Free Pascal Resource support (FCL-res): Reference guide.

Reference guide for FCL-res units.
Document version 3.2.0
June 2020

Giulio Bernardi

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-res (Free Pascal Resource support).

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

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Introduction

This package contains a library to easily work with Microsoft Windows resources in a cross-platform way.

Classes are provided to create, load and write resources from/to different file formats in a transparent way, and to handle most common resource types without having to deal with their internal format.

Whenever possible data caching is performed, helped by a copy-on-write mechanism. This improves performance especially when converting big resources from a file format to another.

Since fcl-res architecture is extensible, it's always possible to extend the library with custom resource types or new file readers/writers.

Please note that resources aren't limited to Windows platform: Free Pascal can use them also on ELF and Mach-O targets. Moreover, this library can be useful for cross-compilation purposes even on other targets.

It is highly recommended to read Basic Usage (**??**) topic if you are approaching this library for the first time.

0.2 Basic Usage

Resource files and TResources class

One of the most important classes is TResources (**??**) class, contained in resource (**??**) unit, which represents a format-independent view of a resource file. In fact, while single resources are important, they are of little use alone, since they can't be read or written to file directly: they need to be contained in a TResources (**??**) object.

TResources (**??**) provides methods to read itself from a file or stream, using specific objects that are able to read resource data from such a stream: these are the so called *resource readers*, that descend from TAbstractResourceReader (**??**).

There are also *resource writers* that do the opposite, and that descend from TAbstractResourceWriter (**??**).

Usually readers and writers register themselves with TResources (**??**) in the `initialization` section of the unit they are implemented in, so you only need to add a certain unit to your program `uses` clause to let TResources (**??**) "know" about a particular file format.

Let's see a very simple example: a program that converts a .res file to an object file in COFF format (the object file format used by Microsoft Windows).

```
program res1;

{$mode objfpc}

uses
  Classes, SysUtils, resource, resreader, coffwriter;

var
  resources : TResources;
begin
  resources:=TResources.Create;
  resources.LoadFromFile('myresource.res');
  resources.WriteToFile('myobject.o');
  resources.Free;
end.
```

As you can see, the code is trivial. Note that `resreader` and `coffwriter` units were added to the `uses` clause of the program: this way, the resource reader for .res files and the resource writer for COFF files have been registered, letting the `resources` object know how to handle these file types.

There are cases where one doesn't want to let the TResources (**??**) object to choose readers and writers by itself. In fact, while generally it is a good idea to let TResources (**??**) probe all readers it knows to find one able to read the input file, this isn't true when it comes to write files: writers are selected based on the file extension, so if you are trying to write a file with .o extension you can't be sure about which writer will be selected: it could be the COFF or the ELF writer (it depends on which writer gets registered first). Moreover, writers generally make an object file for the host architecture, so if you are running the program on a i386 machine it will produce a COFF or ELF file for i386.

The solution is to provide TResources (**??**) with a specific writer. In the following example the reader is automatically chosen among various readers, and we use a specific writer to produce an ELF file for SPARC.

```
program res2;

{$mode objfpc}

uses
  Classes, SysUtils, resource,
  resreader, coffreader, elfreader, winpeimagereader, //readers
  elfwriter, elfconsts;

var
  resources : TResources;
  writer : TElfResourceWriter;
begin
  resources:=TResources.Create;
  resources.LoadFromFile(paramstr(1));
  writer:=TElfResourceWriter.Create;
  writer.MachineType:=emtsparc;
  resources.WriteToFile(ChangeFileExt(paramstr(1),'.o'),writer);
  resources.Free;
  writer.Free;
end.
```

Note that the file to convert is taken from the command line. Its format is automatically detected among res (resreader (**??**)), coff (coffreader (**??**)), elf (elfreader (**??**)), PE (winpeimagereader (**??**), e.g. a Windows exe or dll), and is written as an ELF file for SPARC. Note that we had to use elfconsts (**??**) unit since we used emtsparc (**??**) constant to specify the machine type of the object file to generate.

With a small change to the above program we can let the user know which reader was selected to read the input file: we can use TResources.FindReader (**??**) class method to obtain the appropriate reader for a given stream.

```
program res3;

{$mode objfpc}

uses
  Classes, SysUtils, resource,
  resreader, coffreader, elfreader, winpeimagereader, //readers
  elfwriter, elfconsts;

var
  resources : TResources;
  writer : TElfResourceWriter;
  reader : TAbstractResourceReader;
  inFile : TFileStream;
begin
  resources:=TResources.Create;
  inFile:=TFileStream.Create(paramstr(1), fmOpenRead or fmShareDenyNone);
  reader:=TResources.FindReader(inFile);
  writeln('Selected reader: ',reader.Description);
  resources.LoadFromStream(inFile,reader);
  writer:=TElfResourceWriter.Create;
  writer.MachineType:=emtsparc;
  resources.WriteToFile(ChangeFileExt(paramstr(1),'.o'),writer);
  resources.Free;
  reader.Free;
  writer.Free;
  inFile.Free;
end.
```

Output example:

```
user@localhost:~$ ./res3 myresource.res
Selected reader: .res resource reader
user@localhost:~$
```

Single resources

You can do more with resources than simply converting between file formats.

TResources.Items (**??**) property provides a simple way to access all resources contained in the TResources (**??**) object.

In the following example we read a resource file and then dump each resource data in a file whose name is built from type and name of the dumped resource.

```
program res4;
```

```
{$mode objfpc}

uses
  Classes, SysUtils, resource, resreader;

var
  resources : TResources;
  dumpFile : TFileStream;
  i : integer;
  fname : string;
begin
  resources:=TResources.Create;
  resources.LoadFromFile('myresource.res');
  for i:=0 to resources.Count-1 do
  begin
    fname:=resources[i]._Type.Name+'_'+resources[i].Name.Name;
    dumpFile:=TFileStream.Create(fname,fmCreate or fmShareDenyWrite);
    dumpFile.CopyFrom(resources[i].RawData,resources[i].RawData.Size);
    dumpFile.Free;
  end;
  resources.Free;
end.
```

This code simply copies the content of each resource's RawData (**??**) stream to a file stream, whose name is *resourcetype_resourcename*.

Resource raw data isn't always what one expected, however. While some resource types simply contain a copy of a file in their raw data, other types do some processing, so that dumping raw data doesn't result in a file in the format one expected.

E.g. a resource of type RT_MANIFEST (**??**) is of the former type: its raw data is like an XML manifest file. On the other hand, in a resource of type RT_BITMAP (**??**) the RawData (**??**) stream isn't like a BMP file.

For this reason, several classes (descendants of TAbstractResource (**??**)) are provided to handle the peculiarities of this or that resource type. Much like it's done with readers and writers, resource classes can be registered: adding the unit that contains a resource class to the `uses` clause of your program registers that class. This way, when resources are read from a file, they are created with the class that is registered for their type (the class responsible to do this is TResourceFactory (**??**), but probably you won't need to use it unless you're implementing a new resource reader or resource class).

In the following example, we read a resource file and then dump data of each resource of type RT_BITMAP (**??**) as a BMP file.

```
program res5;

{$mode objfpc}

uses
  Classes, SysUtils, resource, resreader, bitmapresource;

var
  resources : TResources;
  dumpFile : TFileStream;
```

```
  i : integer;
  fname : string;
begin
  resources:=TResources.Create;
  resources.LoadFromFile('myresource.res');
  for i:=0 to resources.Count-1 do
    if resources[i] is TBitmapResource then
      with resources[i] as TBitmapResource do
      begin
        fname:=Name.Name+'.bmp';
        dumpFile:=TFileStream.Create(fname,fmCreate or fmShareDenyWrite);
        dumpFile.CopyFrom(BitmapData,BitmapData.Size);
        dumpFile.Free;
      end;
  resources.Free;
end.
```

Note that we included bitmapresource (**??**) in the `uses` clause of our program. This way, resources of type RT_BITMAP (**??**) are created from TBitmapResource (**??**) class. This class provides a stream, BitmapData (**??**) that allows resource raw data to be accessed as if it was a bmp file.

We can of course do the opposite. In the following code we are creating a manifest resource from `manifest.xml` file.

```
program res6;

{$mode objfpc}

uses
  Classes, SysUtils, resource, reswriter;

var
  resources : TResources;
  inFile : TFileStream;
  res : TGenericResource;
  rname,rtype : TResourceDesc;
begin
  inFile:=TFileStream.Create('manifest.xml',fmOpenRead or fmShareDenyNone);
  rtype:=TResourceDesc.Create(RT_MANIFEST);
  rname:=TResourceDesc.Create(1);
  res:=TGenericResource.Create(rtype,rname);
  rtype.Free; //no longer needed
  rname.Free;
  res.SetCustomRawDataStream(inFile);
  resources:=TResources.Create;
  resources.Add(res);
  resources.WriteToFile('myresource.res');
  resources.Free;  //frees res as well
  inFile.Free;
end.
```

Note that resources of type RT_MANIFEST (**??**) contain a straight copy of a xml file, so TGenericResource (**??**) class fits our needs. TGenericResource (**??**) is a basic implementation of TAbstractResource (**??**). It is the default class used by TResourceFactory (**??**) when it must create a resource whose type wasn't registered with any resource class.

Please note that instead of copying `inFile` contents to RawData (**??**) we used SetCustomRawDataStream (**??**) method: it sets a stream as the underlying stream for RawData (**??**), so that when final resource file is written, data is read directly from the original file.

Let's see a similar example, in which we use a specific class instead of TGenericResource (**??**). In the following code we are creating a resource containing the main program icon, which is read from `mainicon.ico` file.

```
program res7;

{$mode objfpc}

uses
  Classes, SysUtils, resource, reswriter, groupiconresource;

var
  resources : TResources;
  inFile : TFileStream;
  iconres : TGroupIconResource;
  name : TResourceDesc;
begin
  inFile:=TFileStream.Create('mainicon.ico',fmOpenRead or fmShareDenyNone);
  name:=TResourceDesc.Create('MAINICON');
  //type is always RT_GROUP_ICON for this resource class
  iconres:=TGroupIconResource.Create(nil,name);
  iconres.SetCustomItemDataStream(inFile);
  resources:=TResources.Create;
  resources.Add(iconres);
  resources.WriteToFile('myicon.res');
  resources.Free;  //frees iconres as well
  inFile.Free;
  name.Free;
end.
```

In this program we created a new TGroupIconResource (**??**) with 'MAINICON' as name, and we loaded its contents from file 'mainicon.ico'. Please note that RT_GROUP_ICON (**??**) resource raw data doesn't contain a .ico file, so we have to write to ItemData (**??**) which is a ico-like stream. As we did for `res6` program, we tell the resource to use our stream as the underlying stream for resource data: the only difference is that we are using TGroupResource.SetCustomItemDataStream (**??**) instead of TAbstractResource.SetCustomRawDataStream (**??**) method, for obvious reasons.

Other resource types

There are other resource types that allow to easily deal with resource data. E.g. TVersionResource (**??**) makes it easy to create and read version information for Windows executables and dlls, TStringTableResource (**??**) deals with string tables, and so on.

Data caching

Whenever possible, fcl-res tries to avoid to duplicate data. Generally a reader doesn't copy resource data from the original stream to RawData (**??**) stream: instead, it only informs the resource about where its raw data is in the original stream. RawData (**??**) uses a caching system so that it appears as a stream while it only redirects operations to its underlying stream, with a copy-on-write mechanism. Of course this behaviour can be controlled by the user. For further information, see TAbstractResource (**??**) and TAbstractResource.RawData (**??**).

0.3 How to implement a new resource class

Remark This chapter assumes you have some experience in using this library.

Some considerations

Usually, a specific resource class is needed when resource data is encoded in a specific binary format that makes working with RawData (**??**) uncomfortable.

However, there aren't many reasons to design a new binary format requiring a specific resource class: the classes provided with this package exist for compatibility with Microsoft Windows, but in the general case one should avoid such approach.

In Microsoft Windows, some resource types have a specific format, and the operating system supports them at runtime making it easy to access that kind of data: e.g. icons and bitmaps are stored in resources in a format that is slightly different from the one found in regular files, but the operating system allows the user to easily load them using `LoadImage` function, without having to deal with their internal format. Other resource types are used to obtain information about the executable: RT_VERSION (**??**) resource type and RT_GROUP_ICON (**??**) contain version information and program icon that can be displayed in Windows Explorer, respectively.

Using this kind of resources in a cross-platform perspective doesn't make much sense however, since there is no support by other operating systems for these types of resources (and for resources in general), and this means that it's up to you to provide support at runtime for these binary formats. So if you need to store images as resources it's better to use TGenericResource (**??**) and store an image in PNG format (for instance), which can be read by existing libraries at runtime, instead of creating a RT_BITMAP (**??**) resource with TBitmapResource (**??**), since libraries that read BMP files can't handle that resource contents.

New resource classes thus make sense when you want to add support for existing Windows-specific resources, e.g. because you are writing a resource compiler or editor, but in the general case they should be avoided.

Now that you've been warned, let's start with the topic of this chapter.

How to implement a new resource class

A resource class is a descendant of TAbstractResource (**??**), and it's usually implemented in a unit named `typeresource`, where *type* is resource type.

If you are implementing a new resource class, you are doing it to provide additional methods or properties to utilize resource data. You resource class must thus be able to read its RawData (**??**) stream format and write data back to it when it is requested to do so.

Generally, your class shouldn't create private objects, records or memory buffers to provide these specific means of accessing data until it's requested to do so (lazy loading), and it should free these things when it is requested to write back data to the RawData (**??**) stream.

We'll see these points in more detail, using TAcceleratorsResource (**??**) as an example.

TAcceleratorsResource (**??**) holds a collection of accelerators. An accelerator is a record defined as follows:

```
type
  TAccelerator = packed record
    Flags : word;
    Ansi : word;
    Id : word;
    padding : word;
  end;
```

The resource simply contains accelerators, one immediately following the other. Last accelerator

must have $80 in its flags.

To provide easy access to the elements it contains, our accelerator resource class exposes these methods and properties in its public section:

```
procedure Add(aItem : TAccelerator);
procedure Clear;
procedure Delete(aIndex : integer);
property Count : integer read GetCount;
property Items[index : integer] : TAccelerator read GetItem write SetItem; default;
```

We must also implement abstract methods (and an abstract constructor) of TAbstractResource (**??**):

```
protected
  function GetType : TResourceDesc; override;
  function GetName : TResourceDesc; override;
  function ChangeDescTypeAllowed(aDesc : TResourceDesc) : boolean; override;
  function ChangeDescValueAllowed(aDesc : TResourceDesc) : boolean; override;
  procedure NotifyResourcesLoaded; override;
public
  constructor Create(aType,aName : TResourceDesc); override;
  procedure UpdateRawData; override;
```

The protected methods are very easy to implement, so let's start from them. For GetType (**??**) and GetName (**??**), we need to add two private fields, `fType` and `fName`, of type TResourceDesc (**??**). We create them in the constructor and destroy them in the destructor. Type will always be RT_ACCELERATOR (**??**). We make the parameterless constructor of TAbstractResource (**??**) public, using `1` as the resource name, while in the other constructor we use the name passed as parameter, ignoring the type (since it must always be RT_ACCELERATOR (**??**)).

So, GetType (**??**), GetName (**??**), the constructors and the destructor are implemented as follows:

```
function TAcceleratorsResource.GetType: TResourceDesc;
begin
  Result:=fType;
end;

function TAcceleratorsResource.GetName: TResourceDesc;
begin
  Result:=fName;
end;

constructor TAcceleratorsResource.Create;
begin
  inherited Create;
  fType:=TResourceDesc.Create(RT_ACCELERATOR);
  fName:=TResourceDesc.Create(1);
  SetDescOwner(fType);
  SetDescOwner(fName);
end;

constructor TAcceleratorsResource.Create(aType, aName: TResourceDesc);
begin
  Create;
  fName.Assign(aName);
```

```
end;

destructor TAcceleratorsResource.Destroy;
begin
  fType.Free;
  fName.Free;
  inherited Destroy;
end;
```

Note that we used SetDescOwner (**??**) to let type and name know the resource that owns them.

Now ChangeDescTypeAllowed (**??**) and ChangeDescValueAllowed (**??**) come. As we said, resource type must be RT_ACCELERATOR (**??**), always. Thus, we only allow name to change value or type:

```
function TAcceleratorsResource.ChangeDescTypeAllowed(aDesc: TResourceDesc): boolean;
begin
  Result:=aDesc=fName;
end;

function TAcceleratorsResource.ChangeDescValueAllowed(aDesc: TResourceDesc): boolean
begin
  Result:=aDesc=fName;
end;
```

NotifyResourcesLoaded (**??**) is called by TResources (**??**) when it finishes loading all resources. Since we are not interested in this fact, we simply leave this method empty. This method is useful for resources that "own" other resources, like TGroupIconResource (**??**) and TGroupCursorResource (**??**) (note: you should **not** implement resource types that depend on other resources: it complicates things a lot and gives you a lot of headaches).

Now, let's see the more interesting - and more difficult - part: providing an easy way to deal with accelerators.

As we said earlier, we must implement some methods and properties to do so. Surely we'll need a list to hold pointers to accelerator records, but we must think a little bit about how this list is created, populated, written to RawData (**??**) and so on.

When the object is created, we don't have to create (yet) single accelerator records, as said before; but if the user tries to access them we must do it. If the object is created and its RawData (**??**) contains data (e.g. because a reader has created the resource when loading a resource file) and the user tries to access an accelerator, we must create accelerators from RawData (**??**) contents. So, until a user tries to access accelerators our class doesn't do anything, while when the user does so it "lazy-loads" data, or creates empty structures if RawData (**??**) is empty.

When data is loaded, RawData (**??**) contents aren't considered anymore. When, however, our resource is requested to update RawData (**??**) (that is, when UpdateRawData (**??**) method is invoked), our "lazy-loaded" data must be freed. In fact, a user could ask our resource to update raw data, then he/she could modify it directly and then could use our resource's specialized methods and properties to do further processing: for this reason, when RawData (**??**) is written, other buffered things must be freed, so that they'll created again from RawData (**??**) if needed.

Note that other resource classes could behave differently: e.g. TBitmapResource (**??**) uses a copy-on-write mechanism on top of RawData (**??**) to insert a BMP file header at the beginning of the stream, but it doesn't copy RawData (**??**) contents whenever possible.

Coming back to our TAcceleratorsResource (**??**) example, let's see how to implement this behaviour.

Let's add a `fList` field in the `private` section of our class:

```
fList : TFPList;
```

In the constructor, we set this field to `nil`: we use it to check if data has been loaded from RawData (**??**) or not. Consequently in the destructor we'll free the list only if it's not `nil`:

```
destructor TAcceleratorsResource.Destroy;
begin
  fType.Free;
  fName.Free;
  if fList<>nil then
  begin
    Clear;
    fList.Free;
  end;
  inherited Destroy;
end;
```

(we did not implement `Clear` method yet: we'll see it later).

We said that our resource must load data only when needed; to do this we add a private method, `CheckDataLoaded` that ensures that data is loaded. This method is called by whatever method needs to operate on the list: if data has already been loaded it will quickly return, otherwise it will load data.

```
procedure TAcceleratorsResource.CheckDataLoaded;
var acc : TAccelerator;
    tot, i : integer;
    p : PAccelerator;
begin
  if fList<>nil then exit;
  fList:=TFPList.Create;
  if RawData.Size=0 then exit;
  RawData.Position:=0;
  tot:=RawData.Size div 8;
  for i:=1 to tot do
  begin
    RawData.ReadBuffer(acc,sizeof(acc));
    GetMem(p,sizeof(TAccelerator));
    p^:=acc;
    fList.Add(p);
  end;
end;
```

If `fList` is not `nil` data is already loaded, so exit. Otherwise, create the list. If RawData (**??**) is empty we don't need to load anything, so exit. Otherwise allocate room for accelerators, read them from the stream, and add them to the list.

Note that if we are running on a big endian system we should swap the bytes we read, e.g. calling `SwapEndian` function, but for simplicity this is omitted.

The counterpart of `CheckDataLoaded` is UpdateRawData (**??**). When it is called, data from the list must be written back to RawData (**??**), and the list and its contents must be freed:

```
procedure TAcceleratorsResource.UpdateRawData;
var acc : TAccelerator;
```

```
    i : integer;
begin
  if fList=nil then exit;
  RawData.Size:=0;
  RawData.Position:=0;

  for i:=0 to fList.Count-1 do
  begin
    acc:=PAccelerator(fList[i])^;
    // $80 means 'this is the last entry', so be sure only the last one has this bit
    if i=Count-1 then acc.Flags:=acc.Flags or $80
    else acc.Flags:=acc.Flags and $7F;
    RawData.WriteBuffer(acc,sizeof(acc));
  end;
  Clear;
  FreeAndNil(fList);
end;
```

If `fList` is `nil` data hasn't been loaded, so RawData (**??**) is up to date, so exit. Otherwise, write each accelerator (ensuring that only last one has `$80` flag set), clear the list, free it and set it to nil. Again, if we are on a big endian system we should swap each accelerator contents before writing it, but for simplicity this is omitted.

Other methods we named earlier (`Add`, `Delete`, `Clear`) are trivial to implement: remember only to call `CheckDataLoaded` before doing anything. The same is true for accessor methods of relevant properties (`Count`, `Items`).

If you are curious, you can check the full implementation of TAcceleratorsResource (**??**) looking at source code.

0.4 How to implement a new resource reader

Remark This chapter assumes you have some experience in using this library.

We'll see how to implement a reader for a new resource file format. A resource reader is a descendant of TAbstractResourceReader (**??**), and it's usually implemented in a unit named `namereader`, where *name* is file format name.

Suppose we must write a reader for file format *foo*; we could start with a unit like this:

```
unit fooreader;

{$MODE OBJFPC} {$H+}

interface

uses
  Classes, SysUtils, resource;

type
  TFooResourceReader = class(TAbstractResourceReader)
  protected
    function GetExtensions : string; override;
    function GetDescription : string; override;
    procedure Load(aResources : TResources; aStream : TStream); override;
```

```
    function CheckMagic(aStream : TStream) : boolean; override;
  public
    constructor Create; override;
  end;

implementation

function TFooResourceReader.GetExtensions: string;
begin

end;

function TFooResourceReader.GetDescription: string;
begin

end;

procedure TFooResourceReader.Load(aResources: TResources; aStream: TStream);
begin

end;

function TFooResourceReader.CheckMagic(aStream: TStream): boolean;
begin

end;

constructor TFooResourceReader.Create;
begin

end;

initialization
  TResources.RegisterReader('.foo',TFooResourceReader);

end.
```

Note that in the `initialization` section, `TFooResourceReader` is registered for extension `.foo`.

We must implement abstract methods of TAbstractResourceReader (**??**). Let's start with the simpler ones, GetExtensions (**??**) and GetDescription (**??**).

```
function TFooResourceReader.GetExtensions: string;
begin
  Result:='.foo';
end;

function TFooResourceReader.GetDescription: string;
begin
  Result:='FOO resource reader';
end;
```

Now let's see CheckMagic (**??**) method. This method is called with a stream as a parameter, and

the reader must return `true` if it recognizes the stream as a valid one. Usually this means checking some magic number or header.

```
function TFooResourceReader.CheckMagic(aStream: TStream): boolean;
var signature : array[0..3] of char;
begin
  Result:=false;
  try
    aStream.ReadBuffer(signature[0],4);
  except
    on e : EReadError do exit;
  end;
  Result:=signature='FOO*';
end;
```

Suppose our foo files start with a 4-byte signature `'FOO*'`. This method checks the signature and returns `true` if it is verified. Note that it catches `EReadError` exception raised by `TStream`: this way, if the stream is too short it returns `false` (as it should, since magic is not valid) instead of letting the exception to propagate.

Now let's see Load (**??**). This method must read the stream and create resources in the TResources (**??**) object, with information about their name, type, position and size of their raw data, and so on.

```
procedure TFooResourceReader.Load(aResources: TResources; aStream: TStream);
begin
  if not CheckMagic(aStream) then
    raise EResourceReaderWrongFormatException.Create('');
  try
    ReadResources(aResources,aStream);
  except
    on e : EReadError do
      raise EResourceReaderUnexpectedEndOfStreamException.Create('');
  end;
end;
```

First of all, this method checks file magic number, calling CheckMagic (**??**) method we already implemented. This is necessary since CheckMagic (**??**) is not called before Load (**??**): CheckMagic (**??**) is invoked by TResources (**??**) when probing a stream, while Load (**??**) is invoked when loading resources (so if the user passed a reader object to a TResources (**??**) object, CheckMagic (**??**) is never called). Note also that the stream is always at its starting position when these methods are called.

If magic number is ok, our method invokes another method to do the actual loading. If during this process the stream can't be read, an EResourceReaderUnexpectedEndOfStreamException (**??**) exception is raised.

So, let's implement the private method which will load resources.

Suppose that our foo format is very simple:

- the header is made by the 4-byte signature we already saw, a `longword` holding the number of resources in the file, and other 8 bytes of padding.
- each resource has a 16-byte header containing:
  - a longword for the resource type (only IDs are allowed for types)
  - a longword for the resource name (only IDs are allowed for names)

  - a longword for the language ID
  - a longword for the size of the resource data

- after the resource header immediately comes the resource data, possibly padded so that it ends on a 4 byte boundary.
- file format is little-endian

To start with, our method will be:

```
procedure TFooResourceReader.ReadResources(aResources: TResources; aStream: TStream)
var Count, i: longword;
    aType, aName, aLangID : longword;
    aDataSize : longword;
begin
  //read remaining file header
  aStream.ReadBuffer(Count,sizeof(Count));
  aStream.Seek(8,soFromCurrent);

  for i:=1 to Count do
  begin
    //read resource header
    aStream.ReadBuffer(aType,sizeof(aType));
    aStream.ReadBuffer(aName,sizeof(aName));
    aStream.ReadBuffer(aLangID,sizeof(aLangID));
    aStream.ReadBuffer(aDataSize,sizeof(aDataSize));

  end;
end;
```

Since in Load (**??**) we called CheckMagic (**??**), which read the first 4 bytes of the header, we must read the remaining 12: we read the number of resources, and we skip the other 8 bytes of padding.

Then, for each resource, we read the resource header. Note that if we are running on a big endian system we should swap the bytes we read, e.g. calling `SwapEndian` function, but for simplicity this is omitted.

Now, we should create a resource. Of which class? Well, we must use resfactory (**??**) unit. In fact it contains TResourceFactory (**??**) class, which is an expert in creating resources of the right class: when the user adds a unit containing a resource class to the `uses` clause of its program, the resource class registers itself with TResourceFactory (**??**). This way it knows how to map resource types to resource classes.

We need to have type and name of the resource to create as TResourceDesc (**??**) objects: instead of creating and destroying these objects for each resource, we'll create a couple in the creator of our reader and we'll destroy them in the destructor, so that they will live for the whole life of our reader. Let's name them `workType` and `workName`.

Our code becomes:

```
uses
  resfactory;

procedure TFooResourceReader.ReadResources(aResources: TResources; aStream: TStream)
var Count, i: longword;
    aType, aName, aLangID : longword;
```

```
    aDataSize : longword;
    aRes : TAbstractResource;
begin
  //read remaining file header
  aStream.ReadBuffer(Count,sizeof(Count));
  aStream.Seek(8,soFromCurrent);

  for i:=1 to Count do
  begin
    //read resource header
    aStream.ReadBuffer(aType,sizeof(aType));
    aStream.ReadBuffer(aName,sizeof(aName));
    aStream.ReadBuffer(aLangID,sizeof(aLangID));
    aStream.ReadBuffer(aDataSize,sizeof(aDataSize));

    //create the resource
    workType.ID:=aType;
    workName.ID:=aName;
    aRes:=TResourceFactory.CreateResource(workType,workName);
    SetDataSize(aRes,aDataSize);
    SetDataOffset(aRes,aStream.Position);
    aRes.LangID:=aLangID;

  end;
end;
```

Note that after the resource has been created we set its data size and data offset. Data offset is the current position in the stream, since in our FOO file format resource data immediately follows resource header.

What else do we need to do? Of course we must create RawData (**??**) stream for our resource, so that raw data can be accessed with the caching mechanism. We will create a TResourceDataStream (**??**) object, telling it which resource and stream it is associated to, which its size will be and which class its underlying cached stream must be created from.

So we add resdatastream (**??**) to the `uses` clause, declare another local variable

```
aRawData : TResourceDataStream;
```

and add these lines in the for loop

```
aRawData:=TResourceDataStream.Create(aStream,aRes,aRes.DataSize,TCachedResourceDataS
SetRawData(aRes,aRawData);
```

That is, aRawData will create its underlying stream as a TCachedResourceDataStream (**??**) over the portion of `aStream` that starts at current position and ends after `aRes.DataSize bytes`.

