Free Component Library (FCL): Reference guide.

Reference guide for FCL units.
Document version 2.1
November 2005

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 'contnrs'

1.1 Used units

Table 1.1: Used units by unit 'contnrs'

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

1.2 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

1.3 Constants, types and variables

1.3.1 Types

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

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

1.4 Procedures and functions

1.4.1 RSHash

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

Visibility: default

1.5 EDuplicate

1.6 EKeyNotFound

1.7 TClassList

1.7.1 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.

1.7.2 Method overview

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

1.7.3 Property overview

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

1.7.4 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 (11), #rtl.classes.tlist.add (**??**)

1.7.5 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 (11), #rtl.classes.Tlist.Extract (**??**)

1.7.6 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 (11), #rtl.classes.Tlist.Remove (**??**)

1.7.7 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 (**??**)

1.7.8 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 (12), TClassList.Pack (10)

1.7.9 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 (12), TClassList.Pack (10)

1.7.10 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 (10), TClassList.Remove (11)

1.7.11 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 (**??**)

1.8 TComponentList

1.8.1 Description

`TComponentList` is a TObjectList (27) 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 (27), `TComponentList` removes any `TComponent` from the list if the `TComponent` instance was freed externally. It uses the `FreeNotification` mechanism for this.

1.8.2 Method overview

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

1.8.3 Property overview

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

1.8.4 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 (27), #rtl.classes.TComponent (**??**)

1.8.5 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 (9)

1.8.6 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 (27) 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 (14), TObjectList.Extract (28)

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

Errors: None.

See also: TComponentList.Extract (14), TObjectList.Remove (28)

1.8.8 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 (28)

1.8.9 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 (15), TObjectList.First (29)

1.8.10 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 (15), TObjectList.Last (29)

1.8.11 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 (29), TComponentList.Add (13), TComponentList.Remove (14)

1.8.12 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 (30)

1.9 TFPHashTable

1.9.1 Method overview

Page	Property	Description
17	Add	
16	ChangeTableSize	
17	Clear	
16	Create	
16	CreateWith	
17	Delete	
16	Destroy	
17	Find	
17	IsEmpty	

1.9.2 Property overview

Page	Property	Access	Description
18	AVGChainLen	r	
17	Count	r	
19	Density	r	
17	HashFunction	rw	
18	HashTable	r	
17	HashTableSize	rw	
18	Items	rw	
18	LoadFactor	r	
18	MaxChainLength	r	
18	NumberOfCollisions	r	
18	VoidSlots	r	

1.9.3 TFPHashTable.Create

Declaration: `constructor Create`

Visibility: public

1.9.4 TFPHashTable.CreateWith

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

Visibility: public

1.9.5 TFPHashTable.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

1.9.6 TFPHashTable.ChangeTableSize

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

Visibility: public

1.9.7 TFPHashTable.Clear

Declaration: `procedure Clear;  Virtual`

Visibility: public

1.9.8 TFPHashTable.Add

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

Visibility: public

1.9.9 TFPHashTable.Delete

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

Visibility: public

1.9.10 TFPHashTable.Find

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

Visibility: public

1.9.11 TFPHashTable.IsEmpty

Declaration: `function IsEmpty : Boolean`

Visibility: public

1.9.12 TFPHashTable.HashFunction

Declaration: `Property HashFunction : THashFunction`

Visibility: public

Access: Read,Write

1.9.13 TFPHashTable.Count

Declaration: `Property Count : Int64`

Visibility: public

Access: Read

1.9.14 TFPHashTable.HashTableSize

Declaration: `Property HashTableSize : LongWord`

Visibility: public

Access: Read,Write

1.9.15 TFPHashTable.Items

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

Visibility: public

Access: Read,Write

1.9.16 TFPHashTable.HashTable

Declaration: `Property HashTable : TFPObjectList`

Visibility: public

Access: Read

1.9.17 TFPHashTable.VoidSlots

Declaration: `Property VoidSlots : LongWord`

Visibility: public

Access: Read

1.9.18 TFPHashTable.LoadFactor

Declaration: `Property LoadFactor : double`

Visibility: public

Access: Read

1.9.19 TFPHashTable.AVGChainLen

Declaration: `Property AVGChainLen : double`

Visibility: public

Access: Read

1.9.20 TFPHashTable.MaxChainLength

Declaration: `Property MaxChainLength : Int64`

Visibility: public

Access: Read

1.9.21 TFPHashTable.NumberOfCollisions

Declaration: `Property NumberOfCollisions : Int64`

Visibility: public

Access: Read

1.9.22 TFPHashTable.Density

Declaration: `Property Density : LongWord`

Visibility: public

Access: Read

1.10 TFPObjectList

1.10.1 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 (27), `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).

