Руководство программиста

ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК RSL

R-Style Softlab

Copyright © 2005 – 2011 R-Style Softlab

Компания "R-Style Softlab" не дает гарантий относительно содержания или использования настоящего Руководства, в частности, гарантий его коммерческих преимуществ или пригодности для конкретных целей. Компания оставляет за собой право перерабатывать настоящее издание, вносить в него изменения и дополнения, не уведомляя об этом частных лиц и организации.

Программное обеспечение, описанное в настоящем документе, поставляется строго по лицензионному соглашению. Авторские права компании "R-Style Softlab" защищены законом. Копирование и распространение программного обеспечения и документации к нему в какой бы то ни было форме и любыми средствами, включая фотокопирование и запись на магнитные носители, в отсутствие специального соглашения является противозаконным.

RS-Bank V.6 – зарегистрированная торговая марка R-Style.

© 2005 – 2011 R-Style Softlab
Все права защищены.

Ответственный за подготовку
Руководства пользователя
Петринская Л.В.

Дата редакции: 8 сентября 2011 г.

Оглавление

Элементы языка 9

Служебные слова 9
Имена 9
Область видимости имен 10
Комментарии 10
Объекты языка 11
Типы данных 11
Переменные с декларацией типа данных 12
Переменные без декларации типа данных 13
Символические константы и специальные значения переменных 13
Скалярные типы 14
Особенности реализации типов 15
Объектные типы 15
Константы 16
Выражения 18
Синтаксис 18
Семантика 19

Структура программы 23

Загрузка и кэширование модулей программы исполняющей системой RSL 27

Конструкции языка RSL 29

Пустая инструкция 29
Инструкция "выражение" 29
Условная инструкция IF 29
Инструкция цикла WHILE 30
Инструкция цикла FOR 30
Инструкции прерывания BREAK и продолжения CONTINUE 32
Инструкция возврата RETURN 32
Инструкция вывода 33
Формат управляющей строки 33
Определение переменных VAR 34
Определение классов и объектов 35
Определение символических констант CONST 39
Определение процедуры MACRO 40
Процедуры языка RSL 40
Передача параметров 41
Определение массивов 43
Определение массивов с помощью конструкции ARRAY 43
Стандартный класс TArray 44
Поддержка технологии ActiveX в RSL 46
Использование специальных значений 48
Соответствие типов данных языка RSL и ActiveX 48
Поддержка стандартных коллекций 49
Обращение к свойствам объектов с параметрами 50

Доступ к объектам Object RSL из других языков 51
Обработка событий 52
Класс TRslEvHandler 52
Свойства класса TRslEvHandler 52
Методы класса TRslEvHandler 54
Класс RslTimer 58
Методы класса *RslTimer* 58
Передача параметров 60
Поддержка модулей платформы .NET 61
Обработка ошибок, возникающих во время выполнения программы 63
Автоматическое создание объектов Object RSL 65
Конструкция WITH 65

Организация ввода/вывода 67

Спецификаторы форматирования 67
Формирование отчетов с использованием шаблонов 69
Стандартный класс TRepForm 70
Стандартный класс TPattFieldR 73
Свойства класса 73
Поддержка интерактивного режима 73
Меню 73
Вертикальное меню 74
Горизонтальное меню 74
Диалоговые окна 75
Процедура обработки сообщений 76
Список обрабатываемых сообщений 77
Список обрабатываемых сообщений процедуры RunDialog 77
Список обрабатываемых сообщений процедуры RunScroll 78
Значение, возвращаемое процедурой обработки сообщений 79
Скроллинг 83
Создание источников данных с помощью класса ToolsDataAdapter 88

Работа с файлами и таблицами баз данных 97

Использование стандартного класса Tbfile 98
Методы класса Tbfile 99
Свойства класса Tbfile 101
Использование стандартного класса TRecHandler 102
Использование конструкций FILE и RECORD 105
Работа с текстовыми файлами 108
Работа с файлами в формате DBF 110
Работа с записями таблиц как с записями переменной длины 111
Доступ к источникам данных с помощью библиотеки RSD 112
Описание библиотеки RSD 112
Класс RsdEnvironment 113
Свойства класса 114
Методы класса 114
Класс RsdConnection 114
Свойства класса 115
Методы класса 115
Класс RsdCommand 116
Свойства класса 117
Методы класса 118
Класс RsdRecordset 119
Свойства класса 120
Методы класса 121

Класс RsdError....122
Класс RsdField....122
Свойства класса....123
Методы класса....123
Класс RsdParameter....123
Использование RSD в программах на RSL....123
Обработка ошибок....125
Обработка транзакций....128

Работа с текстовыми и двоичными файлами....131

Использование класса TStream....131
Свойства класса....131
Методы класса....132
Использование класса TStreamDoc....134
Свойства класса....135
Методы класса....135

Управление файлами и каталогами....139

Использование класса TDirList....140
Методы класса TDirList....140
Свойства класса TDirList....143

Использование "домашних" каталогов пользователей....145

Использование RSCOM-объектов из программ на языке RSL..147

Описание стандартного модуля RCW....148
Процедуры....148
Класс TRslChanel....151
Методы класса TRslChanel....152
Свойства класса TRslChanel....152
Класс TRcwSite....153
Обработка ошибок RSCOM в языке RSL....153
Обработка сообщений от RSCOM-объектов....154
Стандартные RSCOM-серверы....156
RSCOM-сервер rsax.d32....156
Методы класса TRsAxServer....157
RSCOM-сервер rcwhost.d32....158
Использование класса TRcwHost....159
Свойства класса TRcwHost....159
Методы класса TRcwHost....160
RSCOM-сервер rsclr.d32....162
Методы класса TClrHost....162

Встроенные процедуры....165

Стандартные процедуры ввода данных с клавиатуры....165
Стандартные процедуры вывода....167
Преобразование типов значений переменных....171
Процедуры для работы с типом DoubleL....177
Работа со строками....177
Параметры процедур....180
Математические процедуры....181

Внешние программы ... 182
Удаленный запуск макропрограмм ... 183
Обработка меню ... 184
Обработка диалоговых окон ... 186
Обработка скроллинга ... 190
Макропроцедуры модуля RslScr ... 190
Переменные ... 190
Макропроцедуры ... 190
Макропроцедуры модуля rslx ... 192
Контроль ввода записи в скроллинг ... 197
Работа с отладчиком RSL ... 199
Файлы и структуры ... 200
Управление файлами и каталогами ... 215
Классы и объекты ... 218
Индикаторы выполнения процессов ... 223
Другие процедуры ... 224

Средство разработки расширений для языка RSL (DLM SDK) . 239

Создание и использование DLM-модулей ... 239
Передача параметров и возврат значений ... 241

Сводка синтаксиса RSL ... 243

Алфавитный указатель ... 247

Введение

Настоящее Руководство содержит описание языка RSL, являющегося неотъемлемой частью систем, разработанных программистами компании R-Style Softlab, при помощи которого пользователь имеет возможность создавать собственные программы. Изучение этого Руководства предполагает знание языков программирования высокого уровня.

Наличие хорошо структурированного языка, тесно связанного с базой данных, значительно расширяет возможности системы. Язык может быть использован для создания силами работников банка специфических отчетов, отсутствующих в системе, или других процедур, реализующих дополнительные процедуры.

Язык интерпретатора RSL – это язык высокого уровня. Он достаточно прост в изучении и применении и одинаково хорош как для квалифицированного, так и для неквалифицированного пользователей системы, так как имеет широкий спектр функциональных возможностей. В отличие от других подобных ему языков, язык интерпретатора интегрирован с системами, разработанными программистами компании R-Style Softlab. Он вызывается непосредственно из меню системы, обеспечивает доступ к базе данных, использует одинаковые с ней форматы данных, работает с экранными формами.

RSL – современный язык, разработанный с учетом нынешних требований к программному обеспечению. В компании R-Style Softlab ведется постоянная работа по его развитию и совершенствованию.

Элементы языка

Служебные слова

Служебное, или зарезервированное, слово – это имя, с которым в языке жестко сопоставлены определенные смысловые значения, и которое не может быть использовано ни для каких других целей. Ниже приведен список служебных слов языка RSL (они записываются в любом регистре).

and	**if**	**record**
array	**import**	**return**
const	**macro**	**this**
class	**not**	**true**
elif	**NULL**	**var**
end	**onerror**	**with**
false	**or**	**while**
file	**local**	**private**

Кроме этого, существует отдельный список ключевых слов, действующих в пределах определений FILE и RECORD.

btr	**key**	**txt**
dbf	**mem**	**sort**
dialog	**write**	

Значение каждого служебного слова поясняется при описании конструкций языка.

Имена

Под ***именем*** (***идентификатором***) в программе понимается не являющаяся служебным словом последовательность букв (латинских и русских) и цифр, начинающаяся с буквы; символ "_" (подчеркивание) рассматривается как буква. Длина идентификатора в RSL не должна превышать 80 символов. Прописные и строчные буквы не рассматриваются как различные символы, то есть регистр игнорируется. Имена используются для идентификации объектов RSL.

Кроме обычных идентификаторов, в RSL применяются имена специальных переменных, которые могут содержать любые символы, включая специальные, без ограничений. Такие имена должны быть заключены в фигурные скобки "{}", которые также входят в имя переменной.

Специальные переменные являются глобальными и используются обычно для передачи значений из вызывающих модулей, написанных на языках C или C++, в программу на языке RSL. Для обозначения полей в структурах и файлах, описанных в словаре базы данных, используются составные имена. Составные имена состоят из имени, идентифицирующего файл или структуру, после которого следует точка и название поля либо индекс поля в круглых скобках.

