Rebol Руководство программиста

Перевод *pochinok@bk.ru*

Содержание:

1. Предисловие

На протяжении многих лет ответ на вопрос "Какая книга лучшая для изучения Rebol?" был "Реболом программирования" Оливье Оверло. Как вы, наверное, догадались по названию, он написан на французском языке. Вскоре после того, как он был опубликован в конце 2001 года, началась работа над переводом. Он даже был указан с датой публика- ции на Amazon.com. К сожалению, он еще не увидел свет. В начале 2007 года Оливье опубликовал "Rebol - Guide du programmeur". Я надеялся на новости о переводе; но никто не при- шел. Казалось, единственный вариант - прочитать книгу на француз- ском. Так почему бы не перевести книгу самому? Очень приятно было перевести "Rebol - Guide du programmeur". Письмо Оливье красноречиво и информативно. Его команда Ребола просвечивает. Я многому научился, переводя книгу, немного Ребола и немного французского. Я надеюсь, что вы тоже, хотя, возможно, не столько француз, как я.

2. Благодарности

Я хотел бы поблагодарить Оливье не только за то, что он написал такую хорошую книгу, но и в равной степени за всю помощь и поддержку, ко- торые он оказал, когда я работал над переводом. В сообществе Rebol есть много замечательных людей. В частности, Грегг Ирвин (Gregg Irwin) и Сунанда (Sunanda) были постоянным источником дружбы и поддержки. Спасибо. И последнее, но не менее важное: я хотел бы поблагодарить Нориати, мою жену, и Ханну Сару, нашу дочь, за их любовь, безоговорочную поддержку и терпение в течение многих часов, которые я сидел, глядя в свой компьютер. Peter W A Wood

3. Предисловие

3.1 Какова цель этой книги?

В течение пяти лет Оливье имел удовольствие работать во французском компьютерном журнале Login и написал много статей о REBOL. К сожалению, Login больше не публикуется, и Оливье счел необходимым и целесообразным консолидировать свою работу, чтобы накопленные знания попрежнему были доступны сообществу Rebol. Понятно, что изначально он опубликовал свою работу как "Rebol - Guide de Programmeur". Эта книга является переводом оригинала на английский язык. Основная идея заключалась в том, чтобы просто перегруппировать статьи по определенной тематике, но проект оказался намного масштабнее. Многие статьи требовали частичного переписывания, другие - дополнений и, наконец, были добавлены некоторые ранее неопубликованные элементы. "Ремикс" был намного больше, чем предполагалось, но необходимо было обеспечить целостную структуру книги. В итоге книга оказалась весьма практичным руководством по Rebol для программистов.

Разнообразие тем делает книгу скорее справочником, чем полным вве- дением. Он развивает ряд моментов из предыдущей книги Оливье "Про- граммирование Rebol", опубликованной Eyrolles (ISBN: 2-212-11017-0).

3.2 Для кого эта книга?

Эта книга в первую очередь предназначена для разработчиков Rebol. Он был задуман и разработан с учётом их потребностей, давая максимум знаний по широкому кругу тем в краткой форме. Что наиболее важно, это может сэкономить им много времени благодаря многочисленным представленным примерам. Он также отвечает потребностям разработчиков и студентов, желающих изучить Rebol. Каждая глава объясняет сильные стороны языка, а затем применяет полученные знания. Кроме того, он отвечает потребностям некоторых, описывая возможности технологии Rebol. Для изучения реализации языка с помощью примеров используются различные тематические исследования и презентации продуктов. Наконец, он предназначен для системных администраторов, заинтересованных в использовании Rebol для автоматизации определенных задач (администрирование Unix-серверов, управление гридвычислениями и т. д.). Этим темам посвящены многие разделы этой книги.

3.3 Как это представленно?

Книга состоит из семи глав, каждая из которых построена вокруг одной темы:

- **Глава 1** "Откройте для себя Ребол за час" представляет собой обзор Rebol на примерах,
- **Глава 2** "Язык Ребола" посвящена основам языка,
- **Глава 3** "Графический интерфейс, графика и звук" рассматривает графические интерфейсы, графика и звук.
- **Глава 4** "Сеть и Интернет" описывает расширенные возможности сетевого программирования Rebol.
- **Глава 5** "Ребол для профессионалов" предназначена для профессиональных пользователей Rebol. Он представляет использование Rebol в контексте электронного бизнеса,
- **Глава 6** "Ребол для гиков" предоставляет расширенную информацию для разработчиков Rebol,
- **Глава 7** "Практическое применение" представляет собой серию семинаров, предназначенных для облегчения изучения язык через практику.

4. Вступление

Rebol (Relative Expression-Based Object Language) это работа Карла Сассенрата, который был одним из основных создателей известной операционной системы AmigaOS. В 1995 году он решил работать над новым языком программирования, и его усилия увенчались успехом в 1997 году, когда появилась версия 1.0 Rebol.

Целью Карла было разработать язык, который будет простым в изучении, многоплатформенным и полностью адаптированным для обмена информацией между разнородными системами. Таким образом, язык очень эффективен в области сетевого программирования. Он поддерживает основные сетевые протоколы и позволяет легко управлять сокетами. Rebol - это амбициозная концепция, реализованная в ряде продуктов.

4.1 Присоединяйся к REBOLюции !

Идея, лежащий в основе концепции Rebol, основан на простом, но убедительном наблюдении о том, что с ростом важности компьютеров, взаимодействующих друг с другом, заключается в том, что программистам требуется новое поколение более выразительных языков с синтаксисом, близким к человеческому, и способностью использовать стандартные протоколы для обмена информацией через TCP / IP. Современное программное обеспечение должно разрабатываться не с учетом картины одного компьютера, а с учетом сети взаимосвязанных систем для обмена данными и программами. Подобно Python, Ruby и некоторым аспектам решений Java и .NET, Rebol является амбициозным инновационным языком сценариев. Это гораздо больше, чем простой язык программирования, его можно назвать метаязыком, сердцем устройства, предназначенного для обмена информацией по сетям.

4.1.1 Программист, конструктор слов

В общепринятом смысле Rebol - это интерпретируемый, а не компилируемый язык. Код не преобразуется в исполняемый двоичный или даже байт-код, как Java. Фактически, Rebol - это оценщик, подобный Lisp и Scheme. Они создают словарный запас из сценария и интерпретируют слова, которые вы задали, а также слова, принадлежащие стандартному словарю языка. Такой режим работы делает Rebol метаязыком: то есть языком, способным описывать, переопределять и обогащать свои собственные внутренние механизмы.
В Rebol все сводится к словам. Инструкции, переменные, функции и объекты - все это слова. Некоторые просто предоставляют определенные функции. Все эти слова принадлежат либо общему контексту (глобальному контексту), либо конкретному контексту. Как и в разговорной речи, слово может иметь особое значение в зависимости от контекста, в котором оно оценивается.
Слова также могут быть определены на диалекте, который является своего рода мини-языком, который в наши дни часто называют языком предметной области или DSL. Диалект - это язык внутри языка, предназначенный для выполнения очень специальной задачи. Чтобы лучше понимать диалекты, мы можем взять в качестве примера инженерию. Когда инженеры говорят о своей профессии, человеку, не относящемуся к их дисциплине, часто бывает очень трудно понять смысл разговора. По общему признанию, эти инженеры говорят по русски, но обогащают его техническими терминами, которые образуют диалект, известный только между ними! В сценарии Rebol диалекты позволяют описывать определенные части приложения с помощью специального словаря, которым может управлять не только программист, но и сторонний наблюдатель, который не обязательно является разработчиком. С помощью диалектов мы можем определять бизнес-правила, графические интерфейсы или последовательность экранов во время установки программы.

4.1.2 Виртуальная машина

Оценщик фактически представляет собой виртуальную машину, которая имитирует идентичную среду независимо от базовой операционной системы и архитектуры оборудования. В отличие от Java, он очень легкий и весит от 250 до 400 КБ в зависимости от версии. Его также очень легко установить, поскольку он состоит из одного исполняемого файла. Нет! вы не мечтаете, виртуальная машина Rebol целиком содержится в одном файле. Нет никаких DLL или библиотек для установки какой бы то ни было операционной системы. За несколько секунд любой компьютер можно превратить в машину, способную использовать ваши приложения, написанные на Rebol.

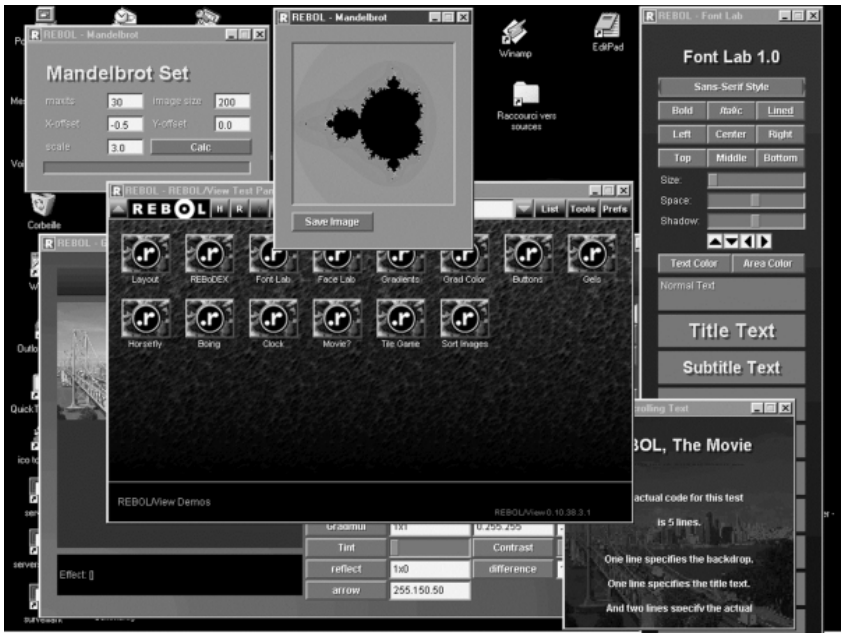

Рисунок 0-1. *Запуск приложения Rebol.*

Эта виртуальная машина содержит оценщик Rebol, стандартный словарь, поддержку TCP / IP, сборщик мусора и диспетчер безопасности.

4.1.3 Семейство продуктов

Rebol выпускается в шести версиях, нацеленных на разную аудиторию, а также на разные области применения.

4.1.3.1 Rebol/Core

Rebol/Core - это фундамент. Этот компактный оценщик, размером около 350 КБ, является сердцем языка. Он включает в себя оценщик, сборщик мусора, диспетчер безопасности, сетевые протоколы и слова из словаря Rebol. Распространяемый по "бесплатной" лицензии, этот продукт можно бесплатно загрузить с сайта rebol.com для многих операционных систем и свободно использовать в коммерческих продуктах.

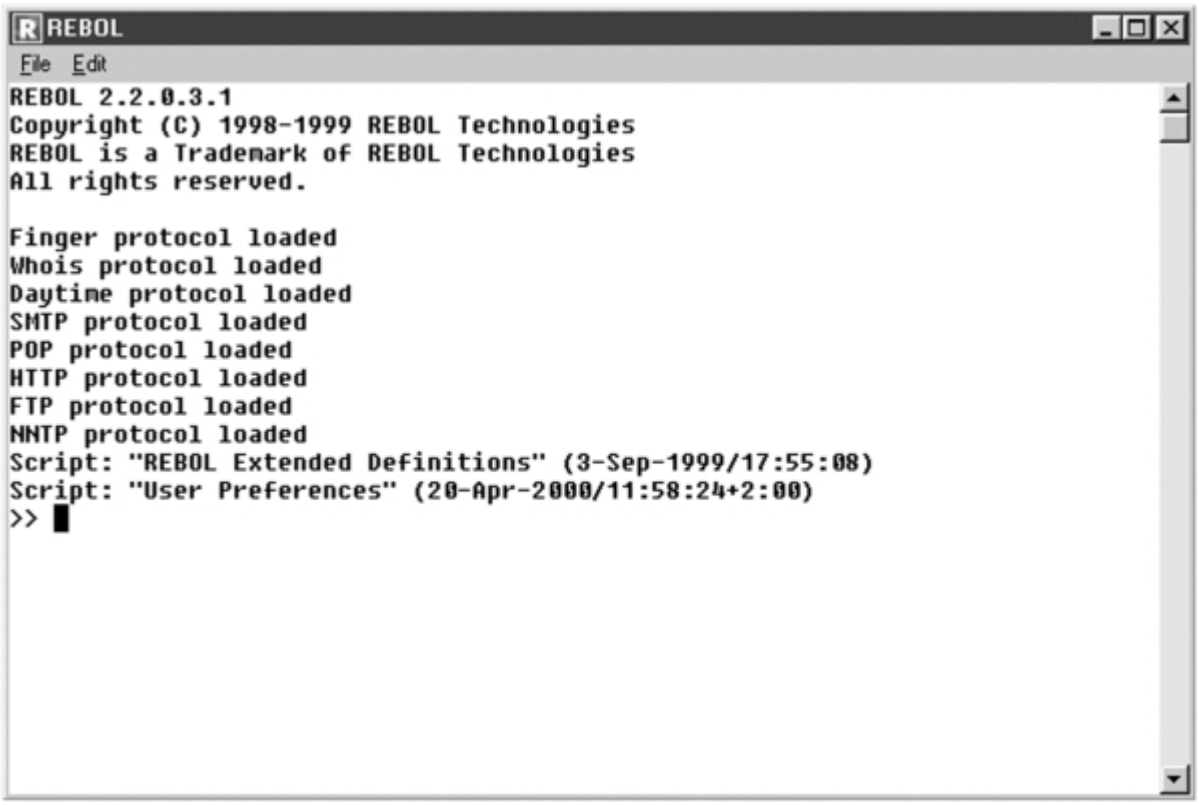

Рисунок 0-2. *Консоль Rebol/Core.*

Rebol/Core не является демонстрационной версией или ограниченным продуктом. Это идеальный инструмент для разработки приложений для обмена информацией, написания CGI или сценариев системного администрирования.

4.1.3.2 Rebol/View

Rebol/View - это наложение поверх Rebol/Core, которое добавляет возможность разработки графических приложений. Начиная с версии 1.3, Rebol/View использует очень мощную графическую библиотеку AGG в качестве механизма отображения. Его возможности впечатляют, поскольку AGG предоставляет расширенные графические примитивы, отличную скорость отображения, управление сглаживанием, векторные шрифты и невероятные эффекты, которые можно применять в реальном времени (поворот, перспектива, размер, прозрачность, обработка цвета и т. Д.). Для создания пользовательских интерфейсов Rebol/View включает VID (Visual Interface Dialect), позволяющий определять экраны с минимумом инструкций. Основные компоненты пользовательского интерфейса включены в стандартную комплектацию (кнопка, поле

ввода, ползунок, изображение, индикатор выполнения и т. Д.). Вы также можете изменить их внешний вид с помощью таблицы стилей и даже создать свои собственные компоненты. VID - это диалект, который позволяет гибко реализовывать сложные графические интерфейсы на нескольких платформах. Все это происходит в одном исполняемом файле размером всего несколько сотен килобайт.

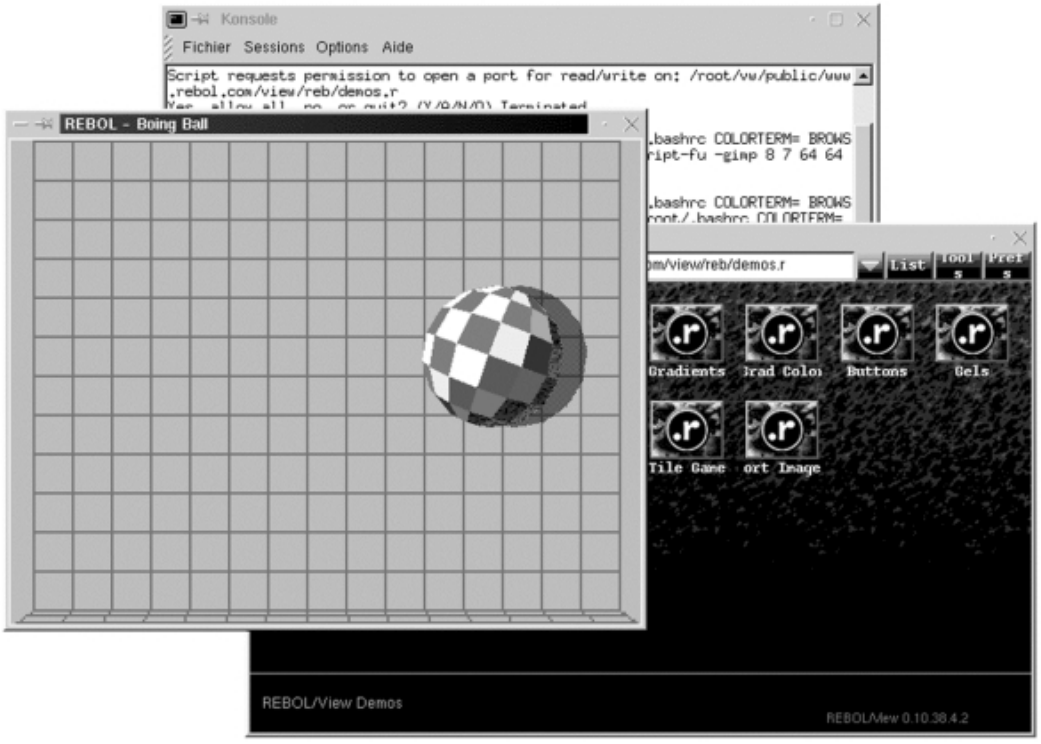

Рисунок 0-3. *Графическое программирование с Rebol/View*

Rebol/View также бесплатен и может распространяться без ограничений. Также доступна коммерческая версия под названием Rebol/View / Pro, которая добавляет к оценщику группу функций шифрования (DES, RSA, DH и управление электронной подписью), а также поддержку протокола HTTPS.

4.1.3.3 Rebol/Command

Для приложений электронного бизнеса вы можете использовать коммерческий продукт под названием Rebol/Command. Эта версия также доступна для нескольких операционных систем и обеспечивает функции шифрования, протокол HTTPS и доступ к источникам данных ODBC и базам данных MySQL и Oracle.
Rebol/Command идеально адаптирован для разработки сложных динамических веб-сайтов и, благодаря встроенной поддержке FastCGI, позволяет создавать высокопроизводительные приложения, которые могут быть распределены по нескольким серверам с использованием многоуровневой архитектуры.

4.1.3.4 Rebol/SDK

Rebol/SDK - это комплект средств разработки, позволяющий создавать исполняемые файлы из сценария Rebol. С помощью SDK становится возможным распространять приложение, написанное на Rebol, без необходимости установки пользователем Rebol/Core, Rebol/View или Rebol/Command. Одна из особенностей SDK заключается в том, что он использует алгоритм шифрования для сокрытия исходного кода сценария Rebol. Таким образом, появляется возможность распространять коммерческое приложение, содержащее проприетарную технологию.

4.1.3.5 Rebol/IOS

Наконец, если вы хотите создать интрасеть с высокой степенью безопасности и в которой сообщество пользователей может обмениваться информацией, приложениями и файлами данных, вы можете использовать платформу группового программного обеспечения Rebol/IOS. Он состоит из двух элементов: сервера Rebol/Serve и многоплатформенного клиента Rebol/Link. Rebol/IOS не только поставляется с большим количеством готовых к использованию приложений (управление задачами, телефонная книга, публикация новостей, инструменты администрирования и т.д.), Но также включает комплект для разработки, который позволяет писать собственные приложения.

4.1.4 Язык сетевого программирования

Эти различные версии Rebol поддерживают несколько протоколов TCP / IP и упрощают разработку приложений, способных обмениваться данными по сети. Поддержка HTTP позволяет разрабатывать веб-клиентов, способных работать во всемирной паутине или вашей собственной интрасети для получения данных. FTP предлагает возможность получать файлы или отправлять их на удаленный сервер. Протоколы POP3 и SMTP позволяют получать и отправлять электронную почту. Доступ к новостным форумам можно получить с помощью протокола NNTP.

Сценарий также может опросить ваш DNS-сервер, чтобы найти машины в вашей сети или получить информацию о пользователе или сервере с помощью протоколов Finger и Whois.

Если этого недостаточно, вы также можете разработать свои собственные протоколы. Фактически, Rebol обеспечивает прямой доступ к портам TCP и UDP на вашем компьютере. Можно писать не только клиентские приложения, но и серверы. Простой HTTP-сервер может быть реализован в Rebol менее чем за 50 строк кода!

4.1.4.1 Манипулирование информацией

Помимо своих талантов в области коммуникаций, Rebol также преуспевает в управлении информацией. Он отличается от других языков программирования количеством и специализацией типов данных. Rebol, очевидно, способен манипулировать числами, действительными числами, строками или логическими значениями, но на этом не останавливается. Rebol - один из немногих языков, который естественным образом распознает и использует IP-адреса и URL-адреса.

Эти простые типы данных можно сгруппировать в списки. Списки - основа языка. В Rebol списки вездесущи. Скрипт Rebol - это список слов. Массив - это список значений. Rebol предоставляет программисту набор слов для доступа к этой информации, такой как просмотр списка и поиск или извлечение данных с минимальным количеством операций.

4.1.4.2 Объектное программирование

Данные могут содержаться в объектах, свойствах и методах группировки. Объекты, написанные на Rebol, позволяют разрабатывать модульные приложения, облегчают командную работу над проектом и улучшают продуктивные аспекты языка. Концептуально намного проще, чем JavaBeans, объекты Rebol также могут передаваться по сети, совместно использоваться несколькими приложениями и иметь механизм самоанализа.

4.1.5 Основные приложения написанные на Rebol

Многие приложения написаны на Реболе. Почти все они бесплатны и работают на всех платформах, на которых есть оценщик Rebol. Их эклектика демонстрирует универсальность простого, элегантного языка. Подключайтесь к World Wide Reb и загружайте протоколы, диалекты, веб-приложения, утилиты и даже видеоигры.

4.1.5.1 Протоколы и диалекты

Протоколы и диалекты расширяют возможности оценщика Rebol. Протоколы относятся в основном к сети. Винсент Демонгодин написал клиент SNMP (Simple Network Management Protocol) для Rebol для сбора информации об активных элементах в сети. Другие важные продукты включают протоколы MySQL и PostgreSQL Ненада Ракочевича (www.softinnov.org). Бесплатные и совместимые со всеми версиями Rebol, они позволяют взаимодействовать с этими известными базами данных SQL. На самом деле существует ряд дополнительных сетевых протоколов, которые вы можете добавить в свой оценщик Rebol.

Диалекты - это языки предметной области. Каждый из них является языком внутри языка и предназначен для наиболее эффективного выполнения задачи. PDF-Maker Габриэле Сантилли упрощает создание PDF-документов. Если вам нужно отображать числовые данные в виде кривых или гистограмм, воспользуйтесь превосходным Q-Plot, который является чрезвычайно эффективным диалектом для таких операций. В веб-домене не упустите диалект SWF, который позволяет динамически создавать Flash-анимацию.

4.1.5.2 Rebol и Интернет

Поскольку Rebol ориентирован на сеть, неудивительно, что он нашел множество приложений для Интернета. MailReader и Rim - это два коммуникационных инструмента, почтовый клиент и клиент чата. Rix - это поисковая система, посвященная документам об исходном коде Rebol и Rebol. ShearchCenter позволяет опрашивать несколько поисковых систем. Vanilla - это веб-приложение для создания сайтов для совместной работы.

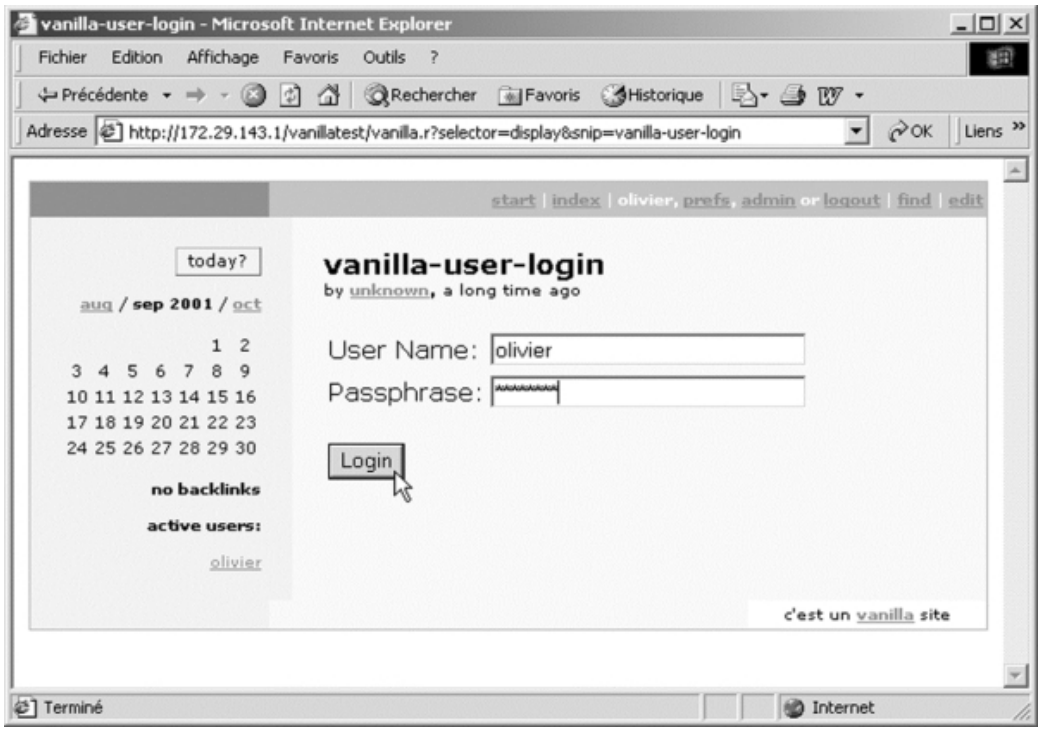

Рисунок 0-4. *The Vanilla wiki.*

Cheyenne - очень мощный HTTP-сервер, полностью написанный на Rebol. AltMe впечатляет, и его трудно игнорировать, он позволяет создавать виртуальные сообщества.

4.1.5.3 Коммунальные услуги как будто идет дождь!

Castro - это файловый менеджер, позволяющий просматривать, переименовывать, перемещать и удалять файлы. С помощью Michka Франсуа Жуэна вы можете классифицировать свои PDF-документы.

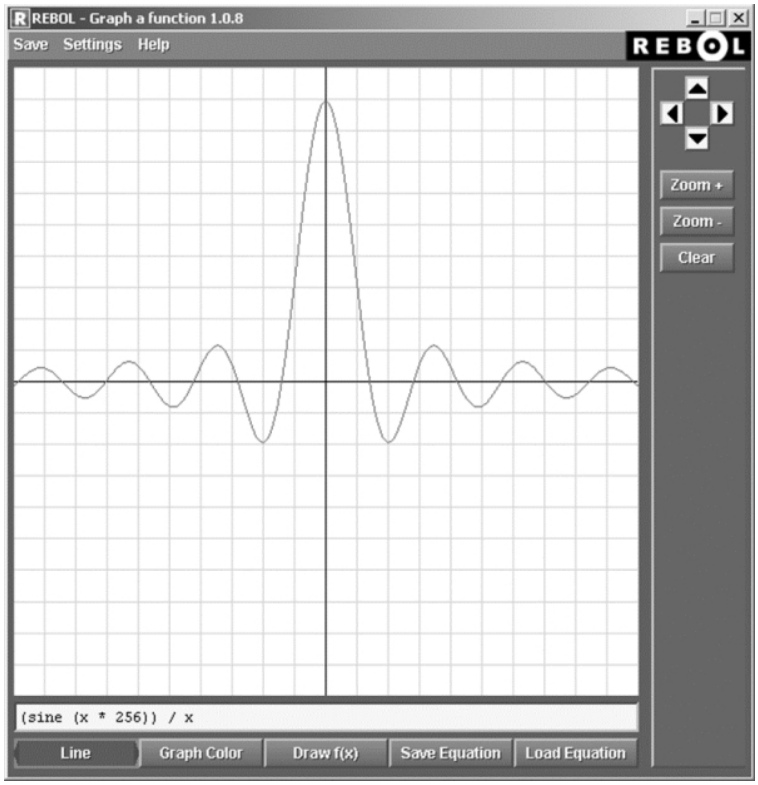

Рисунок 0-5. *Нарисуйте кривые с помощью Grapher.*

Важное значение для разработчиков, Anamonitor полностью контролирует системный объект оценщика Rebol. Наконец, Grapher порадует поклонников математики отслеживанием и повторением всех мыслимых кривых.

4.1.5.4 Мультимедиа и игры

Ребол хорош для игр. Графический движок GCS полностью независим от исполнительной платформы и способен управлять сложными графическими операциями. Он может рисовать, применять эффекты (яркость, поворот, прозрачность и т. Д.) И даже читать изображения во многих форматах (PNG, GIF, JPG и BMP). Самая впечатляющая демонстрация - это, вероятно, Reviewer, отличный продукт для хранения и редактирования графики.

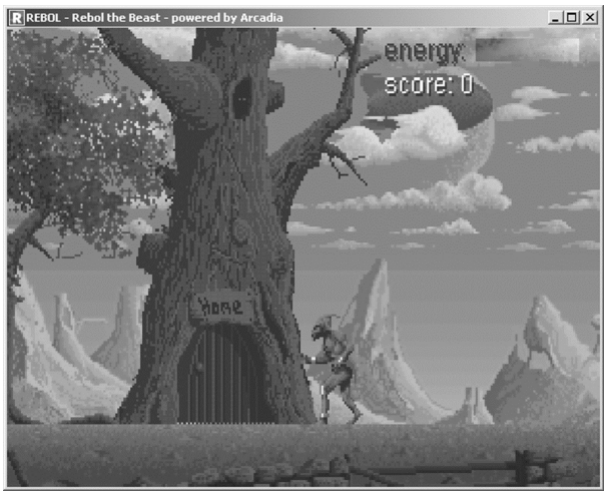

Рисунок 0-6. *Игра, созданная на движке Arcadia.*

Rebol легок, универсален и очень прост в развертывании. В нем есть все необходимое для

разработки игр и мультимедийных приложений. Rebol, сетевой язык с отличными графическими и анимационными возможностями, является фантастическим решением для создания онлайн-игр.

Рисунок 0-7. *Игра R-Box.*

Талантливые программисты уже неоднократно демонстрировали онлайн-возможности Rebol. Взгляните на R-Box2, который является адаптацией классической видеоигры. Со своей стороны, Сайфра, не колеблясь, разработал движок для видеоигр под названием Arcadia, включающий в себя множество функций и специализированный диалект.

4.1.6 Скачивание и установка

Rebol/Core можно загрузить в разделе "Загрузки" на сайте www.rebol.com. Просто выберите файловый архив для вашей операционной системы. Полученный файл находится в формате zip для Microsoft Windows и tar.gz для Unix-систем. На них команда оболочки tar xvzf, за которой следует имя архива, позволяет распаковать файлы в текущий каталог. Будет ряд файлов, включая оценщик Rebol, установочные документы, обновления и сценарии Rebol, которые можно узнать по расширению .r. В системах типа Unix распространенной практикой является копирование исполняемого файла Rebol в каталог / usr / local/bin, чтобы к нему могли получить доступ все пользователи машины.

Рисунок 0-8. *Сайт www.rebol.com.*

После того, как вы это сделаете, вы должны настроить оценщик, чтобы получить от него максимальную пользу, указав свой адрес электронной почты, сервер STMP, сервер POP и любой прокси-сервер, который вы хотите использовать. Используйте для этого сценарий setup.r, введя следующую команду rebol setup.r в командной строке. Этот сценарий представляет собой интерактивную программу, в которой вы просто отвечаете на различные вопросы. Результатом является файл user.r, специфичный для каждого пользователя. В папке также находится файл rebol.r. Как и user.r, этот файл загружается интерпретатором Rebol всякий раз, когда он загружается. Файл rebol.r предназначен для хранения функций, которые будут автоматически включены в оценщик. Чтобы обнаружить все слова, доступные в Rebol/Core, вы можете запустить сценарий words.r с помощью синтаксиса командной строки rebol words.r. Затем вы получите файл words.html, содержащий все определения каждого элемента словаря Rebol.

В архиве Rebol/View всего три файла и два из них - документация. Фактически, вы просто запускаете исполняемый файл Rebol, чтобы запустить процедуру установки, чтобы указать каталог, в который будет скопирован Rebol/View, а также указать настройки вашей электронной почты и, возможно, прокси-сервера.

После завершения настройки оценщик запускается и отображает рабочий стол, который даёт вам доступ к различным ресурсам.

Пункт меню "Пользователь" в верхней части окна позволяет изменить вашу конфигурацию. С помощью опции меню Goto вы можете указать URL-адрес сценария Rebol в Интернете и, таким образом, запустить его. Слева папка Rebol.com дает вам доступ к приложениям, написанным на Rebol, которые доступны через Интернет и могут быть загружены через Интернет для работы на вашем компьютере. После загрузки эти сценарии, хранящиеся в кеше, всегда доступны вам независимо от того, подключены вы к Интернету или нет. Это иллюстрирует одну из ключевых концепций Rebol: X-Internet. Эта идея описывает сеть обмена информацией, в которой каждый узел является активным участником. Rebol/View, имеющий вес менее 600 КБ, ориентированный на сеть язык сценариев, платформенно-независимый, экономичный с системными ресурсами и включающий расширенные графические функции, является модельным клиентом для такой архитектуры.

Под папкой Rebol.com папка Public предоставляет доступ к библиотеке сценариев Rebol, размещённой на Rebol.org, и множеству сайтов RebSites, размещённых различными членами сообщества Rebol. Они содержат сотни полезных сценариев и, следуя принципу X-Internet, также сохранятся в кэше, чтобы их можно было использовать, когда вы не подключены к Интернету.

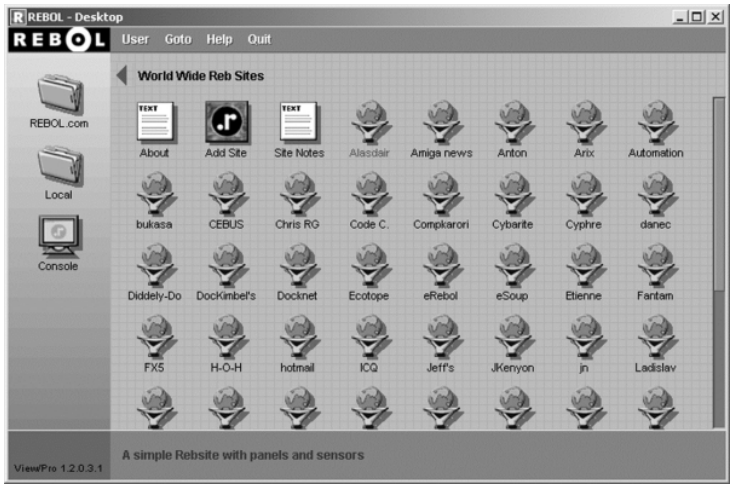

Рисунок 0-9. *Rebol/View предоставляет доступ к RebSites.*

Затем вы увидите локальную папку для классификации и доступа к различным скриптам Rebol на вашем компьютере. Наконец, опция Console позволяет запускать интерактивную консоль, идентичную консоли Rebol/Core.

4.1.6.1 Использование консоли

Консоль Rebol содержит множество возможностей, облегчающих работу с Rebol. Во-первых, это позволяет вам работать в интерактивном режиме и тестировать последовательности кода. Например, вы можете ввести 2 + 2, а затем нажать клавишу ввода. Сразу же инструкции обрабатываются, и результат отображается в консоли. Кодовые последовательности не ограничиваются одной строкой, можно вводить полные блоки перед их оценкой. Во время набора текста клавиши со стрелками вправо и влево перемещают курсор, чтобы вы могли вносить исправления. Используя клавиши со стрелками вверх и вниз, вы можете прокручивать предыдущие команды и, нажав клавишу ввода, повторно выполнить предыдущие инструкции. Одно нажатие клавиши табуляции активирует функцию автозаполнения Rebol. При быстром двойном нажатии на нее отображается список слов Rebol, которые начинаются с уже набранных символов. Для получения дополнительной информации о конкретном слове вы можете использовать слово help, за которым следует слово Rebol, о котором вы спрашиваете. Каждое слово Rebol задокументировано самостоятельно, и вы можете сделать то же самое для своих собственных функций. Функция справки отображает значение слова, его различные параметры и его использование в консоли.

Есть разные способы запустить скрипт, находящийся на вашем жестком диске. Вы можете указать имя файла в командной строке, создать папки, используемые локально Rebol/View, или использовать слово do, за которым следует имя файла. Таким образом, чтобы оценить файл myscript.r, вы набираете в консоли команду do %myscript.r. Символ "%" сообщает Rebol, что аргумент имеет тип файла (файл!) или путь (путь!). В более широком смысле, поскольку Rebol является сетевым языком, вы также можете выполнить сценарий, хранящийся на удаленном сервере, который будет прочитан с помощью протоколов HTTP или FTP TCP. Инструкция do http://myserver/scripts/transfer.r запускает приложение, хранящееся в файловой системе веб-сервера.

Чтобы помочь вам разрабатывать собственные сценарии, консоль включает некоторые специальные инструменты, такие как слово trace, которая принимает логическое значение (включено или выключено) для включения или отключения мониторинга кода во время его выполнения. Эта инструкция также позволяет вам отслеживать действия в сети и сообщать о различных вызовах функций. Через слово echo вы можете указать имя файла журнала, в котором будет сохраняться информация, отображаемая в консоли. Когда слово получает аргумент со значением none, запись в журнал останавливается. recycle дает контроль над сборщиком мусора Rebol. Автоматическое управление памятью можно включить или отключить.

4.1.6.2 Создание рабочей среды

Консоль - очень удобный инструмент, но его недостаточно для написания приложений Rebol. Вам, конечно, понадобится редактор кода для ввода ваших скриптов. В этой области есть из чего выбирать, и это в основном вопрос вкуса. Существуют файлы конфигурации для многих редакторов, которые обеспечивают цветовую кодировку кода Rebol.

Вы можете выбрать такие продукты, как Vim (www.eandem.co.uk/mrw/vim/syntax) или Emacs (www.rebol.com/tools/rebol.el) для Unix.

В KDE редактор Kate также можно настроить для распознавания сценария Rebol (www.errru.net/rebol/utilities/katesyntax). В MacOS X отличный редактор SubEthaEdit (www.codingmonkeys.de) включает файл раскраски синтаксиса. В Windows вы найдете замечательный и бесплатный редактор Crimson Editor (www.crimsoneditor.com). Если вы регулярно переключаетесь между разными машинами, вы можете попробовать бесплатный JEdit (www.jedit.org), который работает на любой машине с установленной Java и включает не только раскраску кода Rebol, но и красивую функцию автоматического выравнивания скобок.

Если вы используете Rebol/View, вы можете использовать его встроенный редактор, вы можете активировать его, щелкнув правой кнопкой мыши значок файла на рабочем столе View и нажав кнопку Edit или набрав editor, за которым следует имя файла, который вы хотите отредактировать. в консоли Rebol/View.

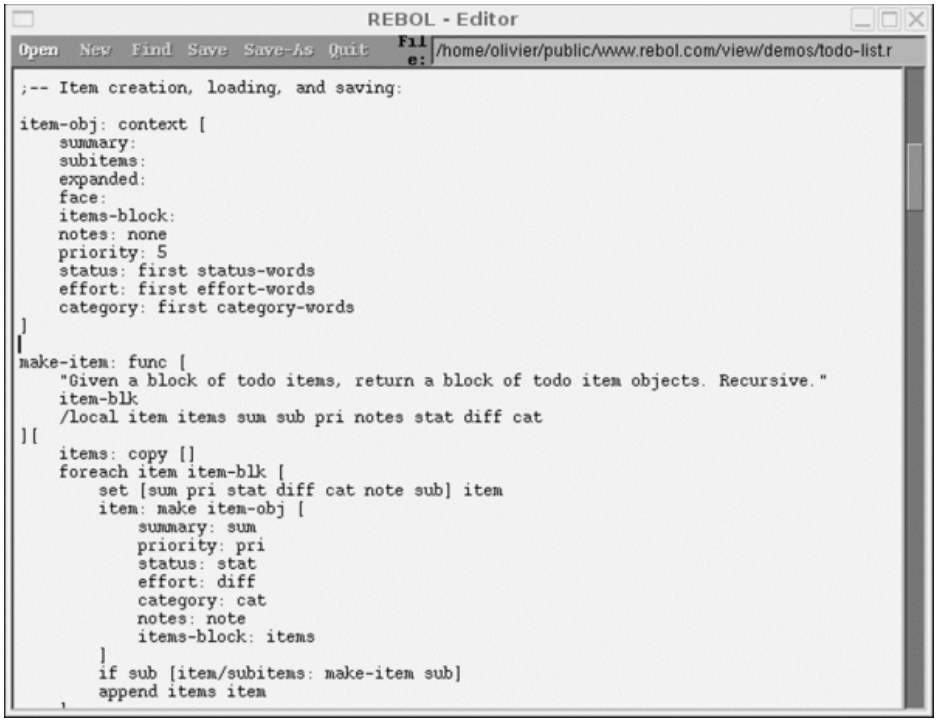

Рисунок 0-10. *Редактор кода Rebol/View.*

Мы подошли к концу этого введения в Rebol. До сих пор все это было очень теоретическим, но с этого момента мы начнем программировать, даже в самой первой главе.

5. Откройте для себя Rebol за час

Для обучения в Rebol вы собираетесь использовать Rebol/Core или Rebol/View, которые бесплатно доступны для всех основных платформ разработки. Независимо от того, используете ли вы Mac OS X, Linux или Windows, вы можете начать изучать этот фантастический язык программирования.

5.1 Проект robotFTP

К концу этого введения вы разработаете полную программу и узнаете много разных аспектов Rebol. Теперь вы начнете краткое руководство, чтобы узнать об основных аспектах программирования в Rebol. Для этого лучше всего реализовать настоящий проект.

5.1.1 Введение в проект

Целью является разработка автоматизированного FTP-клиента. Это означает, что вы собираетесь написать сценарий, способный извлекать файлы и отправлять файлы на анонимный FTP-сервер. Различные операции будут описаны в файле конфигурации. Этот сценарий позволит вам развернуть файлы по расписанию или сделать резервную копию файлов, хранящихся на удаленном сервере. При использовании вместе с командой Unix cron ваш сценарий может запускаться периодически без какого-либо вмешательства человека.

5.1.2 Технические соображения

Фактически, вы собираетесь написать интерпретатор, задача которого будет заключаться в реализации набора элементов управления, аналогичных тем, которые присутствуют в обычном FTP-клиенте. Приложение может перемещаться по древовидной структуре сервера или клиента, отправлять или получать файл, обрабатывать пакеты файлов по их расширению, создавать каталог или удалять файл или каталог на удаленном компьютере. Для описания его операций вы

будете использовать объект Rebol, содержащий имя удаленного хоста, логин и пароль, которые будут использоваться, логическое значение, указывающее на FTP-соединение в пассивном режиме, и, наконец, список команд FTP, которые должны быть выполнены.
С Rebol/Core скрипт работает "тихо" (не требует ввода с клавиатуры и не производит вывод на экран) и может быть полностью автоматизирован. Однако, если сценарий запускается в REBOl/View, для отображения хода выполнения операций будет использоваться графический дисплей. Имя необходимо для любого проекта, и для этого это будет robotFTP. Среда исполнения
Сначала создайте файл с именем robotftp.r в текстовом редакторе.
Чтобы этот файл был исполняемым в Unix (и Mac OS X), вы должны указать путь к исполняемому файлу Rebol в первой строке файла, чтобы вызвать интерпретатор сценария. Таким образом, вы должны использовать последовательность символов #! за которым следует путь к исполняемому файлу Rebol. Это известно как строка Shebang.
Часто бывает полезно предоставить Rebol некоторую дополнительную информацию для определения среды выполнения. Список опций можно получить, запустив Rebol с опцией –help. Параметр -c используется для запуска сценариев CGI. Если вы хотите напрямую оценить выражение Rebol из командной строки, вы можете использовать параметр --do, за которым следуют инструкции Rebol. Итак, команда rebol --do "print 2 + 2" отобразит результат оценки. Эта опция полезна, когда вы хотите установить значение переменной перед запуском скрипта.
Сценарии Rebol, работающие под Windows, не нуждаются в строке Shebang. Вы можете просто дважды щелкнуть скрипт, и он автоматически запустит Rebol (при условии, что имя файла заканчивается на .r и расширение файла r связано с Rebol). Когда вы запускаете сценарии Rebol таким образом, невозможно предоставить дополнительную информацию для определения среды выполнения для каждого сценария. Для этого вам нужно запустить Rebol из командной строки Windows; например c: Rebol \ Rebol.exe –c robotftp.r.
Для нашего проекта robotFTP вы собираетесь использовать наиболее распространенные параметры: -q для запуска в тихом режиме (без отображения какой-либо информации) и -s для отключения встроенного диспетчера безопасности Rebol, чтобы интерпретатор не запрашивал подтверждение от пользователя для доступ к каждому ресурсу. Менеджер безопасности Rebol позволяет точно контролировать свободу действий сценария. Это очень важно для сетевого языка программирования, с помощью которого можно загружать и запускать приложения через Интернет. Менеджер безопасности позволяет контролировать доступ к файловой системе, а также доступ к внешним сетям.

5.1.3 Менеджер по безопасности

Слово "secure" определяет поведение интерпретатора в ответ на требования сценария к доступу к файлам (чтение(read), запись(write), выполнение(execute) и все(all)), устанавливая правила, основанные на четырех вариантах (разрешить(allow), спросить(ask), запретить(throw) и выйти(quit)). Логика диспетчера безопасности состоит в том, чтобы разрешить все, что явно разрешено, и даже разрешить изменение политики безопасности, если это подкрепляется новым правилом.
В противном случае диспетчер безопасности вынужден запрашивать у пользователя конкретное разрешение, прежде чем разрешить доступ к любому файлу. Чтобы избежать переопределения некоторых критических функций сценарием, диспетчер безопасности также предоставляет пользователю слово protect-system. Как видите, в Rebol нелегко относятся к безопасности, поскольку он обеспечивает чрезвычайно безопасную среду выполнения. Для нашего проекта robotFTP первая строка скрипта выглядит так: #!/usr/bin/rebol -qs (очевидно, путь к файлу зависит от вашей установки).

5.1.4 Написание заголовка

Следующим шагом будет набросок заголовка. Каждый сценарий Rebol начинается с информационного раздела, целью которого является документирование кода. В блоке данных Rebol программист имеет возможность использовать различные поля для представления сценария. Эти метаданные могут быть непосредственно прочитаны человеком. Выбор полей полностью оставлен на усмотрение автора. Однако разумно следовать стандартам, установленным в документации по языку Rebol.
В пятой главе "Руководства пользователя Rebol/Core" представлены некоторые правила, которым нужно следовать, чтобы писать четкие и разборчивые сценарии Rebol, а также предлагаются стандартные поля для использования для информации заголовка. Следуя этому стандарту, становится возможным использовать отражающие функции Rebol и выполнять поиск по автору,

категории, дате или номеру версии скриптов Rebol на жестком диске.
Основные поля, рекомендуемые в документации, это title для заголовка скрипта, date для даты написания и author для имени программиста. Вы также можете указать имя сценария (name), имя файла (file), номер версии (version) или версию Rebol, необходимую для запуска сценария (needs). С помощью полей purpose(назначение), note(примечание) и history(история) программист может предоставить подробное описание цели сценария, указать его функциональные возможности и показать, как он развивался.

```
REBOL [
    title: "automatic FTP client"
    author: "Olivier Auverlot"
    version: 1.0
    purpose: {
        RobotFTP executes of FTP commands
        in order to automate the sending and receipt of files
    }
usage:  {
    robotftp commands-script
}
]
```

5.1.5 Основные типы данных

Создание заголовка - это возможность открыть для себя множество типов данных, доступных в Rebol. Наряду с простыми типами, такими как integer!, string!, logic! или char!, язык предоставляет типы, адаптированные для хранения и обработки больших объемов информации с помощью block!, hash! и binary!.
Rebol демонстрирует свою адаптацию к сетевому программированию и обмену информацией с помощью ряда специальных типов, таких как tag!, url!, tuple! и email!. Фактически, в вашем распоряжении более пятидесяти различных типов данных. Все, что вам нужно сделать, чтобы получить их список, - это ввести команду help datatype! в консоли Rebol.

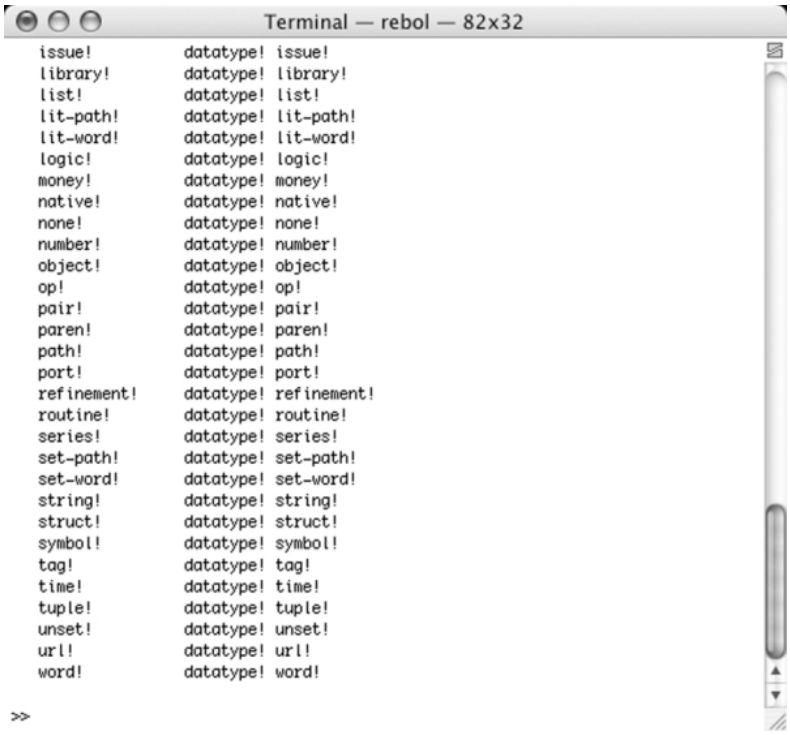

Рисунок 1-1. *Отображение типов данных Rebol в консоли.*

Вас не должно вводить в заблуждение такое обилие типов; Rebol не является строго типизированным языком (в традиционном смысле) и отдаёт предпочтение гибкости. Даже если вы можете объявить переменную с заданным типом, тип может измениться в любой момент во время выполнения сценария.
Имейте в виду, что Rebol может определять тип данных. Итак, если набрать type? olivier.auverlot@domaine.fr, в консоли Rebol интерпретатор однозначно вернё т ответ: email!.

5.1.6 А теперь код!

Фактически, когда запускается сценарий robotftp.r, он принимает в качестве аргумента имя файла, содержащего различные команды FTP, которые будут выполняться.Для этого мы будем использовать свойство args объекта options, содержащегося в объекте system. Последний является ядром интерпретатора и представляет собой древовидную структуру, состоящую из ряда объектов, содержащих множество рабочих параметров, таких как тип и версия интерпретатора, каталог установки, каталог пользователя, сетевые протоколы и собранная статистика. Он также содержит весь словарь Rebol. Это уникальное хранилище содержит константы, функции, написанные на Rebol, и собственные функции, написанные на C. Если вы хотите приступить к изучению этого объекта, все, что вам нужно сделать, это ввести в консоли Rebol команду "print mold system".
Доступ к свойству или методу объекта достигается с помощью символа / (слеш), который устанавливает путь доступа. Чтобы прочитать содержимое args, вы должны написать system/options/args. Это путь содержит блок символьных строк. Следовательно, вы должны получить первый элемент этого пути и преобразовать его в тип файла, прежде чем загружать объект, описывающий операции FTP, в память и оценивать (выполнять) его. В Rebol этот набор операций выполняется с помощью однострочной команды: do load to-file first system/options/args. Чтобы избежать ошибки выполнения, если список пуст или файл не существует, эту строку следует заключить в "устойчивый к ошибкам" блок, которому предшествует слово try. Блок всегда оценивается. Если объект типа error! возвращается, второй блок выполняется для решения проблемы.

```
if error? try [
    commands: do load to-file first system/options/args
] [ error/fatal "Command script could not be loaded" ]
```

5.1.7 Объявление функции

Если возникает проблема, вызывается функция error. Её роль - вывести сообщение для пользователя и, при необходимости, остановить выполнение скрипта. Это решение указывается с помощью функции REBOL, известной под термином "уточнение" (refinement). Этот механизм позволяет изменять поведение функции и использовать различное количество параметров. В этом случае используется уточнение /fatal (фатальная ошибка), которая объявляется внутри функции.

```
error: func [ "Display an error message"
    msg [ string! ] "Description of the problem"
    /fatal "The error requires stopping the program"
] [
    print msg
    if fatal [ quit ]
]
```

Функция - это слово типа данных function!. Его можно объявить по-разному, используя синтаксис make function!, func или does.
Слово error принимает в качестве параметра символьную строку с именем msg и обеспечивает уточнение /fatal. Опять же, Rebol позволяет программисту включать метаданные в код. Можно указать цель функции и задокументировать поддерживаемые параметры и уточнения. После

оценки функции команда help извлекает эту информацию и отображает её в консоли Rebol. Тело функции просто отображает сообщение, предоставленное в качестве параметра, и выполняет простой тест, чтобы определить, использовалось ли уточнение /fatal.

5.1.8 Управление URL-адресами

После анализа содержимого командной строки вы можете создать URL-адрес для подключения к FTP-серверу. В объекте, описывающем операции, имя (user) и пароль (password) пользователя. Если имя пользователя отсутствует, соединение будет анонимным. Поэтому вы должны получить доступ к содержимому объекта user и убедиться, что его значение не равно нулю. Вы заметите в коде, что перед user стоит символ ' (апостроф). Это сообщает интерпретатору Rebol, что этот элемент данных является символом и не должен оцениваться интерпретатором.

Имя сервера указывается с помощью свойства host. Поскольку все данные уже загружены в объект commands, все, что вам нужно сделать, это использовать функции обработки строк Rebol для сборки URL. В Rebol строка - это список символов, которыми можно управлять и перемещаться так же, как блоки. Словарь содержит множество функций для копирования, извлечения, объединения и поиска последовательностей символов. Инициализация сетевых параметров заканчивается активацией пассивного режима путём изменения логического значения свойства system/scheme/ftp/passive. Наконец, сценарий назначает текущее дерево каталогов FTP-сервера слову curdir. Когда пользователь перемещается по папкам, эта переменная будет отвечать за сохранение местоположения.

```
ftpurl: copy "ftp://"
if not none? get in commands 'user [
 ftpurl: join ftpurl [ commands/user ":" commands/password "@" ] ]
ftpurl: join ftpurl commands/host

if not none? get in commands 'passive [
 if commands/passive = true [ system/schemes/ftp/passive: true ] ]
curdir: copy %/
```

5.2 Выполнение команд FTP

Теперь вам нужно определить, как различные команды FTP будут описаны в файлах конфигурации. Это объекты, состоящие из различных свойств, которые указывают настройки подключения (host (хост), user (пользователь), password (пароль) и passive (пассив)) и операции, которые необходимо выполнить. Для этого вы используете свойство, называемое script, содержимое которого является блоком. Это позволит вам использовать одну из самых интересных возможностей Rebol: диалекты.

