Free Component Library (FCL) : Reference guide.

Reference guide for FCL units.
Document version 1.9
August 2004

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.

Chapter 1

Reference for unit 'Classes'

1.1 Used units

Table 1.1: Used units by unit 'Classes'

Name	Page
rtlconst	20
sysutils	20
types	20
typinfo	20

1.2 Overview

This documentation describes the FPC classes unit. The Classes unit contains basic classes for the Free Component Library (FCL):

- a TList (112) class for maintaining lists of pointers,
- TStringList (149) for lists of strings,
- TCollection (76) to manage collections of objects
- TStream (140) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

1.3 Constants, types and variables

Constants

```
BITSHIFT = 5
```

Used to calculate the size of a bits array

`FilerSignature : Array[1..4] of Char`

Constant that is found at the start of a binary stream containing a streamed component.

`fmCreate = \$FFFF`

TFileStream.Create (109) creates a new file if needed.

`fmOpenRead = 0`

TFileStream.Create (109) opens a file with read-only access.

`fmOpenReadWrite = 2`

TFileStream.Create (109) opens a file with read-write access.

`fmOpenWrite = 1`

TFileStream.Create (109) opens a file with write-only access.

`MASK = 31`

Bitmask with all bits on.

`MaxBitFlags = MaxBitRec * 32`

Maximum number of bits in TBits collection.

`MaxBitRec = \$FFFF div ( SizeOf ( longint ) )`

Maximum number of bit records in TBits.

`MaxListSize = Maxint div 16`

This constant sets the maximum number of elements in a TList (112).

`scAlt = \$8000`

Indicates ALT key in a keyboard shortcut.

`scCtrl = \$4000`

indicates CTRL key in a keyboard shortcut.

`scNone = 0`

Indicates no special key is presed in a keyboard shortcut.

`scShift = \$2000`

Indicates Shift key in a keyboard shortcut.

`soFromBeginning = 0`

Seek (142) starts relative to the stream origin.

```
soFromCurrent = 1
```

Seek (142) starts relative to the current position in the stream.

```
soFromEnd = 2
```

Seek (142) starts relative to the stream end.

```
toEOF = Char ( 0 )
```

Value returned by TParser.Token (125) when the end of the input stream was reached.

```
toFloat = Char ( 4 )
```

Value returned by TParser.Token (125) when a floating point value was found in the input stream.

```
toInteger = Char ( 3 )
```

Value returned by TParser.Token (125) when an integer was found in the input stream.

```
toString = Char ( 2 )
```

Value returned by TParser.Token (125) when a string was found in the input stream.

```
toSymbol = Char ( 1 )
```

Value returned by TParser.Token (125) when a symbol was found in the input stream.

Types

```
HModule = HModule
```

FPC doesn't support modules yet, so this is a dummy type.

```
HRSRC = LongInt
```

This type is provided for Delphi compatibilty, it is used for resource streams.

```
PPointerList = \^ TPointerList
```

Pointer to an array of pointers.

```
PStringItem = \^ TStringItem
```

Pointer to a TStringItem (29) record.

```
PStringItemList = \^ TStringItemList
```

Pointer to a TStringItemList (29).

```
TActiveXRegType = (axrComponentOnly,axrIncludeDescendants)
```

This type is provided for compatibility only, and is currently not used in Free Pascal.

Table 1.2: Enumeration values for type TActiveXRegType

Value	Explanation
axrComponentOnly	
axrIncludeDescendants	

Table 1.3: Enumeration values for type TAlignment

Value	Explanation
taCenter	Text is displayed centered.
taLeftJustify	Text is displayed aligned to the left
taRightJustify	Text is displayed aligned to the right.

```
TAlignment = (taLeftJustify,taRightJustify,taCenter)
```

The `TAlignment` type is used to specify the alignment of the text in controls that display a text.

```
TAncestorNotFoundEvent = procedure(Reader: TReader;
                                   const ComponentName: String;
                                   ComponentClass: TPersistentClass;
                                   var Component: TComponent) of object
```

This event occurs when an ancestor component cannot be found.

```
TBasicActionClass = Class of TBasicAction
```

TBasicAction (54) class reference.

```
TBasicActionLinkClass = Class of TBasicActionLink
```

TBasicActionLink (59) class reference.

```
TBitArray = Array[0..MaxBitRec-1] of cardinal
```

Array to store bits.

```
TCollectionItemClass = Class of TCollectionItem
```

`TCollectionItemClass` is used by the TCollection.ItemClass (82) property of TCollection (76) to identify the descendent class of TCollectionItem (83) which should be created and managed.

```
TComponentClass = Class of TComponent
```

The `TComponentClass` type is used when constructing TComponent (87) descendent instances and when registering components.

```
TComponentName = String
```

Names of components are of type `TComponentName`. By specifying a different type, the Object inspector can handle this property differently than a standard string property.

```
TComponentState= Set of (csLoading,csReading,csWriting,csDestroying,
                         csDesigning,csAncestor,csUpdating,csFixups,
                         csFreeNotification,csInline,csDesignInstance)
```

Indicates the state of the component during the streaming process.

```
TComponentStyle= Set of (csInheritable,csCheckPropAvail)
```

Describes the style of the component.

```
TCreateComponentEvent = procedure(Reader: TReader;
                                  ComponentClass: TComponentClass;
                                  var Component: TComponent) of object
```

Event handler type, occurs when a component instance must be created when a component is read from a stream.

```
TDuplicates = (dupIgnore,dupAccept,dupError)
```

Table 1.4: Enumeration values for type TDuplicates

Value	Explanation
dupAccept	Duplicate values can be added to the list.
dupError	If an attempt is made to add a duplicate value to the list, an EStringListError (43) exception is raised.
dupIgnore	Duplicate values will not be be added to the list, but no error will be triggered.

Type to describe what to do with duplicate values in a TStringlist (149).

```
TFilerFlag = (ffInherited,ffChildPos,ffInline)
```

Table 1.5: Enumeration values for type TFilerFlag

Value	Explanation
ffChildPos	The position of the child on it's parent is included.
ffInherited	Stored object is an inherited object.
ffInline	Used for frames.

The `TFiler` class uses this enumeration type to decide whether the streamed object was streamed as part of an inherited form or not.

```
TFilerFlags= Set of (ffChildPos,ffInherited,ffInline)
```

Set of TFilerFlag (24)

```
TFindAncestorEvent = procedure(Writer: TWriter;Component: TComponent;
                               const Name: String;
                               var Ancestor: TComponent;
                               var RootAncestor: TComponent) of object
```

Event that occurs w

```
TFindComponentClassEvent = procedure(Reader: TReader;
                                     const ClassName: String;
                                     var ComponentClass: TComponentClass)
                                      of object
```

Event handler type, occurs when a component class pointer must be found when reading a component from a stream.

```
TFindGlobalComponent = function(const Name: String) : TComponent
```

`TFindGlobalComponent` is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

The variable FindGlobalComponent (30) is a callback of type `TFindGlobalComponent`. It can be set by the IDE when an unknown reference is found, to offer the designer to redirect the link to a new component.

```
TFindMethodEvent = procedure(Reader: TReader;const MethodName: String;
                             var Address: Pointer;var Error: Boolean)
                              of object
```

If a TReader (128) instance needs to locate a method and it doesn't find it in the streamed form, then the OnFindMethod (136) event handler will be called, if one is installed. This event can be assigned in order to use different locating methods. If a method is found, then its address should be returned in `Address`. The `Error` should be set to `True` if the reader should raise an exception after the event was handled. If it is set to `False` no exception will be raised, even if no method was found. On entry, `Error` will be set to `True`.

```
TGetChildProc = procedure(Child: TComponent) of object
```

Callback used when obtaining child components.

```
TGetStrProc = procedure(const S: String) of object
```

This event is used as a callback to retrieve string values. It is used, among other things, to pass along string properties in property editors.

```
THandle = THandle
```

This type is used as the handle for THandleStream (110) stream descendents

```
THelpContext = -MaxLongint..MaxLongint
```

Range type to specify help contexts.

```
THelpEvent = function(Command: Word;Data: LongInt;var CallHelp: Boolean)
                       : Boolean of object
```

This event is used for display of online help.

```
THelpType = (htKeyword,htContext)
```

Enumeration type specifying the kind of help requested.

Table 1.6: Enumeration values for type THelpType

Value	Explanation
htContext	
htKeyword	

```
TIdentMapEntry = record
  Value : Integer;
  Name : String;
end
```

`TIdentMapEntry` is used internally by the IdentToInt (33) and IntToIdent (34) calls to store the mapping between the identifiers and the integers they represent.

```
TIdentToInt = function(const Ident: String;var Int: LongInt) : Boolean
```

`TIdentToInt` is a callback used to look up identifiers (`Ident`) and return an integer value corresponding to this identifier (`Int`). The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIdentToInt` should be specified when an integer is registered using the RegisterIntegerConsts (38) call.

```
TInitComponentHandler = function(Instance: TComponent;
                                 RootAncestor: TClass) : Boolean
```

```
TIntToIdent = function(Int: LongInt;var Ident: String) : Boolean
```

`TIdentToInt` is a callback used to look up integers (`Ident`) and return an identifier (`Ident`) that can be used to represent this integer value in an IDE. The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIntToIdent` should be specified when an integer is registered using the RegisterIntegerConsts (38) call.

```
TListNotification = (lnAdded,lnExtracted,lnDeleted)
```

Table 1.7: Enumeration values for type TListNotification

Value	Explanation
lnAdded	
lnDeleted	
lnExtracted	

Kind of list notification event.

```
TListSortCompare = function(Item1: Pointer;Item2: Pointer) : Integer
```

Callback type for the list sort algorithm.

```
TNotifyEvent = procedure(Sender: TObject) of object
```

Most event handlers are implemented as a property of type `TNotifyEvent`. When this is set to a certain method of a class, when the event occurs, the method will be called, and the class that generated the event will pass itself along as the `Sender` argument.

```
TOperation = (opInsert,opRemove)
```

Table 1.8: Enumeration values for type TOperation

Value	Explanation
opInsert	A new component is being inserted in the child component list.
opRemove	A component is being removed from the child component list.

Operation of which a component is notified.

```
TPersistentClass = Class of TPersistent
```

`TPersistentClass` is the class reference type for the TPersistent (125) class.

```
TPoint = TPoint
```

This record describes a coordinate. It is used to handle the Top (87) and Left (87) properties of TComponent (87).

X represents the X-Coordinate of the point described by the record. Y represents the Y-Coordinate of the point described by the record.

```
TPointerList = Array[0..MaxListSize-1] of Pointer
```

Type for an Array of pointers.

```
TPropertyNotFoundEvent = procedure(Reader: TReader;
                                   Instance: TPersistent;
                                   var PropName: String;IsPath: Boolean;
                                   var Handled: Boolean;
                                   var Skip: Boolean) of object
```

```
TReadComponentsProc = procedure(Component: TComponent) of object
```

Callback type when reading a component from a stream

```
TReaderError = procedure(Reader: TReader;const Message: String;
                         var Handled: Boolean) of object
```

Event handler type, called when an error occurs during the streaming.

```
TReaderProc = procedure(Reader: TReader) of object
```

The `TReaderProc` reader procedure is a callback procedure which will be used by a TPersistent (125) descendent to read user properties from a stream during the streaming process. The `Reader` argument is the writer object which can be used read properties from the stream.

```
TRect = TRect
```

`TRect` describes a rectangle in space with its upper-left (in (`Top`,`Left>`)) and lower-right (in (`Bottom` ,`Right`)) corners.

```
TReferenceNameEvent = procedure(Reader: TReader;var Name: String)
                                of object
```

Occurs when a named object needs to be looked up.

```
TSeekOrigin = (soBeginning,soCurrent,soEnd)
```

Table 1.9: Enumeration values for type TSeekOrigin

Value	Explanation
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

Specifies the origin of the TStream.Seek (142) method.

```
TSetMethodPropertyEvent = procedure(Reader: TReader;
                                    Instance: TPersistent;
                                    PropInfo: PPropInfo;
                                    const TheMethodName: String;
                                    var Handled: Boolean) of object
```

```
TSetNameEvent = procedure(Reader: TReader;Component: TComponent;
                          var Name: String) of object
```

Occurs when the reader needs to set a component's name.

```
TShiftState= Set of (ssShift,ssAlt,ssCtrl,ssLeft,ssRight,ssMiddle,
                     ssDouble,ssMeta,ssSuper,ssHyper,ssAltGr,ssCaps,
                     ssNum,ssScroll,ssTriple,ssQuad)
```

This type is used when describing a shortcut key or when describing what special keys are pressed on a keyboard when a key event is generated.

The set contains the special keys that can be used in combination with a 'normal' key.

```
TShortCut =  ( Word )..High ( Word )
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = TSmallPoint
```

Same as TPoint (27), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

```
TStreamProc = procedure(Stream: TStream) of object
```

Procedure type used in streaming.

```
TStringItem = record
  FString : String;
  FObject : TObject;
end
```

The `TStringItem` is used to store the string and object items in a TStringList (149) string list instance. It should never be used directly.

```
TStringItemList = Array[0..MaxListSize] of TStringItem
```

This declaration is provided for Delphi compatibility, it is not used in Free Pascal.

```
TStringListSortCompare = function(List: TStringList;Index1: Integer;
                                  Index2: Integer) : Integer
```

Callback type used in stringlist compares.

```
TThreadMethod = procedure of object
```

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle,tpLowest,tpLower,tpNormal,tpHigher,tpHighest,
                   tpTimeCritical)
```

Table 1.10: Enumeration values for type TThreadPriority

Value	Explanation
tpHigher	Thread runs at high priority
tpHighest	Thread runs at highest possible priority.
tpIdle	Thread only runs when other processes are idle.
tpLower	Thread runs at a lower priority.
tpLowest	Thread runs at the lowest priority.
tpNormal	Thread runs at normal process priority.
tpTimeCritical	Thread runs at realtime priority.

Enumeration specifying the priority at which a thread runs.

```
TValueType = (vaNull,vaList,vaInt8,vaInt16,vaInt32,vaExtended,vaString,
              vaIdent,vaFalse,vaTrue,vaBinary,vaSet,vaLString,vaNil,
              vaCollection,vaSingle,vaCurrency,vaDate,vaWString,vaInt64)
```

Enumerated type used to identify the kind of streamed property

```
TWriteMethodPropertyEvent = procedure(Writer: TWriter;
                                      Instance: TPersistent;
                                      PropInfo: PPropInfo;
                                      const MethodValue: TMethod;
                                      const DefMethodCodeValue: Pointer;
                                      var Handled: Boolean) of object
```

```
TWriterProc = procedure(Writer: TWriter) of object
```

The `TWriterProc` writer procedure is a callback procedure which will be used by a TPersistent (125) descendent to write user properties from a stream during the streaming process. The `Writer` argument is the writer object which can be used write properties to the stream.

Table 1.11: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows
vaCurrency	Currency value follows
vaDate	Date value follows
vaExtended	Extended value.
vaFalse	Boolean `False` value.
vaIdent	Identifier.
vaInt16	Integer value, 16 bits long.
vaInt32	Integer value, 32 bits long.
vaInt64	Integer value, 64 bits long.
vaInt8	Integer value, 8 bits long.
vaList	Identifies the start of a list of values
vaLString	Ansistring data follows.
vaNil	Nil pointer.
vaNull	Empty value. Ends a list.
vaSet	Set data follows.
vaSingle	Single type follows.
vaString	String value.
vaTrue	Boolean `True` value.
vaWString	Widestring value follows.

Variables

```
AddDataModule : procedure(DataModule: TDataModule) of object
```

```
ApplicationHandleException : procedure(Sender: TObject) of object
```

```
ApplicationShowException : procedure(E: Exception) of object
```

```
FindGlobalComponent : TFindGlobalComponent
```

`FindGlobalComponent` is a callback of type TFindGlobalComponent (25). It can be set by the IDE when an unknown reference is found, to offer the user to redirect the link to a new component.

It is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

```
MainThreadID : THandle
```

ID of main thread. Unused at this point.

```
RegisterComponentsProc : procedure(const Page: String;
          ComponentClasses: Array[] of TComponentClass)
```

`RegisterComponentsProc` can be set by an IDE to be notified when new components are being registered. Application programmers should never have to set `RegisterComponentsProc`

`RegisterNoIconProc : procedure(ComponentClasses: Array[] of TComponentClass)`

`RegisterNoIconProc` can be set by an IDE to be notified when new components are being registered, and which do not need an Icon in the component palette. Application programmers should never have to set `RegisterComponentsProc`

`RemoveDataModule : procedure(DataModule: TDataModule) of object`

1.4 Procedures and functions

BeginGlobalLoading

Synopsis: Not yet implemented

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented

Bounds

Synopsis: Returns a `TRect` structure with the bounding rect of the given location and size.

Declaration:
```
function Bounds(ALeft: Integer;ATop: Integer;AWidth: Integer;
                AHeight: Integer) : TRect
```

Visibility: default

Description: `Bounds` returns a TRect (28) record with the given origin `(ALeft,ATop)` and dimensions `(AWidth,AHeight)` filled in.

CollectionsEqual

Synopsis: Returns `True` if two collections are equal.

Declaration: `function CollectionsEqual(C1: TCollection;C2: TCollection) : Boolean`

Visibility: default

Description: `CollectionsEqual` is not yet implemented. It simply returns `False`

EndGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure EndGlobalLoading`

Visibility: default

Description: Not yet implemented.

FindClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function FindClass(const AClassName: String) : TPersistentClass`

Visibility: default

Description: `FindClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, an exception is raised.

The GetClass (32) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: RegisterClass (37), GetClass (32)

FindNestedComponent

Synopsis: Finds the component with name path starting at the indicated root component.

Declaration:
```
function FindNestedComponent(Root: TComponent;const NamePath: String)
                              : TComponent
```

Visibility: default

Description: `FindNestedComponent` will descend through the list of owned components (starting at `Root`) and will return the component whose name path matches `NamePath`. As a path separator the characters . (dot), - (dash) and > (greater than) can be used

See also: GlobalFixupReferences (33)

GetClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function GetClass(const AClassName: String) : TPersistentClass`

Visibility: default

Description: `GetClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, `Nil` is returned.

The FindClass (32) function will raise an exception if the does not find the class.

See also: RegisterClass (37), GetClass (32)

GetFixupInstanceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component, whose reference contains `ReferenceRootName`

Declaration:
```
procedure GetFixupInstanceNames(Root: TComponent;
                                const ReferenceRootName: String;
                                Names: TStrings)
```

Visibility: default

Description: `GetFixupInstanceNames` examines the list of unresolved references and returns the names of classes that contain unresolved references to the `Root` component in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupReferenceNames (33), GlobalFixupReferences (33)

GetFixupReferenceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component.

Declaration: `procedure GetFixupReferenceNames(Root: TComponent;Names: TStrings)`

Visibility: default

Description: `GetFixupReferenceNames` examines the list of unresolved references and returns the names of properties that must be resolved for the component `Root` in the list `Names`. The list is not cleared prior to filling it.

See also: GetFixupInstanceNames (32), GlobalFixupReferences (33)

GlobalFixupReferences

Synopsis: Called to resolve unresolved references after forms are loaded.

Declaration: `procedure GlobalFixupReferences`

Visibility: default

Description: `GlobalFixupReferences` runs over the list of unresolved references and tries to resolve them. This routine should under normal circumstances not be called in an application programmer's code. It is called automatically by the streaming system after a component has been instantiated and its properties read from a stream. It will attempt to resolve references to other global components.

See also: GetFixupReferenceNames (33), GetFixupInstanceNames (32)

IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration:
```
function IdentToInt(const Ident: String;var Int: LongInt;
                    const Map: Array[] of TIdentMapEntry) : Boolean
```

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Name` field matches `Ident` and returns the corresponding integer value in `Int`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (26), TIntToIdent (26), IntToIdent (34), TIdentMapEntry (26)

InitComponentRes

Synopsis: Provided for Delphi compatibility only

Declaration:
```
function InitComponentRes(const ResName: String;Instance: TComponent)
                           : Boolean
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `false`.

See also: ReadComponentRes (36)

InitInheritedComponent

Synopsis: Initializes a component descending from `RootAncestor`

Declaration:
```
function InitInheritedComponent(Instance: TComponent;
                                RootAncestor: TClass) : Boolean
```

Visibility: default

Description: `InitInheritedComponent` should be called from a constructor to read properties of the component `Instance` from the streaming system. The `RootAncestor` class is the root class from which `Instance` is a descendent. This must be one of `TDatamodule`, `TCustomForm` or `TFrame`.

The function returns `True` if the properties were successfully read from a stream or `False` if some error occurred.

See also: ReadComponentRes (36), ReadComponentResEx (36), ReadComponentResFile (36)

IntToIdent

Synopsis: Looks up an identifier for an integer value in a identifier-to-integer map list.

Declaration:
```
function IntToIdent(Int: LongInt;var Ident: String;
                    const Map: Array[] of TIdentMapEntry) : Boolean
```

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Value` field matches `Int` and returns the corresponding identifier in `Ident`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: TIdentToInt (26), TintToIdent (26), IdentToInt (33), TIdentMapEntry (26)

LineStart

Synopsis: Finds the start of a line in `Buffer` before `BufPos`.

Declaration:
```
function LineStart(Buffer: PChar;BufPos: PChar) : PChar
```

Visibility: default

Description: `LineStart` reversely scans `Buffer` starting at `BufPos` for a linefeed character. It returns a pointer at the linefeed character.

NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration:
```
procedure NotifyGlobalLoading
```

Visibility: default

Description: Not yet implemented.

ObjectBinaryToText

Synopsis: Converts an object stream from a binary to a text format.

Declaration: `procedure ObjectBinaryToText(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectBinaryToText` reads an object stream in binary format from `Input` and writes the object stream in text format to `Output`. No components are instantiated during the process, this is a pure conversion routine.

See also: ObjectTextToBinary (35)

ObjectResourceToText

Synopsis: Converts an object stream from a (windows) resource to a text format.

Declaration: `procedure ObjectResourceToText(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectResourceToText` reads the resource header from the `Input` stream and then passes the streams to ObjectBinaryToText (35)

See also: ObjectBinaryToText (35), ObjectTextToResource (35)

ObjectTextToBinary

Synopsis: Converts an object stream from a text to a binary format.

Declaration: `procedure ObjectTextToBinary(Input: TStream;Output: TStream)`

Visibility: default

Description: Converts an object stream from a text to a binary format.

ObjectTextToResource

Synopsis: Converts an object stream from a text to a (windows) resource format.

Declaration: `procedure ObjectTextToResource(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectTextToResource` reads an object stream in text format from `Input` and writes a resource stream to `Output`.

Note that for the current implementation of this method in Free Pascal, the output stream should support positioning. (e.g. it should not be a pipe)

See also: ObjectBinaryToText (35), ObjectResourceToText (35)

Point

Synopsis: Returns a `TPoint` record with the given coordinates.

Declaration:
```
function Point(AX: Integer;AY: Integer) : TPoint
```

Visibility: default

Description: `Point` returns a TPoint (27) record with the given coordinates AX and AY filled in.

See also: TPoint (27), SmallPoint (40), Rect (36), Bounds (31)

