**Java: Средства построения отчётов для Java-приложений**

Дмитрий Левиков

Это краткий обзор средств построения отчётов для Java-приложений. Я попытался получить демо-версию каждого из них и построить отчёт определеного вида, связав его с программой на языке Java. Если Вам известно что-либо помимо того, что я тут понаписал, напишите мне, ваши добавления будут кстати. И так...

**Enterprise Reports (TM) for Java**

Краткое описание:

Продукт полностью реализован на Java. Источниками данных могут быть JDBC-совместимые источники, Java-классы и EJB.

Возможности:

Extract Data - ERW может получать данные из традиционной базы данных или из приложения. ERW использует JDBC или ODBC для получения данных из БД. Для данных приложения, ERW предоставляет удобный механизм для определения структур классов приложения и получения данных через них. Предоставления данных из приложения быстрее, масштабируемее и идеально подходит для распределённых приложений.

Analyze Data - ERW предоставляет такие возможности как: много секционность, перекрестные таблицы (crosstabs), диаграммы/графики, формулы и Содержание (Table of Contents). ERW Report Engine позволяет строить отчеты в реальном времени

Format Data - ERW предоставляет простой в использовании дизайнер отчётов с широким набором средст форматирования как шрифты, цвета, выравнивание, перенос, разрыв страниц, перекрывающиеся компоненты и т.д.

Output Data - ERW может экспортировать любой отчёт в PDF, HTML, Hi-Res HTML, или ASCII-CSV

|  |
| --- |
| Цены  |
| Версия | Цена |
| ERW 4.0 Pro (Includes both AWT & Swing versions) | $495 |
| Подписка на год | $995 |
| Limited Free Runtime (100 копий) | Прилагается |

URL: http://www.enterprisesoft.com/Products/ReportWriter.html

Примечание:

Из опробованных продуктов Enterprise Reports имеет наиболее удобный дизайнер отчётов с очень богатым набором функций. Размещение и выравнивание компонентов сделано очень удобно. Да и сам по себе дизайнер красивый. Имеется API для встраивания в приложение. Поддерживается передача параметров. Текущая версия содержит много ошибок, поэтому часть заявленных функций проверить не удалось. Кроме того, имеются серьёзные проблемы с созданием русскоязычных отчётов. Сохранять шаблоны русскоязычных отчетов можно только с использованием UTF-8. Это само по себе нормально. Только для сохранения отчётов в различных форматах ребята используются конструкции типа:

if(mode == 1)

 dos.writeBytes(s); !!

else

 dos.writeUTF(s);

Так что сохранять шаблон отчёта в UTF-8 и печатать его на принтер ещё можно. Но вот записать его в формате PDF или HTML с русскими буквами не получится. Версия 4.02 является релизом, но вот только ошибок в ней столько, что хватит на пару альфа-версий. Нарисовать сложный отчёт в этом продукте запросто, а вот заставить его работать оказалось непросто.

Rreport 1.1

Возможности:

Предварительный просмотр с масштабированием

Используется дизайнер MS Access для построения отчётов и создания исходного кода на Java

Режим экспорта в HTML

Поддержка JDK 1.1 и 1.2

Часть исходных кодов доступна после регистрации

|  |
| --- |
| Цены  |
| Версия | Цена |
| RChart Binary version(Includes only source code of the applet) | $10 |
| Rchart with source code. (Includes source code of all classes) | $20 |
| Rreport. Binary version | $30 |
| Rreport with source code. (Includes source code of all classes) | $40 |

URL: http://rreport.port5.com/    http://rreport.8m.com

Примечание: Не смотря на привлекательные цены, в существующем виде продукт не подходит для профессиональных приложений. Использование MS Access в качестве дизайнера отчётов является удобным с точки зрения простоты создания форм. Можно рассмотреть возможность приобретения исходных кодов для их последующей переделки.

JClass Page Layout 4.5

JClass PageLayout является набором компонент реализующих функции просмотра отчётов и печати. Продукт предоставляет функции API для добавления текста, изображений и таблиц к документам.

Возможности:

Шаблоны страниц

Книжную и альбомную ориентация

Предопределённые стили таблиц

Интеграция с электронными таблицами

Встроенную поддержку диаграмм

Заголовки и подвалы для страниц

Автоматическую нумерацию страниц

Поддержка разбиения страницы на столбцы

Выравнивание текста, отступы и табуляции

Выделение цветом для страниц, разделов и текста

Рисование линий и библиотека форм

Импорт изображений из GIF, JPG и EPS

Предварительный просмотр

Печать на любой системный принтер (JDK 1.2 или выше)

Вывод в PostScript, PDF, HTML, или HP PCL 5

|  |
| --- |
| Цены  |
| Версия | Цена |
| PageLayout Bytecode (Includes 1 year GSS) 4.5 | $875 |
| Gold Support with Subscription (purchased separately) \*(Includes source code of all classes) | $325 |

URL: http://www.klgroup.com

Примечание:

Хотя продукт и поддерживает внешнее описание форм страниц, он не имеет дизайнера для выполнения этой работы. Подразумевается, что формы создаются в XML редакторе. Описание форм таким способом не является простым делом. Помимо XML c описанием отчёта ещё надо и DTD приложить.