We almost finished: now we must add the newly created resource to the TResources (**??**) object and move stream position to the next resource header. Complete code for `ReadResources` method is:

```
procedure TFooResourceReader.ReadResources(aResources: TResources; aStream: TStream)
var Count, i: longword;
    aType, aName, aLangID : longword;
    aDataSize : longword;
    aRes : TAbstractResource;
```

```
    aRawData : TResourceDataStream;
begin
  //read remaining file header
  aStream.ReadBuffer(Count,sizeof(Count));
  aStream.Seek(8,soFromCurrent);

  for i:=1 to Count do
  begin
    //read resource header
    aStream.ReadBuffer(aType,sizeof(aType));
    aStream.ReadBuffer(aName,sizeof(aName));
    aStream.ReadBuffer(aLangID,sizeof(aLangID));
    aStream.ReadBuffer(aDataSize,sizeof(aDataSize));

    //create the resource
    workType.ID:=aType;
    workName.ID:=aName;
    aRes:=TResourceFactory.CreateResource(workType,workName);
    SetDataSize(aRes,aDataSize);
    SetDataOffset(aRes,aStream.Position);
    aRes.LangID:=aLangID;

    //set raw data
    aRawData:=TResourceDataStream.Create(aStream,aRes,aRes.DataSize,TCachedResourceD
    SetRawData(aRes,aRawData);

    //add to aResources
    try
      aResources.Add(aRes);
    except
      on e : EResourceDuplicateException do
      begin
        aRes.Free;
        raise;
      end;
    end;

    //go to next resource header
    aStream.Seek(aDataSize,soFromCurrent);
    Align4Bytes(aStream);
  end;
end;
```

`Align4Bytes` is a private method (not shown for simplicity) that sets stream position to the next multiple of 4 if needed, since FOO file format specifies that resource data must be padded to end on a 4 byte boundary.

Note: We have used Add (**??**) method to populate the TResources (**??**) object. More complex file formats store resources in a tree hierarchy; since TResources (**??**) internally stores resources in this way too, a reader can choose to acquire a reference to the internal tree used by the TResources (**??**) object (see TAbstractResourceReader.GetTree (**??**)), populate it and notify the TResources (**??**) object about the added resources (see TAbstractResourceReader.AddNoTree (**??**)). For these file formats resources can be loaded faster, since there is no overhead involved in keeping a separate resource tree in the reader.

That's all. Now you should be able to create a real resource reader.

0.5 How to implement a new resource writer

Remark This chapter assumes you have some experience in using this library.

We'll see how to implement a writer for a new resource file format. A resource writer is a descendant of TAbstractResourceWriter (**??**), and it's usually implemented in a unit named `namewriter`, where *name* is file format name.

Suppose we must write a writer for file format *foo*; we could start with a unit like this:

```
unit foowriter;

{$MODE OBJFPC} {$H+}

interface

uses
  Classes, SysUtils, resource;

type
  TFooResourceWriter = class (TAbstractResourceWriter)
  protected
    function GetExtensions : string; override;
    function GetDescription : string; override;
    procedure Write(aResources : TResources; aStream : TStream); override;
  public
    constructor Create; override;
  end;

implementation

function TFooResourceWriter.GetExtensions: string;
begin

end;

function TFooResourceWriter.GetDescription: string;
begin

end;

procedure TFooResourceWriter.Write(aResources: TResources; aStream: TStream);
begin

end;

constructor TFooResourceWriter.Create;
begin

end;

initialization
```

```
  TResources.RegisterWriter('.foo',TFooResourceWriter);

end.
```

Note that in the `initialization` section, `TFooResourceWriter` is registered for extension `.foo`.

We must implement abstract methods of TAbstractResourceWriter (**??**). Let's start with the simpler ones, GetExtensions (**??**) and GetDescription (**??**).

```
function TFooResourceWriter.GetExtensions: string;
begin
  Result:='.foo';
end;

function TFooResourceWriter.GetDescription: string;
begin
  Result:='FOO resource writer';
end;
```

Now let's see Write (**??**). This method must write all resources in the TResources (**??**) object to the stream.

Suppose that our foo format is very simple:

- the header is made by a 4-byte signature (`FOO*`), a `longword` holding the number of resources in the file, and other 8 bytes of padding.
- each resource has a 16-byte header containing:
  - a longword for the resource type (only IDs are allowed for types)
  - a longword for the resource name (only IDs are allowed for names)
  - a longword for the language ID
  - a longword for the size of the resource data
- after the resource header immediately comes the resource data, possibly padded so that it ends on a 4 byte boundary.
- file format is little-endian

Our Write (**??**) method could be something like this:

```
procedure TFooResourceWriter.Write(aResources: TResources; aStream: TStream);
var i : integer;
begin
  WriteFileHeader(aStream,aResources);
  for i:=0 to aResources.Count-1 do
    WriteResource(aStream,aResources[i]);
end;
```

So we must implement `WriteFileHeader`, which writes the 16-byte file header, and `WriteResource`, which writes a single resource with its header.

Let's start from the first one:

```
procedure TFooResourceWriter.WriteFileHeader(aStream: TStream; aResources: TResource
var signature : array[0..3] of char;
    rescount : longword;
    padding : qword;
begin
  signature:='FOO*';
  rescount:=aResources.Count;
  padding:=0;

  aStream.WriteBuffer(signature[0],4);
  aStream.WriteBuffer(rescount,sizeof(rescount));
  aStream.WriteBuffer(padding,sizeof(padding));
end;
```

This code simply writes the file header as defined in foo format. Note that if we are running on a big endian system we should swap bytes before writing, e.g. calling `SwapEndian` function, but for simplicity this is omitted.

Now `WriteResource` comes. This method could be like this:

```
procedure TFooResourceWriter.WriteResource(aStream: TStream; aResource: TAbstractRes
var aType : longword;
    aName : longword;
    aLangID : longword;
    aDataSize : longword;
begin
  aType:=aResource._Type.ID;
  aName:=aResource.Name.ID;
  aLangID:=aResource.LangID;
  aDataSize:=aResource.RawData.Size;

  //write resource header
  aStream.WriteBuffer(aType,sizeof(aType));
  aStream.WriteBuffer(aName,sizeof(aName));
  aStream.WriteBuffer(aLangID,sizeof(aLangID));
  aStream.WriteBuffer(aDataSize,sizeof(aDataSize));

  //write resource data
  aResource.RawData.Position:=0;
  aStream.CopyFrom(aResource.RawData,aResource.RawData.Size);

  //align data so that it ends on a 4-byte boundary
  Align4Bytes(aStream);
end;
```

We write the 16-byte resource header, and then resource data. Note that if resources have been loaded from a stream and the user didn't modify resource data, we are copying data directly from the original stream.

`Align4Bytes` is a private method (not shown for simplicity) that writes some padding bytes to the stream if needed, since FOO file format specifies that resource data must be padded to end on a 4 byte boundary. Again, we didn't consider endianess for simplicity. And finally, note that foo format only supports IDs for types and names, so if one of them is a name (that is, a string) this code might cause exceptions to be raised.

Note: More complex file formats store resources in a tree hierarchy; since TResources (**??**) internally stores resources in this way too, a writer can choose to acquire a reference to the internal tree used by the TResources (**??**) object (see TAbstractResourceWriter.GetTree (**??**)) and use it directly. For these file formats resources can be written faster, since there is no overhead involved in building a separate resource tree in the writer.

That's all. Now you should be able to create a real resource writer.

0.6 Format of resources in object files

Introduction

This appendix describes the format in which resources are stored in object files. This doesn't apply to PECOFF file format, where a format already exists and is well described in documentation from Microsoft.

On Microsoft Windows, resources are natively supported by the operating system. On other systems, Free Pascal RTL provides access to resources, which are embedded in the executable file in the format that is here described.

This appendix doesn't describe a particular object file format implementation (e.g. ELF or Mach-O), but the layout of the sections involved in supporting resources: the way these sections are actually laid out on a file are subject to the rules of the hosting object file format.

For external resource file details, see Description of external resource file format (**??**)

Conventions

In this document, data sizes are specified with pascal-style data types. They are the following:

Table 1:

Name	Meaning
`longword`	Unsigned 32 bit integer.
`ptruint`	Unsigned integer of the size of a pointer

Byte order is the one of the target machine.

All data structures in the sections must be aligned on machine boundaries (4 bytes for 32 bit machines, 8 bytes for 64 bit machines).

Note that all pointers must be valid at runtime. This means that relocations must be written to the object file so that the linker can relocate pointers to the addresses they will have at runtime. Note also that pointers to strings are really pointers, and not offsets to the beginning of the string table.

Resource sections

Free Pascal uses two sections to store resource information:

- `fpc.resources`. This is a data section that contains all required structures. It must be writable.
- `fpc.rehandles`. This is a bss section whose size must be equal to `(total number of resources)*(size of pointer)`. It must be writable.

The rest of this chapter will describe the contents of `fpc.resources` section.

Resource section layout

The `fpc.resources` section consists of these parts:

- The initial header
- The resource tree, in the form of nodes
- The string table, which can be optional
- The resource data

The initial header

The `fpc.resources` section starts with this header:

Table 2:

Name	Offset	Length	Description
rootptr	0	ptruint	Pointer to the root node.
count	4/8	longword	Total number of resources.
usedhandles	8/12	longword	Used at runtime. Set to zero in object files.
handles	12/16	ptruint	Pointer to the first byte of `fpc.reshandles` section.

The resource tree

Immediately following the initial header, the resource tree comes. It is made up by nodes that represent resource types, names and language ids.

Data is organized so that resource information (type, name and language id) is represented by a tree: root node contains resource types, that in turn contain resource names, which contain language ids, which describe resource data.

Given a node, its sub-nodes are ordered as follows:

- First the "named" nodes (nodes that use a string as identifier) come, then the id ones (nodes that use an integer as identifier).
- Named nodes are alphabetically sorted, in ascending order.
- Id nodes are sorted in ascending order.

In the file, all sub-nodes of a node are written in the order described above. Then, all sub-nodes of the first sub-node are written, and so on.

Example:

There are three resources:

1. a `BITMAP` resource with name `MYBITMAP` and language id `$0409`
2. a `BITMAP` resource with name `1` and language id `0`
3. a resource with type `MYTYPE` and name `1` and language id `0`

Nodes are laid out this way (note that `BITMAP` resources have type `2`):

```
root | MYTYPE 2 | 1 | 0 | MYBITMAP 1 | $0409 | 0
```

That is, types (`MYTYPE` is a string, so it comes before `2` which is `BITMAP`), then names for `MYTYPE` (`1`), then language id for resource 3 (`0`), then names for `BITMAP` (`MYBITMAP` and `1`), then language id for resource 1 (`$0409`), then language id for resource 2 (`0`).

Node format

Table 3:

Name	Offset	Length	Description
nameid	0	ptruint	name pointer, integer id or language id
ncount	4/8	longword	named sub-nodes count
idcountsize	8/12	longword	id sub-nodes count or resource size
subptr	12/16	ptruint	pointer to first sub-node

If the node is identified by a string, `nameid` is a pointer to the null-terminated string holding the name. If it is identified by an id, `nameid` is that id. Language id nodes are always identified by and ID.

`ncount` is the number of named sub-nodes of this node (nodes that are identified by a string).

`idcountsize` is the number of id sub-nodes of this node (nodes that are identified by an integer id). For language id nodes, this field holds the size of the resource data.

`subptr` is an pointer to the first subnode of this node. Note that it allows to access every subnode of this node, since subnodes of a node always come one after the other. For language id nodes, `subptr` is the pointer to the resource data.

The string table

The string table is used to store strings used for resource types and names. If all resources use integer ids for name and types, it may not be present in the file.

The string table simply contains null-terminated strings, one after the other.

If present, the string table always contains a `0` (zero) at the beginning. This way, the empty string is located at the first byte of the string table.

The resource data

This part of the file contains raw resource data. As written before, all data structures must be aligned on machine boundaries, so if a resource data size is not a multiple of 4 (for 32 bit machines) or 8 (for 64 bit machines), bytes of padding must be inserted after that resource data.

Exported symbols

Object files containing resources must export a pointer to the first byte of `fpc.resources` section. The name of this symbol is `FPC_RESSYMBOL`.

Mach-O specific notes

`fpc.resources` and `fpc.reshandles` sections are to be contained in a `__DATA` segment.

Chapter 1

Reference for unit 'acceleratorsresource'

1.1 Used units

Table 1.1: Used units by unit 'acceleratorsresource'

Name	Page
Classes	??
resource	129
sysutils	??

1.2 Overview

This unit contains TAcceleratorsResource (44), a TAbstractResource (43) descendant specialized in handling resource of type RT_ACCELERATOR (43).

Adding this unit to a program's `uses` clause registers class TAcceleratorsResource (44) for type RT_ACCELERATOR (43) with TResourceFactory (43).

1.3 Constants, types and variables

1.3.1 Constants

`FAlt = 16`

This flag is valid only if the key is a virtual key.

`FControl = 8`

This flag is valid only if the key is a virtual key.

`FNoInvert = 2`

This flag is obsolete and is provided only for compatibility with 16 bit Windows.

```
FShift = 4
```

This flag is valid only if the key is a virtual key.

```
FVirtKey = 1
```

When this flag is set, the accelerator key is a virtual key code. Otherwise, it is a ASCII character

1.3.2 Types

```
PAccelerator = ^TAccelerator
```

A pointer to a TAccelerator record

1.4 TAccelerator

```
TAccelerator = packed record
  Flags : Word;
  Ansi : Word;
  Id
  : Word;
  padding : Word;
end
```

A single accelerator entry in the accelerator table resource.

The key associated with the accelerator is represented by `Ansi` field: it can be a character or a virtual-key code (in the latter case, FVirtKey (44) flag must be active).

The accelerator is identified by the value of `id` field.

`Flags` is a combination of the following values:

- FVirtKey (44)
- FShift (44)
- FNoInvert (44)
- FControl (43)
- FAlt (43)

1.5 TAcceleratorsResource

1.5.1 Description

This class represents a resource of type RT_ACCELERATOR (43).

An accelerator table resource is a collection of accelerators (represented by TAccelerator (44) records).

An accelerator represents a keystroke that can be associated with some action.

This resource type is very Microsoft Windows-specific, so it might not be of interest for many users.

Methods are provided to add, delete and modify single accelerators.

Remark This class doesn't allow its type to be changed to anything else than RT_ACCELERATOR (43). Attempts to do so result in a EResourceDescChangeNotAllowedException (43).

1.5.2 Method overview

Page	Method	Description
46	Add	Adds a new accelerator to the table
45	ChangeDescTypeAllowed	
45	ChangeDescValueAllowed	
46	Clear	Empties the accelerator table
46	Create	Creates a new accelerator table resource
47	Delete	Deletes an accelerator from the table
46	Destroy	
45	GetName	
45	GetType	
46	NotifyResourcesLoaded	
46	UpdateRawData	

1.5.3 Property overview

Page	Properties	Access	Description
47	Count	r	The number of accelerators in the table
47	Items	rw	Indexed array of accelerators in the table

1.5.4 TAcceleratorsResource.GetType

Declaration: `function GetType : TResourceDesc;  Override`

Visibility: protected

1.5.5 TAcceleratorsResource.GetName

Declaration: `function GetName : TResourceDesc;  Override`

Visibility: protected

1.5.6 TAcceleratorsResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                ;  Override
```

Visibility: protected

1.5.7 TAcceleratorsResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                 ;  Override
```

Visibility: protected

1.5.8 TAcceleratorsResource.NotifyResourcesLoaded

Declaration: `procedure NotifyResourcesLoaded;  Override`

Visibility: protected

1.5.9 TAcceleratorsResource.Create

Synopsis: Creates a new accelerator table resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                  ;  Override
```

Visibility: public

Description: Please note that `aType` parameter is not used, since this class always uses RT_ACCELERATOR (43) as type.

1.5.10 TAcceleratorsResource.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

1.5.11 TAcceleratorsResource.UpdateRawData

Declaration: `procedure UpdateRawData;  Override`

Visibility: public

1.5.12 TAcceleratorsResource.Add

Synopsis: Adds a new accelerator to the table

Declaration: `procedure Add(aItem: TAccelerator)`

Visibility: public

See also: TAcceleratorsResource.Items (47)

1.5.13 TAcceleratorsResource.Clear

Synopsis: Empties the accelerator table

Declaration: `procedure Clear`

Visibility: public

See also: TAcceleratorsResource.Items (47), TAcceleratorsResource.Delete (47)

1.5.14 TAcceleratorsResource.Delete

Synopsis: Deletes an accelerator from the table

Declaration: `procedure Delete(aIndex: Integer)`

Visibility: public

See also: TAcceleratorsResource.Items (47), TAcceleratorsResource.Clear (46)

1.5.15 TAcceleratorsResource.Count

Synopsis: The number of accelerators in the table

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

See also: TAcceleratorsResource.Items (47)

1.5.16 TAcceleratorsResource.Items

Synopsis: Indexed array of accelerators in the table

Declaration: `Property Items[index: Integer]: TAccelerator; default`

Visibility: public

Access: Read,Write

Description: This property can be used to access all accelerators in the object.

Remark This array is 0-based: valid elements range from 0 to Count (47)-1.

Remark If you need to access RawData (43) after you added, deleted or modified accelerators, be sure to call UpdateRawData (43) first. This isn't needed however when resource is written to a stream, since TResources (43) takes care of it.

See also: TAcceleratorsResource.Count (47), TAccelerator (44)

Chapter 2

Reference for unit 'bitmapresource'

2.1 Used units

Table 2.1: Used units by unit 'bitmapresource'

Name	Page
Classes	??
resource	129
sysutils	??

2.2 Overview

This unit contains TBitmapResource (48), a TAbstractResource (48) descendant specialized in handling resource of type RT_BITMAP (48).

Adding this unit to a program's `uses` clause registers class TBitmapResource (48) for type RT_BITMAP (48) with TResourceFactory (48).

2.3 TBitmapResource

2.3.1 Description

This class represents a resource of type RT_BITMAP (48).

A bitmap resource contents is very similar to a BMP file. However some differences exists, so RawData (48) is not appropriate if you need to read and write BMP data. Instead, BitmapData (51) property gives access to a BMP file-like stream.

Remark This class doesn't allow its type to be changed to anything else than RT_BITMAP (48). Attempts to do so result in a EResourceDescChangeNotAllowedException (48).

See also: BitmapData (51), TAbstractResource.RawData (48)

2.3.2 Method overview

Page	Method	Description
49	ChangeDescTypeAllowed	
49	ChangeDescValueAllowed	
50	Create	Creates a new bitmap resource
50	Destroy	
49	GetName	
49	GetType	
49	NotifyResourcesLoaded	
50	SetCustomBitmapDataStream	Sets a custom stream as the underlying stream for BitmapData
50	UpdateRawData	

2.3.3 Property overview

Page	Properties	Access	Description
51	BitmapData	r	Resource data as a BMP stream

2.3.4 TBitmapResource.GetType

Declaration: `function GetType : TResourceDesc;  Override`

Visibility: protected

2.3.5 TBitmapResource.GetName

Declaration: `function GetName : TResourceDesc;  Override`

Visibility: protected

2.3.6 TBitmapResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                ;  Override
```

Visibility: protected

2.3.7 TBitmapResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                 ;  Override
```

Visibility: protected

2.3.8 TBitmapResource.NotifyResourcesLoaded

Declaration: `procedure NotifyResourcesLoaded;  Override`

Visibility: protected

2.3.9 TBitmapResource.Create

Synopsis: Creates a new bitmap resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                   ;  Override
```

Visibility: public

Description: Please note that `aType` parameter is not used, since this class always uses RT_BITMAP (48) as type.

2.3.10 TBitmapResource.Destroy

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

Visibility: public

2.3.11 TBitmapResource.UpdateRawData

Declaration:
```
procedure UpdateRawData;  Override
```

Visibility: public

2.3.12 TBitmapResource.SetCustomBitmapDataStream

Synopsis: Sets a custom stream as the underlying stream for BitmapData

Declaration:
```
procedure SetCustomBitmapDataStream(aStream: TStream)
```

Visibility: public

Description: This method allows the user to use a custom stream as the underlying stream for BitmapData (51). This is useful when you want a TBitmapResource (48) to be created from a bmp file for which you have a stream.

Sample code

This code creates a resource containing a bitmap

```
var
  aName : TResourceDesc;
  aRes : TBitmapResource;
  aFile : TFileStream;
  Resources : TResources;
begin
  Resources:=TResources.Create;
  aName:=TResourceDesc.Create('MYBITMAP');
  aRes:=TBitmapResource.Create(nil,aName); //type is always RT_BITMAP
  aName.Free; //not needed anymore
  aFile:=TFileStream.Create('mybitmap.bmp',fmOpenRead or fmShareDenyNone);
  aRes.SetCustomBitmapDataStream(aFile);
  Resources.Add(aRes);
  Resources.WriteToFile('myresource.res');

  Resources.Free; //it destroys aRes as well.
```

```
  aFile.Free;
end;
```

See also: TBitmapResource.BitmapData (51), TAbstractResource.UpdateRawData (48)

2.3.13 TBitmapResource.BitmapData

Synopsis: Resource data as a BMP stream

Declaration: `Property BitmapData : TStream`

Visibility: public

Access: Read

Description: BitmapData (51) property gives access to resource data in a BMP file-like stream, unlike RawData (48).

BitmapData (51) does not create a copy of RawData (48) so memory usage is generally kept limited.

You can also set a custom stream as the underlying stream for BitmapData (51) via SetCustomBitmapDataStream (50), much like SetCustomRawDataStream (48) does for RawData (48). This is useful when you want a TBitmapResource (48) to be created from a bmp file for which you have a stream.

Remark If you need to access RawData (48) after you modified BitmapData (51), be sure to call UpdateRawData (48) first. This isn't needed however when resource is written to a stream, since TResources (48) takes care of it.

See also: TBitmapResource.SetCustomBitmapDataStream (50), TAbstractResource.RawData (48), TAbstractResource.UpdateRawData (48)

Chapter 3

Reference for unit 'coffreader'

3.1 Used units

Table 3.1: Used units by unit 'coffreader'

Name	Page
Classes	**??**
cofftypes	55
resource	129
resourcetree	164
sysutils	**??**

3.2 Overview

This unit contains TCoffResourceReader (52), a TAbstractResourceReader (52) descendant that is able to read COFF object files containing resources.

Adding this unit to a program's `uses` clause registers class TCoffResourceReader (52) with TResources (52).

3.3 TCoffResourceReader

3.3.1 Description

This class provides a reader for COFF object files containing resources.

COFF is the file format used by Microsoft Windows object files. Usually resources get stored in a object file that can be given to a linker to produce an executable.

After an object file has been read, MachineType (54) property holds the machine type the object file was built for.

Remark This reader is not able to read full PE images. Use TWinPEImageResourceReader (52) instead.

See also: TCoffResourceReader.MachineType (54), TAbstractResourceReader (52), TWinPEImageResourceReader (52), TCoffResourceWriter (52)

3.3.2 Method overview

Page	Method	Description
53	CheckMagic	
53	Create	
53	Destroy	
53	GetDescription	
53	GetExtensions	
53	Load	

3.3.3 Property overview

Page	Properties	Access	Description
54	MachineType	r	The machine type of the object file

3.3.4 TCoffResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

3.3.5 TCoffResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

3.3.6 TCoffResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

3.3.7 TCoffResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

3.3.8 TCoffResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

3.3.9 TCoffResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

3.3.10 TCoffResourceReader.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TCoffMachineType`

Visibility: public

Access: Read

Description: This property holds the machine type of the object file that has been read.

Remark Obviously, this property is meaningful only after an object file has been read.

See also: TCoffMachineType (52)

Chapter 4

Reference for unit 'cofftypes'

4.1 Overview

These types are used internally by TCoffResourceWriter (55) and TCoffResourceReader (55).
The only type of interest for the user is TCoffMachineType (55).

4.2 Constants, types and variables

4.2.1 Constants

```
RSRCSectName : TSectionName = '.rsrc' + #0 + #0 + #0
```

4.2.2 Types

```
TCoffMachineType = (cmti386,cmtarm,cmtx8664,cmtppc32aix,cmtppc64aix
  )
```

Table 4.1: Enumeration values for type TCoffMachineType

Value	Explanation
cmtarm	ARM
cmti386	Intel i386
cmtppc32aix	
cmtppc64aix	
cmtx8664	AMD x86_64

This enumeration specifies the COFF machine type.
It is used by TCoffResourceWriter (55) to specify the machine type of the generated object file and by TCoffResourceReader (55) to read the machine type of the object file that has been read.

```
TCoffSectionTable = TCoffSymtableEntry
```

```
TCoffSymtableEntry = packed record
case Byte of
1: (
  Name : TSectionName
  ;
  Value : LongWord;
  SectionNumber : Word;
  _type : Word;
  StorageClass
  : Byte;
  NumAuxSymbol : Byte;
);
2: (
  n_name : TSectionName;
  n_value : LongWord;
  n_scnum : Word;
  n_type : Word;
  n_sclass
  : Byte;
  n_numaux : Byte;
);
end


Array = Array[0..7] of Char
```

4.3 TCoffHeader

```
TCoffHeader = packed record
  Machine : Word;
  NumSects : Word;
  TimeStamp : LongWord;
  SymTablePtr : LongWord;
  SymNum : LongWord
  ;
  OptHdrSize : Word;
  Characteristics : Word;
end
```

4.4 TCoffSectionHeader

```
TCoffSectionHeader = packed record
  Name : TSectionName;
  VirtualSize
  : LongWord;
  VirtualAddress : LongWord;
  SizeOfRawData : LongWord
  ;
  PointerToRawData : LongWord;
  PointerToRelocations : LongWord
```

```
  ;
  PointerToLineNumbers : LongWord;
  NumberOfRelocations : Word
  ;
  NumberOfLineNumbers : Word;
  Characteristics : LongWord;
end
```

4.5 TResDataEntry

```
TResDataEntry = packed record
  DataRVA : LongWord;
  Size : LongWord
  ;
  Codepage : LongWord;
  Reserved : LongWord;
end
```

4.6 TResDirEntry

```
TResDirEntry = packed record
  NameID : LongWord;
  DataSubDirRVA
  : LongWord;
end
```

4.7 TResDirTable

```
TResDirTable = packed record
  Characteristics : LongWord;
  TimeStamp
  : LongWord;
  VerMajor : Word;
  VerMinor : Word;
  NamedEntriesCount
  : Word;
  IDEntriesCount : Word;
end
```

4.8 TXCoff32SectionHeader

```
TXCoff32SectionHeader = packed record
  s_name : TSectionName;
  s_paddr
```

```
  : LongWord;
  s_vaddr : LongWord;
  s_size : LongWord;
  s_scnptr
  : LongWord;
  s_relptr : LongWord;
  s_lnnoptr : LongWord;
  s_nreloc
  : Word;
  s_nlnno : Word;
  s_flags : LongWord;
end
```

4.9 TXCoffAuxSymbol32

```
TXCoffAuxSymbol32 = packed record
  x_scnlen : LongWord;
  x_parmhash
  : LongWord;
  x_snhash : Word;
  x_smtyp : Byte;
  x_smclas : Byte
  ;
  x_stab : LongWord;
  x_snstab : Word;
end
```

Chapter 5

Reference for unit 'coffwriter'

5.1 Used units

Table 5.1: Used units by unit 'coffwriter'

Name	Page
Classes	??
cofftypes	55
resource	129
resourcetree	164
sysutils	??

5.2 Overview

This unit contains TCoffResourceWriter (62), a TAbstractResourceWriter (59) descendant that is able to write COFF object files containing resources.

Adding this unit to a program's `uses` clause registers class TCoffResourceWriter (62) with TResources (59).

5.3 Constants, types and variables

5.3.1 Types

```
PCoffRelocation = ^TCoffRelocation
```

This type is used internally by TCoffResourceWriter (62).

5.4 TCoffRelocation

```
TCoffRelocation = packed record
  VirtualAddress : LongWord;
  SymTableIndex
```

```
  : LongWord;
  _type : Word;
end
```

This record is used internally by TCoffResourceWriter (62).

5.5 TCoffRelocations

5.5.1 Description

This class is used internally by TCoffResourceWriter (62).