ReadComponentRes

Synopsis: Read component properties from a resource in the current module

Declaration:
```
function ReadComponentRes(const ResName: String;Instance: TComponent)
                          : TComponent
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

ReadComponentResEx

Synopsis: Read component properties from a resource in the specified module

Declaration:
```
function ReadComponentResEx(HInstance: THandle;const ResName: String)
                            : TComponent
```

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

ReadComponentResFile

Synopsis: Read component properties from a specified resource file

Declaration:
```
function ReadComponentResFile(const FileName: String;
                              Instance: TComponent) : TComponent
```

Visibility: default

Description: `ReadComponentResFile` starts reading properties for `Instance` from the file `FileName`. It creates a filestream from `FileName` and then calls the TStream.ReadComponentRes (143) method to read the state of the component from the stream.

See also: TStream.ReadComponentRes (143), WriteComponentResFile (40)

Rect

Synopsis: Returns a `TRect` record with the given coordinates.

Declaration:
```
function Rect(ALeft: Integer;ATop: Integer;ARight: Integer;
             ABottom: Integer) : TRect
```

Visibility: default

Description: `Rect` returns a TRect (28) record with the given top-left `(ALeft,ATop)` and bottom-right `(ABottom,ARight)` corners filled in.

No checking is done to see whether the coordinates are valid.

See also: TRect (28), Point (36), SmallPoint (40), Bounds (31)

RedirectFixupReferences

Synopsis: Redirects references under the `root` object from `OldRootName` to `NewRootName`

Declaration:
```
procedure RedirectFixupReferences(Root: TComponent;
                                  const OldRootName: String;
                                  const NewRootName: String)
```

Visibility: default

Description: `RedirectFixupReferences` examines the list of unresolved references and replaces references to a root object named `OldRootName` with references to root object `NewRootName`.

An application programmer should never need to call `RedirectFixupReferences`. This function can be used by an IDE to support redirection of broken component links.

See also: RemoveFixupReferences (39)

RegisterClass

Synopsis: Registers a class with the streaming system.

Declaration:
```
procedure RegisterClass(AClass: TPersistentClass)
```

Visibility: default

Description: `RegisterClass` registers the class `AClass` in the streaming system. After the class has been registered, it can be read from a stream when a reference to this class is encountered.

See also: RegisterClasses (38), RegisterClassAlias (37), RegisterComponents (38), UnregisterClass (40)

RegisterClassAlias

Synopsis: Registers a class alias with the streaming system.

Declaration:
```
procedure RegisterClassAlias(AClass: TPersistentClass;
                             const Alias: String)
```

Visibility: default

Description: `RegisterClassAlias` registers a class alias in the streaming system. If a reference to a class `Alias` is encountered in a stream, then an instance of the class `AClass` will be created instead by the streaming code.

See also: RegisterClass (37), RegisterClasses (38), RegisterComponents (38), UnregisterClass (40)

RegisterClasses

Synopsis: Registers multiple classes with the streaming system.

Declaration:
```
procedure RegisterClasses(AClasses: Array[] of TPersistentClass)
```

Visibility: default

Description: `RegisterClasses` registers the specified classes `AClass` in the streaming system. After the classes have been registered, they can be read from a stream when a reference to this class is encountered.

See also: RegisterClass (37), RegisterClassAlias (37), RegisterComponents (38), UnregisterClass (40)

RegisterComponents

Synopsis: Registers components for the component palette.

Declaration:
```
procedure RegisterComponents(const Page: String;
                             ComponentClasses: Array[] of TComponentClass)
```

Visibility: default

Description: `RegisterComponents` registers the component on the appropriate component page. The component pages can be used by an IDE to display the known components so an application programmer may pick and use the components in his programs.

Registercomponents inserts the component class in the correct component page. If the `RegisterComponentsProc` procedure is set, this is called as well. Note that this behaviour is different from Delphi's behaviour where an exception will be raised if the procedural variable is not set.

See also: RegisterClass (37), RegisterNoIcon (39)

RegisterInitComponentHandler

Declaration:
```
procedure RegisterInitComponentHandler(ComponentClass: TComponentClass;
                                       Handler: TInitComponentHandler)
```

Visibility: default

RegisterIntegerConsts

Synopsis: Registers some integer-to-identifier mappings.

Declaration:
```
procedure RegisterIntegerConsts(IntegerType: Pointer;
                                IdentToIntFn: TIdentToInt;
                                IntToIdentFn: TIntToIdent)
```

Visibility: default

Description: `RegisterIntegerConsts` registers a pair of callbacks to be used when an integer of type `IntegerType` must be mapped to an identifier (using `IntToIdentFn`) or when an identifier must be mapper to an integer (using `IdentToIntFn`).

Component programmers can use `RegisterIntegerConsts` to associate a series of identifier strings with integer values for a property. A necessary condition is that the property should have a separate type declared using the `type integer` syntax. If a type of integer is defined in this way, an IDE can show symbolic names for the values of these properties.

The `IntegerType` should be a pointer to the type information of the integer type. The `IntToIdentFn` and `IdentToIntFn` are two callbacks that will be used when converting between the identifier and integer value and vice versa. The functions IdentToInt (33) and IntToIdent (34) can be used to implement these callback functions.

See also: TIdentToInt (26), TIntToIdent (26), IdentToInt (33), IntToIdent (34)

RegisterNoIcon

Synopsis: Registers components that have no icon on the component palette.

Declaration:
```
procedure RegisterNoIcon(ComponentClasses: Array[] of TComponentClass)
```

Visibility: default

Description: `RegisterNoIcon` performs the same function as RegisterComponents (38) except that it calls RegisterNoIconProc (31) instead of RegisterComponentsProc (30)

See also: RegisterNoIconProc (31), RegisterComponents (38)

RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration:
```
procedure RegisterNonActiveX
                             (ComponentClasses: Array[] of TComponentClass;
                             AxRegType: TActiveXRegType)
```

Visibility: default

Description: Not yet implemented in Free Pascal

RemoveFixupReferences

Synopsis: Removes references to rootname from the fixup list.

Declaration:
```
procedure RemoveFixupReferences(Root: TComponent;const RootName: String)
```

Visibility: default

Description: `RemoveFixupReferences` examines the list of unresolved references and removes references to a root object pointing at `Root` or a root component named `RootName`.

An application programmer should never need to call `RemoveFixupReferences`. This function can be used by an IDE to support removal of broken component links.

See also: RedirectFixupReferences (37)

RemoveFixups

Synopsis: Removes `Instance` from the fixup list.

Declaration:
```
procedure RemoveFixups(Instance: TPersistent)
```

Visibility: default

Description: `RemoveFixups` removes all entries for component `Instance` from the list of unresolved references.a

See also: RedirectFixupReferences (37), RemoveFixupReferences (39)

SmallPoint

Synopsis: Returns a `TSmallPoint` record with the given coordinates.

Declaration: `function SmallPoint(AX: SmallInt;AY: SmallInt) : TSmallPoint`

Visibility: default

Description: `SmallPoint` returns a TSmallPoint (28) record with the given coordinates `AX` and `AY` filled in.

See also: TSmallPoint (28), Point (36), Rect (36), Bounds (31)

UnRegisterClass

Synopsis: Unregisters a class from the streaming system.

Declaration: `procedure UnRegisterClass(AClass: TPersistentClass)`

Visibility: default

Description: `UnregisterClass` removes the class `AClass` from the class definitions in the streaming system.

See also: UnRegisterClasses (40), UnRegisterModuleClasses (40), RegisterClass (37)

UnRegisterClasses

Synopsis: Unregisters multiple classes from the streaming system.

Declaration: `procedure UnRegisterClasses(AClasses: Array[] of TPersistentClass)`

Visibility: default

Description: `UnregisterClasses` removes the classes in `AClasses` from the class definitions in the streaming system.

UnRegisterModuleClasses

Synopsis: Unregisters classes registered by module.

Declaration: `procedure UnRegisterModuleClasses(Module: HModule)`

Visibility: default

Description: `UnRegisterModuleClasses` unregisters all classes which reside in the module `Module`. For each registered class, the definition pointer is checked to see whether it resides in the module, and if it does, the definition is removed.

See also: UnRegisterClass (40), UnRegisterClasses (40), RegisterClasses (38)

WriteComponentResFile

Synopsis: Write component properties to a specified resource file

Declaration:
```
procedure WriteComponentResFile(const FileName: String;
                                Instance: TComponent)
```

Visibility: default

Description: `WriteComponentResFile` starts writing properties of `Instance` to the file `FileName`. It creates a filestream from `FileName` and then calls TStream.WriteComponentRes (144) method to write the state of the component to the stream.

See also: TStream.WriteComponentRes (144), ReadComponentResFile (36)

1.5 EBitsError

Description

When an index of a bit in a TBits (69) is out of the valid range (`0` to `Count-1`) then a `EBitsError` exception is raised.

1.6 EClassNotFound

Description

When the streaming system needs to create a component, it looks for the class pointer (VMT) in the list of registered classes by its name. If this name is not found, then an `EClassNotFound` is raised.

1.7 EComponentError

Description

When an error occurs during the registration of a component, or when naming a component, then a `EComponentError` is raised. Possible causes are:

1. An name with an illegal character was assigned to a component.
2. A component with the same name and owner already exists.
3. The component registration system isn't set up properly.

1.8 EFCreateError

Description

When the operating system reports an error during creation of a new file in the Filestream Constructor (109), a `EFCreateError` is raised.

1.9 EFilerError

Description

This class serves as an ancestor class for exceptions that are raised when an error occurs during component streaming. A `EFilerError` exception is raised when a class is registered twice.

1.10 EFOpenError

Description

When the operating system reports an error during the opening of a file in the Filestream Constructor (109), a `EFOpenError` is raised.

1.11 EInvalidImage

Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

1.12 EInvalidOperation

Description

This exception is not used in Free Pascal, it is defined for Delphi compatibiliy purposes only.

1.13 EListError

Description

If an error occurs in one of the TList (112) or TStrings (155) methods, then a `EListError` exception is raised. This can occur in one of the following cases:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. An attempt was made to reduce the capacity of the list below the current element count.
4. An attempt was made to set the list count to a negative value.
5. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
6. An attempt was made to move an item to a position outside the list's bounds.

1.14 EMethodNotFound

Description

This exception is no longer used in the streaming system. This error is replaced by a EReadError (43).

1.15 EOutOfResources

Description

This exception is not used in Free Pascal, it is defined for Delphi compatibiliy purposes only.

1.16 EParserError

Description

When an error occurs during the parsing of a stream, an `EParserError` is raised. Usually this indicates that an invalid token was found on the input stream, or the token read from the stream wasn't the expected token.

1.17 EReadError

Description

If an error occurs when reading from a stream, a `EReadError` exception is raised. Possible causes for this are:

1. Not enough data is available when reading from a stream
2. The stream containing a component's data contains invalid data. this will occurr only when reading a component from a stream.

1.18 EResNotFound

Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

1.19 EStreamError

Description

An `EStreamError` is raised when an error occurs during reading from or writng to a stream: Possible causes are

1. Not enough data is available in the stream.
2. Trying to seek beyond the beginning or end of the stream.
3. Trying to set the capacity of a memory stream and no memory is available.
4. Trying to write to a resource stream.

1.20 EStringListError

Description

When an error occurs in one of the methods of TStrings (155) then an `EStringListError` is raised. This can have one of the following causes:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.

3. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
4. An attempt was made to add a duplicate entry to a TStringList (149) when TStringList.AllowDuplicates (149) is `False`.

1.21 EThread

Description

Thread error exception.

1.22 EThreadDestroyCalled

1.23 EWriteError

Description

If an error occurs when writing to a stream, a `EWriteError` exception is raised. Possible causes for this are:

1. The stream doesn't allow writing.
2. An error occurred when writing a property to a stream.

1.24 IStringsAdapter

Description

Is not yet supported in Free Pascal.

1.25 TAbstractObjectReader

Description

The Free Pascal streaming mechanism, while compatible with Delphi's mechanism, differs from it in the sense that the streaming mechanism uses a driver class when streaming components. The `TAbstractObjectReader` class is the base driver class for reading property values from streams. It consists entirely of abstract methods, which must be implemented by descendent classes.

Different streaming mechanisms can be implemented by making a descendent from `TAbstractObjectReader`. The TBinaryObjectReader (62) class is such a descendent class, which streams data in binary (Delphi compatible) format.

All methods described in this class, mustbe implemented by descendent classes.

Method overview

Page	Method	Description
46	BeginComponent	Marks the reading of a new component.
46	BeginProperty	Marks the reading of a property value.
45	BeginRootComponent	Starts the reading of the root component.
45	NextValue	Returns the type of the next value in the stream.
46	ReadBinary	Read binary data from the stream.
47	ReadDate	Read a date value from the stream.
47	ReadFloat	Read a float value from the stream.
48	ReadIdent	Read an identifier from the stream.
48	ReadInt16	Read a 16-bit integer from the stream.
49	ReadInt32	Read a 32-bit integer from the stream.
49	ReadInt64	Read a 64-bit integer from the stream.
48	ReadInt8	Read an 8-bit integer from the stream.
49	ReadSet	Reads a set from the stream.
47	ReadSingle	Read a single (real-type) value from the stream.
50	ReadStr	Read a shortstring from the stream
50	ReadString	Read a string of type `StringType` from the stream.
45	ReadValue	Reads the type of the next value.
50	SkipComponent	Skip till the end of the component.
51	SkipValue	Skip the current value.

TAbstractObjectReader.NextValue

Synopsis: Returns the type of the next value in the stream.

Declaration: `function NextValue : TValueType;  Virtual;  Abstract`

Visibility: public

Description: This function should return the type of the next value in the stream, but should not read it, i.e. the stream position should not be altered by this method. This is used to 'peek' in the stream what value is next.

See also: TAbstractObjectReader.ReadValue (45)

TAbstractObjectReader.ReadValue

Synopsis: Reads the type of the next value.

Declaration: `function ReadValue : TValueType;  Virtual;  Abstract`

Visibility: public

Description: This function returns the type of the next value in the stream and reads it. i.e. after the call to this method, the stream is positioned to read the value of the type returned by this function.

See also: TAbstractObjectReader.ReadValue (45)

TAbstractObjectReader.BeginRootComponent

Synopsis: Starts the reading of the root component.

Declaration: `procedure BeginRootComponent;  Virtual;  Abstract`

Visibility: public

Description: This function can be used to initialize the driver class for reading a component. It is called once at the beginning of the read process, and is immediatly followed by a call to BeginComponent (46).

See also: TAbstractObjectReader.BeginComponent (46)

TAbstractObjectReader.BeginComponent

Synopsis: Marks the reading of a new component.

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;var AChildPos: Integer;
                         var CompClassName: String;var CompName: String)
                         ;  Virtual;  Abstract
```

Visibility: public

Description: This method is called when the streaming process wants to start reading a new component.

Descendent classes should override this method to read the start of a component new component definition and return the needed arguments. `Flags` should be filled with any flags that were found at the component definition, as well as `AChildPos`. The `CompClassName` should be filled with the class name of the streamed component, and the `CompName` argument should be filled with the name of the component.

See also: TAbstractObjectReader.BeginRootComponent (45), TAbstractObjectReader.BeginProperty (46)

TAbstractObjectReader.BeginProperty

Synopsis: Marks the reading of a property value.

Declaration: `function BeginProperty : String;  Virtual;  Abstract`

Visibility: public

Description: `BeginProperty` is called by the streaming system when it wants to read a new property. The return value of the function is the name of the property which can be read from the stream.

See also: TAbstractObjectReader.BeginComponent (46)

TAbstractObjectReader.ReadBinary

Synopsis: Read binary data from the stream.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream);  Virtual;  Abstract`

Visibility: public

Description: `ReadBinary` is called when binary data should be read from the stream (i.e. after ReadValue (45) returned a valuetype of `vaBinary`). The data should be stored in the `DestData` memory stream by descendent classes.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadFloat

Synopsis: Read a float value from the stream.

Declaration: `function ReadFloat : Extended;  Virtual;  Abstract`

Visibility: public

Description: `ReadFloat` is called by the streaming system when it wants to read a float from the stream (i.e. after ReadValue (45) returned a valuetype of `vaExtended`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadSingle

Synopsis: Read a single (real-type) value from the stream.

Declaration: `function ReadSingle : Single;  Virtual;  Abstract`

Visibility: public

Description: `ReadSingle` is called by the streaming system when it wants to read a single-type float from the stream (i.e. after ReadValue (45) returned a valuetype of `vaSingle`). The return value should be the value of the float.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadDate

Synopsis: Read a date value from the stream.

Declaration: `function ReadDate : TDateTime;  Virtual;  Abstract`

Visibility: public

Description: `ReadDate` is called by the streaming system when it wants to read a date/time value from the stream (i.e. after ReadValue (45) returned a valuetype of `vaDate`). The return value should be the date/time value. (This value can be stored as a float, since `TDateTime` is nothing but a float.)

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadIdent

Synopsis: Read an identifier from the stream.

Declaration: `function ReadIdent(ValueType: TValueType) : String;  Virtual;  Abstract`

Visibility: public

Description: `ReadIdent` is called by the streaming system if it expects to read an identifier of type `ValueType` from the stream after a call to ReadValue (45) returned `vaIdent`. The identifier should be returned as a string. Note that in some cases the identifier does not actually have to be in the stream;

Table 1.12:

ValueType	Expected value
vaIdent	Read from stream.
vaNil	'Nil'. This does not have to be read from the stream.
vaFalse	'False'. This does not have to be read from the stream.
vaTrue	'True'. This does not have to be read from the stream.
vaNull	'Null'. This does not have to be read from the stream.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadInt8

Synopsis: Read an 8-bit integer from the stream.

Declaration: `function ReadInt8 : ShortInt;  Virtual;  Abstract`

Visibility: public

Description: `ReadInt8` is called by the streaming process if it expects to read an integer value with a size of 8 bits (1 byte) from the stream (i.e. after ReadValue (45) returned a valuetype of `vaInt8`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 1 byte.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadInt16

Synopsis: Read a 16-bit integer from the stream.

Declaration: `function ReadInt16 : SmallInt;  Virtual;  Abstract`

Visibility: public

Description: `ReadInt16` is called by the streaming process if it expects to read an integer value with a size of 16 bits (2 bytes) from the stream (i.e. after ReadValue (45) returned a valuetype of `vaInt16`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 2 bytes.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadInt32

Synopsis: Read a 32-bit integer from the stream.

Declaration: `function ReadInt32 : LongInt;  Virtual;  Abstract`

Visibility: public

Description: `ReadInt32` is called by the streaming process if it expects to read an integer value with a size of 32 bits (4 bytes) from the stream (i.e. after ReadValue (45) returned a valuetype of `vaInt32`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 4 bytes.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadInt64

Synopsis: Read a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64;  Virtual;  Abstract`

Visibility: public

Description: `ReadInt64` is called by the streaming process if it expects to read an int64 value with a size of 64 bits (8 bytes) from the stream (i.e. after ReadValue (45) returned a valuetype of `vaInt64`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 8 bytes.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadSet

Synopsis: Reads a set from the stream.

Declaration: `function ReadSet(EnumType: Pointer) : Integer;  Virtual;  Abstract`

Visibility: public

Description: This method is called by the streaming system if it expects to read a set from the stream (i.e. after ReadValue (45) returned a valuetype of `vaSet`). The return value is the contents of the set, encoded in a bitmask the following way:

For each (enumerated) value in the set, the bit corresponding to the ordinal value of the enumerated value should be set. i.e. as `1 shl ord(value)`.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadStr (50), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadStr

Synopsis: Read a shortstring from the stream

Declaration: `function ReadStr : String;  Virtual;  Abstract`

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a shortstring from the stream (i.e. after ReadValue (45) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadString (50)

TAbstractObjectReader.ReadString

Synopsis: Read a string of type `StringType` from the stream.

Declaration:
```
function ReadString(StringType: TValueType) : String;  Virtual
                          ;  Abstract
```

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after ReadValue (45) returned a valuetype of `vaLString`,`vaWstring` or `vaString`). The return value is the string.

See also: TAbstractObjectReader.ReadFloat (47), TAbstractObjectReader.ReadDate (47), TAbstractObjectReader.ReadSingle (47), TAbstractObjectReader.ReadIdent (48), TAbstractObjectReader.ReadInt8 (48), TAbstractObjectReader.ReadInt16 (48), TAbstractObjectReader.ReadInt32 (49), TAbstractObjectReader.ReadInt64 (49), TabstractObjectReader.ReadSet (49), TabstractObjectReader.ReadStr (50)

TAbstractObjectReader.SkipComponent

Synopsis: Skip till the end of the component.

Declaration:
```
procedure SkipComponent(SkipComponentInfos: Boolean);  Virtual
                               ;  Abstract
```

Visibility: public

Description: This method is used to skip the entire declaration of a component in the stream. Each descendent of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: TAbstractObjectReader.BeginComponent (46), TAbstractObjectReader.SkipValue (51)

TAbstractObjectReader.SkipValue

Synopsis: Skip the current value.

Declaration: `procedure SkipValue;  Virtual;  Abstract`

Visibility: public

Description: `SkipValue` should be used when skipping a value in the stream; The method should determine the type of the value which should be skipped by itself, if this is necessary.

See also: TAbstractObjectReader.SkipComponent (50)

1.26 TAbstractObjectWriter

Description

Abstract driver class for writing component data.

Method overview

Page	Method	Description
51	BeginCollection	Start writing a collection.
51	BeginComponent	Start writing a component
52	BeginList	Start writing a list.
52	BeginProperty	Start writing a property
52	EndList	Mark the end of a list.
52	EndProperty	Marks the end of writing of a property.
52	WriteBinary	Writes binary data to the stream.
53	WriteBoolean	Writes a boolean value to the stream.
53	WriteDate	Writes a date type to the stream.
53	WriteFloat	Writes a float value to the stream.
53	WriteIdent	Writes an identifier to the stream.
53	WriteInteger	Writes an integer value to the stream
54	WriteMethodName	Writes a methodname to the stream.
54	WriteSet	Writes a set value to the stream.
53	WriteSingle	Writes a single-type real value to the stream.
54	WriteString	Writes a string value to the stream.

TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection;  Virtual;  Abstract`

Visibility: public

Description: Start writing a collection.

TAbstractObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration:
```
procedure BeginComponent(Component: TComponent;Flags: TFilerFlags;
                         ChildPos: Integer);  Virtual;  Abstract
```

Visibility: public

Description: Start writing a component

TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList;  Virtual;  Abstract`

Visibility: public

Description: Start writing a list.

TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList;  Virtual;  Abstract`

Visibility: public

Description: Mark the end of a list.

TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: String);  Virtual;  Abstract`

Visibility: public

Description: Start writing a property

TAbstractObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty;  Virtual;  Abstract`

Visibility: public

Description: Marks the end of writing of a property.

TAbstractObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer;Count: LongInt);  Virtual;  Abstract`

Visibility: public

Description: Writes binary data to the stream.

TAbstractObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean);  Virtual;  Abstract`

Visibility: public

Description: Writes a boolean value to the stream.

TAbstractObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended);  Virtual;  Abstract`

Visibility: public

Description: Writes a float value to the stream.

TAbstractObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single);  Virtual;  Abstract`

Visibility: public

Description: Writes a single-type real value to the stream.

TAbstractObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime);  Virtual;  Abstract`

Visibility: public

Description: Writes a date type to the stream.

TAbstractObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String);  Virtual;  Abstract`

Visibility: public

Description: Writes an identifier to the stream.

TAbstractObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream

Declaration: `procedure WriteInteger(Value: Int64);  Virtual;  Abstract`

Visibility: public

Description: Writes an integer value to the stream

TAbstractObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String);  Virtual;  Abstract`

Visibility: public

Description: Writes a methodname to the stream.

TAbstractObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt;SetType: Pointer);  Virtual;  Abstract`

Visibility: public

Description: Writes a set value to the stream.

TAbstractObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String);  Virtual;  Abstract`

Visibility: public

Description: Writes a string value to the stream.

1.27 TBasicAction

Description

`TBasicAction` implements a basic action class from which all actions are derived. It introduces all basic methods of an action, and implements functionality to maintain a list of clients, i.e. components that are connected with this action.

Do not create instances of `TBasicAction`. Instead, create a descendent class and create an instance of this class instead.

Method overview

Page	Method	Description
55	Change	Calls the OnChange (58) handler.
55	Create	Creates a new instance of a TBasicAction (54) class.
56	Destroy	Destroys the action.
57	Execute	Triggers the OnExecute (58) event
56	ExecuteTarget	Executes the action on the `Target` object
56	HandlesTarget	Determines whether `Target` can be handled by this action
57	RegisterChanges	Registers a new client with the action.
55	SetOnExecute	Assigns an OnExecute (58) event handler
57	UnRegisterChanges	Unregisters a client from the list of clients
57	Update	Triggers the OnUpdate (59) event
56	UpdateTarget	Notify client controls when the action updates itself.

Property overview

Page	Property	Access	Description
58	ActionComponent	rw	Returns the component that initiated the action.
58	OnChange	rw	Occurs when one of the action's properties changes.
58	OnExecute	rw	Event triggered when the action executes.
59	OnUpdate	rw	Event trigged when the application is idle.

TBasicAction.Change

Synopsis: Calls the OnChange (58) handler.

Declaration: `procedure Change;  Virtual`

Visibility: protected

Description: `Change` calls the OnChange (58) handler if one is assigned.

Application programmers should not call `Change` directly. It is called automatically if a property of an action component changes.

Descendent classes of `TBasicAction` should call explicitly call `Change` if one of their properties that affect client controls changes its value.

TBasicAction.SetOnExecute

Synopsis: Assigns an OnExecute (58) event handler

Declaration: `procedure SetOnExecute(Value: TNotifyEvent);  Virtual`

Visibility: protected

Description: `SetOnExecute` sets the OnExecute (58) handler of the component. It also propagates this event to all client controls, and finally triggers the OnChange (58) event.

See also: TBasicAction.OnExecute (58), TBasicAction.OnChange (58)

TBasicAction.Create

Synopsis: Creates a new instance of a TBasicAction (54) class.

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

Description: `Create` calls the inherited constructor, and then initializes the list of clients controls (or action lists) by adding the `AClient` argument to the list of client controls.

Under normal circumstances it should not be necessary to create a `TBasicAction` descendent manually, actions are created in an IDE.

See also: TBasicAction.Destroy (56), TBasicAction.AssignClient (54)

TBasicAction.Destroy

Synopsis: Destroys the action.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` cleans up the list of client controls and then calls the inherited destructor.

An application programmer should not call `Destroy` directly; Instead `Free` should be called, if it needs to be called at all. Normally the controlling class (e.g. a TActionList) will destroy the action.

TBasicAction.HandlesTarget

Synopsis: Determines whether `Target` can be handled by this action

Declaration: `function HandlesTarget(Target: TObject) : Boolean;  Virtual`

Visibility: public

Description: `HandlesTarget` returns `True` if `Target` is a valid client for this action and if so, if it is in a suitable state to execute the action. An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

In `TBasicAction` this method is empty; descendent classes should override this method to implement appropriate checks.

See also: TBasicAction.UpdateTarget (56), TBasicAction.ExecuteTarget (56)

TBasicAction.UpdateTarget

Synopsis: Notify client controls when the action updates itself.

Declaration: `procedure UpdateTarget(Target: TObject);  Virtual`

Visibility: public

Description: `UpdateTarget` should update the client control specified by `Target` when the action updates itself. In `TBasicAction`, the implementation of `UpdateTarget` is empty. Descendent classes should override and implement `UpdateTarget` to actually update the `Target` object.

An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

See also: TBasicAction.HandlesTarget (56), TBasicAction.ExecuteTarget (56)

TBasicAction.ExecuteTarget

Synopsis: Executes the action on the `Target` object

Declaration: `procedure ExecuteTarget(Target: TObject);  Virtual`

Visibility: public

Description: `ExecuteTarget` performs the action on the `Target` object. In `TBasicAction` this method does nothing. Descendent classes should implement the action to be performed. For instance an action to post data in a dataset could call the `Post` method of the dataset.

An application programmer should never call `ExecuteTarget` directly.

See also: TBasicAction.HandlesTarget (56), TBasicAction.ExecuteTarget (56), TBasicAction.Execute (57)

TBasicAction.Execute

Synopsis: Triggers the OnExecute (58) event

Declaration: `function Execute : Boolean;  Dynamic`

Visibility: public

Description: `Execute` triggers the `OnExecute` event, if one is assigned. It returns `True` if the event handler was called, `False` otherwise.

TBasicAction.RegisterChanges

Synopsis: Registers a new client with the action.

Declaration: `procedure RegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `RegisterChanges` adds `Value` to the list of clients.

See also: TBasicAction.UnregisterChanges (57)

TBasicAction.UnRegisterChanges

Synopsis: Unregisters a client from the list of clients

Declaration: `procedure UnRegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `UnregisterChanges` removes `Value` from the list of clients. This is called for instance when the action is destroyed, or when the client is assigned a new action.

See also: TBasicAction.UnregisterChanges (57), TBasicAction.Destroy (56)

TBasicAction.Update

Synopsis: Triggers the OnUpdate (59) event

Declaration: `function Update : Boolean;  Virtual`

Visibility: public

Description: `Update` triggers the `OnUpdate` event, if one is assigned. It returns `True` if the event was triggered, or `False` if no event was assigned.

Application programmers should never run `Update` directly. The `Update` method is called automatically by the action mechanism; Normally this is in the Idle time of an application. An application programmer should assign the OnUpdate (59) event, and perform any checks in that handler.

See also: TBasicAction.OnUpdate (59), TBasicAction.Execute (57), TBAsicAction.UpdateTarget (56)

TBasicAction.OnChange

Synopsis: Occurs when one of the action's properties changes.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: protected

Access: Read,Write

Description: `OnChange` is the event that is triggered when one of the action's properties changes. This event should be used by client controls or descendent classes to respond to these changes in the properties of the action.

Application programmers should never use the `OnChange` event directly.

TBasicAction.ActionComponent

Synopsis: Returns the component that initiated the action.

Declaration: `Property ActionComponent : TComponent`

Visibility: public

Access: Read,Write

Description: `ActionComponent` is set to the component that caused the action to execute, e.g. a toolbutton or a menu item. The property is set just before the action executes, and is reset to nil after the action was executed.

See also: TBasicAction.Execute (57), TBasicAction.OnExecute (58)

TBasicAction.OnExecute

Synopsis: Event triggered when the action executes.

Declaration: `Property OnExecute : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnExecute` is the event triggered when the action is activated (executed). The event is triggered e.g. when the user clicks e.g. on a menu item or a button associated to the action. The application programmer should provide a `OnExecute` event handler to execute whatever code is necessary when the button is pressed or the menu item is chosen.

Note that assigning an `OnExecute` handler will result in the Execute (57) method returning a `True` value. Predefined actions (such as dataset actions) will check the result of `Execute` and will not perform their normal task if the `OnExecute` handler was called.

See also: TBasicAction.Execute (57), TBasicAction.OnUpdate (59)

TBasicAction.OnUpdate

Synopsis: Event trigged when the application is idle.

Declaration: `Property OnUpdate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnUpdate` is the event triggered when the application is idle, and the action is being updated. The `OnUpdate` event can be used to set the state of the action, for instance disable it if the action cannot be executed at this point in time.

See also: TBasicAction.Update (57), TBasicAction.OnExecute (58)

1.28 TBasicActionLink

Description

`TBasicActionLink` links an Action to its clients. With each client for an action, a `TBasicActionLink` class is instantiated to handle the communication between the action and the client. It passes events between the action and its clients, and thus presents the action with a uniform interface to the clients.

An application programmer should never use a `TBasicActionLink` instance directly; They are created automatically when an action is associated with a component. Component programmers should create specialized descendents of `TBasicActionLink` which communicate changes in the action to the component.

Method overview

Page	Method	Description
59	AssignClient	Assigns a control (client) to the action link.
60	Change	Executed whenever the Action is changed.
61	Create	Creates a new instance of the TBasicActionLink class
61	Destroy	Destroys the TBasicActionLink instance.
61	Execute	Calls the action's Execute method.
60	IsOnExecuteLinked	Returns whether the client has it's OnExecute property linked.
60	SetAction	Sets the action with which the actionlink is associated.
60	SetOnExecute	Assigns the OnExecute (58) handler to the client
62	Update	Calls the action's Update method

Property overview

Page	Property	Access	Description
62	Action	rw	The action to which the link was assigned.
62	OnChange	rw	Event handler triggered when the action's properties change

TBasicActionLink.AssignClient

Synopsis: Assigns a control (client) to the action link.

