Stream Classes User Guide

This class library provides some classes for Delphi 4 to 7 that can be used to extend the functionality of Delphi's *TStream* classes.

Overview

The classes included in this release are as follows:

Class	Unit	Description
TPJStreamWrapper	*PJStreamWrapper*	This is a base class for descendant classes that "wrap" a *TStream* class to provide some form of filter or additional functionality. The wrapped *TStream* is used to do physical i/o. *TStreamWrapper* simply replicates the facilities in the wrapped stream - it is for descendant classes to add functionality.
TPJCustomResWriterStream	*PJResWriterStreams*	Base class for all resource stream writer classes. Provides some basic functionality required by all stream involved in writing resource files.
TPJResWriterStream	*PJResWriterStreams*	Writes a resource file in correct format. It is responsible for writing the required header data. This class should not be used on its own, but should be wrapped by a *TPJResDataStream* class (see below)
TPJResDataStream	*PJResWriterStreams*	Writes data to a resource of a specified type contained in a resource file written using a *TPJResWriterStream* object.
TPJResHeader	*PJResWriterStreams*	Creates a resource file header record of required kind. Properties of the header are exposed and the class can write the header to the current location in any given stream. This class is essentially private to the other classes in this unit.
TPJIStreamWrapper	*PJIStreams*	Implements the *IStream* interface for a wrapped *TStream* object.
TPJHandleIStreamWrapper	*PJIStreams*	Implements an *IStream* interface for a wrapped *THandleStream* object (or descendant such as *TFileStream*). Acts in a similar way to *TPJIStreamWrapper* except that file date stamps are returned by the Stat method.
TPJFileIStream	*PJIStreams*	Implements a *IStream* interface on a file.

TPJStreamWrapper

This class is not normally used on its own. It is designed as a descendant for classes that "wrap" a stream. Such classes will add methods to the base class and/or override the base class's handling of the Read, Write and Seek methods. Study the source code of *TPJCustomResWriterStream* for details of how this can be achieved. The only method that is different to those inherited from *TStream* is

```
constructor Create(const Stream: TStream;
  const CloseStream: Boolean = False);
```

- **`Stream`** is the stream that is "wrapped" by this object
- **`CloseStream`** is an optional parameter that is False by default. When true the parameter causes the wrapped stream to be freed when this object is destroyed. This saves the used from having to remember to free the wrapped stream. It allows the wrapped streams that are created "on the fly" in this constructor call to be freed without keeping a reference to them.

Resource Stream Classes

The purpose of these classes is to allow the creation of resource files that can be linked into executable programs without the use of a resource editor. The classes are particularly useful in creating custom RCDATA resources. The public resource classes all ultimately descend from *TStream* via *TPJStreamWrapper*.

The only classes that need to be accessed by the user are *TPJResWriterStream* (which looks after the format of the resource file itself) and *TPJResDataStream* (which writes individual resources within a resource file). The classes are always used together. *TPJResDataStream* wraps around a *TPJResWriterStream* object which in turn wraps a *TStream* that is attached to the resource file.

It is easiest to explain the purpose of these classes by presenting an example. Suppose we need to write two custom resources name `RES1` and `RES2` to a resource file named `MyResource.res` we can use the following code (Assume Data1 and Data2 are records containing all the data that needs to be written to RES1 and RES2 respectively):

```
var
  FS: TFileStream;
  RCDATA1, RCDATA2: TResDataStream;
  RS: TResWriterStream;
begin
  FS := TFileStream.Create('MyResource.res', fmCreate);
  RS := TPJResWriterStream.Create(FS, True);
  RCDATA1 := TPJResDataStream.Create(FS, 'RES1', RS, False);
  RCDATA1.WriteBuffer(Data1, SizeOf(Data1));
  RCDATA1.Free;
  RCDATA2 := TPJResDataStream.Create(FS, 'RES2', RS, True);
  RCDATA2.WriteBuffer(Data2, SizeOf(Data2));
  RCDATA2.Free;
end;
```

Note that we never free FS or RS. This is because RCDATA2 has been asked to automatically free RS which in turn automatically frees FS.

Once you have got the hang of using the classes in conjunction with each other, the classes are very easy to use - you simply construct the required classes and then write to the stream as you normally would using the WriteBuffer and Write methods of *TStream*. The only methods and properties that vary from *TStream* are explained below.

TPJResWriterStream

```
constructor Create(const Stream: TStream;
  const CloseStream: Boolean = False);
```

- The parameters are the same as for *TPJStreamWrapper*.

TPJResDataStream

```
constructor Create(const ResId: PChar;
  const ResType: PChar;
  const Stream: TResWriterStream;
  const CloseStream: Boolean = False);
constructor Create(const ResId: string;
  const ResType: PChar;
  const Stream: TResWriterStream;
  const CloseStream: Boolean = False);
constructor Create(const ResId: Integer;
  const ResType: PChar;
  const Stream: TResWriterStream;
  const CloseStream: Boolean = False);
```

- **ResId** is the resource identifier. The overloaded versions of the constructor all vary in this parameter - it can either be a name (as a string or a PChar) or an integer ID.
- **ResType** is the type of the resource. This is often RT_RCDATA, but can be any valid value although it should be noted that this class knows nothing of the internal structure of the predefined resource formats.
- **Stream** is the *TPJResWriterStream* that is associated with the resource file. This class outputs its data via that stream.
- **CloseStream** operates in a similar way as the same parameter in *TPJStreamWrapper* - except that it applies to the "wrapped" *TPJResWriterStream* object rather than a *TStream* object.

```
property LanguageID: WORD;
```

- This property sets the language ID that is associated with the resource. See the Windows API help for more information on language IDs.