5.5.2 Method overview

Page	Method	Description
61	Add	
61	AddRelativeToSection	
61	Clear	
60	Create	
61	Destroy	
60	GetCount	
60	GetRelocation	

5.5.3 Property overview

Page	Properties	Access	Description
61	Count	r	
61	Items	r	
61	MachineType	rw	
61	StartAddress	rw	

5.5.4 TCoffRelocations.GetCount

Declaration: `function GetCount : Integer`

Visibility: protected

5.5.5 TCoffRelocations.GetRelocation

Declaration: `function GetRelocation(index: Integer) : PCoffRelocation`

Visibility: protected

5.5.6 TCoffRelocations.Create

Declaration: `constructor Create(aMachineType: TCoffMachineType)`

Visibility: public

5.5.7 TCoffRelocations.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

5.5.8 TCoffRelocations.Add

Declaration: `procedure Add(aAddress: LongWord; aType: Word; aSymTableIndex: LongWord)`

Visibility: public

5.5.9 TCoffRelocations.AddRelativeToSection

Declaration:
```
procedure AddRelativeToSection(aAddress: LongWord;
                               aSectSymTableIndex: LongWord)
```

Visibility: public

5.5.10 TCoffRelocations.Clear

Declaration: `procedure Clear`

Visibility: public

5.5.11 TCoffRelocations.Count

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

5.5.12 TCoffRelocations.Items

Declaration: `Property Items[index: Integer]: PCoffRelocation; default`

Visibility: public

Access: Read

5.5.13 TCoffRelocations.StartAddress

Declaration: `Property StartAddress : LongWord`

Visibility: public

Access: Read,Write

5.5.14 TCoffRelocations.MachineType

Declaration: `Property MachineType : TCoffMachineType`

Visibility: public

Access: Read,Write

5.6 TCoffResourceWriter

5.6.1 Description

This class provides a writer for COFF object files containing resources.

COFF is the file format used by Microsoft Windows object files. Usually resources get stored in a object file that can be given to a linker to produce an executable.

MachineType (64) property can be used to set the machine type of the object file to generate.

See also: TCoffResourceWriter.MachineType (64), TAbstractResourceWriter (59), TCoffResourceWriter (59)

5.6.2 Method overview

Page	Method	Description
64	Create	
64	Destroy	
63	FixCoffHeader	
63	FixSectionHeader	
63	GetDescription	
63	GetExtensions	
63	GetFixedCoffHeader	
64	PrescanNode	
64	PrescanResourceTree	
64	Write	
63	WriteCoffStringTable	
62	WriteEmptyCoffHeader	
62	WriteEmptySectionHeader	
63	WriteRawData	
63	WriteRelocations	
63	WriteResStringTable	
64	WriteSymbolTable	

5.6.3 Property overview

Page	Properties	Access	Description
64	MachineType	rw	The machine type of the object file
65	OppositeEndianess	rw	

5.6.4 TCoffResourceWriter.WriteEmptyCoffHeader

Declaration: `procedure WriteEmptyCoffHeader(aStream: TStream)`

Visibility: protected

5.6.5 TCoffResourceWriter.WriteEmptySectionHeader

Declaration: `procedure WriteEmptySectionHeader(aStream: TStream); Virtual`

Visibility: protected

5.6.6 TCoffResourceWriter.WriteResStringTable

Declaration: `procedure WriteResStringTable(aStream: TStream);  Virtual`

Visibility: protected

5.6.7 TCoffResourceWriter.WriteRawData

Declaration: `procedure WriteRawData(aStream: TStream)`

Visibility: protected

5.6.8 TCoffResourceWriter.WriteRelocations

Declaration: `procedure WriteRelocations(aStream: TStream)`

Visibility: protected

5.6.9 TCoffResourceWriter.WriteCoffStringTable

Declaration: `procedure WriteCoffStringTable(aStream: TStream)`

Visibility: protected

5.6.10 TCoffResourceWriter.GetFixedCoffHeader

Declaration: `function GetFixedCoffHeader : TCoffHeader;  Virtual`

Visibility: protected

5.6.11 TCoffResourceWriter.FixCoffHeader

Declaration: `procedure FixCoffHeader(aStream: TStream)`

Visibility: protected

5.6.12 TCoffResourceWriter.FixSectionHeader

Declaration: `procedure FixSectionHeader(aStream: TStream; aResources: TResources)`
`;  Virtual`

Visibility: protected

5.6.13 TCoffResourceWriter.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

5.6.14 TCoffResourceWriter.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

5.6.15 TCoffResourceWriter.PrescanNode

Declaration: `function PrescanNode(aNode: TResourceTreeNode; aNodeSize: LongWord) : LongWord;  Virtual`

Visibility: protected

5.6.16 TCoffResourceWriter.PrescanResourceTree

Declaration: `procedure PrescanResourceTree;  Virtual`

Visibility: protected

5.6.17 TCoffResourceWriter.Write

Declaration: `procedure Write(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

5.6.18 TCoffResourceWriter.WriteSymbolTable

Declaration: `procedure WriteSymbolTable(aStream: TStream; aResources: TResources) ;  Virtual`

Visibility: protected

5.6.19 TCoffResourceWriter.Create

Declaration: `constructor Create;  Override`

Visibility: public

5.6.20 TCoffResourceWriter.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

5.6.21 TCoffResourceWriter.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TCoffMachineType`

Visibility: public

Access: Read,Write

Description: This property can be used to set the machine type of the object file to write.

See also: TCoffMachineType (59)

5.6.22 TCoffResourceWriter.OppositeEndianess

Declaration: `Property OppositeEndianess : Boolean`

Visibility: public

Access: Read,Write

5.7 TCoffStringTable

5.7.1 Method overview

Page	Method	Description
65	Add	
65	Create	
65	Delete	

5.7.2 Property overview

Page	Properties	Access	Description
65	Size	r	

5.7.3 TCoffStringTable.Create

Declaration: `constructor Create`

Visibility: public

5.7.4 TCoffStringTable.Add

Declaration: `function Add(const S: string) : Integer;  Override`

Visibility: public

5.7.5 TCoffStringTable.Delete

Declaration: `procedure Delete(Index: Integer);  Override`

Visibility: public

5.7.6 TCoffStringTable.Size

Declaration: `Property Size : ptruint`

Visibility: public

Access: Read

5.8 TResourceStringTable

5.8.1 Description

This class is used internally by TCoffResourceWriter (62).

5.8.2 Method overview

Page	Method	Description
66	Add	
66	Clear	
66	Create	
66	Destroy	

5.8.3 Property overview

Page	Properties	Access	Description
66	Count	r	
67	CurrRVA	r	
67	EndRVA	r	
67	Items	r	
67	StartRVA	rw	

5.8.4 TResourceStringTable.Create

Declaration: `constructor Create`

Visibility: public

5.8.5 TResourceStringTable.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

5.8.6 TResourceStringTable.Add

Declaration: `procedure Add(s: string)`

Visibility: public

5.8.7 TResourceStringTable.Clear

Declaration: `procedure Clear`

Visibility: public

5.8.8 TResourceStringTable.Count

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

5.8.9 TResourceStringTable.Items

Declaration: `Property Items[index: Integer]: string; default`

Visibility: public

Access: Read

5.8.10 TResourceStringTable.StartRVA

Declaration: `Property StartRVA : LongWord`

Visibility: public

Access: Read,Write

5.8.11 TResourceStringTable.CurrRVA

Declaration: `Property CurrRVA : LongWord`

Visibility: public

Access: Read

5.8.12 TResourceStringTable.EndRVA

Declaration: `Property EndRVA : LongWord`

Visibility: public

Access: Read

Chapter 6

Reference for unit 'dfmreader'

6.1 Used units

Table 6.1: Used units by unit 'dfmreader'

Name	Page
Classes	??
resource	129
sysutils	??

6.2 Overview

This unit contains TDfmResourceReader (68), a TAbstractResourceReader (68) descendant used to compile DFM files to resources.

Adding this unit to a program's `uses` clause registers class TDfmResourceReader (68) with TResources (68).

6.3 TDfmResourceReader

6.3.1 Description

This class isn't a proper resource reader. It provides a quick way to create a resource file from a DFM/XFM/LFM file, similar to what Delphi does with constructs like `{$R *.dfm}`.

This class reads a DFM, XFM or LFM file, compiles it to binary format if it isn't, and stores it in a resource of type RT_RCDATA (68) with the name of the form specified in the DFM file.

6.3.2 Method overview

Page	Method	Description
69	CheckMagic	
69	Create	
69	Destroy	
69	GetDescription	
69	GetExtensions	
69	Load	

6.3.3 TDfmResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

6.3.4 TDfmResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

6.3.5 TDfmResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

6.3.6 TDfmResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

6.3.7 TDfmResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

6.3.8 TDfmResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

Chapter 7

Reference for unit 'elfconsts'

7.1 Overview

These constants are used internally by TElfResourceWriter (70) and TElfResourceReader (70).
The only type of interest for the user is TElfMachineType (75).

7.2 Constants, types and variables

7.2.1 Constants

```
EF_IA_64_ABI64 = $10
```

```
ELFCLASS32 = 1
```

```
ELFCLASS64 = 2
```

```
ELFCLASSNONE = 0
```

```
ELFDATA2LSB = 1
```

```
ELFDATA2MSB = 2
```

```
ELFDATANONE = 0
```

```
ELFMAGIC = chr($7f) + 'ELF'
```

```
ELFOSABI_ARM = 97
```

```
ELFOSABI_FREEBSD = 9
```

```
ELFOSABI_LINUX = 3

ELFOSABI_NONE = 0

EM_386 = 3

EM_68K = 4

EM_AARCH64 = 183

EM_ALPHA = $9026

EM_ARM = 40

EM_IA_64 = 50

EM_MIPS = 8

EM_MIPS_RS3_LE = 10

EM_MIPS_X = 51

EM_NONE = 0

EM_PPC = 20

EM_PPC64 = 21

EM_SPARC = 2

EM_X86_64 = 62

ET_CORE = 4

ET_DYN = 3

ET_EXEC = 2
```

```
ET_HIOS = $feff

ET_HIPROC = $ffff

ET_LOOS = $fe00

ET_LOPROC = $ff00

ET_NONE = 0

ET_REL = 1

EV_CURRENT = 1

EV_NONE = 0

HANDLESECT_IDX = 2

HandlesSectName = 'fpc.reshandles'

RsrcSectName = 'fpc.resources'

RSRCSECT_IDX = 1

R_386_32 = 1

R_68K_32 = 1

R_AARCH64_ABS64 = 257

R_ALPHA_REFQUAD = 2

R_ARM_ABS32 = 2

R_IA64_DIR64LSB = $27

R_MIPS_32 = 2
```

```
R_PPC64_ADDR64 = 38

R_PPC_ADDR32 = 1

R_SPARC_32 = 3

R_x86_64_64 = 1

SHF_ALLOC = 2

SHF_EXECINSTR = 4

SHF_MASKOS = $0f000000

SHF_MASKPROC = $f0000000

SHF_WRITE = 1

SHT_DYNAMIC = 6

SHT_DYNSYM = 11

SHT_HASH = 5

SHT_HIOS = $ffffffff

SHT_HIPROC = $7fffffff

SHT_LOOS = $80000000

SHT_LOPROC = $70000000

SHT_NOBITS = 8

SHT_NOTE = 7

SHT_NULL = 0
```

```
SHT_PROGBITS = 1

SHT_REL = 9

SHT_RELA = 4

SHT_SHLIB = 10

SHT_STRTAB = 3

SHT_SYMTAB = 2

STB_GLOBAL = 1

STB_HIOS = 12

STB_HIPROC = 15

STB_LOCAL = 0

STB_LOOS = 10

STB_LOPROC = 13

STB_WEAK = 2

STT_COMMON = 5

STT_FILE = 4

STT_FUNC = 2

STT_HIOS = 12

STT_HIPROC = 15

STT_LOOS = 10

STT_LOPROC = 13

STT_NOTYPE = 0

STT_OBJECT = 1

STT_SECTION = 3

STT_SPARC_REGISTER = 13

STT_TLS = 6
```

7.2.2 Types

```
TElfMachineType = (emtnone,emtsparc,emti386,emtm68k,emtppc,emtppc64
  ,
                   emtarm,emtarmeb,emtia64,emtx86_64,emtalpha,emtmips
  ,
                   emtmipsel,emtppc64le,emtaarch64)
```

Table 7.1: Enumeration values for type TElfMachineType

Value	Explanation
emtaarch64	
emtalpha	DEC Alpha machine type
emtarm	ARM machine type
emtarmeb	ARM Big Endian machine type
emti386	Intel 386 machine type
emtia64	Intel IA-64 machine type
emtm68k	Motorola 68000 machine type
emtmips	
emtmipsel	
emtnone	Invalid machine type
emtppc	PowerPC machine type
emtppc64	PowerPC 64 machine type
emtppc64le	
emtsparc	Sparc machine type
emtx86_64	AMD x86_64 machine type

This enumeration specifies the ELF machine type.

It is used by TElfResourceWriter (70) to specify the machine type of the generated object file and by TElfResourceReader (70) to read the machine type of the object file that has been read.

Chapter 8

Reference for unit 'elfreader'

8.1 Used units

Table 8.1: Used units by unit 'elfreader'

Name	Page
Classes	??
elfconsts	70
elftypes	??
resource	129
sysutils	??

8.2 Overview

This unit contains TElfResourceReader (77), a TAbstractResourceReader (76) descendant that is able to read ELF object files containing resources.

Adding this unit to a program's `uses` clause registers class TElfResourceReader (77) with TResources (76).

8.3 EElfResourceReaderException

8.3.1 Description

Base class for elf resource reader-related exceptions

8.4 EElfResourceReaderNoSectionsException

8.4.1 Description

This exception is raised by Load (76) method of TElfResourceReader (77) when no section headers are found.

8.5 EElfResourceReaderNoStringTableException

8.5.1 Description

This exception is raised by Load (76) method of TElfResourceReader (77) when no ELF string table is found.

8.6 EElfResourceReaderUnknownClassException

8.6.1 Description

This exception is raised by Load (76) method of TElfResourceReader (77) when class field of ELF header is neither ELFCLASS32 (76) nor ELFCLASS64 (76).

8.7 EElfResourceReaderUnknownVersionException

8.7.1 Description

This exception is raised by Load (76) method of TElfResourceReader (77) when version field of ELF header is not 1.

8.8 TElfResourceReader

8.8.1 Description

This class provides a reader for ELF object files and images containing resources.

ELF is the file format used by unices and other operating systems for object files and image files (executables, dynamic libraries and so on). Free Pascal can store resources in ELF files in its own format.

After an object file has been read, MachineType (78) property holds the machine type the object file was built for.

Remark This reader can't read ELF files without section headers. These are however very rare.

See also: TElfResourceReader.MachineType (78), TAbstractResourceReader (76), TElfResourceWriter (76), Format of resources in object files (76)

8.8.2 Method overview

Page	Method	Description
78	CheckMagic	
78	Create	
78	Destroy	
78	GetDescription	
78	GetExtensions	
78	Load	

8.8.3 Property overview

Page	Properties	Access	Description
78	MachineType	r	The machine type of the object file

8.8.4 TElfResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

8.8.5 TElfResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

8.8.6 TElfResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

8.8.7 TElfResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

8.8.8 TElfResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

8.8.9 TElfResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

8.8.10 TElfResourceReader.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TElfMachineType`

Visibility: public

Access: Read

Description: This property holds the machine type of the object file that has been read.

Remark Obviously, this property is meaningful only after an object file has been read.

See also: TElfMachineType (76)

Chapter 9

Reference for unit 'elfwriter'

9.1 Used units

Table 9.1: Used units by unit 'elfwriter'

Name	Page
Classes	??
elfconsts	70
elftypes	??
resource	129
sysutils	??

9.2 Overview

This unit contains TElfResourceWriter (80), a TAbstractResourceWriter (79) descendant that is able to write ELF object files containing resources.

Adding this unit to a program's `uses` clause registers class TElfResourceWriter (80) with TResources (79).

9.3 EElfResourceWriterException

9.3.1 Description

Base class for elf resource writer-related exceptions

9.4 EElfResourceWriterUnknownClassException

9.4.1 Description

If this exception is raised, an internal error occurred.

9.5 EElfResourceWriterUnknownMachineException

9.5.1 Description

This exception is raised when an attempt is made to set TElfResourceWriter.MachineType (81) to an unknown machine type.

See also: TElfResourceWriter.MachineType (81)

9.6 EElfResourceWriterUnknownSectionException

9.6.1 Description

If this exception is raised, an internal error occurred.

9.7 TElfResourceWriter

9.7.1 Description

This class provides a writer for ELF object files and images containing resources.

ELF is the file format used by unices and other operating systems for object files and image files (executables, dynamic libraries and so on). Free Pascal can store resources in ELF files in its own format.

MachineType (81) property can be used to set the machine type of the object file to generate.

See also: TElfResourceWriter.MachineType (81), TAbstractResourceWriter (79), TElfResourceReader (79), Format of resources in object files (79)

9.7.2 Method overview

Page	Method	Description
81	Create	
81	Destroy	
81	GetDescription	
80	GetExtensions	
81	Write	

9.7.3 Property overview

Page	Properties	Access	Description
81	MachineType	rw	The machine type of the object file

9.7.4 TElfResourceWriter.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

9.7.5 TElfResourceWriter.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

9.7.6 TElfResourceWriter.Write

Declaration: `procedure Write(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

9.7.7 TElfResourceWriter.Create

Declaration: `constructor Create;  Override`

Visibility: public

9.7.8 TElfResourceWriter.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

9.7.9 TElfResourceWriter.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TElfMachineType`

Visibility: public

Access: Read,Write

Description: This property can be used to set the machine type of the object file to write.

If an attempt is made to set MachineType to an unsupported value, an EElfResourceWriterUnknown-MachineException (80) exception is raised.

See also: TElfMachineType (79), EElfResourceWriterUnknownMachineException (80)

Chapter 10

Reference for unit 'externalreader'

10.1 Used units

Table 10.1: Used units by unit 'externalreader'

Name	Page
Classes	??
externaltypes	85
resource	129
resourcetree	164
sysutils	??

10.2 Overview

This unit contains TExternalResourceReader (82), a TAbstractResourceReader (82) descendant that is able to read standalone resource files in a Free Pascal-specific format.

Adding this unit to a program's `uses` clause registers class TExternalResourceReader (82) with TResources (82).

See also

Free Pascal external resource file format description (82)

10.3 TExternalResourceReader

10.3.1 Description

This class provides a reader for .fpcres files: they are standalone files containing resources.

Standalone files are files that don't get linked with the final executable. They are used as a fallback solution on all those platforms for which an internal resource format is not available.

At runtime the resource file is read by Free Pascal RTL to provide resource support to the application.

After an external file has been read, Endianess (84) property holds the byte order used in the file.

See also: TExternalResourceReader.Endianess (84), TAbstractResourceReader (82), TExternalResourceWriter

(82)

10.3.2 Method overview

Page	Method	Description
83	CheckMagic	
83	Create	
83	Destroy	
83	GetDescription	
83	GetExtensions	
83	Load	

10.3.3 Property overview

Page	Properties	Access	Description
84	Endianess	r	The byte order used in the file

10.3.4 TExternalResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

10.3.5 TExternalResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

10.3.6 TExternalResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

10.3.7 TExternalResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

10.3.8 TExternalResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

10.3.9 TExternalResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

10.3.10 TExternalResourceReader.Endianess

Synopsis: The byte order used in the file

Declaration: `Property Endianess : Byte`

Visibility: public

Access: Read

Description: This property holds the byte order (endianess) of the file that has been read.

Remark Obviously, this property is meaningful only after a file has been read.

See also: EXT_ENDIAN_BIG (82), EXT_ENDIAN_LITTLE (82)

Chapter 11

Reference for unit 'externaltypes'

11.1 Overview

These types are used internally by TExternalResourceReader (85) and TExternalResourceWriter (85).

Two constants are of interest: EXT_ENDIAN_BIG (88) and EXT_ENDIAN_LITTLE (88).

11.2 Description of external resource file format

Introduction

An external resource file (.fpcres extension) provides resource support for those systems where a resource format suitable to be embedded in an object file isn't available.

The file format is designed in a way similar to other internal resource formats. The file is opened at program startup and is mapped in the program address space. Offsets in the file are easily converted to pointers at runtime since those offsets represent a displacement from a base address (the starting address where the file is mapped). Differences from an internal file format hence lie in the fact that resources aren't mapped in the program address space by the program loader but by Free Pascal RTL, and data must be accessed with a displacement mechanism instead of absolute pointers.

For internal resources details, see Format of resources in object files (85)

File layout

An external resource file consists of these parts:

- The initial header, containing various file information
- The resource tree, in the form of nodes
- The string table, which can be optional
- The resource data

The header is made up by initial header, resource tree and string table (if present).

Conventions

In this document, data sizes are specified with pascal-style data types. They are the following:

Table 11.1:

Name	Meaning
`byte`	Unsigned 8 bit integer.
`longword`	Unsigned 32 bit integer.
`qword`	Unsigned 64 bit integer.

Byte order used in the file is specified in the initial header.

All data structures in the file must be aligned on qword boundaries.

The initial header

An external resource file starts with this header:

Table 11.2:

Name	Offset	Length	Description
magic	0	6	Six ASCII characters that form the string `FPCRES`
version	6	byte	File format version. Currently it is `1`.
endianess	7	byte	Byte order. `1` for big endian, `2` for little endian
count	8	longword	Number of resources in the file
nodesize	12	longword	Size of header up to the string table, excluded
hdrsize	16	longword	Full size of header (up to the string table, included)
reserved	20	12	Must be zero

Note that byte order of the file can be read in the `endianess` field of the header. All data fields longer than a byte are written with the byte order specified in `endianess`.

If no resource name or type is identified by strings, string table is optional. When this is the case, `nodesize` and `hdrsize` have the same value.

The resource tree

Immediately following the initial header, the resource tree comes. It is made up by nodes that represent resource types, names and language ids.

Data is organized so that resource information (type, name and language id) is represented by a tree: root node contains resource types, that in turn contain resource names, which contain language ids, which describe resource data.

Given a node, its sub-nodes are ordered as follows:

- First the "named" nodes (nodes that use a string as identifier) come, then the id ones (nodes that use an integer as identifier).
- Named nodes are alphabetically sorted, in ascending order.
- Id nodes are sorted in ascending order.

In the file, all sub-nodes of a node are written in the order described above. Then, all sub-nodes of the first sub-node are written, and so on.

Example:

There are three resources:

1. a `BITMAP` resource with name `MYBITMAP` and language id `$0409`

2. a BITMAP resource with name 1 and language id 0
3. a resource with type MYTYPE and name 1 and language id 0

Nodes are laid out this way (note that BITMAP resources have type 2):

```
root | MYTYPE 2 | 1 | 0 | MYBITMAP 1 | $0409 | 0
```

That is, types (MYTYPE is a string, so it comes before 2 which is BITMAP), then names for MYTYPE (1), then language id for resource 3 (0), then names for BITMAP (MYBITMAP and 1), then language id for resource 1 ($0409), then language id for resource 2 (0).

Node format

Table 11.3:

Name	Offset	Length	Description
nameid	0	longword	name offset, integer id or language id
ncount	4	longword	named sub-nodes count
idcountsize	8	longword	id sub-nodes count or resource size
subptr	12	longword	offset to first sub-node

Note that all offset are always relative to the beginning of the file.

If the node is identified by a string, nameid is an offset to the null-terminated string holding the name. If it is identified by an id, nameid is that id. Language id nodes are always identified by and ID.

ncount is the number of named sub-nodes of this node (nodes that are identified by a string).

idcountsize is the number of id sub-nodes of this node (nodes that are identified by an integer id). For language id nodes, this field holds the size of the resource data.

subptr is an offset to the first subnode of this node. Note that it allows to access every subnode of this node, since subnodes of a node always come one after the other. For language id nodes, subptr is the offset to the resource data.

The string table

The string table is used to store strings used for resource types and names. If all resources use integer ids for name and types, it may not be present in the file.

The string table simply contains null-terminated strings, one after the other.

If present, the string table always contains a 0 (zero) at the beginning. This way, the empty string is located at the offset of the string table (whose value is held in nodesize field of the initial header).

The resource data

This part of the file contains raw resource data. As written before, all data structures must be aligned on qword boundaries, so if a resource data size is not a multiple of 8, bytes of padding must be inserted after that resource data.

11.3 Constants, types and variables

11.3.1 Constants

```
EXTERNAL_RESMAGIC : TExternalResMagic = 'FPCRES'
```

This value is used for TExtHeader.magic (88).

```
EXT_CURRENT_VERSION = 1
```

This value is used for TExtHeader.version (88).

```
EXT_ENDIAN_BIG = 1
```

This value is used for TExtHeader.endianess (88).

```
EXT_ENDIAN_LITTLE = 2
```

This value is used for TExtHeader.endianess (88).

11.3.2 Types

```
Array = Array[1..6] of Char
```

Type used for the magic identifier in external resource files

11.4 TExtHeader

```
TExtHeader = packed record
  magic : TExternalResMagic;
  version
  : Byte;
  endianess : Byte;
  count : LongWord;
  nodesize : LongWord
  ;
  hdrsize : LongWord;
  reserved1 : LongWord;
  reserved2 : LongWord
  ;
  reserved3 : LongWord;
end
```

This header describes the data structure present at the beginning of an external resource file.

11.5 TResInfoNode

```
TResInfoNode = packed record
  nameid : LongWord;
  ncount : LongWord
  ;
  idcountsize : LongWord;
  subptr : LongWord;
end
```

This record represents a node used in a resource tree. A node contains information about a certain resource type, name or language id.

Chapter 12

Reference for unit 'externalwriter'

12.1 Used units

Table 12.1: Used units by unit 'externalwriter'

Name	Page
Classes	??
externaltypes	85
resource	129
resourcetree	164
strtable	??
sysutils	??

12.2 Overview

This unit contains TExternalResourceWriter (90), a TAbstractResourceWriter (89) descendant that is able to write standalone resource files in a Free Pascal-specific format.

Adding this unit to a program's `uses` clause registers class TExternalResourceWriter (90) with TResources (89).

See also

Free Pascal external resource file format description (89)

12.3 EExternalResInvalidEndianessException

12.3.1 Description

This exception is raised when an attempt is made to set Endianess (91) property of a TExternalResourceWriter (90) object to a value other than EXT_ENDIAN_BIG (89) or EXT_ENDIAN_LITTLE (89).

See also: TExternalResourceWriter.Endianess (91)

12.4 EExternalResourceWriterException

12.4.1 Description

Base class for external resource writer-related exceptions

12.5 TExternalResourceWriter

12.5.1 Description

This class provides a writer for .fpcres files: they are standalone files containing resources.

Standalone files are files that don't get linked with the final executable. They are used as a fallback solution on all those platforms for which an internal resource format is not available.

At runtime the resource file is read by Free Pascal RTL to provide resource support to the application.

Endianess (91) property can be used to set the byte order to use in the file to generate.

See also: TExternalResourceWriter.Endianess (91), TAbstractResourceWriter (89), TExternalResourceReader (89)

12.5.2 Method overview

Page	Method	Description
91	Create	
91	Destroy	
90	GetDescription	
90	GetExtensions	
90	Write	

12.5.3 Property overview

Page	Properties	Access	Description
91	Endianess	rw	The byte order to use in the file

12.5.4 TExternalResourceWriter.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

12.5.5 TExternalResourceWriter.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

12.5.6 TExternalResourceWriter.Write

Declaration: `procedure Write(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

12.5.7 TExternalResourceWriter.Create

Declaration: `constructor Create; Override`

Visibility: public

12.5.8 TExternalResourceWriter.Destroy

Declaration: `destructor Destroy; Override`

Visibility: public

12.5.9 TExternalResourceWriter.Endianess

Synopsis: The byte order to use in the file

Declaration: `Property Endianess : Byte`

Visibility: public

Access: Read,Write

Description: This property can be used to set the byte order (endianess) of the file to write.

Remark If a value other than EXT_ENDIAN_BIG (89) or EXT_ENDIAN_LITTLE (89) is used, an EExternalResInvalidEndianessException (89) exception is raised.

See also: EXT_ENDIAN_BIG (89), EXT_ENDIAN_LITTLE (89)

Chapter 13

Reference for unit 'groupcursorresource'

13.1 Used units

Table 13.1: Used units by unit 'groupcursorresource'

Name	Page
Classes	??
groupresource	100
resource	129
sysutils	??

13.2 Overview

This unit contains TGroupCursorResource (92), a TAbstractResource (92) descendant specialized in handling resource of type RT_GROUP_CURSOR (92).

Adding this unit to a program's `uses` clause registers class TGroupCursorResource (92) for type RT_GROUP_CURSOR (92) with TResourceFactory (92).

13.3 TGroupCursorResource

13.3.1 Description

This class represents a resource of type RT_GROUP_CURSOR (92).

Resources of this type are strictly related to .cur files: typically a resource compiler creates resources of this type when it is instructed to insert a cursor from a .cur file.