Declaration: `procedure AssignClient(AClient: TObject);  Virtual`

Visibility: protected

Description: `AssignClient` assigns a control to the actionlink and hence to the action. Descendent classes can override `AssignClient` to check whether the new client is a suitable client for this action.

See also: TBasicActionLink.Action (62)

TBasicActionLink.Change

Synopsis: Executed whenever the Action is changed.

Declaration: `procedure Change;  Virtual`

Visibility: protected

Description: `Change` is executed whenever the action changes. It executes the OnChange (62) handler, if one is assigned.

Component programmers may decide to override the `Change` procedure in descendent classes to perform aditional actions when the properties of the action changes.

See also: TBasicActionLink.OnChange (62), TBasicAction.Change (55)

TBasicActionLink.IsOnExecuteLinked

Synopsis: Returns whether the client has it's OnExecute property linked.

Declaration: `function IsOnExecuteLinked : Boolean;  Virtual`

Visibility: protected

Description: `IsOnExecuteLinked` always returns true in `TBasicActionLink`. Descendent classes can override this method to provide a different result.

TBasicActionLink.SetAction

Synopsis: Sets the action with which the actionlink is associated.

Declaration: `procedure SetAction(Value: TBasicAction);  Virtual`

Visibility: protected

Description: `SetAction` is the write handler for the Action (62) property. It sets the `Action` property to it's new value, after unregistering itself with the old action, if there was one.

See also: TBasicActionLink.Action (62), TBasicAction (54)

TBasicActionLink.SetOnExecute

Synopsis: Assigns the OnExecute (58) handler to the client

Declaration: `procedure SetOnExecute(Value: TNotifyEvent);  Virtual`

Visibility: protected

Description: `SetOnExecute` must be overridden by descendent classes to pass the `OnExecute` handler of the associated action to the client control. It will attach the `OnExecute` handler to whatever handler is appropriate for the client control.

See also: TBasicAction.OnExecute (58), TBasicAction (54)

TBasicActionLink.Create

Synopsis: Creates a new instance of the TBasicActionLink class

Declaration: `constructor Create(AClient: TObject);  Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TBasicActionLink` and assigns `AClient` as the client of the link.

Application programmers should never instantiate `TBasicActionLink` classes directly. An instance is created automatically when an action is assigned to a control (client).

Component programmers can override the create constructor to initialize further properties.

See also: TBasicActionLink.Destroy (61)

TBasicActionLink.Destroy

Synopsis: Destroys the TBasicActionLink instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` unregisters the `TBasicActionLink` with the action, and then calls the inherited destructor.

Application programmers should never call `Destroy` directly. If a link should be destroyed at all, the `Free` method should be called instead.

See also: TBasicActionLink.Create (61)

TBasicActionLink.Execute

Synopsis: Calls the action's Execute method.

Declaration: `function Execute(AComponent: TComponent) : Boolean;  Virtual`

Visibility: public

Description: `Execute` sets the ActionComponent (58) property of the associated Action (62) to `AComponent` and then calls the Action's execute (57) method. After the action has executed, the `ActionComponent` property is cleared again.

The return value of the function is the return value of the Action's execute method.

Application programmers should never call `Execute` directly. This method will be called automatically when the associated control is activated. (e.g. a button is clicked on)

Component programmers should call `Execute` whenever the action should be activated.

See also: TBasicActionLink.Action (62), TBasicAction.ActionComponent (58), TBasicAction.Execute (57), TBasicAction.onExecute (58)

TBasicActionLink.Update

Synopsis: Calls the action's Update method

Declaration: `function Update : Boolean;  Virtual`

Visibility: public

Description: `Update` calls the associated Action's Update (57) methoda.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

TBasicActionLink.Action

Synopsis: The action to which the link was assigned.

Declaration: `Property Action : TBasicAction`

Visibility: public

Access: Read,Write

Description: `Action` represents the Action (54) which was assigned to the client. Setting this property will unregister the client at the old action (if one existed) and registers the client at the new action.

See also: TBasicAction (54)

TBasicActionLink.OnChange

Synopsis: Event handler triggered when the action's properties change

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` is the event triggered when the action's properties change.

Application programmers should never need to assign this event. Component programmers can assign this event to have a client control reflect any changes in an Action's properties.

See also: TBasicActionLink.Change (60), TBasicAction.Change (55)

1.29 TBinaryObjectReader

Description

The `TBinaryObjectReader` class reads component data stored in binary form in a file. For this, it overrides or implements all abstract methods from TAbstractObjectReader (44). No new functionality is added by this class, it is a driver class for the streaming system.

Method overview

Page	Method	Description
64	BeginComponent	
64	BeginProperty	
64	BeginRootComponent	
63	Create	Creates a new binary data reader instance.
63	Destroy	Destroys the binary data reader.
64	NextValue	
64	ReadBinary	
65	ReadDate	
64	ReadFloat	
65	ReadIdent	
65	ReadInt16	
65	ReadInt32	
65	ReadInt64	
65	ReadInt8	
65	ReadSet	
64	ReadSingle	
65	ReadStr	
65	ReadString	
64	ReadValue	
66	SkipComponent	
66	SkipValue	

TBinaryObjectReader.Create

Synopsis: Creates a new binary data reader instance.

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: `Create` instantiates a new binary component data reader. The `Stream` stream is the stream from which data will be read. The `BufSize` argument is the size of the internal buffer that will be used by the reader. This can be used to optimize the reading process.

See also: TAbstractObjectReader (44)

TBinaryObjectReader.Destroy

Synopsis: Destroys the binary data reader.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` frees the buffer allocated when the instance was created. It also positions the stream on the last used position in the stream (the buffering may cause the reader to read more bytes than were actually used.)

See also: TBinaryObjectReader.Create (63)

TBinaryObjectReader.NextValue

Declaration: `function NextValue : TValueType;  Override`

Visibility: public

TBinaryObjectReader.ReadValue

Declaration: `function ReadValue : TValueType;  Override`

Visibility: public

TBinaryObjectReader.BeginRootComponent

Declaration: `procedure BeginRootComponent;  Override`

Visibility: public

TBinaryObjectReader.BeginComponent

Declaration:
```
procedure BeginComponent(var Flags: TFilerFlags;var AChildPos: Integer;
                        var CompClassName: String;var CompName: String)
                        ;  Override
```

Visibility: public

TBinaryObjectReader.BeginProperty

Declaration: `function BeginProperty : String;  Override`

Visibility: public

TBinaryObjectReader.ReadBinary

Declaration: `procedure ReadBinary(const DestData: TMemoryStream);  Override`

Visibility: public

TBinaryObjectReader.ReadFloat

Declaration: `function ReadFloat : Extended;  Override`

Visibility: public

TBinaryObjectReader.ReadSingle

Declaration: `function ReadSingle : Single;  Override`

Visibility: public

TBinaryObjectReader.ReadDate

Declaration: `function ReadDate : TDateTime;  Override`

Visibility: public

TBinaryObjectReader.ReadIdent

Declaration: `function ReadIdent(ValueType: TValueType) : String;  Override`

Visibility: public

TBinaryObjectReader.ReadInt8

Declaration: `function ReadInt8 : ShortInt;  Override`

Visibility: public

TBinaryObjectReader.ReadInt16

Declaration: `function ReadInt16 : SmallInt;  Override`

Visibility: public

TBinaryObjectReader.ReadInt32

Declaration: `function ReadInt32 : LongInt;  Override`

Visibility: public

TBinaryObjectReader.ReadInt64

Declaration: `function ReadInt64 : Int64;  Override`

Visibility: public

TBinaryObjectReader.ReadSet

Declaration: `function ReadSet(EnumType: Pointer) : Integer;  Override`

Visibility: public

TBinaryObjectReader.ReadStr

Declaration: `function ReadStr : String;  Override`

Visibility: public

TBinaryObjectReader.ReadString

Declaration: `function ReadString(StringType: TValueType) : String;  Override`

Visibility: public

TBinaryObjectReader.SkipComponent

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean);  Override`

Visibility: public

TBinaryObjectReader.SkipValue

Declaration: `procedure SkipValue;  Override`

Visibility: public

1.30 TBinaryObjectWriter

Description

Driver class which stores component data in binary form.

Method overview

Page	Method	Description
67	BeginCollection	Start writing a collection.
67	BeginComponent	Start writing a component
67	BeginList	Start writing a list.
67	BeginProperty	Start writing a property
66	Create	Creates a new instance of a binary object writer.
66	Destroy	Destroys an instance of the binary object writer.
67	EndList	Mark the end of a list.
67	EndProperty	Marks the end of writing of a property.
68	WriteBinary	Writes binary data to the stream.
68	WriteBoolean	Writes a boolean value to the stream.
68	WriteDate	Writes a date type to the stream.
68	WriteFloat	Writes a float value to the stream.
68	WriteIdent	Writes an identifier to the stream.
68	WriteInteger	Writes an integer value to the stream.
69	WriteMethodName	Writes a methodname to the stream.
69	WriteSet	Writes a set value to the stream.
68	WriteSingle	Writes a single-type real value to the stream.
69	WriteString	Writes a string value to the stream.

TBinaryObjectWriter.Create

Synopsis: Creates a new instance of a binary object writer.

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: Creates a new instance of a binary object writer.

TBinaryObjectWriter.Destroy

Synopsis: Destroys an instance of the binary object writer.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys an instance of the binary object writer.

TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection;  Override`

Visibility: public

TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration:
```
procedure BeginComponent(Component: TComponent;Flags: TFilerFlags;
                               ChildPos: Integer);  Override
```

Visibility: public

TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList;  Override`

Visibility: public

TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList;  Override`

Visibility: public

TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: String);  Override`

Visibility: public

TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty;  Override`

Visibility: public

TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer;Count: LongInt);  Override`

Visibility: public

TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean);  Override`

Visibility: public

TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended);  Override`

Visibility: public

TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single);  Override`

Visibility: public

TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime);  Override`

Visibility: public

TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String);  Override`

Visibility: public

TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64);  Override`

Visibility: public

TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String);  Override`

Visibility: public

TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt;SetType: Pointer);  Override`

Visibility: public

TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String);  Override`

Visibility: public

1.31 TBits

Description

`TBits` can be used to store collections of bits in an indexed array. This is especially useful for storing collections of booleans: Normally the size of a boolean is the size of the smallest enumerated type, i.e. 1 byte. Since a bit can take 2 values it can be used to store a boolean as well. Since TBits can store 8 bits in a byte, it takes 8 times less space to store an array of booleans in a TBits class then it would take to stoe them in a conventional array.

`TBits` introduces methods to store and retrieve bit values, apply masks, and search for bits.

Method overview

Page	Method	Description
72	AndBits	Performs an `and` operation on the bits.
71	Clear	Clears a particular bit.
71	Clearall	Clears all bits in the array.
70	Create	Creates a new bits collection.
70	Destroy	Destroys a bit collection
73	Equals	Determines whether the bits of 2 arrays are equal.
74	FindFirstBit	Find first bit with a particular value
74	FindNextBit	Searches the next bit with a particular value.
75	FindPrevBit	Searches the previous bit with a particular value.
73	Get	Retrieve the value of a particular bit
71	GetFSize	Returns the number of records used to store the bits.
73	Grow	Expands the bits array to the requested size.
73	NotBits	Performs a `not` operation on the bits.
75	OpenBit	Returns the position of the first bit that is set to `False`.
72	OrBits	Performs an `or` operation on the bits.
74	SetIndex	Sets the start position for FindNextBit (74) and FindPrevBit (75)
71	SetOn	Turn a particular bit on.
72	XorBits	Performs a `xor` operation on the bits.

Property overview

Page	Property	Access	Description
75	Bits	rw	Access to all bits in the array.
76	Size	rw	Current size of the array of bits.

TBits.Create

Synopsis: Creates a new bits collection.

Declaration: `constructor Create(TheSize: LongInt);  Virtual`

Visibility: public

Description: `Create` creates a new bit collection with initial size `TheSize`. The size of the collection can be changed later on.

All bits are initially set to zero.

See also: TBits.Destroy (70)

TBits.Destroy

Synopsis: Destroys a bit collection

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys a previously created bit collection and releases all memory used to store the bit collection.

`Destroy` should never be called directly, `Free` should be used instead.

Errors: None.

See also: TBits.Create (70)

TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : LongInt`

Visibility: public

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: TBits.Size (76)

TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: LongInt)`

Visibility: public

Description: `SetOn` turns on the bit at position `bit`, i.e. sets it to 1. If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (73).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (41) exception is raised.

See also: TBits.Bits (75), TBits.clear (71)

TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: LongInt)`

Visibility: public

Description: `Clear` clears the bit at position `bit`. If the array If `bit` is at a position bigger than the current size, the collection is expanded to the required size using Grow (73).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an EBitsError (41) exception is raised.

See also: TBits.Bits (75), TBits.clear (71)

TBits.Clearall

Synopsis: Clears all bits in the array.

Declaration: `procedure Clearall`

Visibility: public

Description: `ClearAll` clears all bits in the array, i.e. sets them to zero. `ClearAll` works faster than clearing all individual bits, since it uses the packed nature of the bits.

Errors: None.

See also: TBits.Bits (75), TBits.clear (71)

TBits.AndBits

Synopsis: Performs an `and` operation on the bits.

Declaration: `procedure AndBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `and` operation on the bits in the array with the bits of array `BitSet`. If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are cleared.

Errors: None.

See also: TBits.clearall (71), TBits.orbits (72), TBits.xorbits (72), TBits.notbits (73)

TBits.OrBits

Synopsis: Performs an `or` operation on the bits.

Declaration: `procedure OrBits(BitSet: TBits)`

Visibility: public

Description: `andbits` performs an `or` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `or` operation is performed.

Errors: None.

See also: TBits.clearall (71), TBits.andbits (72), TBits.xorbits (72), TBits.notbits (73)

TBits.XorBits

Synopsis: Performs a `xor` operation on the bits.

Declaration: `procedure XorBits(BitSet: TBits)`

Visibility: public

Description: `XorBits` performs a `xor` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `xor` operation is performed.

Errors: None.

See also: TBits.clearall (71), TBits.andbits (72), TBits.orbits (72), TBits.notbits (73)

TBits.NotBits

Synopsis: Performs a `not` operation on the bits.

Declaration: `procedure NotBits(BitSet: TBits)`

Visibility: public

Description: `NotBits` performs a `not` operation on the bits in the array with the bits of array `Bitset`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

Errors: None.

See also: TBits.clearall (71), TBits.andbits (72), TBits.orbits (72), TBits.xorbits (72)

TBits.Get

Synopsis: Retrieve the value of a particular bit

Declaration: `function Get(Bit: LongInt) : Boolean`

Visibility: public

Description: `Get` returns `True` if the bit at position `bit` is set, or `False` if it is not set.

Errors: If `bit` is not a valid bit index then an EBitsError (41) exception is raised.

See also: TBits.Bits (75), TBits.FindFirstBit (74), TBits.seton (71)

TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration: `procedure Grow(NBit: LongInt)`

Visibility: public

Description: `Grow` expands the bit array so it can at least contain `nbit` bits. If `nbit` is less than the current size, nothing happens.

Errors: If there is not enough memory to complete the operation, then an EBitsError (41)is raised.

See also: TBits.Size (76)

TBits.Equals

Synopsis: Determines whether the bits of 2 arrays are equal.

Declaration: `function Equals(BitSet: TBits) : Boolean`

Visibility: public

Description: `equals` returns `True` if all the bits in `BitSet` are the same as the ones in the current BitSet; if not, `False` is returned.

If the sizes of the two BitSets are different, the arrays are still reported equal when all the bits in the larger set, which are not present in the smaller set, are zero.

Errors: None.

See also: TBits.clearall (71), TBits.andbits (72), TBits.orbits (72), TBits.xorbits (72)

TBits.SetIndex

Synopsis: Sets the start position for FindNextBit (74) and FindPrevBit (75)

Declaration: `procedure SetIndex(Index: LongInt)`

Visibility: public

Description: `SetIndex` sets the search start position forFindNextBit (74) and FindPrevBit (75) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to FindFirstBit (74) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: TBits.FindNextBit (74), TBits.FindPrevBit (75), TBits.FindFirstBit (74), TBits.OpenBit (75)

TBits.FindFirstBit

Synopsis: Find first bit with a particular value

Declaration: `function FindFirstBit(State: Boolean) : LongInt`

Visibility: public

Description: `FindFirstBit` searches for the first bit with value `State`. It returns the position of this bit, or `-1` if no such bit was found.

The search starts at position 0 in the array. If the first search returned a positive result, the found position is saved, and the FindNextBit (74) and FindPrevBit (75) will use this position to resume the search. To start a search from a certain position, the start position can be set with the SetIndex (74) instead.

Errors: None.

See also: TBits.FindNextBit (74), TBits.FindPrevBit (75), TBits.OpenBit (75), TBits.SetIndex (74)

TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : LongInt`

Visibility: public

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the FindFirstBit (74). The search is done towards the end of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (74).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: TBits.FindFirstBit (74), TBits.FindPrevBit (75), TBits.OpenBit (75), TBits.SetIndex (74)

TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : LongInt`

Visibility: public

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the FindFirstBit (74). The search is done towards the beginning of the array and starts at the position last reported by one of the `Find` calls or at the position set with SetIndex (74).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: TBits.FindFirstBit (74), TBits.FindNextBit (74), TBits.OpenBit (75), TBits.SetIndex (74)

TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : LongInt`

Visibility: public

Description: `OpenBit` returns the position of the first bit whose value is `0` (`False`), or `-1` if no open bit was found. This call is equivalent to `FindFirstBit(False)`, except that it doesn't set the position for the next searches.

Errors: None.

See also: TBits.FindFirstBit (74), TBits.FindPrevBit (75), TBits.FindFirstBit (74), TBits.SetIndex (74)

TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: LongInt]: Boolean; default`

Visibility: public

Access: Read,Write

Description: `Bits` allows indexed access to all of the bits in the array. It gives `True` if the bit is `1`, `False` otherwise; Assigning to this property will set, respectively clear the bit.

Errors: If an index is specified which is out of the allowed range then an EBitsError (41) exception is raised.

See also: TBits.Size (76)

TBits.Size

Synopsis: Current size of the array of bits.

Declaration: `Property Size : LongInt`

Visibility: public

Access: Read,Write

Description: `Size` is the current size of the bit array. Setting this property will adjust the size; this is equivalent to calling `Grow(Value-1)`

Errors: If an invalid size (negative or too large) is specified, a EBitsError (41) exception is raised.

See also: TBits.Bits (75)

1.32 TCollection

Description

`TCollection` implements functionality to manage a collection of named objects. Each of these objects needs to be a descendent of the TCollectionItem (83) class. Exactly which type of object is managed can be seen from the TCollection.ItemClass (82) property.

Normally, no `TCollection` is created directly. Instead, a descendents of `TCollection` and TCollectionItem (83) are created as a pair.

Method overview

Page	Method	Description
80	Add	Creates and adds a new item to the collection.
80	Assign	Assigns one collection to another.
80	BeginUpdate	Start an update batch.
78	Changed	Procedure called if an item is added to or removed from the collection.
81	Clear	Removes all items from the collection.
79	Create	Creates a new collection.
80	Destroy	Destroys the collection and frees all the objects it manages.
81	EndUpdate	Ends an update batch.
81	FindItemID	Searches for an Item in the collection, based on its TCollectionItem.ID (86) property.
77	GetAttr	Returns an attribute of the collection.
77	GetAttrCount	Returns the count of attributes associated with each item.
78	GetItem	Read handler for the TCollection.Items (82) property.
77	GetItemAttr	Returns the attributes of an item.
77	GetNamePath	Overrides TPersistent.GetNamePath (127) to return a proper pathname.
78	SetItem	Write handler for the TCollection.Items (82) property.
79	SetItemName	Virtual method to set the name of the specified item
79	SetPropName	Write handler for the TCollection.PropName (82) property
79	Update	Hander called when an item in the collection has changed.

Property overview

Page	Property	Access	Description
82	Count	r	Number of items in the collection.
82	ItemClass	r	Class pointer for each item in the collection.
82	Items	rw	Indexed array of items in the collection.
82	PropName	rw	Name of the property that this collection represents.

TCollection.GetAttrCount

Synopsis: Returns the count of attributes associated with each item.

Declaration: `function GetAttrCount : Integer;  Dynamic`

Visibility: protected

Description: `GetAttrCount` returns 0 in the `TCollection` implementation. It can be used to determine the number of attributes associated with each collection item. Descendent objects should override this method to return the number of attributes.

This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: TCollection.GetAttr (77), TCollection.GetItemAttr (77)

TCollection.GetAttr

Synopsis: Returns an attribute of the collection.

Declaration: `function GetAttr(Index: Integer) : String;  Dynamic`

Visibility: protected

Description: This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: TCollection.GetAttrCount (77), TCollection.GetItemAttr (77)

TCollection.GetItemAttr

Synopsis: Returns the attributes of an item.

Declaration:
```
function GetItemAttr(Index: Integer;ItemIndex: Integer) : String
                    ;  Dynamic
```

Visibility: protected

Description: This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: TCollection.GetAttr (77), TCollection.GetAttrCount (77)

TCollection.GetNamePath

Synopsis: Overrides TPersistent.GetNamePath (127) to return a proper pathname.

Declaration: `function GetNamePath : String;  Override`

Visibility: protected

Description: `GetNamePath` returns the name path for this collection. If the following conditions are satisfied:

1.There is an owner object.

2.The owner object returns a non-empty name path.

3.The TCollection.Propname (82) property is not empty

collection has an owner and the owning object has a name, then the function returns the owner name, followed by the propname. If one of the conditions is not satisfied, then the classname is returned.

See also: TCollection.GetOwner (76), TCollection.Propname (82)

TCollection.Changed

Synopsis: Procedure called if an item is added to or removed from the collection.

Declaration: `procedure Changed`

Visibility: protected

Description: `Changed` is called if a change takes place in the collection managed by the class. If the update count ghas reached zero, it calls TCollection.Update (79) with a nil argument.

See also: TCollection.Update (79), TCollection.Add (80), TCollection.Clear (81)

TCollection.GetItem

Synopsis: Read handler for the TCollection.Items (82) property.

Declaration: `function GetItem(Index: Integer) : TCollectionItem`

Visibility: protected

Description: `GetItem` is the read handler for the TCollection.Items (82) property. It returns the `Index`-th element from the list of objects.

Errors: If `Index` is outside the allowed range, then an EListError (42) exception is raised.

See also: TCollection.Items (82), TCollection.Count (82), TCollection.SetItem (78)

TCollection.SetItem

Synopsis: Write handler for the TCollection.Items (82) property.

Declaration: `procedure SetItem(Index: Integer;Value: TCollectionItem)`

Visibility: protected

Description: `SetItem` implements the write handler for the TCollection.Items (82) property. It assignes `Value` to the `Index`-th element in the array. For this to work properly, the TPersistent.Assign (127) method of the `Item` must work correctly.

Errors: If `Index` is outside the allowed range, then an EListError (42) exception is raised.

See also: TCollection.Items (82), TCollection.Count (82), TCollection.GetItem (78)

TCollection.SetItemName

Synopsis: Virtual method to set the name of the specified item

Declaration: `procedure SetItemName(Item: TCollectionItem);  Virtual`

Visibility: protected

Description: Virtual method to set the name of the specified item

TCollection.SetPropName

Synopsis: Write handler for the TCollection.PropName (82) property

Declaration: `procedure SetPropName;  Virtual`

Visibility: protected

Description: `SetPropName` must be overridden by descendent objects to set the TCollection.PropName (82) property to a suitable value. By default, `SetPropName` sets the `PropName` property to empty.

See also: TCollection.PropName (82)

TCollection.Update

Synopsis: Hander called when an item in the collection has changed.

Declaration: `procedure Update(Item: TCollectionItem);  Virtual`

Visibility: protected

Description: `Update` is called in the following cases:

1. An item is added to or removed from the collection.
2. An item is moved in the list, i.e. its TCollectionItem.Index (86) property changes.
3. An item's TCollectionItem.DisplayName (87) property changes.

Descendent classes can override this method to perform additional actions when the collection changes. The `Item` parameter indicates the item that was changed. This can be `Nil`

See also: TCollection.Changed (78)

TCollection.Create

Synopsis: Creates a new collection.

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: public

Description: `Create` instantiates a new instance of the `TCollection` class which will manage objects of class `AItemClass`. It creates the list used to hold all objects, and stores the `AItemClass` for the adding of new objects to the collection.

See also: TCollection.ItemClass (82), TCollection.Destroy (80)

TCollection.Destroy

Synopsis: Destroys the collection and frees all the objects it manages.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` first clears the collection, and then frees all memory allocated to this instance.

Don't call `Destroy` directly, call `Free` instead.

See also: TCollection.Create (79)

TCollection.Add

Synopsis: Creates and adds a new item to the collection.

Declaration: `function Add : TCollectionItem`

Visibility: public

Description: `Add` instantiates a new item of class TCollection.ItemClass (82) and adds it to the list. The newly created object is returned.

See also: TCollection.ItemClass (82), TCollection.Clear (81)

TCollection.Assign

Synopsis: Assigns one collection to another.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` assigns the contents of one collection to another. It does this by clearing the items list, and adding as much elements as there are in the `Source` collection; it assigns to each created element the contents of it's counterpart in the `Source` element.

Two collections cannot be assigned to each other if instances of the `ItemClass` classes cannot be assigned to each other.

Errors: If the objects in the collections cannot be assigned to one another, then an `EConvertError` is raised.

See also: TPersistent.Assign (127), TCollectionItem (83)

TCollection.BeginUpdate

Synopsis: Start an update batch.

Declaration: `procedure BeginUpdate`

Visibility: public

Description: `BeginUpdate` is called at the beginning of a batch update. It raises the update count with 1.

Call `BeginUpdate` at the beginning of a series of operations that will change the state of the collection. This will avoid the call to TCollection.Update (79) for each operation. At the end of the operations, a corresponding call to `EndUpdate` must be made. It is best to do this in the context of a `Try ... finally` block:

```
With MyCollection Do
    try
      BeginUpdate;
      // Some Lengthy operations
    finally
      EndUpdate;
    end;
```

This insures that the number of calls to `BeginUpdate` always matches the number of calls to TCollection.EndUpdate (81), even in case of an exception.

See also: TCollection.EndUpdate (81), TCollection.Changed (78), TCollection.Update (79)

TCollection.Clear

Synopsis: Removes all items from the collection.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` will clear the collection, i.e. each item in the collection is destroyed and removed from memory. After a call to `Clear`, `Count` is zero.

See also: TCollection.Add (80), TCollectionItem.Destroy (85), TCollection.Destroy (80)

TCollection.EndUpdate

Synopsis: Ends an update batch.

Declaration: `procedure EndUpdate`

Visibility: public

Description: `EndUpdate` signals the end of a series of operations that change the state of the collection, possibly triggering an update event. It does this by decreasing the update count with 1 and calling TCollection.Changed (78) it should always be used in conjunction with TCollection.BeginUpdate (80), preferably in the `Finally` section of a `Try ... Finally` block.

See also: TCollection.BeginUpdate (80), TCollection.Changed (78), TCollection.Update (79)

TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its TCollectionItem.ID (86) property.

Declaration: `function FindItemID(ID: Integer) : TCollectionItem`

Visibility: public

Description: `FindItemID` searches through the collection for the item that has a value of `ID` for its TCollectionItem.ID (86) property, and returns the found item. If no such item is found in the collection, `Nil` is returned.

The routine performs a linear search, so this can be slow on very large collections.