JReport Professional 2.2

Возможности:

Поддерживается JDK 1.1, 1.2 и Microsoft Java VM

Построение сводных отчётов с диаграммами, многотабличных, подчиненных отчётов, поддержка функций вывода на принтер, HTML, PDF, CSV, e-mail, или в текст

Определяемые пользователем шаблоны, формулы и функции выхода

Доступ к JDBC или пользовательскому источнику данных

Catalog Browser - компонент используемый для построения отчёта, включающий источники данных, запросы, формулы, параметры и различные данные, и графические объекты

Report Inspector - свойства и значения для всех объектов в отчёте отображаются в таблице свойств и могу быть интерактивно изменены и отображены

|  |
| --- |
| Цены  |
| Версия | Цена |
| JReport Professional Report designer with 100 local Runtime | $995 |

URL: http://www.jinfonet.com/\_vti\_bin/shtml.exe/JReportRegister.htm

Примечание:

Исходя из описания, JReport Professional заслуживает серьёзного рассмотрения. Однако не удалось получить демонстрационную версию данного продукта. Если кто имеет хотя бы эвалюшен - буду признателен если поделитесь.

Elexir Report 1.2

Elixir Report - кросс платформенное Java решение для построения отчётов с использованием различных источников данных, включая XML. Elixir Report может использоваться для разработки отчётов и встраиваться в Java-приложения.

Возможности:

Различные типы источников данных - Java Objects и JDBC

Поддержка аплетов

Динамическая загрузка изображений

среда управления проектами с контролем версий

мастера построения отчётов и подключения к источникам данных

Бесплатный runtime

|  |
| --- |
| Цены  |
| Версия | Цена |
| Single Developer Licence | $149 |
| 5-Developer Licence Pack | $675 |
| 15-Developer Licence Pack | $1900 |
| 25-Developer Licence Pack | $2980 |
| \* Gold Support and Subscription per Developer (renewable annually) | $500 |

URL: http://www.elixirtech.com/ElixirReport/

Примечание:

Elixir Report - в целом произвел хорошее впечатление. Прилагаемый к нему дизайнер отчётов обладает необходимой функциональностью, хотя и уступает ERW 4.0.2. Поддерживается передача параметров для построения отчёта. Имеются мастера подключения к источникам данных и построители шаблонов отчётов. Нет проблем с построением русскоязычных отчётов. Сами шаблоны отчётов хранятся в XML. Кроме того есть встроенные средства управления проектом. Полная русификация runtime не возможна. В целом не очень красивый, но работающий продукт.

Root River Delta

Root River Delta - набор API для встраивания функций просмотра и печати отчётов в приложения. Поддерживает различные источники данных, включая JDBC, EJB и Java Objects. Дизайнера отчётов пока не имеет.

Возможности:

Внешний вид отчета полностью не зависит от формата его вывода. Поддерживается вывод отчётов в Bitmap, Java-совместимый принтер, PDF, HTML, PCL5, ASCII или CSV текст

Имеются средства просмотра отчётов для встраивания в приложения и аплеты

Реализация на AWT 1.1 обеспечивает максимальную переносимость

Поддерживаются различные шрифты и стили, изображения

API построено с целью упростить и ускорить построение больших по объему отчётов. Предусмотрена подгрузка данных в отчёт в процессе просмотра

|  |
| --- |
| Цены  |
| Версия | Цена |
| Рer developer (runtime components of RR Delta for internal use only) | $250 |

URL: http://www.rrsys.com/

Примечание:

Производит хорошее впечатление. Однако отсутствие дизайнера отчётов, не очень удобная и подробная документация, а так же некоторая сложность API требуют достаточно высокой квалификации для его использования. Root River Delta уступает аналогичным продуктам, таким как Style Report и JClass Page Layout.

Generic Report Writer

Поддерживает PostgreSQL, MySQL, и Access. Также совместим с любыми базами данных имеющими Type 4 JDBC драйвер. Реализован на Java 1.2. Распространяется в рамках лицензии GPL (доступны исходные коды). Имеется примитивный дизайнер отчетов. Отчёты можно получить только в текстовом формате. Шрифты и изображения не поддерживаются.

Цены: Free

URL: http://www.geocities.com/SiliconValley/Ridge/4280/GenericReportWriter/grwhome.html

Примечание:

... Ну что месье хотел за один @ ? Но в некоторых случаях может и подойти. Автор обещает развивать этот продукт.

Java Report Engine - i-net Crystal-Clear

i-net Crystal-ClearT это исполняющая среда для отчётов созданных в Seagate Crystal Reports Designer. Предназначен для добавления функций просмотра и печати отчётов в приложения и аплеты. i-net Crystal-ClearT может использоваться как платформно-независимое расширение для решений на базе Crystal Report для встраивания в приложения. Crystal-ClearT поддерживает форматы Seagate Crystal Reports 6.0, 7.0 и 8.0

|  |
| --- |
| Цены  |
| Версия | Цена |
| 20 User License | $200 |
| Enterprise License | $500 |
| Web-License | $700 |
| Application License | $1990 |

URL: http://www.inetsoftware.de/English/Produkte/CrystalClear/Default.htm

URL: http://www.seagatesoftware.com/products/crystalreports/default.asp

Примечание:

Основное достоинство данного продукта - поддержка Seagate Crystal Reports, который фактически может считаться промышленным стандартом для корпоративных систем. Хотя заявлено, что процессор i-net Crystal-ClearT работает быстрее оригинального Seagate Crystal Reports, это не заметно. Имеется красивый runtime, которым ещё и достаточно гибко управлять можно. Единтсвеное, что замечено плохого, свихивается на сложных SQL выражениях. Имеется про него и эха на news.inetsoftware.de/crystalclear. Короче, есть смысл обратить на него внимание.