There is although a big difference between .cur files and cursor resources: a .cur file contains a cursor, which is made of several different images (for different sizes and color depth), but while a file of this type is self-contained, when it comes to resources data is scattered over several different resources: an RT_GROUP_CURSOR (92) resource only contains information about the single images, which are contained each in a different resource of type RT_CURSOR (92). The single resources are pretty

unuseful alone, since they only consist of raw image data: they must be accessed in the contest of the RT_GROUP_CURSOR (92) resource, which provides information about them.

TGroupCursorResource (92) provides a way to handle a cursor as if it was a .cur file, via ItemData (92) property. Single cursor resources are automatically created or destroyed as needed.

Remark This class doesn't allow its type to be changed to anything else than RT_GROUP_CURSOR (92). Attempts to do so result in a EResourceDescChangeNotAllowedException (92).

See also: TGroupResource.ItemData (92), TGroupIconResource (92)

13.3.2 Method overview

Page	Method	Description
94	ChangeDescTypeAllowed	
94	ChangeDescValueAllowed	
94	ClearItemList	
94	Create	Creates a new group cursor resource
93	CreateSubItem	
94	DeleteSubItems	
94	GetName	
94	GetSubStream	
94	GetType	
93	ReadResourceItemHeader	
93	UpdateItemOwner	
93	WriteHeader	

13.3.3 TGroupCursorResource.ReadResourceItemHeader

Declaration: `procedure ReadResourceItemHeader; Override`

Visibility: protected

13.3.4 TGroupCursorResource.WriteHeader

Declaration: `procedure WriteHeader(aStream: TStream); Override`

Visibility: protected

13.3.5 TGroupCursorResource.CreateSubItem

Declaration: `procedure CreateSubItem; Override`

Visibility: protected

13.3.6 TGroupCursorResource.UpdateItemOwner

Declaration: `procedure UpdateItemOwner(index: Integer); Override`

Visibility: protected

13.3.7 TGroupCursorResource.ClearItemList

Declaration: `procedure ClearItemList;  Override`

Visibility: protected

13.3.8 TGroupCursorResource.DeleteSubItems

Declaration: `procedure DeleteSubItems;  Override`

Visibility: protected

13.3.9 TGroupCursorResource.GetSubStream

Declaration:
```
function GetSubStream(const index: Integer; out aSize: Int64) : TStream
                          ;  Override
```

Visibility: protected

13.3.10 TGroupCursorResource.GetType

Declaration: `function GetType : TResourceDesc;  Override`

Visibility: protected

13.3.11 TGroupCursorResource.GetName

Declaration: `function GetName : TResourceDesc;  Override`

Visibility: protected

13.3.12 TGroupCursorResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                  ;  Override
```

Visibility: protected

13.3.13 TGroupCursorResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                   ;  Override
```

Visibility: protected

13.3.14 TGroupCursorResource.Create

Synopsis: Creates a new group cursor resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                    ;  Override
```

Visibility: public

Description: Please note that `aType` parameter is not used, since this class always uses RT_GROUP_CURSOR (92) as type.

Chapter 14

Reference for unit 'groupiconresource'

14.1 Used units

Table 14.1: Used units by unit 'groupiconresource'

Name	Page
Classes	??
groupresource	100
resource	129
sysutils	??

14.2 Overview

This unit contains TGroupIconResource (96), a TAbstractResource (96) descendant specialized in handling resource of type RT_GROUP_ICON (96).

Adding this unit to a program's `uses` clause registers class TGroupIconResource (96) for type RT_GROUP_ICON (96) with TResourceFactory (96).

14.3 TGroupIconResource

14.3.1 Description

This class represents a resource of type RT_GROUP_ICON (96).

Resources of this type are strictly related to .ico files: typically a resource compiler creates resources of this type when it is instructed to insert an icon from an .ico file.

There is although a big difference between .ico files and icon resources: an .ico file contains an icon, which is made of several different images (for different sizes and color depth), but while a file of this type is self-contained, when it comes to resources data is scattered over several different resources: an RT_GROUP_ICON (96) resource only contains information about the single images, which are contained each in a different resource of type RT_ICON (96). The single resources are

pretty unuseful alone, since they only consist of raw image data: they must be accessed in the contest of the RT_GROUP_ICON (96) resource, which provides information about them.

TGroupIconResource (96) provides a way to handle an icon as if it was a .ico file, via ItemData (96) property. Single icon resources are automatically created or destroyed as needed.

Remark This class doesn't allow its type to be changed to anything else than RT_GROUP_ICON (96). Attempts to do so result in a EResourceDescChangeNotAllowedException (96).

See also: TGroupResource.ItemData (96), TGroupCursorResource (96)

14.3.2 Method overview

Page	Method	Description
98	ChangeDescTypeAllowed	
98	ChangeDescValueAllowed	
98	ClearItemList	
98	Create	Creates a new group icon resource
97	CreateSubItem	
98	DeleteSubItems	
98	GetName	
98	GetSubStream	
98	GetType	
97	ReadResourceItemHeader	
97	UpdateItemOwner	
97	WriteHeader	

14.3.3 TGroupIconResource.ReadResourceItemHeader

Declaration: `procedure ReadResourceItemHeader; Override`

Visibility: protected

14.3.4 TGroupIconResource.WriteHeader

Declaration: `procedure WriteHeader(aStream: TStream); Override`

Visibility: protected

14.3.5 TGroupIconResource.CreateSubItem

Declaration: `procedure CreateSubItem; Override`

Visibility: protected

14.3.6 TGroupIconResource.UpdateItemOwner

Declaration: `procedure UpdateItemOwner(index: Integer); Override`

Visibility: protected

14.3.7 TGroupIconResource.ClearItemList

Declaration: `procedure ClearItemList;  Override`

Visibility: protected

14.3.8 TGroupIconResource.DeleteSubItems

Declaration: `procedure DeleteSubItems;  Override`

Visibility: protected

14.3.9 TGroupIconResource.GetSubStream

Declaration:
```
function GetSubStream(const index: Integer; out aSize: Int64) : TStream
                          ;  Override
```

Visibility: protected

14.3.10 TGroupIconResource.GetType

Declaration: `function GetType : TResourceDesc;  Override`

Visibility: protected

14.3.11 TGroupIconResource.GetName

Declaration: `function GetName : TResourceDesc;  Override`

Visibility: protected

14.3.12 TGroupIconResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                  ;  Override
```

Visibility: protected

14.3.13 TGroupIconResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                   ;  Override
```

Visibility: protected

14.3.14 TGroupIconResource.Create

Synopsis: Creates a new group icon resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                    ;  Override
```

Visibility: public

Description: Please note that `aType` parameter is not used, since this class always uses RT_GROUP_ICON (96) as type.

Chapter 15

Reference for unit 'groupresource'

15.1 Used units

Table 15.1: Used units by unit 'groupresource'

Name	Page
Classes	??
resdatastream	119
resource	129
sysutils	??

15.2 Overview

This unit contains TGroupResource (101) and TGroupCachedDataStream (100), two classes used for resources of type RT_GROUP_ICON (100) and RT_GROUP_CURSOR (100).

The former is an abstract resource class which is implemented by TGroupIconResource (100) and TGroupCursorResource (100), and the latter is a TCachedDataStream (100) descendant used to provide .ico/.cur like streams for resource classes mentioned earlier.

This unit shouldn't be of interest for the user, who should look at documentation for groupiconresource (100) and groupcursorresource (100) units instead.

15.3 TGroupCachedDataStream

15.3.1 Description

This class is used by TGroupResource (101) descendants to provide an .ico/.cur like stream.

Unlike TCachedResourceDataStream (100), which provides a stream-like interface over a portion of another stream, this class lets multiple stream to be seen as one: this way, several RT_ICON (100) or RT_CURSOR (100) resources can appear like a single .ico or .cur file.

See also: TGroupResource (101), TGroupIconResource (100), TGroupCursorResource (100), TCachedDataStream (100), TCachedResourceDataStream (100)

15.3.2 Method overview

Page	Method	Description
101	Create	
101	Destroy	
101	Read	

15.3.3 TGroupCachedDataStream.Create

Declaration:
```
constructor Create(aStream: TStream; aResource: TAbstractResource;
                   aSize: Int64);  Override
```

Visibility: public

15.3.4 TGroupCachedDataStream.Destroy

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

Visibility: public

15.3.5 TGroupCachedDataStream.Read

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

Visibility: public

15.4 TGroupResource

15.4.1 Description

This class provides common functionalities that are extended by TGroupIconResource (100) and TGroupCursorResource (100).

Resources of type RT_GROUP_ICON (100) and RT_GROUP_CURSOR (100) represent a .ico or .cur file, respectively. However, data isn't contained in a single resource, but it's scattered over several different resources. That is, a .ico file contains an icon, which is made of several different images (for different sizes and color depth); when it is represented as a resource, however, the RT_GROUP_ICON (100) resource only contains information about the single images, which are contained each in a different resource of type RT_ICON (100). The single resources are pretty unuseful alone, since they only consist of raw image data: they must be accessed in the contest of the RT_GROUP_ICON (100) resource, which provides information about them.

TGroupIconResource (100) and TGroupCursorResource (100) provide a way to handle resources of these types as if they were .ico or .cur files. This class implements common functionalities, since icons and cursors are very similar.

Remark An object of this class should never be directly instantiated: use a descendant class instead.

See also: TGroupIconResource (100), TGroupCursorResource (100)

15.4.2 Method overview

Page	Method	Description
102	CheckBuildItemStream	
103	ClearItemList	
104	CompareContents	
103	CreateSubItem	
103	CreateSubItems	
103	DeleteSubItems	
104	Destroy	
102	FindSubResources	
102	GetItemData	
103	GetSubStream	
103	GetSubStreamCount	
104	NotifyResourcesLoaded	
102	ReadResourceItemHeader	
104	SetCustomItemDataStream	Sets a custom stream as the underlying stream for ItemData
104	SetOwnerList	
103	UpdateItemOwner	
105	UpdateRawData	
103	WriteHeader	
103	WriteResHeader	

15.4.3 Property overview

Page	Properties	Access	Description
105	ItemData	r	Resource data as an ICO/CUR stream

15.4.4 TGroupResource.FindSubResources

Declaration: `procedure FindSubResources`

Visibility: protected

15.4.5 TGroupResource.ReadResourceItemHeader

Declaration: `procedure ReadResourceItemHeader; Virtual; Abstract`

Visibility: protected

15.4.6 TGroupResource.CheckBuildItemStream

Declaration: `procedure CheckBuildItemStream`

Visibility: protected

15.4.7 TGroupResource.GetItemData

Declaration: `function GetItemData : TStream`

Visibility: protected

15.4.8 TGroupResource.WriteHeader

Declaration: `procedure WriteHeader(aStream: TStream); Virtual; Abstract`

Visibility: protected

15.4.9 TGroupResource.WriteResHeader

Declaration: `function WriteResHeader : Word`

Visibility: protected

15.4.10 TGroupResource.CreateSubItems

Declaration: `procedure CreateSubItems`

Visibility: protected

15.4.11 TGroupResource.CreateSubItem

Declaration: `procedure CreateSubItem; Virtual; Abstract`

Visibility: protected

15.4.12 TGroupResource.UpdateItemOwner

Declaration: `procedure UpdateItemOwner(index: Integer); Virtual; Abstract`

Visibility: protected

15.4.13 TGroupResource.ClearItemList

Declaration: `procedure ClearItemList; Virtual; Abstract`

Visibility: protected

15.4.14 TGroupResource.DeleteSubItems

Declaration: `procedure DeleteSubItems; Virtual; Abstract`

Visibility: protected

15.4.15 TGroupResource.GetSubStreamCount

Declaration: `function GetSubStreamCount : Integer`

Visibility: protected

15.4.16 TGroupResource.GetSubStream

Declaration:
```
function GetSubStream(const index: Integer; out aSize: Int64) : TStream
                      ; Virtual; Abstract
```

Visibility: protected

15.4.17 TGroupResource.SetOwnerList

Declaration: `procedure SetOwnerList(aResources: TResources);  Override`

Visibility: protected

15.4.18 TGroupResource.NotifyResourcesLoaded

Declaration: `procedure NotifyResourcesLoaded;  Override`

Visibility: protected

15.4.19 TGroupResource.Destroy

Synopsis:

Declaration: `destructor Destroy;  Override`

Visibility: public

Description:

15.4.20 TGroupResource.CompareContents

Declaration:
```
function CompareContents(aResource: TAbstractResource) : Boolean
                         ;  Override
```

Visibility: public

15.4.21 TGroupResource.SetCustomItemDataStream

Synopsis: Sets a custom stream as the underlying stream for ItemData

Declaration: `procedure SetCustomItemDataStream(aStream: TStream)`

Visibility: public

Description: This method allows the user to use a custom stream as the underlying stream for ItemData (105). This is useful when you want a TGroupIconResource (100) or TGroupCursorResource (100) to be created from a ico or cur file for which you have a stream.

Sample code

This code creates a resource containing an icon

```
var
  aName : TResourceDesc;
  aRes : TGroupIconResource;
  aFile : TFileStream;
  Resources : TResources;
begin
  Resources:=TResources.Create;
  aName:=TResourceDesc.Create('MAINICON');
  aRes:=TGroupIconResource.Create(nil,aName); //type is always RT_GROUP_ICON
  aName.Free; //not needed anymore
  aFile:=TFileStream.Create('mainicon.ico',fmOpenRead or fmShareDenyNone);
```

```
  aRes.SetCustomItemDataStream(aFile);
  Resources.Add(aRes);
  Resources.WriteToFile('myresource.res');

  Resources.Free; //it destroys aRes as well.
  aFile.Free;
end;
```

See also: TGroupResource.ItemData (105), TGroupIconResource (100), TGroupCursorResource (100), TAbstractResource.UpdateRawData (100)

15.4.22 TGroupResource.UpdateRawData

Declaration: `procedure UpdateRawData;  Override`

Visibility: public

15.4.23 TGroupResource.ItemData

Synopsis: Resource data as an ICO/CUR stream

Declaration: `Property ItemData : TStream`

Visibility: public

Access: Read

Description: This property gives access to resource data in a (ICO or CUR) file-like stream, unlike RawData (100).

The exact format of the stream (ico or cur) is determined by the descendant class of TGroupResource (101) that is used.

ItemData does not create a copy of RawData (100) so memory usage is generally kept limited.

You can also set a custom stream as the underlying stream for ItemData via SetCustomItemDataStream (104), much like SetCustomRawDataStream (100) does for RawData (100). This is useful when you want a TGroupIconResource (100) or TGroupCursorResource (100) to be created from a ico or cur file for which you have a stream.

Remark If you need to access RawData (100) after you modified ItemData, be sure to call UpdateRawData (100) first. This isn't needed however when resource is written to a stream, since TResources (100) takes care of it.

See also: TGroupResource.SetCustomItemDataStream (104), TGroupIconResource (100), TGroupCursorResource (100), TAbstractResource.RawData (100), TAbstractResource.UpdateRawData (100)

Chapter 16

Reference for unit 'machoreader'

16.1 Used units

Table 16.1: Used units by unit 'machoreader'

Name	Page
Classes	??
machotypes	109
resource	129
sysutils	??

16.2 Overview

This unit contains TMachOResourceReader (106), a TAbstractResourceReader (106) descendant that is able to read Mach-O object files containing resources.

Adding this unit to a program's `uses` clause registers class TMachOResourceReader (106) with TResources (106).

16.3 TMachOResourceReader

16.3.1 Description

This class provides a reader for Mach-O object files and images containing resources.

Mach-O is the file format used by Darwin and Mac OS X for object files and image files (executables, dynamic libraries and so on). Free Pascal can store resources in Mach-O files in its own format.

After an object file has been read, MachineType (108) property holds the machine type the object file was built for.

Remark This reader can't read multiple-architecture Mach-O files (like universal binary). To read a particular Mach-O file in a multiple-architecture file, extract it with lipo command.

See also: TMachOResourceReader.MachineType (108), TAbstractResourceReader (106), TMachOResourceWriter (106), Format of resources in object files (106)

16.3.2 Method overview

Page	Method	Description
107	CheckMagic	
107	Create	
107	Destroy	
107	GetDescription	
107	GetExtensions	
107	Load	

16.3.3 Property overview

Page	Properties	Access	Description
108	MachineType	r	The machine type of the object file

16.3.4 TMachOResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

16.3.5 TMachOResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

16.3.6 TMachOResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

16.3.7 TMachOResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

16.3.8 TMachOResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

16.3.9 TMachOResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

16.3.10 TMachOResourceReader.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TMachOMachineType`

Visibility: public

Access: Read

Description: This property holds the machine type of the object file that has been read.

Remark Obviously, this property is meaningful only after an object file has been read.

See also: TMachOMachineType (106)

Chapter 17

Reference for unit 'machotypes'

17.1 Overview

These constants are used internally by TMachOResourceWriter (109) and TMachOResourceReader (109).

The only type of interest for the user is TMachOMachineType (109).

17.2 Constants, types and variables

17.2.1 Types

```
PNList32 = ^TNList32
```

```
PNList64 = ^TNList64
```

```
PRelocationInfo = ^TRelocationInfo
```

```
TMachOMachineType = (mmtpowerpc,mmtpowerpc64,mmti386,mmtx86_64,mmtarm
  ,
                     mmtarm64)
```

Table 17.1: Enumeration values for type TMachOMachineType

Value	Explanation
mmtarm	
mmtarm64	
mmti386	Intel 386 machine type
mmtpowerpc	PowerPC machine type
mmtpowerpc64	PowerPC 64 machine type
mmtx86_64	AMD x86_64 machine type

This enumeration specifies the Mach-O machine type.

It is used by TMachOResourceWriter (109) to specify the machine type of the generated object file and by TMachOResourceReader (109) to read the machine type of the object file that has been read.

```
TMachOSubMachineType386 = (msm386_all)
```

Table 17.2: Enumeration values for type TMachOSubMachineType386

Value	Explanation
msm386_all	

```
TMachOSubMachineTypeAarch64 = (msmaarch64_all)
```

Table 17.3: Enumeration values for type TMachOSubMachineTypeAarch64

Value	Explanation
msmaarch64_all	

```
TMachOSubMachineTypeArm = (msmarm_all,msmarm_v4t,msmarm_v6,msmarm_v5tej
  ,
                           msmarm_xscale,msmarm_v7)
```

Table 17.4: Enumeration values for type TMachOSubMachineTypeArm

Value	Explanation
msmarm_all	
msmarm_v4t	
msmarm_v5tej	
msmarm_v6	
msmarm_v7	
msmarm_xscale	

```
TMachOSubMachineTypePowerPC = (msmppc_all)
```

Table 17.5: Enumeration values for type TMachOSubMachineTypePowerPC

Value	Explanation
msmppc_all	

```
TMachOSubMachineTypePowerPC64 = (msmppc64_all)
```

Table 17.6: Enumeration values for type TMachOSubMachineTypePowerPC64

Value	Explanation
msmppc64_all	

```
TMachOSubMachineTypex64 = (msmx64_all)
```

Table 17.7: Enumeration values for type TMachOSubMachineTypex64

Value	Explanation
msmx64_all	

```
Array = Array[0..15] of Char
```

17.3 TDySymtabCommand

```
TDySymtabCommand = record
  ilocalsym : LongWord;
  nlocalsym : LongWord
  ;
  iextdefsym : LongWord;
  nextdefsym : LongWord;
  iundefsym :
  LongWord;
  nundefsym : LongWord;
  tocoff : LongWord;
  ntoc : LongWord
  ;
  modtaboff : LongWord;
  nmodtab : LongWord;
  extrefsymoff : LongWord
  ;
  nextrefsyms : LongWord;
  indirectsymoff : LongWord;
  nindirectsyms
  : LongWord;
  extreloff : LongWord;
  nextrel : LongWord;
  locreloff
  : LongWord;
  nlocrel : LongWord;
end
```

17.4 TLoadCommand

```
TLoadCommand = record
  cmd : LongWord;
  cmdsize : LongWord;
end
```

17.5 TMachHdr

```
TMachHdr = record
  magic : LongWord;
  cputype : LongInt;
  cpusubtype
  : LongInt;
  filetype : LongWord;
  ncmds : LongWord;
  sizeofcmds
  : LongWord;
  flags : LongWord;
end
```

17.6 TNList32

```
TNList32 = record
  strx : LongWord;
  _type : Byte;
  sect : Byte
  ;
  desc : Word;
  value : LongWord;
end
```

17.7 TNList64

```
TNList64 = record
  strx : LongWord;
  _type : Byte;
  sect : Byte
  ;
  desc : Word;
  value : QWord;
end
```

17.8 TRelocationInfo

```
TRelocationInfo = record
  address : LongWord;
  flags : LongWord
  ;
end
```

17.9 TSection32

```
TSection32 = record
  sectname : TSegSectName;
  segname : TSegSectName
  ;
  addr : LongWord;
  size : LongWord;
  offset : LongWord;
  align
  : LongWord;
  reloff : LongWord;
  nreloc : LongWord;
  flags : LongWord
  ;
  reserved1 : LongWord;
  reserved2 : LongWord;
end
```

17.10 TSection64

```
TSection64 = record
  sectname : TSegSectName;
  segname : TSegSectName
  ;
  addr : QWord;
  size : QWord;
  offset : LongWord;
  align : LongWord
  ;
  reloff : LongWord;
  nreloc : LongWord;
  flags : LongWord;
  reserved1 : LongWord;
  reserved2 : LongWord;
  reserved3 : LongWord
  ;
end
```

17.11 TSegmentCommand32

```
TSegmentCommand32 = record
  name : TSegSectName;
  vmaddr : LongWord
  ;
  vmsize : LongWord;
  fileoff : LongWord;
  filesize : LongWord
  ;
```

```
  maxprot : LongInt;
  initprot : LongInt;
  nsects : LongWord;
  flags : LongWord;
end
```

17.12 TSegmentCommand64

```
TSegmentCommand64 = record
  name : TSegSectName;
  vmaddr : QWord
  ;
  vmsize : QWord;
  fileoff : QWord;
  filesize : QWord;
  maxprot
  : LongInt;
  initprot : LongInt;
  nsects : LongWord;
  flags : LongWord
  ;
end
```

17.13 TSymtabCommand

```
TSymtabCommand = record
  symoff : LongWord;
  nsyms : LongWord;
  stroff : LongWord;
  strsize : LongWord;
end
```

Chapter 18

Reference for unit 'machowriter'

18.1 Used units

Table 18.1: Used units by unit 'machowriter'

Name	Page
Classes	??
machotypes	109
resource	129
sysutils	??

18.2 Overview

This unit contains TMachOResourceWriter (116), a TAbstractResourceWriter (115) descendant that is able to write Mach-O object files containing resources.

Adding this unit to a program's `uses` clause registers class TMachOResourceWriter (116) with TResources (115).

18.3 Constants, types and variables

18.3.1 Types

```
TMachoSubMachineType = record
case TMachOMachineType of
msmppc_all
  : (
  fPpcSubType : TMachOSubMachineTypePowerPC;
);
msmppc64_all:
  (
  fPpc64SubType : TMachOSubMachineTypePowerPC64;
);
msm386_all
  : (
```

```
  f386SubType : TMachOSubMachineType386;
);
msmx64_all: (
  fX64SubType
  : TMachOSubMachineTypex64;
);
mmtarm: (
  fArmSubType : TMachOSubMachineTypeArm
  ;
);
mmtarm64: (
  fArm64SubType : TMachOSubMachineTypeAarch64;
)
  ;
end
```

18.4 EMachOResourceWriterException

18.4.1 Description

Base class for Mach-O resource writer-related exceptions

18.5 EMachOResourceWriterSymbolTableWrongOrderException

18.6 EMachOResourceWriterUnknownBitSizeException

18.6.1 Description

If this exception is raised, an internal error occurred.

18.7 TMachOResourceWriter

18.7.1 Description

This class provides a writer for Mach-O object files and images containing resources.

Mach-O is the file format used by Darwin and Mac OS X for object files and image files (executables, dynamic libraries and so on). Free Pascal can store resources in Mach-O files in its own format.

MachineType (117) property can be used to set the machine type of the object file to generate.

See also: TMachOResourceWriter.MachineType (117), TAbstractResourceWriter (115), TMachOResourceReader (115), Format of resources in object files (115)

18.7.2 Method overview

Page	Method	Description
117	Create	
117	Destroy	
117	GetDescription	
117	GetExtensions	
117	Write	

18.7.3 Property overview

Page	Properties	Access	Description
117	MachineType	rw	The machine type of the object file
118	SubMachineType	rw	

18.7.4 TMachOResourceWriter.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

18.7.5 TMachOResourceWriter.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

18.7.6 TMachOResourceWriter.Write

Declaration: `procedure Write(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

18.7.7 TMachOResourceWriter.Create

Declaration: `constructor Create;  Override`

Visibility: public

18.7.8 TMachOResourceWriter.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

18.7.9 TMachOResourceWriter.MachineType

Synopsis: The machine type of the object file

Declaration: `Property MachineType : TMachOMachineType`

Visibility: public

Access: Read,Write

Description: This property can be used to set the machine type of the object file to write.

See also: TMachOMachineType (115)

18.7.10 TMachOResourceWriter.SubMachineType

Declaration: `Property SubMachineType : TMachoSubMachineType`

Visibility: public

Access: Read,Write

Chapter 19

Reference for unit 'resdatastream'

19.1 Used units

Table 19.1: Used units by unit 'resdatastream'

Name	Page
Classes	??
resource	129
sysutils	??

19.2 Overview

This unit contains various streams that are used to provide copy-on-write mechanism for TAbstractResource.RawData (119), via more levels of indirection.

Main class is TResourceDataStream (122), which is the stream used for TAbstractResource.RawData (119). This class uses an underlying stream, to which it redirects operations.

At a lower level, a TCachedDataStream (120) descendant provides a layer between the original stream and the TResourceDataStream (122).

19.3 Constants, types and variables

19.3.1 Types

```
TCachedStreamClass = Class of TCachedDataStream
```

Cached stream metaclass

```
TUnderlyingStreamType = (usCached,usMemory,usCustom)
```

Table 19.2: Enumeration values for type TUnderlyingStreamType

Value	Explanation
usCached	The underlying stream is a TCachedResourceDataStream descendant
usCustom	The underlying stream is a custom stream
usMemory	The underlying stream is a memory stream

The type of the underlying stream of TResourceDataStream

19.4 TCachedDataStream

19.4.1 Description

This abstract class provides basic cached stream functionalities.

A cached stream is a read-only stream that operates on a portion of another stream. That is, it creates a "window" on the original stream from which to read data. Since it is a read-only stream, trying to write to the stream or to set its size cause an EInvalidOperation exception to be raised.

This class is used by TResourceDataStream (122) to access the raw data of a resource. When an attempt to write to the stream is detected, TResourceDataStream (122) replaces it with a memory stream and copies the contents of the cached stream to the memory one, thus providing a copy-on-write mechanism.

Remark An object of this class should never be directly instantiated: use a descendant class instead.

See also: TResourceDataStream (122), TCachedResourceDataStream (121)

19.4.2 Method overview

Page	Method	Description
121	Create	Creates a new object
120	GetPosition	
121	GetSize	
121	Seek	
120	SetPosition	
121	SetSize64	
121	Write	

19.4.3 TCachedDataStream.GetPosition

Declaration: `function GetPosition : Int64;  Override`

Visibility: protected

19.4.4 TCachedDataStream.SetPosition

Declaration: `procedure SetPosition(const Pos: Int64);  Override`

Visibility: protected

19.4.5 TCachedDataStream.GetSize

Declaration: `function GetSize : Int64;  Override`

Visibility: protected

19.4.6 TCachedDataStream.SetSize64

Declaration: `procedure SetSize64(const NewSize: Int64);  Override`

Visibility: protected

19.4.7 TCachedDataStream.Create

Synopsis: Creates a new object

Declaration:
```
constructor Create(aStream: TStream; aResource: TAbstractResource;
                   aSize: Int64);  Virtual
```

Visibility: public

Description: A new cached stream is created on top of the `aStream` stream.

See also: TCachedDataStream (120), TResourceDataStream (122), TResourceDataStream.Create (123)

19.4.8 TCachedDataStream.Write

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

Visibility: public

19.4.9 TCachedDataStream.Seek

Declaration:
```
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
              ;  Override
```

Visibility: public

19.5 TCachedResourceDataStream

19.5.1 Description

This class provides an implementation of TCachedDataStream (120).

Usually resource readers create a TResourceDataStream (122) with a TCachedResourceDataStream as the underlying stream.

See also: TCachedDataStream (120), TResourceDataStream (122)

19.5.2 Method overview

Page	Method	Description
122	Create	
122	Read	

19.5.3 TCachedResourceDataStream.Create

Declaration: `constructor Create(aStream: TStream; aResource: TAbstractResource; aSize: Int64); Override`

Visibility: public

19.5.4 TCachedResourceDataStream.Read

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

Visibility: public

19.6 TResourceDataStream

19.6.1 Description

This class provides the copy-on-write mechanism of TAbstractResource.RawData (119), via more levels of indirection.