See also: TCollection.Items (82), TCollectionItem.ID (86)

TCollection.PropName

Synopsis: Name of the property that this collection represents.

Declaration: `Property PropName : String`

Visibility: protected

Access: Read,Write

Description: `PropName` indicates the name of the property that this collection is supposed to represent. By default, this is the empty string. Descendents can override this property to return the name of the property that is represented by this collection.

See also: TCollection.SetPropName (79), TCollection.GetPropName (76)

TCollection.Count

Synopsis: Number of items in the collection.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` contains the number of items in the collection.

Remark: The items in the collection are identified by their TCollectionItem.Index (86) property, which is a zero-based index, meaning that it can take values between `0` and `Count`.

See also: TCollectionItem.Index (86), TCollection.Items (82)

TCollection.ItemClass

Synopsis: Class pointer for each item in the collection.

Declaration: `Property ItemClass : TCollectionItemClass`

Visibility: public

Access: Read

Description: `ItemClass` is the class pointer with which each new item in the collection is created. It is the value that was passed to the collection's constructor when it was created, and does not change during the lifetime of the collection.

See also: TCollectionItem (83), TCollection.Items (82)

TCollection.Items

Synopsis: Indexed array of items in the collection.

Declaration: `Property Items[Index: Integer]: TCollectionItem`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the collection. Since the array is zero-based, `Index` should be an integer between `0` and `Count-1`.

It is possible to set or retrieve an element in the array. When setting an element of the array, the object that is assigned should be compatible with the class of the objects in the collection, as given by the TCollection.ItemClass (82) property.

Adding an element to the array can be done with the TCollection.Add (80) method. The array can be cleared with the TCollection.Clear (81) method. Removing an element of the array should be done by freeing that element.

See also: TCollection.Count (82), TCollection.ItemClass (82), TCollection.Clear (81), TCollection.Add (80)

1.33 TCollectionItem

Description

`TCollectionItem` and TCollection (76) form a pair of base classes that manage a collection of named objects. The `TCollectionItem` is the named object that is managed, it represents one item in the collection. An item in the collection is represented by two properties: TCollectionItem.DisplayName (87), TCollection.Index (76) and TCollectionItem.ID (86).

A `TCollectionItem` object is never created directly. To manage a set of named items, it is necessary to make a descendent of `TCollectionItem` to which needed properties and methods are added. This descendant can then be managed with a TCollection (76) class. The managing collection will create and destroy it's items by itself, it should therefore never be necessary to create `TCollectionItem` descendents manually.

Method overview

Page	Method	Description
83	Changed	Method to notify the managing collection that the name or index of this item has changed.
85	Create	Creates a new instance of this collection item.
85	Destroy	Destroyes this collection item.
84	GetDisplayName	Returns the TCollectionItem.DisplayName (87) of the collectionitem
84	GetNamePath	Returns the namepath of this collection item.
84	GetOwner	Returns the managing collection.
85	SetDisplayName	Write method for the TCollectionItem.DisplayName (87) property
85	SetIndex	Write method for the TCollectionItem.Index (86) property.

Property overview

Page	Property	Access	Description
86	Collection	rw	Pointer to the collection managing this item.
87	DisplayName	rw	Name of the item, displayed in the object inspector.
86	ID	r	Initial index of this item.
86	Index	rw	Index of the item in its managing collection TCollection.Items (82) property.

TCollectionItem.Changed

Synopsis: Method to notify the managing collection that the name or index of this item has changed.

Declaration: `procedure Changed(AllItems: Boolean)`

Visibility: protected

Description: This method is called when the TCollectionItem.DisplayName (87) is set or when the TCollectionItem.Index (86) is changed.

See also: TCollectionItem.Id (86), TCollectionItem.Index (86), TCollection.Update (79)

TCollectionItem.GetNamePath

Synopsis: Returns the namepath of this collection item.

Declaration: `function GetNamePath : String;  Override`

Visibility: protected

Description: `GetNamePath` overrides the TPersistent.GetNamePath (127) method to return the name of the managing collection and appends its TCollectionItem.Index (86) property.

See also: TCollectionItem.Collection (86), TPersistent.GetNamePath (127), TCollectionItem.Index (86)

TCollectionItem.GetOwner

Synopsis: Returns the managing collection.

Declaration: `function GetOwner : TPersistent;  Override`

Visibility: protected

Description: `TCollectionItem` overrides TPersistent.GetOwner (126) to and returns the TCollectionItem.Collection (86) that manages it.

See also: TPersistent.GetOwner (126), TCollectionItem.Collection (86)

TCollectionItem.GetDisplayName

Synopsis: Returns the TCollectionItem.DisplayName (87) of the collectionitem

Declaration: `function GetDisplayName : String;  Virtual`

Visibility: protected

Description: `GetDisplayName` returns the value of the TCollectionItem.DisplayName (87) property. By default, this is the classname of the actual `TCollectionItem` descendant.

Descendants of `TCollectionItem` can and should override this method to return a more meaningful value.

See also: TCollectionItem.DisplayName (87)

TCollectionItem.SetIndex

Synopsis: Write method for the TCollectionItem.Index (86) property.

Declaration: `procedure SetIndex(Value: Integer);  Virtual`

Visibility: protected

Description: `SetIndex` implements the write handler for the TCollectionItem.Index (86) property. It requests the managing collection to move this item to the desired index `value`.

See also: TCollectionItem.Index (86)

TCollectionItem.SetDisplayName

Synopsis: Write method for the TCollectionItem.DisplayName (87) property

Declaration: `procedure SetDisplayName(const Value: String);  Virtual`

Visibility: protected

Description: `SetDisplayName` is the write method for the TCollectionItem.DisplayName (87) property. It does nothing but notifying the managing collection that the displayname has changed. It does NOT store the actual `Value`.

Descendants of `TCollectionItem` should override this method to store the actual displayname if this is required.

See also: TCollectionItem.DisplayName (87)

TCollectionItem.Create

Synopsis: Creates a new instance of this collection item.

Declaration: `constructor Create(ACollection: TCollection);  Virtual`

Visibility: public

Description: `Create` instantiates a new item in a TCollection (76). It is called by the TCollection.Add (80) function and should under normal circumstances never be called directly. called

See also: TCollectionItem.Destroy (85)

TCollectionItem.Destroy

Synopsis: Destroyes this collection item.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` removes the item from the managing collection and Destroys the item instance.

This is the only way to remove items from a collection;

See also: TCollectionItem.Create (85)

TCollectionItem.Collection

Synopsis: Pointer to the collection managing this item.

Declaration: `Property Collection : TCollection`

Visibility: public

Access: Read,Write

Description: `Collection` points to the collection managing this item. This property can be set to point to a new collection. If this is done, the old collection will be notified that the item should no longer be managed, and the new collection is notified that it should manage this item as well.

See also: TCollection (76)

TCollectionItem.ID

Synopsis: Initial index of this item.

Declaration: `Property ID : Integer`

Visibility: public

Access: Read

Description: `ID` is the initial value of TCollectionItem.Index (86); it doesn't change after the index changes. It can be used to uniquely identify the item. The `ID` property doesn't change as items are added and removed from the collection.

While the TCollectionItem.Index (86) property forms a continuous series, `ID` does not. If items are removed from the collection, their `ID` is not used again, leaving gaps. Only when the collection is initialiiy created, the `ID` and `Index` properties will be equal.

See also: TCollection.Items (82), TCollectionItem.Index (86)

TCollectionItem.Index

Synopsis: Index of the item in its managing collection TCollection.Items (82) property.

Declaration: `Property Index : Integer`

Visibility: public

Access: Read,Write

Description: `Index` is the current index of the item in its managing collection's TCollection.Items (82) property. This property may change as items are added and removed from the collection.

The index of an item is zero-based, i.e. the first item has index zero. The last item has index `Count-1` where `Count` is the number of items in the collection.

The `Index` property of the items in a collection form a continuous series ranging from `0` to `Count-1`. The TCollectionItem.ID (86) property does not form a continuous series, but can also be used to identify an item.

See also: TCollectionItem.ID (86), TCollection.Items (82)

TCollectionItem.DisplayName

Synopsis: Name of the item, displayed in the object inspector.

Declaration: `Property DisplayName : String`

Visibility: public

Access: Read,Write

Description: `DisplayName` contains the name of this item as shown in the object inspector. For `TCollectionItem` this returns always the class name of the managing collection, followed by the index of the item.

`TCollectionItem` does not implement any functionality to store the `DisplayName` property. The property can be set, but this will have no effect other than that the managing collection is notified of a change. The actual displayname will remain unchanged. To store the `DisplayName` property,`TCollectionItem` descendants should override the TCollectionItem.SetDisplayName (85) and TCollectionItem.GetDisplayName (84) to add storage functionality.

See also: TCollectionItem.Index (86), TCollectionItem.ID (86), TCollectionItem.GetDisplayName (84), TCollectionItem.SetDisplayName (85)

1.34 TComponent

Description

`TComponent` is the base class for any set of classes that needs owner-owned functionality, and which needs support for property streaming. All classes that should be handled by an IDE (Integrated Development Environment) must descend from `TComponent`, as it includes all support for streaming all its published properties.

Components can 'own' other components. `TComponent` introduces methods for enumerating the child components. It also allows to name the owned components with a unique name. Furthermore, functionality for sending notifications when a component is removed from the list or removed from memory alltogether is also introduced in `TComponent`

`TComponent` introduces a form of automatic memory management: When a component is destroyed, all its child components will be destroyed first.

Method overview

Page	Method	Description
89	ChangeName	Actually sets the component name.
95	Create	Creates a new instance of the component.
89	DefineProperties	Defines fake top,left properties for handling in the IDE.
95	Destroy	Destroys the instance of the component.
95	DestroyComponents	Destroy child components.
96	Destroying	Called when the component is being destroyed
96	ExecuteAction	
96	FindComponent	Finds and returns the named component in the owned components.
96	FreeNotification	Ask the component to notify called when it is being destroyed.
97	FreeOnRelease	Part of the `IVCLComObject` interface.
90	GetChildOwner	Returns the owner of any children.
90	GetChildParent	Returns the parent of any children.
89	GetChildren	Must be overridden by descendents to return all child components that must be streamed.
90	GetNamePath	Returns the name path of this component.
90	GetOwner	Returns the owner of this component.
97	GetParentComponent	Returns the parent component.
97	HasParent	Does the component have a parent ?
97	InsertComponent	Insert the given component in the list of owned components.
91	Loaded	Called when the component has finished loading.
91	Notification	Called by components that are freed and which received a FreeNotification.
91	ReadState	Read the component's state from a stream.
98	RemoveComponent	Remove the given component from the list of owned components.
96	RemoveFreeNotification	
98	SafeCallException	Part of the `IVCLComObject` Interface.
92	SetAncestor	Sets the `csAncestor` state of the component.
93	SetChildOrder	Determines the order in which children are streamed/created.
92	SetDesigning	Sets the `csDesigning` state of the component.
92	SetName	Write handler for Name (100) property.
93	SetParentComponent	Set the parent component.
98	UpdateAction	
93	Updated	Ends the `csUpdating` state.
94	UpdateRegistry	For compatibilty only.
93	Updating	Sets the state to `csUpdating`
94	ValidateContainer	??
94	ValidateInsert	Called when an insert must be validated.
94	ValidateRename	Called when a name change must be validated
95	WriteState	Writes the component to a stream.

Property overview

Page	Property	Access	Description
98	ComponentCount	r	Count of owned components
99	ComponentIndex	rw	Index of component in it's owner's list.
98	Components	r	Indexed list (zero-based) of all owned components.
99	ComponentState	r	Current component's state.
99	ComponentStyle	r	Current component's style.
100	DesignInfo	rw	Information for IDE designer.
100	Name	rws	Name of the component.
100	Owner	r	Owner of this component.
101	Tag	rw	Tag value of the component.
100	VCLComObject	rw	Not implemented.

TComponent.ChangeName

Synopsis: Actually sets the component name.

Declaration: `procedure ChangeName(const NewName: TComponentName)`

Visibility: protected

Description: `ChangeName` is called by the SetName (92) procedure when the component name is set and the name has been verified. It actually sets the name of the component to `NewName`, and can be used to bypass the name checks which are done when the Name (100) property is set.

Application programmers should never use `SetName` directly.

See also: TComponent.SetName (92), TComponent.Name (100)

TComponent.DefineProperties

Synopsis: Defines fake top,left properties for handling in the IDE.

Declaration: `procedure DefineProperties(Filer: TFiler);  Override`

Visibility: protected

Description: `DefineProperties` overrides the standard TPersistent.DefineProperties (126) to store the top/left properties used to display an icon for a non-visual component in an IDE.

See also: TPersistent.DefineProperties (126)

TComponent.GetChildren

Synopsis: Must be overridden by descendents to return all child components that must be streamed.

Declaration: `procedure GetChildren(Proc: TGetChildProc;Root: TComponent);  Dynamic`

Visibility: protected

Description: `GetChildren` is called by the streaming system to determine which child components should be streamed as well when the component is being streamed. By default, no child components are streamed, i.e. the `TComponent` implementation is empty.

`TComponent` descendents should override this method. For each child that needs to be streamed, `Proc` should be called with as an argument the child component that must be streamed. The `Root` argument contains the root component relative to which all streaming is done.

See also: TComponent.WriteState (95)

TComponent.GetChildOwner

Synopsis: Returns the owner of any children.

Declaration: `function GetChildOwner : TComponent;  Dynamic`

Visibility: protected

Description: `GetChildOwner` returns the owner of the children that are read from the stream. If the method returns `Nil` (the default) this means that streamed child components are owned by the root component of the streaming process (usually a Form or Datamodule)

Application programmers should not call `GetChildOwner` directly, it is called by the streaming system when needed.

See also: TComponent.WriteState (95), TComponent.ReadState (91), TComponent.Owner (100), TComponent.GetChildParent (90)

TComponent.GetChildParent

Synopsis: Returns the parent of any children.

Declaration: `function GetChildParent : TComponent;  Dynamic`

Visibility: protected

Description: `GetChildParent` returns the parent component of the child components being streamed. The parent property is a visual property, which is not always meaningful. If there is no parent component, the owner of child components that are streamed is returned. If `Nil` is returned, then the root component of the streaming operation is assumed. The `TComponent` implementation of this method returns `Self`.

Application programmers should not call this method, it is called automatically by the streaming mechanism.

See also: TComponent.GetChildOwner (90)

TComponent.GetNamePath

Synopsis: Returns the name path of this component.

Declaration: `function GetNamePath : String;  Override`

Visibility: protected

Description: `GetNamePath` returns the name of the component as it will be shown in the object inspector.

`TComponent` overrides `GetNamePath` so it returns the Name (100) property of the component.

See also: TComponent.Name (100), TPersistent.GetNamePath (127)

TComponent.GetOwner

Synopsis: Returns the owner of this component.

Declaration: `function GetOwner : TPersistent;  Override`

Visibility: protected

Description: `GetOwner` returns the owner of this component as indicated by the Owner (100) property. The `GetOwner` call is introduced in TPersistent (125) and is used by the streaming system to determine the 'owner' of a component.

See also: TPersistent.GetOwner (126), TComponent.Owner (100)

TComponent.Loaded

Synopsis: Called when the component has finished loading.

Declaration: `procedure Loaded;  Virtual`

Visibility: protected

Description: `Loaded` is called by the streaming system when a root component was completely read from a stream and all properties and references to other objects have been resolved by the streaming system. Descendents of `TComponent` should override this method to do some additional processing of properties after all published properties have been set from values obtained from the stream.

Application programmers should never call `Loaded` directly, this is done automatically by the streaming system.

See also: TComponent.ReadState (91), TComponent.ComponentState (99)

TComponent.Notification

Synopsis: Called by components that are freed and which received a FreeNotification.

Declaration:
```
procedure Notification(AComponent: TComponent;Operation: TOperation)
                      ;  Virtual
```

Visibility: protected

Description: `Notification` is called whenever a child component is destroyed, inserted or removed from the list of owned component. Components that were requested to send a notification when they are freed ((with FreeNotification (96)) will also call `Notification` when they are freed.

The `AComponent` parameter specifies which component sends the notification, and `Operation` specifies whether the component is being inserted into or removed from the child component list, or whether it is being destroyed.

Descendents of `TComponent` can use FreeNotification (96) to request notification of the destruction of another object. By overriding the `Notification` method, they can do special processing (typically, set a reference to this component to `Nil`) when this component is destroyed. The `Notification` method is called quite often in the streaming process, so speed should be a consideration when overriding this method.

See also: TOperation (27), TComponent.FreeNotification (96)

TComponent.ReadState

Synopsis: Read the component's state from a stream.

Declaration: `procedure ReadState(Reader: TReader);  Virtual`

Visibility: protected

Description: `ReadState` reads the component's state from a stream through the reader object `reader`. Values for all published properties of the component can be read from the stream. Normally there is no need to call `ReadState` directly. The streaming system calls `ReadState` itself.

The TComponent (87) implementation of `ReadState` simply calls TReader.ReadData (130) Descendent classes can, however, override `ReadState`to provide additional processing of stream data.

See also: TComponent.WriteState (95), TStream.ReadComponent (143), TReader.ReadData (130)

TComponent.SetAncestor

Synopsis: Sets the `csAncestor` state of the component.

Declaration: `procedure SetAncestor(Value: Boolean)`

Visibility: protected

Description: `SetAncestor` includes or excludes the `csAncestor` flag in the ComponentState (99) set property, depending on the boolean `Value`. The flag is set recursively for all owned components as well.

This is normally only done during the streaming system, and should not be called directly by an application programmer.

See also: TComponent.ComponentState (99)

TComponent.SetDesigning

Synopsis: Sets the `csDesigning` state of the component.

Declaration: `procedure SetDesigning(Value: Boolean)`

Visibility: protected

Description: `SetDesigning` includes or excludes the `csDesigning` flag in the ComponentState (99) set property, depending on the boolean `Value`. The flag is set recursively for all owned components as well.

This is normally only done during the streaming system, and should not be called directly by an application programmer.

TComponent.SetName

Synopsis: Write handler for Name (100) property.

Declaration: `procedure SetName(const NewName: TComponentName);  Virtual`

Visibility: protected

Description: `SetName` is the write handler for the Name (100) property. It checks whether the desired name is valid (i.e is a valid identifier) and is unique among the children of the owner component. If either conditions is not satisfied, an exception is raised.

See also: TComponent.Name (100), TComponent.ValidateRename (94)

TComponent.SetChildOrder

Synopsis: Determines the order in which children are streamed/created.

Declaration: `procedure SetChildOrder(Child: TComponent;Order: Integer);  Dynamic`

Visibility: protected

Description: This method does nothing. It can be used to change the order in which child components are streamed and created. This can be used by descendent classes to optimize or correct the order in which child components are streamed.

See also: TComponent.ReadState (91)

TComponent.SetParentComponent

Synopsis: Set the parent component.

Declaration: `procedure SetParentComponent(Value: TComponent);  Dynamic`

Visibility: protected

Description: `SetParentComponent` does nothing, but is called by the streaming system to set the parent component of the current component. This method can be overridden by descendent components to set the parent component of the current component.

See also: TComponent.Owner (100)

TComponent.Updating

Synopsis: Sets the state to `csUpdating`

Declaration: `procedure Updating;  Dynamic`

Visibility: protected

Description: `Updating` includes `csUpdating` in the ComponentState (99) property of the component.

Normally, an application programmer should not call this method directly, it is called automatically by the streaming system.

See also: TComponent.Updated (93), TComponent.ComponentState (99)

TComponent.Updated

Synopsis: Ends the `csUpdating` state.

Declaration: `procedure Updated;  Dynamic`

Visibility: protected

Description: `Updated` excludes `csUpdating` from the ComponentState (99) property of the component.

Normally, an application programmer should not call this method directly, it is called automatically by the streaming system.

See also: TComponent.Updating (93), TComponent.ComponentState (99)

TComponent.UpdateRegistry

Synopsis: For compatibilty only.

Declaration:
```
procedure UpdateRegistry(Register: Boolean;const ClassID: String;
                              const ProgID: String);  Dynamic
```

Visibility: protected

Description: This method does nothing, and is provided for compatibility only.

TComponent.ValidateRename

Synopsis: Called when a name change must be validated

Declaration:
```
procedure ValidateRename(AComponent: TComponent;const CurName: String;
                              const NewName: String);  Virtual
```

Visibility: protected

Description: `ValidateRename` checks whether `NewName` is a valid replacement for `CurName` for component `AComponent`. Two owned components of a component can not have the same name. If a child component with the same name is found, then an exception is raised.

See also: TComponent.SetName (92), TComponent.Name (100)

TComponent.ValidateContainer

Synopsis: ??

Declaration:
```
procedure ValidateContainer(AComponent: TComponent);  Dynamic
```

Visibility: protected

Description: `ValidateContainer` is provided for compatibility only. It doesn't do anything in Free Pascal.

TComponent.ValidateInsert

Synopsis: Called when an insert must be validated.

Declaration:
```
procedure ValidateInsert(AComponent: TComponent);  Dynamic
```

Visibility: protected

Description: `ValidateInsert` should be implemented by descendent components to see whether the `AComponent` component may be inserted in the list of owned components.

This procedure does nothing in the `TComponent` implementation, it should be overridden by descendant components.

See also: TComponent.Insert (87)

TComponent.WriteState

Synopsis: Writes the component to a stream.

Declaration: `procedure WriteState(Writer: TWriter);  Virtual`

Visibility: public

Description: `WriteState` writes the component's current state to a stream through the writer (179) object `writer`. Values for all published properties of the component can be written to the stream. Normally there is no need to call `WriteState` directly. The streaming system calls `WriteState` itself.

The TComponent (87) implementation of `WriteState` simply calls TWriter.WriteData (179). Descendent classes can, however, override `WriteState`to provide additional processing of stream data.

See also: TComponent.ReadState (91), TStream.WriteComponent (144), TWriter.WriteData (179)

TComponent.Create

Synopsis: Creates a new instance of the component.

Declaration: `constructor Create(AOwner: TComponent);  Virtual`

Visibility: public

Description: `Create` creates a new instance of a `TComponent` class. If `AOwner` is not `Nil`, the new component attempts to insert itself in the list of owned components of the owner.

See also: TComponent.Insert (87), TComponent.Owner (100)

TComponent.Destroy

Synopsis: Destroys the instance of the component.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` sends a `opRemove` notification to all components in the free-notification list. After that, all owned components are destroyed by calling DestroyComponents (95) (and hence removed from the list of owned components). When this is done, the component removes itself from its owner's child component list. After that, the parent's destroy method is called.

See also: TComponent.Notification (91), TComponent.Owner (100), TComponent.DestroyComponents (95), TComponent.Components (98)

TComponent.DestroyComponents

Synopsis: Destroy child components.

Declaration: `procedure DestroyComponents`

Visibility: public

Description: `DestroyComponents` calls the destructor of all owned components, till no more components are left in the Components (98) array.

Calling the destructor of an owned component has as the effect that the component will remove itself from the list of owned components, if nothing has disrupted the sequence of destructors.

Errors: If an overridden 'destroy' method does not call it's intherited destructor or raises an exception, it's TComponent.Destroy (95) destructor will not be called, which may result in an endless loop.

See also: TComponent.Destroy (95), TComponent.Components (98)

TComponent.Destroying

Synopsis: Called when the component is being destroyed

Declaration: `procedure Destroying`

Visibility: public

Description: `Destroying` sets the `csDestroying` flag in the component's state (87) property, and does the same for all owned components.

It is not necessary to call `Destroying` directly, the destructor Destroy (95) does this automatically.

See also: TComponent.State (87), TComponent.Destroy (95)

TComponent.ExecuteAction

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean;  Dynamic`

Visibility: public

TComponent.FindComponent

Synopsis: Finds and returns the named component in the owned components.

Declaration: `function FindComponent(const AName: String) : TComponent`

Visibility: public

Description: `FindComponent` searches the component with name `AName` in the list of owned components. If `AName` is empty, then `Nil` is returned.

See also: TComponent.Components (98), TComponent.Name (100)

TComponent.FreeNotification

Synopsis: Ask the component to notify called when it is being destroyed.

Declaration: `procedure FreeNotification(AComponent: TComponent)`

Visibility: public

Description: `FreeNotification` inserts `AComponent` in the freenotification list. When the component is destroyed, the Notification (91) method is called for all components in the freenotification list.

See also: TComponent.Components (98), TComponent.Notification (91)

TComponent.RemoveFreeNotification

Declaration: `procedure RemoveFreeNotification(AComponent: TComponent)`

Visibility: public

TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: public

Description: Provided for Delphi compatibility, but is not yet impltmentedd.

TComponent.GetParentComponent

Synopsis: Returns the parent component.

Declaration: `function GetParentComponent : TComponent;  Dynamic`

Visibility: public

Description: `GetParentComponent` can be implemented to return the parent component of this component. The implementation of this method in `TComponent` always returns `Nil`. Descendent classes must override this method to return the visual parent of the component.

See also: TComponent.HasParent (97), TComponent.Owner (100)

TComponent.HasParent

Synopsis: Does the component have a parent ?

Declaration: `function HasParent : Boolean;  Dynamic`

Visibility: public

Description: `HasParent` can be implemented to return whether the parent of the component exists. The implementation of this method in TComponent always returns `False`, and should be overridden by descendent classes to return `True` when a parent is available. If `HasParent` returns `True`, then GetParentComponent (97) will return the parent component.

See also: TComponent.HasParent (97), TComponent.Owner (100)

TComponent.InsertComponent

Synopsis: Insert the given component in the list of owned components.

Declaration: `procedure InsertComponent(AComponent: TComponent)`

Visibility: public

Description: `InsertComponent` attempts to insert `AComponent` in the list with owned components. It first calls ValidateComponent (87) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling ValidateRename (94). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: TComponent.RemoveComponent (98), TComponent.Insert (87), TComponent.ValidateContainer (94), TComponent.ValidateRename (94), TComponent.Notification (91)

TComponent.RemoveComponent

Synopsis: Remove the given component from the list of owned components.

Declaration: `procedure RemoveComponent(AComponent: TComponent)`

Visibility: public

Description: `RemoveComponent` will send an `opRemove` notification to `AComponent` and will then proceed to remove `AComponent` from the list of owned components.

See also: TComponent.InsertComponent (97), TComponent.Remove (87), TComponent.ValidateRename (94), TComponent.Notification (91)

TComponent.SafeCallException

Synopsis: Part of the `IVCLComObject` Interface.

Declaration:
```
function SafeCallException(ExceptObject: TObject;ExceptAddr: Pointer)
                           : Integer;  Override
```

Visibility: public

Description: Provided for Delphi compatibility, but not implemented.

TComponent.UpdateAction

Declaration: `function UpdateAction(Action: TBasicAction) : Boolean;  Dynamic`

Visibility: public

TComponent.Components

Synopsis: Indexed list (zero-based) of all owned components.

Declaration: `Property Components[Index: Integer]: TComponent`

Visibility: public

Access: Read

Description: `Components` provides indexed access to the list of owned components. `Index` can range from 0 to ComponentCount-1 (98).

See also: TComponent.ComponentCount (98), TComponent.Owner (100)

TComponent.ComponentCount

Synopsis: Count of owned components

Declaration: `Property ComponentCount : Integer`

Visibility: public

Access: Read

Description: `ComponentCount` returns the number of components that the current component owns. It can be used to determine the valid index range in the Component (98) array.

See also: TComponent.Components (98), TComponent.Owner (100)

TComponent.ComponentIndex

Synopsis: Index of component in it's owner's list.

Declaration: `Property ComponentIndex : Integer`

Visibility: public

Access: Read,Write

Description: `ComponentIndex` is the index of the current component in its owner's list of components. If the component has no owner, the value of this property is -1.