5.2.1 Определение и использование диалектов

Фактически, Rebol может определять уникальные языки, функция которых связана с конкретной задачей. Это специализированные словари, которые не являются частью словаря Rebol, но могут определяться и управляться пользователем. Таким образом, используя грамматический анализатор, использующий нотацию BNF (Backus-Naur Form), можно определять мини-языки в Rebol.

Диалекты могут использоваться во многих областях, таких как описание операций, установление сетевого протокола или определение ограничений и правил. Более того, сам интерпретатор Rebol содержит различные диалекты, предназначенные для работы со строками или описания графических интерфейсов. Некоторые библиотеки также используют эту дополнительную функциональность для элегантного расширения возможностей языка, поэтому вы можете найти в Интернете диалект для создания PDF-документов (www.rebol.org) или еще один, который упрощает управление консолями в текстовом режиме (hthttp://rebol.info/forces/articles/tui-dialect).

5.2.2 Диалект robotFTP

В случае проекта robotFTP вам понадобится диалект, синтаксис которого аналогичен командам для FTP-клиентов. Итак, это набор команд, каждая из которых может иметь необязательный параметр. Определение правил происходит в блоке, где каждый символ относится к возможной команде. Если инструкция получает параметр, вы должны определить тип этого параметра и имя переменной, которая получит его значение. Остальная часть определения заключена в круглые скобки и содержит код Rebol, который должен выполняться при обнаружении команды во время анализа.
В следующем коде вы, вероятно, заметите, что определение различных команд заключено в блок any [], и каждая строка заканчивается знаком "|". Причина довольно проста: это указать интерпретатору, что, по крайней мере, одно из грамматических условий должно быть выполнено для того, чтобы анализ был правильным ("|" - это синтаксис для "or" (или)).

```
rules: [
 any [
  'quit (quit) |
  'debug set arg string! (print arg) |
  'lcd set arg file! (exec-cmd [
           change-dir arg ]) |
  'get set arg file! (exec-cmd [
           write arg (read/binary make-ftpurl arg) ]) |
  'put set arg file! (exec-cmd [
           write/binary (make-ftpurl arg) read/binary arg ]) |
  'cd set arg file! (  either arg = %/ [
           curdir: copy %/
           ] [
           append curdir reduce [ arg "/" ] ]) |
  'mkdir set arg file! (exec-cmd [
           make-dir make-ftpurl/directory arg ]) |
  'delete set arg file! (exec-cmd [ delete make-ftpurl arg ]) |
  'rmdir set arg file! (exec-cmd [ delete make-ftpurl/directory arg ]) |
  'mput set arg file! (
    exec-cmd [
     foreach file read %. [
      if (suffix? file) = arg [
       write/binary (make-ftpurl file) read/binary file
      ]
     ]
    ]
  ) |
  'mget set arg file! (
    exec-cmd [
     files: read make-ftpurl ""
      forall files [
       files: first files
       if (suffix? files) = arg [ write/binary files read/binary (make-ftpurl
files) ]
      ]
    ]
  )
 ]
]
```

5.2.3 Управление ошибками

Некоторые команды очень простые. Команда quit (выход) просто закрывает FTP-соединение и завершает сеанс интерпретатора Rebol. Команда debug (отладкf) получает строку символов в качестве параметра, который она отображает на экране. Другие инструкции немного сложнее,

поскольку они предназначены для использования протокола FTP. По этой причине выполнение кода делегируется функции exec-cmd, целью которой является управление ошибками.

```
exec-cmd: func [
 cmd [ block! ] "Rebol Instructions"
 /local err
] [ if error? err: try [ do cmd ] [ print mold disarm err ] ]
```

Эта функция принимает в качестве параметра блок, содержащий код Rebol, и определяет локальную переменную с именем err. Это используется для сбора объектов ошибок, перехваченных словом try. Инструкции Rebol выполняются с помощью команды do, которая просто оценивает (выполняет) блок кода.
В случае возникновения проблемы, disarm преобразует ошибку из типа данных error! в объект, который затем отображается на экране. Этот объект содержит различную информацию, включая текстовое сообщение с описанием ошибки.
Слово mold предназначена для подготовки информации любого типа для отображения на экране. Например, строка символов будет заключена в кавычки, а блок - в квадратные скобки. В этом случае будет отображаться объект.

5.2.4 Условные выражения

Навигация по различным каталогам на FTP-сервере возложена на команду cd. Его работа заключается в обновлении содержимого переменной curdir с помощью простого теста. Для этого Ребол предлагает два слова; if (если) и either (либо).
За if следует условие и блок кода, который должен быть оценён, если условие истинно. either, со своей стороны, следует условие и два блока кода: первый оценивается, если условие истинно, а второй - если условие ложно. Что касается операторов, Rebol очень классический, так как вы можете использовать =, , <=,>= или <>.
В сценарии robotFTP, если предоставленный аргумент является значением корня файловой системы (%/), это значение присваивается curdir. В противном случае результатом оценки блока является добавление аргумента и символа "/" к текущему каталогу.

5.2.5 Управление файлами

В других функциях используются слова, относящиеся к манипулированию файлами. Эти команды взаимодействуют с сервером по протоколу FTP, поэтому необходимо установить URL-адрес на основе аргумента, переданного в качестве параметра. Для этого вы собираетесь определить слово make-ftpurl, которое возвращает значение типа url!. В качестве первого шага аргумент типа file! или string! "очищается" с помощью слова clean-path. Таким образом, путь типа %/home/olivier/docs/../autres/ будет преобразован в %/home/olivier/docs/. Вам также необходимо различать создание URL-адреса для файла или каталога. Для этого вы должны определить уточнения /directory, который, если он используется, добавит символ "/" в конце URL-адреса.

```
make-ftpurl: func [ "Constructs the URL for FTP commands"
 arg [ file! string! ] "Argument passed by the dialect"
 /directory "The URL is for a directory"
] [
 clean-path curdir
 either not directory [
  to-url join ftpurl [ curdir arg ]
 ] [ to-url join ftpurl [ curdir arg "/" ] ]
]
```

Команда lcd может указать локальный каталог с помощью слова change-dir. Создание и удаление каталогов на FTP-сервере поручено командам mkdir и rmdir, которые вызывают слова make-dir и

delete из словаря Rebol. Команды put, get, mput и mget более сложны, поскольку они читают или записывают файлы либо на клиенте, либо на FTP-сервере. Для этого они используют слова Rebol для чтения и записи, а также уточнение /binary для работы в двоичном режиме.
Команда mput позволяет отправлять на FTP-сервер группу файлов с общим расширением. read возвращает список файлов в блоке. Затем этот список просматривается с помощью foreach, и каждое значение, в свою очередь, присваивается переменной file. С помощью слова suffix? его расширение можно сравнить с расширением, указанным в аргументе. Если получено логическое значение "истина" (true), файл передаётся на сервер. Команда mget выполняет обратную операцию, но использует другое слово, предназначенное для просмотра списков: forall.

5.2.6 Комплектование и тестирование robotFTP

Теперь вы можете завершить своё приложение, вставив строку команд parse commands/script rules.
Эта инструкция инициирует применение правил синтаксического анализа к commands/script. Чтобы протестировать ваше приложение, с помощью текстового редактора напишите небольшой тестовый скрипт под названием test.r. Он содержит объект, состоящий из четырёх свойств для параметров соединения и сценарий, описывающий операции, которые необходимо выполнить.

```
context [
 host: "monserveur.domaine.fr"
 user: "olivier"
 password: "homer"
 passive: false

 script: [
  debug "debut"
  lcd %temp
  mget %.r
  get %gscite159.tgz
  mput %.c
  debug "fin"
 ]
]
```

Теперь вам нужно запустить его, набрав ./robotftp.r test.r в оболочке. В случае возникновения проблемы убедитесь, что у вашего скрипта есть права на выполнение (chmod + x). (В Windows введите c:\\rebol robotftp.r test.r). Если все прошло успешно, выполнялись операции, описанные в файле test.r. Программа хорошо подходит для работы под управлением планировщика, такого как утилита cron для Unix, поскольку она может работать в автоматическом режиме. В некоторых случаях может быть интересен и графический вид. Теперь вы можете изменить скрипт, чтобы воспользоваться функциями Rebol/View.

5.3 Добавление графического слоя

Наша цель - добавить в скрипт поддержку графического интерфейса, чтобы продемонстрировать возможности Rebol/View. Мы откроем окно, содержащее индикатор выполнения, и отобразим ход выполнения команд FTP. Скрипт будет запускаться на разных версиях Rebol, поэтому необходимо определить, какая версия интерпретатора используется в скрипте. Для этого вы используете слово view?, которое возвращает значение true, если скрипт выполняется Rebol/View.

5.3.1 Управление индикатором выполнения

В начале скрипта вы добавите строку кода для инициализации переменной stp. Индикатор выполнения работает с диапазоном значений от 0 до 1 (100%), а stp - это значение, которое будет добавлено к индикатору выполнения по завершении каждой команды FTP. Чтобы вычислить разумное значение шага, мы делим число 1 на количество команд в блоке commands/script (по приближению количество элементов, делённое на два).

5.3.2 Открытие окна

Затем мы определяем функцию, progress-window, которая открывает окно в центре экрана. Компоновка (layout) окна определяется с помощью диалекта VID Rebol и назначается объекту ftp-box. Это включает заголовок (title) и индикатор выполнения (progress), присвоенные объекту, progression.

```
if view? [
 stp: 1 / ((length? commands/script) / 2)
 progress-window: does [
  view/new center-face ftp-box: layout [
   title "robotFTP"
   progression: progress
  ]
 ]
]
```

5.3.3 Интеграция со скриптом RobotFTP

Чтобы обновить индикатор выполнения автоматически, необходимо изменить функцию exec-cmd, чтобы она использовала Rebol/View. Для каждой выполненной команды FTP свойство данных объекта progression увеличивается, а графический компонент обновляется.

```
exec-cmd: func [
 cmd [ block! ] "Instructions Rebol"
 /local err
] [
 if error? err: try [
  do cmd
  if view? [
   progression/data: progression/data + stp
   show progression
  ]
 ] [ print mold disarm err ]
]
```

Перед тем, как начать интерпретацию правил, теперь необходимо вызвать функцию progress-window, чтобы окно отображалось. После завершения всех команд перед закрытием главного окна отображается модальное диалоговое окно подтверждения.

```
if view? [
 request/ok "Operations completed"
 unview ftp-box
]
```

5.3.4 Резюме

Вы подошли к концу этого краткого исследования языка Rebol, в котором вы рассмотрели ряд тем, таких как написание сценария, сетевое программирование и разработка графического интерфейса. Этот небольшой проект позволил вам взглянуть на многие аспекты нового языка, совершенно другого, разработанного для поощрения обмена данными.

6. Язык Rebol

Давайте начнём! Введите следующую инструкцию в консоли Rebol:

```
print "Кукушка"
```

Нажмите клавишу "Ввод" на клавиатуре, и вы сразу увидите на экране строку символов "Кукушка". Поздравляем, мы только что выполнили нашу первую оценку. Мы использовали слово print, функция которого заключается в отображении на экране параметра, который сразу за ним следует. Не беспокойтесь о заглавных и строчных буквах, Rebol не заботится: он совершенно не чувствителен к регистру.

6.1 Руководство по выживанию

Давайте продолжим наше исследование, посмотрев на слово what - оно отображает весь словарь Rebol в консоли. Если слово Rebol вас интересует, Rebol может предоставить вам информацию о роли и функции этого слова.
Если вы хотите знать, что делает слов input, просто введите фразу help input, и вы получите описание на экране. Более того, введите source input, теперь вы можете посмотреть исходный код ввода слова. Эта операция возможна для всех слов в словаре, которые определены в самом Rebol. Интерпретатор Rebol различает слова, написанные на Rebol (мезонинные функции), и слова, созданные в машинном коде (собственные функции), для которых недоступен исходный код.

6.1.1 Напишем программу

До сих пор мы только напрямую оценивали наши слова в консоли Rebol. Мы еще не написали программу. Мы можем написать программу Rebol с помощью простого текстового редактора. Все программное обеспечение, написанное на Rebol, должно начинаться с блока описания, позволяющего документировать ваш код. Этот блок может быть пустым, но всегда должен присутствовать.

```
Rebol [
 Subject: "my first program"
 Author: "Olivier Auverlot"
 Version: 1.0
]
```

Вы даже можете создавать свои собственные заголовки в соответствии с вашими потребностями (описания обновлений, требуемые объекты или библиотеки, дата последнего изменения и т.д.). Есть рекомендуемый набор заголовков, но вы не обязаны их использовать. Эта свобода позволяет адаптировать блок описания к потребностям каждого проекта; Важно убедиться, что между различными программными компонентами существует согласованность. Так что наш блок тоже можно было бы записать:

```
Rebol [
Title: "my first program"
Author: "Olivier Auverlot"
Version: 1.0
Date-released: 10/11/2000
]
```

Что будет делать наша первая программа? Он просто отобразит строку символов на экране, подождет, пока не будет нажата клавиша "Enter", а затем прекратит работу. Для этого добавьте следующий код сразу после блока описания:

```
print "Do the REBOLution !" input
quit
```

6.1.2 Запуск вашего скрипта

Сохраните эту программу на свой диск под именем programme1.r. Если вы работаете с Windows, все, что вам нужно сделать, это дважды щёлкнуть значок файла сценария. Под Linux и Mac OSX вы можете запустить сценарий, введя команду оболочки rebol programme1.r

Второй вариант в Linux и Mac OSX - связать скрипт с интерпретатором Rebol. Сначала вставьте инструкцию #!/usr/bin/rebol –qs в качестве первой строки вашего скрипта; убедитесь, что это до открытия блока Rebol. (Эта линия называется линией shebang в мире Unix). Как видите, мы можем передавать параметры выполнения интерпретатору Rebol при его запуске. Когда наш скрипт запущен, интерпретатор Rebol не отображает никаких сообщений и не налагает никаких ограничений безопасности. Вы можете получить полный список доступных параметров выполнения, введя слово usage в консоли Rebol. Затем, как только вы добавили строку shebang и снова сохранили файл, вы должны сделать скрипт исполняемым, введя команду оболочки chmod + x programme1.r.

У вас также есть возможность запустить скрипт прямо из консоли Rebol. Слово do можно использовать для запуска скрипта. Требуется имя сценария для выполнения в качестве параметра; при необходимости вы можете указать путь доступа к скрипту. В Rebol файл - это тип данных, распознаваемый интерпретатором, поскольку он начинается с символа "%". По этой причине, если вы хотите запустить наш сценарий и он хранится в каталоге /home/olivier, вы набираете do %/home/olivier/programme1.r в консоли Rebol.

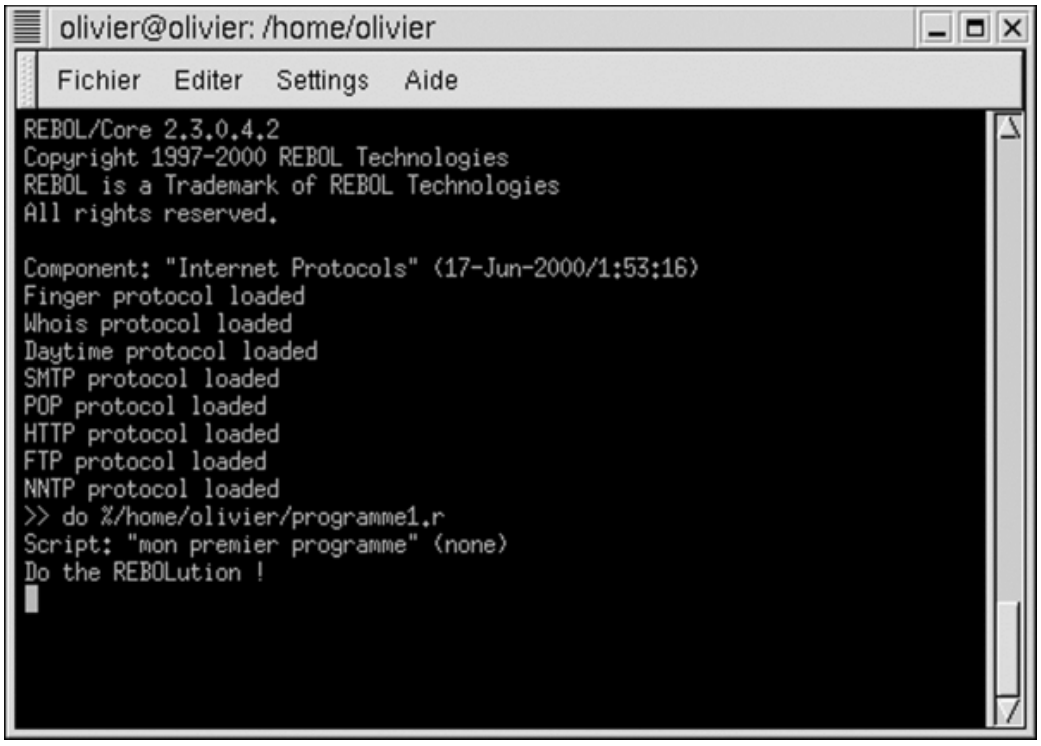

Рисунок 2-1. *Запуск скрипта в консоли.*

6.2 Переменные и типы данных

После того, как мы увидели, как использовать консоль и структуру скрипта Rebol, мы теперь создадим наши первые слова. Программирование Rebol заключается в расширении словаря слов, которые представляют функции, объекты и переменные.

6.2.1 Объявление переменной

Создать слово в Rebol очень просто. Все, что вам нужно сделать, это добавить в конце символ ":" и указать его значение. Таким образом, выражение var: 0 добавляет в словарь слово с именем var со значением 0. Поскольку язык Rebol нечувствителен к регистру, мы могли бы также использовать Var, VaR и т.д. Если затем ввести слово var в консоль, мы получим значение 0, поскольку Rebol оценил содержание слова, которое теперь составляет часть словаря в его глобальный контекст. Однако, если вы используете слово what, кажется, что слово var отсутствует в словаре. Ответ на этот вопрос очевиден: что не отображает переменные. Есть способ проверить, действительно ли была объявлена переменная; используйте функцию справки Rebol: help var (или ее сокращение? var).
Это не только подтверждает, что слово было создано, но также показывает, что тип его значения является целым числом и его значением (0). Вас ничего не удивляет? По какой причине Rebol полностью объявил нашу переменную, когда мы не указали тип? Объяснение состоит в том, что при отсутствии конкретного объявления интерпретатор выбирает тип данных, который кажется подходящим.
Другая концепция может и, возможно, может шокировать многих пуристов: слово не имеет фиксированного типа; он предполагает тип своего содержимого и вполне может меняться в процессе выполнения. Это означает, что наша переменная вполне может быть целым числом, когда скрипт начинает работать, затем на полпути изменяется на символьную строку и, наконец, в конечном итоге становится реальным числом (с плавающей точкой). Во всех этих преобразованиях есть элемент опасности, но все они возможны. Позже вы увидите, насколько полезной может быть эта потенциальная опасность, когда вы научитесь её приручать.

6.2.2 Определение типа переменной

Если во время выполнения скрипта мы хотим узнать тип содержимого переменной, мы используем слово type?. Введите type? var в консоли и она выдаст integer! (целое число), что имеет смысл. Теперь давайте попробуем изменить содержимое var, введя var: 0.82 и таким же образом проверив его тип. Теперь у нас получается decimal! (десятичная дробь). Тип нашего слова теперь правильно изменился одновременно с изменением его содержания с целого числа на число с плавающей запятой.

6.2.3 Использование конструкторов

Использование конструкторов Rebol позволяет указать тип данных слова. Вы, несомненно, видите, что вам не обязательно их использовать, но их использование настоятельно рекомендуется, чтобы помочь понять и поддерживать длинные скрипты. Предположим, мы хотим определить тип нашего слова var как date!, синтаксис для этого - var: make date! 01.01.2000. Не удивляйтесь направленности этого выражения, слово var обязательно будет иметь тип date! и инициализируется 1 января 2000 года. Также не забывайте, что его тип может постоянно меняться во время выполнения.
Практическое использование конструкторов заключается в преобразовании типов. Например, var: make integer! 3.2 дает нам возможность извлечь целую часть числа (слово var будет содержать значение 3 и иметь тип integer!).

6.2.4 Простые типы данных

В этой категории мы найдём основные типы данных, присутствующие в любом языке программирования. Мы уже встречали integer! (целое), decimal! (десятичное) и date! (дата). Rebol также включает типы time! отвечающий за время, money! за денежные суммы, logic! для логического (true (истина) или false (ложь), on (включено) или off (выключено), yes (да) или no (нет)), char! для одиночных символов и none! (ничего) что указывает на отсутствие ценности.

Следующие примеры показывают эти разные типы:

```
int: make integer! 0
float: make decimal! 2.98
hour: make time! 15:35:00
date: make date! 1/1/2000
cash: make money! $10
boole: make logic! true
character: make char! #"A"
undefined: make none!
```

6.2.5 Сложные типы данных

Сложные типы данных Rebol - это серии, состоящие из простых типов. Это инструменты, которые наделяют Rebol его мощью, позволяя программисту легко обрабатывать блоки хранения данных, пути доступа к файлам, URL-адреса или адреса электронной почты. Rebol - действительно язык, адаптированный к повседневным потребностям разработчиков клиент-серверных, многоуровневых и веб-приложений. В Rebol есть более пятнадцати сложных типов данных, которые наверняка удовлетворят все ваши потребности.
Для символьных строках есть тип данных string!. Строки обычно заключаются в двойные кавычки, за исключением случаев, когда строка включает символы возврата каретки (Enter) или двойные кавычки, тогда вы должны заключить строку в фигурные скобки {}.
Эта способность справляться с этими специальными символами действительно ценится веб-программистами, которые могут напрямую вставлять свой HTML в скрипты Rebol:

```
codehtml: {
 <html>
 <head></head>
 <body>
 My HTML page.
 </body>
 </html>
}
```

Для двоичных данных у нас есть проницательный тип binary!, который позволяет нам указать, в какой базе хранятся наши данные. Предположим, мы хотим присвоить нашему слову байт в двоичной системе исчисления, мы должны использовать данные объявления: data: make binary! 2#{01111111}, значение 127.
Имена файлов и пути доступа представлены типом file!. Они всегда начинаются с символа %. Если мы введём в консоль f: %file.txt, тип выражения? f сообщает нам, что у f есть тип file!. Мы всегда можем преобразовать символьную строку в имя файла с помощью конструктора. Используемый синтаксис: f: make file! "file.txt".
Для сети, как для Интернета, так и для интранетов, Rebol предлагает множество типов. Объявление адреса электронной почты - это просто my-email-address: make email! olivier@domaine.fr. Пути доступа к веб-ресурсам (Uniform Resource Locators (унифицированные указатели ресурсов)), как вы, наверное, догадались, представлены типом url!. Выражение web: make url! http://site.domaine.org - это то, как мы назначаем адрес слову. Мастера TCP/IP по достоинству оценят наличие типов tuple! и port!, которые представляют IP-адрес и сокет соответственно.

6.2.6 Блоки

Блоки данных также представляют собой сложный тип данных и являются одной из ключевых концепций Rebol. Они лежат в основе всех компонентов языка. В Rebol все это блоки. Заголовок сценария - это один, данные и инструкции сгруппированы в них, блоки содержатся внутри блоков, а сценарий - это группа блоков. Для программистов Rebol мир - это блок!!!

Если серьёзно, блоки содержат структуры для хранения информации, заключённые в квадратные скобки, то есть [и]. Есть несколько различных конструкторов, с помощью которых можно определять блоки. Главный из них - block! и он работает со списком значений разных типов, организованным без каких-либо ограничений или фиксированных форматов. Таким образом, мы можем смешивать различные типы данных Rebol и, конечно же, включать и другие блоки. Следующий пример полностью верен:

```
myblock: [
    "abcd"
    olivier@domaine.fr
    [ 1 2 3 ]
    %file.txt
    [ 255.255.255.0 [ 80 129 ] ]
]
```

По умолчанию Ребол сделает block! в список значений, если тип не указан. Для больших, часто просматриваемых блоков информации Rebol также предоставляет тип hash!, позволяющий оптимизировать поиск данных.

6.3 Списки обработки

Надеюсь, вы быстро поймёте, что списки являются основой Rebol, поскольку они позволяют хранить данные и управлять ими. Хорошее их понимание необходимо для правильного использования языка.

6.3.1 Массивы

Вероятно, вас не удивит, что в Rebol массив - это список. Слово array позволяет определять массив из n элементов. Введите mytab: array 5 в консоль Rebol.
Вы только что определили массив из пяти элементов; каждый инициализирован значением none. Если вы хотите инициализировать каждый элемент массива целым числом 0, вы можете использовать уточнение /initial (начальное значение):

```
myarray: array/initial 5 0
```

Уточнения предоставляют варианты слов. Фактически, они позволяют передавать в слово любое количество параметров. Дополнительные параметры записываются в той же последовательности, что и уточнения, позволяющие их использовать. Позже мы увидим, как определять уточнения своими словами.
Также можно создать массив с несколькими измерениями, используя список с массивом слов. Предположим, нам нужен массив размером два на пять, вам нужно только ввести:

```
myarray: array [ 2 5 ]
```

Массивы в Rebol - это просто списки, оба обрабатываются одними и теми же словами. Прежде чем мы продолжим, я хотел бы, чтобы вы создали следующий список в консоли Rebol:

```
colours: [ 1 "red" 2 "green" 3 "blue" ]
```

6.3.2 Навигация по серии

Список можно рассматривать как базу данных. Указатель используется для перемещения по различным элементам списка. Он отмечает начальную точку, с которой можно читать данные. Слова head (голова) и tail (хвост) соответственно позволяют помещать его в начало или конец ряда, а затем и назад позволяют продвигаться вперёд или назад в пределах ряда. Чтобы узнать, находимся ли мы в начале или в конце серии, мы можем использовать слова head? и tail? которые возвращают логическое значение. В следующем примере указатель помещается в начало цветовой серии, перемещается на один элемент вперёд и проверяется, нашли ли мы последний элемент.

```
colours: head colours
colours: next colours
tail? colours
```

Слово index? позволяет узнать, на каком элементе находится указатель. Также возможно разместить его прямо в заданной позиции со словом at. Новое положение относится к существующему положению указателя в серии. Если мы теперь введём colours: at colours 1, мы получим не значение 1, а строку символов "red", которая является элементом, с которым был выровнен указатель в то время, когда мы ввели фразу.
Точно так же length? возвращает длину списка из позиции, записанной в указателе. (Между прочим, длина списка в Rebol - это просто количество элементов в списке).

6.3.3 Доступ к элементу

Доступ к элементу в списке очень интуитивно понятен. Сначала вернём наш указатель на первый элемент в списке с colours: head colours. Теперь мы можем использовать слова first (первый), second (второй), third (третий), fourth (четвертый), fifth (пятый), sixth (шестой), seventh (седьмой), eighth (восьмой), ninth (девятый) и tenth (десятый). Если нам нужен второй элемент списка, просто введите second colours. Этот метод идеально подходит для первых десяти элементов списка, но что произойдёт, если в нашем списке их больше десяти? У нас есть возможность получить элемент в определённой позиции в списке. Для этого есть две альтернативы: colours/2 или pick colours 2. Эти две инструкции будут отображать второй элемент списка colours.

6.3.4 Добавление и удаление элементов

Добавление элемента или другого списка к существующему списку может быть выполнено путём вставки или конкатенации с использованием слов insert и join. Предположим, мы хотим добавить блок [4 "yellow"] в наш список, мы можем выбрать один из двух методов: colours: join colours [4 "yellow"] или colours: insert tail colours [4 "yellow"]. Слово join позволяет объединять элементы, в то время как в случае insert добавляет блок в конец блока цветов. Это потому, что мы перемещаем указатель в конец цветов со словом tail. Как и следовало ожидать, если мы хотим вставить ещё один блок в начало colours, все, что нам нужно сделать, это colours: insert head colours [0 "white"].
Слово remove позволяет удалить из списка элемент, на котором находится указатель. Если мы хотим удалить второй элемент от указателя, мы просто вводим remove at colours 2.

6.3.5 Изменение серии

Наряду с выбором, у нас есть поддержка, которая вернёт приятные воспоминания старым программистам на Basic. poke позволяет вам изменить значение на месте в серии. Как и pick, poke index 1 всегда указывает на первый элемент серии от текущего указателя. Выражение poke colors 2 "violet" заменит строку "white" на строку "violet" (при условии, что вы вставили 0 и "white" в начало списка и текущий указатель указывает на заголовок списка) . Слово change также имеет тот же эффект: change at colours 2 "white" возвращает исходное значение в наш список.

6.3.6 Поиск и сортировка серий

В Rebol есть два слова для поиска элементов в списке. Если элемент не найден, эти два слова возвращают значение none. find возвращает остаток списка, начиная с позиции искомого элемента. Если вы введете find colours "blue", интерпретатор отобразит ["blue" 4 "yellow"].
Слово select ведёт себя по-другому, потому что оно возвращает элемент после искомого. select colours 4 даёт нам значение "yellow". select идеально подходит для поиска в файле конфигурации, а find полезен для поиска в базе данных, хранящейся в памяти.
У нас также есть возможность сортировать содержимое списка в порядке возрастания с помощью слова sort. (Rebol даже позволяет программисту заменить стандартный алгоритм сортировки своим собственным).

6.3.7 Копирование и очистка серии

Слово copy позволяет копировать одну серию в другую. Все существующие данные в блоке назначения будут потеряны. Выражение mycolours: copy colors копирует элементы colours в списке mycolours. Почему недостаточно выполнить операцию "mycolours: colors"?
Выполните простой тест, создав список a, содержащий [1 2 3 4]. Теперь присвойте a переменной b выражение b: a. Если вы измените первое значение a на 10 (a/1: 10), а затем наберёте b в консоли, чтобы найти его значение, вы заметите, что первый элемент b также стал 10. Почему? Ответ прост: при назначении одного списка другому они фактически используют одно и то же пространство в памяти. По этой причине изменение записи в одном из них также изменяет его в другом. При использовании такого подхода необходимо соблюдать осторожность, и лучше всего рассмотреть возможность использования слова copy, которое создаёт реальную копию одного списка в другом (в данный момент я хотел бы напомнить вам, что символьная строка также является list!). В этом случае второй список содержит те же элементы, что и исходный, но имеет собственное пространство в памяти и они становятся полностью независимы.
Очистка серии выполняется с помощью слова clear, за которым следует имя списка. Слово, используемое для названия списка, не удаляется из словаря, а очищается от его содержимого. Если вы хотите полностью удалить слово, вы можете использовать слово unset.
Многие из слов, которые мы только что видели, имеют много уточнений, которые значительно облегчают жизнь программисту. Подробное рассмотрение всех этих вопросов здесь заняло бы слишком много времени. Я предлагаю вам воспользоваться справкой в консоли, которая позволит вам изучить все доступные варианты. Постарайтесь полностью понять, как работать со списками; эти слова действительно составляют основу языка.

6.4 Управляющие структуры и циклы

Обнаружив некоторые возможности Rebol по обработке данных, мы теперь рассмотрим управляющие структуры и циклы, доступные в языке. Что удивительно при первом просмотре, так это огромное количество опций, доступных программисту. Редко встречаются языки, предлагающие такую свободу.

6.4.1 Тесты в Rebol

В программировании тесты позволяют изменять путь выполнения программы в соответствии с их оценкой. Схема проста: если условие истинно [сделайте что-нибудь], если нет [сделайте что-нибудь ещё].
Rebol использует эту очень традиционную модель. Этой операции присвоено слово if. Если оценка его первого параметра возвращает логическое значение true, то код, помещённый во второй параметр, будет выполнен. В тестах используются классические операторы, такие как, <>, >, <=, >=, а знак равенства также позволяет сравнивать символьные строки. В отличие от многих других языков, вам не нужно заключать свои тесты в круглые скобки, но опыт показывает, что это может помочь улучшить читаемость программы. Следующая программа отображает "OK", когда пользователь вводит число 0:

```
REBOL [ ]
nbr: to-integer ask "a number ?"
if ( nbr = 0 ) [ print "OK" ]
```

Слова all и any представляют собой краткую форму логических операций И и ИЛИ. При all тест возвращает истину, если все его условия верны. С другой стороны, any возвращает true, если любое из его условий истинно. В следующем примере отображается "ALL true", если обе переменные a и b равны 0, затем отображается "ANY true", когда он обнаруживает, что c не равно 0, но что a равно 0:

```
REBOL []
set [ a b c ] [ 0 0 1 ]
if all [ (a = 0) (b = 0) ] [ print "ALL true" ]
if any [ (a = 0 ) ( c = 0 ) ] [ print "ANY true" ]
```

(Вы заметили, что мы использовали слово set для быстрого создания и инициализации списка переменных? Почему бы не поискать его в консоли Rebol, чтобы узнать о нем больше.)

6.4.2 А "иначе" (either)?

До сих пор мы видели, что происходит только тогда, когда тест оказывается "верным". Но, как правило, мы также должны управлять другим случаем: когда результатом теста является логическое значение false (ложное). В Rebol у нас есть способы сделать это. Прежде всего, мы можем использовать уточнение else, доступное для слова if. Первый блок кода выполняется, если условие возвращает истину, в противном случае оценивается второй блок. В следующем коде мы отображаем истину или ложь в зависимости от того, содержит ли переменная a символьную строку "Rebol" или нет:

```
REBOL []
a: copy "Rebol"
if/else ( a = "Rebol" ) [ print "true" ] [ print "false" ]
```

Мы также можем использовать слово either (либо), что на самом деле больше соответствует духу языка. Немного истории, уточнение другого слова, если бы программисты Rebol просили в списке рассылки. Первоначально только любая из них допускала проверку формы IF… THEN… ELSE. Очень просто и стильно (правда, дело вкуса!):

```
REBOL []
a: copy "Rebol"
either ( a = "Rebol" ) [ print "true" ] [ print "false" ]
```

6.4.3 Множество вариантов

С помощью слова switch вы можете выборочно выполнять части кода; выбор кода для выполнения осуществляется на основе значения переменной. Уточнение по умолчанию позволяет обрабатывать случаи, когда значение не соответствует ни одному из указанных "

```
Rebol [ ]
choice: to-integer ask "What is your choice (1 – 3)?" switch/default choice [
 1 [ print "choice 1" ]
 2 [ print "choice 2" ]
 3 [ print "choice 3" ]
] [ print "This choice was not expected" ]
```

6.4.4 Loop

Слово loop повторяет последовательность кода указанное количество раз. Конечно, указанное число должно быть положительным целым числом. Первый параметр цикла - это количество повторений, а второй - блок, содержащий инструкции Rebol, которые необходимо повторить. В нашем примере вас спрашивают, сколько раз программа должна отображать "Hello" на экране, а затем выполняет операцию.

```
REBOL [ ]
nbr: to-integer ask "Number of repetitions ?" loop nbr [ print "Hello" ]
```

6.4.5 Семейство циклов for

forever (навсегда) - самое простое из этой группы слов. Он просто постоянно выполняет код, переданный ему в качестве параметра: forever [print "Останови меня, нажав ESC!"].
Слово for запускает часть кода определённое количество раз, но на этот раз программист может использовать счётчик для цикла и контролировать его приращение. for - это слово, которое требует не менее 5 параметров:

- Переменная, действующая как счётчик,
- Значение счётчика в начале,
- Значение счётчика на финише,
- Сумма, которую счётчик должен увеличивать на каждой итерации (круге),
- Код Rebol, который необходимо выполнить.

Следующая программа вставляет в список десять целых чисел (от 1 до 10).

```
REBOL [ ]
list: copy []
for i 1 10 1 [ append list i ]
```

Слова forall, forskip и foreach чрезвычайно полезны, поскольку они позволяют перемещаться по элементам списка. На каждой итерации цикла forall перемещает указатель на следующий элемент в списке. Это упрощает отображение содержимого нашего списка следующим образом:

```
forall list [ print first list ]
```

forskip обходит серию, пропуская указанное количество элементов, пока не достигнет конца списка. Это отображает каждый второй элемент нашего списка:

```
forskip list 2 [ print first list ]
```

После того, как forall и forskip выполнят свою работу, указатель списка будет найден в конце списка. Чтобы выполнить больше действий в списке, необходимо вернуть указатель в начало с помощью слова head.
foreach также просматривает каждый элемент списка, но присваивает значение указателя переменной:

```
foreach value list [ print value ]
```

В отличие от forall и forskip, использование foreach не требует возврата указателя списка в начало списка. Он остаётся в исходном положении.

6.4.6 Repeat, until и while

Слово repeat позволяет повторять оценку блока кода заданное количество раз, переменная используется в качестве счётчика для цикла, который фиксируется на начало с 1 и увеличивается на единицу для каждой итерации цикла. Эта переменная не принадлежит глобальному контексту и не существует вне блока кода, который оценивается. В следующем примере отображается 1 2 3, несмотря на то, что i инициализировано значением 5 в начале программы.

```
REBOL []
i: 5
repeat i 3 [ print i ]
print i
; когда цикл закончился, i по прежнему имеет значение 5!
```

Два слова until (до) и while (пока) разрешают выполнение программной последовательности неопределённое количество раз. Разница между этими двумя словами заключается в том, что until оценивает код до тех пор, пока последняя оценка в блоке не вернёт логическое значение true, тогда как while оценивает код, пока предоставленное условие истинно. (Фактически, until продолжает оценивать код до тех пор, пока последняя оценка в блоке не возвращает false или none). В следующем примере используются два метода; цель состоит в том, чтобы увеличивать числовое значение, пока оно не достигнет числа 10:

```
REBOL []
i: 0
until [
    i: i + 1
    ( i = 10 )
]
i: 0
while [ i < 10 ] [ i: i + 1 ]
```

6.4.7 Простая игра в Rebol

Теперь мы собираемся написать короткую программу Rebol, и в этом случае мы начнём с простой игры. Нужно угадать число, загаданное компьютером. Игра подсказывает нам, сообщая больше или меньше задуманное ей число.

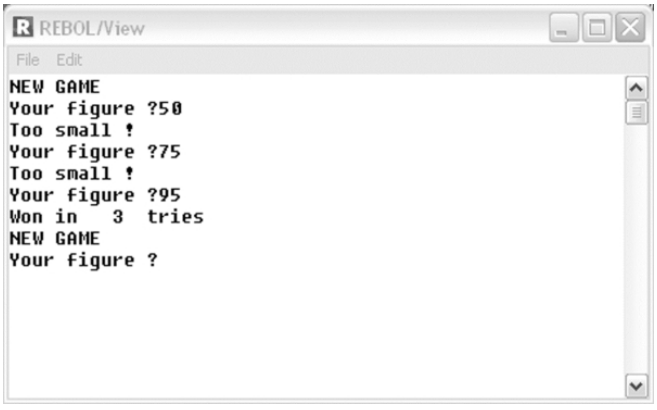

Рисунок 2-2. *Запускаем игру.*

Логика состоит из двух встроенных циклов, которые соответственно выполняют код на неопределённый срок и просят игрока угадать, пока он не найдёт правильное число. Как только предположение окажется правильным, логический end принимает значение true, что останавливает выполнение цикла, определённого словом until. Секретное число получается из слова random, за которым следует максимальное значение. В нашем случае это число находится в диапазоне от 0 до 100. У нас также есть счётчик, инициализированный на 0 в начале попытки, который показывает, сколько угадываний потребовалось игроку, чтобы выиграть. Он увеличивается на 1 для каждого предположения.

```
REBOL []
forever [
  value: random 100
  counter: 0
  end: false
  print "NEW GAME"
  until [
    nbr: to-integer ask "Your figure ?"
    counter: counter + 1
    either (nbr <> value) [
      either (nbr < value) [ print "Too small !" ] [ print "Too big !" ]
    ] [
      print [ "Won in  " counter " tries" ]
      end: true
      ]
    end
  ]
] ;; Замечание - нажмите Esc когда надоест играть.
```

6.5 Функции и объекты

Программирование в Rebol состоит из определения слов с целью расширения словаря. Эти слова могут представлять три разных компонента языка. Вы уже знаете переменные, роль которых заключается в хранении данных. Два других типа позволяют разрабатывать более важные приложения; они функции и объекты.
Эти типы слов позволяют структурировать код, чтобы облегчить читаемость и его повторное использование.
Функция - это группа инструкций, которые выполняют одну или несколько определённых задач и обычно выполняются несколько раз при выполнении программы. Объект содержит как методы, так и свойства, то есть функции и переменные соответственно. Функции - это строительные блоки Rebol; вы эффективно расширяете язык, добавляя функции.
Одна из характеристик объекта - вести себя как чёрный ящик, который хорошо выполняет определённую роль и который можно многократно использовать для экономии работы. Объект - это многократно используемый компонент; его пользователю не нужно знать его внутреннюю работу, только различные точки входа, которые известны как свойства, общедоступные методы и точки выхода.

6.6 Использование функций и объектов

Хотя функция обычно очень зависит от среды, в которой она выполняется, объект, с другой стороны, представляет собой интегрированный модуль, который можно использовать в любом проекте, способ извлечь выгоду из существующего кода, который был протестирован и является надёжным.
Объектно-ориентированное программирование позволяет рассматривать крупномасштабные разработки и командную работу: каждый член команды отвечает за чётко определённую часть программы.
Как и в других языках, функции можно сгруппировать в библиотеки. Объекты могут включать в себя другие объекты, а также свойства и методы, что облегчает интуитивное построение иерархий объектов. Также важно сразу усвоить ключевую концепцию Rebol: контексты. Мы уже видели, что Rebol позволяет определять объем слова. По умолчанию все слова в словаре

принадлежат глобальному контексту; они известны и могут использоваться всеми другими словами в том же контексте. С другой стороны, с помощью слова use (использование) можно определить контексты оценки, отличные от глобального контекста. В этом случае слово не может использовать слово, принадлежащее другому контексту. Как слова определяются в функциях или регистрируются объекты? Будьте очень осторожны с вашими первыми программами Rebol, потому что, в отличие от многих других языков, все переменные, определённые в функции, если они специально не указаны как принадлежащие к локальному контексту, автоматически помещаются в глобальный контекст скрипта.
Противоположное верно для объектов, которые предназначены для обеспечения независимости и модульности и автоматически имеют свой собственный контекст. Метод или свойство существуют только внутри самого объекта. Любые ссылки на эти элементы должны указывать на объект, который их содержит.

6.6.1 Определение функций

Rebol практически не делает различий между кодом и данными. Функция - это просто разновидность данных, что неудивительно, типа function!, который принимает в качестве параметров список слов, соответствующих значениям, отправляемым в функцию при её вызове, и код для оценки. Мы можем определить функцию под названием square, которая, как вы уже догадались, возвращает квадрат числа следующим образом:

```
square: make function! [ number ] [
    number * number
]
```

Первый блок содержит список параметров, второй - последовательность кода, который будет оценивать функция. Наша функция включена в глобальный словарь и теперь может использоваться для всех слов языка. Который мы вызываем, просто набрав square 4 в консоли.
Значение, возвращаемое функцией, всегда является результатом последней оценки, выполненной функцией. Если вы хотите принудительно выполнить возврат из функции, то есть вернуть значение и прекратить выполнение функции, вы можете использовать слово return.

```
square: make function! [ number ] [
    return number * number
]
```

Если вы хотите, вы можете определить переменные, принадлежащие только контексту функции, используя слово use. Любое слово, определённое таким образом, будет существовать только внутри функции:

```
square: make function! [ number ] [
 use [ result ] [
  result: number * number
 ]
 return result
]
```

Слово function упрощает нашу работу. Это слово позволяет вам определить за один шаг входные параметры, локальные переменные и уточнения, с помощью которых вы можете ввести дополнительное поведение в функцию. Он также позволяет вам указать типы данных параметров и предоставить информацию о функции, которую можно найти с консоли, используя слово help.

```
square: function [
 "calculate the square of a number"
 number [ integer! ] "the input - the number to be squared"
 /increment n "increments the result with the value n"
] [
 result [ integer! ]
] [
 result: number * number
 if increment [ result: result + n ]
 return result
]
```

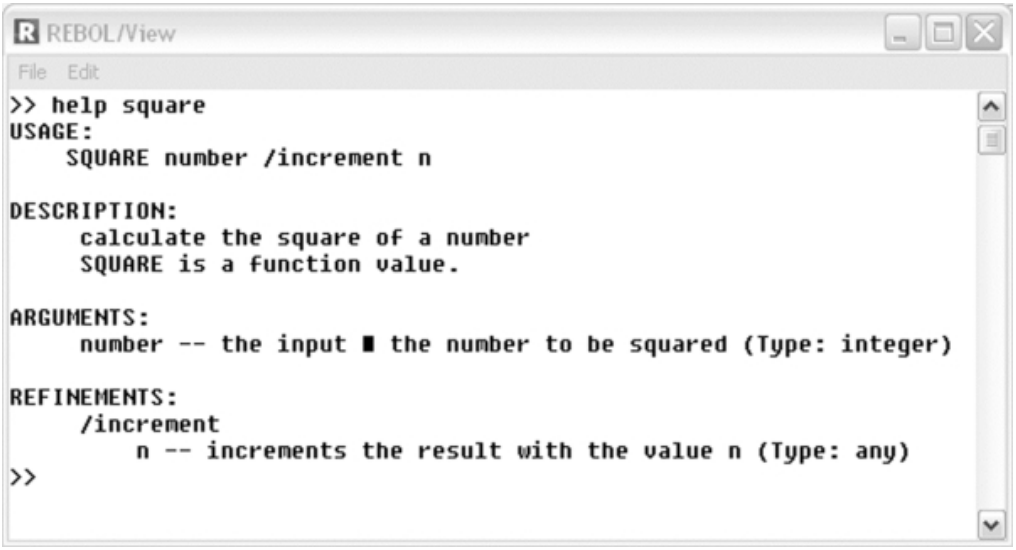

Рисунок 2-3. *Помощь позволяет изучать слова.*

6.6.2 Создание объектов

Определение и использование объектов в Rebol чрезвычайно интуитивно понятно. У объекта тип данных object!. Его блок инициализации просто содержит методы и свойства объекта. В определении свойства используются те же правила, что и для переменной. Написание метода аналогично написанию функции. Объект может также содержать другие объекты.
Принципиальная семантическая разница между объектами в Rebol и многих других языках заключается в том, что вы используете символ "/" вместо "." для указания пути доступа к свойству или методу. Однако он похож на другие языки в том смысле, что объект может ссылаться на себя с помощью слова self.
Мы собираемся определить объект computer, свойство user которого определяет имя пользователя. Подобъект hardware (аппаратное обеспечение) позволяет определять характеристики компьютера, а методы on и off предназначены для включения и выключения машины.

```
computer: make object! [
 user: make string! "Olivier"
 hardware: make object! [
  processor: make string! "Alpha"
  memory: make integer! 32
 ]
 on: make function! [] [ print "I'm switched on" ]
 off: make function! [] [ print "I'm switched off" ]
```

В Rebol объект можно сразу же использовать. Вы можете сразу получить доступ к объекту, не выполняя процесс создания экземпляра, вы работаете напрямую с объектной моделью (программирование по прототипу). Вы можете заменить или изменить значение свойства computer/user, методы обращаются с синтаксисом computer/on или computer/off. Аппаратному обеспечению объекта необходимо полное описание пути доступа, например, computer/hardware/processor. Вы также можете создать экземпляр объекта следующим образом: my-machine: make computer [].
Теперь давайте создадим машину, которую использовал Николас и которая была оснащена сетевой картой:

```
server: make computer [
 user: "Nicholas"
 hardware: make object! [
  processor: "X86"
  memory: 128
  network: make logic! true
 ]
]
```

Там мы унаследовали все свойства объекта computer и добавили новое свойство к унаследованным. Как видно из следующего кода, вы также можете динамически создавать списки объектов:

```
userlist: [ "Olivier" "Nicholas" "Damien" ]
machines: []
foreach the-user userlist [
 append machines make computer [ user: the-user ]
]
```

Впоследствии, если вы захотите найти или изменить один из объектов в списке, вам нужно будет определить объект, который будет использоваться в качестве указателя на объект в списке:

```
ptr: second machines ptr/user: "Natalie"
```

Рисунок 2-4. *Содержимое объектов отображается probe.*

6.7 Парсинг и диалекты

Теперь мы рассмотрим два дополнительных аспекта языка: синтаксический анализ и диалекты. Если пользователи Perl и JavaScript уже знают первый, второй является новым и является одной из сильных сторон Rebol. Сочетание возможностей обработки символьных строк и способности определять свои собственные диалекты, настоящие языки сами по себе, делают Rebol уникальным и глубоко выразительным инструментом.

6.7.1 Искусство обработки символьных строк

Ни для кого не секрет, что обработка символьных строк повторяется, болезненна и утомительна. Мы все регулярно сталкиваемся с написанием алгоритмов, необходимых для извлечения из данных, содержащихся в плоском файле или HTML-коде страницы, для проверки соответствия строки точной спецификации и т.д. Большинство традиционных языков предлагают программисту только ограниченную функциональность для добавления символов или искать одну строку в другой. К счастью для нас, некоторым разработчикам надоело бесконечно повторять эти интеллектуальные искажения.
Так они и решили, что обработка строк может быть смоделирована с помощью набора правил: это техника синтаксического анализа. Perl Ларри Уолла, изначально предназначенный для извлечения данных и создания отчетов, является одним из самых известных языков в области синтаксического анализа. С помощью нескольких символов можно определить действие, которое можно применить ко всем символам в строке.

6.7.2 Rebol парсинг

Rebol - это язык обмена сообщениями. Его основная функция - получение, обработка и передача информации. В Интернете большая часть данных хранится в HTML-страницах или XML-документах, которые являются текстовыми файлами. Логично, что у Rebol есть инструмент для анализа и извлечения информации, содержащейся в символьных строках. Видение парсинга в Rebol намного современнее, чем в большинстве других языков. Речь идет не о вводе непонятных комбинаций символов, а об использовании истинного языка, задуманного для этого типа операций: диалекта.

6.7.3 Парсинг с использованием диалекта

Диалект - это особый язык, интегрированный в Rebol. Это язык в языке. Он посвящен конкретной задаче и только этим занимается. Если вы встречали термин "язык предметной области", то диалект Ребола является одним из них.
Вскоре мы рассмотрим создание графических интерфейсов с помощью Rebol/View. Мы используем VID (Visual Interface Dialect) для определения графических компонентов приложения. Вероятно, это первый диалект синтаксического анализа, который вы увидите, хотя вы должны знать, что Rebol включает несколько других диалектов.
Что так хорошо в диалектах, так это то, что мы можем не только использовать мощные конкретные языки в их области, но и определять свои собственные. В Rebol даже можно реализовать интерпретатор BASIC, позволяющий объединить два языка в одной программе. Фактически, Джон Никласен сделал именно это и реализовал интерпретатор BBC Basic в Rebol. Вы можете найти его на http://www.fys.ku.dk/~niclasen/rebol/bbcbasic.r
Но это не все. Диалекты интереснее: они позволяют разделить приложение на четыре части:

- движок, использующий диалект,
- конфигурация приложения,
- визуальные аспекты приложения (графические компоненты),
- функциональность приложения (правила управления, справочные описания и т.д.).

Если в текущих приложениях у нас есть возможность параметризовать программное обеспечение (максимальные значения, значения по умолчанию и т.д.), Благодаря диалектам, теперь мы можем извлечь логику приложения из его тела. Настройка параметров теперь распространяется на его внутреннюю логику с его поведением. Тогда основной трудностью становится определение специализированного диалекта, который будет достаточно универсальным.

6.7.4 Небольшой разбор

В Rebol вы используете слово parse для выполнения операции синтаксического анализа символьной строки. Двумя параметрами для этого слова являются символьная строка, которой нужно управлять, и блок правил обработки, которые необходимо применить. Если операция состоит только из разделения строки на основе указанного разделителя символов, мы предоставляем вместо блока только разделитель или строку символов, содержащую более одного разделителя символов. Использование значения none в качестве второго параметра указывает, что строка должна быть разделена пробелами и другими стандартными разделительными символами, такими как запятая.
Предположим, у нас есть символьная строка txt, содержащая "Несколько слов на Rebol ". Мы можем отделить отдельные слова друг от друга с помощью выражения parse txt none. Мы получаем блок, который содержит разные строки символов для каждого слова: ["Несколько" "слов" "на" "Rebol"].
Для нашего второго примера у нас есть строка txt, содержащая символы "abcdefg". Мы можем разделить строку, используя символы "b" и "e" в качестве разделителей. Все, что нам нужно сделать, это использовать parse txt "be", чтобы получить желаемый результат: ["a" "cd" "fg"].
Также можно проверить соответствие строки определенной модели. Следующая строка содержит только две последовательности символов "xx"?

```
txt: "xx xx"
parse txt [ 2 "xx" ]
```

Там мы использовали правило, проверяющее наличие двух строк "xx" внутри другой строки. Слово parse затем возвращает логическое значение: true, если строка соответствует правилу синтаксического анализа, в противном случае - false.
Мы также можем извлекать данные из строки с помощью правил. Например, если мы хотим извлечь символы между "xx" и "yy":

```
txt: "some xx words yy on Rebol" parse txt [
 thru "xx"
 copy extract
 to "yy"
 ( print extract )
]
```

Здесь мы разделили содержимое txt от "xx" до "yy". Результат сохраняется в extract, который отображается с помощью print. Диалект синтаксического анализа позволяет выполнять код Rebol внутри его собственного кода.

6.7.5 Определение диалекта

Фактически создание диалекта означает определение правил анализа. Если синтаксис правильный, инструкции диалекта выполняются. Например, мы можем настроить диалект с именем cursor для управления курсором консоли Rebol. Мы определим две инструкции:

- cls - очищает экран
- at - позициях курсора в позиции, обозначенной значением типа данных pair! (пара) (например, 3x5)

Правила просты. Если Rebol находит слово cls, экран очищается. В противном случае, если слово at найдено, за ним должно следовать значение пары типа pair!. Правила эксклюзивные. Выполняемая инструкция - одна из тех, которые были определены (любая), но есть только одна инструкция, которая относится к каждому слову в диалекте.

Для этого мы используем символ "|", соответствующий "or" (или). Функция с именем cursor

принимает блок кода в качестве параметра для анализа и разбора. При использовании слова compose все выражения, заключенные в круглые скобки, оцениваются и заменяются их значением. Эта функция позволяет нам вставлять переменные или код Rebol в код, написанный на этом диалекте.

```
REBOL [
 Subject: "cursor dialect"
 Author: "Olivier Auverlot"
]

cursor-rules: [
 any [
  'CLS (
   print "^(1B)[J"
  ) |
  'AT set pos pair! (
   prin join "^(1B)[" [ pos/y ";" pos/x "H" ]
  )
 ]
]

cursor: function [ code ] [] [ parse (compose code) cursor-rules ]
```

Теперь мы можем использовать наш диалект в наших скриптах Rebol. В следующем примере экран очищается, отображается строка в верхней части экрана, а затем отображается строка из 20 символов под ней:

```
cursor [
 cls
 at 10x2
]
prin "Hello !"
p: 1x4
for x 1 20 1 [
 p/x: x
 cursor [ at (p) ]
 prin "="
]
```

Рисунок 2-5. *Проверка диалекта cursor.*

6.8 Резюме

Rebol - невероятно простой, но мощный язык. Это позволяет выполнять сложные операции с минимумом кода. Он имеет множество встроенных типов данных и облегчает написание очень структурированного кода. Он также поддерживает объектно-ориентированный подход. Наконец, его возможности в области синтаксического анализа и диалектирования существенно отличают его от других языков.