Пример:

Счет	*- простое имя*
Сумма010	*- простое имя*
{Debet account 52.5}	*- имя специальной переменной*
Клиенты.Name	*- составное имя, ссылающееся на поле Name в структуре Клиенты*
Клиенты(10)	*- составное имя, ссылающееся на десятое поле в структуре Клиенты*

Если имя переменной совпадает с именем переменной среды, оно инициализируется ее значением.

Пример:

PATH	*- переменная будет проинициализирована значением системной переменной PATH, содержащей список каталогов для поиска программ*

Присвоение значений этим переменным не изменяет содержимого системных переменных.

Область видимости имен

Глобальными именами считаются те, которые объявлены вне любой процедуры RSL. Имена специальных переменных, независимо от места их появления, считаются глобальными. Для размещения глобальных объектов используется специальная область памяти, через которую осуществляется доступ к их значениям из любого места программы RSL или из вызывающего внешнего модуля.

Локальными именами считаются те, которые объявлены в любой процедуре RSL. Локальные объекты создаются заново при каждом вызове процедуры, в которой были объявлены их имена, и доступны только в этой процедуре, а также во всех вложенных в нее процедурах RSL. Следует отметить, что **определенные явно** имена локальных объектов могут совпадать с именами глобальных объектов. В этом случае имя локального объекта перекрывает имя глобального, и доступ к глобальному объекту будет блокирован.

Каждая RSL-программа образует глобальную область видимости. Каждая процедура, содержащаяся в программе, образует свою, вложенную область видимости. Из вложенной области видимости доступны имена этой области и всех вышележащих. В каждой области видимости можно "перекрывать" имена из вышележащих областей, то есть во вложенной области могут быть определены переменные с такими же именами, как и в вышележащих.

При входе в область видимости переменные конструируются, при выходе разрушаются. Таким образом, время жизни переменной ограничено областью ее видимости.

Комментарии

Любой фрагмент текста, заключенный в скобки вида /* */, является комментарием. Язык RSL разрешает наличие вложенных комментариев:

Пример:

```
/*********************************************************
  Комментарий на языке RSL
 /*
     Вложенный комментарий
 */
 *********************************************************/
```

Данный пример содержит два комментария, один из которых вложен в другой.

Объекты языка

Объект RSL представляет собой совокупность информации, с которой оперирует язык. Объектами языка RSL являются:

- ***Переменная*** – объект, содержащий значение одного из типов данных, который может изменять величину и тип хранимого значения. Это происходит при присвоении переменной нового значения операцией присваивания (при этом старое значение теряется).

Пример:

```
aa=10                 /* aa хранит значение Integer */
aa="Привет"           /* aa хранит значение String */
```

- ***Объект типа FILE*** хранит информацию о таблице в базе данных (структуру таблицы и текущие значения полей).

  Примечание.

  Объект типа FILE является устаревшей конструкцией и поддерживается для совместимости с ранее разработанными приложениями. При работе с таблицами рекомендуется использовать конструкцию Tbfile.

- ***Объект типа RECORD*** хранит информацию о структуре файла или диалоговой панели. Объект типа RECORD, в отличие от объекта типа FILE, не имеет явной связи с файлом. Если это необходимо, то связь с файлом должна быть задана при помощи процедур RSL или неявно в вызывающем модуле, написанном на языке C или C++.
- ***Объект типа DBFFILE*** хранит информацию о файле формата DBF (структуру файла и текущие значения полей).
- ***Объект типа TXTFILE*** хранит информацию о текстовом файле (структуру файла и текущие значения полей).
- ***Объект типа ARRAY*** представляет собой вектор переменных, который можно индексировать.
- ***Объект типа CLASS RSL*** представляет собой совокупность свойств и методов и служит для создания экземпляра класса – объекта.

Все объекты RSL требуют явного определения их имени.

Типы данных

Тип данных RSL определяет функциональные возможности, предоставляемые этими данными, их область значений и средства RSL, предназначенные для работы с этими данными.

Данные, поддерживаемых в RSL типов хранятся в переменных языка RSL, а также в свойствах экземпляров RSL классов.

При присвоении значения переменной, если тип значения не соответствует типу, который может храниться в переменной, выполняется *приведение типов* данных по правилам RSL (см. стр. 19). Состав типов и правила их преобразования могут зависеть от конкретной версии RSL.

Все поддерживаемые типы данных разделяются на *скалярные* (см. стр. 14) и *объектные* (см. стр. 15).

Тип данных, хранящихся в переменной RSL, можно определить во время выполнения RSL-программы при помощи специальной процедуры ***ValType***. Эта процедура принимает в качестве параметра переменную RSL и возвращает целочисленный код типа данных, который в ней хранится. Код типа выражается символическими константами вида ***V_TypeName***.

По умолчанию переменные RSL могут содержать значения любых типов данных RSL.

Если структура описана во внутреннем словаре, то все поля этой структуры получают те типы, которые были заданы в словаре.

RSL не предполагает обязательную декларацию типов данных. В зависимости от конкретной задачи разработчик может явно декларировать тип переменной (см. стр. 12) либо не делать этого (см. стр. 13).

Переменные с декларацией типа данных

При декларации переменной можно явно указать, что переменная может содержать значение любого типа. Для этого используется специальное ключевое слово ***Variant***.

Кроме автоматического определения типа, в Object RSL имеется возможность явно декларировать типы:

- переменных;
- параметров макропроцедур и методов;
- возвращаемых значений макропроцедур и методов.

Рассмотрим правила декларирования типов для различных лексем языка RSL.

- Декларируемый тип *переменной или формального параметра* указывается через двоеточие после объявления. Кроме того, для декларирования типов данных можно использовать и имена классов:

Пример:

Определим переменной для хранения ссылки на объект тип "MyClass", который является именем класса:

Var obj: MyClass

Переменные, которые являются ссылками на файл или структуру, нельзя явно декларировать.

- Тип *возвращаемого значения макропроцедуры или метода класса* указывается после двоеточия, которое следует за списком формальных параметров, а в случае их отсутствия – непосредственно после имени макропроцедуры или метода класса:

Пример:

```
Macro MyStrLen (str: String): Integer
        Return strlen (str);
End;
```

Переменные без декларации типа данных

Декларация типа переменных в RSL необязательна. Декларация типа не приводит к повышению производительности работы RSL-программы и не уменьшает требований к системным ресурсам.

Любая переменная без декларации типа эквивалентна декларации с использованием ключевого слова ***Variant***. В этом случае переменная может содержать значение любого типа.

Пример:

```
Var p1;
Var p2:Variant;
p1 = 10;
p2 = “Test string”;
p2 = p1;
```

Символические константы и специальные значения переменных

Используя символические константы, переменным можно задать специальные значения:

- Если переменная не содержит значения ни одного из поддерживаемых типов данных, то считается, что переменная содержит специальное значение "Пусто". Процедура ***ValType*** для такой переменной возвращает код, соответствующий символической константе V_UNDEF.

  Для специального значения "Пусто" в RSL имеется символическая константа **Null**, которую можно присвоить любой переменной RSL, не зависимо от декларируемого типа переменной.

- Для задания специального значения "Нулевое значение" в RSL применяется символическая константа **NullVal**.

- Для задания специального значения "Умалчиваемое значение" в RSL применяется символическая константа **OptVal**.

Все перечисленные специальные значения служат для передачи в методы **ActiveX** объектов специальных значений COM-автоматизации (см. стр. 48).

Пример:

```
Var p:SpecVal;
P = NullVal;
```

В RSL имеются литеральные константы для задания значений логической величины ***True*** и ***False***.

Скалярные типы

Название типа	Ключевое слово для декларации	Код типа	Принимаемые значения
Целое число	**Integer**	V_INTEGER	От -2147483648 до 2147483647
Число с плавающей точкой	**Double**	V_DOUBLE	15 значащих десятичных цифр
Число с плавающей точкой длинное	**DoubleL**	V_DOUBLEL	15 значащих десятичных цифр (аналог **Double**)
Строка символов	**String**	V_STRING	
Логическая величина	**Bool**	V_BOOL	***True***, ***False***
Дата	**Date**	V_DATE	
Время	**Time**	V_TIME	
Дата и время	**DateTime**	V_DTTM	
Адрес памяти	**MemAddr**	V_MEMADDR	
Ссылка на процедуру RSL	**ProcRef**	V_PROC	
Ссылка на метод объекта RSL	**MethodRef**	V_R2M	
Двоично-десятичное число с 4-мя знаками после запятой	**Decimal**	V_DECIMAL	28 значащих цифр (аналог **Numeric**)
Двоично-десятичное число с плавающей точкой. Используется для вещественных величин (денежных сумм, курсов валют, процентных ставок и пр.)	**Numeric**	V_NUMERIC	28 значащих цифр
Тип для денежных величин	**Money**	V_MONEY	28 значащих цифр (аналог **Numeric**)
Длинный тип для денежных величин	**MoneyL**	V_MONEYL	28 значащих цифр (аналог **Numeric**)
Специальное значение "Нулевое значение", или "Умалчиваемое значение"	**SpecVal**	Специальной константы нет. Код типа: 26	***OptVal*** или ***NullVal***
Специальное значение "Пусто"	-----	V_UNDEF	***Null***

Особенности реализации типов

Некоторые из ранее используемых типов имеют синтаксические эквивалентны:

- лексемы типа **DoubleL** являются синонимом типа **Double**;
- лексемы **MoneyL** являются синонимом **Money**;
- лексемы **Decimal** являются синонимом **Numeric**.