Style Report Data Source Edition Lite

Style Report DataSource Edition - является расширением Style Report Lite 3.0 в который добавлены полнофункциональные средства для работы с различными источниками данных. Style Report Lite - представляет собой набор API с помощь которого можно добавлять функции просмотра и печати в свои приложения. Средства построения запросов и подключения к источникам данных в Style Report DataSource Edition имеются в дизайнере отчётов. Они также доступны и через вызовы API. Поддерживаются следующие типы источников данных:

XML

Реляционные БД (JDBC)

CORBA

Enterprise JavaBeans

Текст

Возможности:

SQL-подобный синтаксис запросов

Секций для расположения и фильтрации данных

Поддерживаются подзапросы

Имеется API интерфейс к управлению запросами

Поддерживается передача параметров в запросы

Печать из аплетов и в PDF

|  |
| --- |
| Цены  |
| Версия | Цена |
| Style Report/Lite | Free |
| Style Report/Pro | $995 |
| Style Report/Source | $3995 |
| Style Report/Enterprise (Include one development server runtime license) | $2980 |
| Runtime | Бесплатный для продуктов, не являющихся средствами разработки. |

URL: http://www.inetsoftcorp.com/

Примечание:

К сожалению, пришлось иметь дело только с бета-версией данного продукта, которую не удалось заставить работать. Однако бета-версия продукта Style Report Lite 3.0 заслуживает самого пристального внимания. Из всех протестированных средств построения отчётов, её в большей степени удалось русифицировать. Управление отчётом на уровне API позволяет добиться большоё гибкости и комбинировать самые разнообразные данные приложения. Поддерживаются шрифты, изображения, богатый набор стилей таблиц. Однако расплатой за это является необходимость программирования отчётов. Дизайнер отчётов больше ориентирован на построение разметки страницы, чем на описания секций отчета, что затрудняет построение отчетов с группировками. Такая обработка выносится на уровень программы, и элементы отчёта предполагается создавать из программы без помощи дизайнера. Версии Style Report DataSource Edition является гораздо более удобным средством для построения отчётов по базам данных, однако посмотреть её функциональность не удалось. В целом этот продукт является серьёзным претендентом на встраивание в приложения. Хотя требует достаточно высокой квалификации для создания отчета. К недостаткам, можно также отнести невозможность запустить выполнения отчета во фрейме приложения. Если нужно составить несколько встроенных в приложение отчётов StyleReport прекрасно для этого подойдёт.

Средства для построения отчётов, не включенные в тестирование

EspressChart.

EspressChartTM is a set of tools that enables you to easily include dynamic charts in your Java applications and on your Web pages. You can display the chart as a Java applet or PNG/GIF/JPEG image in your browser. (Note: EspressChart can be used in a servlet to generate PNG/GIF/JPEG images on the server dynamically.) It is written in 100% pure Java so it is completely platform independent. EspressChart supports JDBC/ODBC for retrieving and plotting data from any database.

URL: http://www.quadbase.com/espresschart/index.html

NetCharts.

NetCharts is a suite of Java applets that allows HTML authors, web developers, and Java programmers to quickly and easily create on-line charts from virtually any data source, with minimal coding. NetCharts supports any Java 1.0.x or 1.1.x enabled platform, including PCs, UNIX and network computers. All applets can be viewed through any Java compatible Web browser and support the use of the JDBC interface to retrieve parameter definitions or data values from one or more database servers or data sources, including Oracle, Sybase, Informix, DB2, Microsoft SQL Server, dBase, Access and Excel.

URL: http://www.cartworks.com

Java: Конфигурация программ.

Содержание:

Зачем нужно конфигурирование?

Что именно стоит настраивать.

Способы хранения настроек.

Ini-файлы.

Файлы Properties.

XML-файлы.

Сериализация.

Базы данных.

Скрипты.

Пример программы с конфигурацией в XML.

**Зачем нужно конфигурирование?**

Профессиональным программистам этот вопрос покажется странным. У начинающих же часто наблюдается явное непонимание важности этой возможности. При этом получается программа, похожая на каменную глыбу с высечеными на ней надписями - если захочется изменить надпись, то придётся делать новый камень.

Есть и другая крайность - когда практически всё выносится в настройки. Такие программы напоминают разлитую жидкость, а чтобы заставить её работать надо прочитать талмуд описания и настроить несколько сотен параметров, к тому же часто взаимосвязанных противоестественным образом.

Как всегда нужно найти золотую середину - с одной стороны надо постараться удовлетворить различные прихоти пользователей, с другой стороны нужно сделать так, чтобы большинству пользователей ничего настраивать не пришлось.

**Что именно стоит настраивать.**

Вот типичные примеры данных, которые часто стоит вынести в настройки:

Всевозможные каталоги. Например - пути до файлов данных, каталоги импорта/экспорта.