7. GUI, графика и звук

В первых главах этой книги мы в основном использовали в примерах версию Rebol/Core. Поскольку эта версия ограничена отображением текста, она не позволяет создавать действительно интересные эффекты презентации. Для создания графических интерфейсов, рисунков и генерации звуков существует версия под названием Rebol/View.
Rebol/View - это расширение Rebol/Core. Он делает все, что делает Rebol/Core, но также позволяет разрабатывать графические интерфейсы, полностью независимые от платформы. Ваше приложение не требует никаких изменений для работы в системе, отличной от той, на которой оно было разработано. На самом деле существует даже две версии Rebol/View. Первый, называемый просто Rebol/View, полностью бесплатен. Вы можете считать это графической версией Rebol/Core.
Вторая, названная Rebol/View/Pro, является коммерческой версией языка. Если вы его купите, у вас будет Rebol/View с множеством расширений, таких как:

- вызов функций из динамических библиотек (DLL под Windows, .so файлы в Linux и др.),
- мощные функции шифрования данных, такие как RSA, DH или DSA.

7.1 GCS и VID

Для отображения графических компонентов на экране Rebol/View содержит систему графического композитинга (Graphical Compositing System). Этот очень мощный движок не только позволяет отображать графику, текст и изображения, но также применять специальные эффекты, такие как градация цвета, зеркальные эффекты или изменять цветовую палитру. Это можно сделать, работая напрямую с GCS, но тогда вам придется делать все вручную. GCS представляет собой функции нижнего уровня Rebol. Лучше всего зарезервировать их использование для очень специфических приложений, таких как дизайн персонализированных графических компонентов или разработка сложных мультимедийных продуктов.
К счастью для вас, разработчики Rebol/View предвидели необходимость избегать изобретать велосипед в каждом приложении. Решение заключается в использовании диалекта под названием VID (Visual Interface Dialect). Это позволяет отображать и обрабатывать основные элементы графического интерфейса всего несколькими инструкциями. Важно помнить, что VID - это лишь один из диалектов. Отличная идея Rebol/View заключается в том, что каждый может разработать свой собственный графический диалект, адаптированный к его собственным потребностям (бизнес-приложения, видеоигры, интерактивные киоски и т.д.). VID - это общий диалект, который позволяет очень легко разрабатывать все типы приложений.

7.1.1 Основные понятия

Определение содержимого окна описывается как список настроек с именем макета. Фактически это блок инструкций для диалога VID.
Вы также можете смешивать код Rebol с объявлением различных графических элементов. Начнем с простого примера. следующий код отображает окно с текстом "Hello" внутри:

```
view layout [ vh1 "Hello" ]
```

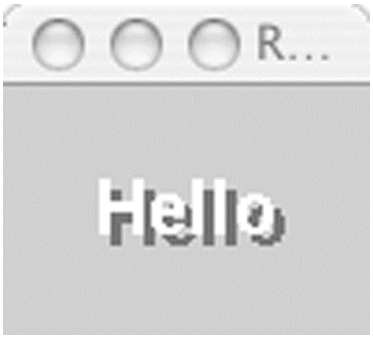

Рисунок 3-1. *Первое окно под Mac OS X.*

Для создания окна с текстовым полем внутри требуется всего одна строка кода. Теперь предположим, что вы хотите добавить кнопку "Quit", чтобы пользователь мог закрыть приложение, тогда код будет выглядеть следующим образом:

```
view layout [
 vh1 "Hello"
 button "Quit" [ quit ]
]
```

Рисунок 3-2. *Добавлена кнопка.*

Инструкции vh1 и button в официальной документации Rebol называются стилями. Вы можете перевести этот термин в слова "компонент", "графический элемент" и, возможно, даже "виджет". Rebol говорит о стиле, потому что форма и поведение таких компонентов полностью изменяются пользователем. Код для обработки событий помещается в блок кода Rebol. Этот блок кода может содержать несколько строк и функций вызова.

7.1.2 Стили

Диалект VID предоставляет ряд предопределенных графических элементов. Их семнадцать только для отображения текста. Наиболее практичными являются текст (text) для простой метки, заголовок (title) и vh1 для заголовков. Слово button (клавиша) позволяет зафиксировать, что пользователь нажал клавишу. Стиль button отображает кнопку.
Стили toggle (переключатель), rotary (поворотный) и choice (выбор) - это кнопка с двумя положениями, поворотная кнопка и раскрывающийся список соответственно.
Стили check (проверка) и radio (радио) определяют флажок и переключатель. field - это область поля для ввода.
Стили list (список) и text-list текстовый список позволяют пользователю выбрать один или несколько элементов из списка. В стиле slider (слайдера) есть слайдеры. Вы также можете использовать стандартные диалоговые окна, такие как селектор файлов, поля ввода или запрос подтверждения пользователя.

В графической области стиль image отображает форматы BMP, GIF, JPEG и PNG и применяет к ним специальные эффекты. Диалект DRAW рисует геометрические формы, такие как линии, круги и т.д. Он также обеспечивает прозрачность графических элементов, что показывает отличные возможности в области видеоигр с Rebol/View, особенно в сетевых играх.
У всех этих стилей есть свойства, позволяющие динамически изменять их внешний вид. Чтобы изменения отразились на экране, их нужно повторно отобразить со словом show.
В зависимости от типа данных VID может знать, для какого атрибута предназначен параметр:

- символ - это текст, который будет отображаться на экране,
- графические координаты (тип данных pair! (пара)) указывают размеры объект,
- tuple (кортеж) дает цвет отображаемого элемента,
- файл соответствует изображению,
- символ - это сочетание клавиш,
- блок - это последовательность кода, которая должна выполняться при получении события.

7.1.3 Макет стиля

Есть два способа разместить графические компоненты в определенном месте:

- вы указываете фиксированное положение для каждого компонента. Этот метод не рекомендуется вообще (многие платформы, на которых Rebo/View не имеют одинакового графического разрешения),
- вы определяете стратегии компоновки. Тогда вопрос уже не в том, где указать расположение элементов, а в том, как их разместить на экране. Например, across (поперек) указывает, что компоненты следует размещать горизонтально, один за другим. С другой стороны, below (ниже) запрашивает вертикальное отображение компонентов. Также можно определить панели для группирования связанных графических компонентов.

7.1.4 Конвертер доллар-евро

Следующий пример - конвертер доллар-евро. Полный код занимает всего 628 байт.

```
REBOL [
 title: "Dollar/Euro converter"
]

convert: function [ value /dollar /euro ] [ sum ] [
 sum: to-decimal value
 either dollar [
  sum: sum / 1.30
 ] [ sum: sum * 1.30 ]
 price/text: copy (to-string sum)
 show price
]
view layout [
 price: field 200x20 ""
 across
 dollar: radio of 'currency true [ convert/dollar price/text ]
 text "Dollar"
 euro: radio of 'currency [ convert/euro price/text ]
 text "Euro"
 return
 button "Quit" 255.0.0 [ quit ]
 button "Calculate" [
  either dollar/data = true [
   convert/euro price/text
  ] [ convert/dollar price/text ]
 ]
]
```

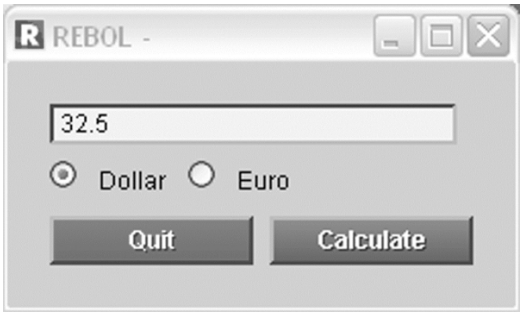

Рисунок 3-3. *Полнофункциональный преобразователь.*

Небольшой размер приложений, разработанных с помощью Rebol/View, позволяет легко загружать их по сети.

7.2 Обработка изображений с VID

Мы продолжим изучение диалекта визуального интерфейса Rebol/View, используя его графические возможности. VID не ограничивается отображением кнопок и полей ввода, он также позволяет отображать изображения, изменять их и применять к ним сложные специальные эффекты.

7.2.1 Использование изображений

Rebol/View может использовать четыре наиболее распространенных формата изображений. Прежде всего, это формат BMP от Microsoft. Совместимость с этим форматом позволяет использовать большое количество существующих изображений. В дополнение к этому стандарту Windows, Rebol/View поддерживает два наиболее широко используемых формата в Интернете, JPEG и GIF, а также новый формат PNG, задуманный для замены GIF.
Если изображение находится в одном из этих четырех форматов, достаточно одной инструкции, чтобы загрузить его в память и сделать доступным. Мы просто используем слово load, за которым следует имя файла. Путь доступа к файлу может быть либо локальным, либо URL-адресом, позволяющим получать изображение по сети. Вы должны сами обрабатывать любые другие типы изображений (что очень сложно, поэтому лучше всего использовать поддерживаемые типы, если это вообще возможно).

```
myimage: load %tuxrebol3.gif
```

Теперь изображение было преобразовано во внутренний формат Rebol в виде списка двоичных значений, который можно просмотреть с помощью mold (словарная форма) и изменить, как и любое другое значение ряда в Rebol.

7.2.2 Отображение изображения

Используя инструкцию image (изображение) диалекта VID, мы можем отображать изображение в окне на экране. Нет необходимости даже загружать изображение в память перед его отображением. Фактически, если мы предоставим имя файла в качестве параметра для изображения инструкции, оно будет извлекать изображение из файла перед его отображением. С другой стороны, если атрибут является переменной, содержащей изображение, инструкция отобразит его напрямую. Фактически, это просто зависит от того, хотим ли мы сохранить изображение в памяти. Следующий скрипт отображает изображение прямо на экране.

```
view layout [
 image %tuxrebol3.gif
]
```

Рисунок 3-4. *Отображение изображения.*

Для создания мощного визуального отображения изображений требуется всего несколько строк кода Rebol. Давайте посмотрим, насколько просто выбрать изображения в диалоговом окне и независимо отобразить их в других окнах в Rebol/View.
Создаем макет (layout), содержащий две кнопки. Первый позволяет выбрать изображение; второй закрывает приложение.
Диалог выбора файла является стандартной функцией Rebol/View и называется файлом-запросом (request-file). (Существует набор стандартных диалогов, о которых вы можете узнать, набрав запрос help request- в консоли.) Уточнение фильтра фильтрует файлы, отображаемые в диалоговом окне. Здесь пользователь может выбрать только изображения BMP, GIF, JPEG или PNG. Если пользователь нажимает "Отмена", request-filec возвращает значение none.
Поскольку у пользователя есть возможность выбрать несколько файлов с помощью клавиши CONTROL, мы должны отобразить все изображения, выбранные с помощью цикла foreach. Для каждого выбранного файла изображение отображается в окне с черным фоном:

```
REBOL [
 subject: "image display"
 author: "Olivier Auverlot"
]

view layout [
 button "Load" [
choice: request-file/filter [
"*.png" "*.gif" "*.jpg" "*.bmp"
]
  if not none? choice [
   foreach file choice [
    view/new layout [
     backdrop 0.0.0
     image file
    ]
   ]
  ]
 ]
 button "Quit" [ quit ]
]
```

7.2.3 Изменение изображений

Rebol/View может делать гораздо больше, чем просто отображать изображения, он также позволяет вам динамически изменять их и применять к ним сложные специальные эффекты.
Размер изображения можно изменить, просто предоставив слово image с размером изображения в pair! значение.
Мы можем применить цвет к изображению, просто предоставив значение RGB в виде tuple! (кортежа). Предположим, мы хотим покрасить изображение tuxrebol3.gif в красный цвет и установить его размер 50 на 50 пикселей, синтаксис будет следующим:

```
view layout [
 image 50x50 255.0.0 %tuxrebol3.gif
]
```

Теперь предположим, что мы хотим уменьшить изображение до 50% от его реального размера, затем мы должны определить его новый размер на основе его исходного размера. Для этого мы используем свойство size (размер) изображения.

```
myimage: load %tuxrebol3.gif

new-size: make pair! reduce [
  (make integer! (myimage/size/x / 100) * 50)
  (make integer! (myimage/size/y / 100) * 50) ]

view layout [ image new-size myimage ]
```

7.2.4 Применение спецэффектов

Без сомнения, наиболее впечатляющей особенностью Rebol/View является его способность применять целый ряд специальных эффектов, вдохновленных самым мощным программным обеспечением для улучшения изображений, таким как Photoshop или Gimp, не только к изображению, но и ко всем графическим компонентам (кнопкам , списки, поля редактирования,…). Таким образом, мы можем применять прозрачность, изменять цвета, изменять размер, обрезать, масштабировать, вращать, зеркально отображать, изменять яркость, создавать градации цвета или создавать эффекты рельефа. Все эти функции включены в стандартный Rebol/View и доступны на всех поддерживаемых платформах. Программисты мультимедиа или производители видеоигр имеют инструмент нового поколения, который одинаково удобен для работы в сети и обработки данных, а также с анимацией и графическими спецэффектами. Rebol/View занимает хорошие позиции на рынке сетевых игр, интерактивных CD-ROM и электронных терминалов. Впервые такие продукты будут совместимы на нескольких платформах без изменений.
Специальные эффекты активируются атрибутом эффекта стиля. В примерах мы будем использовать изображение, но помните, что эти операции можно выполнять с любым графическим компонентом. На следующих примерах мы будем мучить Тукса! Изображение tux.gif использует синюю основу (0.0.255) для фиксации прозрачных зон. Давайте начнем с размещения его на цветной основе, применив эффект прозрачности с помощью ключевого атрибута.

```
view layout [
 backdrop 96.128.128
 image %tux.gif effect [ key 0.0.255 ]
]
```

Теперь хорошенько его встряхнем.

С помощью атрибута поворота мы можем повернуть изображение. Используя ползунок,

пользователь может выбрать угол наклона бедного Тукса:

```
view layout [
 backdrop 96.128.128
 tux: image %tux.gif effect [ key 0.0.255 rotate 0 ]
 pos: slider 100x20 [
  tux/effect/rotate: pick [ 0 90 180 270 ]
  ((make integer! (pos/data) * 3) + 1)
  show tux
 ]
]
```

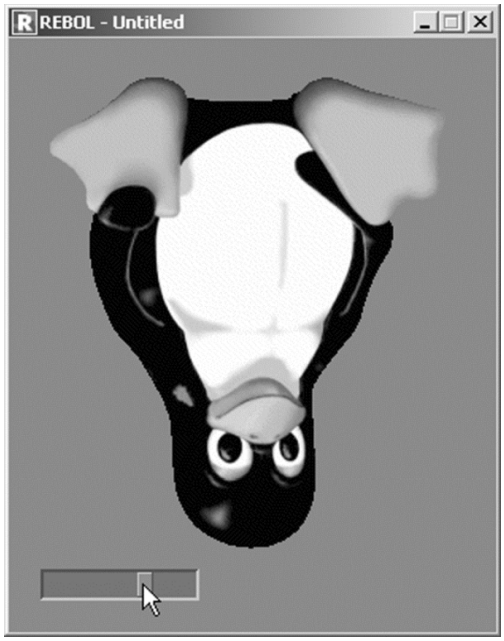

Рисунок 3-5. *Поворот изображения*

Нанести эффект рельефа на талисман Linux не так уж и сложно. Для этого мы собираемся определить флажок "Relief", который в зависимости от текущего состояния добавляет или удаляет эффект тиснения, примененный к изображению.

```
view layout [
 backdrop 96.128.128
 tux: image %tux.gif effect [ key 0.0.255 rotate 0 ]
 across
 pos: slider 100x20 [
  tux/effect/rotate: pick [ 0 90 180 270 ] ((make
  integer! (pos/data) * 3) + 1)
  show tux
 ]
 relief: check "Relief" [
  either relief/data = true [
   append tux/effect [ emboss ]
  ] [ remove (find tux/effect 'emboss) ]
   show tux
 ]
 text "Relief"
]
```

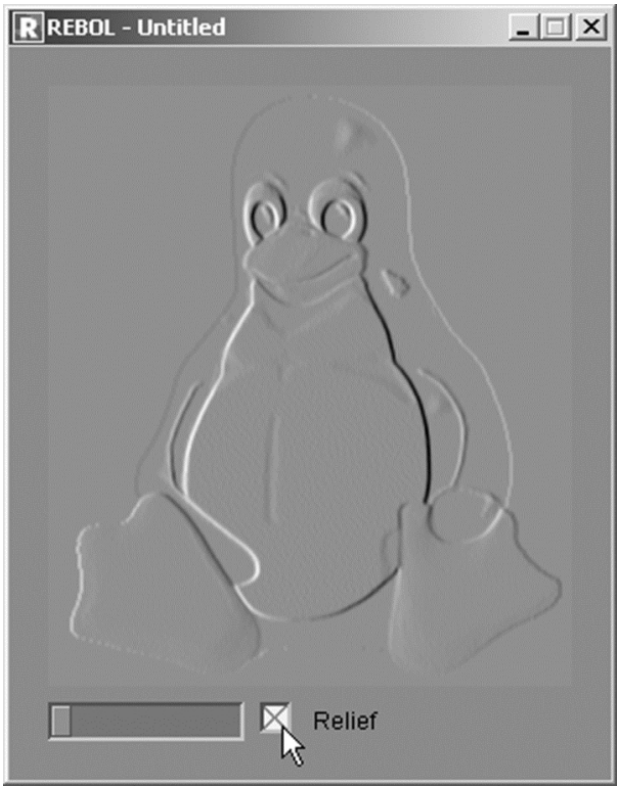

Рисунок 3-6. *Применение эффекта.*

Rebol/View имеет около тридцати различных эффектов, которые поддерживают создание анимации и сложную обработку графики.
Опять же, мощность и скорость Rebol/View GCS делают его единым универсальным инструментом. В то время как его конкуренты требуют написания сотен строк кода и использования дополнительных библиотек, Rebol чрезвычайно лаконичен и самодостаточен.

7.3 Диалект DRAW

Давайте продолжим изучение графического программирования с Rebol/View. Теперь мы рассмотрим функции рисования геометрических фигур на экране. Эти функции являются частью диалекта DRAW, который является неотъемлемой частью атрибута эффекта каждого графического компонента. Это может показаться немного отталкивающим или слишком сложным, но на самом деле это очень мощно.

7.3.1 Нарисуем линию

Диалект DRAW фактически является частью VID как субдиалект. Вы можете использовать его, только включив его в атрибут effect (эффекта) стиля.
Вы размещаете инструкции DRAW в любом стиле, даже поверх изображения или кнопки. В целом, наиболее практичным стилем для рисования является коробка (box), потому что из нее получается отличная доска для рисования, а также она может быть оснащена таймером. Благодаря этому создавать анимацию действительно легко.
Нарисованный цвет определяется инструкцией pen (пера), за которой следует имя цвета или его код RGB (3 октета, представляющие красный, зеленый и синий).
Чтобы нарисовать линию или разместить точку, вы используете инструкцию line, за которой следуют две координаты, типа pair! (пара), для начальной и конечной точек. Оси координат являются декартовыми, а начало координат классически помещается в верхнем левом углу компонента.
Инструкция line-pattern с соответствующими настройками pen (пера) позволяет изменять внешний вид линии с помощью пунктирных и пунктирных линий. Прежде чем указывать узор линии, необходимо сначала настроить перо на два цвета, которыми вы хотите нарисовать линию.

Обычно мы устанавливаем для линии один цвет, поэтому мы указываем, что первый из двух цветов не имеет значения. Это делает первый цвет полностью прозрачным. (Будьте осторожны, потому что это работает, только если ни один не указан перед другим цветом). Затем вы предоставляете line-pattern с четным числом целочисленных параметров.
Когда вы в следующий раз рисуете линию, диалект использует первый цвет для количества пикселей, заданного первым параметром, второй цвет для количества пикселей, заданного вторым параметром, затем первый цвет для количества пикселей, заданного третьим параметром. параметр и так далее. (На момент написания диалект Draw применяется не более чем к четырем параметрам.)

```
view layout [
 box 100x100 effect [
  draw [
   pen white
   line 0x0 100x100
   pen none white
   line-pattern 4 2 5 10
   line 0x100 100x0
   pen white
  ]
 ]
]
```

7.3.2 Другие функции рисования

В вашем распоряжении полный набор инструкций по рисованию. Вы можете нарисовать прямоугольник словом box, за которым следуют две координаты (тип данных pair!), Первая - это верхняя левая точка прямоугольника, вторая - нижний правый угол.
Команда circle рисует окружность, определяемую ее центром и радиусом.
Чтобы заполнить фигуру цветом, вы просто используете инструкцию fill-pen, а затем цвет, прежде чем рисовать фигуру. Фактически вы можете указать fill-pen для использования очень сложных цветовых градаций, указав дополнительные параметры. (Если вы хотите получить пустую фигуру после того, как вы установили перо-заливку, вы должны специально установить для нее значение none). polygon - это мощная инструкция, которая рисует прямые линии между заданными координатами. Геометрическая форма закрывается, так как многоугольник автоматически соединяет последнюю точку с первой. Вы можете заполнить фигуру, просто установив fill-pen перед ее рисованием.

```
view layout [
 box 300x200 effect [
  draw [
   pen white
   box 10x10 100x100
   circle 150x50 30
   polygon 200x10 290x60 270x90 210x110
   fill-pen red
   circle 150x140 40
   box 20x140 100x190
  ]
 ]
]
```

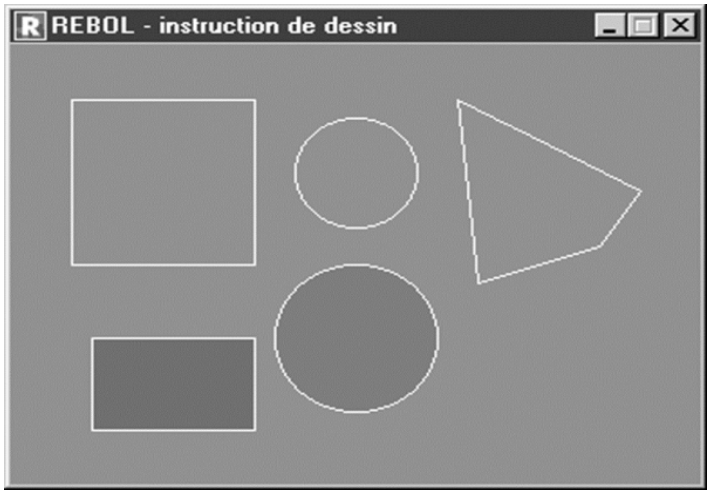

Рисунок 3-7. *Использование диалекта DRAW.*

7.3.3 Добавление текста

Инструкция text отображает строку символов с указанной координатой. Цвет текста фиксируется инструкции pen. Также очень просто создать специальные эффекты, такие как затенение текста:

```
view layout [
 box 200x100 effect [
  draw [
   pen white
   line 0x0 200x100
   line 0x100 200x0
   pen black
   text 2x42 "Text added with DRAW"
   pen red
   text 0x40 "Text added with DRAW"
  ]
 ]
]
```

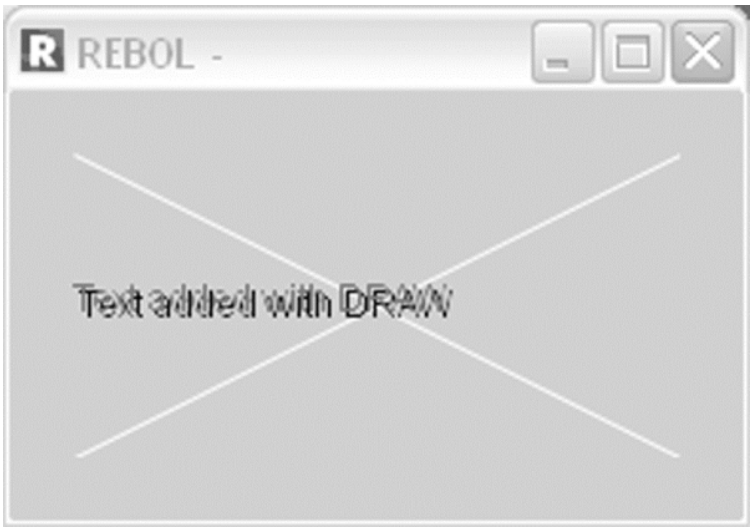

Рисунок 3-8. *Отображение текста с помощью DRAW.*

7.3.4 Манипулирование изображениями

В зоне рисования вы также можете отображать изображения BMP, GIF, JPEG и PNG с помощью инструкции image (изображение). Вам нужно только указать слово, содержащее само изображение, а все остальное сделает диалект Draw. Если вы предпочитаете немного больше контроля, вы можете указать левую верхнюю координату, верхнюю левую и нижнюю правую координаты или координаты всех четырех углов, чтобы указать, где будет отображаться изображение. Вы также можете указать код RGB для прозрачной визуализации цвета.

```
img-tux: load %tux.bmp
monster: load %monstre.bmp

view layout [
 box 170x180 effect [
  draw [
   image img-tux 0x0
   image monster 30x96 0.0.255
  ]
 ]
]
```

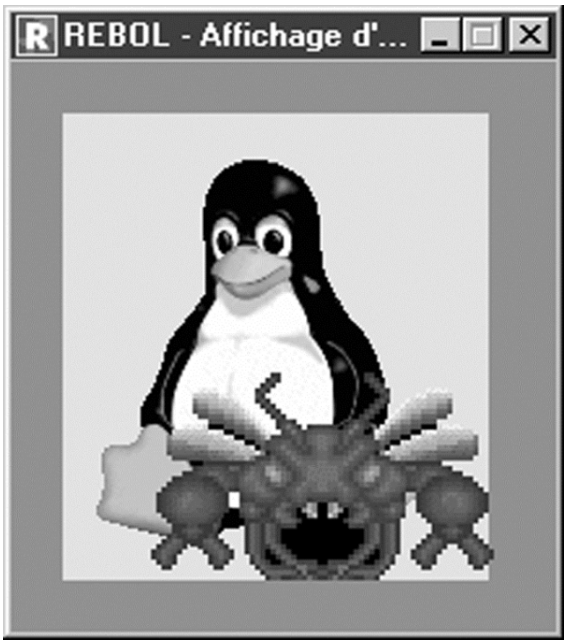

Рисунок 3-9. *Добавление изображения с прозрачностью.*

Эта способность манипулировать изображениями позволяет очень легко создавать спрайты, наложенные изображения, которые появляются на экране перед фоном. Эта функция очень полезна при разработке видеоигр.

7.3.5 Создание файлов изображений.

Когда вы закончите рисовать, вы можете создать из него изображение. Такое изображение может оставаться в памяти и использоваться таким же образом, как и любое другое изображение (применять эффекты, отображать, назначать стиль и т.д.). Его также можно сохранить на диск в одном из двух форматов: PNG или BMP. Вам нужно только использовать слово save (сохранить) с правильным уточнением (/bmp или /png) и преобразовать ваш рисунок в тип image! (изображение). В результате вы получаете файл, который может быть прочитан другим программным обеспечением.

```
view layout [
 mybox: box 100x100 effect [
  draw [
   pen white
   line 0x0 100x100
   line 0x100 100x0
  ]
 ]
 button "save" [ save/png %test.png (make image! mybox) ] ]
```

7.3.6 Динамическое использование DRAW

До сих пор примеры, которые мы видели, были очень статичными. Вы, наверное, задаёте себе ряд вопросов: как изобразить серию фигур на графике или как заставить спрайт двигаться внутри кадра? Не паникуйте! VID и DRAW чрезвычайно гибкие.

7.3.7 Создание инструкций DRAW

Фактически, инструкции DRAW - это просто блок данных. Если вы хотите динамически создавать инструкции для рисования, все, что вам нужно сделать, это изменить содержимое блока стиля effect/draw.
Чтобы лучше понять это, мы построим гистограмму из ряда чисел. Они содержатся в серии и представляют высоту (в пикселях) каждого прямоугольника:

```
numbers: [ 100 130 80 110 50 90 ]
```

Когда пользователь нажимает кнопку Trace, скрипт инициализирует блок graph/effect/draw рисования цветом, который нужно нарисовать, и рисует две оси (x и y). Затем цикл foreach по очереди считывает каждое значение и вычисляет координаты прямоугольника для представления данных. На каждом проходе цикла блок добавляется к блоку graph/effect/draw. После завершения этой операции остается только обновить стиль, называемый графом. Гистограмма появится на экране.

```
view layout [
 graph: box 200x200 effect [ draw [] ]
 button "Trace" [
  pos-dep: 10x190
  graph/effect/draw: copy [
   pen white
   line 5x0 5x200
   line 0x190 200x190
  ]
  foreach value numbers [
   pos-fin: pos-dep + make pair! reduce [
    20 (value * -1)
   ]
   append graph/effect/draw reduce [
    'box pos-dep pos-fin
   ]
   pos-dep: pos-dep + 20x0
  ]
  show graph
 ]
]
```

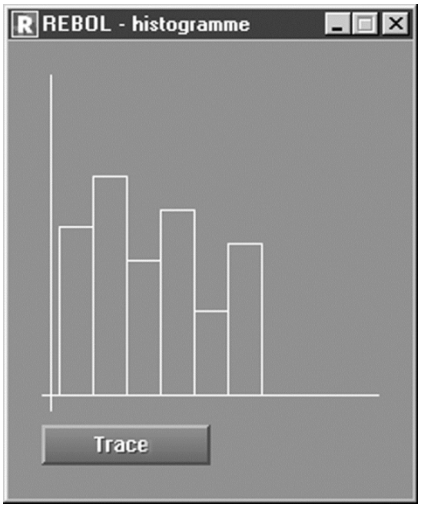

Рисунок 3-10. *Сгенерированная гистограмма*

7.3.8 Немного анимации

С помощью диалекта DRAW можно создавать качественные анимации. Одна из причин этого заключается в том, что Rebol использует виртуальный экран для создания экранных дисплеев. Слово show копирует этот виртуальный экран на физический. Так что мерцания нет. Более того, VID предлагает программисту таймеры для установки временных интервалов. Благодаря им вы можете, например, анимировать спрайты каждые 5 десятых секунды. Это позволяет достичь скорости анимации, эквивалентной мощности компьютера, на котором запущен сценарий.
В VID большинство стилей имеют встроенный независимый таймер. Все, что нужно, - это использовать атрибут rate (скорость), чтобы указать время задержки или количество раз в секунду, которое должен быть активирован стиль. В следующем примере десять раз в секунду рисуется случайная линия:

```
view layout [
 mybox: box 200x200 rate 10 effect [ draw [] ] feel [
  engage: func [ f a e ] [
   if a = 'time [
    append mybox/effect/draw reduce [
     'pen (random 255.255.255)
     'line (random 200x200) (random
    200x200)
    ]
    show mybox
   ]
  ]
 ]
]
```

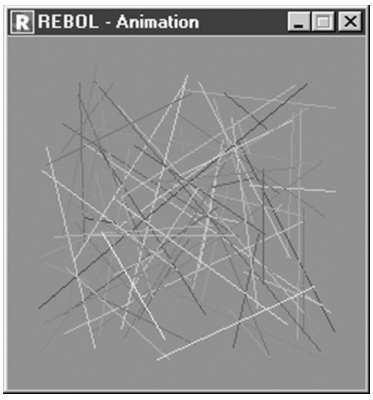

Рисунок 3-11. *Рисование случайных линий.*

Вы, наверное, заметили, что обработчик событий должен быть настроен с атрибутом feel для обнаружения и обработки временного события.

7.4 Обработка событий с помощью VID

Диалект VID поддерживает простую и быструю разработку графических пользовательских интерфейсов, полностью независимых от исполняющей платформы. Как вы видели, вы можете легко размещать и отображать на экране графические компоненты, такие как кнопки, текст и изображения. Следующим этапом является взаимодействие с этими компонентами в ответ на действия пользователей. Теперь мы подробно рассмотрим программирование, управляемое событиями, с помощью Rebol.

7.4.1 Событийно-ориентированное программирование

Сценарий, использующий графический интерфейс пользователя, работает, имея цикл, который ожидает событий, созданных действиями пользователя (нажатие клавиши, выбор графического компонента, перемещение мыши и т.д.), А также событий, генерируемых системой (часы, получение пакеты данных из сети и т. д.). Суть приложения VID состоит в маршалинге кода правильного компонента для выполнения в зависимости от полученного события. Диалект также позволяет вам полностью переопределить поведение компонента, чтобы вы могли создавать новые стили, специально адаптированные для вашего проекта.

7.4.2 Поведение по умолчанию

Каждый из многих компонентов VID имеет поведение по умолчанию. Это заключенный в скобки блок кода, который выполняется для определенного действия. Например, кнопка оценивает свой код, когда пользователь нажимает на нее. Радиокнопка или флажок также реагирует на нажатие на нее пользователем. С другой стороны, поле ввода активирует свой код каждый раз, когда пользователь нажимает клавишу Enter. Ползунок изменяет свое значение при каждом перемещении.
В следующем примере в окне отображается ползунок и кнопка. При каждом изменении положения курсора или нажатии кнопки в консоли отображается значение ползунка (от 0 до 1):

```
view layout [
 myslider: slider 100x20 [
  print myslider/data
 ]
 button "Value" [
  print myslider/data
 ]
]
```

Рисунок 3-12. *Стили, реагирующие на действия пользователя.*

Вы, наверное, понимаете, что это очень простой метод, но он не самый мощный. В то же время он позволяет быстро обеспечить базовое взаимодействие между пользователем и приложением. Чтобы пойти дальше, необходимо немного углубиться в механику управления событиями в Rebol/View.

7.4.3 Отслеживание событий

Каждый объект VID имеет атрибут, называемый feel (чувство), для обработки определенных событий. Обработчик событий применяется к самому конкретному объекту, а не к любому другому объекту, основанному на том же стиле. Таким образом, две кнопки или два изображения могут иметь совершенно разное поведение. Если вы решили изменить поведение стиля, вы должны создать новый стиль, производный от исходной модели (часто называемой прототипом).

В атрибуте feel вы можете поместить четыре функции, каждая из которых играет определенную роль. Функция detect перехватывает все входящие события и распределяет их по трем другим функциям. Engage позволяет вам определять mouse-click (щелчок мышью) или time! (время). Over определяет, когда указатель мыши проходит над компонентом, а также когда он выходит из компонента. Наконец, redraw (перерисовка) позволяет перерисовать компонент.

Последняя операция выполняется автоматически в VID, но её можно деактивировать, чтобы увеличить производительность приложения или чтобы программист мог обрабатывать отдельные случаи индивидуально.

Каждая из этих функций получает свои параметры, которые описывают событие, непосредственно из системы графического композитинга (Graphical Compositing System (GCS)) Rebol/View.

Информация о событии передаётся в виде объекта в переменной face. Переменная action (действия) предоставляет тип события, перехваченного GCS. Положение указателя мыши можно найти в offset (смещение). Описание события содержится в объекте event (событие). Логическая переменная over? указывает, находится ли мышь внутри или вне компонента.

Приведенный ниже код представляет собой полный каркас для обработки событий button (кнопка) VID. С помощью этой модели вы можете полностью переопределить, как она реагирует на действия пользователя:

```
view layout [
    mybutton: button "Hello" feel [
        engage: func [ face action event ] [
        ]
        over: func [ face over? offset ] [
        ]
        detect: func [ face event ] [
        ]
        redraw: func [ face action offset ] [
        ]
    ]
]
```

На данный момент ваша кнопка "Hello" кажется неработающей. Фактически, вы переопределили все его реакции на действия пользователя, чтобы они ничего не делали. Предположим, вы хотите отображать некоторый текст каждый раз, когда указатель мыши находится над кнопкой или когда это не так, все, что вам нужно сделать, это изменить функцию over (поверх) следующим образом:

```
over: func [ face over? offset ] [
    either over? [
        print "over"
    ] [ print "outside" ]
]
```

7.4.4 Объект event (событие)

Этот объект подробно описывает событие, перехваченное GCS, с семью атрибутами.

Вы можете определить, какое событие было перехвачено, по свойству стандартного type. Кнопки мыши обеспечивают события down (вниз), alt-down и up (вверх). Перемещение мыши вызывает событие move (перемещения). Нажатие клавиши на клавиатуре генерирует событие key (кнопка). Пользовательские изменения в окнах отображения создают события resize (изменения размера), close (закрытия), active (активный), inactive (неактивный) и offset (смещение). Наконец, timer (таймер) с регулярным интервалом запускает событие time (время).

После того, как вы определили тип события, вы можете использовать другие свойства объекта event (события), которые содержат дополнительную информацию о событии.

Свойство offset (смещение) возвращает положение мыши. Свойство key содержит значение клавиши, нажатой пользователем. Состояние клавиш Ctrl и Shift можно определить по двум свойствам, имеющим то же имя, что и клавиши; они содержат логические значения. Свойство time - это отметка времени, когда произошло событие, и, наконец, свойство face - это объект, содержащий компонент, активированный в событии.

Как видите, очень легко изменить поведение стиля VID. В нескольких строках кода вы можете, например, изменить цвет кнопки, когда пользователь наводит на неё указатель мыши или когда пользователь нажимает левую кнопку мыши. Для этого вы должны изменить методы engage и over кнопки и обработать событие down (нажатие) (левая кнопка мыши, нажатая пользователем) и параметр over?.

```
view layout [
    mybutton: button "Hello" feel [
        engage: func [ face action event ] [
            if event/type = 'down [
                face/color: 0.255.0
                show face
            ]
        ]
        over: func [ face over? offset ] [
            either over? [
                face/color: 255.0.0
            ] [ face/color: 0.0.255 ]
            show face
        ]
        redraw: func [ face action offset ] [
        ]
    ]
]
```

Рисунок 3-13. *Цвет кнопки меняется при наведении на неё указателя.*

7.4.5 Управление окнами

Для окон в вашем приложении наиболее эффективным методом является установка единого обработчика для всех событий, которые они производят (открытие, закрытие, перемещение и т.д.). С помощью слова insert-event-func вы вставляете функцию в основной цикл мониторинга событий GCS. Целью этого является реагирование на каждое из событий, производимых окнами вашего приложения. Один из двух параметров, которые получит вставленная функция, - это объект event (событие), который позволяет вам определить, какое окно затронуто (event/face) и тип события (event/type). Функция должна заканчиваться словом event, чтобы позволить сценарию продолжить выполнение, запустив обработку любых событий, ожидающих в очереди. Слово remove-event-func позволяет удалить обработчик событий, если он больше не нужен. В приведённом ниже примере показаны все события окна с именем wind:

```
display-events: function [ face event ] [] [
    if event/face = wind [ prin [ event/type " " ] ]
    event
]
insert-event-func :display-events

wind: layout [
    button "Quit" [
    remove-event-func :display-events
    quit
    ]
]
view/options wind 'resize
```

Рисунок 3-14. *Некоторые перехваченные события.*

И снова Rebol остаётся верным одной из своих основных концепций: предоставить программисту полную свободу. Rebol даёт вам очень точный контроль над событиями в вашем приложении.

7.5 Управление стилями

VID является прекрасным примером того, насколько эффективным может быть использование Rebol. Несколько строк кода - это все, что нужно для разработки сложного графического интерфейса. У программиста есть свобода персонализировать многочисленные стандартные стили и добавлять новые. Эти элементы могут быть собраны вместе в настоящие таблицы стилей. VID позволяет программисту изменять все доступные стили: у вас есть средства адаптировать их внешний вид, чтобы вы могли создать уникальный внешний вид для своего приложения.

Если презентация VID вам не удобна, ничто не мешает вам добавить в свой скрипт немного QNX, Aqua или даже Windows. Здесь ничего не высечено из камня; всё это можно легко изменить.

7.5.1 Определение стиля

В макете определяется стиль, который будет его использовать. За ключевым словом style следует имя и атрибуты стиля, который вы хотите создать. Ваш стиль наследует свойства и методы указанной модели (её прототипа), но отличается от того, в котором вы указали изменения.

Чтобы облегчить понимание, давайте начнём с простого: наша цель - определить стиль, называемый red-button (красная кнопка). Она похожа на стандартную кнопку, за исключением того, что она красная с белым текстом. Вы "извлечёте" стиль кнопки и измените её цвет:

```
view layout [
    style red-button button red
    red-button "button1" [ print "button1" ]
    button "button2" [ print "button2" ]
    red-button "button3" [ print "button3" ]
]
```

После того, как стиль был определён в макете, вы можете использовать его точно так же, как вы использовали бы стандартный стиль VID. Единственное ограничение - вы должны определять новый стиль в начале каждого макета, в котором он используется. Чтобы решить эту проблему, VID поддерживает создание таблиц стилей, позволяющих разделить логику приложения и настройки отображения.

7.5.2 Применение таблицы стилей

Принципы таблиц стилей в Rebol очень близки к принципам CSS с HTML. В начале ваших приложений или в отдельном файле вы описываете внешний вид и поведение ваших персонализированных стилей. Затем у вас есть возможность подключить одну или несколько таблиц стилей к разным макетам (layout) в вашей программе. Таблица стилей определяется с помощью слова stylize, за которым следует блок, содержащий описание различных стилей.

Этот блок назначается слову не только для того, чтобы его можно было идентифицировать, но и для того, чтобы его можно было присоединить к макету (layout) с помощью ключевого слова styles.

Давайте возьмём предыдущий пример и добавим к нему ещё один стиль, называемый red-text (красный текст), который будет простой меткой с красным текстом. Таблица стилей носит имя my-styles:

```
my-styles: stylize [
    red-button: button red
    red-text: text red
]
```

Всё, что вам нужно сделать сейчас, чтобы использовать ваши новые стили, - это включить вашу таблицу стилей в макет, в котором вы хотите их использовать.

```
view layout [
    styles my-styles
    red-text "I am written in red text"
    red-button "Button1" [ print "Button1" ]
    button "Button2" [ print "Button2" ]
]
```

Этот метод позволяет вам использовать множество различных таблиц стилей в одном приложении и, что очень полезно, сделать внешний вид вашего программного обеспечения очень гибким. В идеале таблицы стилей можно хранить в файлах, а затем включать их с помощью слова do при запуске сценария. Простое изменение этих таблиц стилей позволяет кардинально изменить внешний вид программного обеспечения. Подобная организация таблиц стилей даёт компании возможность определить стандарты представления для всех своих приложений или адаптировать внешний вид приложения к требованиям заказчика.

7.5.3 Изменение аспектов стиля

Для одной конкретной потребности или в случае создания нового сложного стиля вы также можете внести глубокие изменения в стандартный стиль VID. Таким образом, можно изменить атрибуты, определяющие контур стиля, шрифт символов, используемый для отображения текста, или даже характеристики абзаца.

Чтобы изменить стиль границы, вы должны изменить атрибуты edge (край) блока, который существует для каждого компонента VID. Атрибут size (размер) - это pair! (пара!) и фиксирует ширину и высоту границы компонента. С помощью color (цвет) вы указываете цвет границы, который отображается, как указано в атрибуте effect (эффект). Этот последний атрибут принимает одно из значений ((‘bevel, ‘ibevel, ‘bezel, ‘ibezel, ‘nubs), которое определяет, как будет прорисовываться край.

С помощью блока с именем font вы можете выбрать символьный шрифт (используя ключевого слово name (имя)) из одного из трёх шрифтов (font-serif, font-sans-serif, font-fixed). Параметр style позволяет выбрать стиль текста (‘bold (полужирный), ‘italic (курсив) или ‘underline (подчёркивание)). size (размер) и color (цвет) задают размер и цвет символа.

Вы можете определить, как текст будет выровнен, используя параметр align (выровнять), за которым следует одно из значений left (лево), right (право) или center (центер).

Форматирование абзаца контролируется параметром para (параграф), который принимает блок параметров. Вы устанавливаете точное положение, в котором текст отображается с параметром origin (источник). С помощью scroll (прокрутка) вы можете разрешить горизонтальную или вертикальную прокрутку содержимого стиля VID. Параметр tabs (табуляция) может содержать одно или несколько значений для создания позиций табуляции в тексте. Наконец, wrap? (завернуть?) - логическое значение для включения или выключения переноса слов.

Давайте посмотрим на пример, чтобы лучше понять: цель - создать новый стиль, называемый big-button (большая кнопка). Здесь отображается шрифт внушительного размера с рельефным эффектом по краям. Для этого лучше всего начать с простого стиля box (коробка). Это, вероятно, наиболее нейтральный стиль VID, поэтому его можно использовать для создания большинства новых стилей. С его помощью у вас есть большая свобода в управлении событиями. Определение этого поля указывает используемый шрифт, выравнивание текста и внешний вид границы.

```
my-styles: stylize [
    big-button: box font [
        name: font-serif
        size: 40
        align: 'center
        valign: 'middle
    ] edge [
        size: 10x10
        color: 192.192.192
        effect: 'bevel
    ]
]
```

Теперь все, что вам нужно сделать, это включить таблицу стилей в свой макет, чтобы вы могли её использовать.

```
view layout [
    styles my-styles
    backcolor 0.0.255
    big-button "Rebol !" 200x100 0.0.0
]
```

7.5.4 Определение поведения стиля

Несмотря на изменённый внешний вид, определённый вами стиль принимает поведение своего прототипа. На данный момент ваша "кнопка" реагирует на любое действие пользователя так же, как и стандартный блок (box).

Чтобы она работала как настоящая кнопка, вы должны определить обработчики событий для этого нового стиля.

Ваша цель состоит в том, чтобы кнопка инвертировалась при нажатии левой кнопки мыши и в то же время выполнялся некоторый код Rebol. Для этого вы должны переопределить метод engage, роль которого состоит в том, чтобы сигнализировать о щелчке и отпускании кнопки мыши.

Вы изменяете высоту границы, изменяя свойство effect (эффект) edge (грани) текущего активного объекта (находится в face). Код, переданный в качестве параметра, выполняется функцией do-face.

```
my-styles: stylize [
    big-button: box font [
        name: font-serif
        size: 40
        align: 'center
        valign: 'middle
    ] edge [
        size: 10x10
        color: 192.192.192
        effect: 'bevel
    ] feel [
        engage: func [ face action event ] [
            ;;print event
            either event/type = 'down [
                face/edge/effect: 'ibevel
                do-face face none
            ] [
                face/edge/effect: 'bevel
            ]
            show face
        ]
    ]
]

view layout [
    styles my-styles
    backcolor 0.0.255
    big-button "Rebol !" 200x100 0.0.0 [
        print "Super big button !"
    ]
]
```

Теперь ваш стиль реагирует на действия пользователя и выполняет код при нажатии.

7.6 Rebol и звук

Изучив графические возможности Rebol/View, вы теперь узнаете, что он предлагает на арене звука. Rebol может считывать образцы звуков и манипулировать ими, что позволяет разрабатывать мультимедийные приложения, такие как игры, интерактивные компакт-диски и информационные киоски. Мы узнаем об этом, разработав звуковой проигрыватель, который позволит нам применить наши знания о VID.

7.6.1 Открытие и закрытие звукового порта

Rebol/View и Rebol/Command включают порт, называемый sound (звук), специально для воспроизведения звуков. Этот порт нужно открывать словом open и закрывать словом close. Будьте осторожны с портами этого типа, поскольку может пройти некоторое время, прежде чем он будет активирован системой. Не стесняйтесь использовать слово wait (ждать), чтобы установить короткую паузу (две или три десятых секунды должно быть достаточно) перед воспроизведением первого звука, если вам нужно сделать это сразу после открытия порта. Проверка на наличие ошибки при открытии (open) порта позволяет узнать, есть ли в используемой вами версии Rebol звуковые функции. Таким образом вы можете изменить логическое значение, чтобы указать, настроен ли ваш скрипт на звук. В нашем примере, если мы не можем использовать звуковой порт, мы просто завершим скрипт сообщением об ошибке:

```
if error? try [
    sound-port: open sound://
    close sound-port
] [
    alert "You don't have access to sound"
    quit
]
```

7.6.2 Загрузка и обработка звуковых образцов

Rebol может обрабатывать несжатые (WAVeform) звуковые образцы. Файл WAV загружается в память с помощью слова load (загрузка), которое создаёт объект с пятью свойствами, которые можно как читать, так и записывать:

- data содержат данные, которые будут воспроизводиться звуковой картой вашего устройства,
- volume определяет громкость звука на выходе (от 0 до 1),
- channels указывает количество звуковых каналов (1 или 2),
- bits указывает количество битов, используемых для выборки. Если значение 8 бит, каждое значение выборки составляет амплитуду от 0 до 255. Для 16 бит диапазон значений простирается от 0 до 65535,
- rate указывает частоту дискретизации в герцах.

Качество звукового образца в основном зависит от количества используемых битов и их частоты. Звук, отобранный с частотой 44100 Гц с использованием 16 бит, что является нормой для аудио CD, будет более качественным, чем звук с частотой 8000 Гц с 8 битами (близко к качеству телефонной линии).
Теперь мы можем приступить к созданию основного экрана нашего звукового проигрывателя WAV. Макет (layout) содержит кнопку File (файл) для загрузки звукового фрагмента в слово echantillon (французское слово "образец"):

```
button "File" [
    files: request-file/filter "*.wav"
    if all [
        (not none? files)
        (not empty? files)
    ] [echantillon: load first files ]
]
```

С помощью кнопки Edit (редактировать) мы можем отобразить диалоговое окно, в котором можно изменить свойства звука. Затем пользователь может изменить громкость, выбрать другое количество каналов, выбрать другое качество дискретизации и зафиксировать скорость чтения, используя частоту дискретизации. Окно появляется только в том случае, если в память загружен звук. Поля в блоке редактирования обновляются перед отображением.

```
button "Edit" [
    if not none? echantillon [
        volume/data: echantillon/volume
        either echantillon/channels = 1 [
            can1/data: true
            can2/data: false
        ] [
            can1/data: false can2/data: true
        ]
        either echantillon/bits = 8 [
            bits8/data: true
            bits16/data: false
        ] [
            bits8/data: false
            bits16/data: true
        ]
        freq/text: echantillon/rate
        view/new/title editor "Editor"
    ]
]
```

Макет (layout) с именем Editor содержит ползунок для регулировки громкости и поле ввода для установки частоты. Количество бит и канал выбираются с помощью переключателей:

```
editor: layout [
    across
    text "Volume :"
    volume: slider 100x20
    return
    text "Number of channels:" return
    can1: radio of 'can text "1"
    can2: radio of 'can text "2" return
    text "Precision:" return
    bits8: radio of 'precis text "8 bits"
    bits16: radio of 'precis text "16 bits" return
    text "Frequency:" freq: field 50 return
    button "Cancel" [ unview edition ]
    button "Update" [
        echantillon/volume: volume/data
        either can1/data = true [
            echantillon/channels: 1
        ] [echantillon/channels: 2 ]
        either bits8/data = true [
            echantillon/bits: 8
        ] [echantillon/bits: 16 ]
        echantillon/rate: to-integer freq/text
        unview editor
    ]
]
```

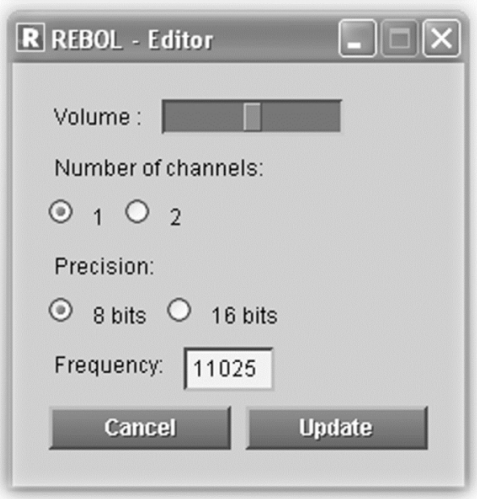

Рисунок 3-15. *Окно конфигурации WAV-плеера.*

Кнопка Cancel (отмена) позволяет пользователю закрыть диалоговое окно без изменения свойств звука в памяти. С другой стороны, кнопка Update (обновить) применяет изменения, внесённые пользователем в образец.

7.6.3 Воспроизведение сэмплов

Пользователь нашей программы нажимает кнопку Play (воспроизвести), чтобы начать воспроизведение звуковых образцов. Если сэмпл загружен в память, данные вставляются в порт звуковой карты.

```
button "Start" [
    if not none? echantillon [
        port-sound: open sound://
        wait 0.1
        insert port-sound sample
        wait port-sound
        close port-sound
    ]
]
```

После вставки в порт семпл немедленно воспроизводится. Необязательное использование слова wait указывает интерпретатору дождаться конца звукового образца, прежде чем продолжить выполнение сценария.

7.6.4 Графическое отображение звукового образца

Наш проигрыватель файлов WAV уже умеет читать и воспроизводить звук. Чтобы улучшить его внешний вид, мы можем отображать выбранные значения в виде графика в реальном времени. Формат данных, хранящихся в свойстве data объекта, содержащего образец, является образцом простоты. Когда выборка 8-битная, каждый байт представляет собой амплитуду. Если звук стереофонический, он состоит из двух каналов. Чётные байты в данных, например 0 2 4 6 и т.д., относятся к левому каналу. Звуки для правого канала представляют собой байты с нечётными номерами, например 1 3 5 7 и т.д.. Для звуков, дискретизированных по 16 бит, логика такая же, за исключением того, что каждое значение представлено двумя байтами.

Мы будем использовать стиль box (коробка) и диалект draw (рисование), чтобы визуализировать различные звуковые волны на экране. Для экономии времени данные анализируются при загрузке файла WAV. Два списка (data-left и data-right) используются для хранения значений, извлечённых из двоичных данных в echantillon/data. Сначала циклы forskip просматривают данные. Внутренний

цикл (loop) выполняется один или два раза в зависимости от количества каналов, присутствующих в файле.

Генерация 16-битного значения из двух байтов выполняется простой операцией: (byte1 * 256) + byte2 (умножение двоичного значения на 256 имеет тот же эффект, что и его сдвиг на восемь бит влево).

```
forskip son lg-package [
    canal: 1
    package: copy/part son lg-package

    loop echantillon/channels [

        either echantillon/bits = 8 [
            valeur: first package
        ] [
            valeur: ((first package) * 256) + (second package)
        ]

        either canal = 1 [
            insert tail data-right valeur
            canal: 2
        ] [
            insert tail data-left valeur
        ]

        packet: skip packet next-channel
    ]
]
```

Нам осталось только сгенерировать график на диалекте Draw. Мы построим кривую функцией trace-curve. Два уточнения (/left и /right) сообщают функции, какой канал отображается.

```
trace-curve: func [ data /left /right ] [

    either left [
        append canaux/effect/draw [ pen 255.0.0 ]
        depy: 99
        mult: -1
    ] [
        append canaux/effect/draw [ pen 0.255.0 ]
        depy: 101
        mult: 1
    ]

    data: at data (make integer! (sl-canaux/data / stp))

    repeat x 450 [
        xy1: make pair! reduce [ x depy ]
        finy: (100 + (mult * (make integer! ((first data) / stpy))))
        xy2: make pair! reduce [ x  finy ]
        append canaux/effect/draw reduce [ 'line xy1 xy2 ]
        data: next data
    ]
]
```

Рисунок 3-16. *Графическое отображение звукового образца.*

Полная версия этого звукового проигрывателя содержит меньше 4 КБ кода и может быть улучшена путём добавления новых функций, таких как вырезание и вставка для выполнения цифрового редактирования или применения эффектов (эхо, реверберация и т.п.). Это ещё одно доказательство простоты и невероятной универсальности Rebol.