See also: TComponent.Components (98), TComponent.ComponentCount (98), TComponent.Owner (100)

TComponent.ComponentState

Synopsis: Current component's state.

Declaration: `Property ComponentState : TComponentState`

Visibility: public

Access: Read

Description: `ComponentState` indicates the current state of the component. It is a set of flags which indicate the various stages in the lifetime of a component. The following values can occur in this set:

Table 1.13: Component states

Flag	Meaning
csLoading	The component is being loaded from stream
csReading	Component properties are being read from stream.
csWriting	Component properties are weing written to stream.
csDestroying	The component or one of it's owners is being destoyed.
csAncestor	The component is being streamed as part of a frame
csUpdating	The component is being updated
csFixups	References to other components are being resolved
csFreeNotification	The component has freenotifications.
csInline	The component is being loaded as part of a frame
csDesignInstance	? not used.

The component state is set by various actions such as reading it from stream, destroying it etc.

See also: TComponent.SetAncestor (92), TComponent.SetDesigning (92), TComponent.SetInline (87), TComponent.SetDesignInstance (87), TComponent.Updating (93), TComponent.Updated (93), TComponent.Loaded (91)

TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: public

Access: Read

Description: Current component's style.

TComponent.DesignInfo

Synopsis: Information for IDE designer.

Declaration: `Property DesignInfo : LongInt`

Visibility: public

Access: Read,Write

Description: `DesignInformation` can be used by an IDE to store design information in the component. It should not be used by an application programmer.

See also: TComponent.Tag (101)

TComponent.Owner

Synopsis: Owner of this component.

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read

Description: `Owner` returns the owner of this component. The owner cannot be set except by explicitly inserting the component in another component's owned components list using that component's InsertComponent (97) method, or by removing the component from it's owner's owned component list using the RemoveComponent (98) method.

See also: TComponent.Components (98), TComponent.InsertComponent (97), TComponent.RemoveComponent (98)

TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: public

Access: Read,Write

Description: `VCLComObject` is not yet implemented in Free Pascal.

TComponent.Name

Synopsis: Name of the component.

Declaration: `Property Name : TComponentName`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the component. This name should be a valid identifier, i.e. must start with a letter, and can contain only letters, numbers and the underscore character. When attempting to set the name of a component, the name will be checked for validity. Furthermore, when a component is owned by another component, the name must be either empty or must be unique among the child component names.

Errors: Attempting to set the name to an invalid value will result in an exception being raised.

See also: TComponent.ValidateRename (94), TComponent.Owner (100)

TComponent.Tag

Synopsis: Tag value of the component.

Declaration: `Property Tag : LongInt`

Visibility: published

Access: Read,Write

Description: `Tag` can be used to store an integer value in the component. This value is streamed together with all other published properties. It can be used for instance to quickly identify a component in an event handler.

See also: TComponent.Name (100)

1.35 TCustomMemoryStream

Description

`TCustomMemoryStream` is the parent class for streams that stored their data in memory. It introduces all needed functions to handle reading from and navigating through the memory, and introduces a Memory (103) property which points to the memory area where the stream data is kept.

The only thing which `TCustomMemoryStream` does not do is obtain memory to store data when writing data or the writing of data. This functionality is implemented in descendent streams such as TMemoryStream (118). The reason for this approach is that this way it is possible to create e.g. read-only descendents of `TCustomMemoryStream` that point to a fixed part in memory which can be read from, but not written to.

Remark: Since `TCustomMemoryStream` is an abstract class, do not create instances of `TMemoryStream` directly. Instead, create instances of descendents such as TMemoryStream (118).

Method overview

Page	Method	Description
102	Read	Reads `Count` bytes from the stream into `buffer`.
103	SaveToFile	Writes the contents of the stream to a file.
102	SaveToStream	Writes the contents of the memory stream to another stream.
102	Seek	Sets a new position in the stream.
101	SetPointer	Sets the internal memory pointer and size of the memory block.

Property overview

Page	Property	Access	Description
103	Memory	r	Pointer to the data kept in the memory stream.

TCustomMemoryStream.SetPointer

Synopsis: Sets the internal memory pointer and size of the memory block.

Declaration: `procedure SetPointer(Ptr: Pointer;ASize: LongInt)`

Visibility: protected

Description: `SetPointer` updates the internal memory pointer and the size of the memory area pointed to.

Descendent memory streams should call this method whenever they set or reset the memory the stream should read from or write to.

See also: TCustomMemoryStream.Memory (103), TStream.Size (148)

TCustomMemoryStream.Read

Synopsis: Reads `Count` bytes from the stream into `buffer`.

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

Visibility: public

Description: `Read` reads `Count` bytes from the stream into the memory pointed to by `buffer`. It returns the number of bytes actually read.

This method overrides the abstract TStream.Read (141) method of TStream (140). It will read as much bytes as are still available in the memory area pointer to by Memory (103). After the bytes are read, the internal stream position is updated.

See also: TCustomMemoryStream.Memory (103), TStream.Read (141)

TCustomMemoryStream.Seek

Synopsis: Sets a new position in the stream.

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

Visibility: public

Description: `Seek` overrides the abstract TStream.Seek (142) method. It simply updates the internal stream position, and returns the new position.

Errors: No checking is done whether the new position is still a valid position, i.e. whether the position is still within the range `0..Size`. Attempting a seek outside the valid memory range of the stream may result in an exception at the next read or write operation.

See also: TStream.Position (148), TStream.Size (148), TCustomMemoryStream.Memory (103)

TCustomMemoryStream.SaveToStream

Synopsis: Writes the contents of the memory stream to another stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: public

Description: `SaveToStream` writes the contents of the memory stream to `Stream`. The content of `Stream` is not cleared first. The current position of the memory stream is not changed by this action.

Remark: This method will work much faster than the use of the TStream.CopyFrom (143) method:

```
Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToStream` writes the contents of the memory as one big block.

Errors: If an error occurs when writing to `Stream` an EStreamError (43) exception will be raised.

See also: TCustomMemoryStream.SaveToFile (103), TStream.CopyFrom (143)

TCustomMemoryStream.SaveToFile

Synopsis: Writes the contents of the stream to a file.

Declaration: `procedure SaveToFile(const FileName: String)`

Visibility: public

Description: `SaveToFile` writes the contents of the stream to a file with name `FileName`. It simply creates a filestream and writes the contents of the memorystream to this file stream using TCustomMemoryStream.SaveToStream (102).

Remark: This method will work much faster than the use of the TStream.CopyFrom (143) method:

```
Stream:=TFileStream.Create(fmCreate,FileName);
  Seek(0,soFromBeginning);
  Stream.CopyFrom(Self,Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToFile` writes the contents of the memory as one big block.

Errors: If an error occurs when creating or writing to the file, an EStreamError (43) exception may occur.

See also: TCustomMemoryStream.SaveToStream (102), TFileStream (108), TStream.CopyFrom (143)

TCustomMemoryStream.Memory

Synopsis: Pointer to the data kept in the memory stream.

Declaration: `Property Memory : Pointer`

Visibility: public

Access: Read

Description: `Memory` points to the memory area where stream keeps it's data. The property is read-only, so the pointer cannot be set this way.

Remark: Do not write to the memory pointed to by `Memory`, since the memory content may be read-only, and thus writing to it may cause errors.

See also: TStream.Size (148)

1.36 TDataModule

Method overview

Page	Method	Description
105	AfterConstruction	
105	BeforeDestruction	
105	Create	
105	CreateNew	
104	DefineProperties	
105	Destroy	
104	DoCreate	
104	DoDestroy	
104	GetChildren	
105	HandleCreateException	
105	ReadState	

Property overview

Page	Property	Access	Description
105	DesignOffset	rw	
106	DesignSize	rw	
106	OldCreateOrder	rw	
106	OnCreate	rw	
106	OnDestroy	rw	

TDataModule.DoCreate

Declaration: `procedure DoCreate;  Virtual`

Visibility: protected

TDataModule.DoDestroy

Declaration: `procedure DoDestroy;  Virtual`

Visibility: protected

TDataModule.DefineProperties

Declaration: `procedure DefineProperties(Filer: TFiler);  Override`

Visibility: protected

TDataModule.GetChildren

Declaration: `procedure GetChildren(Proc: TGetChildProc;Root: TComponent);  Override`

Visibility: protected

TDataModule.HandleCreateException

Declaration: `function HandleCreateException : Boolean;  Virtual`

Visibility: protected

TDataModule.ReadState

Declaration: `procedure ReadState(Reader: TReader);  Override`

Visibility: protected

TDataModule.Create

Declaration: `constructor Create(AOwner: TComponent);  Override`

Visibility: public

TDataModule.CreateNew

Declaration:
```
constructor CreateNew(AOwner: TComponent)
constructor CreateNew(AOwner: TComponent;CreateMode: Integer);  Virtual
```

Visibility: public

TDataModule.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

TDataModule.AfterConstruction

Declaration: `procedure AfterConstruction;  Override`

Visibility: public

TDataModule.BeforeDestruction

Declaration: `procedure BeforeDestruction;  Override`

Visibility: public

TDataModule.DesignOffset

Declaration: `Property DesignOffset : TPoint`

Visibility: public

Access: Read,Write

TDataModule.DesignSize

Declaration: `Property DesignSize : TPoint`

Visibility: public

Access: Read,Write

TDataModule.OnCreate

Declaration: `Property OnCreate : TNotifyEvent`

Visibility: published

Access: Read,Write

TDataModule.OnDestroy

Declaration: `Property OnDestroy : TNotifyEvent`

Visibility: published

Access: Read,Write

TDataModule.OldCreateOrder

Declaration: `Property OldCreateOrder : Boolean`

Visibility: published

Access: Read,Write

1.37 TFiler

Description

Class responsible for streaming of components.

Method overview

Page	Method	Description
107	DefineBinaryProperty	
107	DefineProperty	
107	SetRoot	Sets the root component which is being streamed.

Property overview

Page	Property	Access	Description
108	Ancestor	rw	Ancestor component from which an inherited component is streamed.
108	IgnoreChildren	rw	Determines whether children will be streamed as well.
107	LookupRoot	r	Component used to look up ancestor components.
107	Root	rw	The root component is the initial component which is being streamed.

TFiler.SetRoot

Synopsis: Sets the root component which is being streamed.

Declaration:
```
procedure SetRoot(ARoot: TComponent);  Virtual
```

Visibility: protected

Description: Sets the root component. The root component is the initial component which is being streamed.

TFiler.DefineProperty

Synopsis:

Declaration:
```
procedure DefineProperty(const Name: String;ReadData: TReaderProc;
                        WriteData: TWriterProc;HasData: Boolean)
                        ;  Virtual;  Abstract
```

Visibility: public

Description:

TFiler.DefineBinaryProperty

Synopsis:

Declaration:
```
procedure DefineBinaryProperty(const Name: String;ReadData: TStreamProc;
                              WriteData: TStreamProc;HasData: Boolean)
                              ;  Virtual;  Abstract
```

Visibility: public

Description:

TFiler.Root

Synopsis: The root component is the initial component which is being streamed.

Declaration:
```
Property Root : TComponent
```

Visibility: public

Access: Read,Write

Description: The streaming process will stream a component and all the components which it owns. The `Root` component is the component which is initially streamed.

See also: TFiler.LookupRoot (107)

TFiler.LookupRoot

Synopsis: Component used to look up ancestor components.

Declaration:
```
Property LookupRoot : TComponent
```

Visibility: public

Access: Read

Description: When comparing inherited component's values against parent values, the values are compared with the component in `LookupRoot`. Initially, it is set to Root (107).

See also: TFiler.Root (107)

TFiler.Ancestor

Synopsis: Ancestor component from which an inherited component is streamed.

Declaration: `Property Ancestor : TPersistent`

Visibility: public

Access: Read,Write

Description: When streaming a component, this is the parent component. Only properties that differ from the parent's property value will be streamed.

See also: TFiler.Root (107), TFiler.LookupRoot (107)

TFiler.IgnoreChildren

Synopsis: Determines whether children will be streamed as well.

Declaration: `Property IgnoreChildren : Boolean`

Visibility: public

Access: Read,Write

Description: By default, all children (i.e. owned objects) will also be streamed when streaming a component. This property can be used to prevent owned objects from being streamed.

1.38 TFileStream

Description

`TFileStream` is a TStream (140) descdendent that stores or reads it's data from a named file in the filesystem of the operating system.

To this end, it overrides some of the abstract methods in `TStream` and implements them for the case of files on disk, and it adds the FileName (109) property to the list of public properties.

Method overview

Page	Method	Description
109	Create	Creates a file stream.
109	Destroy	Destroys the file stream.

Property overview

Page	Property	Access	Description
109	FileName	r	The filename of the stream.

TFileStream.Create

Synopsis: Creates a file stream.

Declaration:
```
constructor Create(const AFileName: String;Mode: Word)
constructor Create(const AFileName: String;Mode: Word;Rights: Cardinal)
```

Visibility: public

Description: `Create` creates a new instance of a `TFileStream` class. It opens the file `AFileName` with mode `Mode`, which can have one of the following values:

Table 1.14:

fmCreate	TFileStream.Create (109) creates a new file if needed.
fmOpenRead	TFileStream.Create (109) opens a file with read-only access.
fmOpenWrite	TFileStream.Create (109) opens a file with write-only access.
fmOpenReadWrite	TFileStream.Create (109) opens a file with read-write access.

After the file has been opened in the requested mode and a handle has been obtained from the operating system, the inherited constructor is called.

Errors: If the file could not be opened in the requested mode, an EFOpenError (42) exception is raised.

See also: TStream (140), TFileStream.FileName (109), THandleStream.Create (110)

TFileStream.Destroy

Synopsis: Destroys the file stream.

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

Visibility: public

Description: `Destroy` closes the file (causing possible buffered data to be written to disk) and then calls the inherited destructor.

Do not call `destroy` directly, instead call the `Free` method. Destroy does not check whether `Self` is nil, while `Free` does.

See also: TFileStream.Create (109)

TFileStream.FileName

Synopsis: The filename of the stream.

Declaration:
```
Property FileName : String
```

Visibility: public

Access: Read

Description: `FileName` is the name of the file that the stream reads from or writes to. It is the name as passed in the constructor of the stream; it cannot be changed. To write to another file, the stream must be freed and created again with the new filename.

See also: TFileStream.Create (109)

1.39 THandleStream

Description

`THandleStream` is an abstract descendent of the TStream (140) class that provides methods for a stream to handle all reading and writing to and from a handle, provided by the underlying OS. To this end, it overrides the Read (111) and Write (111) methods of TStream.

Remark:

- `ThandleStream` does not obtain a handle from the OS by itself, it just handles reading and writing to such a handle by wrapping the system calls for reading and writing; Descendent classes should obtain a handle from the OS by themselves and pass it on in the inherited constructor.
- Contrary to Delphi, no seek is implemented for `THandleStream`, since pipes and sockets do not support this. The seek is implemented in descendent methods that support it.

Method overview

Page	Method	Description
110	Create	Create a handlestream from an OS Handle.
111	Read	Overrides standard read method.
111	Seek	
110	SetSize	
111	Write	Overrides standard write method.

Property overview

Page	Property	Access	Description
111	Handle	r	The OS handle of the stream.

THandleStream.SetSize

Declaration:
```
procedure SetSize(NewSize: LongInt);  Override
procedure SetSize(NewSize: Int64);  Override
```

Visibility: protected

THandleStream.Create

Synopsis: Create a handlestream from an OS Handle.

Declaration: `constructor Create(AHandle: Integer)`

Visibility: public

Description: `Create` creates a new instance of a `THandleStream` class. It stores `AHandle` in an internal variable and then calls the inherited constructor.

See also: TStream (140)

THandleStream.Read

Synopsis: Overrides standard read method.

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

Visibility: public

Description: `Read` implements the abstract Read (141) method of `TStream`. It uses the Handle (111) property to read the `Count` bytes into `Buffer`

If no error occurs while reading, the number of bytes actually read will be returned.

Errors: If the operating system reports an error while reading from the handle, -1 is returned.

See also: TStream.Read (141), THandleStream.Write (111), THandleStream.Handle (111)

THandleStream.Write

Synopsis: Overrides standard write method.

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

Visibility: public

Description: `Write` implements the abstract Write (141) method of `TStream`. It uses the Handle (111) property to write the `Count` bytes from `Buffer`.

If no error occurs while writing, the number of bytes actually written will be returned.

Errors: If the operating system reports an error while writing to handle, -1 is returned.

See also: TStream.Read (141), THandleStream.Write (111), THandleStream.Handle (111)

THandleStream.Seek

Declaration: `function Seek(Offset: Int64;Origin: TSeekOrigin) : Int64;  Override`

Visibility: public

THandleStream.Handle

Synopsis: The OS handle of the stream.

Declaration: `Property Handle : Integer`

Visibility: public

Access: Read

Description: `Handle` represents the Operating system handle to which reading and writing is done. The handle can be read only, i.e. it cannot be set after the `THandlestream` instance was created. It should be passed to the constructor THandleStream.Create (110)

See also: THandleStream (110), THandleStream.Create (110)

1.40 TList

Description

`TList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed.

To manage collections of strings, it is better to use a TStrings (155) descendent such as TStringList (149). To manage general objects, a TCollection (76) class exists, from which a descendent can be made to manage collections of various kinds.

Method overview

Page	Method	Description
113	Add	Adds a new pointer to the list.
116	Assign	
114	Clear	Clears the pointer list.
114	Delete	Removes a pointer from the list.
113	Destroy	Destroys the list and releases the memory used to store the list elements.
114	Error	Raises an EListError (42) exception.
114	Exchange	Exchanges two pointers in the list.
114	Expand	Increases the capacity of the list if needed.
115	Extract	
115	First	Returns the first non-nil pointer in the list.
112	Get	
113	Grow	
115	IndexOf	Returns the index of a given pointer.
115	Insert	Inserts a new pointer in the list at a given position.
116	Last	Returns the last non-nil pointer in the list.
116	Move	Moves a pointer from one position in the list to another.
113	Notify	
116	Pack	Removes `Nil` pointers from the list and frees unused memory.
113	Put	
116	Remove	Removes a value from the list.
113	SetCapacity	
113	SetCount	
117	Sort	Sorts the pointers in the list.

Property overview

Page	Property	Access	Description
117	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
117	Count	rw	Current number of pointers in the list.
118	Items	rw	Probides access to the pointers in the list.
118	List	r	Memory array where pointers are stored.

TList.Get

Declaration: `function Get(Index: Integer) : Pointer`

Visibility: protected

TList.Grow

Declaration: `procedure Grow;  Virtual`

Visibility: protected

TList.Put

Declaration: `procedure Put(Index: Integer;Item: Pointer)`

Visibility: protected

TList.Notify

Declaration: `procedure Notify(Ptr: Pointer;Action: TListNotification);  Virtual`

Visibility: protected

TList.SetCapacity

Declaration: `procedure SetCapacity(NewCapacity: Integer)`

Visibility: protected

TList.SetCount

Declaration: `procedure SetCount(NewCount: Integer)`

Visibility: protected

TList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any meomory they point to must be explicitly released before calling the destructor.

TList.Add

Synopsis: Adds a new pointer to the list.

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

Visibility: public

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the Grow (113) method.

To insert a pointer at a certain position in the list, use the Insert (115) method instead.

See also: TList.Delete (114), TList.Grow (113), TList.Insert (115)

TList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear;  Dynamic`

Visibility: public

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: TList.Destroy (113)

TList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: public

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

TList.Error

Synopsis: Raises an EListError (42) exception.

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

Visibility: public

Description: `Error` raises an EListError (42) exception, with a message formatted with `Msg` and `Data`.

TList.Exchange

Synopsis: Exchanges two pointers in the list.

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

Visibility: public

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be withing the current range of the list, or an EListError (42) exception will be raised.

TList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TList`

Visibility: public

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

1. If the capacity is less than 3, the capacity is increased with 4.
2. If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
3. If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: TList.Capacity (117),

TList.Extract

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

Visibility: public

TList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TList.Last (116)

TList.IndexOf

Synopsis: Returns the index of a given pointer.

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

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

TList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer;Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a EListError (42) exception is raised.

See also: TList.Add (113), Tlist.Delete (114)

TList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: TList.First (115)

TList.Move

Synopsis: Moves a pointer from one position in the list to another.

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

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

If `CurIndex` or `Newindex` are not inside the valid range of indices, an EListError (42) exception is raised.

See also: TList.Exchange (114)

TList.Assign

Declaration: `procedure Assign(Obj: TList)`

Visibility: public

TList.Remove

Synopsis: Removes a value from the list.

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

Visibility: public

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: TList.Delete (114), TList.IndexOf (115), Tlist.Insert (115)

TList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded nil pointers in it.

See also: TList.Clear (114)

TList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort>` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

TList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using add (113) or insert (115), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: TList.SetCapacity (113), TList.Count (117)

TList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

TList.Items

Synopsis: Probides access to the pointers in the list.

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

Visibility: public

Access: Read,Write

Description: `Items` is used to access the pointers in the list. It is the default property of the `TList` class, so it can be omitted.

The list is zero-based, so Index must be in the range `0` to `Count-1`.

TList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: public

Access: Read

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pinters to another location.

1.41 TMemoryStream

Description

`TMemoryStream` is a TStream (140) descendent that stores it's data in memory. It descends directly from TCustomMemoryStream (101) and implements the necessary to allocate and de-allocate memory diretly from the heap. It implements the Write (120) method which is missing in `TCustomMemoryStream`.

`TMemoryStream` also introduces methods to load the contents of another stream or a file into the memory stream.

It is not necessary to do any memory management manually, as the stream will allocate or de-allocate memory as needed. When the stream is freed, all allocated memory will be freed as well.

Method overview

Page	Method	Description
119	Clear	Zeroes the position, capacity and size of the stream.
119	Destroy	Frees any allocated memory and destroys the memory stream.
120	LoadFromFile	Loads the contents of a file into memory.
119	LoadFromStream	Loads the contents of a stream into memory.
119	Realloc	Sets the new capacity for the memory stream
120	SetSize	Sets the size for the memory stream.
120	Write	Writes data to the stream's memory.

Property overview

Page	Property	Access	Description
121	Capacity	rw	Current capacity of the stream.

TMemoryStream.Realloc

Synopsis: Sets the new capacity for the memory stream

Declaration: `function Realloc(var NewCapacity: LongInt) : Pointer;  Virtual`

Visibility: protected

Description: `SetCapacity` sets the capacity of the memory stream, i.e. does the actual allocation or de-allocation of memory for the stream. It allocates at least `NewCapacity` bytes on the heap, moves the current contents of the stream to this location (as much as fits in) and returns the new memory location. Extra allocated memory is not initialized, i.e. may contain garbage.

Memory is allocated in blocks of 4 Kb; this can be changed by overriding the method.

See also: TMemoryStream.Capacity (121)

TMemoryStream.Destroy

Synopsis: Frees any allocated memory and destroys the memory stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Free` clears the memory stream, thus in effect freeing any memory allocated for it, and then frees the memory stream.

TMemoryStream.Clear

Synopsis: Zeroes the position, capacity and size of the stream.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` sets the position and size to 0, and sets the capacity of the stream to 0, thus freeing all memory allocated for the stream.

See also: TStream.Size (148), TStream.Position (148), TCustomMemoryStream.Memory (103)

TMemoryStream.LoadFromStream

Synopsis: Loads the contents of a stream into memory.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the memorybuffer of the stream. Any previous contents of the memory stream are overwritten. Memory is allocated as needed.

Remark: The `LoadFromStream` uses the Size (148) property of `Stream` to determine how much memory must be allocated. Some streams do not allow the stream size to be determined, so care must be taken when using this method.

This method will work much faster than the use of the TStream.CopyFrom (143) method:

```
Seek(0,soFromBeginning);
  CopyFrom(Stream,Stream.Size);
```

because the `CopyFrom` method copies the contents in blocks, while `LoadFromStream` reads the contents of the stream as one big block.

Errors: If an error occurs when reading from the stream, an EStreamError (43) may occur.

See also: TStream.CopyFrom (143), TMemoryStream.LoadFromFile (120)

TMemoryStream.LoadFromFile

Synopsis: Loads the contents of a file into memory.

Declaration: `procedure LoadFromFile(const FileName: String)`

Visibility: public

Description: `LoadFromFile` loads the contents of the file with name `FileName` into the memory stream. The current contents of the memory stream is replaced by the contents of the file. Memory is allocated as needed.

The `LoadFromFile` method simply creates a filestream and then calls the TMemoryStream.LoadFromStream (119) method.

See also: TMemoryStream.LoadFromStream (119)

TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(NewSize: LongInt);  Override`

Visibility: public

Description: `SetSize` sets the size of the memory stream to `NewSize`. This will set the capacity of the stream to `NewSize` and correct the current position in the stream when needed.

See also: TStream.Position (148), TStream.Size (148)

TMemoryStream.Write

Synopsis: Writes data to the stream's memory.

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

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream's memory, starting at the current position in the stream. If more memory is needed than currently allocated, more memory will be allocated. Any contents in the memory stream at the current position will be overwritten. The function returns the number of bytes actually written (which should under normal circumstances always equal `Count`).

This method overrides the abstract TStream.Write (141) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: TCustomMemoryStream.Read (102)

TMemoryStream.Capacity

Synopsis: Current capacity of the stream.

Declaration: `Property Capacity : LongInt`

Visibility: protected

Access: Read,Write

Description: `Capacity` is the current capacity of the stream, this is the current size of the memory allocated to the stream. This is not necessarily equal to the size of the stream, but will always be larger than or equal to the size of the stream. When writing to the stream, the TMemoryStream.Write (120) sets the capacity to the needed value.

If a lot of write operations will occur, performance may be improved by setting the capacity to a large value, so less reallocations of memory will occur while writing to the stream.

See also: TMemoryStream.ReAlloc (119)

1.42 TOwnedCollection

Method overview

Page	Method	Description
121	Create	
121	GetOwner	

TOwnedCollection.GetOwner

Declaration: `function GetOwner : TPersistent;  Override`

Visibility: protected

TOwnedCollection.Create

Declaration: `constructor Create(AOwner: TPersistent;AItemClass: TCollectionItemClass)`

Visibility: public

1.43 TParser

Description

Class to parse the contents of a stream containing text data.

Method overview

Page	Method	Description
122	CheckToken	Checks whether the token if of the given type.
123	CheckTokenSymbol	Checks whether the token equals the given symbol
122	Create	Creates a new parser instance.
122	Destroy	Destroys the parser instance.
123	Error	Raises an EParserError (43) exception with the given message
123	ErrorFmt	Raises an EParserError (43) exception and formats the message.
123	ErrorStr	Raises an EParserError (43) exception with the given message
123	HexToBinary	Writes hexadecimal data to the stream.
123	NextToken	Reads the next token and returns its type.
124	SourcePos	Returns the current position in the stream.
124	TokenComponentIdent	Checks whether the current token is a component identifier.
124	TokenFloat	Returns the current token as a float.
124	TokenInt	Returns the current token as an integer.
124	TokenString	Returns the current token as a string.
124	TokenSymbolIs	Returns `True` if the current token is a symbol.

Property overview

Page	Property	Access	Description
125	SourceLine	r	Current source linenumber.
125	Token	r	Contents of the current token.

TParser.Create

Synopsis: Creates a new parser instance.

Declaration: `constructor Create(Stream: TStream)`

Visibility: public

Description: Creates a new parser instance.

TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the parser instance.

TParser.CheckToken

Synopsis: Checks whether the token if of the given type.

Declaration: `procedure CheckToken(T: Char)`

Visibility: public

Description: Checks whether the token if of the given type.