Сетевые настройки. Имена серверов, IP-адреса, порты, имена и пароли для автоматического доступа.

Настройки баз данных. Имена JDBC-драйверов, URL базы данных, SQL-запросы, зависимые от используемой БД.

Настройки внешнего вида. Настройки Swing-овского Look & Feel-а, используемые шрифты, размеры, цвета, настройки горячих клавиш.

Прочее... Любые другие вещи, которые могут менятся от пользователя к пользователю.

Например, довольно часто встречаемая ситуация - настройка соединения с БД. Начинающие программисты часто пишут нечто подобное:

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection con = DriverManager.getConnection("jdbc:odbc:MyDatabase",user,password);

Таким образом программа привязывается к конкретному JDBC драйверу. Использовать другой драйвер, например заменить мост на RMI-прокси или, в случае Oracle, OCI на Thin без перекомпиляции уже нельзя.

**Способы хранения настроек.**

В объектном программировании всё представляется в виде объектов. Настройки лучше всего при этом рассматривать как свойства определённых объектов, которые хранятся в файлах конфигураций. То, каким образом эти настройки считываются и записываются тесно взаимосвязано с форматом файлов и выбраной стратегией администрирования. Рассмотрим идеальный вариант:

Настраиваемый объект не должен содержать знаний о формате файлов и способе чтения/записи. Это позволило бы, в случае необходимости, заменить один способ другим.

Большинство настроек должны выполняться при помощи программы (подпункт меню или отдельная программа настройки). Это сильно облегчает жизнь человека, который занимается администрированием. У большинства "юниксоидов" это может вызвать непонимание :-), но редактированием текстовых файлов в современном мире во многих случаях не обойтись.

Должно быть установлено разумное умолчание для отсутствующих параметров. Другими словами - необходимо, чтобы большинству пользователей для запуска программы нужно было бы сделать минимум настроек. Как правило это оставляет благоприятное первое впечатление о программе, а часто именно оно - самое важное.

К сожалению этот идеальный вариант довольно трудно сделать на практике. Первое требование предполагает разработку универсального механизма сохранения объектов. Такие системы уже есть готовые, но часто они не подходят по тем или иным параметрам. Разработать же самому такую систему - далеко не каждому под силу.

Второе требование подразумевает, что для каждого объекта пишется своя панель (или диалог) для редактирования настроек. В случае большого количества объектов стоит попробовать использовать универсальные механизмы. Один из вариантов - использование стандарта JavaBeans. Этот стандарт разрабатывался для визуальных систем программирования, но, из-за сходства решаемых задач, также хорошо подходит для универсального конфигурирования. Но это тоже не самая простая задача, поэтому часто разумно предусмотреть возможность альтернативного варианта конфигурирования для пожарных случаев - например, при помощи обычных текстовых редакторов в случае использования текстовых форматов файлов.

Разумное же умолчание для параметров часто просто невозможно представить. Например, что поставить в качестве имени SMTP-сервера? В случае Unix-систем можно попробовать поставить localhost, но для Windows-мира это редко кому подойдёт.

Рассмотрим наиболее распространённые варианты:

Ini-файлы.

Ini-файлы - это был самый распространённый вариант в эпоху Windows 3.x. Сейчас в виндовых программах он стал вытесняться хранением настроек в реестре. Тем не менее ini - это один из простейших вариантов хранения настроек. К сожалению довольно часто эта простота заставляет прибегать к различно рода ухищрениям. Пример типичного ini-файла:

[Communication]

InputDir=INPUT

OutputDir=OUTPUT

ArchDir=ARHIV

TransferPath = a:\cour

[Warning]

NoReceived=No

[Addons]

Numb = 3

MenuName1 = ~N~orton

ProgName1 = mousesav c:\command.com /c nc

MenuName2 = Win - ~Б~локнот

ProgName2 = notepad

MenuName3 = Импорт из формата АБ "Инкомбанк"

ProgName3 = incom.bat

В Java нет стандартного класса для чтения ini-файлов, но это не проблема. Т.к. формат очень прост, его легко сделать самому:

import java.io.\*;

import java.util.\*;

public class INIFile

{

 Properties iniProperty = new Properties();

 public INIFile(File f) { this( f.getPath() ); }

 public INIFile(String fname) throws IOException { loadFile( fname ); }

 private void loadFile( String fname ) throws IOException

 {

 BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(fname)));

 try

 {

 String section = "";

 String line;

 while( (line = br.readLine())!=null )

 {

 if( line.startsWith(";") ) continue;

 if( line.startsWith("[") )

 {

 section = line.substring(1,line.lastIndexOf("]")).trim();

 continue;

 }

 addProperty(section,line);

 }

 }

 finally { br.close(); }

 }

 private void addProperty(String section,String line)

 {

 int equalIndex = line.indexOf("=");

 if( equalIndex > 0 )

 {

 String name = section+'.'+line.substring(0,equalIndex).trim();

 String value = line.substring(equalIndex+1).trim();

 iniProperty.put(name,value);

 }

 }

 public String getProperty(String section,String var,String def)

 {

 return iniProperty.getProperty(section+'.'+var,def);

 }

 public int getProperty(String section,String var,int def)

 {

 String sval = getProperty(section,var,Integer.toString(def));

 return Integer.decode(sval).intValue();

 }

 public boolean getProperty(String section,String var,boolean def)

 {

 String sval = getProperty(section,var,def ? "True":"False");

 return sval.equalsIgnoreCase("Yes") || sval.equalsIgnoreCase("True");

 }

}

Файлы Properties.