Для внутреннего представления переменных типа **Money** используется тип **Numeric.**

Эти типы, используемые в ранних версиях RSL, оставлены только для совместимости исходного кода на RSL.

Объектные типы

Объектные типы данных подразделяются на специализированные объекты и обобщённые объекты (GenObject).

Обобщённые объекты создаются либо непосредственным вызовом конструктора RSL-класса, либо при помощи стандартной процедуры GenObject, принимающей имя RSL-класса в качестве строкового параметра. И конструктор, и процедура GenObect возвращают ссылку на созданный объект.

В настоящее время специализированные объекты являются устаревшими. В новом коде рекомендуется использовать обобщённые объекты, реализованные на замену специализированных объектов.

Название типа	Ключевое слово для декларации	Код типа	Тип аналогичного обобщённого объекта
Ссылка на специализированный объект "файл базы данных"	**BtFileRef**	V_FILE	TBfile
Ссылка на специализированный объект "структура в памяти"	**StrucRef**	V_STRUC	TRecHandler
Ссылка на специализированный объект "массив значений"	**ArrayRef**	V_ARRAY	TArray
Ссылка на специализированный объект "текстовый файл"	**TxtFileRef**	V_TXTFILE	Нет
Ссылка на специализированный объект "Файл dBase"	**DbfFileRef**	V_DBFFILE	Нет

Ссылка на обобщённый объект RSL класса	**Object** для любого объектного типа, или <**Имя RSL класс**>	V_GENOBJ	

Пример:

```
//Пример создания обобщённого объекта RSL-класса и переменных
для ссылки на него:
Class X
End;
Var p1:Object;
Var p2:X;
p1 = X;
p2 = p1;
```

Константы

Под константой понимается значение одного из типов данных.

Символьные константы имеют тип **String** и задаются в виде последовательности символов, заключенных в кавычки:

Примеры:

```
“a“
"F"
"абв_12"
“\n"
"\t"
```

Максимальная длина символьной константы – 2047 символов. Если задана более длинная строка, то генерируется ошибка RSL. Для ввода строки длиннее 2047 символов необходимо использовать операцию "+" для "склеивания" символьных констант во время выполнения RSL-программы. Получаемые таким образом динамические строки не имеют ограничения по длине.

Пример:

```
//пример получения строки длиной 24056 байт:
macro GetLongString
a = "1234567890 ";
a = a + a + a;
a = a + a + a;
a = a + a + a;
a = a + a + a;
a = a + a + a;
a = a + a + a;
a = a + a + a;
return a;
end;
str = GetLongString;
len = "\n\nДлина вставленной строки равна " + string (strlen (str));
```

Последовательности, начинающиеся с обратной косой черты, ничем не отличаются от таковых в языке C:

- **\n** – новая строка;
- **\r** – возврат каретки;
- **\t** – символ табуляции;
- **\f** – перевод формата;
- **\xHH**, \XHH – символ, заданный кодом (здесь HH – две шестнадцатеричные цифры);
- \\ – задает одиночный символ '\'.

Целочисленные константы типа **Integer** имеют диапазон допустимых значений от -2'147'483'648 до 2'147'483'648. Они задаются следующим образом:

2345, 1236 и т.п.

Числа с плавающей точкой имеют тип Double и могут находиться в диапазоне от 1.7e-308 до 1.7e308. При их записи используются точка и латинская буква "**e**", например:

4356.234, 345., .1234, 1231.2341e-23.

При записи денежных сумм в начале ставят знак доллара "$". Эти константы имеют тип Money и могут принимать максимальное значение 9'999'999'999'999.99.

Пример:

$146, $765.23, -$12.34.

Денежная константа, в которой не задана ни одна цифра, считается недействительной. При компиляции будет выдана ошибка: "Неверная денежная константа".

Для записи шестнадцатеричных констант используется знак "#".

Пример:

#F2;

#125ab2;

Логические константы типа **Bool** могут принимать только два значения: TRUE – истина, FALSE – ложь.

Язык RSL позволяет определить символические константы при помощи определения **CONST**. Ниже перечислены предопределенные символические константы:

TRUE	*- задает значение "истина" типа Bool*
FALSE	*- задает значение "ложь" типа Bool*
NULL	*- задает значение типа V_UNDEF*

Предопределенные константы для работы с диалоговыми окнами описаны в разделе, посвященном вводу/выводу (см. стр. 75).

Выражения

Синтаксис

Выражения представляют собой последовательность операндов, разделенных знаками операций.

Пример.

Name - *выражение, состоящее из одного операнда – простого имени;*
MgFun(10) - *выражение, состоящее из одного операнда – сложного имени;*
a+b - *сумма двух переменных;*
*b*10+c* - *произведение и сумма переменных.*

В приведенной ниже таблице операции располагаются в порядке возрастания их приоритета при вычислении выражений. Приоритет операций в каждой группе одинаков.

Таблица видов операций:

Обозначение	Операция
" = "	Присваивание
" == " " != " " < " " > " " <= " " >= "	Равно Не равно Меньше Больше Меньше или равно Больше или равно
" + " " – " " OR "	Сложение Вычитание Дизъюнкция
" * " " / " " AND "	Умножение Деление Конъюнкция
" – " " + "	Унарный минус Унарный плюс
" NOT "	Отрицание
" @ "	Получение ссылки на процедуру

Для того чтобы явно задать порядок вычисления, необходимо применять круглые скобки.

Пример:

a+B*c *эквивалентно* ***a+(b*c)***

Все операции, кроме операции присваивания, имеют левую ассоциативность. Это означает, что операции с одинаковым приоритетом выполняются слева направо. Например, выражение a+b+c интерпретируется как (a+b)+c. Для использования другого порядка вычисления необходимо использовать скобки.

Операция присваивания имеет правую ассоциативность. То есть выражение a = b = 10 интерпретируется как a = (b= 10).

Последовательность вычисления операндов в выражении не определена. Исключение составляют операции "AND" и "OR". Для них гарантируется строгий порядок вычисления операндов: сначала вычисляется значение левого операнда, и если после этого результат операции уже известен, вычисление правого операнда не производится.

Все перечисленные операции используются при работе с любыми переменными, за исключением переменных, хранящих ссылки на объекты RSL-классов, которые можно сравнивать друг с другом или с константой NULL с помощью операций == и !=.

С помощью операций == и != также можно сравнивать между собой объекты RSL-классов и целые числа, при этом операция == всегда возвращает значение FALSE, а операция != возвращает значение TRUE.

Примечание.
Операции сравнения объектов и целых чисел в RSL-программах предусмотрены в связи с необходимостью поддержки технологий работы с данными СУБД Oracle.

Семантика

Для типов данных в RSL предусмотрены операции преобразования из одного типа данных в другой. Когда в арифметической операции участвуют два операнда разных типов, для которых не определена явная процедура вычисления операции, применятся автоматическое преобразование типа одного из операндов к типу другого операнда.

Преобразуется тип операнда с меньшим весом типа к типу операнда с большим весом типа. Ниже представлены варианты преобразования типов в соответствии с их весами. Тип слева преобразуется в тип справа:

Integer -> Double -> DoubleL -> Numeric -> String

Явные преобразования выполняются при помощи специальных процедур: ***string***, ***int***, ***double***, ***money***, ***decimal***, ***numeric***.

Преобразование также происходит при присвоении значения переменной с явно декларированным типом данных.

Пример:

```
var a : String;
a = 10; /* a будет содержать значение строковое “10”.*/
```

При преобразовании значения типа **Money** в тип **Double** результат представляет собой число в рублях с копейками, отделенными точкой. При преобразовании значения типа **Double** или **Integer** в тип **Money** действует обратное правило: число с десятичной точкой преобразуется в соответствующее значение в копейках.

Пример:

```
var a : Double;
    b : Money;
a = $12.34;  /* a примет значение 12.34. */
b = a;       /* b примет значение 1234.  */
```

Существует возможность преобразования строкового типа данных ***String*** в типы данных ***Date*** или ***Time***.

Пример.

```
var d:Date;
    t:Time;
d = "31.12.2010"; /*d примет значение 31.12.2010*/
t = "23:59:59"; /*t примет значение 23:59:59*/
```

Логические операции AND, OR и NOT определены только для операндов типа **Bool**. Все арифметические операции определены для всех числовых типов данных (**Integer**, **Double, Money**). Тип результата бинарных операций будет таким же, как у операнда с большим приоритетом, за исключением особых случаев с операндами типа **Money**.

Операция "+" определена для операндов типа **String** как конкатенация (соединение строк):

Пример:

```
aa = "Stroka1" + "Stroka2";     /* Переменная aa имеет тип String и
                                   содержит строку "Stroka1Stroka2" */
```

Операции "+" и "-" определены для операндов типа **Date** и **Integer**. Результат операции имеет тип **Date**:

Пример:

```
Сегодня = date;                 /* Переменные в левой части имеют тип
Завтра  = Сегодня + 1;          Date */
Вчера   = Сегодня – 1;
```

Операция "-" определена для операндов типа **Date**. Результат имеет тип **Integer** и представляет собой количество дней между двумя датами:

Пример:

```
n = Завтра – Вчера;             /*Переменная n получает тип Integer и
                                  значение2*/
```

Операции "+" и "-" определены для операндов типа **Time**. Результат операции имеет тип **Time**:

Пример:

```
n = time(10,0) + time(0,15);    /*Переменная n получает тип Time и
                                  значение 10:15*/
```

Результатом операций "+" и "-" с операндами типа **Money** будет значение типа **Money**. Результатом операций "*" и "/" с операндами типа **Money** будет значение типа **Double**.