1.10.2 Method overview

Page	Property	Description
20	Add	Add an object to the list.
24	Assign	Copy the contents of a list.
20	Clear	Clear all elements in the list.
19	Create	Create a new object list
21	Delete	Delete an element from the list.
20	Destroy	Clears the list and destroys the list instance
21	Exchange	Exchange the location of two objects
21	Expand	Expand the capacity of the list.
21	Extract	Extract an object from the list
22	FindInstanceOf	Search for an instance of a certain class
23	First	Return the first non-nil object in the list
22	IndexOf	Search for an object in the list
23	Insert	Insert a new object in the list
23	Last	Return the last non-nil object in the list.
23	Move	Move an object to another location in the list.
24	Pack	Remove all `Nil` references from the list
22	Remove	Remove an item from the list.
24	Sort	Sort the list of objects

1.10.3 Property overview

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

1.10.4 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 (20), TFPObjectList.OwnsObjects (25), TObjectList (27)

1.10.5 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 (25) is `True`.

See also: TFPObjectList.OwnsObjects (25), TObjectList.Create (27)

1.10.6 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 (25) is `True`.

See also: TObjectList.Destroy (27)

1.10.7 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 (25) 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 (25), TFPObjectList.Delete (21)

1.10.8 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 (25) is `True`, the object is also freed.

Errors: An access violation may occur when OwnsObjects (25) is `True` and either the object was freed externally, or when the same object is in the same list twice.

See also: TTFPObjectList.Remove (9), TFPObjectList.Extract (21), TFPObjectList.OwnsObjects (25), TTFPObjectList.Add (9), TTFPObjectList.Clear (9)

1.10.9 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 (9), TTFPObjectList.Delete (9)

1.10.10 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 (24), TFPObjectList.Clear (20), #rtl.classes.tfplist.expand (**??**)

1.10.11 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 (24), TFPObjectList.Clear (20), TFPObjectList.Remove (22), TFPObjectList.Delete (21)

1.10.12 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 (25) 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 (24), TFPObjectList.Clear (20), TFPObjectList.Delete (21), TFPObjectList.Extract (21)

1.10.13 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 (25), TFPObjectList.Remove (22), TFPObjectList.Extract (21)

1.10.14 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 (22)

1.10.15 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 (20), TFPObjectList.Delete (21)

1.10.16 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 (23), TFPObjectList.Pack (24)

1.10.17 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 (23), TFPObjectList.Pack (24)

1.10.18 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 (21), 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 (21), TFPObjectList.Delete (21), TFPObjectList.Insert (23)

1.10.19 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.

1.10.20 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 (23), TFPObjectList.Last (23)

1.10.21 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 (**??**)

1.10.22 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 (25).

See also: TFPObjectList.Count (25)

1.10.23 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 (25)

1.10.24 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 (19), TFPObjectList.Delete (21), TFPObjectList.Remove (22), TFPObjectList.Clear (20)

1.10.25 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 (25)

1.10.26 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 (**??**)

1.11 THTNode

1.11.1 Method overview

Page	Property	Description
26	CreateWith	
26	HasKey	

1.11.2 Property overview

Page	Property	Access	Description
26	Data	rw	
26	Key	r	

1.11.3 THTNode.CreateWith

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

Visibility: public

1.11.4 THTNode.HasKey

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

Visibility: public

1.11.5 THTNode.Key

Declaration: `Property Key : String`

Visibility: public

Access: Read

1.11.6 THTNode.Data

Declaration: `Property Data : pointer`

Visibility: public

Access: Read,Write

1.12 TObjectList

1.12.1 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 (19), `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.