7.7 Резюме

Rebol/View позволяет легко разрабатывать платформенно-независимые графические пользовательские интерфейсы. Диалект VID может обрабатывать изображения, имеет множество специальных эффектов и поддерживает простое переопределение внешнего вида и поведения графических компонентов. Параметры звука ограничены, но достаточны для многих мультимедийных проектов.

8. Сеть и Интернет

Rebol - это язык обмена сообщениями, предназначенный для извлечения, обработки и распространения данных по сетям. По этой причине в нем есть много инструментов сетевого программирования.

8.1 Использование протоколов TCP/IP

Rebol - отличный язык для сетевого программирования. Без необходимости в единственном расширении интерпретатор Rebol может использовать основные протоколы TCP/IP. Это также позволяет определять новые сетевые протоколы. Таким образом, можно не только разработать клиент или сервер, который использует существующий протокол, но также и тот, который использует новый протокол, адаптированный к вашим потребностям.

8.1.1 Протоколы в Rebol

Все версии Rebol поддерживают десять распространённых протоколов TCP/IP. HTTP (Hypertext Transport Protocol) позволяет читать документы из Интернета. Благодаря ему вы можете извлекать HTML-документы, изображения, мультимедийные документы и даже запускать выполнение cgi-скриптов на удалённом сервере. FTP (File Transfer Protocol) поддерживает получение и отправку файлов с помощью другого компьютера, подключенного к сети. Электронная почта управляется с помощью протокола SMTP для отправки сообщений (Simple Mail Transport Protocol) и протоколов IMAP (Internet Message Access Protocol) и POP3 (Post Office Protocol) для их получения. Новости в Интернете читаются по протоколу NNTP (Network News

Transmission Protocol). Также существуют протоколы для сбора информации. Можно узнать IP-адрес сервера, опросив сервер доменных имён (DNS), и наоборот. "Маленькие" протоколы, такие как Finger, Whois и Daytime, позволяют получить информацию об учётной записи пользователя, административные данные доменного имени, а также дату и время с другого компьютера.

8.1.2 Конфигурация сети

Прежде чем делать что-либо на сетевом уровне с Rebol, вы должны сначала убедиться, что ваш интерпретатор Rebol правильно настроен. Если это не так, маловероятно, что вы сможете подключиться к Интернету или получить свою электронную почту. Если вы не предоставили свои сетевые данные для Rebol при его установке, вы должны использовать слово set-net, параметр которого представляет собой блок, содержащий шесть значений. Вы должны внести в этот список:

- Ваш адрес электронной почты,
- Имя или IP-адрес сервера для отправки почты,
- Имя или IP-адрес сервера для приёма почты,
- Имя или IP-адрес прокси,
- Номер порта прокси,
- Тип прокси.

Прокси-сервер - это приложение, которое подключает своих клиентов к Интернету. Это метод отправки HTTP-запросов в Интернет и предоставления ответов компьютеру, который отправил запрос.
Если вы не хотите (вы не хотите отправлять электронную почту с Rebol) или не должны (ваш компьютер напрямую подключен к Интернету) указывать параметр, замените его значением none.

```
set-net [
    olivier.auverlot@domaine.fr
    smtp.domaine.fr
    pop3.domaine.fr
    proxy.domaine.fr
    8080
    generic
]
```

Rebol поддерживает четыре разных типа прокси:

- socks,
- socks4,
- socks5,
- generic.

8.1.3 Отправка и получение электронной почты

Теперь вы можете отправлять и получать электронную почту. Для этого вы просто используете слова send (отправить) и read (прочитать).
Мы можем отправить короткое сообщение на почтовый ящик с помощью команды

```
send olivier.auverlot@domaine.fr "Привет, я - сообщение"
```

Если тело сообщения содержит возврат каретки (enter, несколько строк), вы должны заключить строку символов между "{" и "}", которые указывают на "многострочную" строку:

```
send olivier.auverlot@domaine.fr {
    Привет
    я - сообщение
}
```

Используя уточнение /header (заголовк), вы можете добавить такую информацию, как тема. Дополнительным параметром для этого уточнения является объект, различные свойства которого соответствуют множеству элементов, которые могут быть указаны в заголовке электронной почты.

```
header: make system/standard/email [
    Subject: "message subject"
]
send/header olivier.auverlot@domaine.fr "hello"
```

Многие поставщики интернет-услуг недавно начали использовать расширенный простой протокол передачи почты - Extended Simple Mail Transfer Protocol (ESMTP) в качестве меры для повышения безопасности и борьбы со спамом. Текущие версии Rebol включают поддержку ESMTP за счёт добавления двух новых параметров в set-net; это имя учётной записи и пароль:

```
set-net [
    olivier.auverlot@domaine.fr
    smtp.domaine.fr
    pop3.domaine.fr
    proxy.domaine.fr
    8080
    generic
    olivier
    homer
]
```

Если вы не укажете имя учётной записи и пароль, Rebol запросит их, когда вы попытаетесь отправить сообщение. (Вы, конечно, должны быть очень осторожны, чтобы не сохранить свой пароль в исходном или текстовом файле Rebol).

Чтобы прочитать ваши сообщения, всё, что вам нужно сделать, это использовать слово read, за которым следует URL-адрес, состоящий из протокола (POP3 или IMAP), имени вашей учётной записи, вашего пароля и имени почтового сервера:

```
print read pop://olivier:homer@pop3.domaine.fr
```

Если имя вашей учётной записи электронной почты является вашим полным адресом электронной почты, включая @domain, использовать эту простую форму для чтения почты не очень удобно. Возможно, вам захочется использовать более длинный, но более читаемый метод определения почтового ящика:

```
mailbox: [
    scheme: 'pop
    host: "pop3.domaine.fr"
    user: "olivier.auverlot@domaine.fr"
    pass: "homer"
]

print read mailbox
```

Каждое сообщение принимается в виде многострочной символьной строки, которая помещается в список. Есть много способов легко управлять отправкой и получением писем с помощью Rebol. Любое приложение можно быстро расширить для обмена сообщениями с людьми (сигналы тревоги, автоматические отчёты и т.п.) Или другими приложениями.

8.1.4 Доступ к веб-ресурсам

С помощью протоколов HTTP и FTP вы можете читать и отправлять документы по сети. Это позволяет нам использовать Rebol в качестве навигатора для получения файлов HTML или XML. Эта функция облегчает создание агентов, предназначенных для поиска и сбора информации. Фактически, всё, что вам нужно сделать, это указать URL-адрес для чтения ресурса в сети. В следующем примере отображается содержимое домашней страницы Rebol Technologies:

```
print read http://www.rebol.com
```

Назначив результат запроса символьной строке с помощью слова copy (копия), вы можете выполнять поиск по нему и извлекать из него информацию. При load/markup (загрузка/разметка) полученный файл анализируется интерпретатором Rebol. Затем вы получите блок, состоящий из тегов HTML или XML (tag!) и строк символов.

С помощью FTP вы можете как отправлять, так и получать файлы. Используются два слова: read (чтение) и write (запись). URL-адрес, переданный в качестве параметра, содержит учётную запись пользователя и пароль, необходимые для установления соединения.

С Rebol операции, необходимые для управления удалёнными файлами, такие же, как и для локальных. В следующем примере мы прочитаем файл с именем rapport.txt с FTP-сервера и отобразим его на экране:

```
print read ftp://olivier:passwd@ftp.domaine.fr/docs/rapport.txt
```

8.1.5 А другие протоколы?

В Rebol все протоколы следуют одной и той же модели; нет особых случаев или исключений. Полный набор протоколов основан на едином синтаксисе, что делает его очень интуитивно понятным в использовании.

Предположим, вы хотите узнать IP-адрес машины. Вы просто используете синтаксис

```
ip-address: read dns://webserver
```

Другой пример? Вы хотите знать дату и время сервера в сети. Синтаксис очевиден:

```
print read daytime://myserver
```

8.1.6 Клиенты и серверы

Если есть один домен, в котором Rebol действительно впечатляет, это, вероятно, простота разработки клиент-серверных приложений с использованием сетевых протоколов TCP или UDP.

Чтобы написать клиента, то есть программу, отправляющую данные на сервер и ожидающую ответа, вы сначала открываете сокет (socket) (комбинацию IP-адреса и порта).

Затем, вы вставляете информацию, которая будет отправлена на сервер, и после получаете ответ. Последний этап - закрыть сокет, чтобы освободить его.

```
REBOL [
    Subject: "client TCP"
]

; этот скрипт открывает соединение с машиной 172.29.143.1
; на порту 8000/tcp
p: open tcp://172.29.143.1:8000
; посылается запрос (оканчивающийся ентером)
insert p "hello^/"
; ответ сохраняется в переменной
response: copy ""
read-io p response 255
print response
close p
```

Написание сервера не представляет особых проблем. Просто откройте порт прослушивания и дождитесь получения данных. Когда данные поступают, вы их обрабатываете и возвращаете ответ клиенту.

```
REBOL [
    Subject: "TCP server"
]
; открываем порт
p: open tcp://:8000
; сервер входит в бесконечный цикл
forever [
    wait p
    ; соединение обнаружено
    conn: first p
    ; читаем запрос
    request: copy ""
    read-io conn request 255
    print [ "request ->" request ]
    ; отправляем ответ
    insert conn "OK^/"
    ; закрываем соединение с клиентом
    close conn
]
```

8.2 Создание сетевых протоколов в Rebol

Разработка сетевых протоколов, вероятно, является одним из самых захватывающих аспектов языка Rebol. Благодаря им вы можете связывать свои приложения с TCP/IP, обмениваться данными, создавать одноранговые (точка-точка) сообщества или использовать веб-службы.

По умолчанию все версии Rebol содержат группу протоколов, охватывающих широкий спектр приложений. С Rebol/Core и Rebol/View у вас есть десять сетевых протоколов, готовых к использованию (HTTP, POP и т. Д.). Коммерческие версии Rebol предоставляют другие более специализированные сетевые протоколы, ориентированные на электронный бизнес, в частности протоколы для использования основных систем управления базами данных (СУБД), таких как MySQL и Oracle, а также тех, которые обеспечивают безопасный обмен данными (HTTPS). Если, однако, вы не можете найти то, что хотите среди этих протоколов, вполне возможно создать новый протокол, который можно будет добавить к существующим, который можно будет использовать так же, как и стандартные. Просматривая различные веб-сайты Rebol, вы сможете найти новые протоколы, которые вы можете расширить (доступ к базам данных, telnet, SNMP и т.д.).

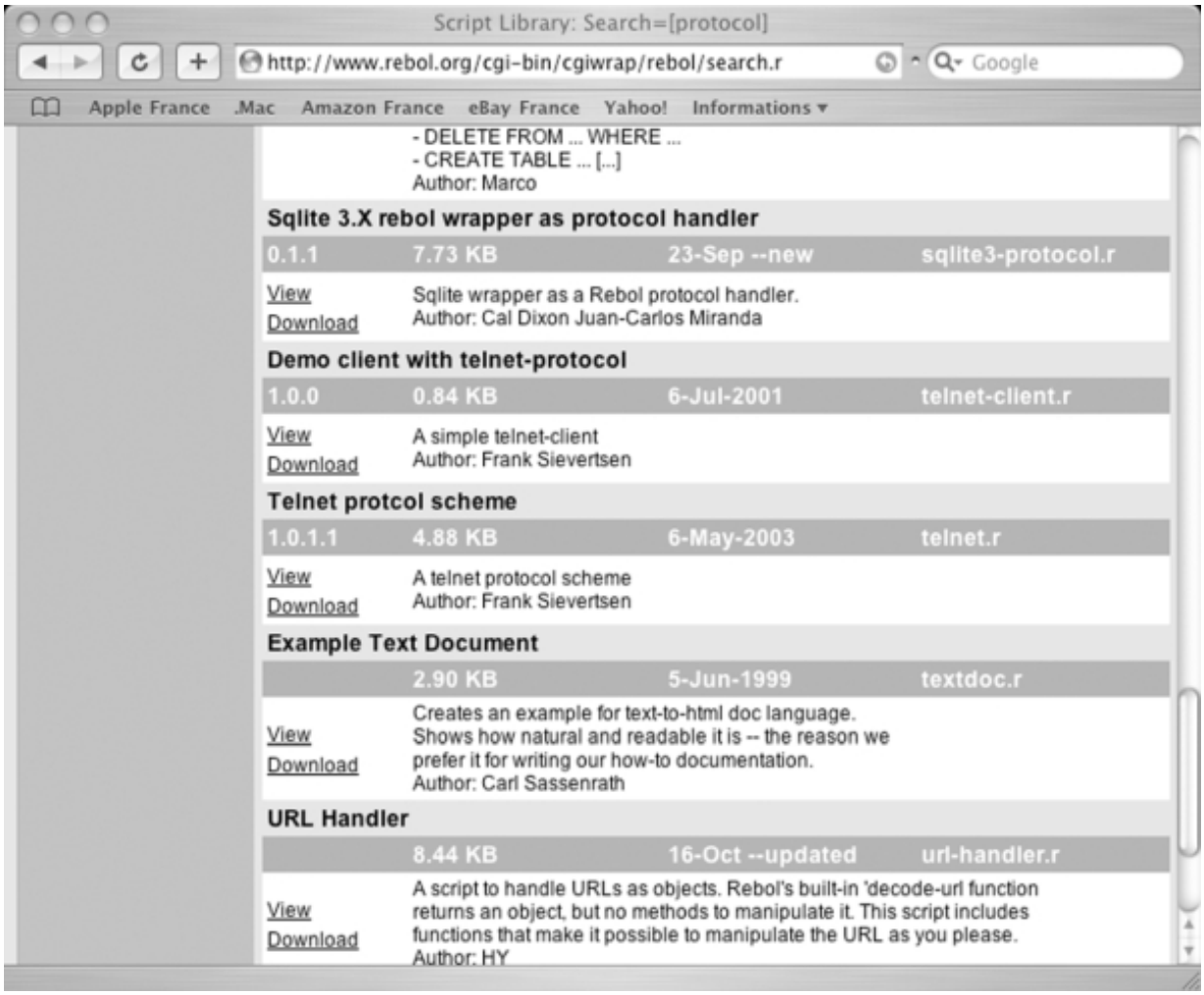

Рисунок 4-1. *Многие протоколы доступны на www.rebol.org.*

Вы, наверное, думаете, что это действительно интересно, но при этом очень сложно, совсем нет! Rebol снова демонстрирует превосходный компромисс между лёгкостью и мощностью. Разработка новых сетевых протоколов действительно доступна каждому.

8.2.1 Стандартные протоколы

Чтобы понять, как работают сетевые протоколы в Rebol, лучше всего изучить те, которые включены в интерпретатор. Rebol - это метаязык, части Rebol написаны на Rebol, и это касается протоколов. Различные протоколы хранятся в свойстве схем системного объекта. Чтобы сохранить их на жёсткий диск, все, что вам нужно сделать, это ввести в консоль следующие инструкции:

```
echo %protocols.txt
probe system/schemes
echo none
```

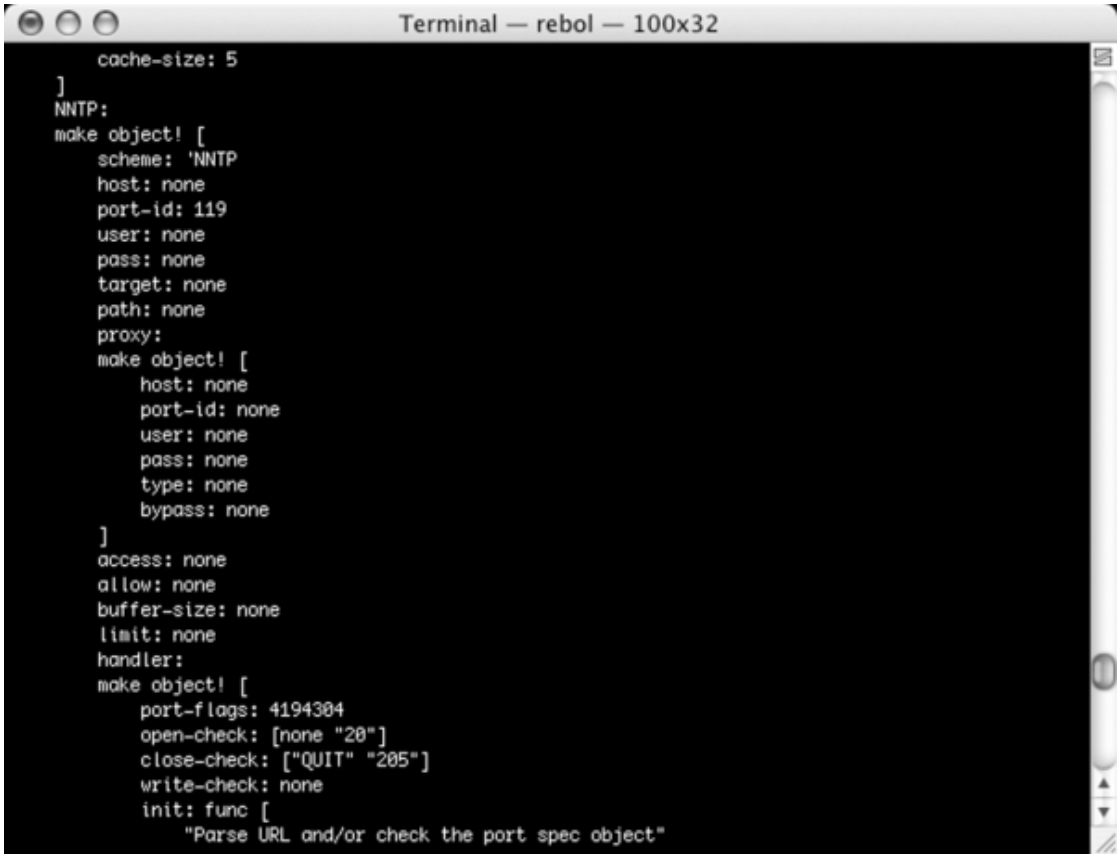

Рисунок 4-2. *Протоколы, включённые в Rebol.*

Затем вы получите текстовый файл, содержащий исходный код всех протоколов в Rebol. Каждый из них является объектом, наследующим свойства и методы корневого объекта, корневого протокола (root-protocol).

8.2.2 Корневой протокол

Чтобы просмотреть код корневого протокола, вам нужно ввести в консоли команду source root-protocol. Этот объект, на котором основаны все сетевые протоколы Rebol, содержит модель полностью функционирующего протокола.

Он уже может запускать соединение, вставлять данные и считывать данные из порта TCP или UDP, а также останавливать соединение, чтобы освободить все используемые ресурсы. Фактически, всё, что программисту нужно сделать для создания нового протокола, - это решить, должен ли он использовать порт TCP или UDP. Самый удивительный пример - это, наверное, daytime протокол в Rebol. Его реализация занимает всего одну строчку кода.

```
make Root-Protocol [ net-utils/net-install Daytime self 13 ]
```

Этой единственной строки достаточно для создания объекта, производного от корневого протокола (root-protocol). Единственная модификация, которая абсолютно необходима, - это, по сути, метод net-util/net-install.

У этого есть три функции, так как он определяет используемый порт, имя протокола и, особенно, вставляет новый протокол в свойство system/scheme.

Очевидно, что вы можете разработать более продвинутые протоколы, и в этом случае вам, вероятно, будет необходимо знать и изменять различные свойства и методы, содержащиеся в объекте корневого протокола (root-protocol). Вы быстро увидите, что каждый из них выполняет очень точную роль.

8.2.3 Свойства объекта root-protocol

Объект root-protocol определяет четыре свойства: port-flag (флаг порта), open-check (проверка открытия), close-check (проверка закрытия) и write-check (проверка записи). Первый, port-flag, вероятно, наиболее трудный для понимания, поскольку он позволяет вам указать базовый протокол связи, TCP или UDP, который будет использоваться для обмена данными.

В Rebol есть два режима работы с сетевыми протоколами: прямой доступ (direct access) и контролируемый доступ (controlled access). При использовании порта прямого доступа (system/standard/port-flags/direct) данные доступны посимвольно или построчно. Каждый раз, когда данные берутся из порта, они немедленно удаляются, чтобы разрешить доступ к следующим данным. Этот режим особенно хорошо подходит для обработки больших объёмов информации.

Управляемый режим (system/standard/port-flags/pass-thru) даёт программисту большую свободу в обмен на дополнительную работу. Фактически, разработчику остаётся явным образом управлять получением и хранением данных. Информация обычно хранится в свойстве протокола, поэтому необходимо переопределить все существующие методы, чтобы они возвращали все данные, собранные в программу. Поведение портов можно дополнительно улучшить, применив набор применимых констант с помощью бинарного оператора or (или). Итак, чтобы настроить порт на работу в управляемом режиме с двоичными данными, достаточно использовать синтаксис:

```
port-flags: system/standard/port-flags/pass-thru or 32
```

Свойства open-check, write-check и close-check упрощают автоматизацию процесса при согласовании подключения к удалённому процессу, при записи данных или закрытии соединения. Различные этапы согласования указываются с помощью блока, содержащего значения, которые должны быть переданы, и данные, ожидаемые взамен. Таким образом, блок ["bye" [100 200]] говорит клиенту передать строку "bye" и дождаться возвращаемых значений 100 или 200, прежде чем продолжить. Отправка и получение этих значений не входит в обязанности программиста, поскольку методы объекта корневого протокола справляются со всем этим с помощью net-utils/confirm.

8.2.4 Методы объекта root-protocol

Объект root-protocol (корневой протокол) содержит 10 методов, которые принимают порт в качестве параметра. Каждый из них играет определённую роль в работе протокола.

Метод init устанавливает URL-адрес сервера, к которому должен быть подключен клиент. Этот метод инициализирует свойства user, pass, host, port-id, path и target объекта порта.

Метод open-proto запускает соединение с сервером. Этот сложный метод имеет дело со многими операциями, такими как поддержка прокси. По умолчанию сетевые протоколы Rebol используют TCP, но можно выбрать UDP, указав уточнение /sub-protocol с параметром "udp" во время вызова этого метода.

open-proto - важный метод, но не настоящая точка входа в протокол. По сути, это open (открыть), что он и делает. По умолчанию метод open фактически вызывает open-proto. Такое поведение по умолчанию редко бывает достаточным, и вам часто придётся изменить этот метод, как задумано дизайнерами.

Метод close вызывается после завершения соединения и освобождает все ресурсы, используемые соединением.

Методы write (запись) и read (чтение) позволяют записывать или читать данные в порт или из порта и вызываются, когда в сценарии используются слова write-io и read-io. Метод get-sub-port возвращает порт, используемый для связи между клиентом и сервером. Методы get-Mode и set-Mode обеспечивают доступ для чтения и записи к свойствам порта. Наконец, метод awake (пробуждение) вызывается при обнаружении поступления данных в порт.

8.2.5 Реализация эха

Чтобы лучше понять, как писать сетевые протоколы с помощью Rebol, мы сейчас рассмотрим реализацию протокола эха. Эта служба, определённая в RFC 862, очень полезна для определения того, подключён ли компьютер к сети, и даже для оценки производительности сети. Этот протокол довольно прост; он основан на отправке символьной строки от клиента и получении её обратно от сервера. В Rebol вы реализуете протокол эха, который отправляет строку "hello" (привет) на удалённый сервер и ждёт его ответа. Как только это получено, клиент протокола возвращает время, затраченное на отслеживание производительности сети.

Протокол будет добавлен в список сетевых протоколов Rebol, и его можно будет использовать, предоставив URL-адрес с указанием имени протокола echo (эхо) и IP-адреса или доменного имени сервера.

Первый шаг состоит в создании объекта, унаследованном от корневого протокола (root-protocol), и установке значения его свойств. Было бы пустой тратой времени изменять open-check, write-check или close-check, поскольку для подключения к серверу с этим протоколом не требуется никаких согласований. С другой стороны, значения port-flags (флаг порта) в system/standard/port-flags/pass-thru должны быть установлены для работы в контролируемом режиме.

Какие методы нужно добавить или изменить, когда слово read (прочить) применяется к протоколу?

Фактически, read (чтение) вызывает три метода: open, copy и close.

Метод open вызывает open-proto, и нет необходимости изменять его поведение для протокола эха. Close закрывает порт TCP или UDP, используемый для соединения, и его также не нужно изменять. Таким образом, вам нужно добавить только один метод, copy, который выполняет задание по отправке и получению символьных строк в порт и из порта. Имейте в виду, что вы переопределите copy в контексте протокола, а это означает, что использование слова copy вызовет вызов метода копирования, который вы определили, а не copy в глобальном контексте. Перед его изменением вы должны сохранить слово-копию, просто создав другое слово (sys-copy). Тогда "новый" экземпляр не представляет особых проблем.

Используемый порт передаётся методу в качестве параметра, а затем все, что нужно, - это вставить строку символов и дождаться её возврата сервером. Поскольку в этом случае нет специальных символов конца строки, слово read-io используется для ожидания получения 5 символов. Затем метод завершает свою работу, возвращая время, затраченное на весь процесс.

Чтобы добавить протокол эха в список известных Rebol, установите имя протокола и, особенно, укажите номер используемого порта TCP, для этого необходимо заполнить объявление протокола с помощью инструкции net-utils/net-install.

```
echo-protocol: make root-protocol [
    port-flags: system/standard/port-flags/pass-thru
    sys-copy: get in system/words 'copy

    copy: func [ port /local reponse ] [
        t1: now/time/precise
        reponse: sys-copy ""
        insert port/sub-port "hello"
        read-io port/sub-port reponse 5
        (now/time/precise - t1)
    ]

    net-utils/net-install echo self 7
]
```

Как только протокол загружен в интерпретатор Rebol, достаточно просто ввести read echo://server-name, чтобы узнать, доступен ли сервер и какова задержка.

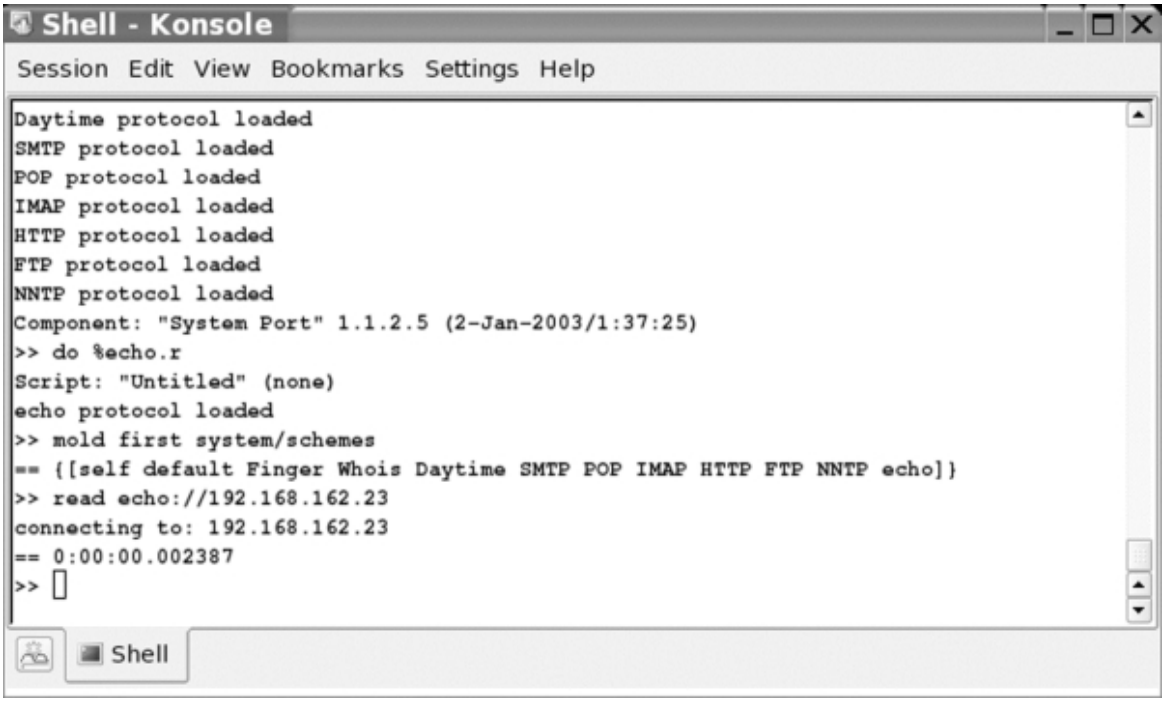

Рисунок 4-3. *Использование протокола эха.*

8.2.6 Разработка протокола gopher

Протокол gopher был разработан Миннесотским университетом. Он определен в RFC 1436 и представляет собой распределенную информационную систему. Он не только обеспечивает доступ к данным (документам и файлам), но и к приложениям (telnet, терминал 3270) через древовидную структуру. Основываясь на модели клиент / сервер, gopher работает очень просто. Клиент отправляет запрос на TCP-порт 70 сервера. Это указывает на то, что клиент хочет прочитать документ на сервере или описание содержимого каталога. Конец запроса обозначается с помощью символов CR и LF. В случае, если сервер возвращает содержимое файла, данные заканчиваются символами CR и LF. Для описания содержимого каталога каждый элемент описывается в строке символов, заканчивающейся возвратом каретки, а символы CR и LF отмечают конец ответа.

Чтобы использовать gopher с Rebol, клиент протокола предоставит два разных метода. Программист сможет использовать read (чтение), за которым следует URL-адрес, содержащий имя протокола (gopher), имя или IP-адрес сервера gopher и, наконец, путь к запрошенному файлу.

Другой метод состоит в открытии порта с использованием протокола gopher со словом open, вставке запроса в этот порт и последующем чтении ответа перед закрытием порта (close).

Как и в случае с протоколом эха, вы должны использовать значение system/standard/port-flags/pass-thru для свойства port-flag. Свойство path предназначено для хранения клиентского запроса. Свойство filetype содержит типы элементов, распознаваемых gopher. Таким образом, если сервер возвращает, что элемент test.txt имеет тип "0", клиентский протокол знает, что это файл.

Методы insert и copy отправляют оба клиентских запроса через метод send-cmd. Фактически, метод insert вызывается, когда вы используете слово insert в глобальном контексте, но не при использовании read (чтение).

В этом случае используются только методы open, copy и close; для отправки запроса необходимо полагаться на копию (copy). Просто нужно предотвратить это дважды. Это контролируется с помощью send-cmd и уточнения /path. После отправки команды в порт связи (port/sub-port) данные считываются методом копирования (copy). Если последний символ клиентского запроса - "/", данные, полученные клиентом, будут описанием содержимого каталога. Метод list-directory обрабатывает результат. Если запрос был для файла, содержимое ответа возвращается в виде копии (copy).

```
gopher-protocol: make root-protocol [

    port-flags: system/standard/port-flags/pass-thru

    sys-copy: get in system/words 'copy
    sys-insert: get in system/words 'insert
    sys-close: get in system/words 'close
filepath: none

filetype: [
    #"0" "FILE"
    #"1" "DIRECTORY"
    #"2" "CSOPHONEBOOK"
    #"3" "ERROR"
    #"4" "BINHEX"
    #"5" "BINDOS"
    #"6" "UUENCODE"
    #"7" "INDEX"
    #"8" "TELNET"
    #"9" "BIN"
    #"+" "REDUNDANTSERVER"
    #"T" "TERM3270"
    #"g" "GIF"
    #"I" "IMAGE"
    #"s" "AUDIO"
    #"M" "MIME"
    #";" "ANIMATION"
    #"h" "HTML"
]

send-cmd: func [ port /path data ] [
    if none? filepath  [
        either not path [
            either any [
                (none? port/path)
                (port/path = "/")
            ] [
                filepath: sys-copy ""
            ] [
                filepath: sys-copy port/path
                if not none? port/target [
                    append filepath port/target
                ]
            ]
        ] [ filepath: sys-copy data ]
        sys-insert port/sub-port filepath
    ]
]

list-directory: func [ data /local content ] [
    content: sys-copy []
    foreach file data [
        file: parse file ""
        append/only content reduce [
            select filetype (first file/1)
            sys-copy/part (at file/1 2) ((length? file/1) - 1)
            sys-copy/part (at file/2 2) ((length?    file/2) - 1)
            to-tuple file/3
            to-integer file/4
        ]
    ]
    content
]
```

```
insert: func [ port data ] [ send-cmd/path port data ]

copy: func [ port /local item buffer result ] [
    send-cmd port
    buffer: sys-copy port/sub-port
    either (length? filepath) > 0 [
        either (last filepath) = #"/" [
            result: list-directory buffer
        ] [ result: sys-copy buffer ]
    ] [  result: list-directory buffer ]
]

close: func [ port ] [
    filepath: none
    sys-close port/sub-port
]

net-utils/net-install gopher self 70
]
```

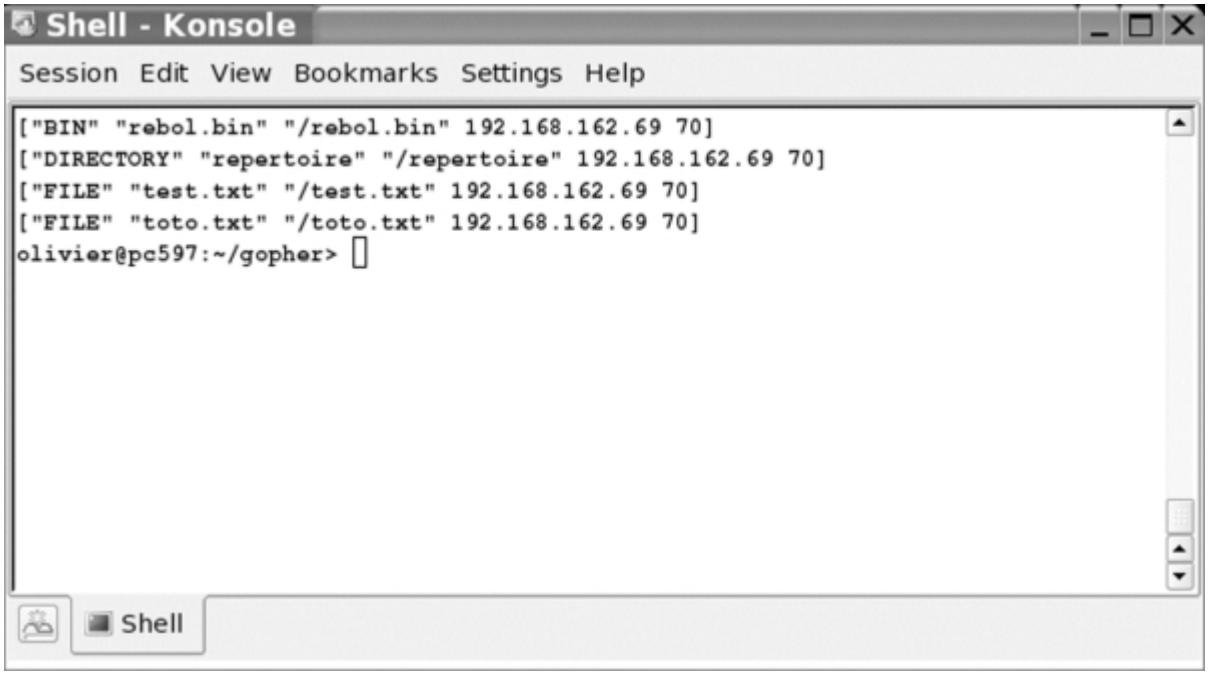

Рисунок 4-4. *Используя протокол gopher .*

Rebol позволяет разрабатывать сетевые протоколы всего несколькими десятками строк кода. Эта характеристика делает его идеальным языком для сетевых приложений и обмена информацией. Теперь вы можете расширить свой интерпретатор Rebol множеством существующих протоколов и, почему бы и нет, даже изобрести некоторые для своих нужд.

8.3 Скрипты Rebol и CGI

Теперь мы переключаем наше внимание на разработку веб-приложений на http-сервере. С помощью Rebol/Core и Rebol/Command вы можете создавать сценарии CGI, программы, которые запускаются на http-сервере, графический пользовательский интерфейс которого состоит из HTML-страниц, отображаемых в браузере.

8.3.1 Общая картина

Веб-приложение - это программа, установленная на http-сервере. Клиентские машины подключаются к этому приложению через браузер. Принцип работы основан на парадигме запрос/ответ. Клиент запрашивает сервер HTTP-запросом, и сервер обычно отвечает HTML-страницей. Эта технология хороша тем, что вам не нужно устанавливать приложение на рабочий стол клиента.

Всё централизовано на некоторых машинах, о которых могут легко позаботиться системные администраторы. Более того, использование многоуровневой архитектуры (клиент, http-сервер, сервер базы данных) предотвращает прямой доступ клиентов к данным. Для подключения к СУБД настроен только HTTP-сервер. Это решение даёт ряд преимуществ для путешествующих пользователей; особенно те, которые отделены от своей обычной рабочей станции.

Если вы хотите создать торговый центр или информационный сайт в Интернете, CGI - это простой и быстрый способ разработать веб-приложение.

8.3.2 Обзор GCI

Пользователь получает информацию на веб-странице с помощью HTML-формы. При нажатии кнопки отправки в форме (отправить) данные в форме передаются в сценарий CGI, указанный в URL-адресе. Эта программа может получать данные одним из двух способов:

- Метод GET указывает, что параметры будут передаваться в URL-адрес в браузере,
- Метод POST означает, что данные передаются в CGI. в отдельности.

Выбор того или иного метода зависит от контекста. Метод GET ограничивает длину данных, передаваемых по URL-адресу, и они видны в веб-браузере. С помощью POST вы не ограничены размером, и данные скрыты от пользователя. С другой стороны, извлечение данных с помощью сценария CGI немного сложнее.

После получения параметров сценарий CGI может делать всё, что хочет. Чаще всего этот тип приложения подключается к базе данных для сохранения или получения информации, но ничто не мешает вам отправлять электронные письма или получать доступ к файлам на сервере. Ваша свобода зависит от прав, предоставленных вам системным администратором машины.

Во всех случаях работа CGI заканчивается созданием документа, формат которого определяется его типом MIME: HTML-страница, подтверждающая успешность запрошенной операции, или PDF-файл, собранный веб-сервером, чтобы пользователь мог его распечатать, и т.д..

8.3.3 Как писать CGI-скрипты в Rebol?

Прежде всего, вы должны настроить свой веб-сервер так, чтобы он позволял запускать сценарии CGI. При использовании Apache необходимо просмотреть директивы ScriptAlias и AddHandler. Вы также должны сохранить свой код в каталоге, авторизованном для сценариев CGI. На всех основных веб-серверах есть каталог cgi-bin, предназначенный для этой задачи.

Давайте начнем с чего-нибудь очень простого, а именно с отображения небольшой текстовой строки в окне браузера. В системе UNIX или Mac OS X вы должны использовать первую строку вашего скрипта, чтобы связать ваш скрипт CGI с интерпретатором Rebol. Синтаксис: #!/Usr/bin/rebol –cs. Параметр -c заставляет интерпретатор работать в режиме CGI. Что касается опции -s, она отключает защиту файлов Rebol. (Если вы используете Apache в системе Windows, вы можете использовать специальную версию Windows #!C:/rebol/rebol.exe –cs).

Затем вы должны указать тип документа (Content-Type), предназначенный для клиентского браузера. Это часть заголовка http, отделённая от тела документа двумя символами возврата каретки. Далее следует запускаемый код Rebol.

```
#!/usr/bin/rebol -cs

REBOL [ ]
print "Content-Type: text/plain^/"
print "Hello !"
```

Этот файл, называемый test.cgi, помечается как исполняемый файл в системах Unix и Mac OS X с помощью команды chmod + x test.cgi. Вам остаётся только протестировать скрипт в своём браузере. URL-адрес должен выглядеть примерно так http:///cgi-bin/test.cgi.

8.3.4 Чтение параметров

Предположим, теперь вы хотите отправить содержимое HTML-формы в ваш CGI-скрипт. Это создаёт проблему чтения информации, отправляемой браузером. Для этого у вас есть группа переменных среды, доступных с помощью объекта Rebol system/options/cgi. Каждое из его свойств сопоставляется с одной из переменных среды http-сервера. Единственным исключением является свойство other-headers, которое позволяет использовать различные характеристики протокола HTTP, такие как файлы cookie. Следующий сценарий CGI отображает все переменные в вашем браузере:

```
REBOL [ ]
print "Content-Type: text/plain^/"
print mold system/options/cgi
```

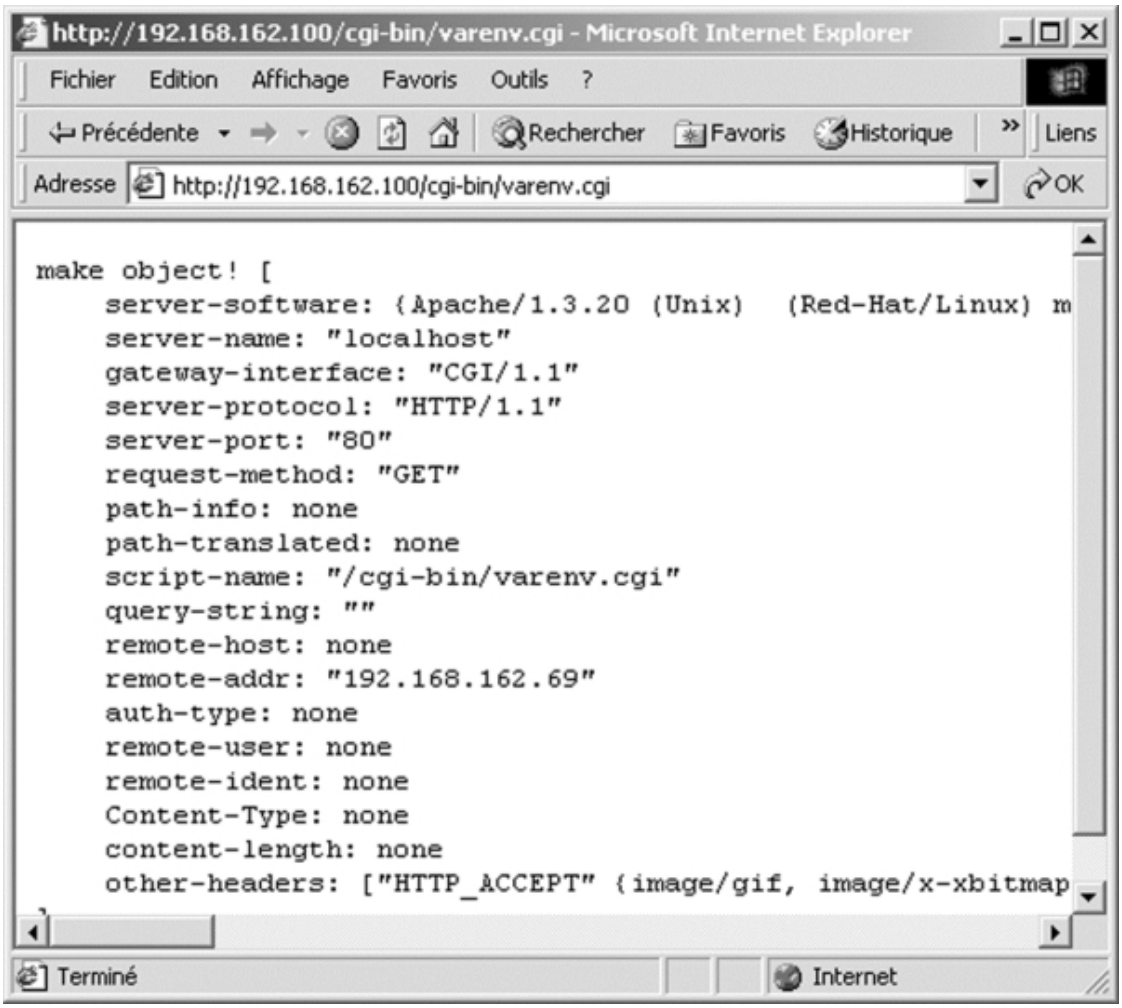

Рисунок 4-5. *Переменные среды CGI.*

Чтобы понять использование этих переменных, создайте небольшое веб-приложение, в котором пользователь вводит своё имя. Сервер отвечает страницей приветствия. Например, если пользователь вводит "Оливье" и "Оверло", сервер создаёт страницу, содержащую "Привет, Оливье Оверло". Сначала вы должны разработать HTML-форму, чтобы записать имя пользователя и сохранить его в каталоге в древовидной структуре вашего HTTP-сервера.

```
<html>
<body>
<form name="myform" method="get"action="http://server.domaine.org/cgi-
bin/hello.cgi">
<input name="firstname" type="field" value=""><br>
<input name="name" type="field" value=""><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Эта HTML-страница определяет простую форму, и как только пользователь нажимает кнопку "Отправить", содержимое передаётся в сценарий hello.cgi с использованием метода GET.

Рисунок 4-6. *Отправка данных на http-сервер.*

На сервере сценарий должен создать HTML-страницу на основе содержимого переменной QUERY-STRING. Данные, передаваемые браузером, имеют формат MIME и не могут использоваться напрямую. В этом примере переменная QUERY-STRING содержит "firstname = olivier & name = auverlot".

Чтобы облегчить вашу рабочую нагрузку, Rebol содержит слово decode-cgi, которое создаёт объект из символьной строки формата MIME: каждое из его свойств соответствует полю в форме.

```
rebol [ ]
print "content-type: text/html^/"
info: make object! decode-cgi system/options/cgi/query-string
print "<html><body>"
print [ "hello " info/firstname info/name ]
print "</body></html>"
```

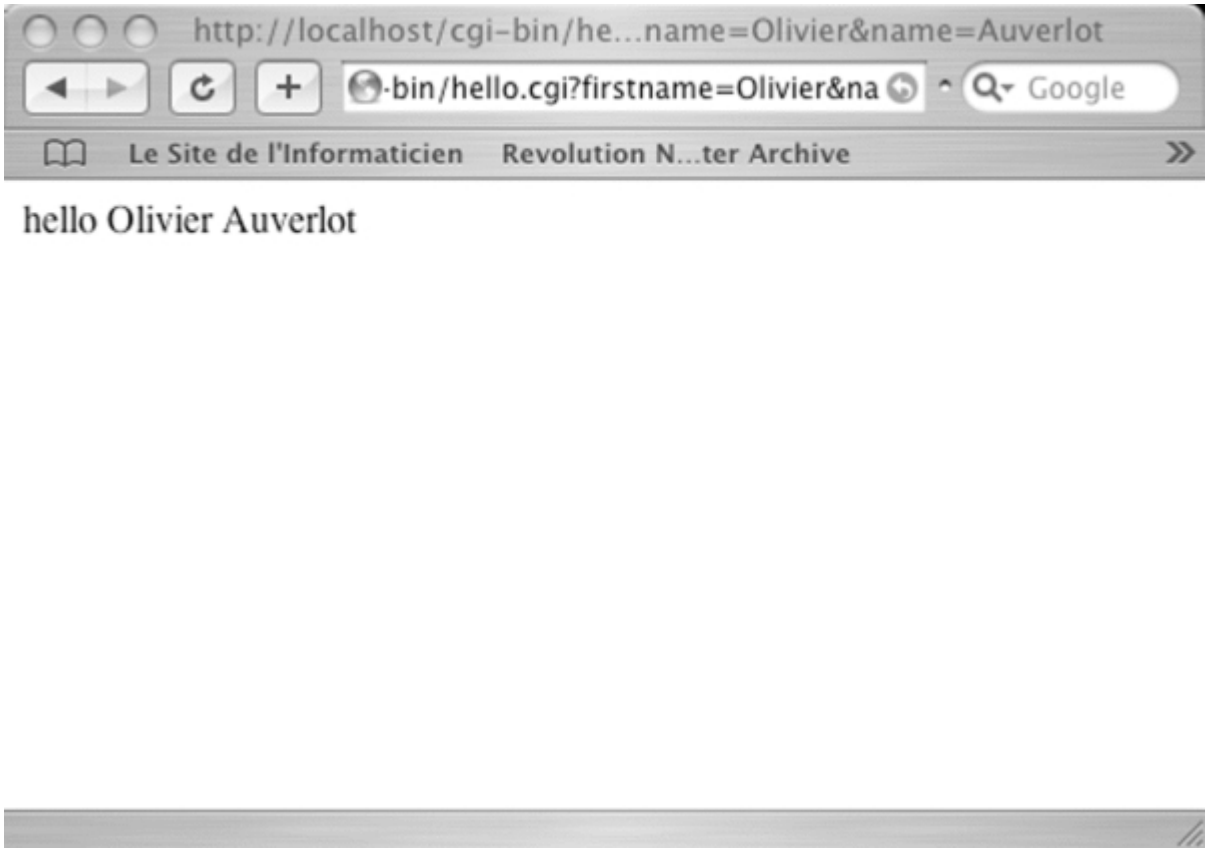

Рисунок 4-7. *Вывод сценария CGI.*

Используя переменную REQUEST-METHOD, ваш сценарий CGI может определить метод передачи данных, используемый клиентским компьютером. Просто проверьте содержимое этой переменной, чтобы автоматически адаптировать ваш сценарий к тому или иному из двух методов. Если содержимое формы передается с использованием метода POST, сценарий CGI должен также прочитать количество байтов, указанное в свойстве CONTENT-LENGTH, из стандартного входного файла.

```
rebol [ ]
print "content-type: text/html^/"
print "<html><body>"
either system/options/cgi/request-method = "get" [
    data: system/options/cgi/query-string
] [
    data: copy ""
    length: to-integer system/options/cgi/content-length
    until [
        buffer: copy ""
        read-io system/ports/input buffer (to-integer
                system/options/cgi/content-length)
        append data buffer
        ((length? data) = length)
    ]
]
info: make object! decode-cgi data
print "<html><body>"
print [ "hello " info/firstname info/name ]
print "</body></html>"
```

Рисунок 4-8. *Переданное с помощью метода POST не отображается в URL-адресе.*

Написание CGI-скриптов в Rebol совсем не сложно. Это простая и надежная технология, с помощью которой можно быстро разрабатывать веб-приложения.

8.4 Создание динамических веб-документов

CGI - не единственный способ выполнить код Rebol на веб-сервере. Magic!, расширение веб-сервера Apache, написанное на Rebol, позволяет размещать инструкции Rebol внутри ваших HTML-страниц. Как и PHP, JSP и ASP, Magic! выполняет код Rebol на веб-сервере. Это полностью прозрачно для клиента и позволяет очень быстро разрабатывать интерактивные веб-сайты. Основное преимущество этого решения заключается в том, что нет необходимости устанавливать права каждого пользователя на выполнение сценариев CGI.

Динамические страницы, созданные с помощью Magic! такие же, как и другие документы на веб-сервере, и хранятся в том же каталоге, что и статические html-страницы.

8.4.1 Как Magic! работает

Magic! принимает форму расширения HTTP-сервера Apache. Написанный на Rebol, он работает на всех платформах, поддерживаемых Rebol/Core, а также может использовать дополнительные функции View/Pro и Rebol/Command, если одна из них присутствует. Magic! по сути, представляет собой простой, короткий CGI, который содержит полный набор функций, позволяющих снять всю тяжёлую работу по разработке динамических страниц. Механизм, который он использует, очень прост: когда сервер Apache встречает страницу *.rhtml, он выполняет сценарий magic.cgi, который анализирует документ и выполняет код Rebol, найденный внутри него. Окончательный результат передаётся клиенту, который видит только простую HTML-страницу. Используя Magic! вы не ограничены только HTML-страницами, вы также можете создавать документы в других форматах, таких как XML-документ, изображение или даже PDF-файл.

8.4.2 Установка

Magic! и его документация на французском языке доступна по адресу http://www.auverlot.fr/Fichiers.html. Сценарий magic.cgi необходимо поместить в cgi-bin вашего сервера Apache с правильными правами выполнения. Вам также может потребоваться изменить строку shebang (первую строку) в файле magic.cgi, чтобы она правильно описывала путь к вашему интерпретатору Rebol.

Остальные работы по установке Magic! состоит из настройки Apache, чтобы он мог управлять своими новообретёнными возможностями. Для этого вы должны изменить файл Apache httpd.conf. Первый этап состоит из авторизации использования сценариев CGI, чтобы сделать magic.cgi доступным для работы.

Таким образом, вы должны объявить каталог по умолчанию для сценариев CGI и указать расширение файла, которое будет использоваться ими:

```
ScriptAlias /cgi-bin/ "/Library/WebServer/CGI-Executables/" AddHandler cgi-
script .cgi
```

После этого вы можете изменить файл httpd.conf так, чтобы он распознавал Magic! расширение файла (.rhtml). Это делается с помощью директивы Addhandler, вы назначаете "волшебное" событие файлам с расширением rhtml, а с помощью директивы Action вы сообщаете Apache, что все файлы, вызывающие "волшебное" событие, должны быть перенаправлены в сценарий magic.cgi.

```
AddHandler magic .rhtml
Action magic /cgi-bin/magic.cgi
```

Вы также можете изменить настройку директивы DirectoryIndex, чтобы добавить страницу index.rhmtl к страницам сайтов по умолчанию. Это простое изменение позволяет настроить динамическую домашнюю страницу.

```
DirectoryIndex index.rhtml index.html
```

Все, что осталось сделать, это перезапустить сервер Apache, чтобы применить изменения, которые вы только что внесли в конфигурацию. Для этого пользователи Linux RedHat должны использовать команду service httpd restart. В других системах (MacOS X, BSD или других дистрибутивах Linux) вы используете apachectl restart.

8.4.3 Когда статика становится динамической

Чтобы проверить Magic! установки, наиболее эффективным методом является ввод вашей первой динамической страницы с именем test.rhtml. Это выглядит как обычная HTML-страница, за исключением того, что документ содержит два новых тега: и . Между этими тегами простая строка кода Rebol, отображающая текущее время.