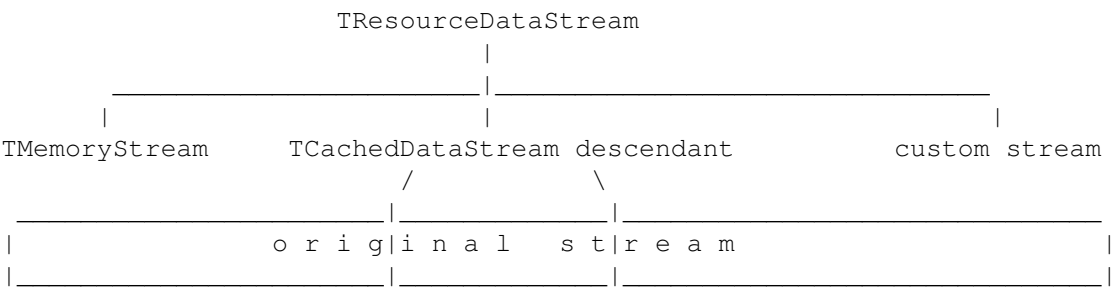
It uses an underlying stream, to which it redirects operations.

The underlying stream can be a TCachedDataStream (120) descendant, a memory stream or a custom stream. Usually when a resource is loaded from a stream, the underlying stream is a TCachedDataStream (120) descendant, which provides a read-only stream-like interface over a portion of the original stream (that is, the part of the original stream where resource data resides). When TResourceDataStream (122) is requested to write data, it replaces the underlying stream with a memory stream, whose contents are copied from the previous underlying stream: this way, copy-on-write functionality can be achieved.

As said before, third possibility is to have a custom stream as the underlying stream: a user can set this stream via TAbstractResource.SetCustomRawDataStream (119) method, which in turn calls TResourceDataStream.SetCustomStream (124)

Figure: Levels of indirection

.



See also: TCachedDataStream (120), TResourceDataStream.Create (123), TResourceDataStream.SetCustomStream (124)

19.6.2 Method overview

Page	Method	Description
124	Compare	Compares the stream to another one
123	Create	Creates a new object
124	Destroy	
123	GetPosition	
123	GetSize	
124	Read	
125	Seek	
124	SetCustomStream	Sets a custom stream as the underlying stream
123	SetPosition	
123	SetSize64	
124	Write	

19.6.3 Property overview

Page	Properties	Access	Description
125	Cached	rw	Controls the copy-on-write behaviour of the stream

19.6.4 TResourceDataStream.GetPosition

Declaration: `function GetPosition : Int64;  Override`

Visibility: protected

19.6.5 TResourceDataStream.SetPosition

Declaration: `procedure SetPosition(const Pos: Int64);  Override`

Visibility: protected

19.6.6 TResourceDataStream.GetSize

Declaration: `function GetSize : Int64;  Override`

Visibility: protected

19.6.7 TResourceDataStream.SetSize64

Declaration: `procedure SetSize64(const NewSize: Int64);  Override`

Visibility: protected

19.6.8 TResourceDataStream.Create

Synopsis: Creates a new object

Declaration:
```
constructor Create(aStream: TStream; aResource: TAbstractResource;
                   aSize: Int64; aClass: TCachedStreamClass)
```

Visibility: public

Description: A new TResourceDataStream (122) object is created.

If `aStream` is `nil`, the underlying stream is a memory stream. Otherwise, a cached stream of the class specified in `aClass` is created and set as the underlying stream.

See also: TResourceDataStream (122), TCachedDataStream (120), TResourceDataStream.SetCustomStream (124)

19.6.9 TResourceDataStream.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

19.6.10 TResourceDataStream.Compare

Synopsis: Compares the stream to another one

Declaration: `function Compare(aStream: TStream) : Boolean`

Visibility: public

Description: This methods compares the stream with `aStream`. If they are of the same length and their contents are the same, `true` is returned, `false` otherwise.

See also: TAbstractResource.CompareContents (119)

19.6.11 TResourceDataStream.SetCustomStream

Synopsis: Sets a custom stream as the underlying stream

Declaration: `procedure SetCustomStream(aStream: TStream)`

Visibility: public

Description: This method sets a custom stream as the underlying stream.

If `aStream` is `nil`, a new memory stream is used as the underlying stream. This can be used to remove a previously set custom stream as the underlying stream.

Usually it is called by TAbstractResource.SetCustomRawDataStream (119).

See also: TResourceDataStream (122), TAbstractResource.SetCustomRawDataStream (119)

19.6.12 TResourceDataStream.Read

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

Visibility: public

19.6.13 TResourceDataStream.Write

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

Visibility: public

19.6.14 TResourceDataStream.Seek

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64 ; Override`

Visibility: public

19.6.15 TResourceDataStream.Cached

Synopsis: Controls the copy-on-write behaviour of the stream

Declaration: `Property Cached : Boolean`

Visibility: public

Access: Read,Write

Description: When this property is set to `true`, a cached stream is used as the underlying stream for read operations. If it is set to `false`, no caching is performed and data is always copied to a memory stream.

Note that this property does nothing if the underlying stream is a custom stream.

By default this property is true.

See also: TResourceDataStream (122), TAbstractResource.CacheData (119)

Chapter 20

Reference for unit 'resfactory'

20.1 Used units

Table 20.1: Used units by unit 'resfactory'

Name	Page
Classes	??
resource	129
sysutils	??

20.2 Overview

This unit contains a factory class that provides an easy way to create resources of the right class.

Resource classes can be registered with TResourceFactory (127) so that the class knows how to create a resource of a specific type.

20.3 Constants, types and variables

20.3.1 Resource strings

```
SAlreadyRegistered =
  'A resource class for the type %s is already registered.'
```

20.4 EResourceClassAlreadyRegisteredException

20.4.1 Description

This exception is raised by class method RegisterResourceClass (127) of TResourceFactory (127) when an attempt is made to register a class for an already registered type.

See also: TResourceFactory.RegisterResourceClass (127)

20.5 EResourceFactoryException

20.5.1 Description

Base class for resource factory-related exceptions

20.6 TResourceFactory

20.6.1 Description

Resources are represented by descendants of TAbstractResource (126). Some applications don't need specialized resource classes, and a TGenericResource (126) can be enough. On the other hand, sometimes it is required that a resource of a specific type is created with a more specialized class. This class provides a centralized point for the creation of resources.

TResourceFactory (127) holds a list of registered classes with an associated resource type. When it is requested to create a resource for a given type, it creates a resource of the class associated with that type. If no class matching that type is found, TGenericResource (126) is used.

Usually each resource class registers itself in the `initialization` section of the unit in which it is implemented.

See also: TResourceFactory.RegisterResourceClass (127), TResourceFactory.CreateResource (127), TAbstractResource (126), TGenericResource (126)

20.6.2 Method overview

Page	Method	Description
127	CreateResource	Creates a new resource
127	RegisterResourceClass	Registers a resource class

20.6.3 TResourceFactory.RegisterResourceClass

Synopsis: Registers a resource class

Declaration:
```
class procedure RegisterResourceClass(aType: TResID;
                                      aClass: TResourceClass);  Overload
class procedure RegisterResourceClass(aType: TResName;
                                      aClass: TResourceClass);  Overload
class procedure RegisterResourceClass(aType: TResourceDesc;
                                      aClass: TResourceClass);  Overload
```

Visibility: public

Description: This class method registers a resource class for the given resource type.

Errors: If a class has already been registered for the given resource type, an EResourceClassAlreadyRegisteredException (126) exception is raised.

See also: TResourceFactory (127)

20.6.4 TResourceFactory.CreateResource

Synopsis: Creates a new resource

Declaration: `class function CreateResource(aType: TResourceDesc; aName: TResourceDesc) : TAbstractResource`

Visibility: public

Description: This class method creates a new resource of the class associated with the given type, and sets its name and type based on the values passed as parameters.

If no class matching the given type is found, the resource is created with TGenericResource (126) class.

See also: TResourceFactory (127), TResourceFactory.RegisterResourceClass (127), TGenericResource (126)

Chapter 21

Reference for unit 'resource'

21.1 Used units

Table 21.1: Used units by unit 'resource'

Name	Page
Classes	??
sysutils	??

21.2 Overview

This unit contains base classes needed to work with resources.

Single resources are represented by an instance of a TAbstractResource (135) descendant. They are grouped in a TResources (156) instance which can be read (written) to (from) a stream via a TAbstractResourceReader (145) (TAbstractResourceWriter (150)) descendant.

TGenericResource (152) provides a basic implementation of TAbstractResource (135).

21.3 Constants, types and variables

21.3.1 Resource strings

```
SDescChangeNotAllowed = 'Cannot modify %s resource description'
```

```
SLangIDChangeNotAllowed = 'Cannot modify %s resource language ID'
```

```
SReaderNotFoundExt =
  'Cannot find resource reader for extension ''%s'''
```

```
SReaderNotFoundProbe =
  'Cannot find a resource reader: unknown format.'
```

```
SResDuplicate =
  'Duplicate resource: Type = %s, Name = %s, Lang ID = %.4x'


SResourceNotFound = 'Cannot find resource: Type = %s, Name = %s'


SResourceNotFoundLang =
  'Cannot find resource: Type = %s, Name = %s, Lang ID = %.4x'


SWriterNotFoundExt =
  'Cannot find resource writer for extension ''%s'''
```

21.3.2 Constants

```
CREATEPROCESS_MANIFEST_RESOURCE_ID = 1


ISOLATIONAWARE_MANIFEST_RESOURCE_ID = 2


ISOLATIONAWARE_NOSTATICIMPORT_MANIFEST_RESOURCE_ID = 3


MAXIMUM_RESERVED_MANIFEST_RESOURCE_ID = 16


MF_DISCARDABLE = $1000
```

This flag is ignored by Windows and Free Pascal RTL. It's provided for compatibility with 16-bit Windows.

```
MF_MOVEABLE = $0010
```

This flag is ignored by Windows and Free Pascal RTL. It's provided for compatibility with 16-bit Windows.

```
MF_PRELOAD = $0040
```

This flag is ignored by Windows and Free Pascal RTL. It's provided for compatibility with 16-bit Windows.

```
MF_PURE = $0020
```

This flag is ignored by Windows and Free Pascal RTL. It's provided for compatibility with 16-bit Windows.

```
MINIMUM_RESERVED_MANIFEST_RESOURCE_ID = 1


RT_ACCELERATOR = 9
```

Accelerator table resource

```
RT_ANICURSOR = 21
```

This resource type contains raw binary data taken from a .ani file

```
RT_ANIICON = 22
```

This resource type contains raw binary data taken from a .ani file

```
RT_BITMAP = 2
```

Bitmap resource

```
RT_CURSOR = 1
```

A single image in a cursor. Don't use it directly.

```
RT_DIALOG = 5
```

Dialog resource

```
RT_DLGINCLUDE = 17
```

This resource is used internally by resource compilers but will never appear in compiled form

```
RT_FONT = 8
```

This resource type is obsolete and never appears in 32 bit resources.

```
RT_FONTDIR = 7
```

This resource type is obsolete and never appears in 32 bit resources.

```
RT_GROUP_CURSOR = 12
```

This resource type contains a cursor and it's the equivalent of a .cur file

Remark Please note that is is made up of several RT_CURSOR (131) resources (the single cursor images) that shouldn't be accessed singularly.

```
RT_GROUP_ICON = 14
```

This resource type contains an icon and it's the equivalent of a .ico file

Remark Please note that is is made up of several RT_ICON (132) resources (the single icon images) that shouldn't be accessed singularly.

```
RT_HTML = 23
```

This resource type contains an HTML file.

```
RT_ICON = 3
```

A single image in a icon. Don't use it directly.

```
RT_MANIFEST = 24
```

This resource contains data taken from a .manifest file

Remark Resource name must be one of CREATEPROCESS_MANIFEST_RESOURCE_ID (130) (mainly used for executables), ISOLATIONAWARE_MANIFEST_RESOURCE_ID (130) or ISOLATIONAWARE_NOSTATICIMPORT_MANIFEST_RESOURCE_ID (130) (mainly used for DLLs)

```
RT_MENU = 4
```

Menu resource

```
RT_MESSAGETABLE = 11
```

Message table resource

```
RT_PLUGPLAY = 19
```

Plug and Play resource

```
RT_RCDATA = 10
```

This resource type contains arbitrary binary data

Note that Delphi dfm files are stored in compiled form as a RCDATA resource

```
RT_STRING = 6
```

String table resource

```
RT_VERSION = 16
```

This resource defines version information which is visible when viewing properties of a Windows executable or DLL.

```
RT_VXD = 20
```

VXD resource

21.3.3 Types

```
TDescType = (dtName,dtID)
```

Table 21.2: Enumeration values for type TDescType

Value	Explanation
dtID	The resource type or name is an ID
dtName	The resource type or name is a string

The type of a resource type or name

```
TLangID = Word
```

A resource language ID

```
TResID = LongWord
```

A resource type or name in ID form

```
TResName = String
```

A resource type or name in string form

```
TResourceClass = Class of TAbstractResource
```

Resource metaclass

```
TResourceReaderClass = Class of TAbstractResourceReader
```

Resource reader metaclass

```
TResourceWriterClass = Class of TAbstractResourceWriter
```

Resource writer metaclass

21.4 ENoMoreFreeIDsException

21.4.1 Description

This exception is raised by TResources.AddAutoID (158) method when it is not possible to generate an ID to use as a name for the given resource, because all possible 65536 IDs are already assigned to resources of the same type as the given one.

See also: TResources.AddAutoID (158)

21.5 EResourceDescChangeNotAllowedException

21.5.1 Description

This exception is raised when a resource description (either type or name) is tried to be changed, but the resource class doesn't allow it.

See also: TAbstractResource._Type (141), TAbstractResource.Name (141)

21.6 EResourceDescTypeException

21.6.1 Description

This exception is raised when a resource description is of type dtName (132) and TResourceDesc.ID (156) property is read.

See also: TResourceDesc.ID (156)

21.7 EResourceDuplicateException

21.7.1 Description

This exception is raised when a resource is being added to a TResources (156) object, but another resource with the same type, name and language ID already exists.

See also: TResources.Add (157), TResources.MoveFrom (159)

21.8 EResourceException

21.8.1 Description

Base class for resource-related exceptions

21.9 EResourceLangIDChangeNotAllowedException

21.9.1 Description

This exception is raised when the resource language ID is tried to be changed, but the resource is contained in a TResources (156) object.

See also: TAbstractResource.LangID (141)

21.10 EResourceNotFoundException

21.10.1 Description

This exception is raised when searching for a resource in a TResources (156) object fails.

See also: TResources.Find (158), TResources.Remove (159)

21.11 EResourceReaderException

21.11.1 Description

Base class for resource reader-related exceptions

21.12 EResourceReaderNotFoundException

21.12.1 Description

This exception is raised when no TAbstractResourceReader (145) descendant able to read a stream was found.

See also: TResources.FindReader (159), TResources.LoadFromStream (160), TResources.LoadFromFile (160)

21.13 EResourceReaderUnexpectedEndOfStreamException

21.13.1 Description

This exception is raised by Load (148) method of a TAbstractResourceReader (145) descendant when the stream it was asked to read resources from ended prematurely.

See also: TAbstractResourceReader.Load (148)

21.14 EResourceReaderWrongFormatException

21.14.1 Description

This exception is raised by Load (148) method of a TAbstractResourceReader (145) descendant when the stream it was asked to read resources from is not in the format it supports.

See also: TAbstractResourceReader.Load (148)

21.15 EResourceWriterException

21.15.1 Description

Base class for resource writer-related exceptions

21.16 EResourceWriterNotFoundException

21.16.1 Description

This exception is raised by WriteToFile (162) method of TResources (156) when no TAbstractResourceWriter (150) descendant matching filename extension was found.

See also: TResources.WriteToFile (162)

21.17 TAbstractResource

21.17.1 Description

This is the base class that represents a resource.

A resource is identified by its type (141), name (141) and language ID (141) even if some file formats or operating systems don't consider the latter.

There are also additional properties that aren't always present in all file formats, so their values aren't always meaningful: however, they can be used to display detailed information when possible.

Every resource has a RawData (144) stream that holds resource data. This stream uses a copy-on-write mechanism: if the resource has been read from a stream or file, RawData (144) redirects read operations to the original stream. This is particularly useful when a resource file must be converted from a format to another, or when more resource files must be merged, since (potentially large) resource data is directly copied from the original to the destination stream without the need of allocating a lot of memory.

When resource data is encoded in a resource-specific format, RawData (144) can be uncomfortable: it's often better to use a more specialized descendant class that provides additional properties and methods.

Resources cannot be read or written alone from/to a stream: they need to be contained in a TResources (156) object, which represents an abstract view of a resource file.

Usually each descendant registers itself with TResourceFactory (129) class in the `initialization` section of the unit in which it is implemented: this way TResourceFactory (129) class can know which class to use to instantiate a resource of a given type.

Remark An object of this class should never be directly instantiated: use a descendant class instead.

See also: TGenericResource (152), TAcceleratorsResource (129), TBitmapResource (129), TGroupCursorResource (129), TGroupIconResource (129), TStringTableResource (129), TVersionResource (129), TResources (156), TResourceFactory (129)

21.17.2 Method overview

Page	Method	Description
138	ChangeDescTypeAllowed	Reports whether changing the type of resource type or name is allowed
138	ChangeDescValueAllowed	Reports whether changing the value of resource type or name is allowed
139	CompareContents	Compares the contents of the resource to the contents of another one
139	Create	Creates a new resource
139	Destroy	Destroys the object
138	GetName	Returns the name of the resource
137	GetType	Returns the type of the resource
139	NotifyResourcesLoaded	Tells the resource that all resources have been loaded
137	SetChildOwner	Protected method to let a resource set itself as the owner of a sub-resource
140	SetCustomRawDataStream	Sets a custom stream as the underlying stream for RawData
137	SetDescOwner	Sets this resource as the owner of the given TResourceDesc
137	SetOwnerList	Protected method to let a resource list set itself as the owner of the resource
140	UpdateRawData	Updates RawData stream.

21.17.3 Property overview

Page	Properties	Access	Description
144	CacheData	rw	Controls the copy-on-write behaviour of the resource
143	Characteristics	rw	A user defined piece of data
143	CodePage	rw	The code page of the resource
143	DataOffset	r	The offset of resource data from the beginning of the stream
141	DataSize	r	The size of resource raw data
142	DataVersion	rw	The version of the resource data
142	HeaderSize	r	The size of resource header
141	LangID	rw	The language ID of the resource
142	MemoryFlags	rw	The memory flags of the resource
141	Name	r	The name of the resource
145	Owner	r	The owner of this resource
144	OwnerList	r	The resource list that owns this resource
144	RawData	r	The raw resource data stream
143	Version	rw	A user defined version number
141	_Type	r	The type of the resource

21.17.4 TAbstractResource.SetDescOwner

Synopsis: Sets this resource as the owner of the given TResourceDesc

Declaration: `procedure SetDescOwner(aDesc: TResourceDesc)`

Visibility: protected

Description: This method is provided so that descendants of TAbstractResource (135) can set themselves as the owners of the given TResourceDesc

21.17.5 TAbstractResource.SetOwnerList

Synopsis: Protected method to let a resource list set itself as the owner of the resource

Declaration: `procedure SetOwnerList(aResources: TResources); Virtual`

Visibility: protected

21.17.6 TAbstractResource.SetChildOwner

Synopsis: Protected method to let a resource set itself as the owner of a sub-resource

Declaration: `procedure SetChildOwner(aChild: TAbstractResource)`

Visibility: protected

21.17.7 TAbstractResource.GetType

Synopsis: Returns the type of the resource

Declaration: `function GetType : TResourceDesc; Virtual; Abstract`

Visibility: protected

Description: Descendant classes must implement this method to provide access to the resource type.

21.17.8 TAbstractResource.GetName

Synopsis: Returns the name of the resource

Declaration:
```
function GetName : TResourceDesc;  Virtual;  Abstract
```

Visibility: protected

Description: Descendant classes must implement this method to provide access to the resource name.

21.17.9 TAbstractResource.ChangeDescTypeAllowed

Synopsis: Reports whether changing the type of resource type or name is allowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean;  Virtual
                                    ;  Abstract
```

Visibility: protected

Description: Descendant classes must implement this method to declare if the resource allows changing the type of one of its resource description (type or name): that is, if it allows one of its descriptions type to change from dtName (132) to dtID (132) or vice versa.

Example:

A certain resource class allows its name only to be changed: e.g. a TBitmapResource (129) doesn't want its type to be anything else than RT_BITMAP (131). It then allows changing the type of the description only if the description is the resource name:

```
Result:=aDesc=fName;
```

See also: TAbstractResource.ChangDescValueAllowed (135)

21.17.10 TAbstractResource.ChangeDescValueAllowed

Synopsis: Reports whether changing the value of resource type or name is allowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                    ;  Virtual;  Abstract
```

Visibility: protected

Description: Descendant classes must implement this method to declare if the resource allows changing the value of one of its resource description (type or name).

Example:

A certain resource class allows its name only to be changed: e.g. a TBitmapResource (129) doesn't want its type to be anything else than RT_BITMAP (131). It then allows changing the value of the description only if the description is the resource name:

```
Result:=aDesc=fName;
```

See also: TAbstractResource.ChangDescTypeAllowed (135)

21.17.11 TAbstractResource.NotifyResourcesLoaded

Synopsis: Tells the resource that all resources have been loaded

Declaration:
```
procedure NotifyResourcesLoaded;  Virtual;  Abstract
```

Visibility: protected

Description: This method is called by a TResources (156) object when the loading of all resources from a stream has completed.

Example:

A Group resource (e.g. TGroupIconResource (129)) can use this method to find all its sub-resources, since all resources have been loaded from a stream.

21.17.12 TAbstractResource.Create

Synopsis: Creates a new resource

Declaration:
```
constructor Create;  Virtual;  Overload
constructor Create(aType: TResourceDesc; aName: TResourceDesc);  Virtual
                  ;  Abstract;  Overload
```

Visibility: protected

Description: A new resource is created with the given type and name.

Remark Please note that the resource doesn't take ownership of the TResourceDesc (154) objects passed as parameters, it simply copies them: it's user responsibility to free them when no longer needed.

21.17.13 TAbstractResource.Destroy

Synopsis: Destroys the object

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

Visibility: public

21.17.14 TAbstractResource.CompareContents

Synopsis: Compares the contents of the resource to the contents of another one

Declaration:
```
function CompareContents(aResource: TAbstractResource) : Boolean
                              ;  Virtual
```

Visibility: public

Description: This methods compares the contents of the resource with the ones of `aResource`. If they are of the same length and their contents are the same, `true` is returned, `false` otherwise.

Usually this methods compares the contents of the two RawData (144) streams, calling TResourceDataStream.Compare (129), but descendent classes can implement a different algorithm.

See also: TResourceDataStream.Compare (129)

21.17.15 TAbstractResource.UpdateRawData

Synopsis: Updates RawData stream.

Declaration: `procedure UpdateRawData; Virtual; Abstract`

Visibility: public

Description: When operating on resource data with more high-level streams than RawData (144) (e.g: TBitmapResource.BitmapData (129)) RawData contents are no longer valid. This method ensures that RawData (144) stream is properly synchronized with the contents of the higher-level stream.

Remark Normally a resource writer doesn't need to call this method when it is about to write the resource data to a stream, since TResources (156) class takes care of this before telling the resource writer to write resources to a stream.

See also: TAbstractResource.RawData (144)

21.17.16 TAbstractResource.SetCustomRawDataStream

Synopsis: Sets a custom stream as the underlying stream for RawData

Declaration: `procedure SetCustomRawDataStream(aStream: TStream)`

Visibility: public

Description: Normally, RawData (144) uses a memory stream or the original resource stream (e.g. the original file containing the resource) as its underlying stream. This method allows the user to use a custom stream as the underlying stream. This can be useful when a resource must be created from the contents of an original file as-is.

If `aStream` is `nil`, a new memory stream is used as the underlying stream. This can be used to remove a previously set custom stream as the underlying stream.

Sample code

This code creates a resource containing an html file

```
var
  aType, aName : TResourceDesc;
  aRes : TGenericResource;
  aFile : TFileStream;
begin
  aType:=TResourceDesc.Create(RT_HTML);
  aName:=TResourceDesc.Create('index');
  aRes:=TGenericResource.Create(aType,aName);
  aFile:=TFileStream.Create('index.html',fmOpenRead or fmShareDenyNone);
  aRes.SetCustomRawDataStream(aFile);

  //do something...

  aRes.Free;
  aFile.Free;
  aType.Free;
  aName.Free;
end;
```

See also: TAbstractResource.RawData (144)

21.17.17 TAbstractResource._Type

Synopsis: The type of the resource

Declaration: `Property _Type : TResourceDesc`

Visibility: public

Access: Read

Description: **Remark:** Please note that some TAbstractResource (135) descendants don't allow resource type to be changed (e.g: it's not possible for a TBitmapResource (129) to have a type other than RT_BITMAP (131)). If it is the case, an EResourceDescChangeNotAllowedException (133) exception is raised.

Moreover, if the resource is contained in a TResources (156) object, type change is not allowed.

See also: TAbstractResource.ChangeDescTypeAllowed (138), TAbstractResource.ChangeDescValueAllowed (138)

21.17.18 TAbstractResource.Name

Synopsis: The name of the resource

Declaration: `Property Name : TResourceDesc`

Visibility: public

Access: Read

Description: **Remark:** Please note that some TAbstractResource (135) descendants don't allow resource name to be changed (e.g: a TStringTableResource (129) name is determined by the range of strings' ID it contains). If it is the case, an EResourceDescChangeNotAllowedException (133) exception is raised.

Moreover, if the resource is contained in a TResources (156) object, name change is not allowed.

See also: TAbstractResource.ChangeDescTypeAllowed (138), TAbstractResource.ChangeDescValueAllowed (138)

21.17.19 TAbstractResource.LangID

Synopsis: The language ID of the resource

Declaration: `Property LangID : TLangID`

Visibility: public

Access: Read,Write

Description: **Remark:** Please note that if the resource is contained in a TResources (156) object, language ID change is not allowed: trying to do so results in an EResourceLangIDChangeNotAllowedException (134) exception being raised.

21.17.20 TAbstractResource.DataSize

Synopsis: The size of resource raw data

Declaration: `Property DataSize : LongWord`

Visibility: public

Access: Read

Description: DataSize is the length, in bytes, of the resource data, accessible via RawData (144) property.

See also: TAbstractResource.RawData (144), TAbstractResource.DataOffset (143)

21.17.21 TAbstractResource.HeaderSize

Synopsis: The size of resource header

Declaration: `Property HeaderSize : LongWord`

Visibility: public

Access: Read

Description: This property is not always meaningful, since not all file formats support it.

Its value, when nonzero, can be used for information purposes.

21.17.22 TAbstractResource.DataVersion

Synopsis: The version of the resource data

Declaration: `Property DataVersion : LongWord`

Visibility: public

Access: Read,Write

Description: This property is not always meaningful, since not all file formats support it.

Its value, when nonzero, can be used for information purposes.

21.17.23 TAbstractResource.MemoryFlags

Synopsis: The memory flags of the resource

Declaration: `Property MemoryFlags : Word`

Visibility: public

Access: Read,Write

Description: This field is a combination of the following flags

- MF_MOVEABLE (130)
- MF_PURE (130)
- MF_PRELOAD (130)
- MF_DISCARDABLE (130)

By default, a newly created resource has MF_MOVEABLE (130) and MF_DISCARDABLE (130) flags set.

Remark Please note that memory flags are ignored by Windows and Free Pascal RTL. They are provided only for compatibility with 16-bit Windows.

This property is not always meaningful, since not all file formats support it.

Its value, when nonzero, can be used for information purposes.

21.17.24 TAbstractResource.Version

Synopsis: A user defined version number

Declaration: `Property Version : LongWord`

Visibility: public

Access: Read,Write

Description: A tool that writes resource files can write version information in this field.
This property is not always meaningful, since not all file formats support it.
Its value, when nonzero, can be used for information purposes.

See also: TAbstractResource.Characteristics (143)

21.17.25 TAbstractResource.Characteristics

Synopsis: A user defined piece of data

Declaration: `Property Characteristics : LongWord`

Visibility: public

Access: Read,Write

Description: A tool that writes resource files can write arbitrary data in this field.
This property is not always meaningful, since not all file formats support it.
Its value, when nonzero, can be used for information purposes.

See also: TAbstractResource.Version (143)

21.17.26 TAbstractResource.DataOffset

Synopsis: The offset of resource data from the beginning of the stream

Declaration: `Property DataOffset : LongWord`

Visibility: public

Access: Read

Description: A reader sets this property to let the resource know where its raw data begins in the resource stream.