1.12.2 Method overview

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

1.12.3 Property overview

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

1.12.4 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 (27), TObjectList.OwnsObjects (30), TFPObjectList (19)

1.12.5 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 (29), #rtl.classes.TList.Delete (**??**), TObjectList.Extract (28), TObjectList.Remove (28)

1.12.6 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 (28), `Extract` does not free the extracted element if OwnsObjects (30) 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 (28)

1.12.7 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 (30) 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 (28)

1.12.8 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 (29)

1.12.9 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 (28)

1.12.10 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 (27), TObjectList.Remove (28)

1.12.11 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 (29), TObjectList.Pack (27)

1.12.12 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 (29), TObjectList.Pack (27)

1.12.13 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 (27), TObjectList.Delete (27), TObjectList.Remove (28), TObjectList.Clear (27)

1.12.14 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 (**??**)

1.13 TObjectQueue

1.13.1 Method overview

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

1.13.2 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 (31), TObjectQueue.Peek (31)

1.13.3 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 (30), TObjectQueue.Peek (31)

1.13.4 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 (30), TObjectQueue.Pop (31)

1.14 TObjectStack

1.14.1 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.

1.14.2 Method overview

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

1.14.3 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 (32), TObjectStack.Peek (32)

1.14.4 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 (31), TObjectStack.Peek (32)

1.14.5 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 (31), TObjectStack.Pop (32)

1.15 TOrderedList

1.15.1 Description

`TOrderedList` provides the base class for TQueue (34) and TStack (35). 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.

1.15.2 Method overview

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

1.15.3 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 (33)

1.15.4 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 (32)

1.15.5 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 (33)

1.15.6 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 (33)

1.15.7 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 (34), TOrderedList.Peek (34)

1.15.8 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 (34), TOrderedList.Push (34)

1.15.9 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 (34), without actually popping it from the list.

Errors: None.

See also: TOrderedList.Pop (34), TOrderedList.Push (34)

1.16 TQueue

1.16.1 Description

`TQueue` is a descendent of TOrderedList (32) which implements Push (34) and Pop (34) 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 (32)

1.17 TStack

1.17.1 Description

`TStack` is a descendent of TOrderedList (32) which implements Push (34) and Pop (34) 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 (32)

Chapter 2

Reference for unit 'dbugintf'

2.1 Writing a debug server

Writing a debug server is relatively easy. It should instantiate a `TSimpleIPCServer` class from the SimpleIPC (36) 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 (36) 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 (36) unit should also be functional.

2.2 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 (36) 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 (36) unit should also be functional.

2.3 Constants, types and variables

2.3.1 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.

2.3.2 Constants

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

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

2.3.3 Types

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

Table 2.1: 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.

2.4 Procedures and functions

2.4.1 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 (38) call will initialize the debug client when it is first called.

Errors: None.

See also: SendDebug (38), StartDebugServer (41)

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

Errors: None.

See also: SendDebug (38), SendDateTime (38), SendInteger (39), SendPointer (40)

2.4.3 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 (38) 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 (38), SendBoolean (38), SendInteger (39), SendPointer (40)

2.4.4 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 (37).

See also: SendDebugEx (38), SendDebugFmt (39), SendDebugFmtEx (39)

2.4.5 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 (38) uses informational messages.

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

Errors: None.

See also: SendDebug (38), SendDebugFmt (39), SendDebugFmtEx (39)

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

Errors: None.

See also: SendDebug (38), SendDebugEx (38), SendDebugFmtEx (39), #rtl.sysutils.format (**??**)

2.4.7 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 (38) with Debug level `MType`. It exists mainly to avoid the `Format` call in calling code.

Errors: None.

See also: SendDebug (38), SendDebugEx (38), SendDebugFmt (39), #rtl.sysutils.format (**??**)

2.4.8 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 (38) 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 (38), SendBoolean (38), SendDateTime (38), SendPointer (40)

2.4.9 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 (40), 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 (38), SendMethodExit (40), SendSeparator (41)

2.4.10 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 (40), 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 (38), SendMethodEnter (40), SendSeparator (41)

2.4.11 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 (38) 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 (38), SendBoolean (38), SendDateTime (38), SendInteger (39)

2.4.12 SendSeparator

Synopsis: Send a separator message

Declaration: `procedure SendSeparator`

Visibility: default

Description: `SendSeparator` is a simple wrapper around SendDebug (38) 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 (38), SendMethodEnter (40), SendMethodExit (40)

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

Errors: On error, `False` is returned.