Пример:

```
Дебет = $100;
Кредит = $33;
Оборот = Дебет – Кредит;
Результат = Дебет/Кредит;
```
/ Оборот получит тип Money и значение $67. Результат получит тип Double и значение 3.3333(3) */*

Результатом операций "*" между операндами типа **Money** и **Integer** будет значение типа **Money**. Результатом операций "*" между операндами типа **Money** и **Double** будет значение типа **Double**.

Результатом операций "/" между операндами типа **Money** и **Integer** будет значение типа **Money**. Результатом операций "/" между операндами типа **Money** и **Double** будет значение типа **Double**.

Результатом операций "/" между операндами типа **Integer** будет значение типа **Integer**, равное целой части от деления:

Пример:

```
a = 5;
b = 3;
c = a/b; /* Переменная c получает тип Integer и значение 1*/
```

Операции сравнения определены для всех типов данных. Для данных типа **Bool** определены операции сравнения равно "==" и не равно "!=". Результатом операций сравнения является значение типа **Bool** – ложь или истина.

Для переменных с типами, соответствующими константам V_FREF, V_SREF, V_AREF, V_TREF, V_DREF и V_MEMADDR, определены операции сравнения с NULL.

Для переменных типа, соответствующего константе V_MEMADDR, предусмотрена операция сравнения друг с другом.

Структура программы

Программа на языке RSL представляет собой последовательность конструкций языка, разделенных знаком ";".

Конструкции языка – это либо определения объектов RSL, либо определения процедур, либо выполняемые инструкции. Программа, как и процедура, может заканчиваться словом END. В конструкции, которая располагается перед словом END, символ ";" может отсутствовать.

Выполняться RSL-программа начинает с первой встретившейся выполняемой инструкции, которая может находиться как в текущем файле, так и в файле, подключенном посредством директивы ***IMPORT***.

Вот пример самой простой программы на RSL:

```
[ Привет, Мир! ]
```

Этот пример выводит в стандартный выходной поток строку "Привет, Мир!".

Вот более сложный пример, в котором вычисляется самое длинное имя клиента, хранящееся в нашей базе данных:

```
file f (client);
var  n = 0,
     maxLen = 0,
     curLen,
     maxName = "";
macro Report
[ Самое длинное название клиента ####
  #                  ]
(maxLen:l,maxName)
end;
while (next (f))
   n = n + 1;
   message ("Обработано записей ",n);
   curLen = strlen (f.Name_Client);
 if (curLen > maxLen)
     maxLen = curLen;
     maxName = f.Name_Client
   end
end;
Report;
```

Конструкции языка, использованные в этом и последующих примерах, будут описаны в следующем разделе.

Текст программы должен находиться в текстовом файле с расширением "mac". Содержимое файла с таким расширением образует макромодуль.

Примечание.
Длина имени макрофайла не должна превышать 24 символа.

В принципе, всю RSL-программу, содержащуюся в модуле, можно рассматривать как процедуру инициализации, имя которой – это имя файла, в котором находится программа. Синтаксически единственным отличием программы от процедуры является отсутствие у программы заголовка MACRO. Заканчиваться программа может, как и процедура, словом END.

Пример:

```
[ Это пример программы ]
end
```

Все, что расположено после слова END, игнорируется интерпретатором.

В текст макропрограммы можно включать комментарии, которые задаются следующим образом:

//<текст комментария>

или

/<текст комментария>*/*

Макромодуль, или сокращенно модуль, может включать в себя определения переменных, объектов, процедур, а также выполняемые инструкции. Все инструкции модуля неявно образуют процедуру инициализации модуля.

Пример:

Для примера создадим макрофайл test.mac с таким содержимым:

```
a = 10
macro Mac1
  b = 20
end
class MyClass
  var Prop = 100;
  macro Method
    var aa = 200;
  end;
  var p = 10;   // Начало конструктора класса
  println ( p )
end;
            // Подразумевается, что с этого момента начинается
            //   процедура инициализации test макромодуля test.mac
println (a);
end;              // Конец процедуры инициализации
```

По умолчанию все переменные, процедуры и классы, объявленные в макромодуле, автоматически приобретают статус глобальных. Они становятся доступными не только внутри макромодуля, но и во всех остальных модулях.

Таким образом, в нашем примере переменная ***a*** и процедура ***Mac1*** являются глобальными. Инструкция ***println*** образует процедуру инициализации модуля.

Чтобы запретить извне доступ к переменной, процедуре модуля или классу, определенному в модуле, используется модификатор ***private***. Таким образом, переменная или процедура становится недоступна извне и видна только в рамках этого модуля.

Примечание.
*С точки зрения C++ этот модификатор можно рассматривать, как **protected**, либо как **private** с автоматическим объявлением всех классов наследников друзьями базового класса.*

Если ***private*** стоит перед определением свойства или метода класса, то это свойство или метод становятся недоступными извне класса.

Пример:

```
class MyClass
    var prop1 = 500;
    private var prop2 = 1000;
end;
obj = MyClass;
println ( obj.prop1 ); //Свойство доступно
println ( obj.prop2 ); //Свойство недоступно
```

Если этот модификатор используется при определении переменной, то для ее декларации обязательно должно присутствовать ключевое слово VAR (см. стр. 34).

Следует иметь в виду, что обращение через ***this*** (см. стр. 34) к свойствам и методом считается обращением извне **объекта**. Поэтому этот код приведёт к ошибке:

Пример:

```
class X
  private var  prop1;
  this.prop1 = 1;  // Ошибка
end;
```

В языке RSL используется также модификатор ***local***, который при установке перед объектом или переменной указывает, что этот объект RSL, переменная или процедура является локальным объектом процедуры инициализации или конструктора объекта.

Внимание!
Этот модификатор применим только для процедур инициализации модуля и конструктора класса!

Локальные переменные модуля доступны только локальным процедурам модуля. Невозможно обратиться к локальной переменной внутри любой другой процедуры модуля.

Пример:

```
local var str;    //Локальная переменная модуля, видима только
                  //в теле текущего модуля или в локальной процедуре
macro MyProc1()    //Глобальная процедура модуля
    str = "Hello!";    //Ошибка! str недоступна
end;
local macro MyMacro1()  //Локальная процедура модуля
    str = "Hello!";       //Правильно!
end;
MyMacro;
println ( str );
```

Если модификатор ***local*** стоит перед свойством класса, то по аналогии с модулем, оно перестает быть свойством класса как таковым и становится локальной переменной конструктора, то есть может быть доступно только для инструкций конструктора.

Пример:

```
class MyClass
        var prop1 = 200;
        local var 1Var = 300;
    macro Method1
        println ( 1Var );   //Локальная переменная из метода
недоступна
    end;
    local macro LocProc
        println ( 1Var );    //Локальная переменная доступна
    end;                     //из локальной процедуры класса
    LocProc;
end;
```

Процедуры RSL могут быть вызваны как явно при помощи инструкции вызова процедуры, так и неявно вызывающим модулем, написанным на языке C, C++. Примером неявно вызываемой процедуры является ***BtrError***, которая вызывается неявно при возникновении ошибок в процессе обращения к базе данных (конечно, если процедура с именем ***BtrError*** присутствует в тексте программы).

В тексте программы может находиться директива ***IMPORT***. Использование этой директивы позволяет включить в текст макромодуля информацию из других файлов, что дает возможность вынести часто используемые процедуры в отдельный файл и потом при необходимости подключать этот файл директивой ***IMPORT***.

```
IMPORT blnc, common;
```

Примечание.
Директива IMPORT должна находиться вне определения макропроцедур.

Директива ***IMPORT*** позволяет подключать:

- другие RSL-модули, при этом имя модуля должно указываться без расширения. К нему будет по умолчанию добавлено расширение *.MAC*;
- DLM-модули – модули, созданные с помощью инструмента DLM SDK (см. стр. 239), которые имеют расширения *.D32*, *.DLM*;
- встроенные в прикладную систему стандартные модули, которые написаны на языках C или C++.

Внимание!
Импортируемые файлы должны иметь разные названия. Нельзя импортировать файлы с одинаковым названием и разным расширением.

Имена модулей, подключаемых с помощью этой директивы, могут содержать русские буквы.

```
IMPORT  BankInter, Баланс;
```

Поиск указанных в директиве ***IMPORT*** файлов выполняется системой в следующей последовательности:

1. среди имен стандартных модулей, написанных на языке C/C++;
2. среди имен DLM-модулей;
3. среди имен текстовых файлов, имеющих расширение *.MAC*. В этом случае система сначала осуществляет поиск файла с прекомпилированным кодом, который имеет такое же имя и расширение *.RSM*, а затем ищет соответствующий ему макрофайл. Если для макрофайла существует актуальный файл с прекомпилированным кодом, то будет загружен соответствующий файл с расширением *.RSM*. В противном случае загрузится файл с расширением *.MAC*.

Загрузка и кэширование модулей программы исполняющей системой RSL

В процессе выполнения RSL-программы исполняющая система RSL создает в памяти так называемые поименованные экземпляры интерпретатора (ПЭИ). В них загружаются системные и пользовательские RSL-модули, используемые в программе.

ПЭИ с именем *default* создается всегда и для каждой программы. В прикладной системе могут также дополнительно определяться другие ПЭИ с произвольными именами.

