**Потоки данных в Delphi**

Под потоком здесь понимается объект **TStream**, который является базовым объектом для потоков разных типов. В этом объекте реализованы все необходимые свойства и методы, используемые для чтения и записи данных на различные типы носителей. Благодаря этому объекту доступ к разным типам носителей становится одинаковым.   
Объект TFileStream является потомком главного объекта TStream и позволяет получить доступ к диску. Точно так же можно получить доступ:

* к памяти через объект TMemoryStream;
* к сети через объект TWinSocketStream;
* к СОМ-интерфейсу через объект TOleStream;
* к строкам, находящимся в динамической памяти, через объект TStringStream.

Это неполный список объектов потоков. Рассмотрим только базовый объект TStream. Работа с другими объектами идентична и точно так же можно работать с любым другим типом потока.  
**Свойства:**

**Position** — указывает на текущую позицию курсора в потоке. Начиная с этой позиции будет происходить чтение данных;  
**Size** — размер данных в потоке.

**Методы:**

**CopyFrom** — метод предназначен для копирования из другого потока. У него два параметра: указатель на поток, из которого надо копировать, и число, показывающее размер данных, подлежащих копированию;  
**Read** — прочитать данные из потока, начиная с текущей позиции курсора. У этого метода два параметра — буфер, в который будет происходить чтение, и число, показывающее размер данных для копирования;  
**Seek** — переместиться в новую позицию в потоке. У этого метода два параметра:  
 1. Первый параметр: число, указывающее на позицию, в которую надо перейти. Если вам нужно передвинуться на пять байт, то просто укажите цифру 5.

2.Второй параметр: откуда надо двигаться. Тут возможны три варианта:

* 1. soFromBeginning — двигаться на указанное количество байт от начала файла;
  2. soFromCurrent — двигаться на указанное количество байт от текущей позиции в файле к концу файла;
  3. soFromEnd — двигаться от конца файла к началу на указанное количество байт.

**SetSize** — установить размер потока. Здесь только один параметр — число, указывающее новый размер потока. Допустим, что вам надо уменьшить размер файла. В этом случае с помощью метода **SetSize** потока TFileStream вы можете уменьшить или даже увеличить размер файла.  
**Write** — записать данные в текущую позицию потока. У этого метода два параметра:

1. переменная, содержимое которой нужно записать;
2. число байт для записи.

### Упрощаем работу с потоками (TStream)

*Юрий Спектор,*[*http://www.delphikingdom.ru*](http://www.delphikingdom.ru/)

Работа программиста невозможна без работы с данными, которые хранятся в файлах или в памяти. В Delphi введен механизм потокового ввода-вывода, значительно упрощающий наш нелегкий труд. Однако структура данных может быть достаточно сложна. К тому же, в разных проектах она наверняка будет различна. Все это заставляет нас снова и снова писать сотни строчек однообразного кода записи/чтения данных. Утомляет. В этой я покажу, как я решил эту проблему для себя.

#### Совсем немного теории

Для тех, кто знает, что скрывается за страшной аббревиатурой RTTI, рекомендую пропустить этот раздел, ничего нового Вы здесь не найдете. А остальных попытаюсь немного ввести в курс дела.

RTTI (Run-time type information) - как видно из названия, это механизм, позволяющий определить тип данных во время выполнения. Суть его в том, что компилятор генерирует расширенную информацию для почти всех классов, используемых в вашей программе. Я сказал почти? Да, только для классов, объявленных с директивой {$M+} и их потомков, а таким классом, в частности является TPersistent. Потомками этого класса являются все компоненты, графические классы (TFont, TBitmap, TIcon и т.д.) и многие другие. Так вот, я отвлекся, эта информация активно используется самой средой разработки (инспектор объектов, редакторы свойств) и может быть использована программистом. Необходимые средства для работы с RTTI находятся в модуле TypInfo.pas. Проблема лишь в том, что по неизвестным мне причинам, Borland решила не документировать эти возможности (в справке по Delphi7, не нашел ничего связанного с RTTI, кроме упомянутой ранее директивы {$M+/-}, метода TObject.ClassInfo и операторов is и as).

И еще: RTTI позволяет получить информацию о свойствах и методах, объявленных ТОЛЬКО в разделе published. Зачем нам это нужно и как это нам поможет - увидите далее.

#### Ставим задачу

Поставим себе такую задачу: создать класс, который будет искать все свои published-свойства и сохранять их в поток (в файл, в частности). Программисту нужно только ОДИН раз написать код, который реализует сказанное выше, создать потомка этого класса, объявить в нем все необходимые свойства, и вызвать метод SaveToStream (его не надо будет перекрывать для каждого потомка) для сохранения самого себя в поток. Аналогично, метод LoadFromStream прочитает все свойства из потока. Ну-с, приступим-с.

#### Реализация

Класс TRttiObject

// Сохраняет и читает из потока все Published свойства

TRttiObject = **class**(TInterfacedPersistent, IStreamPersist)

**public**

**procedure** SaveToStream(Stream: TStream);

**procedure** LoadFromStream(Stream: TStream);

**constructor** Create; **virtual**;

**end**;

Тут нужно немного пояснить. Класс будет записывать/читать все свойства, тип которых целый (в том числе логический), перечислимый, вещественный, символьный, строковый, а также некоторые классы, которые поддерживают работу с потоками. Отличить эти классы от других, можно запросив у них интерфейс IStreamPersist, который объявлен в Classes.pas так:

IStreamPersist = **interface**

['{B8CD12A3-267A-11D4-83DA-00C04F60B2DD}']

**procedure** LoadFromStream(Stream: TStream);

**procedure** SaveToStream(Stream: TStream);

**end**;

Этот класс реализуют, например, все потомки TGraphic, такие как TBitmap, TIcon, TMetafile, а также наш класс TRttiObject.