```
<html>
<head></head>
<body>
<rebol>
    print now/time
</rebol>
</body>
</html>
```

Очевидно, что вы можете вставить столько блоков , сколько захотите, на страницу RHTML.

Код Rebol также можно разместить в начале и в конце документа. Следующий пример демонстрирует, что легко отделить определение функции от её использования:

```
<rebol>
 current-time: does [ print now/time ]
</rebol>
<html>
<head></head>
<body>
 <rebol> current-time </rebol>
</body>
</html>
```

В веб-приложениях принято передавать данные, введённые в форму, в сценарий CGI или динамическую страницу. Magic! поддерживает эту операцию с помощью специального объекта с именем vars. Каждое свойство в этом объекте соответствует полю в переданной форме. Также Magic! автоматически определяет метод передачи данных, используемый формой (GET или POST). Для программиста это означает, что чтение данных полностью прозрачно. Следующий пример представляет собой HTML-форму, в которой вводятся имя и фамилия пользователя. Данные передаются на страницу post.rhtml методом POST:

```
<html>
<head></head>
<body>
<form name="info" method="post" action="post.rhtml">
 Name:<input type="field" name="name"><br>
 First Name:<input type="field" name="first-name"><br>
 <input type="submit" value="Submit">
</form>
</body>
</html>
```

Затем на целевой странице могут отображаться данные, полученные и доступные через объект vars:

```
<html>
<head></head>
<body>
<rebol>
 print [ "hello " vars/first-name " " vars/name ]
</rebol>
</body>
</html>
```

Как видите, Magic! значительно облегчает работу программиста, но его возможности не исчерпываются. Он предоставляет ряд других функций, предназначенных для поддержки динамических страниц.

8.4.4 Magic! функциональность

Magic! настоящий набор инструментов для веб-разработчиков. Он фактически добавляет набор новых слов в словарь Rebol, позволяя совместно использовать библиотеки кода, защищая веб-сервер, контролируя доступ пользователей, изменяя тип MIME сгенерированных документов, устанавливая и считывая файлы cookie и, наконец, управляя сеансами.

8.4.5 Библиотечные функции

Одна из самых полезных функций Magic! - это, вероятно, его команда library, которая позволяет хранить разделы кода Rebol в отдельных скриптах. С его помощью можно избежать дублирования кода в разных файлах, что облегчает совместное использование компонентов программного обеспечения между разработчиками. Чтобы реализовать эту функцию, всё, что нужно, - это создать каталог библиотеки и сообщить Magic! пути доступа, задав переменную m-library-path в файле magic.cgi. Так что если вы хотите, чтобы все разработчики Magic! Чтобы получить доступ к бесплатному протоколу MySQL от SoftInnov, вы просто сохраняете копию mysql-protocol.r в каталоге библиотеки. В следующем примере показана страница RHTML, использующая этот протокол для доступа к базе данных.

8.4.6 Управление встроенным кодом Rebol

Безопасность - важная характеристика Magic!. Очень важно контролировать действия программиста, чтобы не допустить нарушения стабильности сервера или запуска вредоносных скриптов, которые читают или удаляют данные, им не принадлежащии. Чтобы решить эти проблемы, динамические страницы, разработанные с помощью Magic! по умолчанию соответствуют строгой политике безопасности:

- словарные слова нельзя переопределить,
- скрипт может читать файлы в каталоге библиотеки,
- скрипты могут читать и записывать файлы в текущем каталоге,
- все остальные каталоги и файлы в иерархии файловой системы недоступен.

Эта базовая политика содержится в файле magic.cgi и может быть легко адаптирована к вашим конкретным потребностям. Вы также должны знать, что любой код Rebol, содержащийся в Magic! страница тщательно контролируется, чтобы перехватить любые возможные ошибки выполнения.

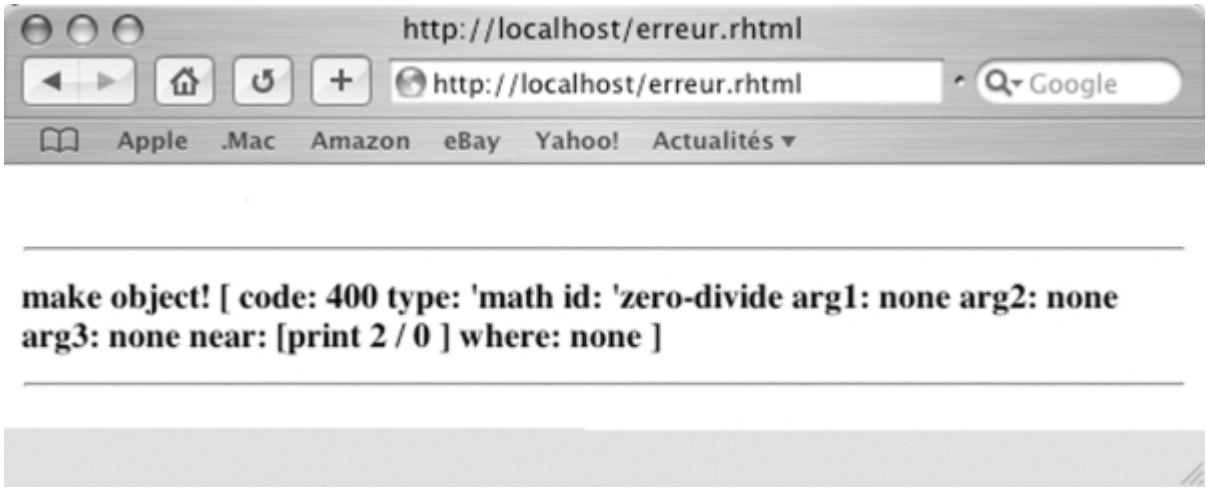

Рисунок 4-9. *Ошибка выполнения, обработанная Magic!.*

Если возникает проблема, оценка кода Rebol немедленно останавливается, и браузер возвращает сообщение об ошибке для отображения.

8.4.7 Обработка типов MIME

Если вы хотите создать что-либо, кроме HTML-страницы, вам следует выбрать другой тип MIME, изменив содержимое HTTP-заголовка, возвращаемого клиенту. Благодаря этой опции вы можете создавать данные XML, ASCII, изображения или даже PDF-документы.

Чтобы указать тип MIME, вы используете слово header, за которым следует строка символов, содержащая тип содержимого HTTP, который вы хотите использовать.

Следующий пример показывает, насколько легко создать PDF-документ с помощью библиотеки для создания PDF-файлов Габриэле Сантилли (доступной на его сайте http://web.tiscalinet.it/rebol/index.r). Чтобы запустить пример ниже, не забудьте сохранить копию файла pdf-maker.r в каталог библиотек Magic!.

Поскольку сгенерированный файл представляет собой документ PDF, содержимое динамической страницы представляет собой один тег , т.к. теги HTML бесполезны.

```
<rebol>
   library %pdf-maker.r
   header "Content-type: application/pdf"
   print to-string layout-pdf [
       [
       textbox ["I am a PDF document"]
       circle 50 250 20
       ]
   ]
</rebol>
```

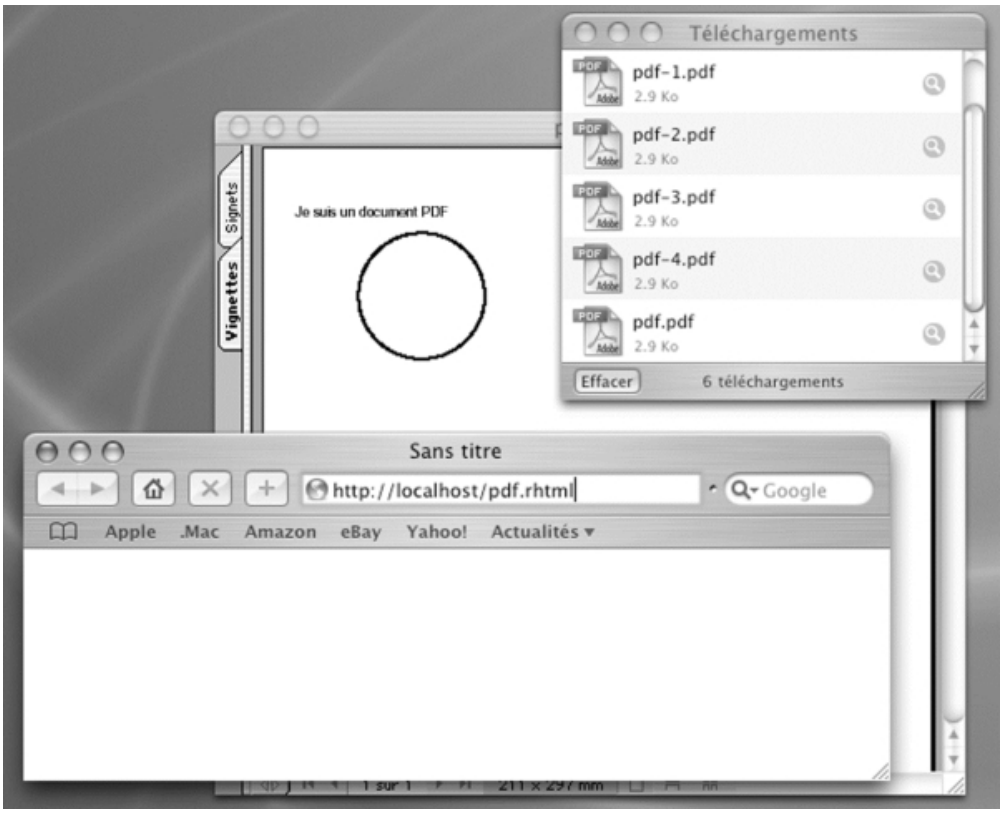

Рисунок 4-10. *Создание PDF-документов.*

8.4.8 Управление куки

Файлы cookie можно управлять с помощью слов setcookie и getcookie. Они предназначены для отправки и получения файлов cookie соответственно. Эти два слова можно использовать в любой точке страницы RHTML и напрямую изменять заголовок http, сгенерированный Magic!. setcookie использует два обязательных параметра: имя файла cookie и его значение. Вы можете точно указать поведение setcookie с помощью таких уточнений, как /expire, который устанавливает срок действия файлов cookie, /domain и /path, которые указывают поле, к которому прикреплен файл cookie, и путь доступа /secure, чтобы избежать отправки cookie на небезопасный сервер. getcookie намного проще в использовании, поскольку он принимает только один параметр, имя файла cookie, для которого вы хотите получить значение.

Следующий пример представляет собой страницу RHTML, которая генерирует файл cookie с именем test, устанавливая время создания на время сеанса и разрешая доступ со всех URL-адресов исходного сервера. Значение этого файла cookie изменяется путём присвоения случайного числа. С помощью getcookie текущее содержимое файла cookie извлекается и отображается на странице RHTML.

```
<html>
<head></head>
<body>
 <rebol>
     random/seed now/time/precise
     v: random 100
     setcookie/path "test" v "/"
 </rebol>
 The value of the cookie is:<rebol>print getcookie "test"</rebol>
 </body>
 </html>
```

8.4.9 Управление сессиями

Magic! является значительным подспорьем в одном из важных аспектов написания веб-приложений в Rebol, обеспечивая поддержку сеансов. Их цель - дать возможность программе идентифицировать пользователя и хранить данные, которые могут использоваться на различных страницах веб-приложения, к которому они обращаются.

Этот механизм служит для компенсации архитектурного выбора, сделанного создателями протокола http: тот факт, что HTTP не имеет состояния, и веб-серверы забывают каждый запрос после того, как они ответили на него. Ситуацию можно резюмировать одним простым предложением: пользователь веб-приложения не существует между двумя страницами. Идея сеансов состоит в том, чтобы присвоить пользователю идентификатор и восстанавливать его тем или иным способом с каждым их запросом. Как только у пользователя есть ссылка и он известен, появляется возможность хранить данные на сервере и упрощать дизайн приложения, избегая использования скрытых переменных (полей, скрытых внутри HTML-форм) и других подобных приёмов.

Magic! сохраняет переменные сеанса в объекте, хранящемся в каталоге, на который указывает слово m-sessions-path в magic.cgi. Для каждого пользователя создаётся уникальный объект; имени файла присваивается уникальное имя: MAGICSESSID.

Чтобы создать сеанс, вы используете слово session-start, которое генерирует идентификатор сеанса. Используя уточнение /cookie, MAGICSESSID будет автоматически передаваться при каждом взаимодействии между клиентом и сервером через cookie.

Используя уточнение /expire (истечение срока действия), вы можете указать время, отведённое для сеанса, переопределив значение по умолчанию. Это определяет продолжительность времени бездействия пользователя до того, как сеанс будет считаться неактивным и может быть уничтожен. Вы можете принудительно отменить сеанс с помощью слова session-destroy. Чтобы убедиться, что сеанс был запущен, вы можете использовать слово session? который возвращает логическое значение.

С помощью слова session-vars вы можете создавать переменные сеанса и назначать им начальное значение. После создания эти переменные становятся доступными на разных страницах вашего приложения с помощью специального объекта, называемого session, каждое свойство которого соответствует переменной сеанса.

В следующем примере динамическая страница проверяет, активен ли уже сеанс, и, если нет, запускает новый сеанс, который завершится через десять минут бездействия. MAGICSESSID нового сеанса автоматически передаётся в cookie.

Также объявляется переменная сеанса last-visit. Затем сценарий отображает идентификатор пользователя и значение last-visit, которое содержит время последнего посещения страницы пользователем. Затем текущее время сохраняется в переменной сеанса.

```
<html>
<head></head>
<body>
    <rebol>
        if not session? [
            session-start/cookie/expire 0:10:00
            session-vars [ [ last-visit: "" ] ]
        ]
        print [ "Your MAGICSESSID is " MAGICSESSID "<br>" ]
        print [ "The time of your last visit was " session/last-visit ]
        session/last-visit: now/time
    </rebol>
</body>
</html>
```

Рисунок 4-11. *Время последнего посещения сохраняется в переменной сеанса.*

Это все, что нужно для создания динамических веб-страниц с помощью Rebol и Magic!. Эта комбинация облегчает быструю и лёгкую разработку веб-приложений, поскольку избавляет программиста от множества неприятных деталей, связанных с этим.

8.5 Обработка XML-документов

Вам действительно нужен XML, когда вы используете Rebol? Ответ на этот вопрос не так очевиден, как может показаться на первый взгляд. Действительно, на бумаге Rebol и XML являются скорее конкурентами, чем партнёрами, поскольку каждый из них основан на собственном универсальном формате для организации и хранения данных.

8.5.1 Проблема интероперабельности

Rebol - это метаязык, а сценарий Rebol изначально представляет собой просто структуру данных, элементы которой становятся инструкциями только во время оценки. Это означает, что синтаксис Rebol позволяет писать сценарии, базы данных или файлы конфигурации. Формат блока отлично адаптирован для хранения данных и их передачи по протоколам TCP/IP.

В следующем примере показано использование Rebol в качестве формата для хранения информации:

```
[
    expiry-date: 15/1/2003
    filename: %journal.txt
    users: [
        [ "Olivier" category: 3 ]
    ]
]
```

Если Rebol и XML работают в одном поле, какой интерес в их совместной работе?

Проблема просто в том, что не все используют Rebol для разработки своих информационных систем. Разумеется, организация может выбрать IOS в качестве сервера, разрабатывать веб-приложения с помощью Rebol/Command и использовать блоки собственного формата Rebol для хранения и передачи данных. Проблема возникает, когда он желает связаться с другой организацией, бизнес-приложения которой используют другие технологии. В этом случае XML - просто единственное решение.

Rebol закрыт и включает в себя только функции, необходимые для обмена данными между его различными версиями. Однако выбор Rebol не отрезает вас от остального мира. Rebol понимает и говорит XML, обратное неверно ...

8.5.2 Встроенный парсер Rebol

Интерпретатор Rebol включает в себя синтаксический анализатор, который может преобразовывать XML-документ в группу легко используемых блоков. XML-документ также переведён в собственные форматы Rebol для оптимизации его использования. Фактически, вам нужно знать только одно слово: parse-xml. Он принимает единственный параметр строки данных, отформатированных в соответствии со спецификациями XML.

Парсер не проверяет тип. Это означает, что он проверяет синтаксис документа, но не достоверность содержащихся в нем данных. Если в XML-коде указано DTD ((Document Type Definition (определение типа документа)), оно будет просто проигнорировано.

Начнём с создания XML-файла с именем contacts.xml с помощью простого текстового редактора. Этот документ содержит список людей, идентифицированных по имени и имени.

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<contacts>
    <person sex="female">
        <name>Dziubak</name>
        <firstname>Nathalie</firstname>
    </person>
    <person sex="male">
        <name>Auverlot</name>
        <firstname>Olivier</firstname>
    </person>
</contacts>
```

Чтобы передать этот файл интерпретатору Rebol, вам нужно набрать:

```
parse-xml read %contacts.xml
```

Слово read считывает XML-файл и возвращает его содержимое в виде символьной строки, которая передаётся синтаксическому анализатору Rebol XML. Данные XML транскрибируются в блоки Rebol, которые отображаются на экране.

8.5.3 Как использовать данные?

Перед извлечением информации из этих блоков вы должны присвоить результат синтаксического анализа XML слову. После завершения этой операции вы можете просто использовать стандартные слова Rebol для обработки списков данных. В отличие от других языков, Rebol не имеет таких библиотек, как DOM или SAX, по той простой причине, что они не очень полезны для Rebol. Фактически, после того, как синтаксический анализатор проанализировал файл, Rebol больше не обрабатывает XML, а только получившиеся структуры данных Rebol.

В памяти интерпретатора Rebol данные файла contacts.xml хранятся следующим образом:

```
[document none [
    ["contacts" none [
        "^/^-" [
            "person" ["sex" "female"] [
                "^/^-^-" ["name" none ["Dziubak"]]
                "^/^-^-" ["firstname" none ["Nathalie"]] "^/^-"
            ]
        ]
         "^/^-" [
            "person" ["sex" "male"] [
                "^/^-^-" ["name" none ["Auverlot"]]
                "^/^-^-" ["firstname" none ["Olivier"]] "^/^-"]] "^/"
            ]
        ]
    ]
]
```

На первый взгляд, это отображение путаницы блоков и возвратов каретки, обозначенных управляющими символами "^/", может немного расстроить.

Не волнуйтесь, это не очень сложно понять, и, особенно, с этой организацией данных очень просто работать в Rebol. Фактически, этот список блоков составлен по следующим правилам:

- каждый объект XML представлен списком, состоящим из трёх значений организованы в модели [элемент атрибут содержимое],
- слово document соответствует корню XML-дерева,
- атрибуты представлены в виде пар имя атрибута - значение,
- если присутствует один или несколько атрибутов, они включаются в список,
- отсутствие атрибутов в теге XML обозначается значением none,
- символы возврата каретки ("^/") предназначены для улучшения отображение данных при использовании печатного слова. Вы можете просто игнорировать их при анализе древовидной структуры документа.

Из этих правил вы можете написать сценарий для доступа к данным в файле contacts.xml. Эта программа отображает имя и фамилию каждого указанного человека. Для каждого человека в коде отображается сообщение, показывающее значение атрибута из тега , указывающее его пол.

В сценарии используется рекурсивная процедура analyse, которая принимает в качестве параметра блок, по которому нужно пройти. Блоку присваиваются слово data, из которого вы можете сделать вывод, что первый, второй и третий элементы соответствуют тегу XML, списку атрибутов и значению содержимого тега. С помощью простых тестов вы также можете отобразить данные, извлечённые для XML-документа. Также возможно присвоить результаты словам для последующего использования.

```
REBOL [
        subject: "Traverse contacts.xml"
]
analyse: function [ data ] [ value ] [
    if block? data [
        if data/1 = "person" [
            print [ "New person" data/2 ]
        ]
        if any [
            data/1 = "name"
            data/1 = "firstname"
        ] [ print data/3 ]
        if not none? data/3 [
            value: data/3
            if (length? value) > 1 [
                forall value [
                    if block? value [
                        analyse first value
                    ]
                ]
            ]
        ]
    ]
]
```

Чтобы использовать функцию analyse, все, что вам нужно сделать, это вызвать её с параметром блока, содержащего данные из файла XML. Это можно сделать с помощью простой строки кода:

```
analyse first third parse-xml read %contacts.xml
```

8.5.4 Внутренности парсера

Сам Rebol XML написан на Rebol. Команда source parse-xml введенная в консоль Rebol, покажет вам, что вызов слова parse-xml на самом деле вызывает метод parse-xml объекта с именем xml-language. Затем вы можете использовать команду source xml-language для обнаружения содержимого этого объекта, который содержит свойства и методы синтаксического анализатора XML.

Обратите внимание, что у вас есть возможность изменить поведение анализатора XML, чтобы адаптировать его к вашим потребностям. Свойства, методы и правила действительно могут быть переопределены по вашему желанию.

Кроме того, свойство xml-language/verbose, которое содержит логическое значение, разрешает выбор (или нет) режима, в котором синтаксический анализатор XML отображает свою активность при анализе данных XML. Метод xml-language/check-version также очень интересен, поскольку его деактивация позволяет избежать отображения номера версии, указанного в заголовке XML-файла.

```
xml-language/verbose: false
xml-language/check-version: false
```

8.5.5 Лучший синтаксический анализатор XML

Поскольку код синтаксического анализатора XML доступен, некоторые программисты попытались улучшить его функциональность. Так обстоит дело с Гэвином Маккензи, который разработал улучшенную версию, которую можно найти в Интернете (http://www.rebol.org). Его версия поддерживает разделы cdata, указывающие теги, которые следует игнорировать во время анализа, и может распознавать элементы внутреннего DTD.

Перед тестированием его версии вы должны изменить документ contacts.xml, чтобы добавить новые элементы, такие как внутренний DTD и раздел CDATA.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE contacts [
<!ELEMENT contacts (person+)>
<!ELEMENT person (nane,firstname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ATTLIST person sex (male|female) #REQUIRED>
]>
<contacts>
        <person sex="female">
                <name>Dziubak</name>
                <firstname>Nathalie</firstname>
        </person>
        <![CDATA[ TEXT NOT USED BY THE XML PARSER]]>
        <person sex="male">
                <name>Auverlot</name>
                <firstname>Olivier</firstname>
        </person>
</contacts>
```

Чтобы использовать синтаксический анализатор XML Гэвина Ф. Маккензи, вы должны включить в свой сценарий библиотечный скрипт xml-parse.r с помощью команды do.

Затем все, что вам нужно сделать, это использовать parse-xml+ для XML-файла, просто используя команду parse-xml+ read% contacts.xml.

По сравнению со стандартным парсером XML Rebol, parse-xml + предоставляет больше информации о документе и его анализ более точен. DTD хранится в подмножестве элементов блока документа. Разделы cdata становятся простыми комментариями.

Для получения дополнительной информации Гэвин Ф. Маккензи предложил оригинальное решение, которое состоит из преобразования XML-документа XML в древовидную структуру, состоящую из объектов. Сценарий библиотеки называется xml-object.r, который должен быть включен в ваши сценарии с помощью do. Следующие инструкции преобразуют ваш документ contacts.xml в группу объектов в контексте объекта obj-xml:

```
obj-xml: make object! xml-to-object parse-xml+ read %contacts.xml
```

Корневой элемент XML-дерева соответствует объекту document, вы можете вывести версию XML из свойства obj-xml/document/version, а DTD можно получить через свойство obj-xml/document/subset. Для каждого подобъекта значение value? позволяет узнать значение элемента XML. Таким образом, становится очень просто извлечь имена всех людей в списке контактов:

```
foreach person obj-xml/document/contacts/person [
    print person/name/value?
]
```

Если элемент содержит комбинацию подэлементов и текста, свойство content? позволяет получать различные значения в блоке. Теги XML представлены словами, а информация - строками символов. Чтобы проверить это свойство, вы можете извлечь содержимое элемента следующим образом:

```
print mold obj-xml/document/contacts/content?
```

Такой объектный подход к XML-документам позволяет программисту легко извлекать из них информацию.

8.5.6 Генерация XML

Использование XML с Rebol не ограничивается только чтением файлов. Во многих ситуациях необходимо сгенерировать XML-документы из структур данных Rebol. Хотя эта функция не входит в базовый словарь, её нетрудно реализовать.

Предположим, вы работаете в компании, каталог запчастей которой структурирован в блоках Rebol. В идеале эти элементы должны быть преобразованы с помощью функции to-xml следующим образом:

```
to-xml [
    catalogue [
        part [ ref "243a" name "nut" ]
        part [ ref "784c" name "bolt" ]
    ]
]
```

Напишем эту функцию. Постепенно XML-документ будет сохранен в переменной docxml. Функция просматривает блок данных по два элемента за раз. Первое значение всегда соответствует элементу XML и вызывает создание открывающего тега XML со словом to-tag. Если второе значение представляет собой символьную строку, данные добавляются в docxml и добавляется закрывающий тег. С другой стороны, если встречается блок, функция to-xml вызывается снова рекурсивно для обхода всего дерева данных.

```
docxml: copy {<?xml version="1.0" encoding="iso-8891-1" ?>^/}

to-xml: function [ blk ] [ ] [
    forskip blk 2 [
            append docxml to-tag first blk
            either block? second blk [
                    to-xml second blk
            ] [
                    append docxml form second blk
            ]
            append docxml to-tag join "/" first blk
    ]
    docxml
]
```

Как видите, с Rebol очень легко импортировать и экспортировать данные в формате XML. В рамках интернет-приложений использование XML во многих случаях представляет очевидный интерес. Вам не составит труда создать привязку XML для веб-браузера или даже удалённых процедур с использованием таких протоколов, как SOAP или XML/RPC.

8.6 Использование веб-служб

Сейчас мы живём в эпоху веб-сервисов, то есть удаленного использования программного обеспечения, распределённого по множеству серверов. Есть несколько способов использовать такую архитектуру. Если доступная поддержка SOAP для Rebol на (http://compkarori.com/soap) ещё не является полностью исчерпывающей, Rugby Маартена Купманса является эффективным брокером, последняя версия доступна на http://www.hmkdesign.dk/rebol/page0/page0.html и очень мощная библиотека XML-RPC под названием RebXR были разработаны Андреасом Болка (http://earl.strain.at). Rebol Technologies недавно выпустила экспериментальную версию Rebol/Services, специально предназначенную для разработки веб-сервисов в Rebol.

8.6.1 Введение в XML-RPC

XML-RPC (Remote Procedure Call (удаленный вызов процедур)) - это протокол связи, разработанный Дэйвом Винером. Он использует транспортный уровень, подобный HTTP, и XML для кодирования данных. Основываясь на модели запрос/ответ, он позволяет клиенту использовать удалённую службу независимо от технологии, используемой на клиенте и сервере. Чтобы узнать больше, вы можете посетить веб-сайт XML-RPC (www.xmlrpc.com), на котором собраны документы, инструменты и список доступных услуг.

8.6.2 Использование XML-RPC

Чтобы использовать XML-RPC с Rebol, необходимо загрузить последний архив с сайта Андреаса Болка. Это позволяет вам писать сервисы XML-RPC на основе технологии cgi, а также клиентов, используя минимальное количество строк кода. Эта библиотека использует синтаксический анализатор XML Гэвина Ф. Маккензи, который вы должны включать в свои проекты.

Чтобы написать клиент, вам нужно будет использовать пять файлов: xml-object.r, xml-parser.r, xmlrpc-client.r, xmlrpc-marshaler.r и xml-writer.r. Для Rebol сервис соответствует объекту, унаследованному от xmlrpc-client.

Используя метод set-server, вы должны предоставить URL-адрес сервера, предоставляющего услугу. Любой используемый прокси может быть указан с помощью метода set-proxy. После завершения настройки всё, что остаётся сделать, это запросить выполнение удаленной службы с помощью метода exec. Требуется блок, содержащий имя удалённого метода и список требуемых им параметров. В следующем примере вы используете сервис Covers.wiw.org, который представляет собой базу данных песен, исполненных другими артистами:

remote: make xmlrpc-client [ ] remote/set-proxy tcp://mon-proxy.domaine.org:8080 remote/set-server http://covers.wiw.org:80/RPC.php print mold remote/exec [covers.Covered "peter gabriel"]

При вызове метода covers.Covered код ищет имена исполнителей, исполнивших каверы на песни в исполнении Питера Гэбриэла. Вам возвращается стандартный блок Rebol, которым вы можете управлять с помощью стандартных слов Rebol.

Простой и эффективный, RebXR - прекрасный пример того, как Rebol и XML дополняют друг друга.

8.7 Плагин Rebol/View

Появление версии плагина для браузера стало долгожданным событием в сообществе программистов Rebol. Его существование открывает множество возможностей для разработки приложений X-internet. Это видение активной сети, в которой лёгкие приложения, диалекты и данные легко обмениваются между машинами, наконец, может стать широко используемой реальностью.

8.7.1 Маленький, но мощный

Такой продукт явно выдерживает сравнение с Java и её апплетами.

Однако этот плагин имеет множество преимуществ, которые могут сделать его будущей звездой Интернета. Принципиальная разница между подключаемым модулем Rebol и Java заключается в огромной разнице в размере, поскольку размер подключаемого модуля составляет всего 600 КБ.

Это лёгкое программное обеспечение можно загрузить и установить на клиентском компьютере за несколько секунд. Даже если у пользователя только медленное коммутируемое соединение, развёртывание плагина не представляет никаких трудностей. Ни на минуту не думайте, что небольшой размер плагина связан с ограниченной функциональностью! Плагин обладает всеми функциональными возможностями последней версии Rebol/View и, таким образом, предоставляет все его возможности; сетевые протоколы, пользовательские интерфейсы, графика, анимация и звук. Программист имеет полный доступ к богатству и универсальности Rebol. Это уже не просто вопрос добавления милой маленькой анимации на веб-страницу, а создание полноценных приложений в HTML-документах.

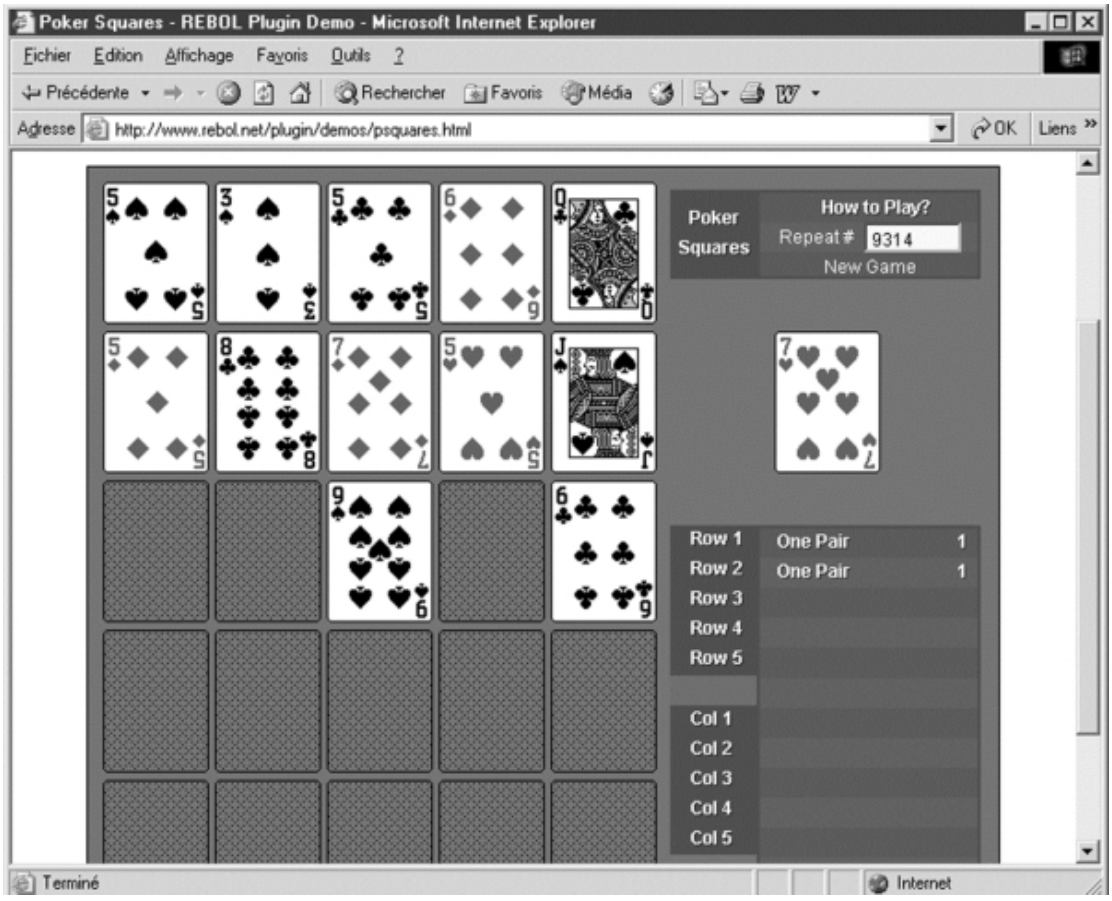

Рисунок 4-12 *Плагин позволяет создавать полноценные приложения.*

Преимущество этих реблетов в том, что они чрезвычайно компактны и могут быть загружены за несколько секунд. Их работа обеспечивается виртуальной машиной, которая использует гораздо меньше ресурсов, чем Java. В отличие от апплетов, скорость загрузки и производительность больше не должны быть препятствием.

8.7.2 Вставка плагина на HTML-страницу

Чтобы протестировать плагин Rebol, вам понадобится компьютер под управлением Microsoft Windows. Поддерживаемые браузеры: Internet Explorer, Firefox, Mozilla и Opera. На данный момент плагин доступен для этих конфигураций, которые Rebol Technologies считает приоритетными. Выбор оправдан тем, что Microsoft Windows в настоящее время является наиболее часто используемой операционной системой, и изначально цель заключалась в том, чтобы предоставить продукт максимальному количеству людей. Пользователи Unix и Mac, а также любители бесплатного программного обеспечения должны быть уверены, что портирование плагина на Linux и Mac OS X запланировано.
Для использования плагина просто необходимо вставить тег object на HTML-страницу. Этот тег относится к загружаемому браузером компоненту, который содержит интерпретатор Rebol/View 1.3. При первом удалённом использовании плагина пользователю предлагается принять его установку одним щелчком мыши. После этого браузер готов к выполнению сценария Rebol, указанного на HTML-странице.

```
<object id="myplugin" width="200" height="100"
classid="CLSID:9DDFB297-9ED8-421d-B2AC-372A0F36E6C5"
codebase="http://www.rebol.com/plugin/rebolb5.cab#Version=0,5,0,0">
    <param name="LaunchURL" value="http://localhost/script1.r">
</object>
```

Тег <object> использует различные атрибуты для управления рядом аспектов плагина. Атрибуты id содержат имя, которое программист присвоил плагину на странице HTML. Атрибуты width и height представляют ширину и высоту окна, которое будет использоваться плагином. Classid - это подпись плагина, которую, разумеется, менять не следует. Атрибуты кодовой базы указывают адрес, откуда плагин может быть загружен, если он ещё не был установлен или нуждается в обновлении. Атрибут LaunchURL в теге указывает местоположение реблета Rebol, который должен быть выполнен плагином.

В листинге выше HTML-страница запрашивает выполнение программы script1.r. Это просто традиционный сценарий Rebol, использующий VID (Visual Interface Dialect) для описания своего графического интерфейса. Например, все, что нужно для отображения текста в окне плагина, - это следующий код:

```
REBOL [
 author: "Olivier Auverlot"
]
view layout/size [ title "Reblet !" ] 300x100
```

8.7.3 Параметры конфигурации

Плагин предоставляет ряд параметров конфигурации, к которым можно получить доступ через тег . С BgColor вы можете установить цвет фона. Для этого вы просто указываете шестнадцатеричное значение компонентов RGB так же, как вы можете указать определённые цвета в HTML или CSS. Обычно используется то же значение, которое используется для фона HTML-страницы, содержащей Reblet.

Можно даже передать данные в Reblet, чтобы настроить его работу. Для этого у вас есть параметр Args, который содержит список значений, которые Reblet считывает перед запуском. Значения, содержащиеся в этом атрибуте, могут быть динамически сгенерированы с помощью серверного сценария или программы (CGI, страница RHTML или PHP, JSP или сервлет и т.д.). Для чтения содержимого параметра Args используется тот же метод, что и для чтения параметров, переданных интерпретатору из командной строки. Вы просто получаете доступ к свойству system/options/args.

В следующем примере устанавливается цвет фона плагина и передаётся список языков программирования через параметр Args.

```
<html>
<head><title>Rebol plugin test</title></head>
<body>
<object id="myplugin"
classid="CLSID:9DDFB297-9ED8-421d-B2AC-372A0F36E6C5"
codebase="http://www.rebol.com/plugin/rebolb5.cab#Version=0,5,0,0"
width="200" height="200">
<param name="LaunchURL" value="http://localhost/script2.r">
<param name="BgColor" value="#000000">
<param name="Args" value="Rebol Java Perl Python Ruby">
</object>
</body>
</html>
```

Для чтения данных из командной строки и отображения их в списке требуется всего несколько строк кода. После проверки наличия данных в свойстве system/options/args имена языков отделяются друг от друга стандартным символом разделения (пробелом). Различные строки помещаются в блок, а затем отображаются в стиле текстового списка (text-list).

```
if not none? system/options/args [
    languages: parse system/options/args ""
]
view layout/size [
    text-list 150x150 data languages
] 300x200
```

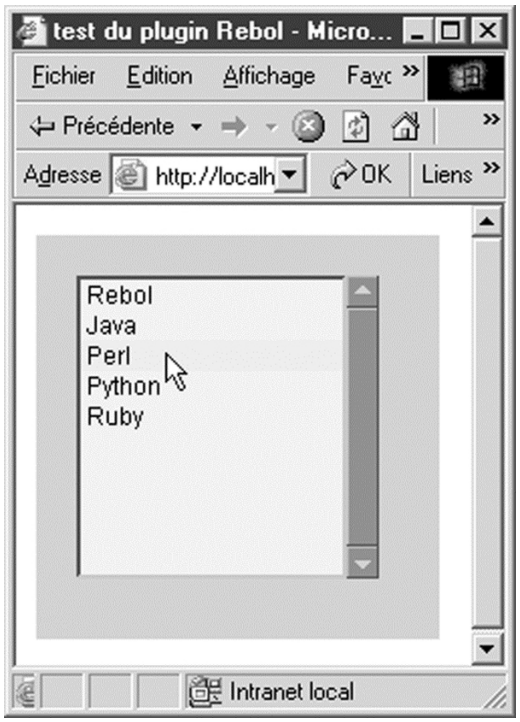

Рисунок 4-13. *Запуск реблета в браузере.*

Также доступен третий и последний параметр, называемый Version. Это соответствует ряду свойств, доступных в плагине Rebol для оптимизации передачи данных между клиентом и сервером.

8.7.4 Кеш, прокси и сжатие

Чтобы сделать реблеты максимально интерактивными, Rebol снабдил свой плагин продуманным механизмом кэширования, который позволяет оптимизировать загрузку реблетов по сети. Когда реблет используется впервые, его код сохраняется во временной папке на клиентском компьютере и в дереве каталогов, посвящённому Rebol.

Таким образом, в Windows с браузером Internet Explorer вы можете найти сохранённые реблеты в папке c:\windows\temp\rebol\plugin\ie\0\public\. Скрипты хранятся в папках, названных в честь сервера, с которого они были загружены. Чтобы обновить эти скрипты, если они были обновлены на сервере, плагин Rebol использует параметр под названием Version со следующим синтаксисом:

```
<param name="Version" value="3.0.0">
```

Если значение этого атрибута выше, чем номер версии реблета, хранящегося в кэше, реблет на сервере считается новой версией и будет загружен. В противном случае будет выполнен сохраненный реблет. Этот простой метод имеет много преимуществ. Используя параметр Version, разработчик веб-страницы, содержащей реблет, может обеспечить использование последней версии всеми пользователями. Этот кеш позволяет значительно снизить сетевой трафик, ограничивая загрузку только обновленными обновлениями. Наконец, время запуска reblet намного быстрее, поскольку оно хранится локально.

Наличие кеша также открывает новые возможности для разработчиков приложений, размещаемых в браузере. Фактически, кеш можно также рассматривать как файловое пространство, зарезервированное для плагина Rebol. В этой специальной древовидной структуре можно создавать и читать файлы.

Это относится только к выделенным каталогам Rebol и больше нигде на жестком диске пользователя. Нет вопроса о чтении или записи файлов вне выделенной структуры каталогов без того, чтобы менеджер безопасности Rebol запросил авторизацию пользователя. Единственный доступный каталог - это кеш.

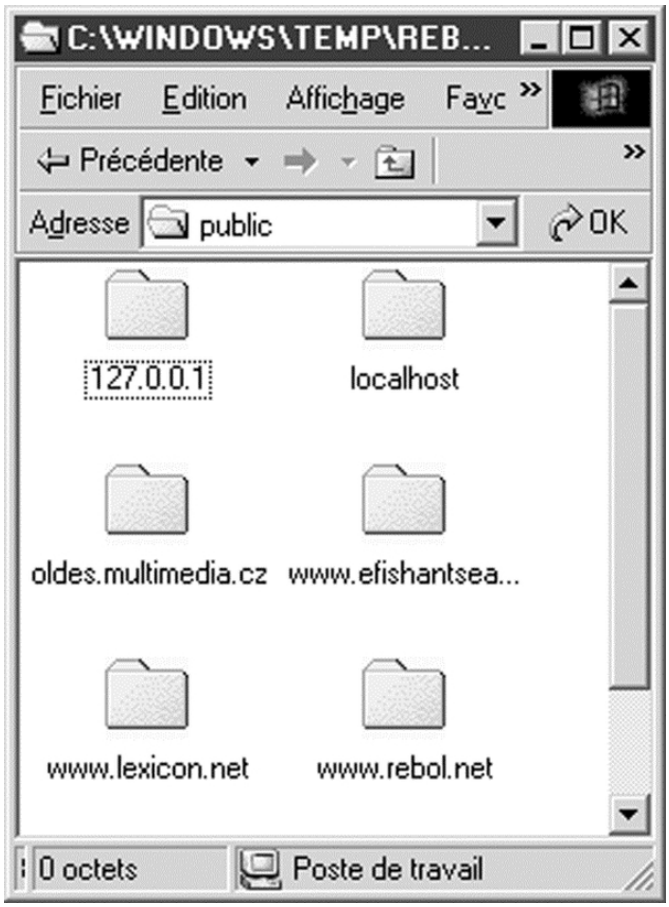

Рисунок 4-14. *Локальный кеш плагина*

Таким образом реблет может создавать временные файлы, хранить данные на локальном диске и даже создавать документы HTML или pdf. В следующем примере отображается форма для ввода имени и имени пользователя. После нажатия кнопки "Сохранить" данные сохраняются на клиенте в файле с именем data.txt. Этот файл создаётся в той же папке, что и реблет.

```
view layout/size [
    across
    lab "Name:" name: field "" return
    lab "First Name:" firstname: field "" return btn "Save" [
        write %data.txt reduce [
            (get-face name) " " (get-face firstname)
        ]
    ]
] 400x200
```

В дополнение к кешу, плагин Rebol предоставляет другие инструменты для значительного уменьшения сетевого трафика и сокращения времени запуска reblet.

Фактически, эти все механизмы, доступные во всех версиях Rebol, но которые приобретают новое измерение в рамках плагина. Например, read-thru позволяет оптимизировать чтение документов по сети. Благодаря этому реблет может легко прочитать содержимое файла по указанному URL-адресу. read-thru также использует кеш плагина Rebol. После подключения к серверу функция read-thru проверяет локальный кеш, чтобы увидеть, есть ли там запрошенный файл. Если это так, файл читается непосредственно из кеша на клиентском компьютере.

Например, реблет может загружать изображения и сохранять их в локальном кеше. При следующем использовании реблета повторно загружать изображения не потребуется. Стоит отметить, что если вы используете read-thru, данные автоматически сохраняются в кеше. С другой стороны, если определённая информация должна оставаться скрытой на сервере, программист всегда может использовать read и решить, что нужно и что не нужно сохранять на самом клиенте.

Плагин Rebol также включает слова compress и decompress для сжатия и расширения данных. Размер ресурсов, необходимых для запуска реблета (изображения, звуки и т.д.), может быть значительно уменьшен и, следовательно, сокращено время их загрузки. Все, что вам нужно для сжатия файлов, - это интерпретатор Rebol. Следующая команда сжимает изображение img.bmp и создает файл с именем img.dat, который затем можно сохранить на веб-сервере:

```
write/binary %img.dat compress read/binary %img.bmp
```

Теперь вы можете написать реблет, который загружает его, распаковывает, сохраняет в кеше плагина и отображает в окне браузера:

```
img: load to-binary decompress read-thru http://localhost/img.dat
view layout/size [
    origin 0x0
    image 179x250 img
] 179x250
```

Как обычно, Rebol очень много делает с минимумом кода. Можно даже дополнительно уменьшить размер сценария, используя уточнение /expand для load-thru, чтобы избежать использования слова decompress.

8.7.5 Взаимодействие с браузером

Можно взаимодействовать с содержимым веб-страницы, на которой размещен реблет, путем доступа к DOM (Document Object Model (объектной модели документа)). Это позволяет читать или изменять определённые аспекты среды, в которой работает реблет. Фактически, все это стало возможным благодаря вызову универсальной функции JavaScript, задача которой просто выполнить код JavaScript, предоставленный реблетом.
Эта функция должна быть вызвана оценкой и сохранена на странице, содержащей плагин:

```
function evaluate(code) {

  return eval(code);
}
```

Эта функция выполняет и возвращает результат кода JavaScript, переданный ей в качестве параметра. В коде плагина вызов функции evaluate() становится возможным с помощью слова do-browser, единственным аргументом которого является код JavaScript, передаваемый в функцию evaluate(). Это слово возвращает результат кода JavaScript. Чтобы лучше понять, вот реблет, который находит имя и номер версии браузеров, в которых он работает. Для этого необходимо получить доступ к свойствам appName и appVersion объекта JavaScript Navigator. HTML-страница, содержащая вызов реблета, также содержит код JavaScript, который хранится между тегами .

```
<html>
<head>
<title>Rebol plugin test</title>
<script language="javascript">
    function evaluate(code) {
        return eval(code);
    }
</script>
</head>
<body>
<object id="myplugin"
    classid="CLSID:9DDFB297-9ED8-421d-B2AC-372A0F36E6C5"
    odebase="http://www.rebol.com/plugin/rebolb5.cab#Version=0,5,0,0"
    width="100" height="50">
<param name="LaunchURL" value="http://localhost/script6.r">
</object>
</body>
</html>
```

Макет reblet содержит простую кнопку, которая при нажатии дважды вызывает функцию JavaScript evaluate() для запроса оценки свойств navigator.appName и navigator.appVersion. После того, как информация собрана, она отображается в диалоговом окне.

```
view layout/size [
    backdrop 255.255.255
    btn "Browser ?" [
        app-name: do-browser {navigator.appName}
        app-version: do-browser {navigator.appVersion}
        alert join "You are using " [
            app-name " " app-version
        ]
    ]
] 100x50
```

Рисунок 4-15. *Отображение сведений о браузере.*

Этот режим работы прост и имеет то преимущество, что не накладывает никаких ограничений. Все, что можно сделать в JavaScript, можно сделать с помощью плагина Rebol. Эта функция даёт плагину Rebol доступ ко всем ресурсам браузера.

С помощью этого метода плагин может читать или изменять значения полей HTML-формы, открывать или закрывать окна браузера, просматривать историю навигации, обрабатывать файлы cookie и т.д.

Включив все функции Rebol/View в плагин для браузера, Rebol, безусловно, совершил большой переворот. Благодаря своей свободе и переносимости, плагин решает проблемы распространения приложений Rebol, обеспечивая при этом полную функциональность на клиентских рабочих станциях, как профессионалами, так и широкой публике. Это лёгкое и инновационное решение должно привлечь внимание многих разработчиков.

8.8 Резюме

Rebol - действительно язык, посвящённый сетевому программированию. С его помощью вы можете получить доступ к ресурсам Интернета, определить свои собственные протоколы обмена данными и разрабатывать клиент-серверные и веб-приложения. Интернет принимает совершенно иную форму. это органическая сеть, состоящая из узлов, обменивающихся информацией и приложениями.

9. Ребол для профессионалов

Rebol имеет множество возможностей для привлечения профессиональных разработчиков. У него есть версия, предназначенная для электронного бизнеса, а другая - для группового программного обеспечения. Его универсальность, лаконичность и портативность делают его идеальным инструментом для быстрой разработки программного обеспечения.

9.1 Rebol/Command

В настоящее время язык Rebol доступен в пяти различных версиях, каждая из которых предназначена для конкретного использования.

Rebol/Core в основном используется для сетевых утилит и сценариев CGI. Rebol/View предусматривает разработку графических клиентских приложений. Rebol/SDK поддерживает разработку коммерческих приложений, которые могут распространяться в исполняемой форме. Rebol/IOS предназначен для групповой работы. Rebol/Command предназначен для разработки серверов для систем электронного бизнеса.

9.1.1 Концентрированная сила

Rebol/Command доступен для ряда операционных систем (Linux, Windows, AIX и др.). Текущая версия - 2.5.6. Он функционально эквивалентен для всех платформ, как Rebol/Core и Rebol/View. У вас есть возможность разработать приложение, не беспокоясь о том, на какой платформе оно будет работать. Программное обеспечение можно писать под Windows и запускать без изменений в Linux.

В нескольких сотнях килобайт интерпретатора Rebol/Command вы найдёте не только все слова и протоколы Rebol/Core, но также впечатляющее количество расширений, которые делают этот продукт невероятно продуктивным. Для веб-приложений, предназначенных для работы через Интернет или интрасеть, имеется поддержка SSL (Secure Sockets Layer) для безопасного обмена данными и FastCGI для разработки высокопроизводительных приложений. Rebol/Command также включает доступ к основным системам управления базами данных, доступным на рынке, возможность вызова библиотек собственного кода, запуск внешних приложений через оболочку хост-системы и основные алгоритмы шифрования.

9.1.2 Доступ к базе данных

Первый из многих талантов Rebol/Command - это лёгкость, с которой вы можете взаимодействовать с СУБД. Требуется всего шесть строк кода, чтобы открыть соединение, отправить запрос SQL, получить ответ и закрыть соединение. В зависимости от платформы, на которой он работает, Rebol/Command, к сожалению, не имеет доступа ко всем тем же базовым базам данных. Хотя все версии Rebol/Command изначально поддерживают Oracle и MySQL,

только версия для Windows имеет доступ к протоколу ODBC. Поэтому, если вы используете другую базу данных (Informix, PostgresSQL, SyBase, IBM DB2 и т.д.), На данный момент вы должны использовать машину с Windows (9x, Me, NT, 2000, XP).

Доступ к базе данных осуществляется с помощью трёх протоколов Rebol, называемых oracle, mysql и odbc. Итак, все, что вам нужно сделать для связи с базой данных - это использовать порт. Первый этап заключается в открытии соединения с параметрами, передаваемыми в виде URL. Параметры будут разными в зависимости от типа используемой базы данных.

Например, в случае ODBC вам нужно будет указать имя источника данных (DSN), тогда как для MySQL вам нужно будет указать имя базы данных.

Предположим, вы хотите подключиться к базе данных MySQL с именем "test", размещённой на машине по адресу 172.29.143.1 в вашей сети. Разумеется, вы должны указать имя пользователя ("homer") и пароль ("duff"). Подключение устанавливается следующим образом:

```
db: open mysql://homer:duff@172.29.143.1/test
```

Теперь вы можете установить порт, используя слово first. После этого вы передаёте свои SQL-запросы в базу данных с помощью слова insert и получаете результаты обратно из базы данных с помощью слова copy. После завершения этих операций вы должны закрыть соединение, закрыв два порта связи:

```
p: first db
insert p "SELECT * FROM myfile" print copy p
close p
close db
```

Этот пример возвращает все строки в таблице myfile в виде блока блоков, которые затем можно обрабатывать с помощью функций Rebol. Rebol/Command также поддерживает более сложные операции с базой данных, такие как транзакции и доступ к определениям таблиц и полей.

9.1.3 Доступ к оболочке

Оболочка - это командный интерпретатор вашей системы. В Linux это обычно bash (Bourne Again SHell). Это позволяет вам общаться с вашим компьютером без необходимости в графическом пользовательском интерфейсе.

Оболочка предоставляет внутренние команды (echo, set и т.д.) и позволяет запускать внешние приложения. Rebol/Command позволяет разработчикам взаимодействовать с оболочкой через слово call. Предположим, вы запускаете Rebol в Linux, вы можете получить содержимое корневого каталога с помощью команды call "ls /".

Это слово имеет несколько очень интересных уточнений. С помощью /error вы можете восстановить код ошибки, возвращаемый оболочкой. Вы также можете перенаправить стандартные потоки ввода и вывода с помощью /input и /output. С помощью /wait можно синхронизировать выполнение разных процессов: задача будет запущена только после завершения другой.

Для тех из вас, кто еще не купил копию Rebol/Command, эта функция была включена в Rebol/Core и Rebol/View с момента выпуска версий 2.6 и 1.3 соответственно.

9.1.4 Использование динамических библиотек

Динамическая библиотека - это набор собственных функций (и, возможно, данных), которые могут совместно использоваться приложениями на одной машине. Функции библиотеки помечены как общедоступные или частные и соответственно доступны или скрыты от приложений. Цель состоит в том, чтобы избежать необходимости включать один и тот же код непосредственно в каждое приложение, которое его использует. Это не только снижает использование памяти, но и упрощает обновление: достаточно обновить библиотеку, чтобы все программное обеспечение, использующее её, обновлялось одновременно.

В Windows динамические библиотеки имеют тип файла DLL. Что касается фанатов Tux, они находят их с не менее известным типом файлов SO.

Rebol/Command может использовать функции собственного кода в этих динамических библиотеках, которые обычно разрабатываются на C, C ++, Pascal или даже Visual Basic. Шаги довольно просты: вы должны сначала включить библиотеку, используя уточнение /library слова load, а затем объявить различные функции, которые вы хотите использовать с типом данных routine!. В этом случае функции считаются принадлежащими стандартному словарю Rebol.

Благодаря динамическим библиотекам вы можете повторно использовать функции нативного кода, разработанные для другого проекта, повысить производительность своего приложения, кодируя критические разделы в нативном коде, и особенно расширяя возможности Rebol за счёт интеграции библиотек, таких как те, которые разрешают доступ к каталогу LDAP или СУБД, напрямую не поддерживаемая Command.

Ещё лучшая новость для тех из вас, кто ещё не купил копию Rebol/Command, доступ к библиотеке был включён в Rebol/View с выпуском версии 2.7.6.

9.1.5 Шифрование данных

Шифрование данных - один из самых интересных аспектов Rebol/Command. Благодаря ему вы можете зашифровать и расшифровать информацию и, следовательно, безопасно передать её по сети или через Интернет. Интерпретатор имеет специальный порт с именем crypt, функция которого заключается в преобразовании данных в соответствии с выбранным алгоритмом. Вы можете использовать алгоритмы, основанные на традиционном симметричном ключе, а также на алгоритме RSA (Ривест, Шамир и Адельман), используя пару ключей (частный и открытый). Также возможно применить цифровую подпись (DSA) для аутентификации данных. Алгоритм DH (Diffie Hellman) используется в процессе авторизации подключения внешних устройств к сети. Для всех этих различных случаев Rebol/Command предоставляет инструменты, необходимые для генерации используемых ключей. Используя эти функции, вы можете передавать конфиденциальную информацию по сети с оптимальной безопасностью.

Несмотря на свой небольшой размер, Rebol/Command - чрезвычайно мощный инструмент. Он обладает всеми функциями, необходимыми для быстрого написания приложений электронного бизнеса с минимумом кода. Если случайно не хватает необходимой функции, у вас есть альтернатива - использовать внешние приложения через доступ к оболочке или, что ещё лучше, включить в свой проект библиотеки динамического нативного кода.

9.2 Rebol, скрипты CGI и MySQL

Как получить доступ к базе данных, если у вас нет Rebol/Command? Действительно, бесплатные версии (Core и View) не могут отправлять запросы SQL в реляционную базу данных. К счастью, ситуация не совсем безвыходная.

Поскольку Rebol является языком сетевого программирования, основанным на TCP / IP, решение заключается в написании специального протокола связи. Ненад Ракочевич создал и сделал доступным отличный протокол MySQL.

9.2.1 Знакомство с протоколом MySQL

Протокол MySQL Ненада Ракочевича работает на всех версиях Rebol. Используемый вместе с Rebol/Core, он позволяет писать компактные сценарии CGI, которые могут взаимодействовать с хорошо известной СУБД MySQL.

С Rebol/View вы можете писать приложения для автоматизации делопроизводства с доступом к MySQL или Reblets (распределённые сетевые приложения). Протокол Nenad бесплатен и даже может быть интегрирован в коммерческие приложения. Протокол полностью независим от исполняющей платформы и поэтому может использоваться во многих операционных системах (Linux, Mac OS X, Windows и т.д.).

Если вы не хотите покупать Rebol/Command, использование Rebol/Core и MySQL является отличной альтернативой для разработки приложений для электронного бизнеса.

9.2.2 Скачивание и установка протокола

Протокол MySQL Ненада Ракочевича доступен для загрузки с http://softinnov.org/rebol/mysql.shtml. (Вы также можете найти ссылку на его аналогичный протокол PostgresQL на этой странице.)

Протокол содержится в ZIP-архиве, который содержит подробное руководство для разработчиков.

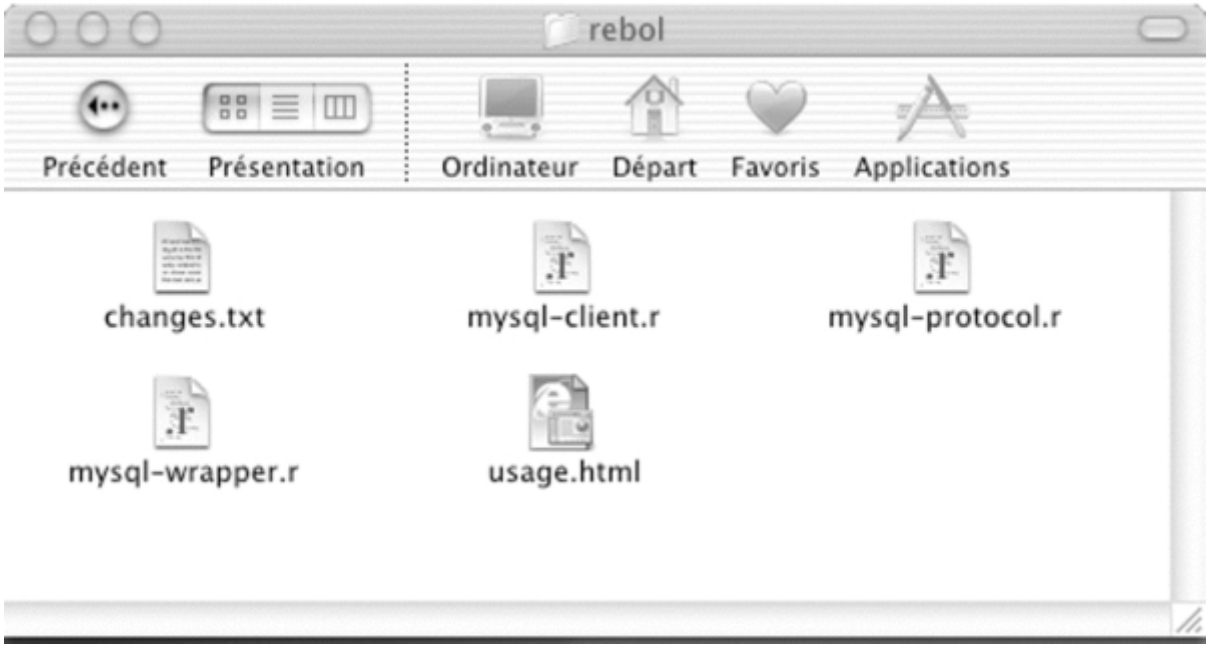

Рисунок 5-1. *Содержимое папки протокола MySQL.*

Среди новых файлов на жестком диске два наиболее важных - это mysql-usage.html и mysql-protocol.r.

Первый - это полная документация по протоколу, второй - код, который нужно включить в ваше приложение, чтобы иметь возможность взаимодействовать с MySQL. Чтобы включить его в свою программу, вы просто запускаете сценарий, содержащий протокол, используя слово do, за которым следует путь к файлу, содержащему протокол. После загрузки протокола в консоли Rebol отображалось сообщение "Протокол MySQL загружен". Теперь вы можете проверить, доступен ли новый протокол, используя команду print mold first system/schemes.

9.2.3 Использование протокола

Теперь осталось протестировать протокол. Для этого вы создадите небольшую базу данных MySQL под названием musicdb, которая содержит только одну таблицу discs. Её цель - хранить детали вашей коллекции записей. Он состоит из пяти полей:

```
CREATE TABLE discs (
 num SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
 artist CHAR(60),
 title CHAR(60),
 year SMALLINT,
 PRIMARY KEY (num)
)
```

База данных будет находиться на сервере Linux с IP-адресом 172.29.143.1. С помощью скрипта Rebol вы собираетесь загрузить эту таблицу данными. Данные, представляющие вашу коллекцию дисков, содержатся в блоке под названием data.

Каждый субблок в блоке данных соответствует диску и содержит имя исполнителя, название и год публикации диска.

Подключение к базе данных осуществляется с помощью инструкции open, за которой следует имя протокола и URL-адрес, в котором вы указываете имя пользователя и пароль (olivier: homer), IP-адрес (или имя) сервера, на котором размещается MySQL СУБД и имя базы данных. Затем эта инструкция возвращает порт, в который вы можете вставлять инструкции SQL и получать данные.

Чтобы загрузить таблицу дисков discs, все, что вам нужно сделать, это перебрать блок данных data с помощью цикла foreach, а затем использовать инструкцию SQL insert into.

Протокол обеспечивает большое удобство для программистов, позволяя заменять "?" для значений переменных, переданных в качестве параметров (первый вопросительный знак - это первая переменная и т. д.). Это сокращает создание сложной строки при использовании параметров приложения. В конце процесса не забудьте закрыть порт с помощью инструкции закрытия, чтобы освободить его.

Протокол обеспечивает большое удобство для программистов, позволяя заменять "?" для значений переменных, переданных в качестве параметров (первый вопросительный знак - это первая переменная и т.д.). Это сокращает создание сложной строки при использовании параметров приложения. В конце процесса не забудьте закрыть порт с помощью инструкции close, чтобы освободить его.