ПЭИ состоит из постоянной части и отгружаемой. Постоянная часть ПЭИ создается при первом использовании программы, после чего она остается в кэше. При последующих запусках RSL-программ, использующих данный ПЭИ, постоянная часть берется из кэша, что существенно ускоряет выполнение RSL программы. Отгружаемая часть выгружается сразу после выполнения.

В постоянной части ПЭИ сохраняются системные модули, определяемые прикладной системой.

Чтобы пользовательские модули попали в постоянную часть всех существующих ПЭИ, их необходимо включить в модуль *autoinc.mac*. Если пользовательский модуль нужно включить в какой-либо определенный ПЭИ, его нужно включить в модуль, имя которого совпадает с именем ПЭИ, например, *default.mac*.

Пример:

```
/*Пример содержимого модуля autoinc.mac:  */
IMPORT a  /* В результате модуль a будет включен в */
          /* постоянную часть всех существующих ПЭИ */
```

Постоянная часть ПЭИ может быть отгружена исполняющей системой, например, в том случае, если изменился код одного из используемых модулей.

Конструкции языка RSL

RSL предоставляет программисту достаточно широкие возможности по управлению данными. Количество конструкций языка невелико, но вполне позволяет писать хорошо структурированные и эффективные программы. В качестве конструкций языка используются:

- выполняемые инструкции;
- определение объектов и процедур.

При описании конструкций языка используются следующие соглашения:

- синтаксические элементы, которые являются необязательными, заключаются в квадратные скобки **[**...**]**;
- синтаксические элементы, которые могут повторяться произвольное число раз или отсутствовать, заключаются в фигурные скобки **{**...**}**.

Пустая инструкция

Пустая инструкция состоит только из разделителя – точки с запятой. Она используется там, где по правилам языка могла бы находится какая-либо инструкция, а по логике программы там ничего выполнять не надо:

Пример:

```
if ( aa != 10 );    /* Пустая инструкция */
 else
    bb = aa
 end
```

Инструкция "выражение"

Эта инструкция представляет собой любое выражение RSL. Вычисленное значение выражения отбрасывается.

Примеры:

```
a;
a + b;
a + (b = 12);
a = 10;
d + f2 (12, c = b = 20);
a = b = c = d = 10;
```

Условная инструкция IF

Условная инструкция в RSL имеет следующий вид:

```
IF '('условное выражение')' список инструкций
{  ELIF '('условное выражение')' список инструкций }
[ELSE список инструкций  ] END
```

Количество конструкций ELIF неограничено.

Пример:

```
 if ( a < 10 )
    a = 10
elif ( a < 20 )
    a = 20
elif (  (a < 100) and (a > 50) )
   a = 0
else
   a = b;
   b = 0
end
```

Первым анализируется условное выражение после IF. Если оно истинно, выполняется список инструкций. В противном случае последовательно, сверху вниз анализируются условные выражения после ELIF, если они есть. Если результатом сравнения какого-либо из них является истина, выполняется список инструкций, который следует за данным условным выражением. Остальные условия при этом не анализируются.

Если в конструкции присутствует ELSE, то соответствующий список инструкций выполняется в том случае, если все условные выражения ложны.

Инструкция цикла WHILE

Для организации циклов в RSL используется инструкция WHILE.

Цикл WHILE имеет следующий синтаксис:

```
WHILE '('условное выражение')'
список инструкций
END
```

Список инструкций выполняется до тех пор, пока остается истинным условное выражение. Если условие ложно до входа в цикл, то список инструкций не выполняется ни разу.

Пример:

```
a = 0;
while ( a < 10 )
    a = a + 1;
    println (a)
end
```

Инструкция цикла FOR

Цикл FOR имеет следующий синтаксис:

```
FOR (loopVar, beginExpr, endExpr, deltaExpr)
    список инструкций
end
```

Инструкция может использоваться для выполнения трёх различных задач:

1. Для реализации бесконечного цикла. В этом случае не нужно задавать ни одного параметра цикла. Можно не указывать и скобки.

Пример:

```
For
  println (“Loop”)
end
```

2. Для реализации перебора элементов стандартных коллекций COM и RSL (т.е. коллекций, для которых можно создать объект-енумератор). В этом случае в качестве параметров инструкции FOR передаётся только переменная цикла **loopVar**, которая последовательно получает значения элементов коллекции и объект, для которого можно запросить объект-енумератор.

   Для перебора всех элементов массива RSL можно использовать один из приведённых ниже примеров. По своей результативности они идентичны, однако использование инструкции FOR в данном случае более наглядно и естественно.

Пример 1:

```
ar = TArray;
…..
ob = ar.createEnum;
while (ob.next)
  println (ob.item)
end;
```

Пример 2:

```
for (item, ar)
  println (item)
end;
```

3. Для выполнения цикла заданное количество раз. В этом случае в инструкции используются три параметра:

   - **beginExpr** – начальное значение счётчика цикла;
   - **endExpr** – конечное значение счетчика;
   - **deltaExpr** – величина, на которую изменяется счётчик цикла при каждой итерации.

   Значения этих параметров приводятся к типу **V_INTEGER**. Значение счётчика цикла присваивается переменной **loopVar**. Все параметры инструкции FOR в этом случае, кроме **loopVar**, являются необязательными:

   - Если не задан параметр **deltaExpr**, то он принимается равным "1" в случае, если значения **endExpr** больше чем или равно **beginExpr**, и "-1" – если **endExpr** меньше **beginExpr**.
   - Если параметр **endExpr** не задан, то в качестве конечного значения счетчика принимается значение, максимальное для типа V_INTEGER – "2147483647".
   - Если не задан параметр **beginExpr** , то вместо него принимается "0".

Пример:

```
for (val, 1, 100)
  println (val)
end;
```

Следует иметь в виду, что переменная цикла **loopVar** может быть объявлена непосредственно в цикле. Областью видимости объявленной таким образом переменной является не только вложенный блок, в котором она объявлена, а **вся** программа.

Пример:

```
for (var i : Numeric, 1, 10)
  var d : String = “80”;
  println (i);
end;
println (i);
```

Инструкции прерывания BREAK и продолжения CONTINUE

Инструкция BREAK прерывает выполнение цикла FOR или WHILE и передаёт управление инструкции, следующей за инструкцией цикла.

Инструкция CONTINUE предназначена для непосредственного перехода к следующей итерации цикла без выполнения инструкций следующих после инструкции continue.

Пример:

```
For
 if (testMacro > 0)
   break;
 end;
 if (otherTest < 0)
   continue
 end;
 DoWork;
end;
```

Если инструкция BREAK или CONTINUE вызвана вне инструкции цикла, то её действие эквивалентно инструкции return без параметров.

Инструкция возврата RETURN

Инструкция RETURN применяется для выхода из процедуры или завершения всей RSL-программы. Во втором случае она должна быть указана вне любой процедуры. Применяется следующая синтаксическая форма:

```
RETURN [выражение];
```

Выражение, заданное после слова RETURN, определяет возвращаемое процедурой значение. Если в инструкции завершения RSL-программы в качестве выражения задать текст, содержащийся в символьной константе или переменной типа String, он будет выведен на стандартное устройство вывода.

Инструкция вывода

Инструкция вывода позволяет определить в тексте программы форму для вывода в стандартный выходной поток. Выводимая форма заключается в квадратные скобки:

```
[  Этот текст без изменений будет выведен в стандартный
выходной
  поток ]
```

Внутри текста формы могут находиться поля, в которые необходимо выводить посчитанные значения из переменных языка RSL. Поля внутри формы задаются последовательностью символов ‘#’. Значения, которые необходимо вывести в поля, указываются сразу после формы в круглых скобках:

```
[ Номер счета  ##########
     Остаток   ########## ]
 ( Account, Summa )
```

В результате выполнения этого примера в выходной поток будет выведен номер счета из переменной ***Account*** и остаток на счете из переменной ***Summa***.

Синтаксическая форма инструкции выглядит следующим образом:

```
'['управляющая строка']' [ '('фактические параметры')']
```

Список фактических параметров заключается в круглые скобки. Он представляет собой список выражений, разделенных запятыми, в котором при необходимости задаются спецификаторы форматирования (см. стр. 67).

Формат управляющей строки

Управляющая строка представляет собой последовательность печатных и управляющих символов. Управляющие символы имеют более высокий приоритет, чем спецификаторы форматирования (см. стр. 67).

В языке RSL применяется только один управляющий символ форматирования – это знак номера "#". Все остальные символы считаются печатными и выводятся на устройство вывода без изменений. Последовательность символов ‘#’ задает поле для вывода в него соответствующего значения из списка параметров. Количество символов ‘#’ задает ширину поля. Совокупность управляющих символов и спецификаторов форматирования определяет характеристики вывода значений (позицию в панели, ширину, выравнивание, количество десятичных знаков и т.д.). Количество полей в панели должно соответствовать количеству параметров, указанных для этой панели.

Пример:

```
[  ########## ] (123.45)          /* Вывод поля шириной 10 символов */
```

Одиночный управляющий символ "#" задает не ширину поля вывода, равную 1, а позицию табуляции, то есть точку, относительно которой происходит выравнивание поля вывода. В этом случае ширина поля вывода задается спецификатором. Если в соответствии с заданным спецификатором поле выравнивается по левому краю ("l"), позиция табуляции задает левую границу поля вывода. Если поле выравнивается по правому краю ("r"), позиция табуляции задает правую границу поля вывода. Если поле выравнивается по центру ("c"), позиция табуляции задает центральную позицию поля вывода.

Пример:

```
a = “Пример”;
[       |
        #
        #
        #  ]( a:l, a:r, a:c)
```

В выходной поток будет выведено:

```
Пример
Пример
Пример
```

Определение переменных VAR

Имена переменных RSL необходимо объявлять явно, используя определение VAR.