Этот формат распространён в Unix-мире. Он ещё проще ini-файлов, т.к. в нём отсутствует понятие секций - всё состоит из ключей и значений. Пример типичного файла:

# Database configuration

Database.Driver=sun.jdbc.odbc.JdbcOdbcDriver

Database.DataURL=jdbc:odbc:MyDatabase

Database.Prop.user=user

Database.Prop.password=password

В Java есть готовый класс для чтения/записи таких файлов (java.util.Properties), но с ним есть некоторые проблемы. Во первых для чтения невозможно задать кодировку файла, а это означает проблемы с русскими буквами. Во вторых стандартная функция записи сохраняет данные в порядке следования хэш-значений ключей, что значит - как ей больше понравится. Но это тоже легко разрешимо - достаточно написать свою читалку/писалку.

XML-файлы.

Этот формат подходит для многих целей, в том числе и для хранения настроек. XML-формат ориентирован на древовидные структуры, что довольно естественым образом отображается на объекты. Пример типичного файла:

<?xml version="1.0" encoding="Windows-1251"?>

<!-- Database configuration -->

<database

 driver="sun.jdbc.odbc.JdbcOdbcDriver"

 dataURL="jdbc:odbc:MyDatabase">

 <prop name="user">user</prop>

 <prop name="password">password</prop>

</database>

Для чтения и записи таких файлов предназначены специальные библиотеки - так называемые XML-парсеры. Таких парсеров уже сделано довольно много, так что писать его самому нет большого смысла - достаточно лишь подобрать подходящий. Для парсеров было разработано два стандартных программных интерфейса - событийный (SAX) и иерархический (DOM). Есть также и парсеры со своим интерфейсом. Размер jar-а с парсером может варьироваться от нескольких килобайт до мегабайта - в зависимости от поддерживаемых интерфейсов и возможностей.

Для XML также написано несколько библиотек для универсального сохранения (сериализации) объектов в файлах XML. Такие библиотеки позволяют отделить алгоритм сохранения от самого объекта, а это, как уже упоминалось, имеет много плюсов.

Сериализация.

Под термином "сериализация" понимают запись содержимого объекта в поток двоичных данных. Обычно имеется в виду универсальный алгоритм, реализуемый классами java.io.ObjectOutputStream и java.io.ObjectInputStream. Пользоваться ими просто настолько, насколько это вообще возможно - обычно достаточно лишь отметить в классе поддержку при помощи интерфейса Serializable и отметить ключевым словом transient те поля объекта, которые сохранять не нужно. Собсно и всё. :-) Пример:

public class SerialObject implements java.io.Serializable

{

 private String name;

 private transient int state;

 public SerialObject() {}

 public SerialObject(String n) { name = n; }

 public String getName() { return name; }

 public void setState(int s) { state = s; }

}

Запись объектов:

SerialObject o = ...;

OutputStream os = ...;

ObjectOutputStream oos = new ObjectOutputStream(os);

oos.writeObject(o);

Чтение объектов:

InputStream is = ...;

ObjectInputStream ois = new ObjectInputStream(is);

SerialObject o = (SerialObject)ois.readObject();

Использование сериализации - это один из самых простых вариантов по реализации, но и у него есть свои недостатки. Получаемые файлы являются двоичными, а значит в текстовом редакторе их уже не подправить - придётся делать редактирование параметров из программы. Кроме того, необходимо следить за изменением сохраняемых объектов, дабы не нарушить совместимость при изменении и развитии программы.

Базы данных.

В базах данных можно хранить любые данные, конфигурация программы - не исключение. Это имеет смысл в нескольких случаях:

Настройки связаны весьма сложным образом и древовидные структуры типа XML подходят плохо.

Доступ к настройкам должен быть только у авторизованых пользователей.

Доступ к этим данным должен быть и из других программ, например из генератора отчётов типа Crystal Reports.

БД могут применятся объектные или реляционные. Другие типы сейчас широкого распространения не имеют. Использовать хорошую объектную БД часто так же просто, как и сериализацию. Для реляционых баз можно применить объектную надстройку, которая также позволяет сильно упростить жизнь. Ну а можно делать обычные SELECT-ы.

Скрипты.

Использование скриптов - это один из самых экстремальных способов конфигурирования. Они позволяют добится максимальной гибкости программы за счёт вынесения логики наружу. В использовании скриптов надо тоже знать меру - в конце концов заказчик платит Вам за программу, решающую задачи, а не за ещё один интерпретатор или компилятор за который ему потребуется посадить ещё одного программиста. А то получается, как в том анекдоте - какую программу не начнёшь писать, всё компилятор получается.

Но часто без скриптов действительно тяжело. Типичные примеры - алгоритмы импорта/экспорта, алгоритмы проверок данных. Вы можете подготовить стандартный набор, а дальше настраивать скриптами под конкретные требования заказчика.

