStreamError

7.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (74).

7.5 TIOStream

7.5.1 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 (73) 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 (75) behaviour based on this position.

7.5.2 Method overview

Page	Property	Description
74	Create	Construct a new instance of TIOStream (74)
74	Read	Read data from the stream.
75	Seek	Set the stream position
75	SetSize	Set the size of the stream
75	Write	Write data to the stream

7.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (74)

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (74), 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 (74), TIOStream.Write (75)

7.5.4 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 (74) 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 (73), TIOStream.Write (75)

7.5.5 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 (74) 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 (73), TIOStream.Read (74)

7.5.6 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 (74)

7.5.7 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 (74) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (74)

Chapter 8

Reference for unit 'Pipes'

8.1 Used units

Table 8.1: Used units by unit 'Pipes'

Name	Page
Classes	??
sysutils	??

8.2 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.

8.3 Constants, types and variables

8.3.1 Constants

```
ENoReadMSg = 'Cannot read from OuputPipeStream.'
```

Constant used in ENoReadPipe (77) exception.

```
ENoSeekMsg = 'Cannot seek on pipes'
```

Constant used in EPipeSeek (78) exception.

```
ENoWriteMsg = 'Cannot write to InputPipeStream.'
```

Constant used in ENoWritePipe (77) exception.

```
EPipeMsg = 'Failed to create pipe.'
```

Constant used in EPipeCreation (77) exception.

8.4 Procedures and functions

8.4.1 CreatePipeHandles

Synopsis: Function to create a set of pipe handles

Declaration:
```
function CreatePipeHandles(var Inhandle: LongInt;var OutHandle: LongInt)
                                  : 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 (77)

8.4.2 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 (77), 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 (77) exception is raised.

See also: CreatePipeHandles (77), TInputPipeStream (78), TOutputPipeStream (79)

8.5 ENoReadPipe

8.5.1 Description

Exception raised when a write operation is attempted on a write-only pipe.

8.6 ENoWritePipe

8.6.1 Description

Exception raised when a read operation is attempted on a read-only pipe.

8.7 EPipeCreation

8.7.1 Description

Exception raised when an error occurred during the creation of a pipe pair.

8.8 EPipeError

8.8.1 Description

Exception raised when an invalid operation is performed on a pipe stream.

8.9 EPipeSeek

8.9.1 Description

Exception raised when an invalid seek operation is attempted on a pipe.

8.10 TInputPipeStream

8.10.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (77) 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.

8.10.2 Method overview

Page	Property	Description
79	Read	Read data from the stream to a buffer.
78	Seek	Set the current position of the stream
78	Write	Write data to the stream.

8.10.3 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 (77) exception is raised when this function is called.

See also: TInputPipeStream.Read (79), TInputPipeStream.Seek (78)

8.10.4 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 (78) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (78), #rtl.classes.tstream.seek (**??**)

8.10.5 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 (78), TInputPipeStream.Seek (78)

8.11 TOutputPipeStream

8.11.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (77) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

8.11.2 Method overview

Page	Property	Description
80	Read	Read data from the stream.
79	Seek	Sets the position in the stream

8.11.3 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 (78) exception is raised if this method is called.

8.11.4 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 (77) exception is raised when this function is called.

See also: TOutputPipeStream.Seek (79)

Chapter 9

Reference for unit 'process'

9.1 Used units

Table 9.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	76
sysutils	??

9.2 Overview

The Process unit contains the code for the TProcess (83) 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.

9.3 Constants, types and variables

9.3.1 Types

```
TProcessOption = (poRunSuspended,poWaitOnExit,poUsePipes,
                  poStderrToOutPut,poNoConsole,poNewConsole,
                  poDefaultErrorMode,poNewProcessGroup,poDebugProcess,
                  poDebugOnlyThisProcess)
```

When a new process is started using TProcess.Execute (85), 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,
                         poNewProcessGroup,poNoConsole,poRunSuspended,
```

Table 9.2: 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.

```
                                poStderrToOutPut,poUsePipes,poWaitOnExit)
```

Set of TProcessOption (81).

```
TProcessPriority = (ppHigh,ppIdle,ppNormal,ppRealTime)
```

Table 9.3: 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.

```
TStartupOptions= Set of (suoUseCountChars,suoUseFillAttribute,
                         suoUsePosition,suoUseShowWindow,suoUseSize)
```

Set of TStartUpOption (82).