See also: TAbstractResource.RawData (144), TAbstractResource.DataSize (141)

21.17.27 TAbstractResource.CodePage

Synopsis: The code page of the resource

Declaration: `Property CodePage : LongWord`

Visibility: public

Access: Read,Write

Description: This property is not always meaningful, since not all file formats support it.
Its value, when nonzero, can be used for information purposes.

21.17.28 TAbstractResource.RawData

Synopsis: The raw resource data stream

Declaration: `Property RawData : TStream`

Visibility: public

Access: Read

Description: This property provides access to the resource raw data in a stream-like way.

When a resource has been read from a stream, RawData redirects read operations to the original stream. When RawData is written to, a copy-on-write mechanism copies data from the original stream to a memory stream.

The copy-on-write behaviour can be controlled via CacheData (144) property.

Note that for some predefined resource types there are better ways to read and write resource data: some resource types use specific formats, so RawData might not always be what one expected. E.g. in a resource of type RT_BITMAP (131), RawData doesn't contain a valid BMP file: in this case it's better to use BitmapData (129) stream of TBitmapResource (129) class to work with a BMP-like stream.

Remark When writing to a "specialized" stream in a descendant class (like the TBitmapResource.BitmapData (129) stream mentioned earlier), RawData contents might not be valid anymore. If you need to access RawData again, be sure to call UpdateRawData (140) method first.

Usually there isn't much penalty in using specialized streams in descendant classes, since data isn't duplicated in two or more streams, whenever possible. So, having a very large bitmap resource and reading/writing it via TBitmapResource.BitmapData (129) doesn't mean the bitmap is allocated two times.

See also: TAbstractResource.CacheData (144), TAbstractResource.UpdateRawData (140), TAbstractResource.SetCustomRawDataStrea (140)

21.17.29 TAbstractResource.CacheData

Synopsis: Controls the copy-on-write behaviour of the resource

Declaration: `Property CacheData : Boolean`

Visibility: public

Access: Read,Write

Description: When CacheData is true, copy-on-write mechanism of RawData (144) is enabled.

Setting CacheData to false forces the raw resource data to be loaded in memory without performing any caching.

By default, CacheData is true.

See also: TAbstractResource.RawData (144), TResources.CacheData (163)

21.17.30 TAbstractResource.OwnerList

Synopsis: The resource list that owns this resource

Declaration: `Property OwnerList : TResources`

Visibility: public

Access: Read

Description: This property identifies the TResources (156) object to which this resource belongs to.

This property can be `nil` if the resource isn't part of a resource list.

21.17.31 TAbstractResource.Owner

Synopsis: The owner of this resource

Declaration: `Property Owner : TAbstractResource`

Visibility: public

Access: Read

Description: This property is meaningful only for those sub-resources that are part of a larger *group resource*.

Example: an icon is made by a RT_GROUP_ICON (131) resource and many RT_ICON (132) resources that hold single-image data. Each RT_ICON (132) resource has a pointer to the RT_GROUP_ICON (131) resource in its Owner property.

21.18 TAbstractResourceReader

21.18.1 Description

This is the base class that represents a resource reader.

A resource reader is an object whose job is to populate a TResources (156) object with resources read from a stream in a specific format.

Typically, a TResources (156) object invokes CheckMagic (149) method of the resource reader when it's searching for a reader able to read a certain stream, and Load (148) method when it wants the reader to read data from the stream.

Usually each resource reader registers itself with TResources (156) class in the `initialization` section of the unit in which it is implemented: this way a TResources (156) object can find it when probing a stream that is to be read.

Remark An object of this class should never be directly instantiated: use a descendant class instead.

See also: TResources (156), TAbstractResource (135), TAbstractResourceWriter (150), TResResourceReader (129), TCoffResourceReader (129), TWinPEResourceReader (129), TElfResourceReader (129), TExternalResourceReader (129), TDfmResourceReader (129)

21.18.2 Method overview

Page	Method	Description
147	AddNoTree	Adds a resource without updating the internal tree
147	CallSubReaderLoad	Calls another reader's Load (148) method
149	CheckMagic	Checks whether a stream is in a format the reader recognizes
149	Create	Creates a new reader object
148	GetDescription	Returns the description of the reader
148	GetExtensions	Returns the extensions the reader is registered for
148	GetTree	Gets the internal resource tree of a TResources object
148	Load	Loads resources from a stream
147	SetDataOffset	Protected method to let a reader set a resource DataOffset (143) property
146	SetDataSize	Protected method to let a reader set a resource DataSize (141) property
146	SetHeaderSize	Protected method to let a reader set a resource HeaderSize (142) property
147	SetRawData	Protected method to let a reader set a resource RawData (144) property

21.18.3 Property overview

Page	Properties	Access	Description
150	Description	r	The reader description
149	Extensions	r	The extensions of file types the reader is able to read

21.18.4 TAbstractResourceReader.SetDataSize

Synopsis: Protected method to let a reader set a resource DataSize (141) property

Declaration: `procedure SetDataSize(aResource: TAbstractResource; aValue: LongWord)`

Visibility: protected

Description: This method allows a descendant class to set DataSize (141) property of a resource that is being loaded.

See also: TAbstractResourceReader.SetHeaderSize (146), TAbstractResourceReader.SetDataOffset (147), TAbstractResourceReader.SetRawData (147)

21.18.5 TAbstractResourceReader.SetHeaderSize

Synopsis: Protected method to let a reader set a resource HeaderSize (142) property

Declaration: `procedure SetHeaderSize(aResource: TAbstractResource; aValue: LongWord)`

Visibility: protected

Description: This method allows a descendant class to set HeaderSize (142) property of a resource that is being loaded.

See also: TAbstractResourceReader.SetDataSize (146), TAbstractResourceReader.SetDataOffset (147), TAbstractResourceReader.SetRawData (147)

21.18.6 TAbstractResourceReader.SetDataOffset

Synopsis: Protected method to let a reader set a resource DataOffset (143) property

Declaration:
```
procedure SetDataOffset(aResource: TAbstractResource; aValue: LongWord)
```

Visibility: protected

Description: This method allows a descendant class to set DataOffset (143) property of a resource that is being loaded.

See also: TAbstractResourceReader.SetDataSize (146), TAbstractResourceReader.SetHeaderSize (146), TAbstractResourceReader.SetRawData (147)

21.18.7 TAbstractResourceReader.SetRawData

Synopsis: Protected method to let a reader set a resource RawData (144) property

Declaration:
```
procedure SetRawData(aResource: TAbstractResource; aStream: TStream)
```

Visibility: protected

Description: This method allows a descendant class to set RawData (144) property of a resource that is being loaded.

See also: TAbstractResourceReader.SetDataSize (146), TAbstractResourceReader.SetHeaderSize (146), TAbstractResourceReader.SetDataOffset (147)

21.18.8 TAbstractResourceReader.CallSubReaderLoad

Synopsis: Calls another reader's Load (148) method

Declaration:
```
procedure CallSubReaderLoad(aReader: TAbstractResourceReader;
                            aResources: TResources; aStream: TStream)
```

Visibility: protected

Description: This method allows a descendant class to call another reader's Load (148) method. This can be useful when a reader needs to use another one.

21.18.9 TAbstractResourceReader.AddNoTree

Synopsis: Adds a resource without updating the internal tree

Declaration:
```
procedure AddNoTree(aResources: TResources;
                    aResource: TAbstractResource)
```

Visibility: protected

Description: This protected method is used by descendents of TAbstractResourceReader (145) after they add new resources to the internal resource tree used by a TResources (156) object. Calling this method notifies the TResources (156) object about the newly-added resource.

See also: TAbstractResourceReader.GetTree (148), TRootResTreeNode (171)

21.18.10 TAbstractResourceReader.GetTree

Synopsis: Gets the internal resource tree of a TResources object

Declaration: `function GetTree(aResources: TResources) : TObject`

Visibility: protected

Description: This protected method can be used by descendents of TAbstractResourceReader (145) to obtain the internal resource tree used by a TResources (156) object. The internal resource tree is an instance of TRootResTreeNode (171).

Some resource readers can improve their performance if, instead of calling TResources.Add (157), add themselves resources to the internal tree used by a TResources (156) object.

Remark After adding a resource to a resource tree, a reader must always call AddNoTree (147) method to let the TResources (156) object know about the newly-added resource.

See also: TAbstractResourceReader.AddNoTree (147), TRootResTreeNode (171)

21.18.11 TAbstractResourceReader.GetExtensions

Synopsis: Returns the extensions the reader is registered for

Declaration: `function GetExtensions : string; Virtual; Abstract`

Visibility: protected

Description: Descendant classes must implement this method to provide access to Extensions (149) property.

See also: TAbstractResourceReader.Extensions (149)

21.18.12 TAbstractResourceReader.GetDescription

Synopsis: Returns the description of the reader

Declaration: `function GetDescription : string; Virtual; Abstract`

Visibility: protected

Description: Descendant classes must implement this method to provide access to Description (150) property.

See also: TAbstractResourceReader.Description (150)

21.18.13 TAbstractResourceReader.Load

Synopsis: Loads resources from a stream

Declaration: `procedure Load(aResources: TResources; aStream: TStream); Virtual ; Abstract`

Visibility: protected

Description: A TResources (156) object invokes this method when it needs to be loaded from a stream, passing itself as the `aResources` parameter and the stream as the `aStream` parameter.

`aStream` position is already correctly set: the reader must start to read from there.

Descendant classes must ensure that the the stream is in a format they recognize, otherwise an EResourceReaderWrongFormatException (135) exception must be raised.

Each resource is then created, read from the stream and added to the TResources (156) object.

When reading a resource, a reader must:

- Create the resource via TResourceFactory.CreateResource (129) class method with the correct type and name.
- Set at least the following resource properties:
  - DataSize (141), via SetDataSize (146) method.
  - DataOffset (143), via SetDataSize (147) method. This is the offset of the resource data from the beginning of the stream.
  - RawData (144). The reader must create a TResourceDataStream (129) object and assign it to the resource via SetRawData (147) method.

Errors: If the stream is in a format not recognized by the reader, a EResourceReaderWrongFormatException (135) exception must be raised.

If the stream ends prematurely, a EResourceReaderUnexpectedEndOfStreamException (135) exception must be raised.

See also: TResources (156), TResources.LoadFromStream (160), TResources.LoadFromFile (160), TAbstractResource (135), TAbstractResource.DataSize (141), TAbstractResource.DataOffset (143), TAbstractResource.RawData (144), TAbstractResourceReader.SetDataSize (146), TAbstractResourceReader.SetDataOffset (147), TAbstractResourceReader.SetRawData (147), TAbstractResourceReader.CheckMagic (149), TResourceDataStream (129)

21.18.14 TAbstractResourceReader.CheckMagic

Synopsis: Checks whether a stream is in a format the reader recognizes

Declaration: `function CheckMagic(aStream: TStream) : Boolean; Virtual; Abstract`

Visibility: protected

Description: A TResources (156) object invokes this method when it is searching for a reader able to read a stream, passing that stream as the `aStream` parameter.

`aStream` position is already correctly set: the reader must start to read from there.

This method should read the minimum amount of information needed to recognize the contents of a stream as a valid format: it usually means reading a magic number or a file header.

See also: TAbstractResourceReader.Load (148), TResources.FindReader (159), TResources.LoadFromStream (160), TResources.LoadFromFile (160)

21.18.15 TAbstractResourceReader.Create

Synopsis: Creates a new reader object

Declaration: `constructor Create; Virtual; Abstract`

Visibility: public

21.18.16 TAbstractResourceReader.Extensions

Synopsis: The extensions of file types the reader is able to read

Declaration: `Property Extensions : string`

Visibility: public

Access: Read

Description: This property is a string made of space-separated file extensions (each including the leading dot), all lowercase.

This property signals which file types the reader is able to read.

21.18.17 TAbstractResourceReader.Description

Synopsis: The reader description

Declaration: `Property Description : string`

Visibility: public

Access: Read

Description: This property provides a textual description of the reader, e.g. "FOO resource reader"

21.19 TAbstractResourceWriter

21.19.1 Description

This is the base class that represents a resource writer.

A resource writer is an object whose job is to write all resources contained in a TResources (156) object to a stream in a specific format.

Typically, a TResources (156) object invokes Write (151) method of the resource writer when it wants the writer to write data to a stream.

Usually each resource writer registers itself with TResources (156) class in the `initialization` section of the unit in which it is implemented: this way a TResources (156) object can find it when it is asked to write itself to a file, and no writer was specified (the writer is found based on the extension of the file to write to).

Remark An object of this class should never be directly instantiated: use a descendant class instead.

See also: TResources (156), TAbstractResource (135), TAbstractResourceReader (145), TResResourceWriter (129), TCoffResourceWriter (129), TElfResourceWriter (129), TExternalResourceWriter (129)

21.19.2 Method overview

Page	Method	Description
152	Create	Creates a new writer object
151	GetDescription	Returns the description of the writer
151	GetExtensions	Returns the extensions the writer is registered for
151	GetTree	Gets the internal resource tree of a TResources object
151	Write	Writes resources to a stream

21.19.3 Property overview

Page	Properties	Access	Description
152	Description	r	The writer description
152	Extensions	r	The extensions of file types the writer is able to write

21.19.4 TAbstractResourceWriter.GetTree

Synopsis: Gets the internal resource tree of a TResources object

Declaration:
```
function GetTree(aResources: TResources) : TObject
```

Visibility: protected

Description: This protected method can be used by descendents of TAbstractResourceWriter (150) to obtain the internal resource tree used by a TResources (156) object. The internal resource tree is an instance of TRootResTreeNode (171).

Some resource writers need to order resources with a tree structure before writing them to a file. Instead of doing this work themselves, they can use the already-ordered internal resource tree of the TResources (156) object they must write.

See also: TRootResTreeNode (171)

21.19.5 TAbstractResourceWriter.GetExtensions

Synopsis: Returns the extensions the writer is registered for

Declaration:
```
function GetExtensions : string;  Virtual;  Abstract
```

Visibility: protected

Description: Descendant classes must implement this method to provide access to Extensions (152) property.

See also: TAbstractResourceWriter.Extensions (152)

21.19.6 TAbstractResourceWriter.GetDescription

Synopsis: Returns the description of the writer

Declaration:
```
function GetDescription : string;  Virtual;  Abstract
```

Visibility: protected

Description: Descendant classes must implement this method to provide access to Description (152) property.

See also: TAbstractResourceWriter.Description (152)

21.19.7 TAbstractResourceWriter.Write

Synopsis: Writes resources to a stream

Declaration:
```
procedure Write(aResources: TResources; aStream: TStream);  Virtual
                    ;  Abstract
```

Visibility: protected

Description: A TResources (156) object invokes this method when it needs to be written to a stream, passing itself as the `aResources` parameter and the stream as the `aStream` parameter.

`aStream` position is already correctly set: the writer must start to write from there.

A writer must write data in the way specified by the format it supports: usually this means writing a header and all resources contained in the TResources (156) object.

For each resource, a writer should write some information about the resource (like type and name) in a way defined by the format it implements, and the resource data, which is accessible by RawData (144) property of the resource.

See also: TResources (156), TResources.WriteToStream (162), TResources.WriteToFile (162), TAbstractResource (135), TAbstractResource.DataSize (141), TAbstractResource.RawData (144)

21.19.8 TAbstractResourceWriter.Create

Synopsis: Creates a new writer object

Declaration: `constructor Create; Virtual; Abstract`

Visibility: public

21.19.9 TAbstractResourceWriter.Extensions

Synopsis: The extensions of file types the writer is able to write

Declaration: `Property Extensions : string`

Visibility: public

Access: Read

Description: This property is a string made of space-separated file extensions (each including the leading dot), all lowercase.

This property signals which file types the writer is able to write.

21.19.10 TAbstractResourceWriter.Description

Synopsis: The writer description

Declaration: `Property Description : string`

Visibility: public

Access: Read

Description: This property provides a textual description of the writer, e.g. "FOO resource writer"

21.20 TGenericResource

21.20.1 Description

This class represents a generic resource.

It is suitable to use in all situations where a higher level class is not needed (e.g. the resource raw data is made of arbitrary data) or when total low-level control is desirable.

This class is also the default one that is used by TResourceFactory (129) when it finds no class matching the given type.

See also: TResourceFactory.CreateResource (129)

21.20.2 Method overview

Page	Method	Description
153	ChangeDescTypeAllowed	
153	ChangeDescValueAllowed	
153	Create	
154	Destroy	
153	GetName	
153	GetType	
153	NotifyResourcesLoaded	
154	UpdateRawData	

21.20.3 TGenericResource.GetType

Declaration:
```
function GetType : TResourceDesc;  Override
```

Visibility: protected

21.20.4 TGenericResource.GetName

Declaration:
```
function GetName : TResourceDesc;  Override
```

Visibility: protected

21.20.5 TGenericResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                ;  Override
```

Visibility: protected

21.20.6 TGenericResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                 ;  Override
```

Visibility: protected

21.20.7 TGenericResource.NotifyResourcesLoaded

Declaration:
```
procedure NotifyResourcesLoaded;  Override
```

Visibility: protected

21.20.8 TGenericResource.Create

Declaration:
```
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                    ;  Override
```

Visibility: public

21.20.9 TGenericResource.Destroy

Declaration: `destructor Destroy; Override`

Visibility: public

21.20.10 TGenericResource.UpdateRawData

Declaration: `procedure UpdateRawData; Override`

Visibility: public

21.21 TResourceDesc

21.21.1 Description

This class represent a resource description (type or name).

Resources are identified by a type, name and (optionally) a language ID.

Type and name can be either a **name** (a string identifier) or an **ID** (an integer identifier in the range 0..65535).

Remark Unfortunately, *name* is used both to refer to a the name of the resource and both to the format of a resource description

Example:

Typically, a Windows executable has a main icon, which is a resource of type RT_GROUP_ICON (131) (type is an ID) and name `MAINICON` (name is a name).

See also: TAbstractResource (135)

21.21.2 Method overview

Page	Method	Description
155	Assign	Assigns the contents of another TResourceDesc object to this one
155	Create	Creates a new TResourceDesc object
155	Equals	Compares the contents of another TResourceDesc object to this one
154	SetOwner	Protected method to let a resource set itself as owner of the TResourceDesc

21.21.3 Property overview

Page	Properties	Access	Description
156	DescType	r	The type of the resource description
156	ID	rw	The value of the resource description as an ID
155	Name	rw	The value of the resource description as a name

21.21.4 TResourceDesc.SetOwner

Synopsis: Protected method to let a resource set itself as owner of the TResourceDesc

Declaration: `procedure SetOwner(aOwner: TAbstractResource)`

Visibility: protected

21.21.5 TResourceDesc.Create

Synopsis: Creates a new TResourceDesc object

Declaration:
```
constructor Create;  Overload
constructor Create(const aID: TResID);  Overload
constructor Create(const aName: TResName);  Overload
```

Visibility: public

Description: Creates a new TResourceDesc object.

Without arguments, resource description is of type dtName (132) and its name is empty.

If an argument is specified, resource description and name/id are set accordingly to the value passed as parameter.

21.21.6 TResourceDesc.Assign

Synopsis: Assigns the contents of another TResourceDesc object to this one

Declaration:
```
procedure Assign(aResourceDesc: TResourceDesc)
```

Visibility: public

Description: Assigns the contents of another TResourceDesc object to this one

Errors: If changing values is not permitted because owner resource doesn't allow it (e.g. because owner resource is a TBitmapResource (129) and values other than RT_BITMAP (131) are not permitted for the resource type) an EResourceDescChangeNotAllowedException (133) exception is raised.

21.21.7 TResourceDesc.Equals

Synopsis: Compares the contents of another TResourceDesc object to this one

Declaration:
```
function Equals(aResDesc: TResourceDesc) : Boolean
```

Visibility: public

21.21.8 TResourceDesc.Name

Synopsis: The value of the resource description as a name

Declaration:
```
Property Name : TResName
```

Visibility: public

Access: Read,Write

Description: Setting this property causes DescType (156) to be dtName (132)

When reading, if DescType (156) is dtID (132), the ID is returned as a string value.

See also: TResourceDesc.ID (156), TResourceDesc.DescType (156)

21.21.9 TResourceDesc.ID

Synopsis: The value of the resource description as an ID

Declaration: `Property ID : TResID`

Visibility: public

Access: Read,Write

Description: Setting this property causes DescType (156) to be dtID (132)

Remark When reading, if DescType (156) is dtName (132), an EResourceDescTypeException (133) exception is raised.

See also: TResourceDesc.Name (155), TResourceDesc.DescType (156)

21.21.10 TResourceDesc.DescType

Synopsis: The type of the resource description

Declaration: `Property DescType : TDescType`

Visibility: public

Access: Read

Description: When DescType is dtName (132), resource description value is a name and can be accessed via Name (155) property

When DescType is dtID (132), resource description value is an ID and can be accessed via ID (156) property

See also: TResourceDesc.Name (155), TResourceDesc.ID (156), TDescType (132)

21.22 TResources

21.22.1 Description

This class represents a format-independent view of a resource file. It holds a collection of resources (TAbstractResource (135) descendants).

Typically, a TResource object is loaded from and written to a stream via format-dependent readers and writers, which are descendants of TAbstractResourceReader (145) and TAbstractResourceWriter (150), respectively.

Single resources can be added, removed, searched and modified: a resource compiler or editor probably will create resources, set their data, and add them to a TResources object which can then be written to file in the desired format.

This class also provides some class methods to register and find resource readers and writers: these classes, once registered, will be used by a TResources object when it is asked to load or save itself to a stream and the user didn't specify a reader or writer.

Remark Because of the copy-on-write mechanism of TAbstractResource (135), care should be taken when loading resources, since by default resource data isn't immediately read from the stream: resources hold a reference to the original stream because they need it when their data is requested. For further information, see TResources.LoadFromStream (160) and TResources.LoadFromFile (160).

See also: TAbstractResource (135), TAbstractResourceReader (145), TAbstractResourceWriter (150)

21.22.2 Method overview

Page	Method	Description
157	Add	Adds a resource
158	AddAutoID	Adds a resource and gives it a new autogenerated name
158	Clear	Deletes all resources
157	Create	Creates a new TResources object
157	Destroy	Destroys the object
158	Find	Searches for a resource
159	FindReader	Searches for a suitable resource reader
160	LoadFromFile	Loads the contents of the object from a file
160	LoadFromStream	Loads the contents of the object from a stream
159	MoveFrom	Moves all resources of another TResources object to itself
161	RegisterReader	Registers a resource reader class
161	RegisterWriter	Registers a resource writer class
159	Remove	Removes a resource
162	WriteToFile	Writes the contents of the object to a file
162	WriteToStream	Writes the contents of the object to a stream

21.22.3 Property overview

Page	Properties	Access	Description
163	CacheData	rw	Controls the copy-on-write behaviour of all resources
162	Count	r	The number of resources in the object
163	Items	r	Indexed array of resources in the object

21.22.4 TResources.Create

Synopsis: Creates a new TResources object

Declaration: `constructor Create`

Visibility: public

21.22.5 TResources.Destroy

Synopsis: Destroys the object

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: All resources are destroyed as well.

Remark:

21.22.6 TResources.Add

Synopsis: Adds a resource

Declaration: `procedure Add(aResource: TAbstractResource)`

Visibility: public

Description: This method adds `aResource` to the object, and sets itself as the owner list of the resource.

Errors: If a resource with the same type, name and language ID already exists, an EResourceDuplicateException (134) exception is raised.

See also: TResources.AddAutoID (158)

21.22.7 TResources.AddAutoID

Synopsis: Adds a resource and gives it a new autogenerated name

Declaration:
```
function AddAutoID(aResource: TAbstractResource) : TResID
```

Visibility: public

Description: This method tries to find a spare ID to use as a name for the given resource, assigns it to the resource, and adds it.

This method is useful when the name of the resource is not important. E.g. a group resource doesn't care about the names of its sub-resources, so it could use this method to ensure that its sub-resources don't clash with names of other sub-resources of the same type.

Errors: If there are no more free IDs for the resource type of the given resource (that is, when the number of resources of the same type of `aResource` with an ID name is equal to 65536) an ENoMoreFreeIDsException (133) exception is raised.

See also: TResources.Add (157)

21.22.8 TResources.Clear

Synopsis: Deletes all resources

Declaration:
```
procedure Clear
```

Visibility: public

Description: This method empties the TResources object destroying all resources.

21.22.9 TResources.Find

Synopsis: Searches for a resource

Declaration:
```
function Find(aType: TResourceDesc; aName: TResourceDesc;
              const aLangID: TLangID) : TAbstractResource;  Overload
function Find(aType: TResourceDesc; aName: TResourceDesc)
              : TAbstractResource;  Overload
function Find(const aType: TResName; const aName: TResName;
              const aLangID: TLangID) : TAbstractResource;  Overload
function Find(const aType: TResName; const aName: TResID;
              const aLangID: TLangID) : TAbstractResource;  Overload
function Find(const aType: TResID; const aName: TResName;
              const aLangID: TLangID) : TAbstractResource;  Overload
function Find(const aType: TResID; const aName: TResID;
              const aLangID: TLangID) : TAbstractResource;  Overload
function Find(const aType: TResName; const aName: TResName)
              : TAbstractResource;  Overload
function Find(const aType: TResName; const aName: TResID)
              : TAbstractResource;  Overload
function Find(const aType: TResID; const aName: TResName)
              : TAbstractResource;  Overload
function Find(const aType: TResID; const aName: TResID)
              : TAbstractResource;  Overload
```

Visibility: public

Description: This method searches for a resource with the given type and name. If a language ID is not provided, the first resource found that matches `aType` and `aName` is returned.

Errors: If the resource is not found, an EResourceNotFoundException (134) exception is raised.

21.22.10 TResources.FindReader

Synopsis: Searches for a suitable resource reader

Declaration:
```
class function FindReader(aStream: TStream; aExtension: string)
                                  : TAbstractResourceReader
class function FindReader(aStream: TStream) : TAbstractResourceReader
```

Visibility: public

Description: This method tries to find a resource reader able to read the stream passed as parameter.

If an extension is specified, only readers matching that extension are tried. The extension is case-insensitive and includes the leading dot, unless the empty string is passed (which means "no extension", e.g. a unix executable, which doesn't have an extension).

If a suitable reader is found, an instance of that reader is returned.

Remark To make a resource reader class known, add that resource reader unit to the uses clause of your program.

Errors: If no suitable reader is found, an EResourceReaderNotFoundException (134) exception is raised.

See also: TAbstractResourceReader (145)

21.22.11 TResources.MoveFrom

Synopsis: Moves all resources of another TResources object to itself

Declaration:
```
procedure MoveFrom(aResources: TResources)
```

Visibility: public

Description: This method takes all resources from object `aResources` and adds them to this object. `aResources` object is left empty.

Errors: If a resource with the same type, name and language ID already exists, an EResourceDuplicateException (134) exception is raised.

See also: TResources.Add (157)

21.22.12 TResources.Remove

Synopsis: Removes a resource

Declaration:
```
function Remove(aType: TResourceDesc; aName: TResourceDesc;
                const aLangID: TLangID) : TAbstractResource;  Overload
function Remove(aType: TResourceDesc; aName: TResourceDesc)
                 : TAbstractResource;  Overload
function Remove(aResource: TAbstractResource) : TAbstractResource
               ;  Overload
function Remove(aIndex: Integer) : TAbstractResource;  Overload
```

Visibility: public

Description: This method searches for a resource based on passed parameters and removes it from the object.

The removed resource is then returned.

Errors: If no matching resource is found, an EResourceNotFoundException (134) exception is raised.

See also: TResources.Find (158)

21.22.13 TResources.LoadFromStream

Synopsis: Loads the contents of the object from a stream

Declaration:
```
procedure LoadFromStream(aStream: TStream);  Overload
procedure LoadFromStream(aStream: TStream;
                         aReader: TAbstractResourceReader);  Overload
```

Visibility: public

Description: This method clears the TResources object and loads its contents from the stream passed as parameter.

If a reader is specified, that reader is used. Otherwise, the stream is probed to find a suitable reader.

Remark If CacheData (163) is set to `true`, the stream will be used as the underlying stream of each resource RawData (144) stream. This means that the stream must not be freed until all resources in the TResources object are freed: this happens when the TResources object is cleared or is loaded again from a different source. If you need to free the stream while there are still resources, disable the copy-on-write mechanism by setting CacheData (163) property to `false`.

Errors: If no reader is passed and probing fails, an EResourceReaderNotFoundException (134) exception is raised.