Синтаксическая форма определения VAR имеет следующий вид:

```
[local | private]VAR идентификатор [: имя типа ] [ '=' выражение ]
{ ',' идентификатор [: имя типа ] [ '=' выражение ] }
```

Список идентификаторов может содержать произвольное количество имен переменных. Инициализация заключается в присвоении идентификатору переменной значения какого-либо выражения. Допускается инициализировать не все объявленные переменные. Конструкции, относящиеся к отдельным переменным, должны быть разделены запятой.

После того, как определение VAR будет обнаружено компилятором в тексте программы хотя бы один раз, любая необъявленная явно переменная в текущем RSL-модуле приведет к сообщению об ошибке.

Пример:

```
var
  Остаток40 : money,
  Дата = date;
```

При необходимости переменная RSL может быть объявлена внутри инструкций ***if***, ***for***, ***while***. Кроме этого, переменную цикла **for** допустимо декларировать непосредственно в цикле, в круглых скобках. Областью видимости объявленных таким образом переменных является не только вложенный блок, в котором они объявлены, а **вся** программа.

Пример:

```
var a = 10;
if (a < 100)
  var b : Integer = 70;
  println (b)
end;
while (true)
  var c = b;
  println (c);
  break
end;
for (var i : Numeric, 1, 10)
  var d : String = “80”;
  println (i);
end;
println (i);
```

Определение классов и объектов

В языке RSL имеется возможность создавать классы и объекты. Классы и объекты, например, используются для поддержки программирования визуальной среды, в котором очень удобен объектно-ориентированный подход: такие ее специфические черты, как отчеты, кнопки, поля редактирования, переключатели всегда можно представить в качестве объектов со своими свойствами и методами.

Синтаксическая форма определения классов выглядит следующим образом:

```
[local | private] Class [ ‘(‘идентификатор базового класса’)’]
идентификатор [‘ (‘список формальных параметров’)’ ]
        <свойства класса>;
        <методы класса>;
End
```

Идентификатор класса – это имя конструктора класса, которое используется при создании экземпляра этого класса. Использование идентификатора класса в выражении создает экземпляр этого класса – объект.

Список формальных параметров содержит разделенные запятыми идентификаторы переменных, которые являются параметрами инициализатора объекта класса.

Все локальные переменные, находящиеся внутри определения класса, являются свойствами класса, а все локальные макропроцедуры – методами. Любой код внутри определения класса, не включенный в методы, является кодом конструктора.

Приведем пример, в котором задается класс "Персона", имеющий два формальных параметра ***p1*** и ***p2*** и содержащий:

- два свойства – "Имя" и "Фамилия";
- метод, выводящий на печать информацию, описываемую объектом;
- код для инициализации объекта.

Пример:

```
Class Персона (p1, p2)
        Var Имя, Фамилия;
Macro Отчет
                Println ("Имя:", Имя);
                Println ("Фамилия:", Фамилия);
        End;
Имя = p1;
Фамилия = p2;
End;
```

Чтобы использовать заданный класс, необходимо создать экземпляр класса – объект:

```
Obj = Персона ("Иван", "Петров");
```

В этой инструкции объект одновременно и создается, и инициализируется. Ссылка на объект присваивается переменной Obj.

В языке RSL предусмотрено автоматическое создание объектов классов при первом обращении к декларированной переменной класса. Если конструктор класса имеет параметры, то в этом случае при создании объекта параметры будут иметь значение NULL.

Пример:

```
Class TestClass
    Var prop = 10;
End;
Var Ob:TestClass; /* Декларируем переменную типа TestClass */
Println (Ob.prop);  /* Объект автоматически создается */
```

Object RSL позволяет наследовать классы объектов от других классов. Имя базового класса указывается в круглых скобках после ключевого слова CLASS. Для инициализации базового класса необходимо вызвать предопределенный метод, название которого образуется путем добавления к имени класса приставки "Init". Вызов инициализатора базового класса может располагаться в любом месте определения дочернего класса.

Пример:

```
Унаследуем класс "Сотрудник" от класса "Персона", добавив к
нему свойство "Должность":
Class ( Персона ) Сотрудник ( п_Имя, п_Фамиля, п_Должность)
        Var Должность;
        Macro Отчет
                Отчет;
                Println ("Должность:", Должность);
        End;
        InitПерсона (п_Имя, п_Фамилия);
        Должность = п_Должность;
End;
```

Множественное наследование в Object RSL не поддерживается.

Обращение к свойствам и методам класса осуществляется следующим образом:

```
Obj.Имя;
Obj.Отчет;
```

Кроме того, Object RSL поддерживает обращение к свойствам RSL-классов не только по имени, но и по индексу.

Пример:

```
Class Test
  Var prop1, prop2;
End;
Ob = Test;
A = ob.prop1; /* Доступ к свойству по имени */
B = ob (0); /* Доступ к свойству по индексу */
```

Для совместимости со стандартом автоматизации в Object RSL добавлена поддержка свойства по умолчанию. Свойством по умолчанию считается свойство, имя которого в выражении можно не указывать.

Пример:

```
Объект ob содержит свойство по умолчанию item. Для обращения
к свойству item можно записать:
ob. (10) = 100;
вместо:
ob.item (10) = 100;
```

Если метод имеет параметры, то они указываются в круглых скобках аналогично параметрам макропроцедур RSL (см. стр. 40).

```
A = ob.met (3);
```

Классы Object RSL могут иметь и свойства с параметрами. В настоящее время такие классы можно создать, например, при помощи специального инструмента DLM SDK (см. стр. 239). Параметры свойства аналогично параметрам метода указываются в круглых скобках:

```
A = ob .prop (3);
```

Классы Object RSL могут содержать деструктор, определяемый пользователем. Для его использования в определении класса необходимо указать метод с предопределенным именем ***Destructor***. Деструктор вызывается автоматически при "разрушении" объекта.

Пример:

```
Class Test
   macro Destructor
      printlm (“Destructor called”);
   end;
end;
```

Если деструктор не определен, то все выделенные при создании объекта ресурсы освобождаются автоматически.

В коде методов все методы и свойства доступны непосредственно по имени. При этом в каждый метод класса передается скрытый параметр ***this***, представляющий собой ссылку на объект, для которого вызывается метод или свойство.

Ссылки на объекты хранятся в переменных RSL. При очистке переменной происходит удаление ссылки на объект. Переменная очищается перед присвоением ей нового значения либо при выходе из области видимости. Таким образом, объект класса будет существовать и его невозможно будет удалить до тех пор, пока на него есть хотя бы одна ссылка. При удалении последней ссылки на объект удаляется сам объект.

В языке RSL также предусмотрено использование так называемых "слабых" ссылок, которые не управляют временем жизни самого объекта, то есть объект можно удалить, если на него существуют "слабые" ссылки. Механизм "слабых" ссылок в настоящее время поддерживается только для объектов классов, созданных на языке RSL.

"Слабая" ссылка реализуется встроенным классом ***WeakRef***. Конструктор класса выглядит следующим образом:

WeakRef (obj:object) : object

Конструктор возвращает объект, который представляет собой "слабую" ссылку на объект ***obj***. Этот объект в выражениях используется точно так же, как объект ***obj***. При попытке использовать "слабую" ссылку на объект, который уже удален, возникает ошибка времени исполнения.

В языке RSL предусмотрен тип данных ***MethodRef***, представляющий собой ссылку на метод объекта. Значение этого типа можно получить при помощи процедуры ***R2M***. Используя ссылку на метод, можно вызвать этот метод при помощи процедур ***ExecMacro***, ***ExecMacro2***, для этого следует передать в качестве первого параметра переменную типа ссылка на процедуру, или при помощи процедуры ***CallR2M***.

Все методы классов являются виртуальными. Добавление в дочерний класс метода с именем, которое уже используется для одного из методов базового класса, вызовет замену метода родительского класса методом дочернего класса. Применение наследования с использованием виртуальных методов позволяет модифицировать функциональность базовых классов. Кроме того, в Object RSL можно заменять методы конкретного экземпляра класса. Для этого используется стандартная процедура ***GenAttach*** (см. стр. 218).

RSL-классы можно наследовать от классов, определенных в DLM-модулях, которые создаются с помощью специального инструмента DLM SDK (см. стр. 239), а также от встроенных классов, таких как ***TBFile***, ***TDirList*** и т.п. Такие наследуемые классы являются внешними по отношению к RSL-классам.

Особенностью использования RSL-классов, унаследованных от внешних классов, является невозможность обращения напрямую к методам родительского класса после того, как в дочернем классе эти методы были переопределены. Для этих целей служит стандартное свойство RSL-классов ***_extObj*,** возвращающее ссылку на внешний объект. Это свойство позволяет переопределять методы базового класса в дочернем и сохраняет возможность вызова методов базового класса.

Пример:

```
class (TBFile) TBfileEx (name,mode,key)
  InitTBFile (name,mode,key);
  macro GetDirect (pos)
    return _extObj.GetDirect (pos);
  onerror
    println (status);
    if (status == 43)
      return false;
    end;
    RunError;
  end;
end;
ob = TBfileEx ("client.dbt","w",0);
ob.next;
pos = ob.getPos;
println (ob.GetDirect (pos));
```

При использовании пользовательских RSL-классов предусмотрено преобразование объектов в строку. Для этого необходимо определить в классе метод с именем ***ToString***, возвращающий символьную строку.