See also: SendDebug (38), InitDebugClient (37)

Chapter 3

Reference for unit 'iostream'

3.1 Used units

Table 3.1: Used units by unit 'iostream'

Name	Page
Classes	**??**

3.2 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`).

3.3 Constants, types and variables

3.3.1 Types

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

Table 3.2: 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 (43) constructor of TIOStream (43), it determines what kind of stream is created.

3.4 EIOStreamError

3.4.1 Description

Error thrown in case of an invalid operation on a TIOStream (43).

3.5 TIOStream

3.5.1 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 (42) 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 (44) behaviour based on this position.

3.5.2 Method overview

Page	Property	Description
43	Create	Construct a new instance of TIOStream (43)
43	Read	Read data from the stream.
44	Seek	Set the stream position
44	SetSize	Set the size of the stream
44	Write	Write data to the stream

3.5.3 TIOStream.Create

Synopsis: Construct a new instance of TIOStream (43)

Declaration: `constructor Create(aIOSType: TIOSType)`

Visibility: public

Description: `Create` creates a new instance of TIOStream (43), 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 (43), TIOStream.Write (44)

3.5.4 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 (43) 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 (42), TIOStream.Write (44)

3.5.5 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 (43) 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 (42), TIOStream.Read (43)

3.5.6 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 (43)

3.5.7 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 (43) exception is raised if the stream does not allow the requested seek operation.

See also: EIOStreamError (43)

Chapter 4

Reference for unit 'Pipes'

4.1 Used units

Table 4.1: Used units by unit 'Pipes'

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

4.2 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.

4.3 Constants, types and variables

4.3.1 Constants

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

Constant used in ENoReadPipe (46) exception.

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

Constant used in EPipeSeek (47) exception.

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

Constant used in ENoWritePipe (46) exception.

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

Constant used in EPipeCreation (46) exception.

4.4 Procedures and functions

4.4.1 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 (46)

4.4.2 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 (46), 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 (46) exception is raised.

See also: CreatePipeHandles (46), TInputPipeStream (47), TOutputPipeStream (48)

4.5 ENoReadPipe

4.5.1 Description

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

4.6 ENoWritePipe

4.6.1 Description

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

4.7 EPipeCreation

4.7.1 Description

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

4.8 EPipeError

4.8.1 Description

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

4.9 EPipeSeek

4.9.1 Description

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

4.10 TInputPipeStream

4.10.1 Description

`TInputPipeStream` is created by the CreatePipeStreams (46) 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.

4.10.2 Method overview

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

4.10.3 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 (46) exception is raised when this function is called.

See also: TInputPipeStream.Read (48), TInputPipeStream.Seek (47)

4.10.4 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 (47) exception is raised if the stream does not allow the requested seek operation.

See also: EPipeSeek (47), #rtl.classes.tstream.seek (**??**)

4.10.5 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 (47), TInputPipeStream.Seek (47)

4.11 TOutputPipeStream

4.11.1 Description

`TOutputPipeStream` is created by the CreatePipeStreams (46) call to represent the writing end of a pipe. It is a TStream (**??**) descendent which does not allow reading.

4.11.2 Method overview

Page	Property	Description
49	Read	Read data from the stream.
48	Seek	Sets the position in the stream

4.11.3 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 (47) exception is raised if this method is called.

4.11.4 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 (46) exception is raised when this function is called.

See also: TOutputPipeStream.Seek (48)

Chapter 5

Reference for unit 'process'

5.1 Used units

Table 5.1: Used units by unit 'process'

Name	Page
Classes	??
Pipes	45
sysutils	??

5.2 Overview

The Process unit contains the code for the TProcess (52) component, a cross-platform component to start and control other programs, offering also access to standard input and output for these programs.