IStream Classes

The purpose of this group of classes is to implement the *IStream* interface for various kinds of stream. The following classes are available. For details of the *IStream* methods see the Windows help file documentation.

TPJIStreamWrapper

This class can wrap any *TStream* derived class and provide access to it by means of an *IStream* interface. The stream to be wrapped is simply passed to the class constructor:

```
constructor Create(const Stream: TStream;
  const CloseStream: Boolean = False);
```

- **Stream** is the stream that is "wrapped" by this object
- **CloseStream** is an optional parameter that is False by default. When true the parameter causes the wrapped stream to be freed when this object is destroyed. This saves the user from having to remember to free the wrapped stream. It allows the wrapped streams that are created "on the fly" in this constructor call to be freed

without keeping a reference to them.

The following comments apply to the *IStream* methods:

```
function Read(pv: Pointer; cb: Longint;
  pcbRead: PLongint): HResult; virtual; stdcall;
```

- Reads a specified number of bytes from the stream object into memory starting at the current seek pointer. Sets pcbRead, if not nil, to number of bytes actually read.

```
function Write(pv: Pointer; cb: Longint;
  pcbWritten: PLongint): HResult; virtual; stdcall;
```

- Writes a specified number of bytes into the stream object starting at the current seek pointer. The number of bytes actually written is returned in pcbWritten if this is non nil.

```
function Seek(dlibMove: Largeint; dwOrigin: Longint;
  out libNewPosition: Largeint): HResult; virtual; stdcall;
```

- Changes the seek pointer to a new location relative to the beginning of the stream, the end of the stream, or the current seek pointer. Returns the new seek pointer position in libNewPosition.

```
function SetSize(libNewSize: Largeint): HResult; virtual; stdcall;
```

- Changes the size of the stream object.

```
function CopyTo(stm: IStream;
  cb: Largeint; out cbRead: Largeint;
  out cbWritten: Largeint): HResult; virtual; stdcall;
```

- Copies a specified number of bytes from the current seek pointer in the stream to the current seek pointer in another stream. The number of bytes actually read and written is recorded in cbRead and cbWritten. If the source stream has less than the required number of bytes available then all remaining bytes are written.

```
function Commit(grfCommitFlags: Longint): HResult; virtual; stdcall;
```

- Provided in *IStream* implementations that support transacted streams to ensure that any changes made to a stream object open in transacted mode are reflected in the parent storage object. Since we don't support transacted mode there's nothing to do here.

```
function Revert: HResult; virtual; stdcall;
```

- Discards all changes that have been made to a transacted stream since the last *IStream.Commit* call. Since we don't supported transacted streams we just return that we've reverted the stream.

```
function LockRegion(libOffset: Largeint; cb: Largeint;
  dwLockType: Longint): HResult; virtual; stdcall;
```

- Restricts access to a specified range of bytes in the stream. It is optional to support this method, and we don't!

```
function UnlockRegion(libOffset: Largeint; cb: Largeint;
  dwLockType: Longint): HResult; virtual; stdcall;
```

- Removes the access restriction on a range of bytes previously restricted with *IStream.LockRegion*. We don't support locking.

```
function Stat(out statstg: TStatStg;
  grfStatFlag: Longint): HResult; virtual; stdcall;
```

- Retrieves the STATSTG structure for this stream. grfStatFlag can be STATFLAG_DEFAULT, which omits the stream name from the structure, or STATFLAG_NORMAL, which includes the stream name. In the latter case the name should be freed using *IMalloc*.

- The only supported *TStatStg* elements are:
    - dwType (= STGTY_STREAM)
    - cbSize (= size of underlying stream)
    - pwcsName (= name of stream made up of wrapper class followed by name of wrapped class in parentheses.

```
function Clone(out stm: IStream): HResult; virtual; stdcall;
```

- Not implemented. (Where implemented Clone creates a new stream object that references the same bytes as the original stream but provides a separate seek pointer to those bytes).

TPJHandleIStreamWrapper

This class provides an *IStream* interface to any wrapped object of type *THandleStream* or descendant (such as *TFileStream*) - i.e. any stream based on a Windows file handle. It acts as *TPJIStreamWrapper* except that the *TStatStg* structure returned by the Stat function also returns information about file creation, modification and access dates where available.

This class has a constructor that only accepts stream which descend from *THandleStream*:

```
constructor Create(const Stream: THandleStream;
  const CloseStream: Boolean);
```

TPJFileIStream

TPJFileIStream provides an interface to a file. It's constructor is like that of *TFileStream*:

```
constructor Create(const FileName: string; Mode: Word);
```

However, in this instance the constructor returns an object of type *TPJFileIStream* which can be cast to *IStream*.

The Stat function, like that of *TPJHandleIStreamWrapper*, provides file date/time stamp information. Additionally, the pwcsName element of the *TStatStg* structure returns the name of the open file rather than a name based on the class name.