Пример:

```
class Test
  macro ToString
    println (“Это мой объект Test”)
  end;
end;
println (Test);
// Выводит:
Это мой объект Test
```

Определение символических констант CONST

Все символические константы должны быть объявлены явно при помощи определения CONST. Кроме объявления имени, необходимо проинициализировать его каким-либо значением.

Список идентификаторов константы будет соответствовать типу заданной величины.

Синтаксическая форма определения CONST имеет следующий вид:

```
[local | private]CONST идентификатор [: имя типа ]  '=' выражение
{ ','  идентификатор [: имя типа ]'=' выражение }
```

Список идентификаторов может содержать произвольное количество проинициализированных имен констант, разделенных запятыми. Попытка изменить значение любой из объявленных констант приведет к сообщению об ошибке при компиляции.

Пример:

```
const
   Количество = 35,
  ФИО = "Иванов Иван Иванович";
```

Определение процедуры MACRO

Определение MACRO используется для описания процедуры RSL:

```
[local | private] MACRO идентификатор [ '('список формальных
параметров')'] [: имя типа ]
    список определений и инструкций
END
```

Идентификатор процедуры – это ее имя, которое используется в инструкции вызова процедуры для выполнения.

Список формальных параметров заключается в круглые скобки. Он имеет следующий вид:

```
идентификатор [ : имя типа] {, идентификатор [ : имя типа]}
```

Этот список содержит разделенные запятыми идентификаторы переменных, которым при вызове этой процедуры будут присвоены значения фактических параметров. Список определений и инструкций может содержать любые инструкции и определения языка RSL, в том числе и другие определения MACRO. Таким образом, структура процедуры повторяет структуру RSL-программы.

Пример:

```
/* Пример определения процедуры, печатающей */
/*  содержимое файла:*/
MACRO Содержимое (ff)
   WHILE (next (ff))
    [    #    |############### |    #   ]
    (ff.Balance,       ff.Name_Part,          ff.Part)
   END;
END;
```

Процедуры языка RSL

Процедуры являются неотъемлемой частью любой программы, написанной на RSL. Они являются фрагментом программы со своим именем, к которому можно обратиться для выполнения необходимых действий.

Кроме этого, допускается использовать стандартные процедуры, входящие в состав языка RSL. Обращение к встроенным процедурам так же осуществляется по имени.

Для косвенного вызова процедуры (определенной пользователем или стандартной) служит стандартная процедура ***execmacro***.

Процедуры RSL могут присутствовать в выражениях, при этом вместо процедуры в выражение будет подставлено значение, возвращаемое процедурой.

Если процедура явно не возвращает значение, то возвращаемое значение будет иметь тип V_UNDEF.

Передача параметров

При описании процедуры может быть указан список параметров:

```
macro My (a,b,c,d)
```

Здесь ***a,b,c,d*** являются формальными параметрами, создаваемыми и инициализируемыми значениями фактических аргументов при входе в процедуру.

После завершения процедуры эти переменные прекращают свое существование.

Нумерация параметров начинается с нуля. При вызове процедуры после ее имени задаются значения фактических параметров, которые заменят формальные при выполнении.

Передача параметров в процедуре производится по значению, однако, параметры типа ARRAY, RECORD и FILE, а также все объекты RSL-классов передаются по ссылке.

Так как передача параметров производится по значению, поэтому изменение формальных параметров в вызванной процедуре не приводит к изменению значений соответствующих им фактических аргументов в вызывающей. Однако пользователю предоставляется возможность изменить значения фактических параметров процедуры. Для этого необходимо выполнение следующих условий:

- при указании типа параметров перед декларацией типа должен находиться символ "@";
- фактические параметры при вызове процедуры должны быть переданы по ссылке.

Пример:

```
/* Пример модификации параметров:*/
macro Test (p1:@variant, p2:@integer)
  p1 = 10;
  p2 = 20;
end;
var v1, v2;
Test (@v1, @v2);
/* В результате переменная v1 будет равна 10, а v2 равна 20. */
```

Изменить значения фактических параметров можно также при помощи процедуры ***setparm*** (см. стр. 180). В этом случае в качестве изменяемого фактического параметра должна быть переменная. Если же это условие не выполняется, значение фактического параметра не изменится, при этом система не сгенерирует сообщения об ошибке.

Пример:

```
setparm (0,10+aa)          /* Замена первого параметра в вызывающей
                              процедуре (с номером 0) на значение 10+aa */
setparm (1,Local (bb));    /* Замена второго параметра в вызывающей
                              процедуре (с номером 1) на значение Local (bb) */
```

Пример процедуры, изменяющей значения фактических параметров в вызывающей процедуре:

```
macro  Demo (number , text)
   setparm (0,10);
   setparm (1,"Привет")
end;
```

Теперь после вызова:

```
var id, name;
Demo (id, name)
```

переменная ***id*** получит значение 10, а переменная ***name*** – значение "Привет".

Если количество фактических параметров меньше количества формальных, то вместо недостающих параметров передаются неопределенные значения типа V_UNDEF. Если количество фактических параметров больше количества формальных, то к избыточным параметрам можно получить доступ через обращение к процедуре ***getparm***.

При описании процедуры можно вообще не указывать формальные параметры. В этом случае доступ к фактическим параметрам производится только через обращение к процедуре ***getparm***.

Пример:

```
getparm (0,parm1);   /* Присвоить переменной parm1 значение первого
                        параметра (с номером 0) из вызывающей процедуры */
getparm (1,parm2);   /* Присвоить переменной parm2 значение второго
                        параметра (с номером 1) из вызывающей процедуры */
```

Наличие процедуры ***getparm*** позволяет писать процедуры, принимающие произвольное количество параметров.

Пример процедуры, вычисляющей и возвращающей сумму всех своих параметров:

```
macro Calc
   var sum = 0, i, val;
   while (getparm (i,val))
      sum = sum + val;
      i = i + 1;
   end;
   return sum
end
```

Теперь можно вызывать процедуру ***Calc*** так:

```
sum = Calc (1,2,3,4,5);
sum = Calc (10,20);
```

Определение массивов

В языке RSL можно объявлять только одномерные массивы. Для этого предусмотрены следующие конструкции:

- конструкция ARRAY (см. стр. 43);
- стандартный класс ***TArray*** (см. стр. 44).

Примечание.
*При определении массивов предпочтительнее использовать класс **TArray**. Конструкция ARRAY в настоящее время поддерживается для совместимости с ранее разработанными программами.*

Определение массивов с помощью конструкции ARRAY

Синтаксическая форма определения ARRAY имеет следующий вид:

[local | private] ARRAY идентификатор { ',' идентификатор }

Список идентификаторов может содержать произвольное количество имен, разделенных запятыми. Идентификатор массива – это имя объекта, через которое можно ссылаться на определенный массив. В выражении идентификатор массива преобразуется к переменной типа V_ARRAY (ссылка на массив). Благодаря этому, идентификатор массива можно присваивать переменным и передавать в качестве параметров процедурам.

Элементы массива могут хранить значения переменных любых типов, включая тип V_ARRAY (ссылка на массив). Их нумерация начинается с 0. Размер массива изменяется динамически. Он может быть задан неявно в момент присвоения значений элементам:

Пример:

```
/* Определение пустых массивов */
array MyVar, First, Second;
/* Присвоение значений элементам массива First*/
First (0) = "Имя";
First (1) = Second;
/* Присвоение значений элементам массива Second */
Second (0) = 1;
Second (1) = 23;
Second (2) = 567;
/* Определение размера массива MyVar */
asize (MyVar, 10 );
/* Присвоение значения элементу массива MyVar */
MyVar (9) = First;
[###############] (MyVar (9)(1)(2) );
/*В стандартный выходной поток */ будет выведено: 567 */
/*Конструкция MyVar (9)(1)(2) эквивалентна конструкции MyVar
(9,1,2).*/
```

Текущий размер массива можно определить явно при помощи процедуры ***asize***:

Пример:

```
cc = asize (aa);
```

Используя процедуру ***asize*** с двумя параметрами, можно изменить размер уже существующего массива. При этом старое содержимое удаляется, и распределяется новый массив заданного размера. Рекомендуется использовать эту возможность, чтобы уменьшить количество перераспределений памяти.

Пример:

```
asize (aa,10);   /* Установить для массива aa новый размер 10 */
i = 0;
while (i < 10)    /* Цикл присвоения элементам новых значений */
  aa (i) = I;
  i = i + 1;
end;
```

Если в левой и правой части инструкции присваивания стоят массивы или переменные типа V_ARRAY, то происходит копирование массивов.

Пример:

```
dd = aa;      /* dd – переменная типа V_ARRAY ссылается на массив aa  */
bb = dd;      /* Создается копия массива dd */
```

Стандартный класс TArray

Стандартный класс ***TArray*** языка RSL используется для реализации динамического массива.

Динамический массив ***TArray*** представляет собой объектную альтернативу стандартной конструкции языка ARRAY. Имеется возможность замены конструкции ARRAY на новую конструкцию ***TArray***:

Пример:

```
Array myArray; /* Прежняя конструкция */
myArray = TArray; /* Замена прежней конструкции на новую */
```

Синтаксическая форма конструктора ***TArray*** имеет следующий вид:

```
TArray (MarshalByVal: bool, rootSz: integer, delta: integer)
```

Память под элементы массива выделяется постранично. Параметр ***MarshalByVal*** задает одноименное свойство для нового объекта. Оставшиеся параметры конструктора класса ***TArray*** задают количество элементов массива на первой странице (значение ***rootSz***) и на последующих страницах (значение ***delta***), под которые выделяется память. По умолчанию первая страница может содержать 50 элементов, все следующие страницы – по 10. Эти значения можно изменить, указав параметры конструктору класса:

Пример:

```
/* Опишем массив, в котором первая страница содержит 100
элементов, а все следующие – по 5:*/
MyArray = TArray ( true, 100, 5 );
```

В качестве параметра можно указать ссылку на уже существующий массив. В этом случае создается новый массив, содержащий копию элементов массива, обозначенного в качестве параметра.