Для программ на Java в качестве скрипт-языка хорошо использовать язык Python в его Java-инкарнации под названием JPython. Там легко организовать двусторонюю связь между программой и скриптом. Если не будет хватать скорости интерпретации, то код на Python-е можно скомпилировать в байт-код - получится обычный Java-класс. Про JPython можно почитать на сайте www.jpython.org или в новой книжке Брюса Эккеля Thinking In Patterns with Java (доступна на www.bruceeckel.com).

**Пример программы с конфигурацией в XML.**

В качестве примера можете посмотреть простенькую программы, использующей XML-файл в качестве конфигурационного. Сохраняемые параметры можно редактировать как из программы, так при помощи текстового редактора.

XMLConfig.java

Пример содержимого конфигурационного файла:

<?xml version="1.0" encoding="Cp1251"?>

<program color="ff0000ff" bounds="263 231 392 195">Просто строчка

Вторая строчка</program>

В качестве XML-парсера используется Sun-овский парсер в режиме DOM. На таком простом примере не видно особых преимуществ формата XML над теми же файлами properties. Они становятся заметны только в достаточно сложных программах, где становится необходимо хранить списки однотипных параметров или же содержимое объектов с уровнем вложенности два или более.

Pascal: Работа с файлами

В прошлом выпуске мы с вами начали писать программу, которая и должна была стать основополагающей в наших дальнейших действиях. Программа называлась "Записная книжка", манипулировала с типом record и еще содержала маленькую ошибочку, которую, кстати, многие заметили.

Обратите внимание на этот код:

|  |
| --- |
| ......  |
| C := ReadKey;  |
| case C of  |
| '1'..'9': begin  |
| Val(C, I, Code);  |
| List\_Item(I);  |
| end;  |
| 'a': New\_Item;  |
| #27: Quit := true;  |
| end;  |
| ...... |

В этом вся собака зарыта. Дело в том, что в условии упоминается о работе с 10-ю записями, а этот алгоритм позволяет использовать всего девять, так как вообще не умеет читать двухзначные числа. Как решить этот вопрос - Ваша задача! Программирование вообще без задач, которые нужно решать, не обходиться...

Ну а мы интенсивно двигаемся дальше. Сегодня - как и было обещано, файлы. Пока только теория, которую вам нужно усвоить (ее довольно много), практика будет в следующем выпуске.

Работа с файлами

В паскале работа с файлами осуществляется через специальные типы, доселе нам не известные. Это файловые типы, которые определяют тип файла, то есть фактически указывают его содержимое. С помощью этой переменной, которой присвоен необходимый тип, и осуществляется вся работа с файлами - открытие, запись, чтение, закрытие и т.п.

При работе с файлами существует определенный порядок действий, которого необходимо придерживаться. Вот все эти действия:

Создание (описание) файловой переменной;

Связывание этой переменной с конкретным файлом на диске или с устройством ввода-вывода (экран, клавиатура, принтер и т.п.);

Открытие файла для записи либо чтения;

Действия с файлом: чтение либо запись;

Закрытие файла.

Первое, на что хочу обратить внимание, это возможность связать файловую переменную не только с физическим файлом на носителе информации, но и с устройством. В качестве такового используются обычные псевдонимы устройств DOS. Вот основные два:

CON - консоль (экран-клавиатура), то есть по записи в это устройство мы будем получать информацию на экран, при чтении информации из этого устройства, будем читать данные с клавиатуры.

PRN - принтер. При записи в это устройство вы получите информацию на принтер.

Далее хочу обратить внимание на последний этап - закрытие файла. В принципе, не обязательное условие для файлов, из которых мы читаем данные. Если не закроем - ошибки это не вызовет, последствий тоже. Однако обязательно закрывать файл, если мы осуществляли в него запись. Дело в том, что если мы пишем данные в файл на диске и забываем его закрыть - информация не сохраниться. Она (информация) помещается во временный буфер, который запишется на диск только при закрытии файла.

Типы файловых переменных

Перед тем, как начинать работу с файлами, давайте посмотрим, какие существуют переменные для работы с ними. В Turbo Pascal имеется три типа таких переменных, которые определяют тип файла. Вот эти типы:

Text - текстовый файл. Из переменной такого типа мы сможем читать строки и символы.

File of \_любой\_тип\_ - так называемые "типизированные" файлы, то есть файлы, имеющие тип. Этот тип определяет, какого рода информация содержится в файле и задается в параметре\_любой\_тип\_. К примеру, если мы напишем так:

F: File of Integer;

То Паскаль будет считать, что файл F содержит числа типа Integer; Соответсвенно, читать из такого файла мы сможем только переменные типа Integer, ровно как и писать. Напишем так:

type

A = record

I,J: Integer;

S: String[20];

end;

var

F: File of A;

То получим очень интересную картину, когда файл содержит в качестве элементов записи. Мы сможем их читать, писать и производить с ними действия. Именно этим мы и займемся в процессе модификации нашей программы "Записная книжка".

File - нетипизированный файл. Когда мы указываем в качестве типа файла просто File, то есть без типа:

F: File;

То получаем "нетипизированный" файл, чтение и запись в который отличается от работы с файлами других типов. Эти действия производятся путем указания количества байт, которые нужно прочитать, а также указанием области памяти, в которую нужно прочитать эти данные. Это тема будущих выпусков.