```
#!/Applications/core/rebol -qs
REBOL [ ]
do %/Library/rebol/mysql-protocol.r

data: [
    [ "Leonard Cohen" "Leonard Cohen in concert" 1994 ]
    [ "Sarah McLachlan" "Remixed" 2001 ]
    [ "Jean-Louis Murat" "Muragostang" 2000 ]
    [ "Leonard Cohen" "Ten new songs" 2001 ]
    [ "Peter Gabriel" "Us" 1992 ]
    [ "Huong Thanh" "Moon and Wind" 1999 ]
]

db: open mysql://olivier:homer@172.29.143.1/musicdb
foreach elem data [
    insert db [ "insert into discs values (0,?,?,?)"
    elem/1 elem/2 elem/3 ]
]
close db
```

9.2.4 Интеграция в CGI-скрипт

Чтобы запросить каталог дисков, вы собираетесь написать короткий сценарий CGI под названием music.cgi. Пользователь должен выбрать исполнителя из раскрывающегося меню, чтобы получить список альбомов исполнителя.

Единственная трудность с этим скриптом - правильно обрабатывать информацию, полученную из формы. Скрипт рекурсивен, и любой параметр будет отправлен ему при первом выполнении. Чтобы определить получение имени исполнителя, вы должны использовать обработку ошибок: код пытается прочитать предоставленную информацию и присваивает результат переменной artist-name.

Если скрипту не удаётся найти поле исполнителя во входных данных, он считает, что параметр не был передан, и инициализирует исполнителя со значением none.

```
REBOL [ ]
do %/Library/rebol/mysql-protocol.r
print "Content-type: text/html^/"
print {
    <HTML>
    <HEAD><TITLE>Music</TITLE></HEAD>
    <BODY>
    <H1>Disk catalogue</H1>
}
if error? try [
    query: make object! decode-cgi system/options/cgi/query-string
    artist-name: query/artist
] [artist-name: none ]
```

В любом случае вы должны увидеть выпадающий список с возможностью выбора, содержащий имена артистов в базе данных.

Во избежание дублирования в запросе SQL используется особая опция, а полученные имена сортируются в алфавитном порядке (order by).

Поскольку приложение не знает, сколько исполнителей находится в базе данных, а это может быть очень большое число, данные считываются из порта один за другим с использованием слова first (первый). Они динамически добавляются в HTML-форму.

```
print {
    <FORM name='catalogue' action='music.cgi' method='GET'>
    <SELECT name='artist' onChange="catalogue.submit();">
    <OPTION>Choose an artist
}
db: open mysql://olivier:homer@172.29.143.1/musicdb
insert db [
    "select distinct artist from discs order by artist"
]
while [ not empty? (name: first db) ] [
    print join "<OPTION>" name
]
print "</SELECT></FORM>"
```

Последний этап - запрос к базе данных, был ли выбран исполнитель.

Нет необходимости повторно открывать соединение, поскольку вы всегда подключены к базе данных MySQL. Результат запроса select присваивается discs переменных, которые представляют собой блок блоков, содержащий блок для каждой строки в ответе.

Создаётся таблица HTML, и данные отображаются в ней с помощью цикла foreach. Осталось только закрыть соединение с базой данных и завершить построение HTML-страницы.

```
if not none? artist-name [
    insert db [
        "select title,year from discs where artist=?"
        artist-name
    ]
    discs: copy db
    print [ "<BR><B>" nom-artiste "</B><BR><TABLE>" ]
    foreach d discs [
        print [ "<TR><TD>" d/1 "</TD><TD>" d/2 "</TD></TR>" ]
    ]
    print "</TABLE>"
]
close db
print "</BODY></HTML>"
```

Благодаря огромной работе Ненада Ракочевича ваши сценарии CGI, написанные с помощью Rebol/Core, теперь могут взаимодействовать с базой данных MySQL. Как вы могли заметить, выполнять SQL-запросы и читать данные очень просто.

9.3 Шифрование данных

Основная цель Rebol - облегчить передачу данных между разнородными компьютерными системами. Передача информации по сети требует учёта важнейшего фактора: безопасности. Язык общения должен иметь необходимые инструменты для обеспечения конфиденциальности и целостности данных. Для этого существует множество основных алгоритмов шифрования, доступных в Rebol/View/Pro, Rebol/SDK, Rebol/Command и Rebol/IOS.

9.3.1 Шифрование с симметричным ключом

Этот тип шифрования основан на одном ключе. Получатель расшифровывает сообщение тем же ключом, который отправитель использовал для его шифрования. Этот метод очень прост, но у него есть существенный недостаток: транспортировка ключа. Фактически, для того, чтобы у отправителя и получателя был один и тот же ключ, в то или иное время ключ должен был пройти между двумя пользователями. Кроме того, для всех пользователей, имеющих один и тот же ключ, этот метод не аутентифицирует документы и даже не проверяет целостность данных. Тем не менее, этот метод очень быстр, и, если вы знаете ограничения, его можно эффективно использовать в определённых обстоятельствах.

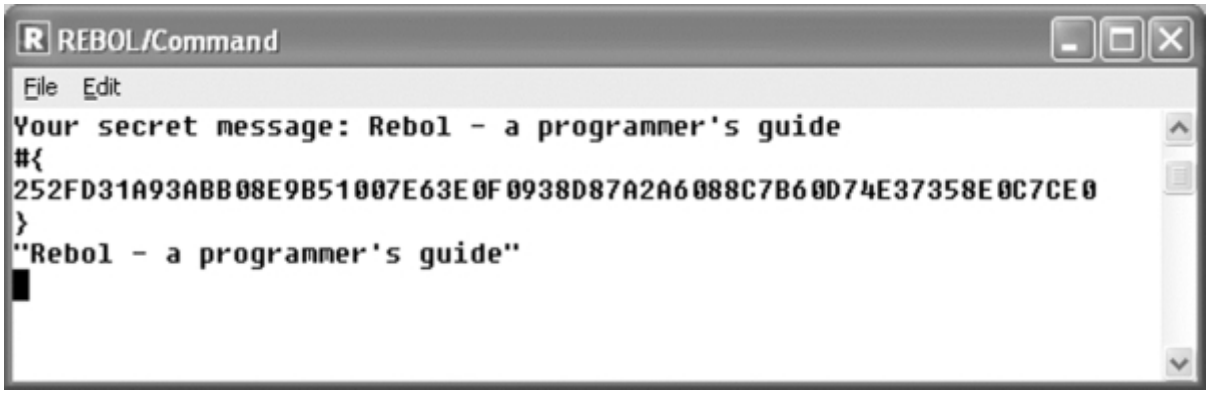

Рисунок 5-2. *Шифрование с симметричным ключом - это просто и быстро.*

Rebol включает протокол под названием crypt для шифрования данных.

Вы просто определяете параметры шифрования и вводите данные в порт для шифрования и дешифрования информации. В следующем примере выполняется шифрование сообщения, введенного пользователем.

Сначала из строки генерируется 128-битный ключ (16 байт).

```
the-key: checksum/secure "RebollobeRReboll"
```

Затем пользователя просят ввести сообщение:

```
msg: ask "Your secret message:"
```

Затем порт определяется с использованием функции 'encrypt (шифровать) протокола crypt. Алгоритм может быть типа 'blowfish или 'rijndael.

Свойство padding обеспечивает совместимость с другими приложениями шифрования.

```
cp: make port! [
    scheme: 'crypt
    algorithm: 'blowfish
    direction: 'encrypt
    strength: 128
    key: the-key
    padding: true
]
```

Сообщение зашифровывается, когда оно вставляется в порт cp. Вы просто используете слово copy для получения результатов шифрования.

```
open cp
insert cp msg
update cp
encrypted-msg: copy cp
close cp
print mold encrypted-msg
```

Чтобы расшифровать сообщение, вы должны использовать тот же метод (и особенно тот же ключ!), Изменив действие протокола на 'decrypt (расшифровать).

```
cp: make port! [
    scheme: 'crypt
    algorithm: 'blowfish
    direction: 'decrypt
    strength: 128
    key: the-key
    padding: true
]

open cp
insert cp encrypted-msg
update cp
print mold to-string copy cp
close cp
```

9.3.2 RSA шифрование

Шифрование RSA (Ривест, Шамир и Адельман) основано на использовании двух ключей, связанных друг с другом математическим соотношением:

- закрытый ключ никогда не раскрывается и не передаётся,
- открытый ключ рассылается всем получателям.

Этот алгоритм не только защищает содержимое сообщения, но и аутентифицирует его автора. Чтобы зашифровать сообщение, отправитель шифрует сообщение открытым ключом получателя (который известен всем). Последние расшифровывают сообщение своим закрытым ключом. Аутентификация сообщения стала возможной благодаря конфиденциальности закрытого ключа. Если отправитель использует свой закрытый ключ для шифрования сообщения, получателю ничего не остаётся, кроме как использовать открытый ключ отправителя для доступа к информации. Если сообщение успешно расшифровано, это означает, что сообщение было отправлено указанным отправителем. Этот метод также позволяет избежать отказа: эмитенты не могут утверждать, что они не отправляли сообщение.

Одним из аспектов шифрования RSA является то, что длина сообщения не может превышать длину ключей. Также RSA медленнее, чем симметричные алгоритмы. По этим причинам основным практическим применением шифрования RSA является обмен временными ключами для использования с симметричными алгоритмами и "подписание" документов.

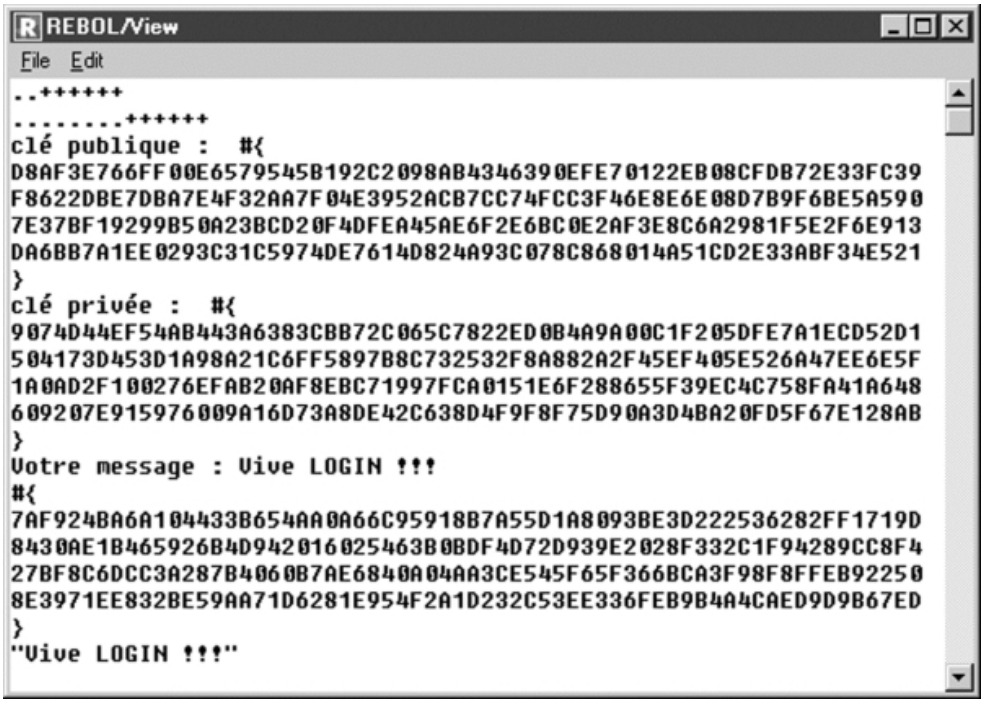

Рисунок 5-3. *RSA использует как закрытый, так и открытый ключ.*

Для хранения двух ключей Rebol использует объект, созданный словом rsa-make-key. Затем ключи могут быть сгенерированы (rsa-generate-key), указав длину ключей и простое число, которое будет использоваться алгоритмом. Свойства n и d объекта, содержащего ключи, - это открытый ключ и закрытый ключ соответственно.

Слово rsa-encrypt может затем использовать эти два ключа для шифрования и дешифрования данных.

```
keys: rsa-make-key
rsa-generate-key keys 1024 3
print [ "public key: " (mold keys/n) ]
print [ "private key: " (mold keys/d) ]

msg: ask "Your message : "

msg-code: rsa-encrypt/private keys (to-binary msg) print mold msg-code
msg-decode: rsa-encrypt/decrypt keys msg-code
print mold to-string msg-decode
```

9.3.3 Diffie Hellman

Алгоритм D-H может устанавливать безопасные соединения в открытых сетях. Принцип также основан на генерации двух ключей. У каждой стороны есть закрытый ключ, который никогда не транслируется и не перемещается. У них также есть открытые ключи, предназначенные для использования между машинами, между которыми должна быть установлена безопасная связь.

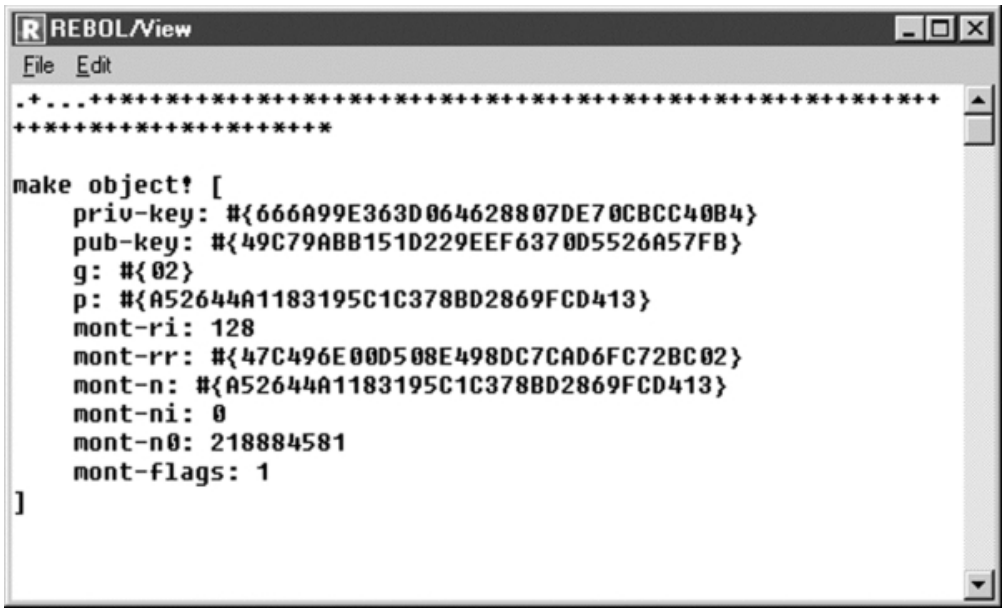

Рисунок 5-4. *Генерация двух ключей с D-H*

После обмена открытыми ключами два корреспондента генерируют сеансовый ключ, который будет использоваться для симметричного шифрования между двумя сторонами.

Использование этого метода шифрования продиктовано просто из-за его скорости: симметричное шифрование позволяет обеим сторонам шифровать и дешифровать данные в реальном времени.

С Rebol очень просто использовать алгоритм D-H. Генерация двух ключей для корреспондента выполняется с использованием слова dh-make-key с уточнением /generate. Аргументами являются длина ключа и номер для заполнения генерации ключа. Затем вам нужно сгенерировать пару ключей со словом dh-generate-key.

```
keys: dh-make-key/generate 128 2
dh-generate-key keys
```

Последний шаг - сгенерировать сеансовый ключ из вашего закрытого ключа и открытого ключа, полученного от вашего корреспондента.

```
; переменная key-other-server содержит
; открытый ключ корреспондента
Session-key: dh-compute-key keys key-other-server
```

С этого момента просто используйте метод симметричного шифрования из начала главы. Два корреспондента независимо друг от друга сгенерируют идентичный сеансовый ключ и теперь могут безопасно обмениваться информацией.

9.3.4 Подписание документов с DSA

DSA (Digital Signature Algorithm (алгоритм цифровой подписи)) позволяет документу иметь цифровую подпись. Он также полагается на использование закрытых и открытых ключей. Его очень просто использовать, поскольку отправитель использует свой закрытый ключ для прикрепления своей подписи к документу. Для аутентификации документа получатель использует открытый ключ отправителя. Если есть совпадение, получатель может быть уверен в авторе. Не запутайтесь в назначении DSA: он предназначен не для защиты содержимого сообщения, а только для предоставления услуги по аутентификации автора документа. По этой причине DSA применяется не ко всем данным, а только к их части. В любом случае алгоритм не может

использоваться для шифрования текста такой же длины или длиннее, чем ключ. Из-за этого он обычно используется в контрольной сумме сообщения. Ключи генерируются с помощью слов dsa-make-key и dsa-generate-key.

Подпись создаётся с помощью dsa-make-signature и проверяется с помощью dsa-verify-signature, который возвращает простое логическое значение.

```
data: checksum/secure "message to which the signature will be applied"

my-key: dsa-make-key/generate 1024
dsa-generate-key my-key
signature: dsa-make-signature/sign my-key data
print mold signature
print dsa-verify-signature key data signature
```

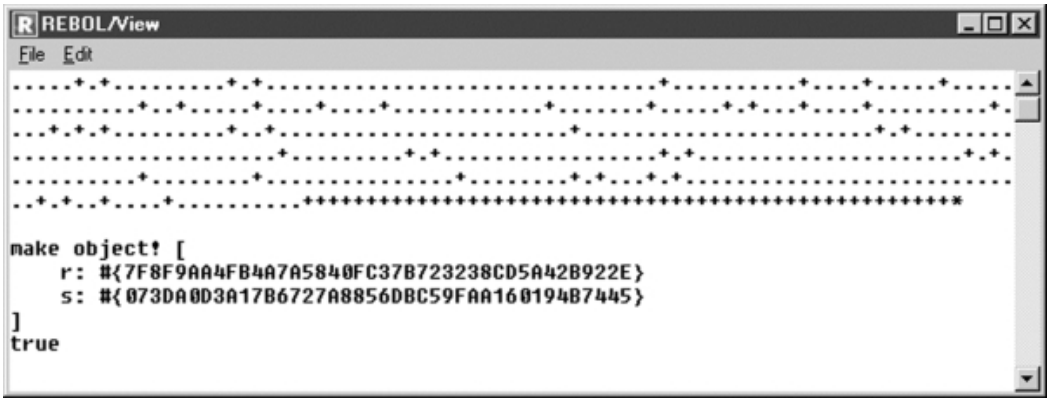

Рисунок 5-5. *Ключи и проверка подписи DSA.*

Как видите, в конце этого быстрого обзора технологий шифрования, доступных в Rebol, он содержит лучшие доступные инструменты для обеспечения безопасности ваших данных при их передаче по сетям.

9.4 Rebol/IOS

Rebol/IOS (Internet Operating System (Интернет-операционная система)) - это сервер приложений Rebol. Повторяя функции других версий языка (Core, View и Command), он позволяет создать сообщество пользователей, которые могут легко обмениваться документами и приложениями. Rebol/IOS - это чрезвычайно глубокий продукт, но при этом невероятно лёгкий. Это сублимация концепций Rebol: материализация живого, интерактивного и динамичного Интернета.

9.4.1 IOS: концепция, которой не хватает разоблачения

Rebol/IOS позволяет группе людей обмениваться информацией, манипулировать ею и делиться ею.

Используя транспортный протокол HTTP, его можно использовать как в локальных сетях, так и в Интернете. Пользователи подключаются к серверу через выделенный клиент под названием Link, который создаёт настоящую рабочую среду, которую можно описать как виртуальный офис. Идентифицированные своей учётной записью пользователя, они могут не только отправлять данные на сервер и получать данные с него, но и использовать приложения, которые он размещает.

Главная особенность IOS, помимо того факта, что вы можете полностью избежать громоздкого браузера, заключается в том, что он по своей сути использует концепцию синхронизации. Каждый раз, когда клиент подключается, его среда обновляется, чтобы отразить все изменения, сделанные на сервере. Поэтому, если приложение Rebol, которое называется реблетом, или даже сам клиент Rebol/Link изменяется, клиент автоматически извлекает и устанавливает новые версии.

Этот механизм зеркалирования и тот факт, что реблеты выполняются под Rebol/Link на клиенте, даёт пользователю возможность работать в интерактивном или автономном режиме. В последнем случае клиент синхронизирует любые изменения, внесённые пользователем, при следующем подключении к серверу. Мобильный пользователь может время от времени подключаться к своему серверу IOS и обновлять свои данные. В течение дня они могут работать в своём виртуальном офисе с помощью приложений Rebol. Ночью пользователь просто подключается к серверу IOS, который автоматически загружает его данные и при необходимости обновляет свою рабочую среду. Для программиста и пользователя все эти операции полностью прозрачны и, прежде всего, выполняются безопасно.

9.4.2 Ультра-безопасная система

Люди из Rebol действительно сосредоточились на безопасности информации, передаваемой между клиентами Link и сервером приложений. Не нужно ничего добавлять к продукту: IOS действительно умеет управлять своей безопасностью.

Первое, что нужно знать, это то, что клиент Link привязан к серверу, к которому он может подключиться. И клиент, и сервер включают сертификат, который блокирует попытки несанкционированного подключения. Клиент Link компании A никогда не сможет подключиться к серверу IOS компании B (если, конечно, обе стороны не захотят такие подключения).
Как только соединение установлено, клиент Link и его сервер совместно используют сеансовый ключ, обмен которого основан на модели RSA (Rivest, Shamir and Adelman). Это означает, что все обмены между машинами будут автоматически зашифрованы. Человек может попытаться подглядывать за вашей сетью, но ничего не узнает.
Если вы продвинете стандартную конфигурацию немного дальше, вы можете отфильтровать IP-адреса и TCP-порты, с которых машинам разрешено подключаться к серверу. Вы даже можете зашифровать данные, которые хранятся на клиенте и сервере. Rebol/IOS также использует различные методы для обеспечения целостности данных, такие как контрольные суммы SHA1.

На сервере пользователь имеет определённые права, определяемые его именем пользователя. Идентификация пользователя основана на классическом логине с паролем. У каждого пользователя могут быть разные права, и они могут настраивать свою рабочую среду. Вы можете скрыть документы и приложения от определённых пользователей: они никогда не узнают о существовании скрытых документов или приложений и не смогут узнать. Более того, все действия пользователя и сервера записываются в файл журнала, который позволяет отслеживать активность IOS и помогает решать возможные проблемы.

9.4.3 Сервер Rebol/Serve

Сердце IOS - это сервер REBOL/Serve. Этот продукт можно установить за пять минут и доступен для Windows и Linux.
ZIP-архив версии для Windows содержит пять каталогов. Сам сервер находится в каталоге rebol-serve и называется, если вы не могли догадаться, rebol-serve.exe. Это простой исполняемый файл размером около 680 КБ. Чтобы активировать его, просто введите команду rebol-serve.exe -wqs. Конечно, вы можете создать командный файл для его запуска или, в системах Unix, использовать командные файлы, предоставленные в каталоге сценариев. Просто введите команды оболочки server start или server stop, чтобы запустить или остановить сервер IOS.

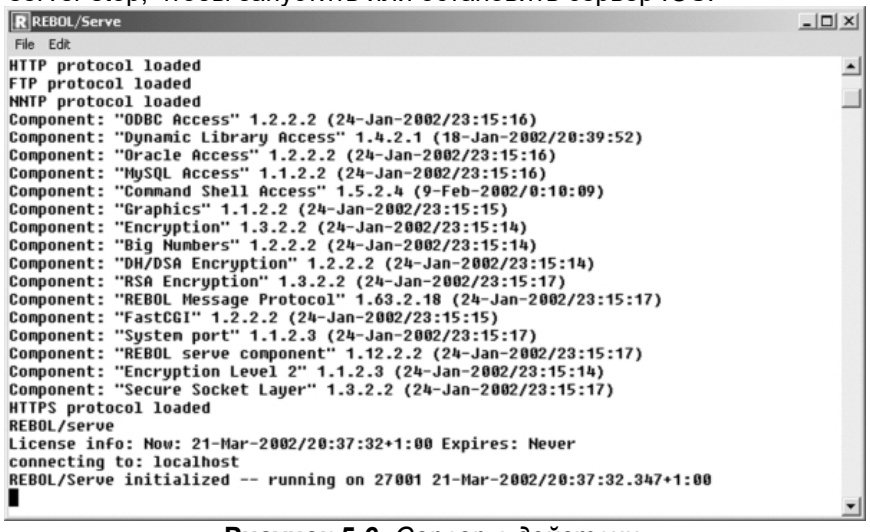

Рисунок 5-6. *Сервер в действии.*

После запуска сервера вы должны установить прокси. Это небольшое приложение размером около 35 КБ находится в каталоге прокси как в исходной, так и в скомпилированной версиях. Различные файлы предназначены для GNU Make и Microsoft Developer Studio, поэтому вы можете собрать эту программу для другой платформы.

Для чего нужен этот прокси? Фактически, для того, чтобы IOS могла работать через Интернет, вам нужен веб-сервер. Это может быть практически любой веб-сервер; единственное условие - поддержка постоянных HTTP-соединений. Будьте уверены, два лидера рынка (Apache и Microsoft IIS) делают это очень хорошо и идеально подходят для IOS. Цель прокси-программ - действовать как посредник между Link-клиентами, подключающимися через HTTP к веб-серверу, и IOS, использующим определённый TCP-порт. Прокси-сервер - это простой крошечный сценарий CGI, вы должны поместить его в виртуальный каталог cgi-bin вашего веб-сервера.

Очень интересно отметить, что HTTP-сервер и, как следствие, сценарий CGI не обязательно должны запускаться на том же компьютере, что и сервер IOS. Вы можете собрать сервер Linux/Apache для приёма соединений от клиентов и сервер Windows 2000, на котором размещён Rebol/IOS.

Архитектура очень гибкая и позволяет использовать лучшие продукты для удовлетворения ваших потребностей.

9.4.4 Клиент Rebol/Link

Когда сервер заработает, к нему будет подключаться только клиент Rebol/Link. Это лёгкое приложение размером около 500 КБ в версии для Windows. Конфигурация, необходимая для его нормальной работы, - это всего лишь микропроцессор с частотой 200 МГц, 64 Мбайта оперативной памяти и около 4 Мбайт дискового пространства. Совсем не вредоносное ПО! (Сообщалось, что Rebol/Link успешно работает под Wine в Linux). Даже с простым модемом Rebol/Link можно загрузить за несколько секунд.

Клиент Rebol/Link содержит собственную процедуру установки. Последний позволяет пользователю ввести имя своей учётной записи и пароль, а также информацию о сети, к которой он принадлежит (почтовый сервер и прокси).

После запуска Rebol/Link принимает вид рабочего стола, разделённого на три зоны:

- в верхней части экрана находится панель кнопок, которая позволяет получить доступ к часто используемым приложениям,
- в левом столбце перечислены папки для разных видов деятельности,
- основная область экрана используется для отображения иконок приложений, папок и файлов.

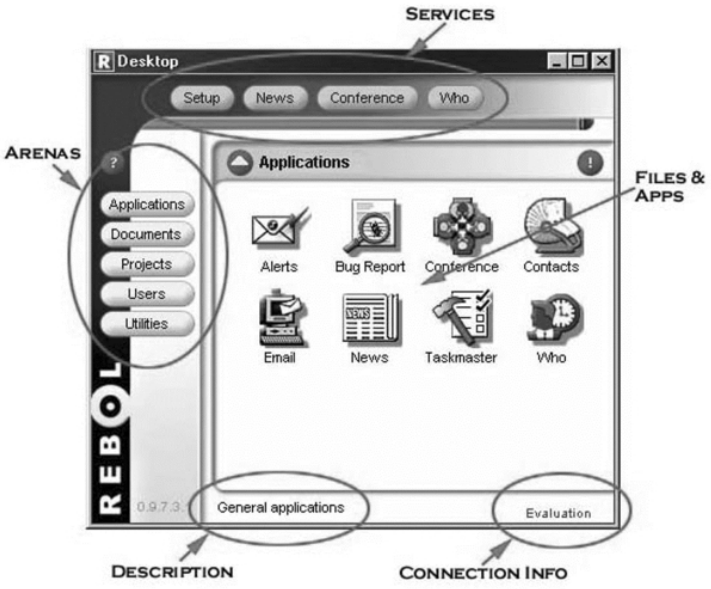

Рисунок 5-7. *Link клиент организации.*

Этот рабочий стол можно сильно персонализировать, а графику можно адаптировать к требованиям компании или организации. Вы можете изменить его цвет или добавить логотип. Для администратора эта работа заключается в создании скина, который будет автоматически загружен клиентам при следующей синхронизации.

Использование Rebol/Link очень интуитивно понятно, потому что полностью управляется мышью. Пользователь должен только щёлкнуть различные значки, представляющие доступные функции. Правая кнопка мыши позволяет изменять свойства документа, а также используется для публикации информации на сервере IOS. Эти документы могут быть файлами, возможно, сгруппированными в папки, гипертекстовыми ссылками на HTML-страницы или приложениями, написанными на Rebol, которые запускаются на клиенте. Если вы используете офисные документы (Word, Excel, Visio, PDF и т.д.), Rebol/Link будет использовать ассоциации файлов (.doc и т.д.) и запустит соответствующее приложение, чтобы открыть файл (при условии, что он доступен на машина пользователя).
Помимо обмена документами внутри рабочей группы, основная роль Rebol/Link, очевидно, состоит в том, чтобы запускать приложения, написанные на Rebol, они называются Reblets (реблеты).

9.4.5 Реблеты

Эти сверхлёгкие приложения являются сердцем стратегии Rebol, известной как X-Internet. Это не просто распространение HTML-страниц, а распространение подлинного программного обеспечения. Каждый клиент Reblet становится активной частью сети и может обмениваться информацией с другими членами сообщества. Приложения больше не работают только на сервере, а построены с использованием распределённой архитектуры. На самом деле существует три разных метода разделения ролей между клиентом и сервером:

- Реблет может быть запущен исключительно клиентом,
- Реблет может выполняться как на клиенте, так и на сервере: в этом случае реблет использует так называемую функцию post, которая находится на сервере в каталоге applications (приложения),
- Реблет может также вызвать выполнение внешних программ на сервер и используйте последующие результаты.

Эта интеграция клиента и сервера в процессе манипулирования и представления информации приводит к значительному снижению нагрузки на сеть и сервер по сравнению с современными традиционными решениями (скрипты CGI, JSP, PHP и т.д.).
Чтобы продемонстрировать возможности своего решения и позиционировать Rebol/IOS в области совместной работы, Rebol предоставляет дюжину реблетов, готовых к использованию и персонализации. Исходный код каждого из них доступен, и вы можете свободно адаптировать их под свои нужды.
Эти реблеты охватывают многие области, и они делают IOS настоящим концентратором в сети. Например, вместо обычно менее безопасного обмена сообщениями приложение Messenger позволяет двум пользователям участвовать в прямом или отложенном диалоге (все сообщения архивируются, и поэтому вы храните историю своих разговоров). С помощью конференции вы можете создавать темы для обсуждения, и несколько пользователей могут обмениваться мнениями одновременно. Для тех, кто любит ностальгию, Rebol/IOS включает небольшую переделку для отправки сообщений по классическому протоколу SMTP.

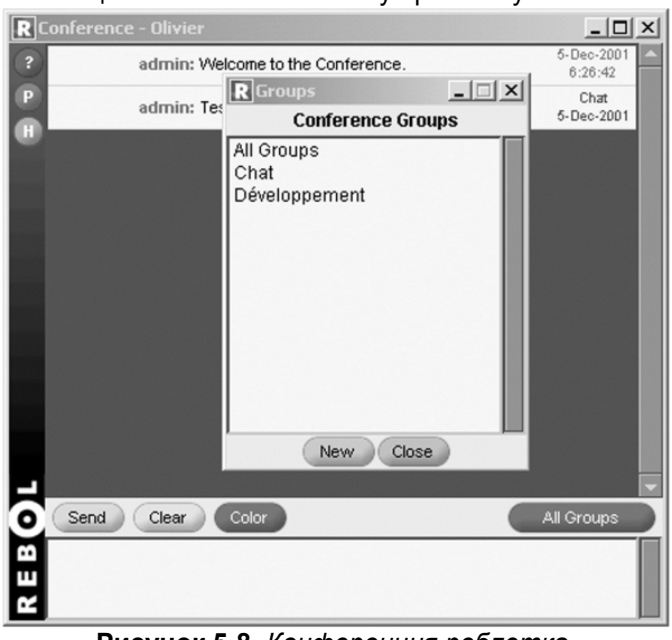

Рисунок 5-8. *Конференция реблетка.*

В лёгком офисном домене у вас есть общий менеджер контактов, в котором каждый может просматривать и поддерживать содержимое. Общий календарь (Calendar) и основной менеджер проекта (Taskmaster) облегчают координацию работы между разными людьми.

Еще одно очень практичное приложение - это Who reblet. Это показывает, кто подключен к серверу, их доступность и местоположение. Просто нажав на одну из нескольких видимых кнопок, вы можете сообщить членам своего сообщества, свободны ли вы, заняты, находитесь на собрании или просто ушли домой. Ваши коллеги даже могут узнать, где вы находитесь: кто показывает, подключены ли вы из офиса или дома.

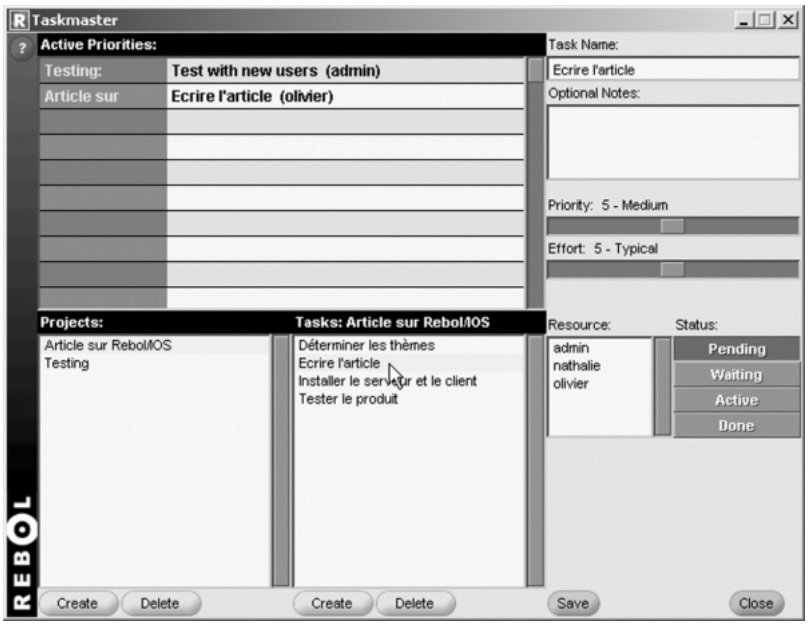

Рисунок 5-9. *Taskmaster - простой руководитель проекта.*

Есть множество реблетов, и Rebol Technologies объявляет о новых, когда они станут доступны; некоторые входят в комплект поставки сервера, некоторые продаются отдельно. Вы можете отслеживать продажи продуктов с помощью приложения "Продажи". Опрос предназначен для организации опросов, в которых могут участвовать пользователи. Вы можете создавать кривые и гистограммы из источников данных с помощью Plot. Presenter - это программа, позволяющая транслировать интерактивную презентацию по сети. Это всего лишь несколько примеров из списка приложений IOS.

В любом случае, если вы не найдёте то, что ищете в списке приложений, ничто не мешает вам изменять приложения или разрабатывать собственные реблеты, полностью адаптированные к вашим потребностям.

9.4.6 Инструменты администрирования

Реблеты настолько универсальны, что их даже используют для администрирования сервера. Вы просто входите в учётную запись "admin", чтобы получить доступ к функциям управления Rebol/IOS. Администратор может подключиться с любого компьютера с клиентом Link.

Среди множества доступных инструментов три основных - это, без сомнения, Alert для отправки сообщений всем пользователям, User-admin для управления учётными записями пользователей и Reg-edit для обновления наборов файлов (объяснено ниже).

Администратор может транслировать сообщение каждому пользователю, подключённому к серверу с помощью Alert. На рабочем столе клиента автоматически откроется окно для отображения предупреждающего сообщения. Эта функция очень полезна для предупреждения пользователей о важном событии, таком как предупреждение о том, что машина или служба будут недоступны.

С User-admin у вас есть отличный инструмент для создания и управления учётными записями пользователей. Для каждого из них необходимо ввести имя и адрес электронной почты. Вы также можете подключить пользователя к одной или нескольким группам и определить конкретные права

для каждого из девяти доступных основных критериев. Система очень гибкая и позволяет очень точно контролировать права каждого пользователя. Вы также можете импортировать пользователей из другой системы Rebol/IOS с помощью простого скрипта Rebol. Возможно даже автоматическое включение элементов из служб каталогов LDAP.

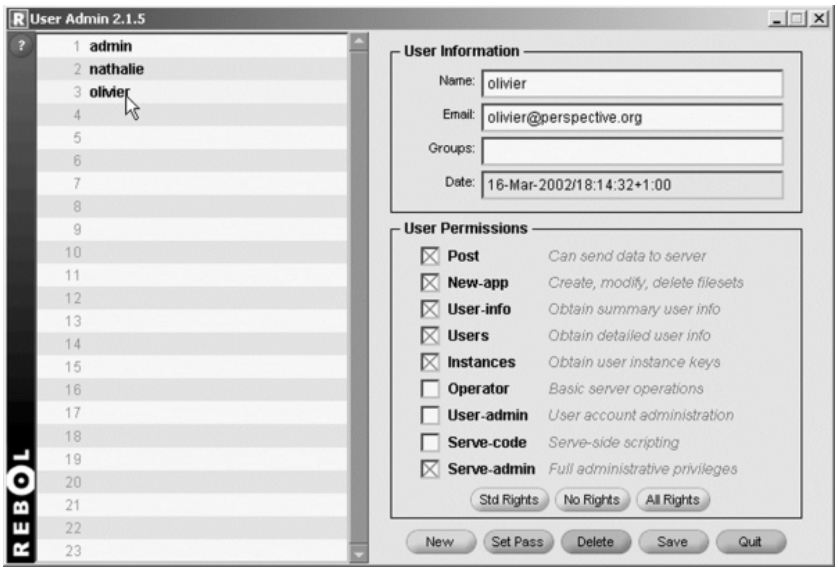

Рисунок 5-10. *Управление учётными записями пользователей.*

Reg-edit используется для администрирования наборов файлов Rebol/IOS, файловой системы, используемой сервером.

Каждый обрабатываемый файл, будь то приложение или данные, принадлежит набору файлов, который определяет характеристики файлов, права доступа, использование, в которое он может быть помещён, и много другой информации, включая значок, который будет использоваться на рабочем столе Link. Публикация реблета на сервере с помощью функции POST сводится к созданию нового набора файлов на сервере.

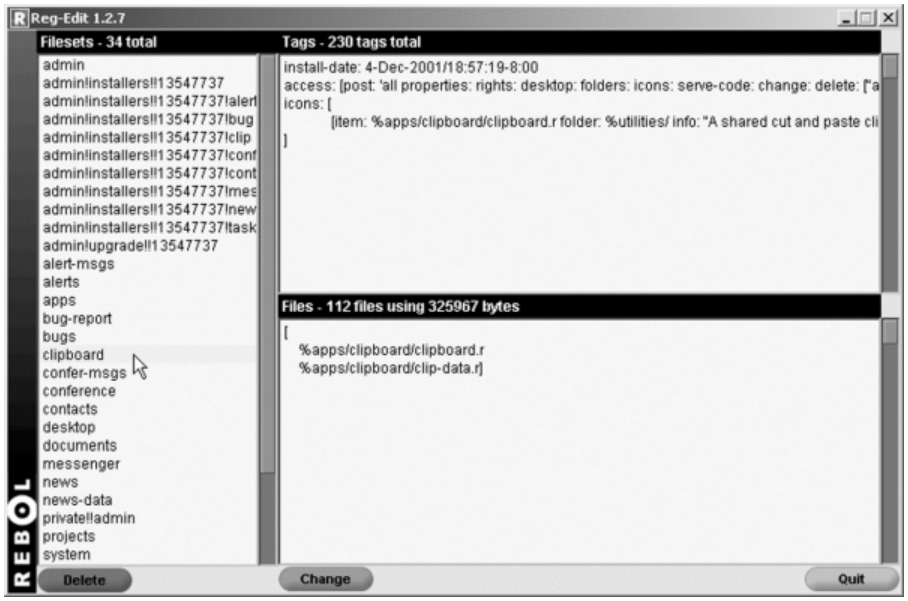

Рисунок 5-11. *Управление базой данных реестра.*

Rebol/IOS - потрясающий продукт, не имеющий аналогов. Просто используйте его в течение нескольких секунд, чтобы понять, что Карл Сассенрат создал платформу, революционизирующую использование Интернета. Rebol/IOS, способный интегрироваться с существующей средой и постепенно заменять её, может стать сердцем информационной системы. Поставка решений на базе сервера Rebol/IOS, специально настроенного с точки зрения внешнего вида и функций в соответствии с конкретными потребностями организации, несомненно, является будущим рынком для разработчиков.

9.5 Управление проектами Rebol

Язык Rebol настолько компактен, что вы можете писать свои программы с минимальным количеством строк кода, как правило, в одном скрипте. Но для более амбициозных проектов обслуживание часто требует разделения проекта на несколько файлов. Затем вы должны использовать менеджер исходного кода Prebol.

9.5.1 Препроцессор Prebol

Утилита Prebol - это препроцессор для кода Rebol. Он позволяет автоматически создавать сценарий Rebol из набора ресурсов, таких как файлы, содержащие код Rebol, данные или мультимедийные файлы (изображения, звуки и т.д.).

Эта генерация может быть статической или динамической в зависимости от набора параметров, определяемых программистом, среды разработки или даже целевой среды выполнения.

Prebol также позволяет оптимизировать код Rebol, поскольку сводит к минимуму размер распространяемого скрипта. Для этого утилита удаляет ненужные символы, такие как метаданные во всех включённых файлах, комментарии и некоторые символы конца строки. Цель этих операций - предоставить сжатый файл, который можно легко распространять через Интернет. Prebol - незаменимый инструмент для разработчиков Rebol.

9.5.2 Установка и использование

Перед установкой Prebol на вашу машину вы должны сначала получить его. У вас есть выбор между платной и бесплатной версиями. Владельцы коммерческого Rebol/SDK (Software Development Kit) имеют эту утилиту в двух формах: двоичный исполняемый файл в папке bin и скрипт в source папке. Чтобы вызвать его из командной строки, вы должны использовать команду prerebol, за которой следует имя источника скрипта и имя создаваемого файла.

```
prerebol project.r program.r
```

В качестве альтернативы его запуску из оболочки вы также можете использовать сценарий prebol.r с теми же аргументами. Его также можно вызвать непосредственно из консоли Rebol, используя слово do с уточнением /args, тогда синтаксис следующий:

```
do/args %prebol.r [ %project.r %program.r ]
```

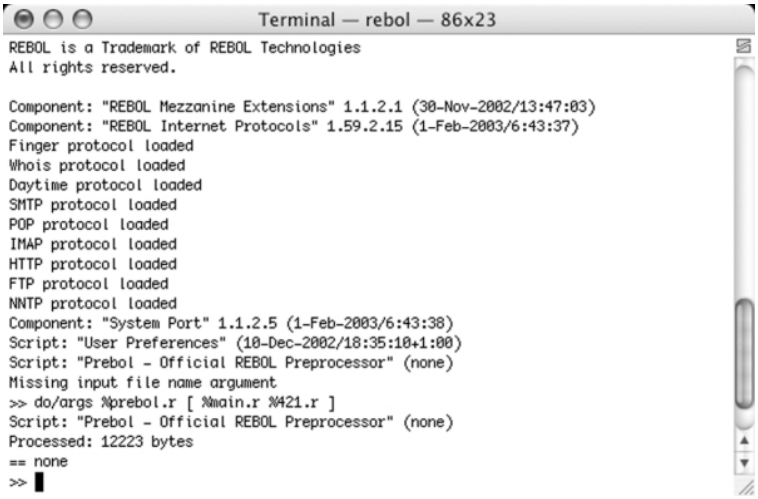

Рисунок 5-12. *Использование препроцессора Ребола.*

SDK поставляется с рядом различных программ для инкапсуляции скрипта. (То есть для создания исполняемого двоичного файла из скрипта). Они автоматически вызывают prerebol всякий раз, когда скрипт преобразуется в двоичный файл. Использование команд prerebol не обязательно, когда вы тестируете свои проекты в различных средах Rebol (Rebol/base, Rebol/face, Rebol/cmd и т.д.).

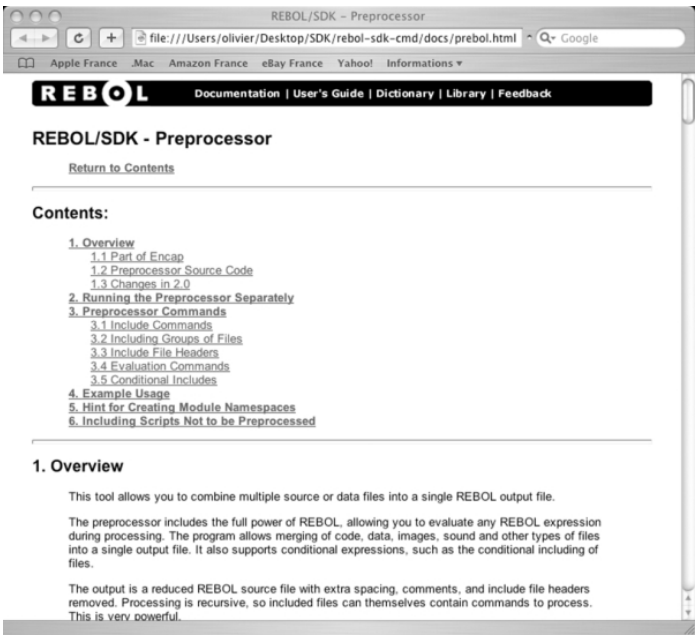

Рисунок 5-13. *Документация препроцессора.*

Если вы не работаете с Rebol/SDK, но используете бесплатные версии Rebol (Core и View), вы можете просто загрузить prebol.r с www.reboltech.com/library/scripts. Тогда вы получите скрипт Rebol, выполняющий те же функции, что и коммерческая версия. То же самое и с результатами. Обратите внимание, что коммерческая версия имеет номер версии 2.0, она легче и быстрее, чем бесплатная версия 1.0. На самом деле различия незаметны для видимого глаза, и все последующие примеры основаны на бесплатной версии препроцессора.

9.5.3 Включаемые файлы

Чтобы проиллюстрировать использование препроцессора Rebol, вы собираетесь разработать сценарий, очень свободно вдохновлённый игрой 421. Он состоит в том, чтобы бросить три кубика, чтобы получить значения 4, 2, 1. Эта программа, разработанная с Rebol/View , состоит из двух скриптов: основной программы с именем main.r и библиотеки с именем launch.r. Приложение также состоит из графических файлов в формате PNG и двух текстовых файлов для облегчения локализации продукта на английском или французском языках.
Различные функции программы находятся в файле main.r. Сюда входят все ресурсы, необходимые для создания единого рабочего сценария под названием 421.r, который можно легко распространять через Интернет.

Рисунок 5-14. *Игра 421.*

Для включения данных в Prebol есть четыре команды: #include, #include-binary, #include-files и #include-string. Эти инструкции принимают параметр типа файла (file!) или, в случае include-files, каталог и блок, содержащие имена файлов. Также можно использовать выражения Rebol, заключённые в квадратные скобки. В этом случае результат оценки используется в качестве параметра.

Чтобы включить launch.r, вы просто вставляете команду #include %launch.r в свой сценарий. Изображения вставляются с помощью команд #include-binary или #include-files. Очевидно, вы можете использовать другие типы данных, например звуковые файлы. Фактически, эти команды просто вставляют двоичные данные независимо от формата. Для проекта 421 изображениями являются название приложения (titre.png) и шесть изображений для разных граней игральной кости. Эти файлы хранятся в подкаталоге pics в каталоге, в котором находится проект. Заголовок импортируется с помощью оператора titre.png: load #include-binary (join %pics/ %titre.png). Здесь мы использовали выражение, которое будет вычислено Rebol для построения пути к файлу. Когда используется препроцессор, содержимое двоичных файлов преобразуется в базу 64 и вставляется в сгенерированный файл. При оценке скрипта данные будут загружены в слово titre.png.

Для разных граней игральной кости вы собираетесь использовать директиву #include-files. Это позволяет вам включить несколько файлов из каталога за одну операцию.

```
#include-files %pics [
    cube1.png
    cube2.png
    cube3.png
    cube4.png
    cube5.png
    cube6.png
]
```

Для каждого файла в списке эта команда создаёт пару значений, состоящую из имени файла (идентификатора ресурса) и содержимого файла. Для автоматического создания слов, содержащих данные, вы можете использовать инструкцию foreach, которая позволяет вам перемещаться по ряду и присваивать все значения словам.

Мы должны использовать результат препроцессора для восстановления каждого идентификатора и его соответствующего значения. Ресурсы легко добавляются в словарь Rebol с помощью set.

```
foreach [ word data ] #include-files %pics [
    cube1.png
    cube2.png
    cube3.png
    cube4.png
    cube5.png
    cube6.png
] [ set word load data ]
```

9.5.4 Оценка и условия

Чтобы локализовать приложение, вам необходимо условно включить файл с заголовками кнопок. Если вы создаёте французскую версию, это будет файл fr.txt. Если продукт будет на английском языке, это будет en.txt. Чтобы указать язык, который будет использоваться препроцессором, вы должны использовать переменную, которая будет оцениваться по мере создания продукта. Для этого у вас есть команда #do.

Эта инструкция очень мощная, поскольку позволяет выполнять код Rebol из Prebol. Он позволяет устанавливать параметры, удалять файлы, отправлять данные на FTP или HTTP-сервер, создавать журнал файлов и т.д.

В своем проекте вы собираетесь использовать эту команду, чтобы просто инициализировать переменную lang значением "fr" или "en".