5.3 Constants, types and variables

5.3.1 Types

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

When a new process is started using TProcess.Execute (54), 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,
                         poStderrToOutPut,poUsePipes,poWaitOnExit)
```

Set of TProcessOption (50).

Table 5.2: 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.

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

Table 5.3: 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 (51).

Table 5.4: 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 5.5: Enumeration values for type TStartupOption

Value	Explanation
suoUseCountChars	Use the console character width as specified in TProcess (52).
suoUseFillAttribute	Use the console fill attribute as specified in TProcess (52).
suoUsePosition	Use the window sizes as specified in TProcess (52).
suoUseShowWindow	Use the Show Window options specified in TShowWindowOption (51)
suoUseSize	Use the window sizes as specified in TProcess (52)

5.4 TProcess

5.4.1 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 (59) property to the full pathname of the program that should be executed, and call Execute (54). To determine whether the process is still running (i.e. has not stopped executing), the Running (63) property can be checked.

More advanced techniques can be used with the Options (61) settings.

5.4.2 Method overview

Page	Property	Description
53	Create	Create a new instance of the `TProcess` class.
53	Destroy	Destroy this instance of `TProcess`
54	Execute	Execute the program with the given options
54	Resume	Resume execution of a suspended process
55	Suspend	Suspend a running process
55	Terminate	Terminate a running process
55	WaitOnExit	Wait for the program to stop executing.

5.4.3 Property overview

Page	Property	Access	Description
59	Active	rw	Start or stop the process.
59	ApplicationName	rw	Name of the application to start
59	CommandLine	rw	Command-line to execute
60	ConsoleTitle	rw	Title of the console window
60	CurrentDirectory	rw	Working directory of the process.
60	DeskTop	rw	Desktop on which to start the process.
61	Environment	rw	Environment variables for the new process
58	ExitStatus	r	Exit status of the process.
65	FillAttribute	rw	Color attributes of the characters in the console window (Windows only)
56	Handle	r	Handle of the process
58	InheritHandles	rw	Should the created process inherit the open handles of the current process.
57	Input	r	Stream connected to standard input of the process.
61	Options	rw	Options to be used when starting the process.
57	OutPut	r	Stream connected to standard output of the process.
61	Priority	rw	Priority at which the process is running.
56	ProcessHandle	r	Alias for Handle (56)
56	ProcessID	r	ID of the process.
63	Running	r	Determines wheter the process is still running.
63	ShowWindow	rw	Determines how the process main window is shown (Windows only)
62	StartUpOptions	rw	Additional (Windows) startup options
58	StdErr	r	Stream connected to standard diagnostic output of the process.
56	ThreadHandle	r	Main process thread handle
57	ThreadID	r	ID of the main process thread
63	WindowColumns	rw	Number of columns in console window (windows only)
64	WindowHeight	rw	Height of the process main window
64	WindowLeft	rw	X-coordinate of the initial window (Windows only)
55	WindowRect	rw	Positions for the main program window.
64	WindowRows	rw	Number of rows in console window (Windows only)
64	WindowTop	rw	Y-coordinate of the initial window (Windows only)
65	WindowWidth	rw	Height of the process main window (Windows only)

5.4.4 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.

5.4.5 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 (53)

5.4.6 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 (59), applying as much as of the specified options as supported on the current platform.

If the `poWaitOnExit` option is specified in Options (61), 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 (55) call can be used to wait for it to close, or the Running (63) call can be used to check whether it is still running.

The TProcess.Terminate (55) call can be used to terminate the program if it is still running, or the Suspend (55) call can be used to temporarily stop the program's execution.

The ExitStatus (58) function can be used to check the program's exit status, after it has stopped executing.

Errors: On error a EProcess (50) exception is raised.

See also: TProcess.Running (63), TProcess.WaitOnExit (55), TProcess.Terminate (55), TProcess.Suspend (55), TProcess.Resume (54), TProcess.ExitStatus (58)

5.4.7 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 (61).

Errors: None.

See also: TProcess.Suspend (55), TProcess.Options (61), TProcess.Execute (54), TProcess.Terminate (55)

5.4.8 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 (54) call.

`Suspend` is fundamentally different from TProcess.Terminate (55) which actually stops the process.

Errors: On error, a nonzero result is returned.

See also: TProcess.Options (61), TProcess.Resume (54), TProcess.Terminate (55), TProcess.Execute (54)

5.4.9 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 (58), TProcess.Suspend (55), TProcess.Execute (54), TProcess.WaitOnExit (55)

5.4.10 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 (58), TProcess.Terminate (55), TProcess.Running (63)

5.4.11 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

5.4.12 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 (54) has been called. It is not reset after the process stopped.