Почему в качестве предка выбран TInterfacedPersistent, а не TObject или TInterfacedObject. Тут несколько причин: во-первых, он является потомком TPersistent, который объявлен с директивой {$M+} (правда ничего не мешало бы сделать это самим), а во-вторых, в нем наиболее удачно для нас реализованы методы интерфейса IInterface (подсчет ссылок, реализованный в TInterfacedObject, нам ни к чему, а если взять TObject, то эти методы нужно будет реализовать самому).

**procedure** TRttiObject.SaveToStream(Stream: TStream);

**var**

TypeData: PTypeData;

PropList: PPropList;

Count,i: Integer;

// Локальные процедуры

**procedure** WriteOrdProp; // Запись целых и перечислимых данных

**var**

Value: Integer;

**begin**

Value:=GetOrdProp(self,PropList[i]);

Stream.Write(Value,SizeOf(Value));

**end**;

**procedure** WriteFloatProp; // Запись вещественных данных

**var**

Value: Extended;

**begin**

Value:=GetFloatProp(self,PropList[i]);

Stream.Write(Value,SizeOf(Value));

**end**;

**procedure** WriteStringProp; // Запись строки

**var**

Value: **String**;

L: Integer;

**begin**

Value:=GetStrProp(self,PropList[i]);

L:=Length(Value);

Stream.Write(L,SizeOf(L));

Stream.Write(PChar(Value)^,Length(Value));

**end**;

**procedure** WriteClassProp; // Запись класса

**var**

Obj: TObject;

SaveLoader: IStreamPersist;

IsEmpty: Boolean;

**begin**

Obj:=GetObjectProp(self,PropList[i]);

**if** (Obj **is** TGraphic) **then** **begin**

IsEmpty:=TGraphic(Obj).Empty;

Stream.Write(IsEmpty,SizeOf(Boolean));

**end**;

**if** Supports(Obj,IStreamPersist,SaveLoader) **then** **begin**

SaveLoader.SaveToStream(Stream);

**end**;

**end**;

// Собственно сама процедура поиска свойств и записи

**begin**

TypeData:=GetTypeData(ClassInfo); // Получаем указатель на информацию

Count:=TypeData.PropCount; // Получаем количество свойств

**if** Count>**0** **then** **begin**

// Выделяем память для списка свойств

GetMem(PropList,SizeOf(PPropInfo)\*Count);

**Try**

// Получаем список свойств

GetPropInfos(ClassInfo,PropList);

// Перебираем все свойства из списка и сохраняем их

// в поток в соответствии с их типом

**for** i:=**0** **to** Count - **1** **do** **begin**

**case** PropList[i].PropType^.Kind **of**

tkEnumeration, tkInteger, tkChar, tkWChar: WriteOrdProp;

tkFloat: WriteFloatProp;

tkString, tkLString: WriteStringProp;

tkClass: WriteClassProp;

**end**;

**end**;

**finally**

// Освобождаем память

FreeMem(PropList,SizeOf(PPropInfo)\*Count);

**end**;

**end**;

**end**;

Я не буду подробно комментировать каждую функцию из TypInfo.pas, по комментариям сами разберетесь, прошу только обратить внимание на локальную процедуру записи класса. Сначала мы получаем экземпляр самого объекта. Потом проверяем, не является ли он потомком TGraphic. Далее записываем в поток, является ли графический объект пустым. Дело в том, что если объект (например Bitmap) пустой, то вызов SaveToStream не запишет в поток ничего. При чтении объект не сможет узнать о том, что он должен быть пустым, и будет, как ни в чем не бывало читать следующие по очереди данные из потока. Само собой это вызовет ошибку. Честно говоря, мне не очень нравится, как я решил эту проблему. Если у Вас есть идеи получше - пишите в обсуждении статьи.

И в конце процедуры WriteClassProp самое главное. Запрашиваем интерфейс IStreamPersist и заодно проверяем, поддерживает ли вообще его объект. Если да, то вызываем метод интерфейса SaveToStream.

Процесс чтения свойств из потока аналогичен. Я не буду его рассматривать в статье, в прилагаемом примере вы его найдете и сами сможете разобраться.

#### Что дальше?

Вы думаете это все? Кроме класса TRttiObject в прилагаемом файле Вы найдете следующие классы:

TRttiList = **class**(TObjectList, IStreamPersist)

Предназначен для составления и управления списками объектов типа TRttiObject. Кроме того, поддерживает интерфейс IStreamPersist и может сам являться свойством объекта TRttiObject. Таким образом, вы можете составлять из этих классов сколь угодно сложные структуры, массивы, деревья и т.д.

TAsyncRttiList = **class**(TRttiList)

Тоже самое, но процесс сохранения и загрузки позволяет выполнять асинхронно, т.е. сразу возвращает управление, а когда процессы завершены, оповещает главный поток (Thread).