```
#do [ lang: "fr" ]
```

Для условий у вас есть команды #if и #either. Они работают по тем же принципам, что и слова if и either в Rebol. Блок оценивается, и если результат true (истина), выполняется следующий блок. Инструкция #either добавляет третий блок, который оценивается только в том случае, если возвращаемое значение false (ложно). В своём проекте вы собираетесь установить для слова launch-txt, которое является заголовком кнопки, строку символов, содержащуюся в файле fr.txt или en.txt.

```
launch-txt: #either [lang = "fr" ] [
    #include-string %fr.txt
] [ #include-string %en.txt ]
```

Осталось только сгенерировать свой скрипт с помощью Prebol. Вы получите файл размером около 16 Кбайт, содержащий код вашего приложения и ресурсы, необходимые для его работы. Этот единственный файл теперь можно транслировать через Интернет.

9.6 Резюме

Rebol - это универсальный язык, идеально подходящий для написания сетевых приложений или офисного программного обеспечения. Rebol/Command посвящён электронному бизнесу. Rebol/IOS - это платформа для групповой работы, которая поражает простотой использования и возможностью расширения. Также можно использовать бесплатные версии Rebol для разработки веб-приложений.

10. Ребол для гиков

Rebol - это увлекательный язык, который позволяет с захватывающей дух лёгкостью использовать оригинальные и эффективные технологии. Развёртывание виртуальных рабочих столов через Интернет и программирование приложений Unix и Windows - обычное дело для программистов Rebol.

10.1 Rebol и виртуальные рабочие столы

Вы можете создавать виртуальные офисы для распространения информации или приложений через Интернет или интранет (локальную сеть) вашей компании с помощью Rebol/View. Это одна из главных революций, представленных Rebol.

10.1.1 Рабочая среда

С Rebol/View каждый пользователь может получить доступ к точке подключения, предоставляющей доступ к частным или общедоступным службам. Это программы Rebol, загружаемые по сети с HTTP-сервера: они известны как Reblets.

Каждый человек в компании или организации, будь то статичный или кочевой, может иметь ряд услуг без необходимости установки приложений на клиентском компьютере. Виртуальный офис может не только предоставлять доступ к файлам данных (текст, XML-документы и т.д.), но также и к классическим веб-сервисам (локальным или удалённым HTML-документам, приложениям интрасети и т.д.). Опять же, инструмент новаторский, но достаточно гибкий, чтобы вписаться в существующую архитектуру. Нет необходимости менять все сразу, различные службы вашей интрасети можно постепенно переносить в виртуальный офис Rebol/View, который затем постепенно становится вашей рабочей средой.

10.1.2 Организация рабочего стола Rebol/View

В верхней части экрана Rebol/View вы найдёте ярлыки, такие как "User" (Пользователь) для установки конфигурации для пользователя или "Goto" (Перейти) для доступа к виртуальному офису через его URL-адрес. Эти различные заголовки можно настроить с помощью файла services.r в каталоге рабочего стола дерева файлов Rebol/View. Внизу окна вы найдёте область, предназначенную для отображения информации о состоянии. Слева консоль Rebol/View доступна простым щелчком кнопки мыши. Также есть значок с настройками по умолчанию для каталога Rebol.com, который открывает двери World Wide Reb. Вы можете добавить дополнительные ярлыки на эту панель окна, отредактировав файл bookmarks.r, который также находится в папке desktop. В качестве упражнения отредактируйте этот файл с помощью простого текстового редактора и дайте ему следующее содержимое:

```
REBOL [Title: "Bookmarks" Type: 'index]
folder "REBOL.com" http://www.rebol.com/index.r
folder "Local site" %local/index.r
folder "rebsite" http://172.29.143.1/rebsite/index.r
file "Console" console icon console
```

Теперь при запуске Rebol/View появятся две новые директории. Папка "Локальный сайт" позволяет вам получить доступ к файлам в локальном каталоге Rebol/View. Вторая - это ссылка на сайт, который вы собираетесь разместить на http-сервере с IP-адресом 172.29.143.1.

В правом нижнем углу окна вы найдёте текущий режим работы Rebol/View. Если это "локальный" режим, вы нажимаете на него, чтобы перейти в подключенный режим. В противном случае ваш рабочий стол не сможет связаться с сервером.

10.1.3 Индекс файла

Вы просто храните файлы, из которых состоит ваш виртуальный офис, на HTTP-сервере. Файловая структура определяется файлами с именем index.r. В каждом подкаталоге есть файл index.r, который определяет содержимое этого каталога. Эти файлы должны включать в заголовок Rebol, что они относятся к типу index. В качестве упражнения вы можете создать файл index.r с указанием имени вашего рабочего стола и доступных ссылок. Приветственное текстовое сообщение содержится в файле с именем welcome.txt. Одна запись загружает реблет, другая - ссылку на главную страницу веб-сайта. В последнем случае Rebol/View отображает HTML-документ с помощью браузера по умолчанию. Изображение в формате GIF используется для "украшения офиса", но возможно и множество других эффектов.

```
REBOL [type: 'index]
title "My rebsite"
backdrop %fond.gif
file "Welcome" %welcome.txt
file "A Reblet" %reblet.r info "an example Reblet"
link "My Website" http://www.mywebsite.org
```

Реблет - это крошечный скрипт, который отображает текст в окне:

```
REBOL []
view/title layout [ title "A Reblet !" ] "A Reblet"
```

Используя виртуальный офис, вы действительно можете улучшить свою интрасеть и превратить её в интеллектуальный интерактивный инструмент.

10.2 Программирование Unix с Rebol

Rebol присутствует на многих платформах, большинство из которых являются членами великого семейства Unix.

Интерпретаторы Core, View и Command доступны для Linux, FreeBSD, OpenBSD, Solaris и Mac OS X. Эти системы имеют множество функций, которые Rebol вполне может использовать для создания высокопроизводительных приложений.

Этот раздел посвящён программированию под Unix с использованием Rebol. Мы собираемся узнать, как эффективно отображать данные в консоли, управлять клавиатурой, взаимодействовать с приложением Rebol с оболочкой, получать и определять права доступа к файлам и общаться через каналы и сокеты.

10.2.1 Отображение данных в консоли

В мире Unix пользователю доступны три типа интерфейса. В X-Windows приложения могут использовать современные графические компоненты, такие как окна и кнопки, с помощью мыши. Другие программы, такие как webmin, запускаются в веб-браузере; их интерфейсы построены на HTML. С Rebol легко разрабатывать оба этих типа программного обеспечения. Оценщик View имеет мощный диалект, называемый VID, который позволяет создавать расширенные графические приложения с минимальным кодом. Для веб-приложений Rebol поддерживает стандартные CGI, FastCGI и с лёгкостью обрабатывает теги и настройки передачи. Эти два аспекта разработки с Rebol уже были рассмотрены в этой книге. Они не связаны исключительно с платформой Unix и по этой причине не рассматриваются в этой главе.

Остаётся третий тип пользовательского интерфейса, старый добрый, надёжный текстовый режим с простой консолью или простым клиентом telnet. Можно подумать, что этот тип программного обеспечения полностью превзойдён, но в Unix это ещё не так. Многие утилиты и даже некоторые тяжёлые приложения для управления по-прежнему используют символьные интерфейсы. Они обеспечивают множество преимуществ, таких как скорость отображения, меньшая стоимость и низкие эксплуатационные расходы терминала, низкое потребление полосы пропускания сети и экономичное использование вычислительной мощности сервера.

Для отображения символьных строк на экране Rebol предоставляет слова print и prin. Единственное различие между ними состоит в том, что print добавляет новую строку (Enter) после отображения данных. Эти два слова принимают один аргумент, который может быть разных типов. Если аргумент является списком, его содержимое автоматически оценивается перед отображением.

Rebol также включает в себя различные управляющие символы, такие как #"^/'" для вставки возврата каретки или #"^(tab)" для символа табуляции. Также можно изменить ширину табуляции с помощью свойства system/console/tab-size.

В следующем примере имя и фамилия человека отображаются в двух строках с использованием отступа (табуляции) из 8 символов:

```
system/console/tab-size: 8
last-name: "Bridge"
first-name: "Peter"
print [ "Name:^(tab)" last-name "^/First Name:^(tab)" first-name ]
```

К счастью для нас, возможности дисплея Rebol не ограничиваются только этим. Чтобы украсить представление ваших приложений, Rebol позволяет отображать последовательности символов, которые будут интерпретироваться терминалом. Эти команды могут очищать экран, перемещать курсор, изменять внешний вид текста и даже использовать цвет. Все эти последовательности символов начинаются с кода ASCII 27, за которым следует символ "[" или в Rebol "^(1B)[".

Зная это, мы можем очистить экран консоли, поместить курсор в десятый столбец четвёртой строки и отобразить короткое сообщение:

```
ESC: "^(1B)["
prin join ESC "J"
prin join ESC [ "4;10H"]
prin "Hello !"
```

Используя последовательность "7n", вы также можете получить количество строк и столбцов в консоли. Это включает создание порта с использованием встроенного ‘console протокола.

После отправки последовательности чтение третьего элемента этого порта возвращает результат, отформатированный с количеством строк, отделённых от количества столбцов знаком ";" (точка с запятой), и строка заканчивается заглавной буквой "R".

```
cons: open/no-wait/binary [ scheme: 'console ]
print "^(1B)[7n"
pos: parse next next to-string copy cons ";R"
close cons
print make pair! reduce [
    (to-integer second pos) (to-integer first pos)
]
```

Вы можете легко создать расширенный интерфейс, используя эти управляющие последовательности. Вы также можете изменить внешний вид текста и использовать жирные, подчёркнутые, курсивные и даже мигающие символы.

Чтобы создать меню или поля ввода, можно поменять местами цвета текста и фона, используя код "7M". В следующем примере показаны различные возможные эффекты. Если использование этих последовательностей не вызывает технических трудностей, имейте в виду, что результаты могут отличаться в зависимости от возможностей используемого терминала.

```
ESC: "^(1B)["

styles: [
[ "Bold" "1m" ]
[ "Normal" "2m" ]
[ "Italic" "3m" ]
[ "Underlined" "4m" ]
[ "Flashing" "5m" ]
[ "Inverted" "7m" ]
]

prin join ESC "J"   ; clear the screen

foreach ele styles [
    print [ (join ESC second ele) (first ele) (join ESC "0m") ]
]
```

Рисунок 6-1. *Различные эффекты последовательности управления.*

Чтобы ещё больше улучшить внешний вид ваших приложений, вы можете выбрать один из восьми предопределённых цветов (black (чёрный), red (красный), green (зелёный), yellow (жёлтый), blue (синий), magenta (пурпурный), cyan (голубой) и white (белый)).

Для каждого из них вы должны использовать определённую последовательность символов, которая различается в зависимости от того, хотите ли вы изменить цвет текста или его фона. В следующем примере показаны различные возможности, которых можно достичь, отображая разные доступные цвета.

```
ESC: "^(1B)["

text-colour: [
    [ "Black" "30m" ] [ "Red" "31m" ] [ "Green" "32m" ]
    [ "Yellow" "33m" ] [ "Blue" "34m" ] [ "Magenta" "35m" ]
    [ "Cyan" "36m" ] [ "White" "37m" ]
]

background-colour: [
    [ "Black" "40m" ] [ "Red" "41m" ] [ "Green" "42m" ]
    [ "Yellow" "43m" ] [ "Blue" "44m" ] [ "Magenta" "45m" ]
    [ "Cyan" "46m" ] [ "White" "47m" ]
]

prin join ESC "J"   ; Clear the screen

foreach bc background-colour [
    prin join ESC (second bc)
    foreach tc text-colour [
        prin join ESC [ (second tc) (first tc) ]
    ]
    prin #"^/"  ; carriage return
]
```

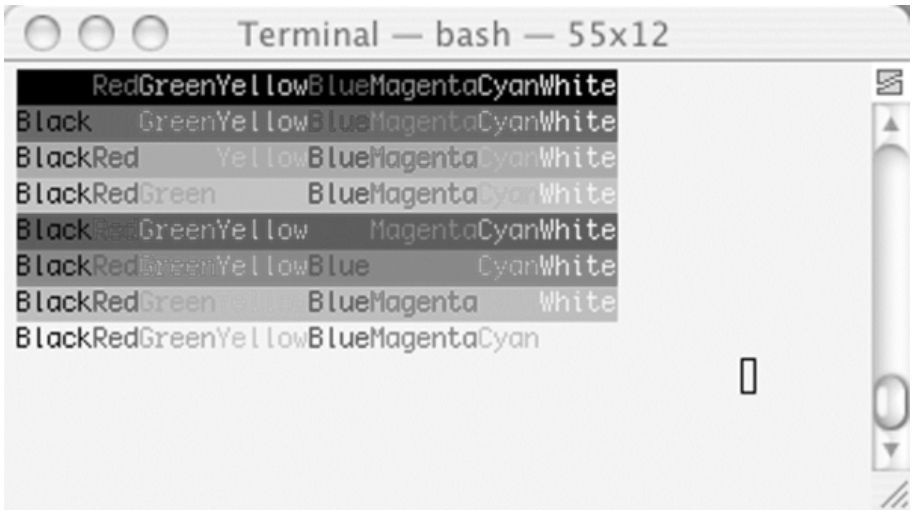

Рисунок 6-2. *Цвет текста и фона можно изменить.*

Если экраны вашего приложения сложны и требуют отображения множества операций, необходимо их оптимизировать.

С точки зрения скорости самый лёгкий выигрыш достигается за счёт отказа от использования слов print и prin для каждой управляющей последовательности, отправляемой на экран.

Идея состоит в том, чтобы построить экран в буфере и отображать его содержимое с помощью одного вызова слова print.

Следующий пример иллюстрирует этот метод, демонстрируя улучшенную версию предыдущего примера:

```
buffer: copy ""
insert tail buffer join ESC "J"   ; clear the screen
foreach bc background-colour [
    insert tail buffer join ESC (second bc)
    foreach tc text-colour [
        insert tail buffer join ESC [ (second tc) (first tc) ]
    ]
    insert tail buffer #"^/"     ; carriage return
]
prin buffer
```

Другой очень логичный и чрезвычайно эффективный подход - обновлять только те части экрана, содержимое которых было изменено. Избегая автоматического повторного отображения всего экрана при каждом изменении, ваше приложение будет более отзывчивым и может даже создавать меньше сетевого трафика.

10.2.2 Управление клавиатурой

Теперь мы знаем, как отображать информацию. Следующий шаг - заставить наши приложения реагировать на действия клавиатуры. Для ввода с клавиатуры, законченного клавишей "Enter", Rebol обеспечивает словами input и ask, которые ждут ввода символов со стандартного периферийного устройства ввода (stdin).

У этих двух слов есть уточнение /hide (скрытие) для сбора конфиденциальной информации (вместо нажатого символа отображается звёздочка). ask принимает аргумент, который представляет собой вопрос, который нужно задать пользователю.

```
print "What is your last name:"
last-name: input
first-name: ask "What is your first name:"
mdp: ask/hide "Your password? "
```

Слово confirm (подтверждение) просит пользователя подтвердить действие. По умолчанию ответ должен быть "Y" или "N", но уточнение /with позволяет указать два символа для обозначения принятия или отказа. Возвращаемый результат - логическое значение.

```
confirm/with "(A)ccept or (D)ecline " [ "a" "d" ]
```

В дополнение к этим простым функциям интерпретатор Rebol предлагает другие функции для управления клавиатурой на более низком уровне. Например, можно отключить клавишу ESC, чтобы запретить пользователю останавливать выполнение приложения. Для этого все, что вам нужно сделать, это установить для свойства break объекта system/console значение false. Вы также можете управлять основными клавишами управления на клавиатуре. Однако будьте осторожны, все клавиатуры не идентичны (попробуйте найти "яблочную" клавишу Mac на ПК!). По этой причине интерпретатор Rebol определяет группу ключей, общих для различных операционных систем, в которых он работает. Помимо отображаемых символов, консоль распознает клавиши ввода, табуляции, удаления, вверх, вниз, вправо, влево и вставки.

Для более точного управления клавиатурой вы должны использовать протокол консоли для получения кода от реальных клавиш при нажатии. Как только порт открыт, приложение может войти в бесконечный цикл и ждать нажатия клавиши с помощью ввода input? Что возвращает

логическое значение true, если было зафиксировано новое нажатие клавиши. При чтении порта извлекается строка, длина которой зависит от того, какая клавиша была нажата. Если последовательность не начинается с управляющего символа ^(1B), это одиночный символ или специальная клавиша, такая как tab, enter или del. Если последовательность начинается с ^(1B), а вторым символом также является ^(1B), сценарий обнаружил использование клавиши ESC. Третий и последний случай - это использование клавиш Insert, Up, End и клавиш со стрелками. Следующий сценарий иллюстрирует этот тип управления клавиатурой и может быть легко адаптирован к любому типу проекта.

```
system/console/break: false
cons: open/no-wait/binary [ scheme: 'console ]
forever [
    until [ input? ]
    touch: copy cons
    either (first touch) <> #"^(1B)" [
        switch/default (first touch) [
            8 [ print "DEL" ]
            9 [ print "TAB" ]
            13 [ print "ENTER" ]
            127 [ print "DEL" ]
        ] [ print to-char first touch ]
    ] [
        either (second touch) = 27 [
            print "ESC"
        ] [
            switch (third touch) [
                50 [ print "INSERT" ]
                65 [ print "UP" ]
                66 [ print "DOWN" ]
                67 [ print "RIGHT" ]
                68 [ print "LEFT" ]
                101 [ print "END" ]
            ]
        ]
    ]
]
close cons
```

Rebol гарантирует, что ваш код совместим на разных платформах, поддерживая ограниченное количество нажатий клавиш. Такой выбор гарантирует, что ваши приложения будут безупречно работать в многочисленных системах, на которых работает Rebol.

10.2.3 Интеграция со средой Unix

Приложения Rebol идеально вписываются в среду Unix, и во время их выполнения ничто не отличает их от приложений, написанных на самых популярных языках Unix, таких как C или Perl. Чтобы сценарий Rebol распознавался как исполняемая программа, вы должны вставить shebang в качестве первой строки в его файл. (Это директива "#!", За которой следует путь доступа интерпретатора Rebol и аргументы, которые вы хотите передать Rebol). В Rebol обычно передаются аргументы -q и -s, чтобы Rebol не отображал информацию о запуске и отключал встроенный диспетчер безопасности. Вы также должны установить права выполнения вашего скрипта с помощью классической команды chmod + x.

Приложения Unix часто получают аргументы, передаваемые из командной строки. В Rebol они хранятся в объекте параметров системного объекта system/options. Свойство args содержит none, если аргументы командной строки не указаны. В другом случае он содержит список из n элементов. Чтобы найти количество параметров, вы просто используете слово length?, которое возвращает длину значения.

В следующем примере проверяется возможное наличие аргументов, переданных в сценарий, и, если они есть, отображается количество аргументов и их соответствующие значения.

```
if not none? system/options/args [
    print [
        "Number of arguments:"
        (length? system/options/args)
    ]
foreach arg system/options/args [ print arg ] ]
```

Рисунок 6-3. *Аргументы передаются из командной строки.*

С помощью объекта system/options вы можете получить точную информацию о среде, в которой работает ваше приложение. Свойство path возвращает текущий каталог. С помощью home вы можете определить путь доступа пользователя к каталогу. Наконец, свойство загрузки содержит расположение интерпретатора Rebol в файловой системе.

Доступ к системным переменным среды возможен с помощью слова get-env. Он принимает единственный параметр, который является именем желаемой переменной среды. Таким образом, синтаксис get-env "SHELL" возвращает путь к оболочке. Поскольку вы работаете в Unix, не забывайте, что вы должны строго придерживаться правильных символов верхнего и нижнего регистра в именах переменных среды. Если get-env не может предоставить результат, он возвращает значение none.

Перенаправление стандартного устройства вывода (stdout) в файл также возможно с помощью слова echo. Это отправляет символы, отображаемые на экране, в файл, имя которого передаётся в качестве параметра. В Rebol имена файлов и путей к файлам должны начинаться с символа "%".

Итак, если вы хотите сохранить изображение экрана в файле с именем test.txt в каталоге /var/tmp, всё, что вам нужно сделать, это вставить команду echo %/var/tmp/test.txt в своё приложение. Отныне все данные, отображаемые на экране, будут сохраняться в файле test.txt. Чтобы отключить это перенаправление, вы используете синтаксис echo none.

10.2.4 Файлы и права доступа

Управление файлами - одно из основных направлений деятельности программистов Unix.

Используя слово info?, вы можете узнать дату последнего изменения файла, его размер в байтах и то, является ли он файлом или каталогом. Это слово возвращает простой объект, состоящий из трёх свойств: size (размер), date (дата) и type (тип).

Чтобы получить эту информацию о файле test.txt, вы просто используете print mold info? %test.txt.

Доступ к различным элементам файлового дерева Unix регулируется с помощью прав доступа, которые можно определить с помощью слова get-mode. Программисты Rebol имеют доступ к этому через порт, указывающий на изучаемый ресурс.

В следующем примере извлекается имя владельца файла test.txt с использованием ‘owner-name в качестве аргумента.

```
p: open %test.txt
print get-modes p 'owner-name
close p
```

Различные атрибуты можно найти, используя значение 'file-mode в качестве аргумента для get-mode. Полученный список является чрезвычайно полным, поскольку у вас есть права на файл для владельца, его группы и других пользователей. Информация также содержит имя владельца, его группу, UID (идентификатор пользователя), GID (идентификатор группы) и даже полный путь доступа к файлу. Следующий сценарий отображает все свойства файла test.txt.

Различные атрибуты можно найти, используя значение 'file-mode в качестве аргумента для get-mode. Полученный список является чрезвычайно полным, поскольку у вас есть права на файл для владельца, его группы и других пользователей. Информация также содержит имя владельца, его группу, UID (идентификатор пользователя), GID (идентификатор группы) и даже полный путь доступа к файлу. Следующий сценарий отображает все свойства файла test.txt.

```
p: open %test.txt
d: get-modes p 'file-modes

forall d [
    print [ (first d) " = " get-modes p (first d) ] ]

close p
```

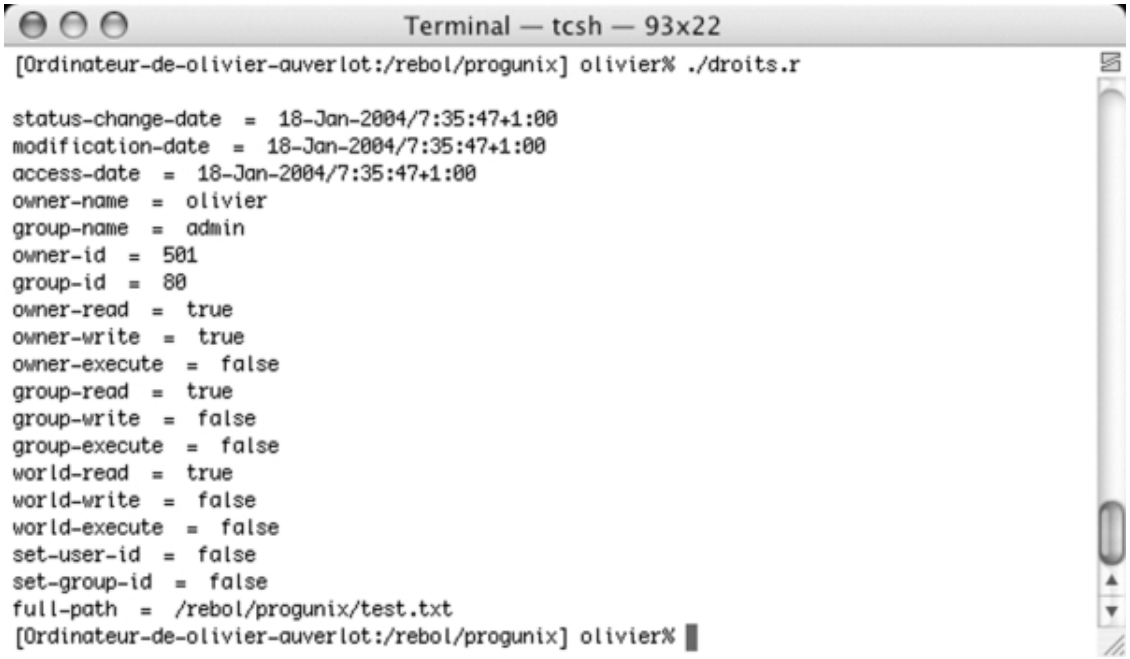

Рисунок 6-4. *Rebol позволяет просматривать и изменять права доступа.*

С помощью set-modes вы можете изменять права файла. В этих словах используются два параметра: порт, указывающий на файл, которым нужно управлять, и блок, содержащий атрибуты, которые необходимо изменить или добавить.

В следующем примере изменяются права доступа к файлу test.txt, предоставляя права записи членам группы владельца и другим пользователям.

```
p: open %test.txt
set-modes p [
    group-write: true
    world-write: true
]
close p
```

Благодаря использованию порта для обработки прав доступа к файлам Rebol предоставляет интерфейс с высоким уровнем абстракции по сравнению с операционной системой и аппаратной платформой.

Эта архитектура никоим образом не наказывает программиста, а оставляет ему или ей полный контроль над операциями и позволяет им легко получить доступ к мощной системе прав доступа в Unix.

Здесь Rebol остается верным себе, демонстрируя, что инструмент может оставаться простым в использовании.

10.2.5 Доступ к оболочке

Если вы хотите запускать внешние приложения, такие как команды Unix, вы должны использовать оболочку операционной системы. Эта функция всегда была доступна в коммерческих версиях rebol, но также была включена в последние бесплатные версии. В Rebol это слово call, которое позволяет вести диалог с оболочкой. Он принимает единственный аргумент, который представляет собой строку пути к запускаемым исполняемым файлам. Параметр напрямую передаётся в оболочку, синтаксис этого пути должен соответствовать стандартам платформы хоста, а не стандартам интерпретатора Rebol. Например, вызов команды "ls -l" отображает список файлов в текущем каталоге. Аналогичным образом, чтобы вызвать исполняемый тест, хранящийся в /usr/local/bin, синтаксис выглядит так: "/usr/local/bin/test".

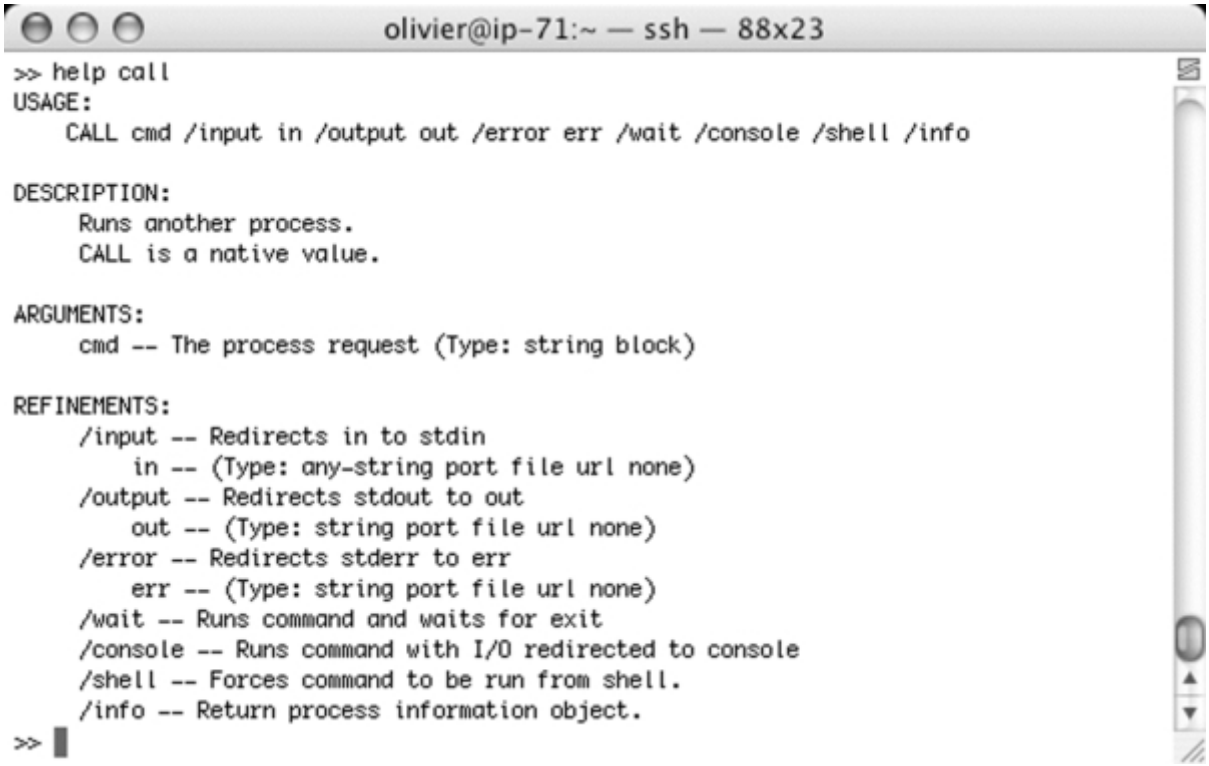

Рисунок 6-5. *Call контролирует выполнение внешних приложений.*

Есть много уточнений, которые делают это слово очень гибким инструментом. С /info слово call возвращает объект, идентификатор свойства которого является PID запущенного процесса. Чтобы захватить отображение оболочки, вы используете уточнение /console.

Синхронизация задач также возможна с уточнением /wait, которое обязывает интерпретатор дождаться завершения запущенного процесса, прежде чем продолжить обработку скрипта. Наконец, уточнения /input и /output разрешают использование перенаправлений Unix. Предполагая, что вы хотите подсчитать количество строк в файле с помощью команды Unix wc, синтаксис Rebol будет выглядеть так: call/input "wc -l " %test.txt (что соответствует wc -l

10.2.6 Межпроцессное взаимодействие

Обмен данными между приложениями - фундаментальная часть программирования Unix. Rebol позволяет выполнять эти операции с помощью анонимных каналов и сокетов.

Pipes (Трубы) - это механизмы однонаправленной связи. Определённый типом S_IFIFO в POSIX, конвейер - это просто узел файловой системы, который состоит из двух отдельных записей в таблице файлов. Эти записи можно как читать, так и писать. Два процесса Unix могут обмениваться информацией в соответствии с моделью отправитель-получатель. Поток данных - это непрерывный поток символов, поскольку получатель не может различить разные передачи от отправителя. Более того, первые данные, вставленные в канал, всегда считываются первыми. После считывания данных собранная информация удаляется из канала, чтобы освободить трафик по нему. Следовательно, невозможно получить прямой доступ к информации, содержащейся в канале: данные должны быть сохранены получателем перед обработкой.

Для чтения и записи данных в анонимном канале Rebol предоставляет порты input и output объекта system/ports. Информация считывается путём выборки данных в system/ports/input по мере того, как она становится доступной. Итак, если мы хотим, чтобы наш сценарий Rebol myscript.r читал и отображал данные, созданные командой Unix ls -l (с использованием синтаксиса ls -l | myscript.r), нам нужен простой цикл, считывающий входной порт:

```
while [
    my-line: pick system/ports/input 1
] [ print my-line ]
```

Запись данных в анонимный канал едва ли сложнее чтения из него. Сценарий Rebol может с лёгкостью передавать данные в другое приложение Unix. Все, что нужно, это вставить данные в порт system/ports/output. Следующий пример иллюстрирует это с помощью инструкции Unix ./myscript.r|more. Последний имеет эффект отображения данных, переданных по конвейеру, на экране:

```
data: {
    a message transmitted
    via an anonymous pipe
}
insert system/ports/output data
```

Если анонимные каналы являются простым и эффективным способом передачи данных между приложениями, они по-прежнему ограничиваются обменом данными между программами, работающими на одном компьютере. Для установления межпроцессного взаимодействия на основе распределённой архитектуры системы Unix предлагают элегантный механизм, использующий протоколы TCP/IP и называемый термином сокет. Введённые в дистрибутивы Berkeley Unix, сокеты представляют собой двусторонние точки связи, с помощью которых процесс может отправлять и получать данные. Как специализированный сетевой язык, Rebol значительно сокращает работу программиста по написанию серверов и клиентов, использующих протоколы TCP или UDP.

Создание TCP-сервера начинается с объявления прослушивающего сокета, который будет использоваться для получения запросов на соединение. В Rebol сокет представлен объектом типа port! и создаётся с помощью open, за которым следует URL. Последний содержит номер порта, используемый для приёма клиентских подключений.

Чтобы изучить структуру сокета и узнать о различных доступных свойствах, всё, что вам нужно сделать, это использовать синтаксис print mold, за которым следует имя сокета.

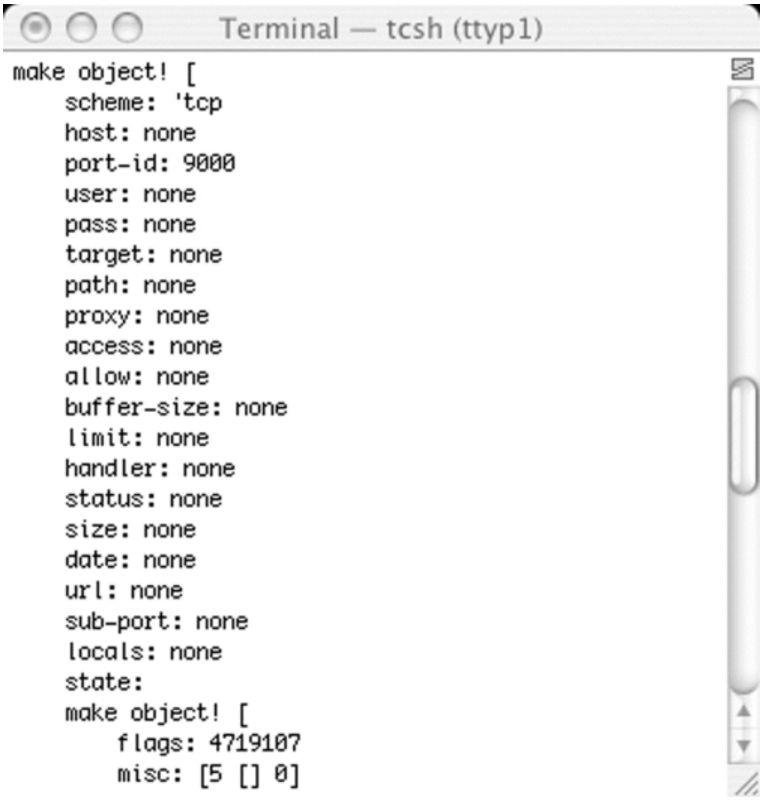

Рисунок 6-6. *Сокет TCP представлен объектом типа port!.*

После открытия сокета сценарий может войти в бесконечный цикл и ждать запросов от клиентов, используя слово wait. После установления контакта клиентский сокет возвращается как первый элемент прослушивающего сокета, а полученные данные восстанавливаются с помощью слова read-io. Ответы от сервера передаются клиенту путём вставки информации в подключённый сокет. Последний затем может быть выпущен со словом close. В следующем примере показан небольшой TCP-сервер, который прослушивает порт 9000. Он получает символьную строку, меняет её содержимое и возвращает клиенту. Клиент использует возврат каретки, чтобы показать, что передача строки завершена.

```
p: open tcp://:9000
forever [
    wait p
    conn: first p
    buffer: copy ""
    until [
        data: copy ""
        read-io conn data 255
        append buffer data
        found? find data "^/"
    ]
    insert conn (head reverse buffer)
    close conn
]
```

Работа клиента начинается с объявления сокета с помощью URL-адреса, содержащего имя или IP-адрес компьютера, на котором размещён сервер, и порт, который сервер прослушивает. Опять же, Rebol считает сокет объектом типа port!. Отправка данных на сервер просто достигается путём

вставки информации в порт, а чтение выполняется с помощью копии. Связи закрываются словом close. В следующем примере показан клиент для сервера, продемонстрированного выше. Пользователь вводит строку символов, которая передаётся на сервер. Ответ сервера отображается на экране до закрытия соединения.

```
forever [
    p: open tcp://localhost:9000
    txt: ask "#"
    insert p join txt "^/"
    print copy p
    close p
]
```

Как правило, для обмена данными между процессами используется протокол TCP. Он обеспечивает надёжный механизм передачи данных. Если скорость является определяющим критерием, Rebol также поддерживает протокол UDP, который используется практически идентично, за исключением описания сокета. Этот протокол также позволяет использовать широковещательную и многоадресную рассылку для доставки информации на несколько машин.

Наконец, помните, что Rebol включает в себя множество различных протоколов высокого уровня, таких как HTTP, SMTP и FTP, и даже можно создавать свои собственные протоколы с помощью корневого объекта, называемого root-protocol.

10.2.7 Управление сигналами Unix

Используя сокеты с Rebol, вы можете быстро разрабатывать фоновые серверные процессы. Этим демонам обычно необходимо связываться как с системой, так и с пользователями, чтобы иметь возможность получать такие команды, как stop (стоп) или restart (перезапуск).

Для выполнения таких операций системы Unix предлагают оригинальный механизм, известный как сигнал (signal). У вас есть доступ к набору сигналов, которые могут быть отправлены одним процессом другому, которые либо имеют предопределённую функцию, либо оставлены на усмотрение программиста.

Сигнал идентифицируется положительным числом и уникальным символическим именем. Событие связано с каждым из них, но сигнал вполне может быть передан другому процессу без создания события. Различные сигналы, доступные в системе UNIX, перечислены в заголовочном файле signal.h компилятора C для машины.

Сигнал	Номер	Описание
SIGHUP	1	Проблема на терминале или остановка дочернего процесса родительский процесс.
SIGINT	2	Процесс был остановлен с помощью комбинаций клавиш ctrl + c.
SIGQUIT	3	Идентичен SIGINT, за исключением сохранения состояния памяти в текущий каталог.
SIGTERM	15	Остановить процесс.
SIGUSR1	16	Определяемый пользователем сигнал.
SIGUSR2	17	Сигнал, определяемый пользователем.

Список управляемых сигналов.

Сигналы - важный аспект разработки под Unix. Фактически, они могут предупредить приложение о том, что произошло важное событие, приложение должно отреагировать соответствующим образом.

Например, если пользователь нажимает последовательность клавиш ctrl + c, чтобы остановить выполнение активной программы, она должна правильно завершить работу, сохранив все данные, которые она обрабатывала. Демоны обычно должны отвечать на приказы, передаваемые через команды Unix, такие как kill или signal. Если приказ остановиться или перезапстить, программа должна иметь возможность правильно выполнить приказ, который был отправлен.

Поэтому для управления сигналами в приложении требуется настроить диспетчер событий, который реагирует на полученные сигналы.

В Rebol у вас есть специальный протокол, называемый system (не путать с системным объектом, который выполняет совершенно иную роль). Протокол позволяет Rebol обрабатывать поступление сигналов SIGHUP, SIGINT, SIGQUIT, SIGTERM, SIGUSR1 и SIGUSR2. Для использования этого протокола порт system должен быть открыт словом open. Как только это будет завершено, вы можете определить фильтр, чтобы указать сигналы, которые должны быть перехвачены обработчиком событий. Слово set-mode принимает два параметра: используемый порт и блок, в котором присутствует свойство сигнала. Это свойство получает значение, которое представляет собой блок, содержащий различные характеристики сигнала. Если все сигналы должны быть обработаны, get-mode возвращает их список в слове 'signal-names.

После того, как фильтр определён, вы можете добавить этот обработчик событий к тем, которые уже присутствуют в Rebol. Просто добавьте порт, который обрабатывает сигналы, в блок system/ports/wait-list. Функция enable-system-trap выполняет эти различные операции и активирует приёмник сигналов в интерпретаторе Rebol.

```
enable-system-trap: does [
    system/ports/system: open [ scheme: 'system ]
    set-modes system/ports/system [
        signal: get-modes system/ports/system 'signal-names
    ]
]

append system/ports/wait-list system/ports/system
```

Чтобы узнать, был ли сигнал перехвачен вашим скриптом Rebol, всё, что вам нужно сделать, это периодически проверять содержимое порта system/ports/system.

Если сигнал прибыл, полученное сообщение будет представлять собой блок, состоящий из двух элементов: слова 'signal и имени сигнала.

Функция check-system-trap выполняет эту операцию и отображает на экране имя захваченного сигнала:

```
check-system-trap: func [ /local msg ] [
    while [ msg: pick system/ports/system 1 ] [
        print form second msg
    ]
]

enable-system-trap

forever [ check-system-trap ]
```

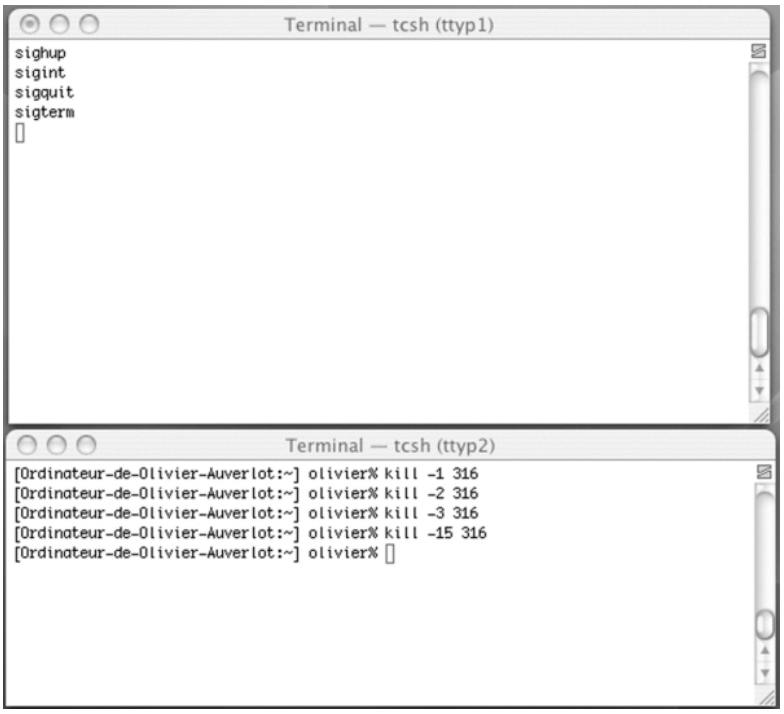

Рисунок 6-7. *Сценарий Rebol обнаруживает сигналы, передаваемые командой Unix kill.*

Обработка сигналов демонстрирует лёгкость, с которой интерпретатор Rebol может взаимодействовать с исполняющей системой своего хоста. Такая лёгкость присутствует, когда сценарий Rebol взаимодействует с функциями и структурами данных в динамических библиотеках.

10.2.8 Взаимодействие с динамическими библиотеками

Ранее только Rebol/IOS, Rebol/View/Pro, Rebol/SDK и Rebol/Command могли взаимодействовать с динамическими библиотеками собственного кода. Начиная с выпуска Rebol 2.7.6, Rebol/View также может обращаться к внешним динамическим библиотекам. Это означает, что скрипты Rebol могут объявлять и вызывать функции, написанные на таких языках, как C или C ++. Некоторые части скриптов можно оптимизировать, вызывая собственные функции, скорость выполнения которых намного выше, чем у интерпретируемого кода Rebol. Rebol может не только использовать преимущества API операционной системы, но и пользоваться множеством специализированных библиотек (создание изображений, создание документов в формате pdf, подключение к ядрам баз данных и т.д.). Такая библиотека используется, чтобы Rebol мог подключаться для использования Berkeley DB (http://www.cs.unm.edu/~whip) и движков баз данных DyBase (http://www.garret.ru/~knizhnik/dybase.html).

Rebol включает в себя все необходимое, чтобы легко и быстро писать обёртки для библиотек. Первый шаг - загрузить динамическую библиотеку, функции которой будут использоваться. Эта операция выполняется со словом load с его уточнением /library, возвращающим идентификатор. Объявление различных функций выполняется с помощью типа данных routine! с объявлением входных параметров функции, её возвращаемым значением, идентификатором библиотеки, содержащей функцию, и именем функции в динамической библиотеке. Полученное слово добавляется в словарь и имеет те же атрибуты, что и все остальные слова. Вы даже можете предоставить документацию, чтобы слово справки содержало информацию о его использовании. Единственная разница в том, что исходный код не виден, поскольку это нативная функция. Наконец, когда динамическая библиотека больше не нужна, вы можете освободить занимаемую ею память, используя слово free, за которым следует имя её идентификатора.

Чтобы лучше понять, как это работает, мы добавим слово get-nprocs в словарь Rebol. Он будет использовать функцию из стандартной библиотеки libc для определения количества процессоров на машине. Эта функция не принимает никаких входных параметров и возвращает только простое целое число.

```
lib-so: load/library %libc.so.6

get-nprocs: make routine! [
    return: [ integer! ]
] lib-so "get_nprocs"

print [ "Number of processors:" get-nprocs ]

free lib-so
```

Предоставление параметров для встроенной функции не представляет большой трудности, если они соответствуют сопоставлению типов данных C и Rebol.

C Language	Rebol
Char	Char!
Short	Integer!
Long	Integer!
Int	Integer!
Float	Decimal!
Double	Decimal!
Struct*	Struct!
Char*	String!

C и эквивалентные типы данных Rebol

В следующем примере объявляется слово put-env, которое добавляет переменную в среду выполнения. Также обратите внимание, что это слово задокументировано для запроса с помощью слова help:

```
lib-so: load/library %libc.so.6

put-env: make routine! [
    "Adds a variable in the environnement"
    new-value [ string! ]
    return: [ integer! ]
] lib-so "putenv"

put-env "MYVARIABLE=test"
print get-env "MYVARIABLE"

free lib-so
```

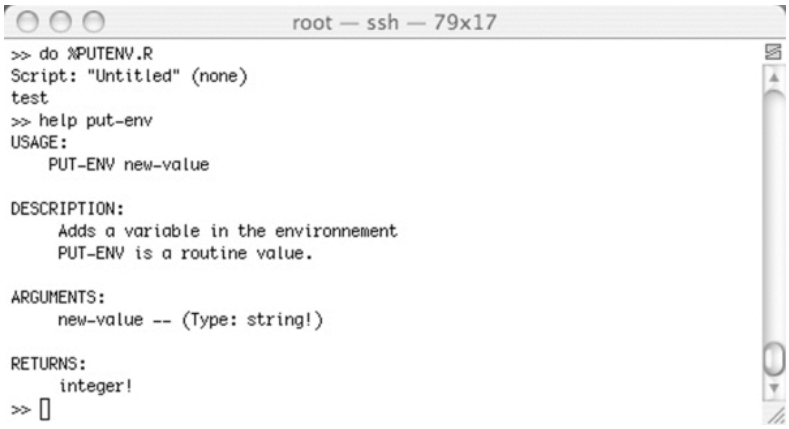

Рисунок 6-8. *Слово put-env добавлено в словарь Rebol.*

Очевидно, что некоторые объявления намного сложнее, чем в предыдущих примерах. Многие функции используют структуры для получения или возврата данных. По этой причине Rebol включает в себя struct! тип данных. Если мы хотим использовать функцию getpwuid из libc, чтобы узнать информацию о пользователе, идентифицированном его UID, мы должны объявить структуру для получения возвращаемых данных:

```
lib-so: load/library %libc.so.6

get-uid: make routine! [
        return: [ integer! ]
] lib-so "getuid"

getpwuid: make routine! [
        uid [ integer! ]
        return: [ struct! [
                pw_name [ string! ]
                pw_passwd [ string! ]
                pw_uid [ integer! ]
                pw_gid [ integer! ]
                pw_gecos [ string! ]
                pw_dir [ string! ]
                pw_shell [ string! ]
        ] ]
] lib-so "getpwuid"

my-uid: get-uid
print [ "My UID is: " my-uid ]
me: getpwuid my-uid
print [ "My Login is:" me/pw_name ]

free lib-so
```

Этот раздел продемонстрировал лёгкость, с которой Rebol может использовать скорость конкретной платформы, и её отличную интеграцию со средой Unix. Имея минимум кода, Rebol позволяет легко разрабатывать простые, мощные приложения, которые могут быть полностью интегрированы в их операционную среду. Универсальность языка делает его предпочтительным инструментом во всех системах Unix.

10.3 Базы данных с RebDB

RebDB был и продолжает развиваться Эшли Трютер. Это движок базы данных, который можно напрямую интегрировать в приложение Rebol, а также использовать в модели клиент/сервер через TCP/IP. Лёгкий и бесплатный RebDB можно использовать даже тогда, когда конечный продукт носит коммерческий характер. Вы можете загрузить его с веб-сайта Эшли http://www.dobeash.com/RebDB/ в виде простого zip-архива. Проект тщательно задокументирован, поскольку руководство по базам данных и руководство по SQL доступны на сайте автора. Документы быстро показывают, что RebDB - это не игрушка, а замечательный продукт с множеством функций, доступных разработчикам. Совместимый со всеми версиями Rebol, он предлагает возможность создавать таблицы данных, манипулировать информацией с использованием синтаксиса, подобного SQL, и даже распределять данные, работая в режиме клиент-сервер. Таким образом, с помощью RebDB любое приложение Rebol может встроить настоящий механизм базы данных, который весит всего семьдесят килобайт. Этот движок, полностью написанный на Rebol, сильно оптимизирован за счёт сильных сторон языка. Он также использует преимущества переносимости языка и может работать таким же образом на любой из платформ, поддерживаемых Rebol.

10.3.1 Начало работы с RebDB

RebDB состоит из трех скриптов Rebol; db.r - это основной механизм базы данных, db-client.r - это клиентский модуль при реализации в режиме клиент/сервер, а SQL.r обеспечивает доступ с консоли SQL к базам данных RebDB. По умолчанию RebDB хранит базы данных в каталоге, из

которого был загружен движок. Это можно изменить, но для того, чтобы воспользоваться функциями автоматического восстановления RebDB, лучше придерживаться поведения по умолчанию. Поэтому, как правило, лучше всего создать каталог, содержащий копии всех сценариев RebDB для каждой из ваших баз данных.

Как только вы это сделаете, всё, что нужно для загрузки ядра базы данных, - это выполнить %filepath/db.r, где filepath - это путь к файлу каталога базы данных относительно текущего каталога.

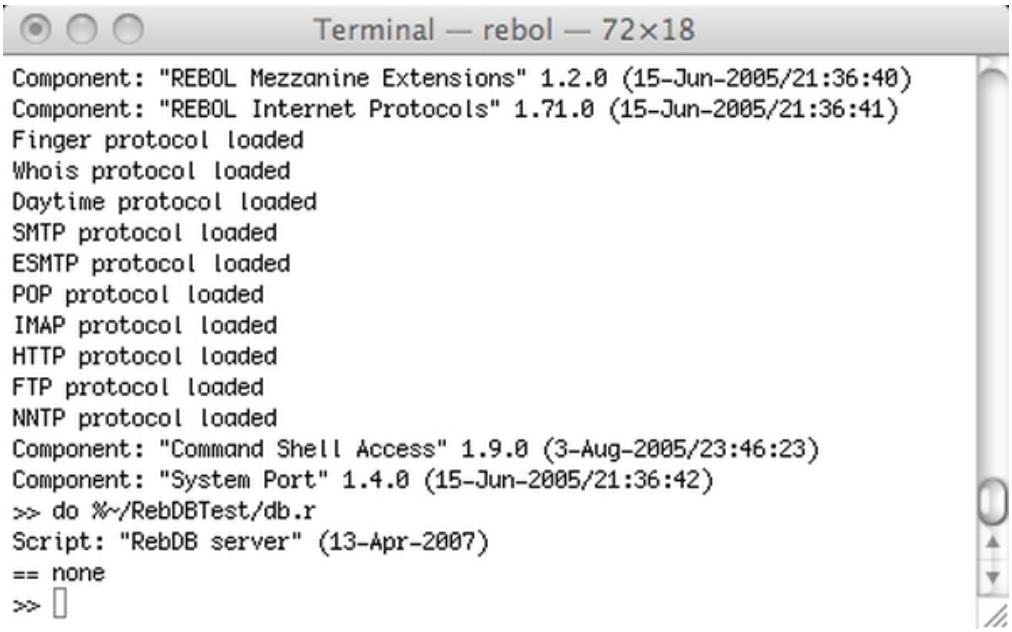

Рисунок 6-9. *Загрузка протокола RebDB.*

После завершения этой простой операции в словарь вашего интерпретатора добавляется набор новых слов с префиксом "db-". Вы можете перечислить эти слова, введя команду help db- в консоли Rebol. Самый эффективный способ узнать о RebDB - работать прямо в консоли, чтобы увидеть, как работает каждая из функций. Результаты каждого запроса данных или команды обслуживания перенаправляются на консоль для отображения.

10.3.2 Таблицы и поля

Для организации и хранения данных RebDB использует традиционную модель таблиц. В этих таблицах каждый столбец является полем, а каждая строка - записью (состоящей из одного или нескольких полей). Количество таблиц, которые можно создать, ограничено только объёмом памяти вашего компьютера. Фактически, хотя RebDB использует файлы для резервного копирования данных, он сохраняет всю информацию в памяти, чтобы минимизировать время обработки. Поэтому RebDB очень эффективен для баз данных с небольшими записями.

С другой стороны, если ваши базы данных состоят из многих тысяч гигабайт, было бы лучше обратиться к проверенным и испытанным альтернативам, таким как MySQL или PostgreSQL.

Один из аспектов RebDB, который даёт ему скорость для обработки большого количества строк данных, заключается в том, что при изменении таблицы она не сохраняется автоматически на диск. Вместо этого он записывает изменения в файл журнала и оставляет программисту возможность сохранить таблицы на диск с помощью команды db-commit, за которой следует имя таблицы. Если ваша программа завершится без сохранения изменений таблиц, базу данных можно быстро восстановить с помощью команды db-replay, за которой следует имя таблицы. Если база данных находится в том же каталоге, что и сценарий db.r, RebDB автоматически выдаёт команду db-replay при запуске для восстановления любых незафиксированных изменений.

Если в любой момент вы захотите отменить какие-либо изменения в базе данных, которые ещё не зафиксированы, вы можете легко сделать это с помощью команды db-rollback, которая принимает имя таблицы в качестве параметра.

Всё, что вам нужно сделать, чтобы создать таблицу, - это использовать слово db-create, за которым следует имя таблицы и список её столбцов. Вам не нужно беспокоиться об указании типа или длины поля. В Rebol типы имеют значения, а не переменные. RebDB создан, чтобы в полной мере использовать это. Также нет необходимости указывать ключи или индексы с RebDB. Его оптимизированный подход в памяти делает их избыточными.

Почему бы не использовать это введение в RebDB для настройки небольшого приложения для управления DVD коллекцией? Начните с создания двух таблиц, которые будут полезны, это таблица dvd и таблица номенклатуры стилей.

db-create dvd [key title style director duration description] db-create styles [key name]

Как только эти две таблицы будут созданы, вы должны увидеть несколько новых файлов в вашем текущем каталоге. Файлы с расширением .dat содержат данные, а файлы с расширением .ctl содержат структуру таблиц. Для каждой таблицы у вас должен быть файл .ctl и файл .dat. Вы можете проверить структуру таблицы с помощью db-describe. Он предоставляет блок Rebol, показывающий имя и тип каждого столбца.

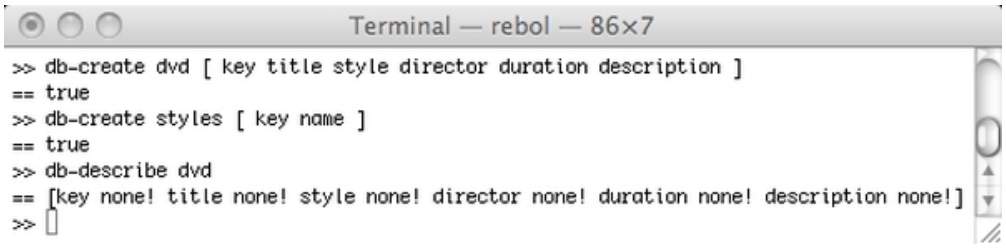

Рисунок 6-10. *Проверка структуры таблицы.*

Вы могли заметить, что для всех столбцов в таблице установлено значение none!. Типы столбцов более информативны, чем обязательны, как в стандартных базах данных SQL. Они отражают типы значений в первой строке таблицы, как мы увидим ниже. Фактически, RebDB с радостью позволит вам смешивать типы в столбце. Хотя, если вы это сделаете, не удивляйтесь, когда встроенные функции агрегирования RebDB, такие как sum и avg, дают странные результаты.

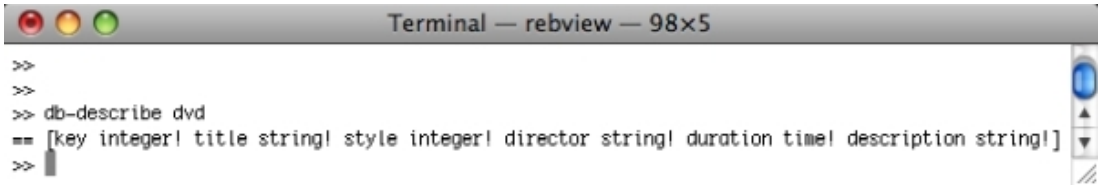

Рисунок 6-11. *Структура таблицы после добавления строки.*

При db-close указанная таблица выгружается из памяти. RebDB включает механизм безопасности, который позволяет вам закрывать только таблицы, которые не были изменены с момента последнего сохранения.

Слово db-drop полностью стирает таблицу. Остерегайтесь, действие этого слова необратимо, и ваши данные будут безвозвратно потеряны.

10.3.3 Манипулирование данными

Управление данными в таблицах возложено на набор слов, предназначенных для вставки, изменения, выбора или удаления информации. Когда он не работает в режиме клиент-сервер, RebDB не позволяет пользователю писать запросы с использованием SQL. Но будьте уверены, синтаксис, выбранный для различных слов, остаётся очень близким к SQL как по духу, так и по форме. Потребуется всего несколько минут, чтобы освоиться с этими новыми словами.

Таким образом, db-insert позволяет вставлять новую строку в таблицу. Это слово принимает два аргумента: имя таблицы и блок, содержащий значения, которые должны быть вставлены в неё. С помощью db-update вы можете изменить одно или несколько полей в данной таблице. Уточнение /where позволяет вам определить, какие записи будут изменены в соответствии с выражением

Rebol, которое должно оцениваться как логическое значение. Удаление данных поручено команде db-delete, которая работает с данной таблицей и также применяет логическое выражение Rebol для определения, какие данные должны быть удалены. Наконец, db-select ищет информацию в таблице. Его два аргумента - это имена выбранных столбцов и имя таблицы для поиска. Эти слова позволяют использовать слово * для выбора всех полей в таблице. Доступны многие уточнения, такие как /where сделать условный выбор или /order и /desc для сортировки данных. Результатом является блок, содержащий данные в каждом файле(???) для разных строк.

10.3.4 Запускаем это в работу

Чтобы лучше понять, как работает RebDB, вы собираетесь создать приложение для управления коллекцией DVD с помощью Rebol/View. Вы уже создали dvd-таблицы и таблицы стилей, используемые вашим продуктом. Сценарий MyDVD начинается с загрузки RebDB и проверки наличия базы данных. В случаях, когда база данных не существует, этот код создаст базу данных и инициализирует номенклатуру стилей, используя предопределённые значения. Внутри аргумента команды db-insert использование слова next указывает, что значение поля должно автоматически увеличиваться для каждой новой записи.