Итак, разобрались с типами файлов. Теперь давайте по порядку разбирать действия, применяемые для работы с файлами. Я говорил о них выше.

Связывание переменной с файлом

Самый универсальный шаг. Выполняется одной и той же процедурой для всех типов файлов, а именно процедурой Assign:

Assign(\_переменная\_файлового\_типа\_, 'путь к файлу');

Как видите, в качестве параметров задаются переменная либого файлового типа и строка - путь к файлу, который, по правилам DOS, не может быть длиннее 79 символов. Вот пример использования assign:

|  |
| --- |
| var  |
| T: Text;  |
| F1: File of Byte;  |
| F2: File;  |
| begin  |
| Assign(T, '1.txt');  |
| Assign(F1, 'C:ProgrammTp2.txt');  |
| Assign(F2, 'F:usr\_inapacheconfhttpd~1.con');  |
| .......... |

Как видите, это очень просто и сама процедура Assign проблем не вызывает. Давайте теперь рассмотрим следующий этап.

Открытие файла

Открытие файла - это уже более усложненный процесс, нежели связывание с ним переменной. Здесь учитывается, зачем открывается файл - для записи или чтения, а также в зависимости от типа файла процедуры выполняют различные действия.

Тем не менее этот процесс не сложен и заключается в использовании одной из трех имеющихся процедур:

Reset(\_любая\_файловая\_переменная\_);

Открыват файл на чтение. В качестве параметра - файловая переменная любого из перечисленных выше типов. Это может быть текстовый, типизированный либо не типизированный файл. В случае с текстовым файлом, он открывается только на чтение. В случае с типизированным и нетипизированным файлом - он открывается на чтение и запись.

Append(T: Text);

Эта процедура открывает текстовый файл (только текстовый!) на запись. Выше я сказал, что Reset при задании параметра типа Text не позволит писать в него данные, открыв файл лишь для чтения. То есть если вы используете текстовый файл и хотите производить в него запись, нужнo использовать Append. Если чтение - Reset. В остальных случаях дело обходиться одной процедурой Reset.

Также обратите внимание, что если вы до этого уже открыли файл на чтение, вам не нужно закрывать его и открывать снова на запись. В этом случае файл закрывается сам и открывается заново. При записи данных в файл при открытии его с помощью этой процедуры они записываются в конец файла.

ReWrite(F) - создает новый файл либо перезаписывает существующий. Будьте осторожны, если не хотите случайно удалить нужный файл. Напомню, файл, открытый с помощью этой процедуры будет полностью перезаписан.

В использовании процедур, думаю, проблем не будет. Однако возникает другой вопрос - что, если файла, который открывается, нет? Если он не существует на диске, то как мы сможем из него читать информацию? Программа выбьет ошибку в такой ситуации. Это реальная проблема, которая, кстати, очень просто решается.

Способ проверки заключается в двух этапах: использовании ключей компилятора и функции IOResult, которая возвращает значение от только что выполненной операции ввода-вывода. С функцией разберемся быстро, а вот с такой штукой как ключи компилятора мы еще не сталкивались, поэтому остановимся подробнее.

Ключи компилятора - это обыкновенные переключатели, которые контролируют ход выполнения программы исключая или включая реакцию на какие-нибудь условия. В нашем случае нас интересует условие, когда физически отсутствует нужный нам файл, либо не удалось открыть его по другим причинам. Ключей у Паскаля довольно много, мы пока изучим один, необходимый нам на данный момент.

Оформляются ключи следующим образом: в скобках комментариев "{}" первым символом после открывающей скобки "{" ставиться знак доллара "$", после чего указывается имя ключа и его значение. Нас интересует ключ, который выключает вывод ошибок ввода-вывода, называется он "I". Выглядит все это следующим образом:

{$I+} - включение вывода ошибок

{$I-} - выключение вывода ошибок

По сути дела отсутствие файла - это ошибка, которая возвращается функцией IOResult. Если же эта функция возвращает 0, то файл успешно открыт, без ошибок. Вот и вырисовывается последовательность действий, необходимых для проверки на наличие файла:

Связываем переменную с файлом;

Выключаем вывод ошибок на экран - {$I-}

Открываем файл необходимой нам процедурой;

Включаем вывод ошибок {$I+} - пусть будет для дальнейшего отслеживания таковых;

Проверяем, если IOResult возвращает нуль, то все было путем и файл открыт. Иначе выводим ошибку.

Вот пример такой программы:

|  |
| --- |
| var  |
| T: Text;  |
| S: String;  |
| begin  |
| Write('Enter filename: ');  |
| Readln(S);  |
| Assign(T, S);  |
| {$I-}  |
| Reset(T); { открываем файл для чтения }  |
| {$I+}  |
| if IOResult <> 0 then { если не нуль, то была ошибка }  |
| begin  |
| Write('Error when open file!');  |
| Halt;  |
| end;  |
| { иначе все в порядке, продолжаем }  |
| ..........  |
| end. |

Закрытие файла

Выше я говорил о том, зачем нужно закрывать файл и когда надо это делать. Закрытие файла производиться с помощью процедуры Close(F), где F - это переменная файлового типа. Эта процедура одна для всех типов файлов.