TParser.CheckTokenSymbol

Synopsis: Checks whether the token equals the given symbol

Declaration: `procedure CheckTokenSymbol(const S: String)`

Visibility: public

Description: Checks whether the token equals the given symbol

TParser.Error

Synopsis: Raises an EParserError (43) exception with the given message

Declaration: `procedure Error(const Ident: String)`

Visibility: public

Description: Raises an EParserError (43) exception with the given message

TParser.ErrorFmt

Synopsis: Raises an EParserError (43) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: String;const Args: Array[] of const)`

Visibility: public

Description: Raises an EParserError (43) exception and formats the message.

TParser.ErrorStr

Synopsis: Raises an EParserError (43) exception with the given message

Declaration: `procedure ErrorStr(const Message: String)`

Visibility: public

Description: Raises an EParserError (43) exception with the given message

TParser.HexToBinary

Synopsis: Writes hexadecimal data to the stream.

Declaration: `procedure HexToBinary(Stream: TStream)`

Visibility: public

Description: Writes hexadecimal data to the stream.

TParser.NextToken

Synopsis: Reads the next token and returns its type.

Declaration: `function NextToken : Char`

Visibility: public

Description: Reads the next token and returns its type.

TParser.SourcePos

Synopsis: Returns the current position in the stream.

Declaration: `function SourcePos : LongInt`

Visibility: public

Description: Returns the current position in the stream.

TParser.TokenComponentIdent

Synopsis: Checks whether the current token is a component identifier.

Declaration: `function TokenComponentIdent : String`

Visibility: public

Description: Checks whether the current token is a component identifier.

TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: public

Description: Returns the current token as a float.

TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : LongInt`

Visibility: public

Description: Returns the current token as an integer.

TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : String`

Visibility: public

Description: Returns the current token as a string.

TParser.TokenSymbolIs

Synopsis: Returns `True` if the current token is a symbol.

Declaration: `function TokenSymbolIs(const S: String) : Boolean`

Visibility: public

Description: Returns `True` if the current token is a symbol.

TParser.SourceLine

Synopsis: Current source linenumber.

Declaration: `Property SourceLine : Integer`

Visibility: public

Access: Read

Description: Current source linenumber.

TParser.Token

Synopsis: Contents of the current token.

Declaration: `Property Token : Char`

Visibility: public

Access: Read

Description: Contents of the current token.

1.44 TPersistent

Description

`TPersistent` is the basic class for the streaming system. Since it is compiled in the `{$M+}` state, the compiler generates RTTI (Run-Time Type Information) for it and all classes that descend from it. This information can be used to stream all properties of classes.

It also introduces functionality to assign the contents of 2 classes to each other.

Method overview

Page	Method	Description
127	Assign	Assign the contents of one class to another.
125	AssignTo	Generic assignment function.
126	DefineProperties	Declare non-published properties that need to be streamed.
127	Destroy	Destroys the `TPersistent` instance.
127	GetNamePath	Returns a string that can be used to identify the class instance.
126	GetOwner	Returns the owner of the component.

TPersistent.AssignTo

Synopsis: Generic assignment function.

Declaration: `procedure AssignTo(Dest: TPersistent);  Virtual`

Visibility: protected

Description: `AssignTo` is the generic function to assign the class' contents to another class. This method must be overridden by descendent classes to actually assign the content of the source instance to the destination instance.

The TPersistent (125) implementation of `Assignto` raises an `EConvertError` exception. This is done for the following reason: If the source class doesn't know how to assign itself to the destination class (using `AssignTo`), the destination class may know how get the data from the source class (using Assign (127)). If all descendent methods are implemented correctly, then if neither of the two classes knows how to assign their contents to each other, execution will end up at TPersistent.Assign (127), which will simply execute

```
Dest.AssignTo(Self);
```

If neither of the classes knows how to assign to/from each other, then execution will end up at the `TPersistent` implementation of `AssignTo`, and an exception will be raised.

See also: TPersistent.Assign (127)

TPersistent.DefineProperties

Synopsis: Declare non-published properties that need to be streamed.

Declaration: `procedure DefineProperties(Filer: TFiler);  Virtual`

Visibility: protected

Description: `DefineProperties` must be overridden by descendent classes to indicate to the streaming system which non-published properties must also be streamed.

The streaming systems stores only published properties in the stream. Sometimes it is necessary to store additional data in the stream, data which is not published. This can be done by overriding the `DefineProperties` method. The `Filer` object is the class that is responsible for writing all properties to the stream.

To define new properties, two methods of the TFiler (106) class should be used:

1. DefineProperty (107), to define a property which can be represented as text.
2. DefineProperty (107), to define a property which contains binary data.

On order for the streaming to work correctly, a call to the inherited `DefineProperties` is also needed, so ancestor objects also get the possibility to read or write their private data to the stream. Failure to call the inherited method will result in component properties not being streamed correctly.

See also: TFiler.DefineProperties (106), TFiler (106)

TPersistent.GetOwner

Synopsis: Returns the owner of the component.

Declaration: `function GetOwner : TPersistent;  Dynamic`

Visibility: protected

Description: `GetOwner` returns the owning component of the classes instane. The `TPersistent` implementation of `GetOwner` returns `Nil`. TComponent (87) overrides this method.

See also: TComponent (87)

TPersistent.Destroy

Synopsis: Destroys the `TPersistent` instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` disposes of the persistent object. This method should never be called directly. Instead the `Free` method should be used.

TPersistent.Assign

Synopsis: Assign the contents of one class to another.

Declaration: `procedure Assign(Source: TPersistent);  Virtual`

Visibility: public

Description: `Assign` copies the contents of `Source` to `Self`, if the classes of the destination and source classes are compatible.

The `TPersistent` implementation of `Assign` does nothing but calling the AssignTo (125) method of source. This means that if the destination class does not know how to assign the contents of the source class, the source class instance is asked to assign itself to the destination class. This means that it is necessary to implement only one of the two methods so that two classes can be assiged to one another.

Remark: In general, a statement of the form

```
Destination:=Source;
```

(where `Destination` and `Source` are classes) does not achieve the same as a statement of the form

```
Destination.Assign(Source);
```

After the former statement, both `Source` and `Destination` will point to the same object. The latter statemtent will copy the *contents* of the `Source` class to the `Destination` class.

See also: TPersistent.AssignTo (125)

TPersistent.GetNamePath

Synopsis: Returns a string that can be used to identify the class instance.

Declaration: `function GetNamePath : String;  Virtual`

Visibility: public

Description: `GetNamePath` returns a string that can be used to identify the class instance. This can be used to display a name for this instance in a Object designer.

`GetNamePath` constructs a name by recursively prepending the `Classname` of the `Owner` instance to the `Classname` of this instance, separated by a dot.

See also: TPersistent.GetOwner (126)

1.45 TReader

Description

The `TReader` class is a reader class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class TAbstractObjectReader (44) to do the actual reading of data. The interface of the `TReader` class should be identical to the interface in Delphi.

Method overview

Page	Method	Description
131	BeginReferences	Initializes the component referencing mechanism.
131	CheckValue	Raises an exception if the next value in the stream is not of type `Value`
135	CopyValue	Copy a value to a writer.
130	Create	Creates a new reader class
131	DefineBinaryProperty	Reads a user-defined binary property from the stream.
131	DefineProperty	Reads a user-defined property from the stream.
130	Destroy	Destroys a reader class.
131	EndOfList	Returns true if the stream contains an end-of-list marker.
132	EndReferences	Finalizes the component referencing mechanism.
129	Error	Calls an installed error handler and passes it `Message`
129	FindMethod	Return the address of a published method.
132	FixupReferences	Tries to resolve all unresolved component references.
132	NextValue	Returns the type of the next value.
130	PropertyError	Skips a property value and raises an exception.
132	ReadBoolean	Reads a boolean from the stream.
132	ReadChar	Reads a character from the stream.
132	ReadCollection	Reads a collection from the stream.
133	ReadComponent	Starts reading a component from the stream.
133	ReadComponents	Starts reading child components from the stream.
130	ReadData	Reads the components data after it has been created.
133	ReadDate	Reads a date from the stream
133	ReadFloat	Reads a float from the stream.
133	ReadIdent	Reads an identifier from the stream.
134	ReadInt64	Reads a 64-bit integer from the stream.
134	ReadInteger	Reads an integer from the stream
134	ReadListBegin	Checks for the beginning of a list.
134	ReadListEnd	Checks for the end of a list.
130	ReadProperty	Read and process a property name
130	ReadPropValue	Reads a property value for `PropInfo`.
134	ReadRootComponent	Starts reading a root component.
133	ReadSingle	Reads a single-type real from the stream.
134	ReadString	Reads a string from the stream.
135	ReadValue	Reads the next value type from the stream.

Property overview

Page	Property	Access	Description
135	CanHandleExceptions	r	Indicates whether the reader is handling exceptions at this stage.
135	Driver	r	The driver in use for streaming the data.
137	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
137	OnCreateComponent	rw	Handler called when a component needs to be created.
136	OnError	rw	Handler called when an error occurs.
138	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
136	OnFindMethod	rw	Handler to find or change a method address.
136	OnPropertyNotFound	rw	
137	OnReferenceName	rw	Handler called when another component is referenced.
137	OnSetMethodProperty	rw	
137	OnSetName	rw	Handler called when setting a component name.
136	Owner	rw	Owner of the component being read
136	Parent	rw	Parent of the component being read.
135	PropName	r	Name of the property being read at this moment.

TReader.Error

Synopsis: Calls an installed error handler and passes it `Message`

Declaration:
```
function Error(const Message: String) : Boolean;  Virtual
```

Visibility: protected

Description: `Error` returns `False` if no TReader.OnError (136) handler is installed. If one is installed, then it will be called, passing the reader instance, message, and function return value as parameters.

If the function result `False`, i.e. when there is no handler installed or the handler restured `False`, then the calling code will raise an exception.

See also: TReader.FindMethod (129)

TReader.FindMethod

Synopsis: Return the address of a published method.

Declaration:
```
function FindMethod(ARoot: TComponent;const AMethodName: String)
                   : Pointer;  Virtual
```

Visibility: protected

Description: `FindMethod` will search for the method in `ARoot`. If it isn't found there, then it will call a `OnFindMethod` handler, if one is installed, passing it the method name `AMethodName`, the result pointer and a variable which says whether an exception should be raised if no method with name `AMethodName` is found.

If the method cannot be found and the OnFindMethod (136) returns `True`, then an exception will be raised.

See also: TReader.OnFindMethod (136), TFindMethodEvent (25)

TReader.ReadProperty

Synopsis: Read and process a property name

Declaration: `procedure ReadProperty(AInstance: TPersistent)`

Visibility: protected

Description: Read and process a property name

TReader.ReadPropValue

Synopsis: Reads a property value for `PropInfo`.

Declaration: `procedure ReadPropValue(Instance: TPersistent;PropInfo: Pointer)`

Visibility: protected

Description: Reads a property value for `PropInfo`.

TReader.PropertyError

Synopsis: Skips a property value and raises an exception.

Declaration: `procedure PropertyError`

Visibility: protected

Description: Skips a property value and raises an exception.

TReader.ReadData

Synopsis: Reads the components data after it has been created.

Declaration: `procedure ReadData(Instance: TComponent)`

Visibility: protected

Description: Reads the components data after it has been created.

TReader.Create

Synopsis: Creates a new reader class

Declaration: `constructor Create(Stream: TStream;BufSize: Integer)`

Visibility: public

Description: Creates a new reader class

TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys a reader class.

TReader.BeginReferences

Synopsis: Initializes the component referencing mechanism.

Declaration: `procedure BeginReferences`

Visibility: public

Description: Initializes the component referencing mechanism.

TReader.CheckValue

Synopsis: Raises an exception if the next value in the stream is not of type `Value`

Declaration: `procedure CheckValue(Value: TValueType)`

Visibility: public

Description: Raises an exception if the next value in the stream is not of type `Value`

TReader.DefineProperty

Synopsis: Reads a user-defined property from the stream.

Declaration:
```
procedure DefineProperty(const Name: String;AReadData: TReaderProc;
                        WriteData: TWriterProc;HasData: Boolean)
                        ;  Override
```

Visibility: public

Description: Reads a user-defined property from the stream.

TReader.DefineBinaryProperty

Synopsis: Reads a user-defined binary property from the stream.

Declaration:
```
procedure DefineBinaryProperty(const Name: String;
                              AReadData: TStreamProc;
                              WriteData: TStreamProc;HasData: Boolean)
                              ;  Override
```

Visibility: public

Description: Reads a user-defined binary property from the stream.

TReader.EndOfList

Synopsis: Returns true if the stream contains an end-of-list marker.

Declaration: `function EndOfList : Boolean`

Visibility: public

Description: Returns true if the stream contains an end-of-list marker.

TReader.EndReferences

Synopsis: Finalizes the component referencing mechanism.

Declaration: `procedure EndReferences`

Visibility: public

Description: Finalizes the component referencing mechanism.

TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: public

Description: Tries to resolve all unresolved component references.

TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: public

Description: Returns the type of the next value.

TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: public

Description: Reads a boolean from the stream.

TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: public

Description: Reads a character from the stream.

TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration: `procedure ReadCollection(Collection: TCollection)`

Visibility: public

Description: Reads a collection from the stream.

TReader.ReadComponent

Synopsis: Starts reading a component from the stream.

Declaration: `function ReadComponent(Component: TComponent) : TComponent`

Visibility: public

Description: Starts reading a component from the stream.

TReader.ReadComponents

Synopsis: Starts reading child components from the stream.

Declaration:
```
procedure ReadComponents(AOwner: TComponent;AParent: TComponent;
                        Proc: TReadComponentsProc)
```

Visibility: public

Description: Starts reading child components from the stream.

TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration: `function ReadFloat : Extended`

Visibility: public

Description: Reads a float from the stream.

TReader.ReadSingle

Synopsis: Reads a single-type real from the stream.

Declaration: `function ReadSingle : Single`

Visibility: public

Description: Reads a single-type real from the stream.

TReader.ReadDate

Synopsis: Reads a date from the stream

Declaration: `function ReadDate : TDateTime`

Visibility: public

Description: Reads a date from the stream

TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : String`

Visibility: public

Description: Reads an identifier from the stream.

TReader.ReadInteger

Synopsis: Reads an integer from the stream

Declaration: `function ReadInteger : LongInt`

Visibility: public

Description: Reads an integer from the stream

TReader.ReadInt64

Synopsis: Reads a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64`

Visibility: public

Description: Reads a 64-bit integer from the stream.

TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: public

Description: Checks for the beginning of a list.

TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: public

Description: Checks for the end of a list.

TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent(ARoot: TComponent) : TComponent`

Visibility: public

Description: Starts reading a root component.

TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : String`

Visibility: public

Description: Reads a string from the stream.

TReader.ReadValue

Synopsis: Reads the next value type from the stream.

Declaration: `function ReadValue : TValueType`

Visibility: public

Description: Reads the next value type from the stream.

TReader.CopyValue

Synopsis: Copy a value to a writer.

Declaration: `procedure CopyValue(Writer: TWriter)`

Visibility: public

Description: Copy a value to a writer.

TReader.PropName

Synopsis: Name of the property being read at this moment.

Declaration: `Property PropName : String`

Visibility: protected

Access: Read

Description: Name of the property being read at this moment.

TReader.CanHandleExceptions

Synopsis: Indicates whether the reader is handling exceptions at this stage.

Declaration: `Property CanHandleExceptions : Boolean`

Visibility: protected

Access: Read

Description: Indicates whether the reader is handling exceptions at this stage.

TReader.Driver

Synopsis: The driver in use for streaming the data.

Declaration: `Property Driver : TAbstractObjectReader`

Visibility: public

Access: Read

Description: The driver in use for streaming the data.

TReader.Owner

Synopsis: Owner of the component being read

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read,Write

Description: Owner of the component being read

TReader.Parent

Synopsis: Parent of the component being read.

Declaration: `Property Parent : TComponent`

Visibility: public

Access: Read,Write

Description: Parent of the component being read.

TReader.OnError

Synopsis: Handler called when an error occurs.

Declaration: `Property OnError : TReaderError`

Visibility: public

Access: Read,Write

Description: Handler called when an error occurs.

TReader.OnPropertyNotFound

Declaration: `Property OnPropertyNotFound : TPropertyNotFoundEvent`

Visibility: public

Access: Read,Write

TReader.OnFindMethod

Synopsis: Handler to find or change a method address.

Declaration: `Property OnFindMethod : TFindMethodEvent`

Visibility: public

Access: Read,Write

Description: Handler to find or change a method address.

TReader.OnSetMethodProperty

Declaration: `Property OnSetMethodProperty : TSetMethodPropertyEvent`

Visibility: public

Access: Read,Write

TReader.OnSetName

Synopsis: Handler called when setting a component name.

Declaration: `Property OnSetName : TSetNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when setting a component name.

TReader.OnReferenceName

Synopsis: Handler called when another component is referenced.

Declaration: `Property OnReferenceName : TReferenceNameEvent`

Visibility: public

Access: Read,Write

Description: Handler called when another component is referenced.

TReader.OnAncestorNotFound

Synopsis: Handler called when the ancestor component cannot be found.

Declaration: `Property OnAncestorNotFound : TAncestorNotFoundEvent`

Visibility: public

Access: Read,Write

Description: Handler called when the ancestor component cannot be found.

TReader.OnCreateComponent

Synopsis: Handler called when a component needs to be created.

Declaration: `Property OnCreateComponent : TCreateComponentEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component needs to be created.

TReader.OnFindComponentClass

Synopsis: Handler called when a component class reference needs to be found.

Declaration: `Property OnFindComponentClass : TFindComponentClassEvent`

Visibility: public

Access: Read,Write

Description: Handler called when a component class reference needs to be found.

1.46 TRecall

Method overview

Page	Method	Description
138	Create	
138	Destroy	
138	Forget	
138	Store	

Property overview

Page	Property	Access	Description
139	Reference	r	

TRecall.Create

Declaration: `constructor Create(AStorage: TPersistent;AReference: TPersistent)`

Visibility: public

TRecall.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

TRecall.Store

Declaration: `procedure Store`

Visibility: public

TRecall.Forget

Declaration: `procedure Forget`

Visibility: public

TRecall.Reference

Declaration: `Property Reference : TPersistent`

Visibility: public

Access: Read

1.47 TResourceStream

Description

Stream that reads its data from a resource object.

Method overview

Page	Method	Description
139	Create	Creates a new instance of a resource stream.
139	CreateFromID	Creates a new instance of a resource stream with resource
139	Destroy	Destroys the instance of the resource stream.
140	Write	`Write` implements the abstract TStream.Write (141) method.

TResourceStream.Create

Synopsis: Creates a new instance of a resource stream.

Declaration:
```
constructor Create(Instance: THandle;const ResName: String;
                   ResType: PChar)
```

Visibility: public

Description: Creates a new instance of a resource stream.

TResourceStream.CreateFromID

Synopsis: Creates a new instance of a resource stream with resource

Declaration:
```
constructor CreateFromID(Instance: THandle;ResID: Integer;
                         ResType: PChar)
```

Visibility: public

Description: Creates a new instance of a resource stream with resource

TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the instance of the resource stream.

TResourceStream.Write

Synopsis: `Write` implements the abstract TStream.Write (141) method.

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

Visibility: public

Description: `Write` implements the abstract TStream.Write (141) method.

1.48 TStream

Description

`TStream` is the base class for all streaming classes. It defines abstract methods for reading (141), writing (141) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendent classes such as TMemoryStream (118) or TFileStream (108) then implement these abstract methods to write streams to memory or file.

Method overview

Page	Method	Description
143	CopyFrom	Copy data from one stream to another
145	FixupResourceHeader	Not implemented in FPC
141	Read	Reads data from the stream to a buffer and returns the number of bytes read.
147	ReadAnsiString	Read an ansistring from the stream and return its value.
142	ReadBuffer	Reads data from the stream to a buffer
146	ReadByte	Read a byte from the stream and return its value.
143	ReadComponent	Reads component data from a stream
143	ReadComponentRes	Reads component data and resource header from a stream
146	ReadDWord	Read a DWord from the stream and return its value.
145	ReadResHeader	Read a resource header from the stream.
146	ReadWord	Read a word from the stream and return its value.
142	Seek	Sets the current position in the stream
141	SetSize	Sets the size of the stream
141	Write	Writes data from the stream to the buffer and returns the number of bytes written.
148	WriteAnsiString	Write an ansistring to the stream.
142	WriteBuffer	Writes data from the stream to the buffer
147	WriteByte	Write a byte to the stream.
144	WriteComponent	Write component data to the stream
144	WriteComponentRes	Write resource header and component data to a stream
144	WriteDescendent	Write component data to a stream, relative to an ancestor
145	WriteDescendentRes	Write resource header and component data to a stream, relative to an ancestor
147	WriteDWord	Write a DWord to the stream.
145	WriteResourceHeader	Write resource header to the stream
147	WriteWord	Write a word to the stream.

Property overview

Page	Property	Access	Description
148	Position	rw	The current position in the stream.
148	Size	rw	The current size of the stream.

TStream.SetSize

Synopsis: Sets the size of the stream

Declaration:
```
procedure SetSize(NewSize: LongInt);  Virtual;  Overload
procedure SetSize(NewSize: Int64);  Virtual;  Overload
```

Visibility: protected

Description: `SetSize` is the write handler for the TStream.Size (148) property. The `TStream` implementation of `SetSize` does nothing, but descendent classes may override this methods to allow programmers to set the size of the stream.

See also: TStream.GetSize (140), TStream.Size (148)

TStream.Read

Synopsis: Reads data from the stream to a buffer and returns the number of bytes read.

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

Visibility: public

Description: `Read` attempts to read `Count` from the stream to `Buffer` and returns the number of bytes actually read.

This method should be used when the number of bytes is not determined. If a specific number of bytes is expected, use TSTream.ReadBuffer (142) instead.

`Read` is an abstract method that is overridden by descendent classes to do the actual reading.

Errors: Descendent classes that do not allow reading from the stream may raise an exception when the `Read` is used.

See also: TStream.Write (141), TStream.ReadBuffer (142)

TStream.Write

Synopsis: Writes data from the stream to the buffer and returns the number of bytes written.

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

Visibility: public

Description: `Write` attempts to write `Count` bytes from `Buffer` to the stream. It returns the actual number of bytes written to the stream.

This method should be used when the number of bytes that should be written is not determined. If a specific number of bytes should be written, use TSTream.WriteBuffer (142) instead.

`Write` is an abstract method that is overridden by descendent classes to do the actual writinging.

Errors: Descendent classes that do not allow writing to the stream may raise an exception when `Write` is used.

See also: TStream.Read (141), TStream.WriteBuffer (142)

TStream.Seek

Synopsis: Sets the current position in the stream

Declaration:
```
function Seek(Offset: LongInt;Origin: Word) : LongInt;  Virtual
                ;  Overload
function Seek(Offset: Int64;Origin: TSeekOrigin) : Int64;  Virtual
                ;  Overload
```

Visibility: public

Description: `Seek` sets the position of the stream to `Offset` bytes from `Origin`. `Origin` can have one of the following values:

Table 1.15:

Constant	Meaning
soFromBeginning	Set the position relative to the start of the stream.
soFromCurrent	Set the position relative to the beginning of the stream.
soFromEnd	Set the position relative to the end of the stream.

`Offset` should be negative when the origin is `SoFromEnd`. It should be positive for `soFromBeginning` and can have both signs for `soFromCurrent`

This is an abstract method, which must be overridden by descendent classes. They may choose not to implement this method for all values of `Origin` and `Offset`.

Errors: An exception may be raised if this method is called with an invalid pair of `Offset`,`Origin` values. e.g. a negative `offset` for `soFromBeginning`.

See also: TStream.Position (148)

TStream.ReadBuffer

Synopsis: Reads data from the stream to a buffer

Declaration: `procedure ReadBuffer(var Buffer;Count: LongInt)`

Visibility: public

Description: `ReadBuffer` reads `Count` bytes of the stream into `Buffer`. If the stream does not contain `Count` bytes, then an exception is raised.

`ReadBuffer` should be used to read in a fixed number of bytes, such as when reading structures or the content of variables. If the number of bytes is not determined, use TStream.Read (141) instead. `ReadBuffer` uses `Read` internally to do the actual reading.

Errors: If the stream does not allow to read `Count` bytes, then an exception is raised.

See also: TStream.Read (141), TStream.WriteBuffer (142)

TStream.WriteBuffer

Synopsis: Writes data from the stream to the buffer

Declaration: `procedure WriteBuffer(const Buffer;Count: LongInt)`

Visibility: public

Description: `WriteBuffer` writes `Count` bytes to the stream from `Buffer`. If the stream does not allow `Count` bytes to be written, then an exception is raised.

`WriteBuffer` should be used to read in a fixed number of bytes, such as when writing structures or the content of variables. If the number of bytes is not determined, use TStream.Write (141) instead. `WriteBuffer` uses `Write` internally to do the actual reading.

Errors: If the stream does not allow to write `Count` bytes, then an exception is raised.

See also: TStream.Write (141), TStream.ReadBuffer (142)

TStream.CopyFrom

Synopsis: Copy data from one stream to another

Declaration: `function CopyFrom(Source: TStream;Count: Int64) : Int64`

Visibility: public

Description: `CopyFrom` reads `Count` bytes from `Source` and writes them to the current stream. This updates the current position in the stream. After the action is completed, the number of bytes copied is returned.

This can be used to quickly copy data from one stream to another or to copy the whole contents of the stream.

See also: TStream.Read (141), TStream.Write (141)

TStream.ReadComponent

Synopsis: Reads component data from a stream

Declaration: `function ReadComponent(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponent` reads a component state from the stream and transfers this state to `Instance`. If `Instance` is nil, then it is created first based on the type stored in the stream. `ReadComponent` returns the component as it is read from the stream.

`ReadComponent` simply creates a TReader (128) object and calls its ReadRootComponent (134) method.

Errors: If an error occurs during the reading of the component, an EFilerError (41) exception is raised.

See also: TStream.WriteComponent (144), TStream.ReadComponentRes (143), TReader.ReadRootComponent (134)

TStream.ReadComponentRes

Synopsis: Reads component data and resource header from a stream

Declaration: `function ReadComponentRes(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponentRes` reads a resource header from the stream, and then calls ReadComponent (143) to read the component state from the stream into `Instance`.

This method is usually called by the global streaming method when instantiating forms and datamodules as created by an IDE. It should be used mainly on Windows, to store components in Windows resources.

Errors: If an error occurs during the reading of the component, an EFilerError (41) exception is raised.

See also: TStream.ReadComponent (143), TStream.WriteComponentRes (144)

TStream.WriteComponent

Synopsis: Write component data to the stream

Declaration: `procedure WriteComponent(Instance: TComponent)`

Visibility: public

Description: `WriteComponent` writes the published properties of `Instance` to the stream, so they can later be read with TStream.ReadComponent (143). This method is intended to be used by an IDE, to preserve the state of a form or datamodule as designed in the IDE.

`WriteComponent` simply calls WriteDescendent (144) with `Nil` ancestor.

See also: TStream.ReadComponent (143), TStream.WriteComponentRes (144)

TStream.WriteComponentRes

Synopsis: Write resource header and component data to a stream

Declaration: `procedure WriteComponentRes(const ResName: String;Instance: TComponent)`

Visibility: public

Description: `WriteComponentRes` writes a `ResName` resource header to the stream and then calls WriteComponent (144) to write the published properties of `Instance` to the stream.

This method is intened for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

See also: TStream.WriteComponent (144), TStream.ReadComponentRes (143)

TStream.WriteDescendent

Synopsis: Write component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendent(Instance: TComponent;Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` writes the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

`WriteDescendent` creates a TWriter (179) object and calls its WriteDescendent (182) object. The writer is passed a binary driver object (66) by default.