See also: TAbstractResourceReader (145), TAbstractResource.RawData (144), TAbstractResource.CacheData (144), TResources.CacheData (163), TResources.LoadFromFile (160), TResources.Clear (158), TResources.FindReader (159)

21.22.14 TResources.LoadFromFile

Synopsis: Loads the contents of the object from a file

Declaration:
```
procedure LoadFromFile(aFileName: string);  Overload
procedure LoadFromFile(aFileName: string;
                       aReader: TAbstractResourceReader);  Overload
```

Visibility: public

Description: This method clears the TResources object and loads its contents from the file passed as parameter.

If a reader is specified, that reader is used. Otherwise, the file is probed to find a suitable reader.

Remark If CacheData (163) is set to `true`, the file will be left open and used as the underlying stream of each resource RawData (144) stream. This means that the file will be open until the TResources object is cleared or is loaded again from a different source. If you want the file to be closed while there are still resources, disable the copy-on-write mechanism by setting CacheData (163) property to `false`.

Sample code

This code extracts resources from an exe file

```
var
  resources : TResources;
begin
  resources:=TResources.Create;
  resources.LoadFromFile('myexe.exe');
  resources.WriteToFile('myexe.res');
  resources.Free;
end;
```

Errors: If no reader is passed and probing fails, an EResourceReaderNotFoundException (134) exception is raised.

See also: TAbstractResourceReader (145), TAbstractResource.RawData (144), TAbstractResource.CacheData (144), TResources.CacheData (163), TResources.LoadFromStream (160), TResources.Clear (158), TResources.FindReader (159)

21.22.15 TResources.RegisterReader

Synopsis: Registers a resource reader class

Declaration:
```
class procedure RegisterReader(const aExtension: string;
                               aClass: TResourceReaderClass)
```

Visibility: public

Description: This class method registers a resource reader class.

When registered, a class is known to TResources class, and can be used by FindReader (159), LoadFromStream (160) and LoadFromFile (160) methods when probing.

Usually this method is called in the `initialization` section of the unit implementing a TAbstractResourceReader (145) descendant.

A class can be registered multiple times, one for each extension it is able to read. Multiple class can be registered for the same extension (e.g. both a coff and a elf reader can be registered for the .o extension). The extension must include the leading dot unless the empty string is used (which means "no extension", e.g. a unix executable, which doesn't have an extension).

See also: TAbstractResourceReader (145), TResources.FindReader (159), TResources.LoadFromStream (160), TResources.LoadFromFile (160)

21.22.16 TResources.RegisterWriter

Synopsis: Registers a resource writer class

Declaration:
```
class procedure RegisterWriter(const aExtension: string;
                               aClass: TResourceWriterClass)
```

Visibility: public

Description: This class method registers a resource writer class.

When registered, a class is known to TResources class, and can be used by WriteToFile (162) method when probing.

Usually this method is called in the `initialization` section of the unit implementing a TAbstractResourceWriter (150) descendant.

A class can be registered multiple times, one for each extension it is able to write. Multiple class can be registered for the same extension (e.g. both a coff and a elf writer can be registered for the .o

extension) although only the first one found will be used when probing. The extension must include the leading dot unless the empty string is used (which means "no extension", e.g. a unix executable, which doesn't have an extension).

See also: TAbstractResourceWriter (150), TResources.WriteToFile (162)

21.22.17 TResources.WriteToStream

Synopsis: Writes the contents of the object to a stream

Declaration:
```
procedure WriteToStream(aStream: TStream;
                        aWriter: TAbstractResourceWriter)
```

Visibility: public

Description: This method writes the contents of the object to a stream via the specified TAbstractResourceWriter (150) descendant

See also: TAbstractResourceWriter (150), TResources.WriteToFile (162)

21.22.18 TResources.WriteToFile

Synopsis: Writes the contents of the object to a file

Declaration:
```
procedure WriteToFile(aFileName: string);  Overload
procedure WriteToFile(aFileName: string;
                      aWriter: TAbstractResourceWriter);  Overload
```

Visibility: public

Description: This method writes the contents of the object to a file whose name is passed as parameter.

If a writer is specified, that writer is used. Otherwise, the first registered writer matching the file name extension is used.

Errors: If no writer is passed and no suitable writer is found, an EResourceWriterNotFoundException (135) exception is raised.

See also: TAbstractResourceWriter (150), TResources.WriteToStream (162)

21.22.19 TResources.Count

Synopsis: The number of resources in the object

Declaration:
```
Property Count : LongWord
```

Visibility: public

Access: Read

See also: TResources.Items (163)

21.22.20 TResources.Items

Synopsis: Indexed array of resources in the object

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

Visibility: public

Access: Read

Description: This property can be used to access all resources in the object.

Remark This array is 0-based: valid elements range from 0 to Count (162)-1.

See also: TResources.Count (162), TAbstractResource (135)

21.22.21 TResources.CacheData

Synopsis: Controls the copy-on-write behaviour of all resources

Declaration: `Property CacheData : Boolean`

Visibility: public

Access: Read,Write

Description: Changing this property changes CacheData (144) property of all resources contained in the object.

This property affects existing resources and resources that are added or loaded.

By default, CacheData is true.

See also: TAbstractResource.CacheData (144), TAbstractResource.RawData (144)

Chapter 22

Reference for unit 'resourcetree'

22.1 Used units

Table 22.1: Used units by unit 'resourcetree'

Name	Page
Classes	??
resource	129
sysutils	??

22.2 Overview

This unit implements classes that represent an ordered tree of resources.

Such a tree is used internally by TResources (164) to speed up operations, and by certain resource readers and writers that deal with resource formats where data is stored as ordered trees of resources. For this reason, only implementors of resource readers and writers should be interested in these classes.

A tree begins with a root node, which is an instance of TRootResTreeNode (171). The root node contains type nodes, that contain name nodes, that contain language id nodes. Finally, a language id node contains a resource.

Each node contains its sub nodes in two lists: a Named list for nodes identified by a name (a string), and an ID list for nodes identified by an ID (an integer). In each list, nodes are sorted in ascending order.

Many resource formats (PECOFF, ELF, Mach-O, and external resource files) use this exact format to store resource information.

Remark When a tree is destroyed, the resources it references aren't destroyed.

22.3 TResourceTreeNode

22.3.1 Description

This class represents a node in a resource tree.

Remark An object of this class should never be directly instantiated. To create a node, call CreateSubNode (167) method of an already existing node.

22.3.2 Method overview

Page	Method	Description
166	Add	Adds a new resource to the tree
167	Clear	Destroys all sub nodes
166	Create	
167	CreateResource	Creates a new resource
167	CreateSubNode	Creates a subnode
166	Destroy	Destroys the object.
168	Find	Searches for a resource
168	FindFreeID	Find a free ID to be used as a resource name
166	GetData	
165	GetIDCount	
166	GetIDEntry	
165	GetNamedCount	
165	GetNamedEntry	
166	InternalFind	
168	IsLeaf	Reports whether the node is a leaf node.
167	Remove	Removes a resource from the tree

22.3.3 Property overview

Page	Properties	Access	Description
171	Data	r	The resource contained in this node
171	DataRVA	rw	To be used by readers and writers
169	Desc	r	The description of the node
169	IDCount	r	The number of ID sub nodes of the node
170	IDEntries	r	Indexed array of ID sub nodes of the node
169	NamedCount	r	The number of named sub nodes of the node
169	NamedEntries	r	Indexed array of named sub nodes of the node
170	NameRVA	rw	To be used by readers and writers
168	Parent	r	
170	SubDirRVA	rw	To be used by readers and writers

22.3.4 TResourceTreeNode.GetNamedCount

Declaration: `function GetNamedCount : LongWord`

Visibility: protected

22.3.5 TResourceTreeNode.GetNamedEntry

Declaration: `function GetNamedEntry(index: Integer) : TResourceTreeNode`

Visibility: protected

22.3.6 TResourceTreeNode.GetIDCount

Declaration: `function GetIDCount : LongWord`

Visibility: protected

22.3.7 TResourceTreeNode.GetIDEntry

Declaration:
```
function GetIDEntry(index: Integer) : TResourceTreeNode
```

Visibility: protected

22.3.8 TResourceTreeNode.GetData

Declaration:
```
function GetData : TAbstractResource;  Virtual
```

Visibility: protected

22.3.9 TResourceTreeNode.InternalFind

Declaration:
```
function InternalFind(aList: TFPList; aDesc: TResourceDesc;
                      out index: Integer) : Boolean;  Overload
function InternalFind(aList: TFPList; aLangID: TLangID;
                      out index: Integer) : Boolean;  Overload
function InternalFind(aType: TResourceDesc; aName: TResourceDesc;
                      const aLangID: TLangID; const noLangID: Boolean;
                      const toDelete: Boolean) : TAbstractResource
                      ;  Virtual;  Abstract;  Overload
```

Visibility: protected

22.3.10 TResourceTreeNode.Create

Declaration:
```
constructor Create;  Virtual;  Overload
```

Visibility: protected

22.3.11 TResourceTreeNode.Destroy

Synopsis: Destroys the object.

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

Visibility: public

Description

Remark Only root nodes (instances of TRootResTreeNode (171)) should be destroyed. Children nodes are destroyed by their parent when needed.

22.3.12 TResourceTreeNode.Add

Synopsis: Adds a new resource to the tree

Declaration:
```
procedure Add(aResource: TAbstractResource);  Virtual;  Abstract
```

Visibility: public

Description: This method adds a new resource to the tree, creating all needed sub nodes

Remark This method should only be called on root nodes (instances of TRootResTreeNode (171)).

Errors: If a resource with the same type, name and language ID already exists, an EResourceDuplicateException (164) exception is raised.

See also: TResourceTreeNode.Remove (167)

22.3.13 TResourceTreeNode.CreateSubNode

Synopsis: Creates a subnode

Declaration:
```
function CreateSubNode(aDesc: TResourceDesc) : TResourceTreeNode
                              ;  Virtual;  Abstract
```

Visibility: public

Description: This method creates a subnode, identified by the given resource description.

See also: TResourceTreeNode.Desc (169)

22.3.14 TResourceTreeNode.CreateResource

Synopsis: Creates a new resource

Declaration:
```
function CreateResource : TAbstractResource;  Virtual
```

Visibility: public

Description: This method creates a new resource.

A new resource is created and its type, name and language id are determined from the ancestor nodes in the tree hierarchy.

Usually `CreateResource` is called by resource readers that read files in which resources are stored as trees.

Remark This method is meaningful only when called on leaf nodes (language id nodes).

See also: TResourceTreeNode.IsLeaf (168)

22.3.15 TResourceTreeNode.Clear

Synopsis: Destroys all sub nodes

Declaration:
```
procedure Clear
```

Visibility: public

Description: This method destroys all sub nodes of the node.

22.3.16 TResourceTreeNode.Remove

Synopsis: Removes a resource from the tree

Declaration:
```
function Remove(aType: TResourceDesc; aName: TResourceDesc)
                    : TAbstractResource;  Overload
function Remove(aType: TResourceDesc; aName: TResourceDesc;
                const aLangID: TLangID) : TAbstractResource;  Overload
```

Visibility: public

Description: This method searches for the specified resource and removes it from the tree. If a language ID is not provided, the first resource found that matches `aType` and `aName` is returned. The removed resource is then returned.

If no resource is found, `nil` is returned.

Remark This method should only be called on root nodes (instances of TRootResTreeNode (171)).

See also: TResourceTreeNode.Add (166)

22.3.17 TResourceTreeNode.Find

Synopsis: Searches for a resource

Declaration:
```
function Find(aType: TResourceDesc; aName: TResourceDesc)
                  : TAbstractResource;  Overload
function Find(aType: TResourceDesc; aName: TResourceDesc;
              const aLangID: TLangID) : TAbstractResource;  Overload
```

Visibility: public

Description: This method searches for a resource with the given type and name in the tree. If a language ID is not provided, the first resource found that matches `aType` and `aName` is returned.

If no resource is found, `nil` is returned.

Remark This method should only be called on root nodes (instances of TRootResTreeNode (171)).

22.3.18 TResourceTreeNode.FindFreeID

Synopsis: Find a free ID to be used as a resource name

Declaration:
```
function FindFreeID(aType: TResourceDesc) : TResID;  Virtual
```

Visibility: public

Description: This method is used to find an available ID to be used as a name for a resource, given a resource type. It is used internally by AddAutoID (164) method of TResources (164).

Remark This method should only be called on root nodes (instances of TRootResTreeNode (171)).

Errors: If there are no free ids left for the given resource type, an ENoMoreFreeIDsException (164) is raised.

22.3.19 TResourceTreeNode.IsLeaf

Synopsis: Reports whether the node is a leaf node.

Declaration:
```
function IsLeaf : Boolean;  Virtual
```

Visibility: public

Description: Returns `true` if the node is a leaf node. A leaf node is a language ID node.

22.3.20 TResourceTreeNode.Parent

Declaration:
```
Property Parent : TResourceTreeNode
```

Visibility: protected

Access: Read

22.3.21 TResourceTreeNode.Desc

Synopsis: The description of the node

Declaration: `Property Desc : TResourceDesc`

Visibility: public

Access: Read

Description: The description of a node identifies that node. According to the type of the node, it can be a resource type, name or language id.

22.3.22 TResourceTreeNode.NamedCount

Synopsis: The number of named sub nodes of the node

Declaration: `Property NamedCount : LongWord`

Visibility: public

Access: Read

See also: TResourceTreeNode.NamedEntries (169), TResourceTreeNode.IDCount (169)

22.3.23 TResourceTreeNode.NamedEntries

Synopsis: Indexed array of named sub nodes of the node

Declaration: `Property NamedEntries[index: Integer]: TResourceTreeNode`

Visibility: public

Access: Read

Description: This property can be used to access all named sub nodes in the node.

Remark This array is 0-based: valid elements range from 0 to NamedCount (169)-1.

See also: TResourceTreeNode.NamedCount (169), TResourceTreeNode.IDEntries (170)

22.3.24 TResourceTreeNode.IDCount

Synopsis: The number of ID sub nodes of the node

Declaration: `Property IDCount : LongWord`

Visibility: public

Access: Read

See also: TResourceTreeNode.IDEntries (170), TResourceTreeNode.NamedCount (169)

22.3.25 TResourceTreeNode.IDEntries

Synopsis: Indexed array of ID sub nodes of the node

Declaration: `Property IDEntries[index: Integer]: TResourceTreeNode`

Visibility: public

Access: Read

Description: This property can be used to access all ID sub nodes in the node.

Remark This array is 0-based: valid elements range from 0 to IDCount (169)-1.

See also: TResourceTreeNode.IDCount (169), TResourceTreeNode.NamedEntries (169)

22.3.26 TResourceTreeNode.NameRVA

Synopsis: To be used by readers and writers

Declaration: `Property NameRVA : LongWord`

Visibility: public

Access: Read,Write

Description: This property can be freely used by resource readers and writers to store a file offset or address needed to load or write other data, since it isn't used by TResourceTreeNode (164) or TResources (164).

Remark Do not rely on the value of this property when accessing a new tree: other readers or writers could have changed it for their internal operations.

See also: TResourceTreeNode.SubDirRVA (170), TResourceTreeNode.DataRVA (171)

22.3.27 TResourceTreeNode.SubDirRVA

Synopsis: To be used by readers and writers

Declaration: `Property SubDirRVA : LongWord`

Visibility: public

Access: Read,Write

Description: This property can be freely used by resource readers and writers to store a file offset or address needed to load or write other data, since it isn't used by TResourceTreeNode (164) or TResources (164).

Remark Do not rely on the value of this property when accessing a new tree: other readers or writers could have changed it for their internal operations.

See also: TResourceTreeNode.NameRVA (170), TResourceTreeNode.DataRVA (171)

22.3.28 TResourceTreeNode.DataRVA

Synopsis: To be used by readers and writers

Declaration: `Property DataRVA : LongWord`

Visibility: public

Access: Read,Write

Description: This property can be freely used by resource readers and writers to store a file offset or address needed to load or write other data, since it isn't used by TResourceTreeNode (164) or TResources (164).

Remark Do not rely on the value of this property when accessing a new tree: other readers or writers could have changed it for their internal operations.

See also: TResourceTreeNode.NameRVA (170), TResourceTreeNode.SubDirRVA (170)

22.3.29 TResourceTreeNode.Data

Synopsis: The resource contained in this node

Declaration: `Property Data : TAbstractResource`

Visibility: public

Access: Read

Description: This property references the resource contained in this node.

Remark This property is meaningful only on leaf nodes (language id nodes).

See also: TResourceTreeNode.IsLeaf (168)

22.4 TRootResTreeNode

22.4.1 Description

This node represents the root node of a resource tree.

It is the only node which must be created with its constructor: other nodes in the tree are automatically created when adding a resource, or calling CreateSubNode (167) method of their parent.

Normally all search, add and delete operations on the tree are performed by calling methods of this node.

22.4.2 Method overview

Page	Method	Description
172	Add	
172	Create	Creates a new root node
172	CreateSubNode	
172	FindFreeID	
172	InternalFind	

22.4.3 TRootResTreeNode.InternalFind

Declaration:
```
function InternalFind(aType: TResourceDesc; aName: TResourceDesc;
                      const aLangID: TLangID; const noLangID: Boolean;
                      const toDelete: Boolean) : TAbstractResource
                      ;  Override
```

Visibility: protected

22.4.4 TRootResTreeNode.Create

Synopsis: Creates a new root node

Declaration:
```
constructor Create;  Override
```

Visibility: public

Description: This method creates a new tree, represented by a root node without children.

Other nodes in the tree are automatically created when adding a resource, or calling CreateSubNode (167) method of their parent.

See also: TResourceTreeNode.CreateSubNode (167), TResourceTreeNode.Add (166)

22.4.5 TRootResTreeNode.CreateSubNode

Declaration:
```
function CreateSubNode(aDesc: TResourceDesc) : TResourceTreeNode
                       ;  Override
```

Visibility: public

22.4.6 TRootResTreeNode.Add

Declaration:
```
procedure Add(aResource: TAbstractResource);  Override
```

Visibility: public

22.4.7 TRootResTreeNode.FindFreeID

Declaration:
```
function FindFreeID(aType: TResourceDesc) : TResID;  Override
```

Visibility: public

Chapter 23

Reference for unit 'resreader'

23.1 Used units

Table 23.1: Used units by unit 'resreader'

Name	Page
Classes	??
resource	129
sysutils	??

23.2 Overview

This unit contains TResResourceReader (173), a TAbstractResourceReader (173) descendant that is able to read .res resource files.

Adding this unit to a program's `uses` clause registers class TResResourceReader (173) with TResources (173).

23.3 TResResourceReader

23.3.1 Description

This class provides a reader for .res resource files.

A .res file is a standard Windows file type that contains resources. This is not an object file format, so it's not suitable for being linked with other object files. For this reason, it is a good choice for a platform-independent format to store resources.

See also: TAbstractResourceReader (173), TResResourceWriter (173)

23.3.2 Method overview

Page	Method	Description
174	CheckMagic	
174	Create	
174	Destroy	
174	GetDescription	
174	GetExtensions	
174	Load	

23.3.3 TResResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

23.3.4 TResResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

23.3.5 TResResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

23.3.6 TResResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

23.3.7 TResResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

23.3.8 TResResourceReader.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

Chapter 24

Reference for unit 'reswriter'

24.1 Used units

Table 24.1: Used units by unit 'reswriter'

Name	Page
Classes	??
resource	129
sysutils	??

24.2 Overview

This unit contains TResResourceWriter (175), a TAbstractResourceWriter (175) descendant that is able to write .res resource files.

Adding this unit to a program's `uses` clause registers class TResResourceWriter (175) with TResources (175).

24.3 TResResourceWriter

24.3.1 Description

This class provides a writer for .res resource files.

A .res file is a standard Windows file type that contains resources. This is not an object file format, so it's not suitable for being linked with other object files. For this reason, it is a good choice for a platform-independent format to store resources.

See also: TAbstractResourceWriter (175), TResResourceReader (175)

24.3.2 Method overview

Page	Method	Description
176	Create	
176	GetDescription	
176	GetExtensions	
176	Write	

24.3.3 TResResourceWriter.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

24.3.4 TResResourceWriter.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

24.3.5 TResResourceWriter.Write

Declaration: `procedure Write(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

24.3.6 TResResourceWriter.Create

Declaration: `constructor Create;  Override`

Visibility: public

Chapter 25

Reference for unit 'stringtableresource'

25.1 Used units

Table 25.1: Used units by unit 'stringtableresource'

Name	Page
Classes	??
resource	129
sysutils	??

25.2 Overview

This unit contains TStringTableResource (178), a TAbstractResource (177) descendant specialized in handling resource of type RT_STRING (177).

Adding this unit to a program's `uses` clause registers class TStringTableResource (178) for type RT_STRING (177) with TResourceFactory (177).

25.3 Constants, types and variables

25.3.1 Resource strings

```
SIndexOutOfBounds = 'String ID out of bounds: %d'
```

```
SNameNotAllowed =
  'Resource ID must be an ordinal in the range 1-4096'
```

25.4 EStringTableIndexOutOfBoundsException

25.4.1 Description

This exception is raised when the id specified to access a string of TStringTableResource.Strings (181) property is not in the range TStringTableResource.FirstID (180) - TStringTableResource.LastID (181).

See also: TStringTableResource (178), TStringTableResource.Strings (181), TStringTableResource.FirstID (180), TStringTableResource.LastID (181)

25.5 EStringTableNameNotAllowedException

25.5.1 Description

This exception is raised by constructor TStringTableResource.Create (180) if the name of the resource isn't of type dtID (177) and/or its name is not in the range 1-4096.

See also: TStringTableResource.Create (180)

25.6 EStringTableResourceException

25.6.1 Description

Base class for string table resource-related exceptions

25.7 TStringTableResource

25.7.1 Description

This class represents a resource of type RT_STRING (177).

A string table is a resource containing strings, identified by an integer id in the range 0-65535. A string table contains exactly 16 strings, and its name is an ID in the range 1-4096, determined by the highest 12 bits of the strings ID it contains, plus one. That is, a string table with 1 as name holds strings with IDs from 0 to 15, string table 2 contains strings with IDs from 16 to 31 and so on. There is no difference between an empty string and a non-existant string.

For these reasons, it is not possible to set the name of a string table: it is autogenerated from the value of FirstID (180) property. Moreover, Count (181) property is always 16.

Strings (181) property is provided to access and modify individual strings.

Remark This class doesn't allow its type to be changed to anything else than RT_BITMAP (177). Its name can't be changed too. Attempts to do so result in a EResourceDescChangeNotAllowedException (177).

25.7.2 Method overview

Page	Method	Description
179	ChangeDescTypeAllowed	
179	ChangeDescValueAllowed	
180	Create	Creates a new string table resource
180	Destroy	
179	GetName	
179	GetType	
179	NotifyResourcesLoaded	
180	UpdateRawData	

25.7.3 Property overview

Page	Properties	Access	Description
181	Count	r	The number of strings contained in the string table
180	FirstID	rw	The ID of first the string contained in the string table
181	LastID	r	The ID of the last string contained in the string table
181	Strings	rw	Indexed array of strings in the string table

25.7.4 TStringTableResource.GetType

Declaration: `function GetType : TResourceDesc;  Override`

Visibility: protected

25.7.5 TStringTableResource.GetName

Declaration: `function GetName : TResourceDesc;  Override`

Visibility: protected

25.7.6 TStringTableResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                ;  Override
```

Visibility: protected

25.7.7 TStringTableResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                 ;  Override
```

Visibility: protected

25.7.8 TStringTableResource.NotifyResourcesLoaded

Declaration: `procedure NotifyResourcesLoaded;  Override`

Visibility: protected

25.7.9 TStringTableResource.Create

Synopsis: Creates a new string table resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                   ;  Override
```

Visibility: public

Description: Please note that `aType` parameter is not used, since this class always uses RT_STRING (177) as type.

Remark `aName` must be a TResourceDesc (177) of type dtID (177) and its value must be in the range 1-4096, otherwise an EStringTableNameNotAllowedException (178) exception is raised.

Errors: If `aName` is not of type dtID (177) and/or its value isn't in the range 1-4096, an EStringTableNameNotAllowedException (178) exception is raised.

See also: TStringTableResource.FirstID (180)

25.7.10 TStringTableResource.Destroy

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

Visibility: public

25.7.11 TStringTableResource.UpdateRawData

Declaration:
```
procedure UpdateRawData;  Override
```

Visibility: public

25.7.12 TStringTableResource.FirstID

Synopsis: The ID of first the string contained in the string table

Declaration:
```
Property FirstID : Word
```

Visibility: public

Access: Read,Write

Description: This property holds the value of the ID of the first string of the table. It is a multiple of 16.

The name of the resource is determined by this property, so changing FirstID also changes the resource name.

Remark If an attempt of setting this property to an integer that isn't a multiple of 16 is made, the value is automatically corrected to the closest multiple of 16 less than the value specified (e.g. setting it to 36 sets it to 32).

See also: TStringTableResource (178), TStringTableResource.LastID (181), TStringTableResource.Strings (181)

25.7.13 TStringTableResource.LastID

Synopsis: The ID of the last string contained in the string table

Declaration: `Property LastID : Word`

Visibility: public

Access: Read

Description: The value of this property is always FirstID (180)+15.

See also: TStringTableResource (178), TStringTableResource.FirstID (180), TStringTableResource.Strings (181)

25.7.14 TStringTableResource.Count

Synopsis: The number of strings contained in the string table

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: Since a string table resource always contains exactly 16 strings, this property is always 16

See also: TStringTableResource (178), TStringTableResource.FirstID (180)

25.7.15 TStringTableResource.Strings

Synopsis: Indexed array of strings in the string table

Declaration: `Property Strings[id: Word]: string; default`

Visibility: public

Access: Read,Write

Description: This property can be used to access all strings in the object.

Remark Strings are accessed by their ID: valid elements range from FirstID (180) to LastID (181). If the index specified isn't in this range, an EStringTableIndexOutOfBoundsException (178) exception is raised.

Remark If you need to access RawData (177) after you modified strings, be sure to call UpdateRawData (177) first. This isn't needed however when resource is written to a stream, since TResources (177) takes care of it.

See also: TStringTableResource (178), TStringTableResource.FirstID (180), TStringTableResource.LastID (181)

Chapter 26

Reference for unit 'versionconsts'

26.1 Overview

This unit contains constants used by TVersionResource (182).
Constants for TVersionFixedInfo.FileFlags (182):

- VS_FF_DEBUG (187).
- VS_FF_INFOINFERRED (187).
- VS_FF_PATCHED (187).
- VS_FF_PRERELEASE (187).
- VS_FF_PRIVATEBUILD (187).
- VS_FF_SPECIALBUILD (187).

Constants for TVersionFixedInfo.FileOS (182):

- VOS_UNKNOWN (186)
- VOS_DOS (185)
- VOS_OS216 (186)
- VOS_OS232 (186)
- VOS_NT (186)
- VOS__WINDOWS16 (186)
- VOS__WINDOWS32 (186)
- VOS__PM16 (186)
- VOS__PM32 (186)
- VOS_DOS_WINDOWS16 (185)
- VOS_DOS_WINDOWS32 (186)
- VOS_OS216_PM16 (186)

- VOS_OS232_PM32 (186)
- VOS_NT_WINDOWS32 (186)

Constants for TVersionFixedInfo.FileType (182):

- VFT_UNKNOWN (185)
- VFT_APP (185)
- VFT_DLL (185)
- VFT_DRV (185)
- VFT_FONT (185)
- VFT_VXD (185)
- VFT_STATIC_LIB (185)

Constants for TVersionFixedInfo.FileSubType (182):

- VFT2_UNKNOWN (185)
- VFT2_DRV_COMM (183)
- VFT2_DRV_PRINTER (184)
- VFT2_DRV_KEYBOARD (184)
- VFT2_DRV_LANGUAGE (184)
- VFT2_DRV_DISPLAY (184)
- VFT2_DRV_MOUSE (184)
- VFT2_DRV_NETWORK (184)
- VFT2_DRV_SYSTEM (184)
- VFT2_DRV_INSTALLABLE (184)
- VFT2_DRV_SOUND (184)
- VFT2_FONT_RASTER (184)
- VFT2_FONT_VECTOR (185)
- VFT2_FONT_TRUETYPE (185)

26.2 Constants, types and variables

26.2.1 Constants

```
VFT2_DRV_COMM = $0000000A
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_DISPLAY = $00000004
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_INSTALLABLE = $00000008
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_KEYBOARD = $00000002
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_LANGUAGE = $00000003
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_MOUSE = $00000005
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_NETWORK = $00000006
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_PRINTER = $00000001
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_SOUND = $00000009
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_DRV_SYSTEM = $00000007
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_DRV (185).

```
VFT2_FONT_RASTER = $00000001
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_FONT (185).