Запись и чтение файлов. Часть I

Сегодня я хочу рассказать о записи и чтении текствых и типизированных файлов, в следующем выпуске рассказ пойдет о чтении файлов без типа. Итак, переходим к непосредственной обработке файловой информации.

Чтение файлов. Чтение файлов производится с помощью отлично известных нам процедур Read и Readln. Они используются также, как и при чтении информации с клавитуры. Отличие лишь в том, что перед переменной, в которую помещается считанное значение, указывается переменная файлового типа (дескриптор файла):

Read(F, C);

Здесь F - дескриптор файла, C - переменная (Char, String - для текстовых, любого типа - для типизированных файлов).

Также сразу хочу упомянуть о одной, пожалуй самой главной функции при чтении файлов. Это функция поверки на конец файла - Eof(F): Boolean;. В качестве параметра - файловая переменная любого типа. Функция возвращает TRUE если достигнут конец файла и FALSE иначе. Здесь все очень просто, демонстрации ради давайте напишем небольшую программку. Пусть имеем текстовый файл. Давайте его распечатаем и заодно посчитаем, например, количество пробелов:

|  |
| --- |
| var  |
| T: Text;  |
| С: Char;  |
| Spaces: Word;  |
| S: String[79]; { 79-макс. длина пути в DOS }  |
| begin  |
| Write('Enter filename: ');  |
| Readln(S);  |
| Assign(T, S);  |
| { открываем файл для чтения }  |
| {$I-}  |
| Reset(T);  |
| {$I+}  |
| { если не нуль, то была ошибка }  |
| if IOResult <> 0 then  |
| begin  |
| Write('Error when open file ', S, ' !');  |
| Halt;  |
| end;  |
| { иначе все в порядке, продолжаем }  |
| { ЦИКЛ: пока НЕ КОНЕЦ ФАЙЛА }  |
| While (not Eof(T)) do  |
| begin  |
| { читаем из файла переменную }  |
| Read(T, C);  |
| { если пробел, увеличиваем счетчик }  |
| If C = ' ' then Inc(Spaces);  |
| Write(C);  |
| end;  |
| Writeln('КОЛИЧЕСТВО ПРОБЕЛОВ: ', Spaces);  |
| Readln;  |
| end. |

Думаю, здесь все ясно. Продолжаем двигаться дальше и посмотрим, как производиться запись в файлы.

Запись в файлы. Вы еще не догадались? Запись в файлы производиться точно так же, как и запись на экран - с помощью процедур Write и Writeln. Как и в случае с чтением, перед записываемой в файл переменной указывается тескриптор файла:

Write(F, S);

Здесь F - дескриптор, S - переменная.

При этом, естественно, переменная должна соответствовать типу файла. Примера ради давайте составим еще одну небольшую программку, которая покажет работу с файлами. На сей раз уже используем типизированные файлы, а именно состоящие из чисел. Итак, мы имеем файл, в котором содержаться числа типа Integer. Давайте отсортируем эти числа в файле по возрастанию.

План дейтсвий:

Отрываем типизированный файл из Integer; (проверяем на ошибку и т.п.)

Читаем все числа в массив (считываем, пока не конец файла)

Сортируем массив по возрастанию;

Записываем отсортированный массив обратно в файл.

Получается такая программа:

|  |
| --- |
| Program Sorting;  |
| uses Crt;  |
| var  |
| F: File of Integer;  |
| I, J, M: Word;  |
| Mas: Array[1..500] of Integer;  |
| S: String;  |
| begin  |
| ClrScr;  |
| Write('Enter filename: ');  |
| Readln(S);  |
| { открываем файл }  |
| Assign(F, S);  |
| {$I-}  |
| Reset(F);  |
| {$I+}  |
| if IOResult <> 0 then  |
| begin  |
| Write('Error when open file!');  |
| Halt;  |
| end;  |
| { пока не конец файла, читаем массив }  |
| While (not Eof(F)) do  |
| begin  |
| Inc(M);  |
| Read(F, Mas[M]);  |
| Write(Mas[M], ' ');  |
| end;  |
| { сортируем массив по возрастанию }  |
| For I := 1 to M do  |
| For j := 1 to M do  |
| if Mas[I] < Mas[J] then  |
| begin  |
| inc(mas[j], mas[i]);  |
| mas[i] := mas[j]-mas[i];  |
| dec(mas[j], mas[i]);  |
| end;  |
| Writeln; Writeln('=============================================');  |
| { перезаписываем файл }  |
| ReWrite(F);  |
| For I := 1 to 100 do  |
| begin  |
| Write(Mas[I], ' ');  |
| Write(F, Mas[i]);  |
| end;  |
| Writeln; Write('Elements in file: ', M);  |
| Close(F);  |
| Readln;  |
| end. |

Программа очень проста и хорошо демонстрирует работу с типизированными файлами. В качестве сортировки массива я использую метод пузырька, чтобы перезаписать файл использую ReWrite. Вроде не должно возникать никаких сложностей... Будут проблемы, пишите. Ну а на сегодня, пожалуй, все. Этой информации вам будет достаточно, в следующем выпуске займемся, как я уже сказал, нетипизированными файлами и продолжим писать программу "Записная книжка".