TStream.WriteDescendentRes

Synopsis: Write resource header and component data to a stream, relative to an ancestor

Declaration:
```
procedure WriteDescendentRes(const ResName: String;Instance: TComponent;
                             Ancestor: TComponent)
```

Visibility: public

Description: `WriteDescendentRes` writes a `ResName` resource header, and then calls WriteDescendent (144) to write the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

This method is intened for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

TStream.WriteResourceHeader

Synopsis: Write resource header to the stream

Declaration:
```
procedure WriteResourceHeader(const ResName: String;
                              var FixupInfo: Integer)
```

Visibility: public

Description: `WriteResourceHeader` writes a resource-file header for a resource called `ResName`. It returns in `FixupInfo` the argument that should be passed on to TStream.FixupResourceHeader (145).

`WriteResourceHeader` should not be used directly. It is called by the TStream.WriteComponentRes (144) and TStream.WriteDescendentRes (145) methods.

See also: TStream.FixupResourceHeader (145), TStream.WriteComponentRes (144), TStream.WriteDescendentRes (145)

TStream.FixupResourceHeader

Synopsis: Not implemented in FPC

Declaration:
```
procedure FixupResourceHeader(FixupInfo: Integer)
```

Visibility: public

Description: `FixupResourceHeader` is used to write the size of the resource after a component was written to stream. The size is determined from the current position, and it is written at position `FixupInfo`. After that the current position is restored.

`FixupResourceHeader` should never be called directly; it is handled by the streaming system.

See also: TStream.WriteResourceHeader (145), TStream.WriteComponentRes (144), TStream.WriteDescendentRes (145)

TStream.ReadResHeader

Synopsis: Read a resource header from the stream.

Declaration:
```
procedure ReadResHeader
```

Visibility: public

Description: `ReadResourceHeader` reads a reasource file header from the stream. It positions the stream just beyond the header.

`ReadResourceHeader` should not be called directly, it is called by the streaming system when needed.

Errors: If the resource header is invalid an EInvalidImage (42) exception is raised.

See also: TStream.ReadComponentRes (143), EInvalidImage (42)

TStream.ReadByte

Synopsis: Read a byte from the stream and return its value.

Declaration: `function ReadByte : Byte`

Visibility: public

Description: `ReadByte` reads one byte from the stream and returns its value.

Errors: If the byte cannot be read, a EStreamError (43) exception will be raised. This is a utility function which symply calls the Read (141) function.

See also: TStream.Read (141), TStream.WriteByte (147), TStream.ReadWord (146), TStream.ReadDWord (146), TStream.ReadAnsiString (147)

TStream.ReadWord

Synopsis: Read a word from the stream and return its value.

Declaration: `function ReadWord : Word`

Visibility: public

Description: `ReadWord` reads one Word (i.e. 2 bytes) from the stream and returns its value. This is a utility function which symply calls the Read (141) function.

Errors: If the word cannot be read, a EStreamError (43) exception will be raised.

See also: TStream.Read (141), TStream.WriteWord (147), TStream.ReadByte (146), TStream.ReadDWord (146), TStream.ReadAnsiString (147)

TStream.ReadDWord

Synopsis: Read a DWord from the stream and return its value.

Declaration: `function ReadDWord : Cardinal`

Visibility: public

Description: `ReadDWord` reads one DWord (i.e. 4 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (141) function.

Errors: If the DWord cannot be read, a EStreamError (43) exception will be raised.

See also: TStream.Read (141), TStream.WriteDWord (147), TStream.ReadByte (146), TStream.ReadWord (146), TStream.ReadAnsiString (147)

TStream.ReadAnsiString

Synopsis: Read an ansistring from the stream and return its value.

Declaration: `function ReadAnsiString : String`

Visibility: public

Description: `ReadAnsiString` reads an ansistring from the stream and returns its value. This is a utility function which simply calls the read function several times. The Ansistring should be stored as 4 bytes (a DWord) representing the length of the string, and then the string value itself. The WriteAnsiString (148) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a EStreamError (43) exception will be raised.

See also: TStream.Read (141), TStream.WriteAnsiString (148), TStream.ReadByte (146), TStream.ReadWord (146), TStream.ReadDWord (146)

TStream.WriteByte

Synopsis: Write a byte to the stream.

Declaration: `procedure WriteByte(b: Byte)`

Visibility: public

Description: `WriteByte` writes the byte `B` to the stream. This is a utility function which simply calls the Write (141) function. The byte can be read from the stream using the ReadByte (146) function.

Errors: If an error occurs when attempting to write, an EStreamError (43) exception will be raised.

See also: TStream.Write (141), TStream.ReadByte (146), TStream.WriteWord (147), TStream.WriteDWord (147), TStream.WriteAnsiString (148)

TStream.WriteWord

Synopsis: Write a word to the stream.

Declaration: `procedure WriteWord(w: Word)`

Visibility: public

Description: `WriteWord` writes the word `W` (i.e. 2 bytes) to the stream. This is a utility function which simply calls the Write (141) function. The word can be read from the stream using the ReadWord (146) function.

Errors: If an error occurs when attempting to write, an EStreamError (43) exception will be raised.

See also: TStream.Write (141), TStream.ReadWord (146), TStream.WriteByte (147), TStream.WriteDWord (147), TStream.WriteAnsiString (148)

TStream.WriteDWord

Synopsis: Write a DWord to the stream.

Declaration: `procedure WriteDWord(d: Cardinal)`

Visibility: public

Description: `WriteDWord` writes the DWord `D` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (141) function. The DWord can be read from the stream using the ReadDWord (146) function.

Errors: If an error occurs when attempting to write, an EStreamError (43) exception will be raised.

See also: TStream.Write (141), TStream.ReadDWord (146), TStream.WriteByte (147), TStream.WriteWord (147), TStream.WriteAnsiString (148)

TStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(S: String)`

Visibility: public

Description: `WriteAnsiString` writes the AnsiString `S` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the Write (141) function. The ansistring is written as a 4 byte length specifier, followed by the ansistring's content. The ansistring can be read from the stream using the ReadAnsiString (147) function.

Errors: If an error occurs when attempting to write, an EStreamError (43) exception will be raised.

See also: TStream.Write (141), TStream.ReadAnsiString (147), TStream.WriteByte (147), TStream.WriteWord (147), TStream.WriteDWord (147)

TStream.Position

Synopsis: The current position in the stream.

Declaration: `Property Position : Int64`

Visibility: public

Access: Read,Write

Description: `Position` can be read to determine the current position in the stream. It can be written to to set the (absolute) position in the stream. The position is zero-based, so to set the position at the beginning of the stream, the position must be set to zero.

Remark: Not all `TStream` descendants support setting the position in the stream, so this should be used with care.

Errors: Some descendents may raise an EStreamError (43) exception if they do not support setting the stream position.

See also: TStream.Size (148), TStream.Seek (142)

TStream.Size

Synopsis: The current size of the stream.

Declaration: `Property Size : Int64`

Visibility: public

Access: Read,Write

Description: `Size` can be read to determine the stream size or to set the stream size.

Remark: Not all descendents of `TStream` support getting or setting the stream size; they may raise an exception if the `Size` property is read or set.

See also: TStream.Position (148), TStream.Seek (142)

1.49 TStringList

Description

`TStringList` is a descendent class of TStrings (155) that implements all of the abstract methods introduced there. It also introduces some additional methods:

- Sort the list, or keep the list sorted at all times
- Special handling of duplicates in sorted lists
- Notification of changes in the list

Method overview

Page	Method	Description
152	Add	Implements the TStrings.Add (160) function.
150	Changed	Called when the list of strings was modified.
150	Changing	Called when the list is changing.
152	Clear	Implements the TStrings.Add (160) function.
154	CustomSort	
152	Delete	Implements the TStrings.Delete (162) function.
152	Destroy	Destroys the stringlist.
153	Exchange	Implements the TStrings.Exchange (163) function.
153	Find	Locates the index for a given string in sorted lists.
150	Get	Overrides the standard read handler for the TStrings.Strings (170) property.
150	GetCapacity	Overrides the standard read handler for the TStrings.Capacity (167) property.
150	GetCount	Overrides the standard read handler for the TStrings.Count (168) property.
151	GetObject	Overrides the standard read handler for the TStrings.Objects (169) property.
153	IndexOf	Overrides the TStrings.IndexOf (163) property.
153	Insert	Overrides the TStrings.Insert (164) method.
151	Put	Overrides the standard write handler for the TStrings.Strings (170) property.
151	PutObject	Overrides the standard write handler for the TStrings.Objects (169) property.
151	SetCapacity	Overrides the standard write handler for the TStrings.Capacity (167) property.
151	SetUpdateState	Overrides the standard TStrings.SetUpdateState (160) call.
154	Sort	Sorts the strings in the list.

Property overview

Page	Property	Access	Description
154	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
155	OnChange	rw	Event triggered after the list was modified.
155	OnChanging	rw	Event triggered when the list is about to be modified.
154	Sorted	rw	Determines whether the list is sorted or not.

TStringList.Changed

Synopsis: Called when the list of strings was modified.

Declaration: `procedure Changed;  Virtual`

Visibility: protected

Description: Called when the list of strings was modified.

TStringList.Changing

Synopsis: Called when the list is changing.

Declaration: `procedure Changing;  Virtual`

Visibility: protected

Description: Called when the list is changing.

TStringList.Get

Synopsis: Overrides the standard read handler for the TStrings.Strings (170) property.

Declaration: `function Get(Index: Integer) : String;  Override`

Visibility: protected

Description: Overrides the standard read handler for the TStrings.Strings (170) property.

TStringList.GetCapacity

Synopsis: Overrides the standard read handler for the TStrings.Capacity (167) property.

Declaration: `function GetCapacity : Integer;  Override`

Visibility: protected

Description: Overrides the standard read handler for the TStrings.Capacity (167) property.

TStringList.GetCount

Synopsis: Overrides the standard read handler for the TStrings.Count (168) property.

Declaration: `function GetCount : Integer;  Override`

Visibility: protected

Description: Overrides the standard read handler for the TStrings.Count (168) property.

TStringList.GetObject

Synopsis: Overrides the standard read handler for the TStrings.Objects (169) property.

Declaration: `function GetObject(Index: Integer) : TObject;  Override`

Visibility: protected

Description: Overrides the standard read handler for the TStrings.Objects (169) property.

TStringList.Put

Synopsis: Overrides the standard write handler for the TStrings.Strings (170) property.

Declaration: `procedure Put(Index: Integer;const S: String);  Override`

Visibility: protected

Description: Overrides the standard write handler for the TStrings.Strings (170) property.

TStringList.PutObject

Synopsis: Overrides the standard write handler for the TStrings.Objects (169) property.

Declaration: `procedure PutObject(Index: Integer;AObject: TObject);  Override`

Visibility: protected

Description: Overrides the standard write handler for the TStrings.Objects (169) property.

TStringList.SetCapacity

Synopsis: Overrides the standard write handler for the TStrings.Capacity (167) property.

Declaration: `procedure SetCapacity(NewCapacity: Integer);  Override`

Visibility: protected

Description: Overrides the standard write handler for the TStrings.Capacity (167) property.

TStringList.SetUpdateState

Synopsis: Overrides the standard TStrings.SetUpdateState (160) call.

Declaration: `procedure SetUpdateState(Updating: Boolean);  Override`

Visibility: protected

Description: Overrides the standard TStrings.SetUpdateState (160) call.

TStringList.Destroy

Synopsis: Destroys the stringlist.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` clears the stringlist, release all memory allocated for the storage of the strings, and then calls the inherited destroy method.

Remark: Any objects associated to strings in the list will *not* be destroyed; it is the responsability of the caller to destroy all objects associated with strings in the list.

TStringList.Add

Synopsis: Implements the TStrings.Add (160) function.

Declaration: `function Add(const S: String) : Integer;  Override`

Visibility: public

Description: `Add` will add `S` to the list. If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (154) is `dupError` then an EStringListError (43) exception is raised. If `Duplicates` is set to `dupIgnore` then the return value is underfined.

If the list is sorted, new strings will not necessarily be added to the end of the list, rather they will be inserted at their alphabetical position.

Errors: If the list is sorted and the string `S` is already present in the list and TStringList.Duplicates (154) is `dupError` then an EStringListError (43) exception is raised.

See also: TStringList.Insert (153), TStringList.Duplicates (154)

TStringList.Clear

Synopsis: Implements the TStrings.Add (160) function.

Declaration: `procedure Clear;  Override`

Visibility: public

Description: Implements the TStrings.Add (160) function.

TStringList.Delete

Synopsis: Implements the TStrings.Delete (162) function.

Declaration: `procedure Delete(Index: Integer);  Override`

Visibility: public

Description: Implements the TStrings.Delete (162) function.

TStringList.Exchange

Synopsis: Implements the TStrings.Exchange (163) function.

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

Visibility: public

Description: `Exchange` will exchange two items in the list as described in TStrings.Exchange (163).

Remark: `Exchange` will not check whether the list os sorted or not; if `Exchange` is called on a sorted list and the strings are not identical, the sort order of the list will be destroyed.

See also: TStringList.Sorted (154), TStrings.Exchange (163)

TStringList.Find

Synopsis: Locates the index for a given string in sorted lists.

Declaration: `function Find(const S: String;var Index: Integer) : Boolean;  Virtual`

Visibility: public

Description: `Find` returns `True` if the string `S` is present in the list. Upon exit, the `Index` parameter will contain the position of the string in the list. If the string is not found, the function will return `False` and `Index` will contain the position where the string will be inserted if it is added to the list.

Remark:

1.Use this method only on sorted lists. For unsorted lists, use TStringList.IndexOf (153) instead.
2.Find uses a binary search method to locate the string

TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf (163) property.

Declaration: `function IndexOf(const S: String) : Integer;  Override`

Visibility: public

Description: `IndexOf` overrides the ancestor method TStrings.indexOf (163). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of `S` if it is found in the list, or -1 if the string is not found in the list.

See also: TStrings.IndexOf (163), TStringList.Find (153)

TStringList.Insert

Synopsis: Overrides the TStrings.Insert (164) method.

Declaration: `procedure Insert(Index: Integer;const S: String);  Override`

Visibility: public

Description: `Insert` will insert the string `S` at position `Index` in the list. If the list is sorted, an EStringListError (43) exception will be raised instead. `Index` is a zero-based position.

Errors: If `Index` contains an invalid value (less than zero or larger than `Count`, or the list is sorted, an EStringListError (43) exception will be raised.

See also: TStringList.Add (152), TStrings.Insert (164), TStringList.InsertObject (149)

TStringList.Sort

Synopsis: Sorts the strings in the list.

Declaration: `procedure Sort;  Virtual`

Visibility: public

Description: `Sort` will sort the strings in the list using the quicksort algorithm. If the list has its TStringList.Sorted (154) property set to `True` then nothing will be done.

See also: TStringList.Sorted (154)

TStringList.CustomSort

Declaration: `procedure CustomSort(CompareFn: TStringListSortCompare)`

Visibility: public

TStringList.Duplicates

Synopsis: Describes the behaviour of a sorted list with respect to duplicate strings.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what to do in case a duplicate value is added to the list:

Table 1.16:

dupIgnore	Duplicate values will not be be added to the list, but no error will be triggered.
dupError	If an attempt is made to add a duplicate value to the list, an EStringListError (43) exception is raised.
dupAccept	Duplicate values can be added to the list.

If the stringlist is not sorted, the `Duplicates` setting is ignored.

TStringList.Sorted

Synopsis: Determines whether the list is sorted or not.

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to `True` in order to cause the list of strings to be sorted. Further additions to the list will be inserted at the correct position so the list remains sorted at all times. Setting the property to `False` has no immediate effect, but will allow strings to be inserted at any position.

Remark:

1. When `Sorted` is `True`, TStringList.Insert (153) cannot be used. For sorted lists, TStringList.Add (152) should be used instead.

2.If `Sorted` is `True`, the TStringList.Duplicates (154) setting has effect. This setting is ignored when `Sorted` is `False`.

See also: TStringList.Sort (154), TStringList.Duplicates (154), TStringList.Add (152), TstringList.Insert (153)

TStringList.OnChange

Synopsis: Event triggered after the list was modified.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChange` can be assigned to respond to changes that have occurred in the list. The handler is called whenever strings are added, moved, modified or deleted from the list.

The `Onchange` event is triggered after the modification took place. When the modification is about to happen, an TstringList.OnChanging (155) event occurs.

See also: TStringList.OnChanging (155)

TStringList.OnChanging

Synopsis: Event triggered when the list is about to be modified.

Declaration: `Property OnChanging : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnChanging` can be assigned to respond to changes that will occurred in the list. The handler is called whenever strings will be added, moved, modified or deleted from the list.

The `Onchanging` event is triggered before the modification will take place. When the modification has happened, an TstringList.OnChange (155) event occurs.

See also: TStringList.OnChange (155)

1.50 TStrings

Description

`TStrings` implements an abstract class to manage an array of strings. It introduces methods to set and retrieve strings in the array, searching for a particular string, concatenating the strings and so on. It also allows an arbitrary object to be associated with each string.

It also introduces methods to manage a series of `name=value` settings, as found in many configuration files.

An instance of `TStrings` is never created directly, instead a descendent class such as TStringList (149) should be created. This is because `TStrings` is an abstract class which does not implement all methods; `TStrings` also doesn't store any strings, this is the functionality introduced in descendents such as TStringList (149).

Method overview

Page	Method	Description
160	Add	Add a string to the list
160	AddObject	Add a string and associated object to the list.
161	AddStrings	Add contents of another stringlist to this list.
161	Append	Add a string to the list.
161	Assign	Assign the contents of another stringlist to this one.
161	BeginUpdate	Mark the beginning of an update batch.
162	Clear	Removes all strings and associated objects from the list.
157	DefineProperties	Method to stream the contents of the string collection
162	Delete	Delete a string from the list.
160	Destroy	Frees all strings and objects, and removes the list from memory.
162	EndUpdate	Mark the end of an update batch.
163	Equals	Compares the contents of two stringlists.
157	Error	Raises an EStringListError (43) exception.
163	Exchange	Exchanges two strings in the list.
157	Get	Abstract read handler for the TStrings.Strings (170) property.
157	GetCapacity	Abstract Read handler for the TStrings.Capacity (167) property.
158	GetCount	Abstract read handler for the TStrings.Count (168) property.
158	GetObject	Abstract read handler for the TStrings.Objects (169) property.
163	GetText	Returns the contents as a PChar
158	GetTextStr	Read handler for the TStrings.Text (170) property.
163	IndexOf	Find a string in the list and return its position.
164	IndexOfName	Finds the index of a name in the name-value pairs.
164	IndexOfObject	Finds an object in the list and returns its index.
164	Insert	Insert a string in the list.
165	InsertObject	Insert a string and associated object in the list.
165	LoadFromFile	Load the contents of a file as a series of strings.
165	LoadFromStream	Load the contents of a stream as a series of strings.
166	Move	Move a string from one place in the list to another.
158	Put	Write handler for the TStrings.Strings (170) property.
159	PutObject	Write handler for the TStrings.Objects (169) property.
166	SaveToFile	Save the contents of the list to a file.
167	SaveToStream	Save the contents of the string to a stream.
159	SetCapacity	Write handler for the TStrings.Capacity (167) property.
167	SetText	Set the contents of the list from a PChar.
159	SetTextStr	Write handler for the TStrings.Text (170) property.
160	SetUpdateState	Sets the update state.

Property overview

Page	Property	Access	Description
167	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
167	CommaText	rw	Contents of the list as a comma-separated string.
168	Count	r	Number of strings in the list.
168	Names	r	Name parts of the name-value pairs in the list.
169	Objects	rw	Indexed access to the objects associated with the strings in the list.
170	Strings	rw	Indexed access to teh strings in the list.
170	StringsAdapter	rw	Not implemented in Free Pascal.
170	Text	rw	Contents of the list as one big string.
169	Values	rw	Value parts of the name-value pairs in the list.

TStrings.DefineProperties

Synopsis: Method to stream the contents of the string collection

Declaration: `procedure DefineProperties(Filer: TFiler);  Override`

Visibility: protected

Description: `DefineProperties` allows the contents of the string collection to be streamed. As such, it overrides TPersistent.DefineProperties (126)

See also: TPersistent.DefineProperties (126)

TStrings.Error

Synopsis: Raises an EStringListError (43) exception.

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

Visibility: protected

Description: `Error` raises an EStringListError (43) exception. It passes `Msg` as a format with `Data` as the only argument.

This method can be used by descendent objects to raise an error.

See also: EStringListError (43)

TStrings.Get

Synopsis: Abstract read handler for the TStrings.Strings (170) property.

Declaration: `function Get(Index: Integer) : String;  Virtual;  Abstract`

Visibility: protected

Description: `Get` is the abstract read handler for the TStrings.Strings (170) property. This is an abstract method, hence it is not implemented in `TStrings`.

Descendent classes, such as TStringList (149) must override this method and implement a routine that retrieves the `Index`-th string in the list. `Index` should have a value between `0` and `Count-1`, in all other cases an error should be raised using TStrings.Error (157).

See also: TStrings.Strings (170), TStrings.Put (158), TStrings.GetObject (158)

TStrings.GetCapacity

Synopsis: Abstract Read handler for the TStrings.Capacity (167) property.

Declaration: `function GetCapacity : Integer;  Virtual`

Visibility: protected

Description: `GetCapacity` is the read handler for the TStrings.Capacity (167) property. The implementation in `TStrings` will return 0.

Descendent classes can override this method. It should return the current number of strings that can be held by the stringlist before it attempts to expand it's storage space.

See also: TStrings.Capacity (167), TStrings.SetCapacity (159)

TStrings.GetCount

Synopsis: Abstract read handler for the TStrings.Count (168) property.

Declaration: `function GetCount : Integer;  Virtual;  Abstract`

Visibility: protected

Description: `GetCount` is the abstract read handler for the TStrings.Count (168) property. This is an abstract method, hence it is not implemented in `TStrings`.

Descendent classes must override this method. It should return the current number of strings in the list. (empty strings included).

See also: TStrings.Count (168)

TStrings.GetObject

Synopsis: Abstract read handler for the TStrings.Objects (169) property.

Declaration: `function GetObject(Index: Integer) : TObject;  Virtual`

Visibility: protected

Description: `GetObject` is the read handler for the TStrings.Objects (169) property. The `TStrings` implementation of this method ignores the `Index` argument and simply returns `Nil`.

Descendent classes that should support object storage should override this method and return the object associated to the `Index`-th string in the list. `Index` should have a value between `0` and `Count-1`. If `Index` is outside the allowed range, an error should be raised using TStrings.Error (157).

See also: TStrings.Objects (169), TStrings.PutObject (159), TStrings.Get (157)

TStrings.GetTextStr

Synopsis: Read handler for the TStrings.Text (170) property.

Declaration: `function GetTextStr : String;  Virtual`

Visibility: protected

Description: `GetTextStr` is the read handler for the TStrings.Text (170) property. It simply concatenates all strings in the list with a linefeed between them, and returns the resulting string.

Descendent classes may override this method to implement an efficienter algorithm which is more suitable to their storage method.

See also: TStrings.Text (170), TStrings.SetTextStr (159)

TStrings.Put

Synopsis: Write handler for the TStrings.Strings (170) property.

Declaration: `procedure Put(Index: Integer;const S: String);  Virtual`

Visibility: protected

Description: `Put` is the write handler for the TStrings.Strings (170) property. It does this by saving the object associated to the `Index`-th string, deleting the `Index`-th string, and inserting `S` and the saved object at position `Index` with TStrings.InsertObject (165)

Descendent classes may wish to override `Put` to implement a more efficient method.

See also: TStrings.Strings (170), TStrings.Get (157), TStrings.PutObject (159)

TStrings.PutObject

Synopsis: Write handler for the TStrings.Objects (169) property.

Declaration: `procedure PutObject(Index: Integer;AObject: TObject);  Virtual`

Visibility: protected

Description: `PutObject` is the write handler for the TStrings.Objects (169) property. The `TStrings` implementation of `PutObject` does nothing.

Descendent objects that should support Object storage must override this method to store the `AObject` so that it is associated with the `Index`-th string in the list. `Index` should have a value between `0` and `Count-1`. If the value of `Index` is out of range, an error should be raised using TStrings.Error (157).

See also: TStrings.Objects (169), TStrings.GetObject (158), TStrings.Put (158)

TStrings.SetCapacity

Synopsis: Write handler for the TStrings.Capacity (167) property.

Declaration: `procedure SetCapacity(NewCapacity: Integer);  Virtual`

Visibility: protected

Description: `SetCapacity` is the write handler for the TStrings.Capacity (167) property. The `TStrings` implementation of `SetCapacity` does nothing.

Descendent classes can override this method to set the current capacity of the stringlist to `NewCapacity`. The capacity is the number of strings the list can hold before it tries to expand its storage space. `NewCapacity` should be no less than `Count`.

See also: TStrings.Capacity (167), TStrings.GetCapacity (157)

TStrings.SetTextStr

Synopsis: Write handler for the TStrings.Text (170) property.

Declaration: `procedure SetTextStr(const Value: String);  Virtual`

Visibility: protected

Description: `SetTextStr` is the write method for the TStrings.Text (170) property. It does nothing other than calling TStrings.SetText (167).

Descendent classes may override this method to implement a more efficient algoritm that fits their storage method better.

See also: TStrings.Text (170), TStrings.GetTextStr (158)

TStrings.SetUpdateState

Synopsis: Sets the update state.

Declaration: `procedure SetUpdateState(Updating: Boolean);  Virtual`

Visibility: protected

Description: `SetUpdateState` sets the update state to `Updating`. The `TStrings` implementation of `SetUpdateState` does nothing.

Descendent objects may override this method to implement optimizations. If `Updating` is `True` then the list of strings is about to be updated (possibly many times). If it is `False` no more updates will take place till the next `SetUpdateState` call.

See also: TStrings.BeginUpdate (161), TStrings.EndUpdate (162)

TStrings.Destroy

Synopsis: Frees all strings and objects, and removes the list from memory.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: `Destroy` is the destructor of `TStrings` it does nothing except calling the inherited destructor.

TStrings.Add

Synopsis: Add a string to the list

Declaration: `function Add(const S: String) : Integer;  Virtual`

Visibility: public

Description: Add adds S at the end of the list and returns the index of S in the list (which should equal Tstrings.Count (168))

See also: TStrings.Items (155), TStrings.AddObject (160), TStrings.Insert (164), TStrings.Delete (162), TStrings.Strings (170), TStrings.Count (168)

TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration: `function AddObject(const S: String;AObject: TObject) : Integer;  Virtual`

Visibility: public

Description: `AddObject` adds S to the list of strings, and associates `AObject` with it. It returns the index of S.

Remark: An object added to the list is not automatically destroyed by the list of the list is destroyed or the string it is associated with is deleted. It is the responsibility of the application to destroy any objects associated with strings.

See also: TStrings.Add (160), Tstrings.Items (155), TStrings.Objects (169), Tstrings.InsertObject (165)

TStrings.Append

Synopsis: Add a string to the list.

Declaration: `procedure Append(const S: String)`

Visibility: public

Description: `Append` does the same as TStrings.Add (160), only it does not return the index of the inserted string.

See also: TStrings.Add (160)

TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

Declaration: `procedure AddStrings(TheStrings: TStrings);  Virtual`

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: TStrings.Add (160), TStrings.Assign (161)

TStrings.Assign

Synopsis: Assign the contents of another stringlist to this one.

Declaration: `procedure Assign(Source: TPersistent);  Override`

Visibility: public

Description: `Assign` replaces the contents of the stringlist with the contents of `Source` if `Source` is also of type `TStrings`. Any associated objects are copied as well.

See also: TStrings.Add (160), TStrings.AddStrings (161), TPersistent.Assign (127)

TStrings.BeginUpdate

Synopsis: Mark the beginning of an update batch.