```
VFT2_FONT_TRUETYPE = $00000003
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_FONT (185).

```
VFT2_FONT_VECTOR = $00000002
```

This constant is used for TVersionFixedInfo.FileSubType (182) when TVersionFixedInfo.FileType (182) is VFT_FONT (185).

```
VFT2_UNKNOWN = $00000000
```

This constant is used for TVersionFixedInfo.FileSubType (182).

```
VFT_APP = $00000001
```

This constant is used for TVersionFixedInfo.FileType (182).

```
VFT_DLL = $00000002
```

This constant is used for TVersionFixedInfo.FileType (182).

```
VFT_DRV = $00000003
```

This constant is used for TVersionFixedInfo.FileType (182).
The device driver type is specified in TVersionFixedInfo.FileSubType (182).

```
VFT_FONT = $00000004
```

This constant is used for TVersionFixedInfo.FileType (182).
The font driver type is specified in TVersionFixedInfo.FileSubType (182).

```
VFT_STATIC_LIB = $00000007
```

This constant is used for TVersionFixedInfo.FileType (182).

```
VFT_UNKNOWN = $00000000
```

This constant is used for TVersionFixedInfo.FileType (182).

```
VFT_VXD = $00000005
```

This constant is used for TVersionFixedInfo.FileType (182).

```
VOS_DOS = $00010000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_DOS_WINDOWS16 = $00010001
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_DOS_WINDOWS32 = $00010004
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_NT = $00040000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_NT_WINDOWS32 = $00040004
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_OS216 = $00020000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_OS216_PM16 = $00020002
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_OS232 = $00030000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_OS232_PM32 = $00030003
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS_UNKNOWN = $00000000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS__BASE = $00000000
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS__PM16 = $00000002
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS__PM32 = $00000003
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS__WINDOWS16 = $00000001
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VOS__WINDOWS32 = $00000004
```

This constant is used for TVersionFixedInfo.FileOS (182).

```
VS_FFI_FILEFLAGSMASK = $0000003F
```

This constant is used mainly for TVersionFixedInfo.FileFlagsMask (182) to signal which bits are used in TVersionFixedInfo.FileFlags (182) field.

```
VS_FF_DEBUG = $00000001
```

This constant is used for TVersionFixedInfo.FileFlags (182).

```
VS_FF_INFOINFERRED = $00000010
```

This constant is used for TVersionFixedInfo.FileFlags (182).

```
VS_FF_PATCHED = $00000004
```

This constant is used for TVersionFixedInfo.FileFlags (182).

```
VS_FF_PRERELEASE = $00000002
```

This constant is used for TVersionFixedInfo.FileFlags (182).

```
VS_FF_PRIVATEBUILD = $00000008
```

This constant is used for TVersionFixedInfo.FileFlags (182).

When this flag is set, a TVersionStringTable (182) in TVersionResource.StringFileInfo (182) should contain an item with `PrivateBuild` as key.

```
VS_FF_SPECIALBUILD = $00000020
```

This constant is used for TVersionFixedInfo.FileFlags (182).

When this flag is set, a TVersionStringTable (182) in TVersionResource.StringFileInfo (182) should contain an item with `SpecialBuild` as key.

Chapter 27

Reference for unit 'versionresource'

27.1 Used units

Table 27.1: Used units by unit 'versionresource'

Name	Page
Classes	**??**
resource	129
sysutils	**??**
versiontypes	193

27.2 Overview

This unit contains TVersionResource (189), a TAbstractResource (188) descendant specialized in handling resource of type RT_VERSION (188).

Adding this unit to a program's `uses` clause registers class TVersionResource (189) for type RT_VERSION (188) with TResourceFactory (188).

27.3 Constants, types and variables

27.3.1 Types

27.4 TVerBlockHeader

```
TVerBlockHeader = packed record
  length : Word;
  vallength : Word
  ;
  valtype : Word;
  key : string;
end
```

This type is internally used by TVersionResource (189)

27.5 TVersionResource

27.5.1 Description

This class represents a resource of type RT_VERSION (188).

A resource of this type provides version information for a Microsoft Windows executable or dll, which is shown when checking properties of a file in Windows Explorer.

Information is stored in FixedInfo (191), StringFileInfo (191) and VarFileInfo (192).

Remark This class doesn't allow its type to be changed to anything else than RT_VERSION (188), and its name to be different from 1. Attempts to do so result in a EResourceDescChangeNotAllowedException (188).

Remark If you need to access RawData (188) after you modified something, be sure to call UpdateRawData (188) first. This isn't needed however when resource is written to a stream, since TResources (188) takes care of it.

Sample code

This code creates a version information resource

```
const
  myVersion : TFileProductVersion = (1,2,0,0);

var
  resources : TResources;
  aRes : TVersionResource;
  st : TVersionStringTable;
  ti : TVerTranslationInfo;
begin
  aRes:=TVersionResource.Create(nil,nil); //it's always RT_VERSION and 1 respectivel
  resources:=TResources.Create;
  resources.Add(aRes);

  aRes.FixedInfo.FileVersion:=myversion;
  aRes.FixedInfo.ProductVersion:=myversion;
  aRes.FixedInfo.FileFlagsMask:=VS_FFI_FILEFLAGSMASK;
  aRes.FixedInfo.FileFlags:=0;
  aRes.FixedInfo.FileOS:=VOS_NT_WINDOWS32;
  aRes.FixedInfo.FileType:=VFT_APP;
  aRes.FixedInfo.FileSubType:=0;
  aRes.FixedInfo.FileDate:=0;

  st:=TVersionStringTable.Create('041004B0'); //Italian, unicode codepage
  st.Add('CompanyName','Foo Corporation');
  st.Add('FileDescription','Foo suite core program');
  st.Add('FileVersion','1.2');
  st.Add('ProductVersion','1.2');
  aRes.StringFileInfo.Add(st);

  ti.language:=$0410; //Italian
  ti.codepage:=$04B0; //Unicode codepage
  aRes.VarFileInfo.Add(ti);

  resources.WriteToFile('myresource.res');
  resources.Free; //destroys aRes as well.
```

```
end;
```

See also: TVersionResource.FixedInfo (191), TVersionResource.StringFileInfo (191), TVersionResource.VarFileInfo (192)

27.5.2 Method overview

Page	Method	Description
190	ChangeDescTypeAllowed	
190	ChangeDescValueAllowed	
191	Create	Creates a new version information resource
191	Destroy	
190	GetName	
190	GetType	
191	NotifyResourcesLoaded	
191	UpdateRawData	

27.5.3 Property overview

Page	Properties	Access	Description
191	FixedInfo	r	Language independent part of version information
191	StringFileInfo	r	Language dependent part of version information
192	VarFileInfo	r	List of supported languages

27.5.4 TVersionResource.GetType

Declaration:
```
function GetType : TResourceDesc;  Override
```

Visibility: protected

27.5.5 TVersionResource.GetName

Declaration:
```
function GetName : TResourceDesc;  Override
```

Visibility: protected

27.5.6 TVersionResource.ChangeDescTypeAllowed

Declaration:
```
function ChangeDescTypeAllowed(aDesc: TResourceDesc) : Boolean
                                ;  Override
```

Visibility: protected

27.5.7 TVersionResource.ChangeDescValueAllowed

Declaration:
```
function ChangeDescValueAllowed(aDesc: TResourceDesc) : Boolean
                                 ;  Override
```

Visibility: protected

27.5.8 TVersionResource.NotifyResourcesLoaded

Declaration: `procedure NotifyResourcesLoaded;  Override`

Visibility: protected

27.5.9 TVersionResource.Create

Synopsis: Creates a new version information resource

Declaration:
```
constructor Create;  Override
constructor Create(aType: TResourceDesc; aName: TResourceDesc)
                  ;  Override
```

Visibility: public

Description: Please note that `aType` and `aName` parameters are not used, since this class always uses RT_VERSION (188) as type and 1 as name.

27.5.10 TVersionResource.Destroy

Declaration: `destructor Destroy;  Override`

Visibility: public

27.5.11 TVersionResource.UpdateRawData

Declaration: `procedure UpdateRawData;  Override`

Visibility: public

27.5.12 TVersionResource.FixedInfo

Synopsis: Language independent part of version information

Declaration: `Property FixedInfo : TVersionFixedInfo`

Visibility: public

Access: Read

See also: TVersionFixedInfo (188)

27.5.13 TVersionResource.StringFileInfo

Synopsis: Language dependent part of version information

Declaration: `Property StringFileInfo : TVersionStringFileInfo`

Visibility: public

Access: Read

See also: TVersionStringFileInfo (188)

27.5.14 TVersionResource.VarFileInfo

Synopsis: List of supported languages

Declaration: `Property VarFileInfo : TVersionVarFileInfo`

Visibility: public

Access: Read

See also: TVersionVarFileInfo (188)

Chapter 28

Reference for unit 'versiontypes'

28.1 Used units

Table 28.1: Used units by unit 'versiontypes'

Name	Page
Classes	??
resource	129
sysutils	??

28.2 Overview

This unit contains classes used by TVersionResource (193).

28.3 Constants, types and variables

28.3.1 Resource strings

```
SVerStrTableDuplicateKey = 'Duplicate key ''%s'''
```

```
SVerStrTableKeyNotFound = 'Key ''%s'' not found'
```

```
SVerStrTableNameNotAllowed =
  'Name ''%s'' is not a valid 8-cipher hex sequence'
```

28.3.2 Types

```
PVerTranslationInfo = ^TVerTranslationInfo
```

Pointer to a TVerTranslationInfo

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

This type is a 4-element array of words that is used to represent a file or product version.

Major version number is stored in the lowest word

Example

Product version 4.2.1.1200 can be represented this way

```
const
  myver : TFileProductVersion = (4,2,1,1200);
```

28.4 TVerTranslationInfo

```
TVerTranslationInfo = packed record
  language : Word;
  codepage
  : Word;
end
```

This record represents a language id - codepage pair that is used by TVersionVarFileInfo (205).

28.5 EDuplicateKeyException

28.5.1 Description

This exception is raised when an attempt is made to add an item to a TVersionStringTable (201) object and the specified key is already present.

See also: TVersionStringTable.Add (202)

28.6 EKeyNotFoundException

28.6.1 Description

This exception is raised when the specified key is not found in the TVersionStringTable (201) object.

See also: TVersionStringTable.Delete (203), TVersionStringTable.Values (204)

28.7 ENameNotAllowedException

28.7.1 Description

This exception is raised when an attempt is made to set Name (203) property of TVersionStringTable (201) with a string that isn't an 8-cipher hexadecimal string.

See also: TVersionStringTable.Create (202), TVersionStringTable.Name (203)

28.8 EVersionStringTableException

28.8.1 Description

Base class for version string table - related exceptions

28.9 TVersionFixedInfo

28.9.1 Description

This class represents the language independent part of version information, and is always present in a version information resource.

See also: TVersionResource (193), TVersionResource.FixedInfo (193)

28.9.2 Method overview

Page	Method	Description
195	Create	

28.9.3 Property overview

Page	Properties	Access	Description
198	FileDate	rw	The file creation timestamp.
196	FileFlags	rw	The file flags
196	FileFlagsMask	rw	Mask for FileFlags
197	FileOS	rw	The operating system the file was designed to run on
198	FileSubType	rw	Additional type information
197	FileType	rw	The type of the file
195	FileVersion	rw	The file version
196	ProductVersion	rw	The product version

28.9.4 TVersionFixedInfo.Create

Declaration: `constructor Create`

Visibility: public

28.9.5 TVersionFixedInfo.FileVersion

Synopsis: The file version

Declaration: `Property FileVersion : TFileProductVersion`

Visibility: public

Access: Read,Write

See also: TFileProductVersion (194)

28.9.6 TVersionFixedInfo.ProductVersion

Synopsis: The product version

Declaration: `Property ProductVersion : TFileProductVersion`

Visibility: public

Access: Read,Write

See also: TFileProductVersion (194)

28.9.7 TVersionFixedInfo.FileFlagsMask

Synopsis: Mask for FileFlags

Declaration: `Property FileFlagsMask : LongWord`

Visibility: public

Access: Read,Write

Description: This property specifies which bits of FileFlags (196) are valid.
Usually it is VS_FFI_FILEFLAGSMASK (193).

See also: TVersionFixedInfo.FileFlags (196)

28.9.8 TVersionFixedInfo.FileFlags

Synopsis: The file flags

Declaration: `Property FileFlags : LongWord`

Visibility: public

Access: Read,Write

Description: It is a combination of the following values:

- VS_FF_DEBUG (193).
- VS_FF_INFOINFERRED (193).
- VS_FF_PATCHED (193).
- VS_FF_PRERELEASE (193).
- VS_FF_PRIVATEBUILD (193).
- VS_FF_SPECIALBUILD (193).

See also: TVersionFixedInfo.FileFlagsMask (196)

28.9.9 TVersionFixedInfo.FileOS

Synopsis: The operating system the file was designed to run on

Declaration: `Property FileOS : LongWord`

Visibility: public

Access: Read,Write

Description: It is one of the following values:

- VOS_UNKNOWN (193)
- VOS_DOS (193)
- VOS_OS216 (193)
- VOS_OS232 (193)
- VOS_NT (193)

combined with one of the following values:

- VOS__WINDOWS16 (193)
- VOS__WINDOWS32 (193)
- VOS__PM16 (193)
- VOS__PM32 (193)

Note: some predefined combinations do exist:

- VOS_DOS_WINDOWS16 (193)
- VOS_DOS_WINDOWS32 (193)
- VOS_OS216_PM16 (193)
- VOS_OS232_PM32 (193)
- VOS_NT_WINDOWS32 (193)

28.9.10 TVersionFixedInfo.FileType

Synopsis: The type of the file

Declaration: `Property FileType : LongWord`

Visibility: public

Access: Read,Write

Description: It can be one of the following values:

- VFT_UNKNOWN (193)
- VFT_APP (193)
- VFT_DLL (193)
- VFT_DRV (193)
- VFT_FONT (193)
- VFT_VXD (193)
- VFT_STATIC_LIB (193)

See also: TVersionFixedInfo.FileSubType (198)

28.9.11 TVersionFixedInfo.FileSubType

Synopsis: Additional type information

Declaration: `Property FileSubType : LongWord`

Visibility: public

Access: Read,Write

Description: This property is meaningful only for some values of FileSubType (198). For all other types, this property must be zero.

If FileSubType (198) is VFT_DRV (193):

- VFT2_UNKNOWN (193)
- VFT2_DRV_COMM (193)
- VFT2_DRV_PRINTER (193)
- VFT2_DRV_KEYBOARD (193)
- VFT2_DRV_LANGUAGE (193)
- VFT2_DRV_DISPLAY (193)
- VFT2_DRV_MOUSE (193)
- VFT2_DRV_NETWORK (193)
- VFT2_DRV_SYSTEM (193)
- VFT2_DRV_INSTALLABLE (193)
- VFT2_DRV_SOUND (193)

If FileSubType (198) is VFT_FONT (193):

- VFT2_UNKNOWN (193)
- VFT2_FONT_RASTER (193)
- VFT2_FONT_VECTOR (193)
- VFT2_FONT_TRUETYPE (193)

See also: TVersionFixedInfo.FileType (197)

28.9.12 TVersionFixedInfo.FileDate

Synopsis: The file creation timestamp.

Declaration: `Property FileDate : QWord`

Visibility: public

Access: Read,Write

Description: It is a 64 bit timestamp.

28.10 TVersionStringFileInfo

28.10.1 Description

This class represents the language dependent part of version information.

It is a list of TVersionStringTable (201) objects, each representing information for a specific language-codepage combination.

See also: TVersionResource (193), TVersionResource.StringFileInfo (193), TVersionStringTable (201)

28.10.2 Method overview

Page	Method	Description
200	Add	Adds a new string table
200	Clear	Deletes all string tables in the list
199	Create	
200	Delete	Deletes a string table
199	Destroy	Destroys the object
199	GetCount	
199	GetItem	
199	SetItem	

28.10.3 Property overview

Page	Properties	Access	Description
200	Count	r	The number of string tables in the object
200	Items	rw	Indexed array of string tables in the object

28.10.4 TVersionStringFileInfo.GetCount

Declaration: `function GetCount : Integer`

Visibility: protected

28.10.5 TVersionStringFileInfo.GetItem

Declaration: `function GetItem(index: Integer) : TVersionStringTable`

Visibility: protected

28.10.6 TVersionStringFileInfo.SetItem

Declaration: `procedure SetItem(index: Integer; aValue: TVersionStringTable)`

Visibility: protected

28.10.7 TVersionStringFileInfo.Create

Declaration: `constructor Create`

Visibility: public

28.10.8 TVersionStringFileInfo.Destroy

Synopsis: Destroys the object

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: All items are destroyed as well.

Remark:

See also: TVersionStringFileInfo.Clear (200)

28.10.9 TVersionStringFileInfo.Add

Synopsis: Adds a new string table

Declaration: `procedure Add(aStrTable: TVersionStringTable)`

Visibility: public

See also: TVersionStringFileInfo.Delete (200)

28.10.10 TVersionStringFileInfo.Clear

Synopsis: Deletes all string tables in the list

Declaration: `procedure Clear`

Visibility: public

Description: This method empties the object. All string tables are freed.

See also: TVersionStringFileInfo.Delete (200), TVersionStringFileInfo.Add (200)

28.10.11 TVersionStringFileInfo.Delete

Synopsis: Deletes a string table

Declaration: `procedure Delete(const aIndex: Integer)`

Visibility: public

Description: This method removes the string table identified by `aIndex` from the list. The string table is freed.

See also: TVersionStringFileInfo.Clear (200), TVersionStringFileInfo.Add (200)

28.10.12 TVersionStringFileInfo.Count

Synopsis: The number of string tables in the object

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

See also: TVersionStringFileInfo.Items (200)

28.10.13 TVersionStringFileInfo.Items

Synopsis: Indexed array of string tables in the object

Declaration: `Property Items[index: Integer]: TVersionStringTable; default`

Visibility: public

Access: Read,Write

Description: This property can be used to access all string tables in the object.

Remark This array is 0-based: valid elements range from 0 to Count (200)-1.

See also: TVersionStringFileInfo.Count (200), TVersionStringTable (201)

28.11 TVersionStringTable

28.11.1 Description

This class represents version information for a specific language-codepage combination.

It is contained in a TVersionStringFileInfo (198) object.

Information is stored as key-value pairs. The name of the string table specifies the language id - codepage to which the object applies.

There are some predefined keys that Microsoft Windows searches for. They are:

- `Comments`
- `CompanyName`
- `FileDescription`
- `FileVersion`
- `InternalName`
- `LegalCopyright`
- `LegalTrademarks`
- `OriginalFilename`
- `PrivateBuild` (only if VS_FF_PRIVATEBUILD (193) is set in TVersionFixedInfo.FileFlags (196))
- `ProductName`
- `ProductVersion`
- `SpecialBuild` (only if VS_FF_SPECIALBUILD (193) is set in TVersionFixedInfo.FileFlags (196))

See also: TVersionStringFileInfo (198), TVersionStringTable.Name (203), TVersionStringTable.Keys (204), TVersionStringTable.Values (204), TVersionStringTable.ValuesByIndex (204)

28.11.2 Method overview

Page	Method	Description
202	Add	Adds a new item to the string table
203	Clear	Deletes all items
202	Create	Creates a new string table
203	Delete	Deletes an item
202	Destroy	Destroys the string table

28.11.3 Property overview

Page	Properties	Access	Description
203	Count	r	The number of items in the object
204	Keys	r	Indexed array of keys in the object
203	Name	r	The name of the string table
204	Values	rw	Array of values in the object, accessed by key
204	ValuesByIndex	rw	Indexed array of values in the object

28.11.4 TVersionStringTable.Create

Synopsis: Creates a new string table

Declaration: `constructor Create(const aName: string)`

Visibility: public

Description: Creates a new string table with the name passed as `aName` parameter.

`aName` must be a hex representation of a 4-bytes unsigned number: first 4 ciphers represent the language id, and last 4 the codepage.

Sample code

This code creates a string table for Italian, with unicode codepage.

```
var
  st : TVersionStringTable;
begin
  //0410 = Italian
  //04B0 = unicode codepage
  st:=TVersionStringTable.Create('041004B0');

  //do something useful...

  st.Free;
end;
```

Errors: If name is not a valid 8-cipher hexadecimal string, an ENameNotAllowedException (194) exception is raised.

See also: TVersionStringTable.Name (203)

28.11.5 TVersionStringTable.Destroy

Synopsis: Destroys the string table

Declaration: `destructor Destroy;  Override`

Visibility: public

28.11.6 TVersionStringTable.Add

Synopsis: Adds a new item to the string table

Declaration: `procedure Add(const aKey: string; const aValue: string)`

Visibility: public

Description: This methods adds a new key-value pair to the string table.

Note that some predefined keys do exist. See TVersionStringTable (201) for further information.

Errors: If an item with the same key already exists, an EDuplicateKeyException (194) exception is raised.

See also: TVersionStringTable (201), TVersionStringTable.Keys (204), TVersionStringTable.Values (204), TVersionStringTable.ValuesByIndex (204)

28.11.7 TVersionStringTable.Clear

Synopsis: Deletes all items

Declaration:
```
procedure Clear
```

Visibility: public

Description: This method empties the object deleting all items.

See also: TVersionStringTable.Delete (203)

28.11.8 TVersionStringTable.Delete

Synopsis: Deletes an item

Declaration:
```
procedure Delete(const aIndex: Integer);  Overload
procedure Delete(const aKey: string);  Overload
```

Visibility: public

Description: The item to delete can be specified by its index or by its key.

Errors: If `aKey` is not found, an EKeyNotFoundException (194) exception is raised.

See also: TVersionStringTable.Keys (204), TVersionStringTable.Values (204), TVersionStringTable.ValuesByIndex (204)

28.11.9 TVersionStringTable.Name

Synopsis: The name of the string table

Declaration:
```
Property Name : string
```

Visibility: public

Access: Read

Description: Name must be a hex representation of a 4-bytes unsigned number: first 4 ciphers represent the language id, and last 4 the codepage.

Errors: If an attempt is made to set Name with a string that isn't an 8-cipher hexadecimal string, an ENameNotAllowedException (194) exception is raised.

See also: TVersionStringTable.Create (202)

28.11.10 TVersionStringTable.Count

Synopsis: The number of items in the object

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

Visibility: public

Access: Read

See also: TVersionStringTable.ValuesByIndex (204)

28.11.11 TVersionStringTable.Keys

Synopsis: Indexed array of keys in the object

Declaration: `Property Keys[index: Integer]: string`

Visibility: public

Access: Read

Description: This property can be used to access all keys in the object.

Values associated to keys are stored in ValuesByIndex (204) property: a key and its value have the same index in the two properties.

Remark This array is 0-based: valid elements range from 0 to Count (203)-1.

Note that some predefined keys do exist. See TVersionStringTable (201) for further information.

See also: TVersionStringTable (201), TVersionStringTable.Values (204), TVersionStringTable.ValuesByIndex (204)

28.11.12 TVersionStringTable.ValuesByIndex

Synopsis: Indexed array of values in the object

Declaration: `Property ValuesByIndex[index: Integer]: string`

Visibility: public

Access: Read,Write

Description: This property can be used to access all values in the object.

Keys associated to values are stored in Keys (204) property: a key and its value have the same index in the two properties.

Remark This array is 0-based: valid elements range from 0 to Count (203)-1.

See also: TVersionStringTable.Keys (204), TVersionStringTable.Values (204)

28.11.13 TVersionStringTable.Values

Synopsis: Array of values in the object, accessed by key

Declaration: `Property Values[Key: string]: string; default`

Visibility: public

Access: Read,Write

Description: This property can be used to retrieve the value of an item when the corresponding key is known.

If you need to iterate over all values of the string table, use ValuesByIndex (204) instead.

If the key is not found, an EKeyNotFoundException (194) exception is raised.

See also: TVersionStringTable.Keys (204), TVersionStringTable.ValuesByIndex (204)

28.12 TVersionVarFileInfo

28.12.1 Description

This class represents the language-codepage pairs that the program or dll supports. It can be used to avoid including several RT_VERSION (193) resources for each language-codepage supported.

It is a list of TVerTranslationInfo (194) records.

See also: TVersionResource.VarFileInfo (193), TVerTranslationInfo (194)

28.12.2 Method overview

Page	Method	Description
206	Add	Adds a new translation information item
206	Clear	Deletes all items
205	Create	
206	Delete	Deletes an item
206	Destroy	Destroys the object
205	GetCount	
205	GetItem	
205	SetItem	

28.12.3 Property overview

Page	Properties	Access	Description
206	Count	r	The number of items in the object
207	Items	rw	Indexed array of items in the object

28.12.4 TVersionVarFileInfo.GetCount

Declaration: `function GetCount : Integer`

Visibility: protected

28.12.5 TVersionVarFileInfo.GetItem

Declaration: `function GetItem(index: Integer) : TVerTranslationInfo`

Visibility: protected

28.12.6 TVersionVarFileInfo.SetItem

Declaration: `procedure SetItem(index: Integer; aValue: TVerTranslationInfo)`

Visibility: protected

28.12.7 TVersionVarFileInfo.Create

Declaration: `constructor Create`

Visibility: public

28.12.8 TVersionVarFileInfo.Destroy

Synopsis: Destroys the object

Declaration: `destructor Destroy;  Override`

Visibility: public

Description: All items are destroyed as well.

See also: TVersionVarFileInfo.Clear (206)

28.12.9 TVersionVarFileInfo.Add

Synopsis: Adds a new translation information item

Declaration: `procedure Add(aInfo: TVerTranslationInfo)`

Visibility: public

See also: TVersionVarFileInfo.Items (207)

28.12.10 TVersionVarFileInfo.Clear

Synopsis: Deletes all items

Declaration: `procedure Clear`

Visibility: public

Description: This method empties the object deleting all items.

See also: TVersionVarFileInfo.Delete (206)

28.12.11 TVersionVarFileInfo.Delete

Synopsis: Deletes an item

Declaration: `procedure Delete(const aIndex: Integer)`

Visibility: public

See also: TVersionVarFileInfo.Items (207)

28.12.12 TVersionVarFileInfo.Count

Synopsis: The number of items in the object

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

See also: TVersionVarFileInfo.Items (207)

28.12.13 TVersionVarFileInfo.Items

Synopsis: Indexed array of items in the object

Declaration: `Property Items[index: Integer]: TVerTranslationInfo; default`

Visibility: public

Access: Read,Write

Description: This property can be used to access all translation information items in the object.

Remark This array is 0-based: valid elements range from 0 to Count (206)-1.

See also: TVersionVarFileInfo.Count (206)

Chapter 29

Reference for unit 'winpeimagereader'

29.1 Used units

Table 29.1: Used units by unit 'winpeimagereader'

Name	Page
Classes	??
coffreader	52
resource	129
sysutils	??

29.2 Overview

This unit contains TWinPEImageResourceReader (208), a TAbstractResourceReader (208) descendant that is able to read Microsoft Windows PE image files (executables, dynamic link libraries and so on).

Adding this unit to a program's `uses` clause registers class TWinPEImageResourceReader (208) with TResources (208).

29.3 TWinPEImageResourceReader

29.3.1 Description

This class provides a reader for Microsoft Windows PE image files containing resources.

PE is the file format used by Microsoft Windows executables, dynamic link libraries and so on.

Remark This reader can only read full PE images, not COFF object files. Use TCoffResourceReader (208) instead.

See also: TAbstractResourceReader (208), TCoffResourceReader (208)

29.3.2 Method overview

Page	Method	Description
209	CheckDosStub	
209	CheckMagic	
209	CheckPESignature	
209	Create	
210	Destroy	
209	GetDescription	
209	GetExtensions	
209	Load	

29.3.3 TWinPEImageResourceReader.CheckDosStub

Declaration: `function CheckDosStub(aStream: TStream) : Boolean`

Visibility: protected

29.3.4 TWinPEImageResourceReader.CheckPESignature

Declaration: `function CheckPESignature(aStream: TStream) : Boolean`

Visibility: protected

29.3.5 TWinPEImageResourceReader.GetExtensions

Declaration: `function GetExtensions : string;  Override`

Visibility: protected

29.3.6 TWinPEImageResourceReader.GetDescription

Declaration: `function GetDescription : string;  Override`

Visibility: protected

29.3.7 TWinPEImageResourceReader.Load

Declaration: `procedure Load(aResources: TResources; aStream: TStream);  Override`

Visibility: protected

29.3.8 TWinPEImageResourceReader.CheckMagic

Declaration: `function CheckMagic(aStream: TStream) : Boolean;  Override`

Visibility: protected

29.3.9 TWinPEImageResourceReader.Create

Declaration: `constructor Create;  Override`

Visibility: public

29.3.10 TWinPEImageResourceReader.Destroy

Declaration: `destructor Destroy; Override`

Visibility: public