See also: TProcess.ThreadHandle (56), TProcess.ProcessID (56), TProcess.ThreadID (57)

5.4.13 TProcess.ProcessHandle

Synopsis: Alias for Handle (56)

Declaration: `Property ProcessHandle : THandle`

Visibility: public

Access: Read

Description: `ProcessHandle` equals Handle (56) and is provided for completeness only.

See also: TProcess.Handle (56), TProcess.ThreadHandle (56), TProcess.ProcessID (56), TProcess.ThreadID (57)

5.4.14 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 (54) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (56), TProcess.ProcessID (56), TProcess.ThreadID (57)

5.4.15 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 (54) has been called. It is not reset after the process stopped.

See also: TProcess.Handle (56), TProcess.ThreadHandle (56), TProcess.ThreadID (57)

5.4.16 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 (54) has been called. It is not reset after the process stopped.

See also: TProcess.ProcessID (56), TProcess.Handle (56), TProcess.ThreadHandle (56)

5.4.17 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 (61).

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 (57), TProcess.StdErr (58), TProcess.Options (61), TProcessOption (50)

5.4.18 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 (61).

The `Output` stream also contains any data written to standard diagnostic output (`stderr`) when the `poStdErrToOutPut` flag is used in Options (61).

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 (57), TProcess.StdErr (58), TProcess.Options (61), TProcessOption (50)

5.4.19 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 (61).

The `Output` stream equals the Output (57) when the `poStdErrToOutPut` flag is used in Options (61).

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 (57), TProcess.Output (57), TProcess.Options (61), TProcessOption (50)

5.4.20 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 (63), TProcess.Terminate (55)

5.4.21 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 (57), TProcess.Output (57), TProcess.StdErr (58)

5.4.22 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 (54), TProcess.Terminate (55)

5.4.23 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 (59). 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 (59)

5.4.24 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 (59)

5.4.25 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 (63), TProcess.WindowRows (64)

5.4.26 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 (61)

5.4.27 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 (57), TProcess.Output (57), TProcess.StdErr (58)

5.4.28 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 (61)

5.4.29 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 (54) call is made.

Table 5.6:

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 (50), TProcessOptions (50), TProcess.Priority (61), TProcess.StartUpOptions (62)

5.4.30 TProcess.Priority

Synopsis: Priority at which the process is running.

Declaration: `Property Priority : TProcessPriority`

Visibility: published

Access: Read,Write

Table 5.7:

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 5.8:

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

See also: TProcessPriority (51)

5.4.31 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 5.9:

Priority	Meaning
suoUseShowWindow	Use the Show Window options specified in ShowWindow (63)
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 (65).

See also: TProcess.ShowWindow (63), TProcess.WindowHeight (64), TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.FillAttribute (65)

5.4.32 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 (55), TProcess.Active (59), TProcess.ExitStatus (58)

5.4.33 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 5.10:

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

5.4.34 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 (62)

See also: TProcess.WindowHeight (64), TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowRows (64), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.35 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 (62)

See also: TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.36 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 (62)

See also: TProcess.WindowHeight (64), TProcess.WindowWidth (65), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.37 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 (62)

See also: TProcess.WindowHeight (64), TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.38 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 (62)

See also: TProcess.WindowHeight (64), TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.39 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 (62)

See also: TProcess.WindowHeight (64), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.FillAttribute (65), TProcess.StartupOptions (62)

5.4.40 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 (64), TProcess.WindowWidth (65), TProcess.WindowLeft (64), TProcess.WindowTop (64), TProcess.WindowColumns (63), TProcess.WindowRows (64), TProcess.StartupOptions (62)

Chapter 6

Reference for unit 'StreamIO'

6.1 Used units

Table 6.1: Used units by unit 'StreamIO'

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

6.2 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.

6.3 Procedures and functions

6.3.1 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 (67)

6.3.2 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 (66).

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

See also: AssignStream (66), #rtl.classes.TStream (**??**)