```
do %~/RebDBTest/db.r

if not exists? %dvd.dat [
    if error? try [
        db-create dvd [key title style director duration description]
        db-create styles [ key name ]
        foreach styledvd [
            "Action" "Adventure" "Humour" "Music" "Kids"
        ] [
            db-insert styles compose [ next (styledvd) ]
        ]
    db-commit styles
    ] [
    alert "Unable to create the database"
    quit
    ]
]
```

Ваше приложение должно допускать множество операций. Четыре кнопки со стрелками позволяют перемещаться по базе данных. Пользователь может перейти непосредственно к первой или последней записи, а также перейти вперёд и назад по одной записи за раз. Чтобы имитировать слайдер, коллекция блоков инициализируется словом init-slider.

```
init-slider: func [ /refresh /filter styl ] [
    either not filter [
        collection: db-select rowid dvd
    ] [ collection: db-select/where rowid dvd compose
    [ style = (styl) ] ]
    if not refresh [
        either (length? collection) > 0 [
            slider: 1
            modif: true
            aff-dvd
        ] [
            modif: false ; by default, we have inserted a ew record
            slider: none
        ]
    ]
]
```

Учитывая, что пользователь может фильтровать фильмы по стилю, это слово использует уточнение /filter, которое обеспечивает соответствующий выбор DVD. Блок коллекции содержит номер строки каждой записи (rowid). Когда пользователь нажимает одну из кнопок со стрелками, переменная ползунка обновляется. Различные поля отображаются с помощью aff-dvd. Он извлекает данные, используя ползунок в качестве индекса для блока коллекций, а затем извлекает запись с этим идентификатором строки.

Рисунок 6-12. *Графический интерфейс приложения "MyDVD".*

Другой запрос к базе данных извлекает стиль DVD путём опроса таблицы номенклатуры. Обновление каждого поля выполняется с помощью нововведения, представленного в версии 1.3 View: "аксессоры" (accessors). Это слова set-face, get-face и clear-face, которые напрямую обращаются к значению графического компонента.

```
aff-dvd: func [ /local data ] [
    data: db-select/where * dvd compose [
        rowid = (pick collection slider)
    ]
    set-face style-dvd db-select/where name styles compose [
        cle = (data/3)
    ]
    set-face title data/2
    set-face director data/4
    set-face duration data/5
    set-face description data/6
    modif: true
]
```

10.3.5 Редактирование и сохранение

Вставка или изменение записи происходит в layout (макет), определяющем графический интерфейс вашего приложения. Кнопка "Save" (Сохранить) выполняет db-update или db-insert в зависимости от значения логической переменной modif. В обоих случаях программа консервативно выдаёт команду db-commit для сохранения данных на диск.

Он также должен выполнить поиск в таблице стилей, чтобы найти ключевое значение стиля, выбранного пользователем, чтобы правильно обновить поле стиля в таблице DVD. Для поиска по строкам необходимо использовать простое выражение Rebol.

```
btn-enter "Save" [
    if error? try [
        either modif [
            db-update/where dvd [
                title style director
                duration description
            ] compose [
                (get-face title)
                (first db-select/where key styles compose [ name = (get-face
style-dvd) ] )
                (get-face director) (to-time get-face duration)
                (get-face description)
            ] compose [ rowid = (pick collection slider) ]
        ] [
            db-insert dvd reduce [
                'next (get-face title)
                (first db-select/where key styles compose [ name = (get-face
style-dvd) ] )
                (get-face director)
                (to-time get-face duration) (get-face description)
            ]
            init-slider/refresh
            slider: length? collection
        ]
    ] [ alert "Unable to save" ]
]
```

Удаление записи требует использования db-delete. Уточнение /where позволяет выбрать строку для удаления. Значение key находится путём выбора активного элемента в блоке collection. В зависимости от случая программа отображает либо запись перед удалённой (или новую первую запись, если она была первой удалённой), либо пустые поля. В последнем случае есть две возможности управлять. Фактически, возможно, что база данных больше не содержит никаких записей, а также что никакие записи не соответствуют стилю, выбранному пользователем.

```
btn "Erase" [
    if not none? slider [
        db-delete/where dvd compose [
            key = (first db-select/where key dvd compose [
                rowid = (pick collection slider)
            ] )
        ]
        db-commit dvd
        init-slider/refresh
        either (length? collection) > 0 [
            slider: slider - 1
            if slider = 0 [ slider: 1 ]
            aff-dvd
        ] [
            slider: none
            empty-fields
        ]
    ]
]
```

10.3.6 Публикация вашей базы данных в сети

Почему бы не поделиться своей коллекцией DVD с друзьями и не отображать различные названия своей коллекции на своём веб-сайте? RebDB имеет режим клиент-сервер, который упрощает эту задачу. Это позволяет одновременно подключаться к базе данных, избегает загрузки таблиц для каждого подключенного клиента и имеет диалект SQL, довольно близкий к исходному. Последний включает в себя основные команды SQL (select (выбор), insert (вставка), delete (удаление) и update (обновление)). Таким образом, можно писать сложные запросы, такие как:

```
select [ title director ]
from dvd
where [ duration < 2:00 ]
order by title desc
```

Установка требует только создания каталога rebdb, копирования библиотек RebDB и файлов вашей базы данных. Затем вы должны написать короткий скрипт для запуска сервера RebDB, используя слово listen. Вы можете выбрать, какой TCP-порт использовать.

```
REBOL [ subject: "Launch RebDB server" ]
do %db.r
listen tcp://:10000
```

После запуска этого сценария сервер RebDB ожидает подключений от клиентов. Теперь осталось написать динамическую страницу. Чтобы облегчить жизнь, воспользуетесь библиотекой Magic!. Сначала скопируйте файлы RebDB в Magic! каталог, предназначенный для хранения данных и обмена библиотеками. Страница dvd.rhtml требует наличия клиентской библиотеки RebDB, а также Magic! Компоненты HTML для удобного представления данных.

Запрос к базе данных выдаётся с использованием слова db-request, за которым следует URL-адрес и сам запрос.

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="magic.css">
</head>
<body>
<rebol>
 library %db-client.r
 library %html.r
 data: db-request tcp://192.168.0.1:10000 [
  select [ title director duration description ]  from  dvd
 ]
 block: copy []
 forskip data 4 [
 append/only block reduce [ data/1 data/2 data/3 data/4 ]
 ]
 html/datagrid block
</rebol>
</body>
</html>
```

Результат запроса присваивается переменным данным, а затем переформатируется в соответствии со структурой данных, ожидаемой компонентом HTML таблицы данных Magic!.

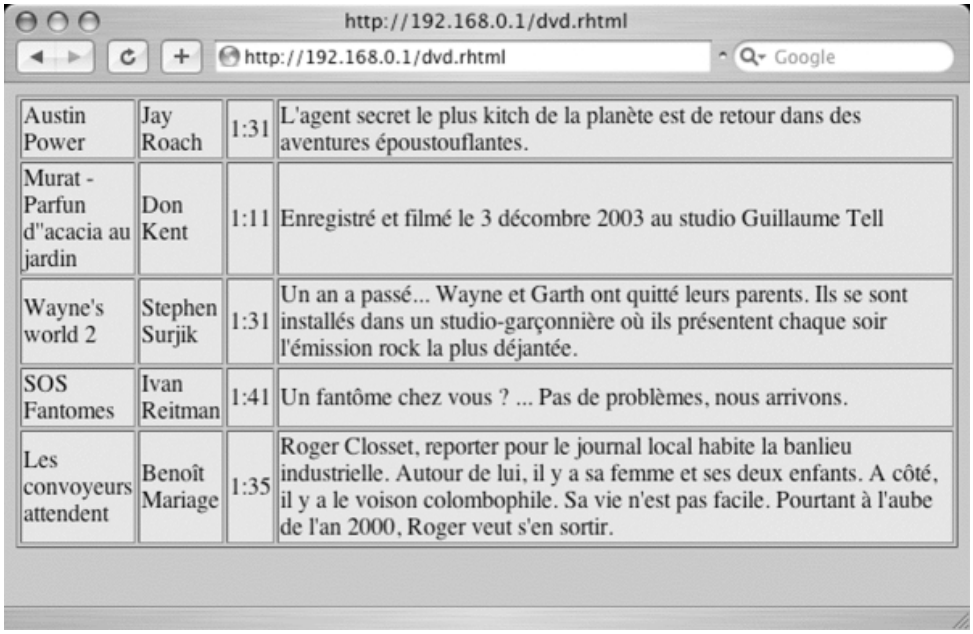

Рисунок 6-13. *MyDVD отображается в браузере.*

RebDB - эффективное решение для управления небольшими базами данных Rebol и, что наиболее важно, помогает избежать использования эффективных, но тяжёлых, многомерных продуктов, таких как MySQL или PostgreSQL, в небольших проектах.

Не требует какой-либо конфигурации на локальном компьютере, не зависит от платформы, совместим со всеми версиями Rebol, может использоваться в локальном или клиент-серверном режиме, он упрощает работу программиста, когда обработка и хранение данных становятся сложными.

10.4 Резюме

Rebol полон оригинальных технологий. Виртуальные офисы могут распространять приложения и информацию через Интернет. В области программирования Unix Rebol имеет множество интерфейсов для использования функций и возможностей этих операционных систем. С RebDB можно разрабатывать многоуровневые приложения, полностью написанные на Rebol.

11. Практическое применение

Эта глава состоит из нескольких семинаров, которые позволят вам применить полученные знания. В этих тематических исследованиях основное внимание уделяется основам создания видеоигры, разработке программы чата, написанию консоли администрирования MySQL и, наконец, созданию реблета для Rebol/IOS.

11.1 Написание движка Raycasting с View

Ребол - это общий язык. На Rebol можно написать практически любую программу. Он лёгкий, очень эффективно обрабатывает данные и обладает неоспоримыми качествами в области сетевого программирования. Это также одарённый язык в области графики и анимации. GCS (Graphical Compositing System) - действительно платформо-независимая мультимедийная библиотека. Чтобы проиллюстрировать его возможности, вы собираетесь создать настоящий движок Raycasting всего с 3 КБ кода.

11.1.1 Что такое Raycasting?

Проще говоря, рейкастинг - один из самых интригующих аспектов в области видеоигр. Ваша память может помочь вам лучше понять ... Некоторые из вас помнят игру Wolfstein, которая

появилась в 1990-х годах. Это была первая игра, в которой игрок был погружен в своё окружение. Ваш персонаж перемещается по лабиринту с анимацией стен и NPC (неигровых персонажей) в реальном времени.

Учитывая производительность машин того времени (Intel 286 и 386), это было просто невероятно! Как Джон Кармак, автор игры, совершил это чудо? Ответ прост: обманом.

11.1.2 Мне это кажется реальным!

Фактически, raycasting - это огромный и большой обман, потому что он вообще не использует 3D-алгоритм. Игрок перемещается по 2D-вселенной, экранная визуализация которой создаётся запускающими лучами.

Игрок просто перемещается по двухмерному столу, каждая ячейка которого может быть пустой или представлять собой виртуальный куб, определяемый цветом или текстурой. Вы просто определяете поле зрения, ширина которого обычно составляет 90 градусов, что соответствует минимальным возможностям человека. После создания этого конуса вы запускаете 90 лучей, чтобы найти первое пересечение со стеной. Осталось только определить высоту сегмента стены в зависимости от расстояния и нарисовать его на экране.

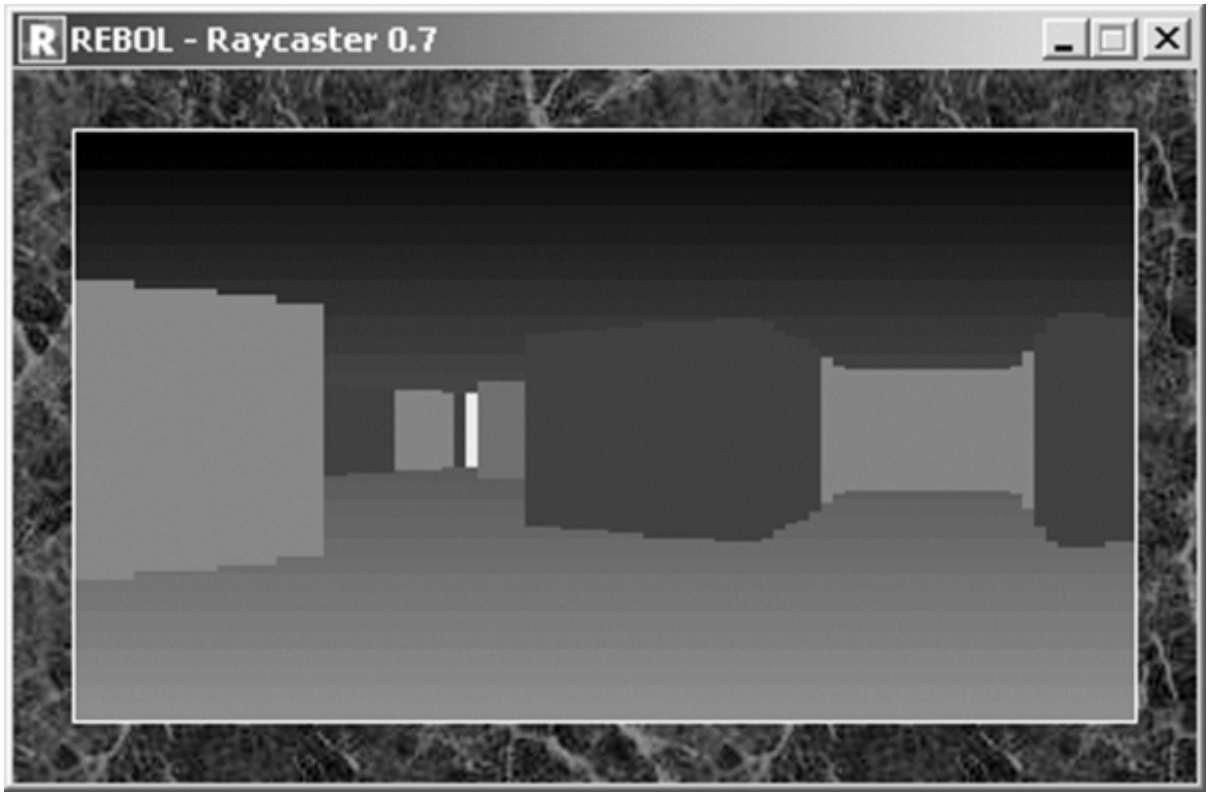

Рисунок 7-1. *Вид лабиринта изнутри.*

В Wolfstein (а также в нашем движке) стены спроектированы так, чтобы быть симметричными относительно оси x, которая представляет горизонт. Этот алгоритм имеет преимущество простоты, но не позволяет игроку смотреть вверх или вниз.

11.1.3 Начиная с основ

Прежде всего, вы должны определить поверхность игры и ряд переменных и вспомогательных функций. Лабиринт (laby) определяется списком блоков (каждый блок соответствует строке таблицы). Если значение поля не равно нулю, это означает, что поле содержит куб.

Чтобы узнать цвет стен этого куба, просто посмотрите на список, инициализированный 16 цветами знаменитой, но древней палитры (palette) Qbasic. Вам также необходимо зафиксировать положение игрока (px и py), расстояние, пройденное за один шаг (stride), направление движения в

градусах (heading) и количество градусов при повороте (turn). Чтобы ускорить вычисления, вы также определяете список, содержащий значения косинусов (ctable) для определённого количества углов. Функция get-angle позволяет извлечь значение из этой таблицы.

```
REBOL [
    subject: "raycasting engine"
    version: 0.7
]
px: 9 * 1024
py: 11 * 1024
stride: 5
heading: 0
turn: 10
laby: [
    [  8  7  8  7  8  7  8  7  8  7  8  7 ]
    [  7  0  0  0  0  0  0  0 13  0  0  8 ]
    [  8  0  0  0  12 0  0  0 14  0  9  7 ]
    [  7  0  0  0  12 0  4  0 13  0  0  8 ]
    [  8  0  4  11 11 0  3  0  0  0  0  7 ]
    [  7  0  3  0  12 3  4  3  4  3  0  8 ]
    [  8  0  4  0  0  0  3  0  3  0  0  7 ]
    [  7  0  3  0  0  0  4  0  4  0  9  8 ]
    [  8  0  4  0  0  0  0  0  0  0  0  7 ]
    [  7  0  5  6  5  6  0  0  0  0  0  8 ]
    [  8  0  0  0  0  0  0  0  0  0  0  7 ]
    [  8  7  8  7  8  7  8  7  8  7  8  7 ]
]

ctable: []
for a 0 (359 + 180) 1 [
    append ctable to-integer (((cosine a ) * 1024) / 10)
]

palette: [
    0.0.128 0.128.0 0.128.128
    0.0.128 128.0.128 128.128.0 192.192.192
    128.128.128 0.0.255 0.255.0 255.255.0
    0.0.255 255.0.255 0.255.255 255.255.255
]

get-angle: func [ v ] [ pick ctable (v + 1) ]
```

Некоторые из этих значений масштабируются путём умножения на 1024. Это позволяет игроку перемещаться на каждом ходу, а не прыгать от коробки к коробке.

11.1.4 Формируем стены

Чтобы рисовать на экране, вам понадобится окно, содержащее картинку. Вот этот макет (layout), называемый screen (экраном). Изображение, используемое для дизайна поверхности, называется display (дисплеем) и имеет размер 360 на 200 пикселей.

Вы уже можете сделать вывод, что каждый запущенный луч будет иметь ширину 4 пикселя (360/90). Gradient позволяет наложить градиент, чтобы представить пол и потолок вашего лабиринта. Функция refresh-display вызывает движок raycasting и отображает содержимое области.

```
refresh-display: does [
    retrace
    show display
]
```

```
screen: layout [
    backtile %marbre.jpg
    display: box 360x200 effect [
        gradient 0x1 0.0.0 128.128.128
        draw []
    ] edge [
        size: 1x1
        color: 255.255.255
    ]
]
refresh-display
view/title screen join "Raycaster " system/script/header/version
```

В основе программы лежит функция retrace. Цикл for находит 90 углов, под которыми должен быть запущен луч. Виртуальная линия проводится от позиции игрока на основе каждого угла по очереди. В первом непустом поле (его значение не равно нулю) функция вычисляет высоту сегмента стены и определяет положение и размеры прямоугольника, который он представляет. Все, что осталось сделать, это получить цвет из палитры и добавить инструкции по рисованию в атрибут effect/draw изображения.

```
retrace: does [
    clear display/effect/draw
    xy1: xy2: 0x0
    angle: remainder (heading - 44) 360
    if angle < 0 [ angle: angle + 360 ]
    for a angle (angle + 89) 1 [
        xx: px
        yy: py
        stepx: get-angle a + 90
        stepy: get-angle a
        l: 0
        until [
            xx: xx - stepx
            yy: yy - stepy
            l: l + 1
            column:  make integer! (xx / 1024)
            line: make integer! (yy / 1024)
            laby/:line/:column <> 0
        ]
        h: make integer! (900 / l)
        xy1/y: 100 - h
        xy2/y: 100 + h
        xy2/x: xy1/x + 3
        color: pick palette laby/:line/:column
        append display/effect/draw reduce [
            'pen color
            'fill-pen color
            'box xy1 xy2
        ]
        xy1/x: xy2/x + 1
    ]
]
```

11.1.5 Завершение взаимодействия

Теперь вы должны управлять клавиатурой, чтобы перемещаться по лабиринту. Для этого вы определяете функцию evt-key, которая вставляется в обработчик событий. В зависимости от нажатой клавиши (up (вверх), down (вниз), left (влево) и right (вправо)) вы вызываете функцию

player-move. Он проверяет движение игрока, проверяя, сталкивается ли игрок со стеной, и перерисовывает экран, вызывая функцию refresh-display.

```
player-move: function [ /backwards ] [ mul ] [
    either backwards [ mul: -1 ] [ mul: 1 ]
    newpx: px - ((get-angle (heading + 90)) * stride * mul)
    newpy: py - ((get-angle heading) * stride * mul)
    c: make integer! (newpx / 1024)
    l: make integer! (newpy / 1024)
    if laby/:l/:c = 0 [
        px: newpx
        py: newpy
        refresh-display
    ]
]

evt-key: function [ f event ] [] [
    if (event/type = 'key) [
        switch event/key [
            up [ player-move ]
            down [ player-move/backwards ]
            left [
                heading: remainder (heading + (360 - turn)) 360
                refresh-display
            ]
            right [
                heading: remainder (heading + turn) 360
                refresh-display
            ]
        ]
    ]
    event
]
insert-event-func :evt-key
```

Представленный здесь алгоритм не самый эффективный, но, вероятно, один из самых простых. Есть ещё много возможностей для оптимизации, и, особенно, внешний вид можно значительно улучшить за счёт использования текстур.

11.2 Запрограммируйте свой "чат" в Rebol

Давайте продолжим изучение Rebol, разработав приложение для чата, которое позволяет группе людей вести обсуждения в Интернете в режиме реального времени. Для этого проекта нам потребуется разработать клиентский модуль с Rebol/View и серверный модуль, состоящий из сценария CGI, написанного на Rebol/Core.

11.2.1 Основные принципы

Концепция довольно проста: каждый клиент использует утилиту, написанную на Rebol/View, для предоставления графического пользовательского интерфейса. Пользователь, идентифицированный по имени (логину), может ввести сообщение и отправить его на сервер. Периодически клиентский модуль проверяет наличие новых сообщений на сервере, извлекает их и отображает на экране. Пользователи могут общаться в чате в реальном времени.

Важно отметить, что выбранная техника заключается в том, что клиенты спрашивают сервер, доступны ли новые сообщения. Сервер никогда не связывается с клиентами, чтобы сообщить им, что сообщения ожидают. В этом режиме работы, основанном на запросе/ответе, мы можем легко использовать сценарии CGI и особенно протокол HTTP.

Таким образом, наш клиент может делать запросы через TCP-порт 80, что позволяет ему беспрепятственно проходить через любой брандмауэр. Наша система чата отлично подходит для использования в Интернете, а не только в частной сети.

11.2.2 Написание клиента

Первый этап состоит в том, чтобы спросить пользователя, как его зовут. Это можно сделать с помощью простого диалогового окна и присвоения введённого значения переменной login.

```
login: request-text/title "Your login:"
```

Рисунок 7-2. *Ввод логина пользователя.*

Главный клиент View состоит из окна, содержащего строку для ввода текста, кнопку для отправки введённого текста и список сообщений с именем отправителя. Поэтому нам необходимо определить макет размером 500 на 500 пикселей, содержащий три графических компонента.

```
view layout/size [
    origin 0x0 space 0x0
    msgs: text-list 500x470 rate delay feel [
        engage: func [ f a e ] [
            if a = 'time [ read-messages ]
        ]
    ]
    across
    txt: field 400x30
    button "Send" 100x30 [ send-message ]
] 500x500
```

Таймер определяется в компоненте с именем msgs. Частота срабатывания таймера определяется переменной delay, которая содержит количество секунд, разделяющих два клиентских соединения с сервером. Затем функция engage вызывает функцию read-message (чтения сообщения) тогда и только тогда, когда обнаружено, что событие относится к типу ‘time.

Для кнопки мы указываем, что когда пользователь нажимает на неё, вызывается функция отправки сообщения. это делает запрос к удалённому HTTP-серверу с помощью метода POST и передаёт ему сообщение, имя отправителя и описательное действие, указывающее, что пользователь отправляет новое сообщение. URL-адрес сервера содержится в переменной server, определённой в начале скрипта.

```
server: http://jupiter/cgi-bin/rchat.cgi

send-message: does [
    read/custom server reduce [
        'POST
        join {action=SEND&login=} [
            login "&message=" txt/text
        ]
    ]
```

```
    insert head msgs/lines join login [ ": " txt/text ]
    fix-slider msgs
    txt/text: copy ""
    show [ txt msgs ]
]
```

Нам нужно правильно инициализировать ползунок списка сообщений. Для этого служебная функция fix-slider регулирует положение и высоту ползунка на вертикальной панели в соответствии с количеством полученных сообщений.

```
fix-slider: func [ faces ] [
    foreach list to-block faces [
        either 0 = length? list/data [
            list/sld/redrag 1
        ] [ list/sld/redrag list/lc / length? list/data ]
    ]
]
```

После отправки содержимое строки сообщения стирается, и сообщение добавляется в список. Эти два компонента обновляются на экране с помощью слова show.

Осталось только написать функцию чтения сообщений read-messages, которая извлекает с сервера сообщения, отправленные другими пользователями.

Это действие определяется как "READ", и передаётся имя пользователя. Затем сценарий CGI возвращает все сообщения, написанные другими людьми, подключенными к серверу, на странице с типом MIME text/plain.

Переменная num-msg содержит номер последнего прочитанного сообщения: сервер возвращает только сообщения с номером больше, чем в этой переменной.

```
read-messages: does [
    responses: read/custom server reduce [
        'POST
        join {action=READ&login=} [ login "&num=" num-msg ]
    ]
    if (length? trim responses) > 0 [
        resp: do responses
        num-msg: (first resp) + 1
        msg: second resp
        forskip msg 2 [
            insert head msgs/lines join msg/1 [ ": " msg/2 ]
        ]
    fix-slider msgs
    show msgs
    ]
]
```

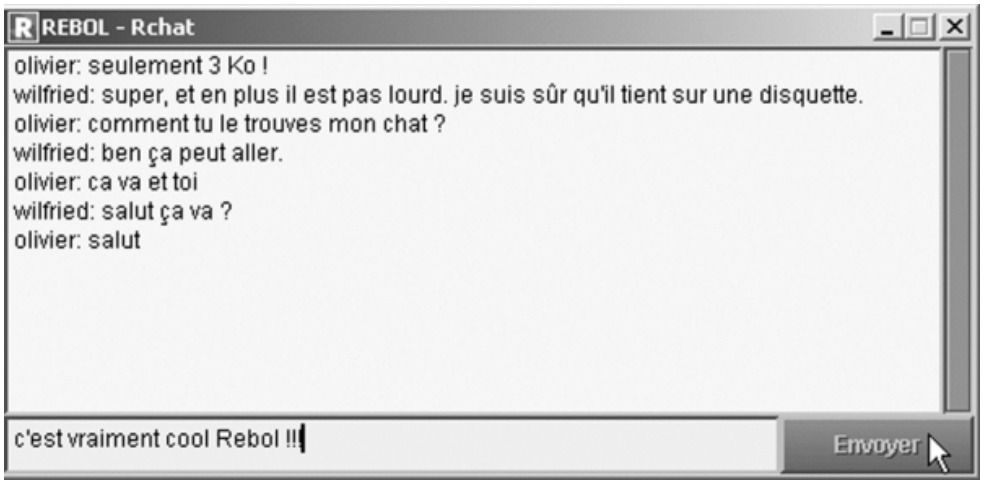

Рисунок 7-3. *Рчат в использовании.*

Когда сценарий начинает работать, переменная num-msg инициализируется путём выполнения запроса с действием "NUM" для определения номера последнего сообщения на сервере.

```
num-msg: trim/all read/custom server [ POST "action=NUM" ] num-msg: (to-
integer num-msg) + 1
```

11.2.3 Настройка сервера

На HTTP-сервере мы собираемся написать сценарий CGI с именем rchat.cgi и сохранить его в каталоге cgi-bin, откуда он будет запускаться.

Этот скрипт генерирует страницу с mime-типом text/plain для возврата клиенту. Его первая задача - прочитать информацию, полученную с помощью метода POST, а затем извлечь данные с помощью слова decode-cgi.

```
#!/usr/bin/rebol -cs
print "Content-Type: text/plain^/"

data: copy ""
len: to-integer system/options/cgi/content-length

until [
    buffer: copy ""
    read-io system/ports/input buffer (to-integer system/options/cgi/content-
length)
    append data buffer
    ((length? data) = len)
]

query: make object! decode-cgi data
```

Для постоянного хранения данных CGI использует два файла с именами num.txt и msg.txt. Первый содержит номер последнего полученного сообщения. Второй хранит сообщения в формате номер [login сообщение]. Чтобы избежать проблем, сценарий CGI сначала проверяет, существует ли файл msg.txt, пытаясь прочитать его. Если возникает проблема, создаются эти два файла.

```
if error? try [
    read %msg.txt
] [
    save %msg.txt [ ]
    write %num.txt 0
]
```

Все, что остаётся, - это реагировать на действия пользователя с помощью switch структуры. Если query/action (запрос/действие) содержат "NUM", сценарий CGI возвращает значение, содержащееся в файле num.txt.

В случае "SEND" сценарий CGI форматирует сообщение, сохраняет его в файле msg.txt и увеличивает значение, хранящееся в num.txt. Наконец, если действие - "READ", сценарий CGI генерирует блок, содержащий номер последнего сообщения, чтобы клиент мог обновить свою переменную num-msg и серию блоков, каждый из которых соответствует сообщению. Благодаря логину скрипт возвращает клиенту только сообщения, отправленные другими пользователями.

```
switch query/action [
    "NUM" [
        print read %num.txt
    ]
    "SEND" [
        num-msg: (make integer! read %num.txt) + 1
        write %num.txt num-msg
        bloc: load %msg.txt
        append bloc num-msg
        append/only bloc reduce [ query/login query/message ]
        save %msg.txt bloc
    ]
    "READ" [
        if error? try [
            msglist: find (load %msg.txt) (do query/num)
            msg: [ ]
            forskip msglist 2 [
                if (make string! msglist/2/1) <>
                query/login [
                    append msg msglist/2
                ]
            ]
            either all [ (not none? msg) ((length? msg) > 0) ] [
                print mold reduce [ (to-integer trim/all  read %num.txt) msg
]
            ] [ print "" ]
        ] [ ]
    ]
]
```

Мы разработали полнофункциональный клиент и сервер чата с объёмом кода менее 4 КБ, которые могут работать непосредственно через Интернет. Этот код может стать основой для многих других проектов. Вы можете улучшить его, чтобы управлять одновременным доступом к файлам данных и добавлять функции (смайлики, разные цвета сообщений в списке, исключение двух одинаковых имён и т.д.). Его также можно использовать как основу для онлайн-игр, написанных на Rebol.

Фактически, взаимодействие View со сценариями CGI настолько простое и интуитивно понятное, что вы можете использовать этот подход во многих ситуациях и создавать реальные приложения с использованием распределённой архитектуры.

11.3 Консоль администрирования MySQL

В предыдущей главе мы исследовали использование основных функций превосходной библиотеки MySQL Ненада Ракочевича.

Напоминаем, что он позволяет бесплатным версиям Rebol (Core и View) выполнять SQL-запросы к базе данных MySQL. Теперь мы реализуем более сложные функции для удалённого администрирования удалённой базы данных, чтобы создать небольшую консоль управления с Rebol/View.

11.3.1 Спецификация

Наша административная консоль - это инструмент, который позволяет нам подключаться к серверу MySQL для визуализации его баз данных. Для каждого из них мы должны уметь понимать таблицы и выполнять SQL-запросы. Как и любой хороший администратор, мы также хотим собирать статистику использования СУБД, иметь возможность останавливать СУБД и, наконец, проверять соединение между MySQL и клиентом. Мы выполним эти амбициозные спецификации менее чем в 3 КБ кода.

11.3.2 Идентификация пользователя

Чтобы защитить данные, которые он размещает, MySQL определяет права пользователей с помощью учётных записей и паролей. Поэтому наша административная консоль должна начать свою работу с идентификации пользователя и выбора сервера MySQL. Эта информация хранится в словах login, password и server имеющих текстовый тип. После их ввода кнопка "Submit" извлекает список баз данных, доступных на сервере, и отображает главное окно консоли.

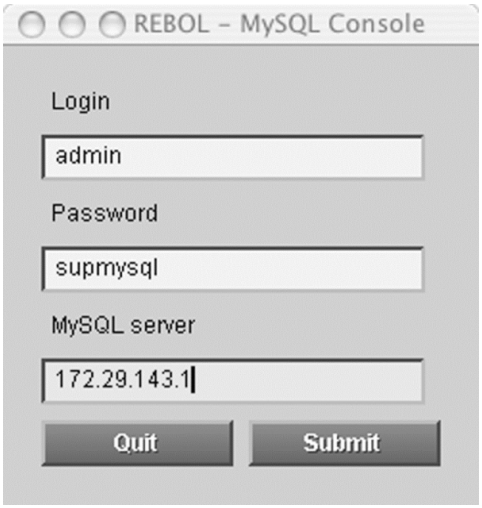

Рисунок 7-4. *Идентификация пользователя*

11.3.3 Создание главного экрана

Наша административная консоль состоит из одного окна, которое называется main (главным). Он имеет три кнопки, функции которых заключаются в отправке команд MySQL (статистика, состояние соединения и остановка СУБД). Четвёртая кнопка позволяет пользователю выйти из приложения.

Два списка (bases и tables), расположенные с двумя вкладками с помощью инструкции tabs, показывают базы данных, управляемые сервером MySQL, к которому мы подключены, и выбранную базу данных. Поле ввода reqsql позволяет вводить и выполнять команду SQL, щёлкнув предоставленную кнопку "Enter".

```
main: layout/size [
    across
    button "Statistics" [ send-command 'statistics ]
    button "Connection ?" [ send-command 'ping ]
    button "Stop MySQL" [ send-command 'shutdown ]
    button "Quit" [ quit ]
    return
    tabs [ 10 250 ]
    text "Bases"
    tab text "Tables" return
    bases: text-list 220x100 [
           load-tables bases/picked
    ]
    tab tables: text-list 220x100 [
        reqsql/text: join "select * from " (first tables/picked)
        show reqsql
    ]
    return
    text "SQL request:"
    across
    reqsql: field 250 ""
    button "Enter" [ user-request ]
] 500x230
```

Рисунок 7-5. *База данных musicdb содержит таблицу дисков.*

11.3.4 Некоторые служебные функции

Чтобы упростить задачу, мы собираемся определить три служебные функции. Во-первых, нам нужно правильно расположить ползунок двух списков главного окна при обновлении их содержимого. Для этого мы должны определить функцию fix-slider, которая принимает в качестве параметра имя списка, содержимое которого изменилось.

```
fix-slider: func [ faces ] [
    foreach list to-block faces [
            either 0 = length? list/data [
                   list/sld/redrag 1
            ] [
                   list/sld/redrag list/lc / length? list/data
            ]
    ]
]
```

Наше приложение должно часто подключаться к серверу MySQL. Таким образом, нам, очевидно, нужна функция, которая автоматически генерирует URL-соединение из учётной записи пользователя, пароля, имени сервера и базы данных MySQL, к которой будет осуществляться доступ. Эта функция, construct-url, принимает имя базы данных в качестве параметра.

```
construct-url: func [ base ] [
    return join mysql:// [ login/text ":" password/text "@" server/text "/"
base
    ]
]
```

Наконец, было бы очень полезно иметь функцию, роль которой заключается в выполнении SQL-запроса. Функция do-request принимает запрос SQL и имя базы данных, в которую он должен быть перенаправлен. Он подключается к базе данных после вызова construct-url. Данные, собранные из базы данных, возвращаются вызывающему абоненту.

```
do-request: function [ base req ] [ p data ele ] [
        db: open construct-url base
        insert db req
        data: copy db
        close db
        return data
]
```

11.3.5 Функции обработки

Начнём с отправки команд серверу MySQL. Библиотека Nenad предоставляет набор функций, которые поддерживают управление удалённым сервером MySQL. Эти команды вставляются непосредственно в порт связи. Они имеют форму блока, первым элементом которого является команда. За ним следуют любые необязательные аргументы. Вы можете изменить активную базу данных (init-db), переключить пользователя (change-user), создать новую базу данных (create-db), уничтожить базу данных (drop-db), обновить параметры сервера MySQL (reload), завершить процесс (process-kill) и переведите сервер в режим "отладки" с помощью команды debug.

Для нашей консоли мы будем использовать команды statistics, ping и shutdown. Они передаются как параметры в форме символа функции send-command.

Он генерирует блок, содержащий предоставленные параметры, если используется уточнение /options. Результат команды отображается в окне предупреждения (alert).

```
send-command: function [ cmd /options opts ] [ db bloc ] [
    if error? try [
        db: open construct-url "mysql"
        bloc: copy []
        append bloc reduce cmd
        if options [ append bloc reduce opts ]
        res: insert db bloc
        close db
        if not none? res [ alert to-string res ]
    ] [
        alert "It was not possible to carry out the command"
    ]
]
```

Функции load-base и load-tables используются для заполнения списков баз и таблиц основного

макета. Имена баз данных можно получить, просто сделав запрос SQL к таблице db базы данных MySQL. В отличие от получения списка таблиц в базе данных, которое может быть выполнено только с помощью команды show tables после использования init-db для выбора активной базы данных.

```
load-bases: does [
    clear bases/lines
    bases/lines: copy []
    foreach ele do-request "mysql" "select db from db" [
        append bases/lines ele
    ]
    fix-slider bases
    show bases
]

load-tables: func [ base ] [
    send-command/options 'init-db base
    tables/lines: copy []
    foreach ele do-request base "show tables" [
        append tables/lines ele
    ]
    fix-slider tables
    show tables
]
```

Выполнение SQL-запроса введённого пользователем выполняется функцией user-request.

Он извлекает значение поля ввода reqsql и отображает строки, прочитанные из базы данных. Использование mold сохраняет форматирование данных на экране в виде блоков.

```
user-request: does [
   either not empty? bases/picked [
          send-command/options 'init-db (first bases/picked)
          print [ "^/>> " reqsql/text ]
          foreach ele do-request (first bases/picked)
 reqsql/text [
                  print mold ele
          ]
   ] [ alert "You must select a database" ]
]
```

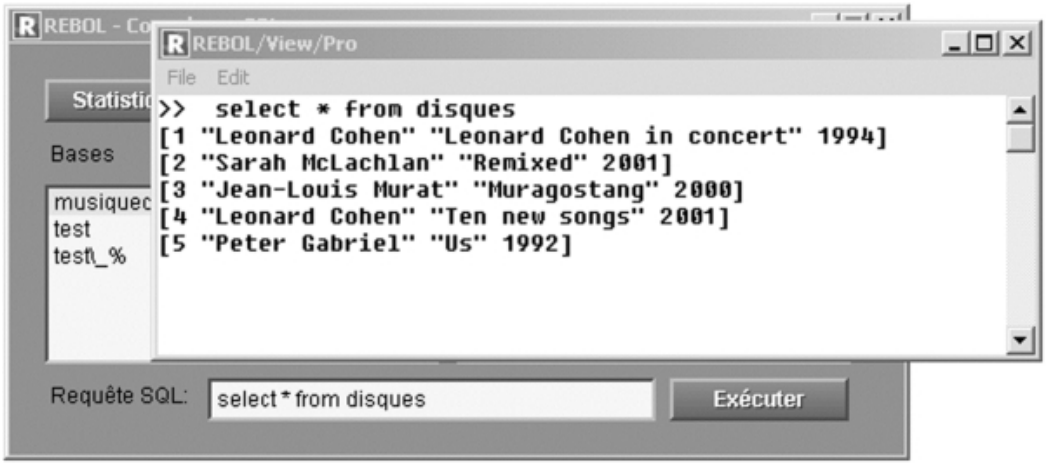

Рисунок 7-6. *Результат запроса SQL отображается в консоли.*

Менее чем за 3 килобайта кода вы разработали визуальную программу для управления сервером MySQL.

Как вы заметили, с Rebol очень легко создать настоящий административный инструмент MySQL.

11.4 Пишем реблет под IOS

Интересный принцип, лежащий в основе IOS, заключается в том, что он не статичен и не заморожен. При минимальных вложениях вполне возможно написать независимые приложения, использующие возможности этого сервера приложений. IOS предоставляет богатый и мощный API для облегчения написания реблетов.

11.4.1 Спецификация

Ваш первый проект разработки Rebol/IOS - это телефонный справочник, который позволяет вашим пользователям IOS обмениваться информацией о своих контактах. Эта информация будет не только храниться на сервере, но и синхронизироваться с каждым из клиентов Rebol/Link. Мобильный пользователь может восстановить контакты, добавленные к серверу с момента их последнего подключения, и даже может просматривать информацию, если они не подключены к серверу IOS. Ваш reblet будет безопасно передавать свои данные по сети, потому что вся информация автоматически шифруется IOS.

11.4.2 Цикл разработки

Для написания реблетов вам понадобится только клиент Link и соответствующий редактор кода. С другой стороны, ваш профиль пользователя IOS должен позволять вам публиковать новые приложения. Это делается с помощью опции new-app в вашей учётной записи.

Сначала создайте каталог на жёстком диске для хранения кода проекта и необходимых ресурсов.

Теперь давайте вернёмся на мгновение назад, чтобы рассмотреть, что такое реблет и как они работают.

Фактически, реблет - это просто сценарий Rebol, загружаемый по сети и выполняемый на клиенте. IOS вводит дополнительную концепцию связи с сервером приложений. Когда реблет выполняется в среде, установленной Link, он может попросить сервер IOS выполнить ряд операций, таких как чтение или удаление файла. Реблеты также могут требовать от IOS выполнения кода на сервере. Это делается с помощью специального метода под названием POST.

Все такие коммуникации выполняются с использованием асинхронной модели, позволяющей клиентскому коду продолжить выполнение, не дожидаясь ответа от сервера.

Чтобы опубликовать реблеты, вам нужно только разработать сценарий установки, который создаст среду для их размещения на сервере (наборе файлов), сохранит клиентский код на сервере и, возможно, создаст метод IOS POST. Очень важно помнить, что вам ни в коем случае не нужно работать на сервере. Все эти операции можно выполнять с помощью клиента Link.

11.4.3 Определение набора файлов

Первый этап состоит из написания сценария с именем install-phonebook.r, задача которого - создать набор файлов, называемый телефонной книгой (phonebook), для нашего приложения и установить метод POST на сервере. Блок заголовка REBOL сценария должен содержать тип 'link-app, чтобы указать Link, что приложение использует функции сервера IOS.

Набор файлов отправляется на сервер с помощью install-fileset. Это даёт права редактирования только пользователям "olivier" и "admin". С другой стороны, всем пользователям сервера предоставляется право использовать метод POST для публикации своих данных.

Тег icons указывает имя приложения и значок, который Link будет отображаться на рабочем столе. files содержит список файлов, из которых состоит клиентское приложение.

```
fileset: 'phonebook
tags: [
    access: [
        properties: rights: delete: change: [
            "olivier" "admin"
        ]
        post: 'all
    ]
    icons: [
        [
            name: "Phonebook"
            item: %apps/phonebook/phonebook.r
            folder: %apps/
            image: %desktop/icons/demo.gif
            info: "Your phone book."
        ]
    ]
]
files: [ [%apps/phonebook/phonebook.r %phonebook.r] ]
```

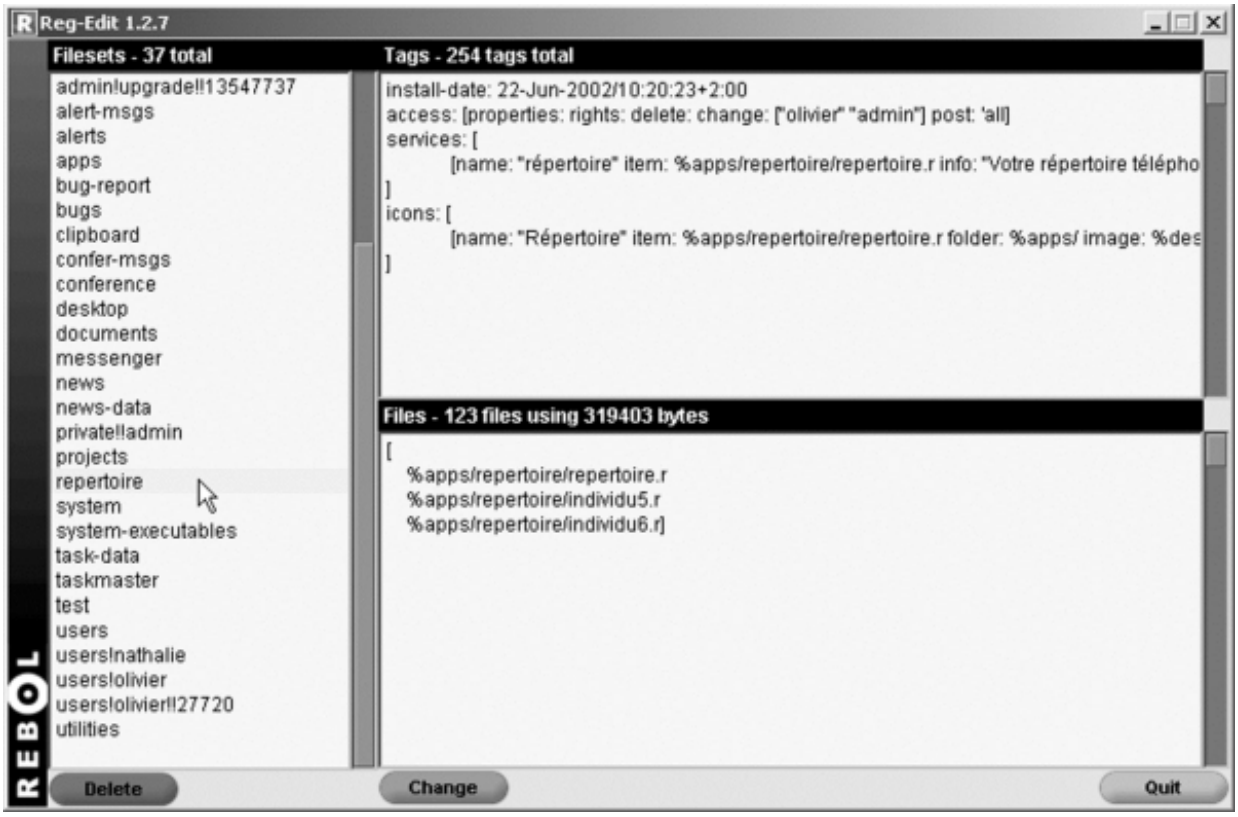

Рисунок 7-7. *Набор файлов приложения "Телефонная книга".*

11.4.4 Метод POST

Блок post-func содержит код метода POST. Локальные переменные, используемые последним, передаются серверу с помощью списка переменных post-locals. Лучше создать отдельный контекст для этого кода, чтобы сохранить словарь сервера и защитить код от конфликтов имён.

```
post-locals: [ base file num ]
post-func: [
    if error? err: try [
        server-code: context [
            num: 0
            base: %apps/phonebook/
            if data-exists? base/num.r [
                num: load-data base/num.r
            ]
            write-data base/num.r mold num + 1
            file: join "individual" [ num ".r" ]
            add-file 'phonebook base/:file (compress message/1)
            true
        ]
    ] [
        print mold/only disarm :err
        false
    ]
]
```

Роль этого фрагмента кода заключается в получении данных от клиента Link и создании файла в наборе файлов сервера. Каждая запись в телефонной книге будет сохранена в файле IndividualX.r (x - это число, увеличивающееся на 1 для каждой новой записи). На сервере этим методом создаётся файл с именем num.r, содержащий номер последнего использованного числа. Вы, наверное, заметили наличие некоторых новых слов, которые являются частью API сервера IOS:

- data-exists? проверяет наличие файла данных,
- write-data создаёт файл данных, хранящийся только на сервере,
- add-file добавляет файл в набор файлов приложения (этот новый файл будет автоматически синхронизируется при подключении клиентов к серверу).

11.4.5 Клиентский код

Теперь вы можете приступить к написанию кода клиента, который будет сохранен в файле с именем phonebook.r. Ещё раз, блок заголовка Rebol должен содержать тип 'link-app, чтобы идентифицировать себя как реблет IOS. Слово connected? позволяет узнать, подключён ли клиент к серверу или нет. В противном случае функции публикации отключаются, и пользователь может видеть только данные, ранее синхронизированные с помощью Link.
Переменная user-name содержит логин пользователя Link. Эта информация - одно из многих свойств user-prefs.
Пользовательский интерфейс приложения состоит из простого окна с раскрывающимся списком cts (ползунок устанавливается с помощью функции fix-slider), трёх полей ввода текста (name, first-name и tel) и трёх кнопок, одна для очистки формы, другая для добавления новой записи и последний для выхода из приложения. Link представил некоторые новые стили для VID, такие как закруглённые кнопки (btn).

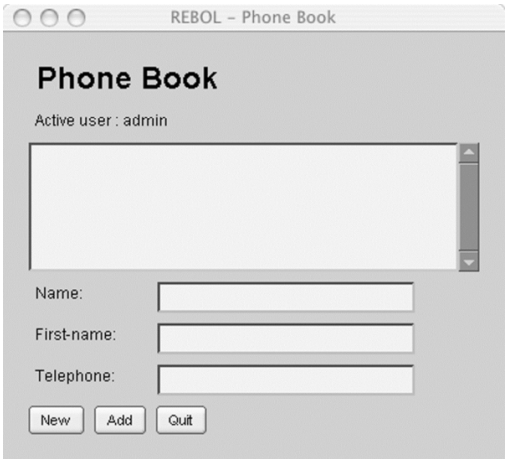

Рисунок 7-8. *Клиентская часть реблеа телефонной книги.*

Использование уточнения new для view откладывает обработку событий. После отображения макета код продолжает выполнение. Это позволяет нам запрашивать у Link информацию, которую он восстановил при подключении к серверу. Для этого мы объявляем обработчик событий для команды get с помощью слова insert-notify. Третий параметр (get-files) соответствует имени функции обратного вызова, вызываемой, когда клиент получает результат запроса. get используется для получения списка файлов в данном наборе файлов. Слово send-link отправляет команду get. Когда данные становятся доступными, вызывается функция get-files. Она деактивирует обработчик событий (remove-notify) и обновляет список, используя синхронизированные файлы, которые были сохранены на клиенте.

```
get-files: function [ event data ] [ list ] [
    remove-notify 'get 'phonebook
    list: copy []
    foreach individual data [
        if found? find individual "ind" [
            append list (read join link-root individual)
        ]
    ]
    append cts/lines (sort list)
    show cts
]
insert-notify 'get none :get-files
send-link 'get 'app-files 'phonebook
```

Как только это будет сделано, обработка событий, которая была ранее отложена, запускается словом do-events.

Новая запись создаётся путём вызова метода POST на сервере с помощью команды send-server.

```
send-server post reduce [ 'phonebook new-cts ]
```

Аргументы - это набор файлов, который будет использоваться, и данные, которые будут передаваться. На сервере метод POST получает его, используя список сообщений.

Рисунок 7-9. *Реблет телефонной книги доступен на рабочем столе Rebol/Link.*

Вы только что закончили писать свой первый реблет. Осталось только опубликовать свою работу на сервере. В Link используйте комбинацию клавиш CTRL + L и выберите файл с именем install-phonebook.r. Код выполнен, и ваш реблет теперь доступен вашим пользователям.

11.5 Резюме

Опять же, эти темы показали универсальность Rebol. Вы разработали движок Raycasting, написали полностью функциональный клиент чата в несколько строк кода, настроили менеджер MySQL и придумали реблет для Rebol/IOS.