Table 9.4: 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 9.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (83).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (83).
suoUsePosition	Use the window sizes as specified in TProcess (83).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (82)
suoUseSize	Use the window sizes as specified in TProcess (83)

9.4 EProcess

9.4.1 Description

Exception raised when an error occurs in a TProcess routine.

9.5 TProcess

9.5.1 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 (91) property to the full pathname of the program that should be executed, and call Execute (85). To determine whether the process is still running (i.e. has not stopped executing), the Running (95) property can be checked.

More advanced techniques can be used with the Options (93) settings.

9.5.2 Method overview

Page	Property	Description
86	CloseInput	Close the input stream of the process
86	CloseOutput	Close the output stream of the process
86	CloseStderr	Close the error stream of the process
85	Create	Create a new instance of the `TProcess` class.
85	Destroy	Destroy this instance of `TProcess`
85	Execute	Execute the program with the given options
86	Resume	Resume execution of a suspended process
87	Suspend	Suspend a running process
87	Terminate	Terminate a running process
87	WaitOnExit	Wait for the program to stop executing.

9.5.3 Property overview

Page	Property	Access	Description
91	Active	rw	Start or stop the process.
91	ApplicationName	rw	Name of the application to start
91	CommandLine	rw	Command-line to execute
92	ConsoleTitle	rw	Title of the console window
92	CurrentDirectory	rw	Working directory of the process.
92	Desktop	rw	Desktop on which to start the process.
93	Environment	rw	Environment variables for the new process
90	ExitStatus	r	Exit status of the process.
97	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
88	Handle	r	Handle of the process
90	InheritHandles	rw	Should the created process inherit the open handles of the current process.
89	Input	r	Stream connected to standard input of the process.
93	Options	rw	Options to be used when starting the process.
89	Output	r	Stream connected to standard output of the process.
93	Priority	rw	Priority at which the process is running.
88	ProcessHandle	r	Alias for Handle (88)
88	ProcessID	r	ID of the process.
95	Running	r	Determines wheter the process is still running.
95	ShowWindow	rw	Determines how the process main window is shown (Windows only)
94	StartupOptions	rw	Additional (Windows) startup options
90	Stderr	r	Stream connected to standard diagnostic output of the process.
88	ThreadHandle	r	Main process thread handle
89	ThreadID	r	ID of the main process thread
95	WindowColumns	rw	Number of columns in console window (windows only)
96	WindowHeight	rw	Height of the process main window
96	WindowLeft	rw	X-coordinate of the initial window (Windows only)
87	WindowRect	rw	Positions for the main program window.
96	WindowRows	rw	Number of rows in console window (Windows only)
96	WindowTop	rw	Y-coordinate of the initial window (Windows only)
97	WindowWidth	rw	Height of the process main window (Windows only)

9.5.4 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.

9.5.5 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 (85)

9.5.6 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 (91), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (93), 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 (87) call can be used to wait for it to close, or the Running (95) call can be used to check whether it is still running.

The TProcess.Terminate (87) call can be used to terminate the program if it is still running, or the Suspend (87) call can be used to temporarily stop the program's execution.

The ExitStatus (90) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (83) exception is raised.

See also: TProcess.Running (95), TProcess.WaitOnExit (87), TProcess.Terminate (87), TProcess.Suspend (87), TProcess.Resume (86), TProcess.ExitStatus (90)

9.5.7 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 (89), TProcess.StdErr (90), TProcess.Output (89), TProcess.CloseOutput (86), TProcess.CloseStdErr (86)

9.5.8 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 (89), TProcess.Input (89), TProcess.StdErr (90), TProcess.CloseInput (86), TProcess.CloseStdErr (86)

9.5.9 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 (89), TProcess.Input (89), TProcess.StdErr (90), TProcess.CloseInput (86), TProcess.CloseStdErr (86)

9.5.10 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 (93).

Errors: None.

See also: TProcess.Suspend (87), TProcess.Options (93), TProcess.Execute (85), TProcess.Terminate (87)

9.5.11 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 (86) call.

`Suspend` is fundamentally different from TProcess.Terminate (87) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (93), TProcess.Resume (86), TProcess.Terminate (87), TProcess.Execute (85)

9.5.12 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 (90), TProcess.Suspend (87), TProcess.Execute (85), TProcess.WaitOnExit (87)

9.5.13 TProcess.WaitOnExit

Synopsis: Wait for the program to stop executing.

Declaration: `function WaitOnExit : DWord`

Visibility: public

Description: `WaitOnExit` waits for the running program to exit and then returns the exit status of the program.