Declaration: `procedure BeginUpdate`

Visibility: public

Description: `BeginUpdate` increases the update count by one. It is advisable to call `BeginUpdate` before lengthy operations on the stringlist. At the end of these operation, TStrings.EndUpdate (162) should be called to mark the end of the operation. Descendent classes may use this information to perform optmizations. e.g. updating the screen only once after many strings were added to the list.

All `TStrings` methods that modify the string list call `BeginUpdate` before the actual operation, and call `endUpdate` when the operation is finished. Descendent classes should also call these methods when modifying the string list.

Remark: Always put the corresponding call to TStrings.EndUpdate (162) in the context of a `Finally` block, to ensure that the update count is always descreased at the end of the operation, even if an exception occurred:

```
With MyStrings do
    try
      BeginUpdate;
      // Some lengthy operation.
    Finally
      EndUpdate
    end;
```

See also: TStrings.EndUpdate (162)

TStrings.Clear

Synopsis: Removes all strings and associated objects from the list.

Declaration: `procedure Clear;  Virtual;  Abstract`

Visibility: public

Description: `Clear` will remove all strings and their associated objects from the list. After a call to clear, TStrings.Count (168) is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendent classes such as TStringList (149) implement this method.

See also: TStrings.Items (155), TStrings.Delete (162), TStrings.Count (168),

TStrings.Delete

Synopsis: Delete a string from the list.

Declaration: `procedure Delete(Index: Integer);  Virtual;  Abstract`

Visibility: public

Description: `Delete` deletes the string at position `Index` from the list. The associated object is also removed from the list, but not destroyed. `Index` is zero-based, and should be in the range `0` to `Count-1`.

Since it is an abstract method, `TStrings` itself does not implement `Delete`. Descendent classes such as TStringList (149) implement this method.

Errors: If `Index` is not in the allowed range, an EStringListError (43) is raised.

See also: TStrings.Insert (164), TStrings.Items (155), TStrings.Clear (162)

TStrings.EndUpdate

Synopsis: Mark the end of an update batch.

Declaration: `procedure EndUpdate`

Visibility: public

Description: `EndUpdate` should be called at the end of a lengthy operation on the stringlist, but only if there was a call to `BeginUpdate` before the operation was started. It is best to put the call to `EndUpdate` in the context of a `Finally` block, so it will be called even if an exception occurs.

For more information, see TStrings.BeginUpdate (161).

See also: TStrings.BeginUpdate (161)

TStrings.Equals

Synopsis: Compares the contents of two stringlists.

Declaration: `function Equals(TheStrings: TStrings) : Boolean`

Visibility: public

Description: `Equals` compares the contents of the stringlist with the contents of `TheStrings`. If the contents match, i.e. the stringlist contain an equal amount of strings, and all strings match, then `True` is returned. If the number of strings in the lists is is unequal, or they contain one or more different strings, `False` is returned.

Remark:

1.The strings are compared case-insensitively.

2.The associated objects are not compared

See also: Tstrings.Items (155), TStrings.Count (168), TStrings.Assign (161)

TStrings.Exchange

Synopsis: Exchanges two strings in the list.

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

Visibility: public

Description: `Exchange` exchanges the strings at positions `Index1` and `Index2`. The associated objects are also exchanged.

Both indexes must be in the range of valid indexes, i.e. must have a value between `0` and `Count-1`.

Errors: If either `Index1` or `Index2` is not in the range of valid indexes, an EStringListError (43) exception is raised.

See also: TStrings.Move (166), TStrings.Strings (170), TStrings.Count (168)

TStrings.GetText

Synopsis: Returns the contents as a PChar

Declaration: `function GetText : PChar;  Virtual`

Visibility: public

Description: `GetText` allocates a memory buffer and compies the contents of the stringlist to this buffer as a series of strings, separated by an end-of-line marker. The buffer is zero terminated.

Remark: The caller is responsible for freeing the returned memory buffer.

TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration: `function IndexOf(const S: String) : Integer;  Virtual`

Visibility: public

Description: `IndexOf` searches the list for `S`. The search is case-insensitive. If a matching entry is found, its position is returned. if no matching string is found, `-1` is returned.

Remark:

1.Only the first occurrence of the string is returned.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOfObject (164), TStrings.IndexOfName (164), TStrings.Strings (170)

TStrings.IndexOfName

Synopsis: Finds the index of a name in the name-value pairs.

Declaration: `function IndexOfName(const Name: String) : Integer`

Visibility: public

Description: `IndexOfName` searches in the list of strings for a name-value pair with name part `Name`. If such a pair is found, it returns the index of the pair in the stringlist. If no such pair is found, the function returns `-1`. The search is done case-insensitive.

Remark:

1.Only the first occurrence of a matching name-value pair is returned.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: TStrings.IndexOf (163), TStrings.IndexOfObject (164), TStrings.Strings (170)

TStrings.IndexOfObject

Synopsis: Finds an object in the list and returns its index.

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

Visibility: public

Description: `IndexOfObject` searchs through the list of strings till it find a string associated with `AObject`, and returns the index of this string. If no such string is found, `-1` is returned.

Remark:

1.Only the first occurrence of a string with associated object `AObject` is returned; if more strings in the list can be associated with `AObject`, they will not be found by this routine.

2.The returned position is zero-based, i.e. 0 indicates the first string in the list.

TStrings.Insert

Synopsis: Insert a string in the list.

Declaration: `procedure Insert(Index: Integer;const S: String);  Virtual;  Abstract`

Visibility: public

Description: `Insert` inserts the string `S` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Remark:

1.All methods that add strings to the list use `Insert` to add a string to the list.

2.If the string has an associated object, use TStrings.InsertObject (165) instead.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (43) exception is raised.

See also: TStrings.Add (160), TStrings.InsertObject (165), TStrings.Append (161), TStrings.Delete (162)

TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration: `procedure InsertObject(Index: Integer;const S: String;AObject: TObject)`

Visibility: public

Description: `InsertObject` inserts the string `S` and its associated object`AObject` at position `Index` in the list. `Index` is a zero-based position, and can have values from `0` to `Count`. If `Index` equals `Count` then the string is appended to the list.

Errors: If `Index` is less than zero or larger than `Count` then an EStringListError (43) exception is raised.

See also: TStrings.Insert (164), TStrings.AddObject (160), TStrings.Append (161), TStrings.Delete (162)

TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration: `procedure LoadFromFile(const FileName: String);  Virtual`

Visibility: public

Description: `LoadFromFile` loads the contents of a file into the stringlist. Each line in the file (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

`LoadFromFile` simply creates a file stream (108) with the given filename, and then executes TStrings.LoadfromStream (165); after that the file stream object is destroyed again.

See also: TStrings.LoadFromStream (165), TStrings.SaveToFile (166), Tstrings.SaveToStream (167)

TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration: `procedure LoadFromStream(Stream: TStream);  Virtual`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the stringlist. Each line in the stream (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

See also: TStrings.LoadFromFile (165), TStrings.SaveToFile (166), Tstrings.SaveToStream (167)

TStrings.Move

Synopsis: Move a string from one place in the list to another.

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

Visibility: public

Description: `Move` moves the string at position `CurIndex` so it has position `NewIndex` after the move operation. The object associated to the string is also moved. `CurIndex` and `NewIndex` should be in the range of `0` to `Count-1`.

Remark: `NewIndex` is *not* the position in the stringlist before the move operation starts. The move operation

1. removes the string from position `CurIndex`
2. inserts the string at position `NewIndex`

This may not lead to the desired result if `NewIndex` is bigger than `CurIndex`. Consider the following example:

```
With MyStrings do
    begin
    Clear;
    Add('String 0');
    Add('String 1');
    Add('String 2');
    Add('String 3');
    Add('String 4');
    Move(1,3);
    end;
```

After the `Move` operation has completed, 'String 1' will be between 'String 3' and 'String 4'.

Errors: If either `CurIndex` or `NewIndex` is outside the allowed range, an EStringListError (43) is raised.

See also: TStrings.Exchange (163)

TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration: `procedure SaveToFile(const FileName: String);  Virtual`

Visibility: public

Description: `SaveToFile` saves the contents of the stringlist to the file with name `FileName`. It writes the strings to the file, separated by end-of-line markers, so each line in the file will contain 1 string from the stringlist.

`SaveToFile` creates a file stream (108) with name `FileName`, calls TStrings.SaveToStream (167) and then destroys the file stream object.

Errors: An EStreamError (43) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: TStrings.SaveToStream (167), Tstrings.LoadFromStream (165), TStrings.LoadFromFile (165)

TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration: `procedure SaveToStream(Stream: TStream);  Virtual`

Visibility: public

Description: `SaveToStream` saves the contents of the stringlist to `Stream`. It writes the strings to the stream, separated by end-of-line markers, so each 'line' in the stream will contain 1 string from the stringlist.

Errors: An EStreamError (43) exception can be raised if the stream cannot be written to.

See also: TStrings.SaveToFile (166), Tstrings.LoadFromStream (165), TStrings.LoadFromFile (165)

TStrings.SetText

Synopsis: Set the contents of the list from a PChar.

Declaration: `procedure SetText(TheText: PChar);  Virtual`

Visibility: public

Description: `SetText` parses the contents of `TheText` and fills the stringlist based on the contents. It regards `TheText` as a series of strings, separated by end-of-line markers. Each of these strings is added to the stringlist.

See also: TStrings.Text (170)

TStrings.Capacity

Synopsis: Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of strings that the list can hold before it tries to allocate more memory.

`TStrings` returns TStrings.Count (168) when read. Trying to set the capacity has no effect. Descendent classes such as TStringlist (149) can override this property such that it actually sets the new capacity.

See also: TStringList (149), TStrings.Count (168)

TStrings.CommaText

Synopsis: Contents of the list as a comma-separated string.

Declaration: `Property CommaText : String`

Visibility: public

Access: Read,Write

Description: `CommaText` represents the stringlist as a single string, consisting of a comma-separated concatenation of the strings in the list. If one of the strings contains spaces, comma's or quotes it will be enclosed by double quotes. Any double quotes in a string will be doubled. For instance the following strings:

```
Comma,string
Quote"string
Space string
NormalSttring
```

is converted to

```
"Comma,string","Quote""String","Space string",NormalString
```

Conversely, when setting the `CommaText` property, the text will be parsed according to the rules outlined above, and the strings will be set accordingly. Note that spaces will in this context be regarded as string separators, unless the string as a whole is contained in double quotes. Spaces that occur next to a delimiter will be ignored. The following string:

```
"Comma,string" , "Quote""String",Space string,, NormalString
```

Will be converted to

```
Comma,String
Quote"String
Space
String

NormalString
```

See also: TStrings.Text (170), TStrings.SetText (167)

TStrings.Count

Synopsis: Number of strings in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` is the current number of strings in the list. `TStrings` does not implement this property; descendent classes should override the property read handler to return the correct value.

Strings in the list are always uniquely identified by their `Index`; the index of a string is zero-based, i.e. it's supported range is `0` to `Count-1`. trying to access a string with an index larger than or equal to `Count` will result in an error. Code that iterates over the list in a stringlist should always take into account the zero-based character of the list index.

See also: TStrings.Strings (170), TStrings.Objects (169), TStrings.Capacity (167)

TStrings.Names

Synopsis: Name parts of the name-value pairs in the list.

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

Visibility: public

Access: Read

Description: `Names` provides indexed access to the names of teh name-value pairs in the list. It returns the name part of the `Index`-th string in the list.

Remark: The index is not an index based on the number of name-value pairs in the list. It is the name part of the name-value pair a string `Index` in the list. If the string at position `Index` is not a name-value pair (i.e. does not contain the equal sign (=)), then an empty name is returned.

See also: TStrings.Values (169), TStrings.IndexOfName (164)

TStrings.Objects

Synopsis: Indexed access to the objects associated with the strings in the list.

Declaration: `Property Objects[Index: Integer]: TObject`

Visibility: public

Access: Read,Write

Description: `Objects` provides indexed access to the objects associated to the strings in the list. `Index` is a zero-based index and must be in the range of `0` to `Count-1`.

Setting the `objects` property will not free the previously associated object, if there was one. The caller is repsonsible for freeing the object that was previously associated to the string.

`TStrings` does not implement any storage for objects. Reading the `Objects` property will always return `Nil`, Setting the property will have no effect. It is the responsability of the descendent classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an EStringListError (43) exception will be raised.

See also: TStrings.Strings (170), TStrings.IndexOfObject (164), TStrings.Names (168), TStrings.Values (169)

TStrings.Values

Synopsis: Value parts of the name-value pairs in the list.

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

Visibility: public

Access: Read,Write

Description: `Values` represents the value parts of the name-value pairs in the list.

When reading this property, if there is a name-value pair in the list of strings that has name part `Name`, then the corresponding value is returned. If there is no such pair, an empty string is returned.

When writing this value, first it is checked whether there exists a name-value pair in the list with name `Name`. If such a pair is found, it's value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified `Name` and value.

Remark:

1. Names are compared case-insensitively.
2. Any character, including whitespace, up till the first equal (=) sign in a string is considered part of the name.

See also: TStrings.Names (168), TStrings.Strings (170), TStrings.Objects (169)

TStrings.Strings

Synopsis: Indexed access to teh strings in the list.

Declaration: `Property Strings[Index: Integer]: String; default`

Visibility: public

Access: Read,Write

Description: `Strings` is the default property of `TStrings`. It provides indexed read-write access to the list of strings. Reading it will return the string at position `Index` in the list. Writing it will set the string at position `Index`.

`Index` is the position of the string in the list. It is zero-based, i.e. valued values range from `0` (the first string in the list) till `Count-1` (the last string in the list). When browsing through the strings in the list, this fact must be taken into account.

To access the objects associated with the strings in the list, use the TStrings.Objects (169) property. The name parts of name-value pairs can be accessed with the TStrings.Names (168) property, and the values can be set or read through the TStrings.Values (169) property.

Searching through the list can be done using the TStrings.IndexOf (163) method.

Errors: If `Index` is outside the allowed range, an EStringListError (43) exception is raised.

See also: TStrings.Count (168), TStrings.Objects (169), TStrings.Names (168), TStrings.Values (169), TStrings.IndexOf (163)

TStrings.Text

Synopsis: Contents of the list as one big string.

Declaration: `Property Text : String`

Visibility: public

Access: Read,Write

Description: `Text` returns, when read, the contents of the stringlist as one big string consisting of all strings in the list, separated by an end-of-line marker. When this property is set, the string will be cut into smaller strings, based on the positions of end-of-line markers in the string. Any previous content of the stringlist will be lost.

Remark: If any of the strings in the list contains an end-of-line marker, then the resulting string will appear to contain more strings than actually present in the list. To avoid this ambiguity, use the TStrings.CommaText (167) property instead.

See also: TStrings.Strings (170), TStrings.Count (168), TStrings.CommaText (167)

TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: `Property StringsAdapter : IStringsAdapter`

Visibility: public

Access: Read,Write

Description: Not implemented in Free Pascal.

1.51 TStringStream

Description

`TStringStream` stores its data in an ansistring. The contents of this string is available as the DataString (173) property. It also introduces some methods to read or write parts of the stringstream's data as a string.

The main purpose of a `TStringSTream` is to be able to treat a string as a stream from which can be read.

Method overview

Page	Method	Description
171	Create	Creates a new stringstream and sets its initial content.
172	Read	Reads from the stream.
172	ReadString	Reads a string of length `Count`
172	Seek	Sets the position in the stream.
171	SetSize	Sets the size of the stream.
172	Write	`Write` implements the abstract TStream.Write (141) method.
172	WriteString	`WriteString` writes a string to the stream.

Property overview

Page	Property	Access	Description
173	DataString	r	Contains the contents of the stream in string form

TStringStream.SetSize

Synopsis: Sets the size of the stream.

Declaration: `procedure SetSize(NewSize: LongInt);  Override`

Visibility: protected

Description: `SetSize` sets the size of the stream to `newsize`. It does this by setting the size of the ansisting in which the stream is stored. `NewSize` can have any value greater than or equal to zero.

Errors: In case there is not enough memory, an exception may be raised.

See also: TStream.Size (148)

TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

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

Visibility: public

Description: `Create` creates a new `TStringStream` instance and sets its initial content to `Astring`. The position is still 0 but the size of the stream will equal the length of the string.

See also: TStringStream.DataString (173)

TStringStream.Read

Synopsis: Reads from the stream.

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

Visibility: public

Description: `Read` implements the abstract Read (141) from TStream (140). It tries to read `Count` bytes into `Buffer`. It returns the number of bytes actually read. The position of the stream is advanced with the number of bytes actually read; When the reading has reached the end of the DataString (173), then the reading stops, i.e. it is not possible to read beyond the end of the datastring.

See also: TStream.Read (141), TStringStream.Write (172), TStringStream.DataString (173)

TStringStream.ReadString

Synopsis: Reads a string of length `Count`

Declaration: `function ReadString(Count: LongInt) : String`

Visibility: public

Description: `ReadString` reads `Count` bytes from the stream and returns the read bytes as a string. If less than `Count` bytes were available, the string has as many characters as bytes could be read.

The `ReadString` method is a wrapper around the Read (172) method. It does not do the same stringas the TStream.ReadAnsiString (147) method, which first reads a length integer to determine the length of the string to be read.

See also: TStringStream.Read (172), TStream.ReadAnsiString (147)

TStringStream.Seek

Synopsis: Sets the position in the stream.

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

Visibility: public

Description: `Seek` implements the abstract Seek (142) method.

TStringStream.Write

Synopsis: `Write` implements the abstract TStream.Write (141) method.

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

Visibility: public

Description: `Write` implements the abstract TStream.Write (141) method.

TStringStream.WriteString

Synopsis: `WriteString` writes a string to the stream.

Declaration: `procedure WriteString(const AString: String)`

Visibility: public

Description: `WriteString` writes a string to the stream.

TStringStream.DataString

Synopsis: Contains the contents of the stream in string form

Declaration: `Property DataString : String`

Visibility: public

Access: Read

Description: Contains the contents of the stream in string form

1.52 TTextObjectWriter

Description

Not yet implemented.

1.53 TThread

Description

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendent of the `TThread` object and override the Execute (174) method. In this method, the thhread's code should be executed. To run a thread, create an instance of the tthread descendent, and call it's execute method.

Method overview

Page	Method	Description
174	Create	Creates a new thread.
174	Destroy	Destroys the thread object.
174	DoTerminate	Terminates the thread.
174	Execute	Execute method. Should be overridden in a descendent thread.
174	Resume	Resumes the thread's execution.
175	Suspend	Suspends the thread's execution.
174	Synchronize	Synchronizes the thread by executing the method in the main thread.
175	Terminate	Signals the thread it should terminate.
175	WaitFor	Waits for the thread to terminate and returns the exit status.

Property overview

Page	Property	Access	Description
177	FatalException	r	
176	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
176	Handle	r	Returns the thread handle.
177	OnTerminate	rw	Event called when the thread terminates.
176	Priority	rw	Returns the thread priority.
175	ReturnValue	rw	Return value of the thread when it stops executing.
176	Suspended	rw	Indicates whether the thread is suspended.
175	Terminated	r	Indicates whether the Terminate (175)method was called by the user.
176	ThreadID	r	Returns the thread ID.

TThread.DoTerminate

Synopsis: Terminates the thread.

Declaration: `procedure DoTerminate;  Virtual`

Visibility: protected

Description: Terminates the thread.

TThread.Execute

Synopsis: Execute method. Should be overridden in a descendent thread.

Declaration: `procedure Execute;  Virtual;  Abstract`

Visibility: protected

Description: Execute method. Should be overridden in a descendent thread.

TThread.Synchronize

Synopsis: Synchronizes the thread by executing the method in the main thread.

Declaration: `procedure Synchronize(Method: TThreadMethod)`

Visibility: protected

Description: Synchronizes the thread by executing the method in the main thread.

TThread.Create

Synopsis: Creates a new thread.

Declaration: `constructor Create(CreateSuspended: Boolean)`

Visibility: public

Description: Creates a new thread.

TThread.Destroy

Synopsis: Destroys the thread object.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the thread object.

TThread.Resume

Synopsis: Resumes the thread's execution.

Declaration: `procedure Resume`

Visibility: public

Description: Resumes the thread's execution.

TThread.Suspend

Synopsis: Suspends the thread's execution.

Declaration: `procedure Suspend`

Visibility: public

Description: Suspends the thread's execution.

TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: public

Description: Signals the thread it should terminate.

TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: public

Description: Waits for the thread to terminate and returns the exit status.

TThread.ReturnValue

Synopsis: Return value of the thread when it stops executing.

Declaration: `Property ReturnValue : Integer`

Visibility: protected

Access: Read,Write

Description: Return value of the thread when it stops executing.

TThread.Terminated

Synopsis: Indicates whether the Terminate (175)method was called by the user.

Declaration: `Property Terminated : Boolean`

Visibility: protected

Access: Read

Description: Indicates whether the Terminate (175)method was called by the user.

TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether the thread should free itself when it stops executing.

TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : THandle`

Visibility: public

Access: Read

Description: Returns the thread handle.

TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: public

Access: Read,Write

Description: Returns the thread priority.

TThread.Suspended

Synopsis: Indicates whether the thread is suspended.

Declaration: `Property Suspended : Boolean`

Visibility: public

Access: Read,Write

Description: Indicates whether the thread is suspended.

TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : THandle`

Visibility: public

Access: Read

Description: Returns the thread ID.

TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: public

Access: Read,Write

Description: Event called when the thread terminates.

TThread.FatalException

Declaration: `Property FatalException : TObject`

Visibility: public

Access: Read

1.54 TThreadList

Description

This class is not yet implemented in Free Pascal.

Method overview

Page	Method	Description
178	Add	Adds an element to the list.
178	Clear	Removes all ementents from the list.
177	Create	Creates a new thread-safe list.
177	Destroy	Destroys the list instance.
178	LockList	Locks the list for exclusive access.
178	Remove	Removes an item from the list.
179	UnlockList	Unlocks the list after it was locked.

TThreadList.Create

Synopsis: Creates a new thread-safe list.

Declaration: `constructor Create`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Destroy

Synopsis: Destroys the list instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Add

Synopsis: Adds an element to the list.

Declaration: `procedure Add(Item: Pointer)`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Clear

Synopsis: Removes all emements from the list.

Declaration: `procedure Clear`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Remove

Synopsis: Removes an item from the list.

Declaration: `procedure Remove(Item: Pointer)`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.UnlockList

Synopsis: Unlocks the list after it was locked.

Declaration: `procedure UnlockList`

Visibility: public

Description: This class is not yet implemented in Free Pascal.

Errors:

1.55 TWriter

Description

Object to write component data to an arbitrary format.

Method overview

Page	Method	Description
180	Create	Creates a new Writer with a stream and bufsize.
181	DefineBinaryProperty	Callback used when defining and streaming custom properties.
181	DefineProperty	Callback used when defining and streaming custom properties.
180	Destroy	Destroys the writer instance.
180	SetRoot	Sets the root component
180	WriteBinary	Writes binary data to the stream.
181	WriteBoolean	Write boolean value to the stream.
182	WriteChar	Write a character to the stream.
181	WriteCollection	Write a collection to the stream.
181	WriteComponent	Stream a component to the stream.
182	WriteDate	Write a date to the stream.
182	WriteDescendent	Write a descendent component to the stream.
182	WriteFloat	Write a float to the stream.
182	WriteIdent	Write an identifier to the stream.
183	WriteInteger	Write an integer to the stream.
183	WriteListBegin	Write a start-of-list marker to the stream.
183	WriteListEnd	Write an end-of-list marker to the stream.
180	WriteProperties	Writes the published properties to the stream.
180	WriteProperty	Writes one property to the stream.
183	WriteRootComponent	Write a root component to the stream.
182	WriteSingle	Write a single-type real to the stream.
183	WriteString	Write a string to the stream.

Property overview

Page	Property	Access	Description
184	Driver	r	Driver used when writing to the stream.
184	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
184	OnWriteMethodProperty	rw	
184	RootAncestor	rw	Ancestor of root component.

TWriter.SetRoot

Synopsis: Sets the root component

Declaration: `procedure SetRoot(ARoot: TComponent);  Override`

Visibility: protected

TWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(AWriteData: TStreamProc)`

Visibility: protected

Description: Writes binary data to the stream.

TWriter.WriteProperty

Synopsis: Writes one property to the stream.

Declaration: `procedure WriteProperty(Instance: TPersistent;PropInfo: Pointer)`

Visibility: protected

Description: Writes one property to the stream.

TWriter.WriteProperties

Synopsis: Writes the published properties to the stream.

Declaration: `procedure WriteProperties(Instance: TPersistent)`

Visibility: protected

Description: Writes the published properties to the stream.

TWriter.Create

Synopsis: Creates a new Writer with a stream and bufsize.

Declaration:
```
constructor Create(ADriver: TAbstractObjectWriter)
constructor Create(Stream: TStream;BufSize: Integer)
```

Visibility: public

Description: Creates a new Writer with a stream and bufsize.

TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: Destroys the writer instance.

TWriter.DefineProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineProperty(const Name: String;ReadData: TReaderProc;
                        AWriteData: TWriterProc;HasData: Boolean)
                        ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

TWriter.DefineBinaryProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:
```
procedure DefineBinaryProperty(const Name: String;ReadData: TStreamProc;
                              AWriteData: TStreamProc;HasData: Boolean)
                              ;  Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean)`

Visibility: public

Description: Write boolean value to the stream.

TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration: `procedure WriteCollection(Value: TCollection)`

Visibility: public

Description: Write a collection to the stream.

TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration: `procedure WriteComponent(Component: TComponent)`

Visibility: public

Description: Stream a component to the stream.

TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: public

Description: Write a character to the stream.

TWriter.WriteDescendent

Synopsis: Write a descendent component to the stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent;AAncestor: TComponent)`

Visibility: public

Description: Write a descendent component to the stream.

TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: public

Description: Write a float to the stream.

TWriter.WriteSingle

Synopsis: Write a single-type real to the stream.

Declaration: `procedure WriteSingle(const Value: Single)`

Visibility: public

Description: Write a single-type real to the stream.

TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime)`

Visibility: public

Description: Write a date to the stream.

TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String)`

Visibility: public

Description: Write an identifier to the stream.

TWriter.WriteInteger

Synopsis: Write an integer to the stream.

Declaration:
```
procedure WriteInteger(Value: LongInt);  Overload
procedure WriteInteger(Value: Int64);  Overload
```

Visibility: public

Description: Write an integer to the stream.

TWriter.WriteListBegin

Synopsis: Write a start-of-list marker to the stream.

Declaration: `procedure WriteListBegin`

Visibility: public

Description: Write a start-of-list marker to the stream.

TWriter.WriteListEnd

Synopsis: Write an end-of-list marker to the stream.

Declaration: `procedure WriteListEnd`

Visibility: public

Description: Write an end-of-list marker to the stream.

TWriter.WriteRootComponent

Synopsis: Write a root component to the stream.

Declaration: `procedure WriteRootComponent(ARoot: TComponent)`

Visibility: public

Description: Write a root component to the stream.

TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const Value: String)`

Visibility: public

Description: Write a string to the stream.

TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: `Property RootAncestor : TComponent`

Visibility: public

Access: Read,Write

Description: Ancestor of root component.

TWriter.OnFindAncestor

Synopsis: Event occurring when an ancestor component must be found.

Declaration: `Property OnFindAncestor : TFindAncestorEvent`

Visibility: public

Access: Read,Write

Description: Event occurring when an ancestor component must be found.

TWriter.OnWriteMethodProperty

Declaration: `Property OnWriteMethodProperty : TWriteMethodPropertyEvent`

Visibility: public

Access: Read,Write

TWriter.Driver

Synopsis: Driver used when writing to the stream.

Declaration: `Property Driver : TAbstractObjectWriter`

Visibility: public

Access: Read

Description: Driver used when writing to the stream.