Errors: On error, -1 is returned. Other values are system dependent.

See also: TProcess.ExitStatus (90), TProcess.Terminate (87), TProcess.Running (95)

9.5.14 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

9.5.15 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 (85) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (88), TProcess.ProcessID (88), TProcess.ThreadID (89)

9.5.16 TProcess.ProcessHandle

Synopsis: Alias for Handle (88)

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (88) and is provided for completeness only.

See also: TProcess.Handle (88), TProcess.ThreadHandle (88), TProcess.ProcessID (88), TProcess.ThreadID (89)

9.5.17 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 (85) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (88), TProcess.ProcessID (88), TProcess.ThreadID (89)

9.5.18 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 (85) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (88), TProcess.ThreadHandle (88), TProcess.ThreadID (89)

9.5.19 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 (85) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (88), TProcess.Handle (88), TProcess.ThreadHandle (88)

9.5.20 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 (93).

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 (89), TProcess.StdErr (90), TProcess.Options (93), TProcessOption (81)

9.5.21 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 (93).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (93).

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 (89), TProcess.StdErr (90), TProcess.Options (93), TProcessOption (81)

9.5.22 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 (93).

The `Output` stream equals the Output (89) when the `poStdErrToOutPut` flag is used in Options (93).

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 (89), TProcess.Output (89), TProcess.Options (93), TProcessOption (81)

9.5.23 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 (95), TProcess.Terminate (87)

9.5.24 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 (89), TProcess.Output (89), TProcess.StdErr (90)

9.5.25 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 (85), TProcess.Terminate (87)

9.5.26 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 (91). 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 (91)

9.5.27 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 (91)

9.5.28 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 (95), TProcess.WindowRows (96)

9.5.29 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 (93)

9.5.30 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 (89), TProcess.Output (89), TProcess.StdErr (90)

9.5.31 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 (93)

9.5.32 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 (85) call is made.

Table 9.6:

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 (81), TProcessOptions (82), TProcess.Priority (93), TProcess.StartUpOptions (94)

9.5.33 TProcess.Priority

Synopsis: Priority at which the process is running.

Declaration: `Property Priority : TProcessPriority`

Visibility: published

Access: Read,Write

Table 9.7:

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.

Description: `Priority` determines the priority at which the process is running.

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 9.8:

Priority	Nice value
`ppHigh`	20
`ppIdle`	20
`ppNormal`	0
`ppRealTime`	-20

See also: TProcessPriority (82)

9.5.34 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.

Table 9.9:

Priority	Meaning
`suoUseShowWindow`	Use the Show Window options specified in ShowWindow (95)
`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 (97).

See also: TProcess.ShowWindow (95), TProcess.WindowHeight (96), TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.FillAttribute (97)

9.5.35 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 (87), TProcess.Active (91), TProcess.ExitStatus (90)

9.5.36 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 9.10:

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

9.5.37 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 (94)

See also: TProcess.WindowHeight (96), TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowRows (96), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.38 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 (94)

See also: TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.39 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 (94)

See also: TProcess.WindowHeight (96), TProcess.WindowWidth (97), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.40 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 (94)

See also: TProcess.WindowHeight (96), TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.41 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 (94)

See also: TProcess.WindowHeight (96), TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.42 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 (94)

See also: TProcess.WindowHeight (96), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.FillAttribute (97), TProcess.StartupOptions (94)

9.5.43 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 (96), TProcess.WindowWidth (97), TProcess.WindowLeft (96), TProcess.WindowTop (96), TProcess.WindowColumns (95), TProcess.WindowRows (96), TProcess.StartupOptions (94)

Chapter 10

Reference for unit 'StreamIO'

10.1 Used units

Table 10.1: Used units by unit 'StreamIO'

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

10.2 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.

10.3 Procedures and functions

10.3.1 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 (99)

10.3.2 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 (98).

Errors: An invalid class reference will be returned if the file was not associated with a stream.

See also: AssignStream (98), #rtl.classes.TStream (**??**)

Chapter 11

Reference for unit 'zstream'

11.1 Used units

Table 11.1: Used units by unit 'zstream'

Name	Page
Classes	**??**
paszlib	100
sysutils	**??**
zbase	100

11.2 Overview

The ZStream unit implements a TStream (**??**) descendent (TCompressionStream (101)) 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 (104)) and decompresses it with the inflate algorithm.

11.3 Constants, types and variables

11.3.1 Types

```
TCompressionLevel = (clNone,clFastest,clDefault,clMax)
```

Compression level for the deflate algorithm

```
TGZOpenMode = (gzOpenRead,gzOpenWrite)
```

Open mode for gzip file.

Table 11.2: 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 11.3: Enumeration values for type TGZOpenMode

Value	Explanation
gzOpenRead	Open file for reading
gzOpenWrite	Open file for writing

11.4 ECompressionError

11.4.1 Description

`ECompressionError` is the exception class used by the TCompressionStream (101) class.

11.5 EDecompressionError

11.5.1 Description

`EDecompressionError` is the exception class used by the TDeCompressionStream (104) class.

11.6 EZlibError

11.6.1 Description

Errors which occur in the zstream unit are signaled by raising an `EZLibError` exception descendent.

11.7 TCompressionStream

11.7.1 Description

`TCompressionStream`

11.7.2 Method overview

Page	Property	Description
102	Create	Create a new instance of the compression stream.
102	Destroy	Flushe data to the output stream and destroys the compression stream.
102	Read	Overridden to raise an exception.
103	Seek	Overrides seek to raise an exception.
103	Write	Write data to the stream

11.7.3 Property overview

Page	Property	Access	Description
103	CompressionRate	r	Running compression rate of compression stream
103	OnProgress		Progress handler

11.7.4 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 (102)

11.7.5 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 (102)

11.7.6 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 (101) exception is raised.

See also: ECompressionError (101), TCompressionStream.Write (103)

11.7.7 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 (101) exception is raised.

See also: TCompressionStream.Read (102), TCompressionStream.Seek (103)

11.7.8 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 (101) exception is raised.

See also: ECompressionError (101), TCompressionStream.Read (102), TCompressionStream.Write (103)

11.7.9 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 (103)

11.7.10 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.

11.8 TCustomZlibStream

11.8.1 Description

`TCustomZlibStream` serves as the ancestor class for the TCompressionStream (101) and TDecompressionStream (104) classes.

It introduces support for a progess handler, and stores the input or output stream.

11.8.2 Method overview

Page	Property	Description
104	Create	Create a new instance of `TCustomZlibStream`

11.8.3 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 (101), TDecompressionStream (104)

11.9 TDecompressionStream

11.9.1 Description

`TDecompressionStream` performs the inverse operation of TCompressionStream (101). 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 (101) compression stream.

11.9.2 Method overview

Page	Property	Description
105	Create	Creates a new instance of the `TDecompressionStream` stream
105	Destroy	Destroys the `TDecompressionStream` instance
105	Read	Read data from the compressed stream
106	Seek	Move stream position to a certain location in the stream.
105	Write	Write data to the stream

11.9.3 Property overview

Page	Property	Access	Description
106	OnProgress		Progress handler

11.9.4 TDecompressionStream.Create

Synopsis: Creates a new instance of the `TDecompressionStream` stream

Declaration: `constructor Create(ASource: TStream)`

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.

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 (105)

11.9.5 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 (105)

11.9.6 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 (101) exception is raised.

See also: TCompressionStream.Write (103)

11.9.7 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 (101) exception, because the `TDecompressionStream` class is read-only.

Errors: An EDeCompressionError (101) exception is always raised.

See also: TDeCompressionStream.Read (105), EDeCompressionError (101)

11.9.8 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 (101) exception is raised if the stream does not allow the requested seek operation.

See also: TDecompressionStream.Read (105)

11.9.9 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.

11.10 TGZFileStream

11.10.1 Description

`TGZFileStream` can be used to read data from a gzip file, or to write data to a gzip file.

11.10.2 Method overview

Page	Property	Description
107	Create	Create a new instance of `TGZFileStream`
107	Destroy	Removes `TGZFileStream` instance
107	Read	Read data from the compressed file
108	Seek	Set the position in the compressed stream.
108	Write	Write data to be compressed

11.10.3 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 (107) 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 (108) method will be stored in the file in compressed (deflated) form.

Errors: If the file is not found, an EZlibError (101) exception is raised.

See also: TGZFileStream.Destroy (107), TGZOpenMode (100)

11.10.4 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 (107)

11.10.5 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 (107), TGZFileStream.Write (108), TGZFileStream.Seek (108)

11.10.6 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 (101) exception is raised.

See also: TGZFileStream.Create (107), TGZFileStream.Read (107), TGZFileStream.Seek (108)

11.10.7 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 (106) for a list of possibilities.

Errors: In case an impossible combination is asked, an EZlibError (101) exception is raised.

See also: TDecompressionStream.Seek (106)