Доступ к элементам массива осуществляется посредством указания индекса элемента в круглых скобках после переменной, ссылающейся на объект класса ***TArray***.

Пример:

```
FirstArray = TArray;
/* Присвоение значения */
FirstArray (0) = 100;
/* Создается копия массива FirstArray */
SecondArray = TArray (FirstArray);
```

Объекты класса ***TArray*** имеют свойство ***Size***, значением которого является текущий размер массива. Присваивая свойству ***Size*** новое значение, можно изменять размер массива. Увеличивается размер массива динамически при обращении к элементу, стоящему за последним элементом массива:

Пример:

```
Println ( "Текущий размер массива:", FirstArray.Size );
```

Объекты класса ***Tarray*** имеют неименованные свойства – элементы массива, к которым можно обращаться только по индексу. Для доступа к ним предусмотрено свойство ***Value (N)*** – значение N-го элемента массива.

Следует отметить, что с появлением у объектов свойств с параметрами возникла проблема, связанная с одинаковым синтаксисом указания индекса (см. стр. 34) и свойства с параметрами. Поэтому для разрешения неоднозначности предусмотрена возможность указания индекса свойства объекта в квадратных скобках.

Пример:

```
Ob = Tarray;
Ob [5] = 100;
```

Пример:

```
/* Пример обращения к элементам массива */
class TTT                       /* Определяем класс TTT */
        var prop = TArray;
        prop (5) = 200;
        macro Me
                return prop;
        end;
end;
ob = TTT;       /* Создаем экземпляр класса TTT  */
/* Распечатаем значение пятого элемента массива, */
/* на который ссылается свойство prop тремя способами: */
println (ob.prop() (5));
println (ob.prop .value (5));
println (ob.prop [5]);
```

Объекты класса ***TArray*** имеют свойство ***MarshalByVal*** типа Bool, которое определяет, каким образом объекты этого класса передаются в RSCOM. Если свойству присвоено значение TRUE, то объекты передаются по значению. По умолчанию свойство имеет значение FALSE, и, соответственно, объекты класса ***TArray*** по умолчанию передаются по ссылке.

Кроме перечисленных выше свойств, объекты класса ***TArray*** имеют метод **Sort (callback:Variant, data:Variant):Bool**. Метод предназначен для сортировки элементов массива в соответствии с порядком, определяемым пользовательским обработчиком ***callback***. Обработчик используется для сравнения двух элементов массива и имеет следующий прототип:

macro Compare (left:Variant, right:Variant, data:Variant):Integer

Алгоритм работы обработчика следующий:

- если левый элемент (***left***) больше правого (***right***), обработчик возвращает значение, которое больше **0**;
- если левый элемент (***left***) меньше правого (***right***), обработчик возвращает значение, которое меньше **0**;
- если элементы равны, обработчик возвращает ***0***.

Указанный обработчик может быть задан в виде строки с именем глобальной процедуры RSL, ссылки на процедуру или ссылки на метод класса.

Параметр дата (***data***) передаётся из процедуры сортировки в пользовательский обработчик.

В случае успешного завершения метод возвращает TRUE; в случае возникновения ошибки или при неверно заданных параметрах метод возвращает FALSE.

Поддержка технологии ActiveX в RSL

В Object RSL предусмотрена поддержка любых ActiveX-объектов, являющихся объектами автоматизации и зарегистрированными в системе. Пользователи Object RSL могут создавать ActiveX-объекты и обращаться к их свойствам и методам из RSL-программы.

Чтобы воспользоваться этой возможностью, необходимо импортировать модуль ***rslx*** и вызвать конструктор ActiveX-объектов. Синтаксическая форма конструкции ***ActiveX*** выглядит следующим образом:

ACTIVEX (имя объекта [, имя удаленного компьютера [, ссылка на объект]])

В качестве первого параметра передается идентификатор необходимого класса COM-объекта (progID). Вторым (необязательным) параметром является имя компьютера, на котором создается объект. В качестве третьего параметра можно передать значение TRUE, в этом случае конструктор будет пытаться вернуть ссылку на уже существующий объект, а не создавать его новый экземпляр.

Пример:

1) Запишем строку в MS Word:

```
ob = ActiveX ("Word.basic");
ob.FileNew;
ob.Insert ("Hello, World!");
ob.AppShow;
```

2) Подключимся к объекту автоматизации Excel.Applicacation и добавим рабочую книгу Excel. Если приложение Excel уже было запущено, то его новая копия запускаться не будет.

```
import rslx;

ob = ActiveX ("Excel.Application", NULL, TRUE);
ob.Visible = TRUE;
ob.Workbooks.add;

MsgBox ("Нажмите любую клавишу для выхода из Excel");

ob.Quit
```

Как видно из примера, способ обращения к свойствам и методам объекта автоматизации ничем не отличается от обращения к свойствам и методам объекта RSL.

Следует иметь в виду, что некоторые методы объектов ActiveX возвращают строку символов, включающую нули. В RSL конец строки определяется по нулевому символу, поэтому информация таких строк доступна только до первого нуля. Для корректного отображения полученной строки результат может быть выведен с помощью нультерминированной строки. С этой целью следует использовать значение ***nullVal***, заданное в качестве последнего параметра свойства или метода объекта ActiveX. Указанное значение не передается в свойства или методы объекта и служит лишь сигналом о том, что возвращаемое значение типа VT_BSTR необходимо вернуть в виде массива строк.

Пример:

```
import rcw;
cpwin;
const cdlOFNAllowMultiselect = #200;
const cdlOFNExplorer = #80000;
const cdlOFNFileMustExist = #1000;
const NoChangeDir = #8;
var srv = CreateObject ("rsax", "TRsAxServer", "RsAxServer", false);
var oDlg = srv.CreateComObject ("MSComDlg.CommonDialog");
oDlg.Flags =
cdlOFNAllowMultiselect+cdlOFNFileMustExist+cdlOFNExplorer+NoChan
geDir;
oDlg.MaxFileSize = 32767;
oDlg.ShowOpen();
var result = oDlg.FileName (nullVal);
var n;
for (n, result)
  println (n)
end;
```

ActiveX является стандартным классом языка RSL, и ActiveX-объект может быть создан с помощью процедуры ***GenObject***. В этом случае синтаксис определения имени класса выглядит следующим образом:

"ActiveX\\<имя объекта>[\\<имя удаленного компьютера>]"

Пример:

Создадим объект Word.Basic:

```
ob = GenObject ("ActiveX\\Word.basic")
```

Доступ к объектам, созданным с помощью этой конструкции, будет только локальным. Если необходимо создавать или запускать ActiveX-объект не только локально, но и удаленно, для этого следует воспользоваться RSCOM-технологией: вместо встроенной конструкции ***ActiveX*** использовать стандартный RSCOM-сервер ***Rsax*** (см. стр. 156).

Использование специальных значений

Для поддержки специальных значений, используемых в COM-автоматизации, в языке RSL предусмотрены два значения:

- **OptVal** – значение используется для передачи опциональных параметров.
- **NullVal** – значение используется для передачи специального значения NULL из COM-автоматизации.

При передаче этих значений в методы объектов COM-автоматизации выполняются следующие преобразования:

```
OptVal -> VT_ERROR, scode = DISP_E_PARAMNOTFOUND
NullVal -> VT_NULL
```

Значение NULL языка RSL транслируется следующим образом:

```
Null -> VT_EMPTY
```

Пример:

```
import rcw;
ob = CreateObject ("rsax","TRsAxServer","RsAxServer",false);
var ex = ob.CreateComObject ("Excel.Application",false);
ex.visible = true;
var wb = ex.WorkBooks.add;
var sh = wb.ActiveSheet;
sh.Range ("A1").value = $100;
// В качестве первого параметра использовать значение по
умолчанию.
var MySheet=wb.Sheets.Add (OptVal,sh);
MySheet.Name = "My sheet";
```

Соответствие типов данных языка RSL и ActiveX

При использовании ActiveX-объектов в программе на языке RSL выполняется преобразование типов данных языка RSL в типы данных ActiveX и наоборот по правилам, приведенным в таблице:

Тип RSL	Тип ActiveX
NullVal	VT_NULL
OptVal	VT_ERROR (DISP_E_PARAMNOTFOUND)
Null	VT_EMPTY
Integer	VT_I4, VT_UI1, VT_I1,VT_UI2, VT_I2, VT_UI4
Double	VT_R8, VT_R4
Money	VT_CY
Bool	VT_BOOL
String	VT_BSTR
Date	VT_DATE
Time	VT_DATE
Dttm	VT_DATE
Object	VT_DISPATCH
TArray	SAFEARRAY(VARIANT)

Типы данных языка RSL, для которых представлено несколько типов ActiveX, при конвертировании в ActiveX преобразуются в первый тип из списка.

При преобразовании типа данных ActiveX VT_DATE в тип данных RSL действует следующее правило:

- если в переменной типа VT_DATE содержится непустое время и непустая дата, результатом будет RSL-тип Dttm;
- если в переменной типа VT_DATE содержится пустое время и непустая дата, результатом будет тип Date;
- если в переменной типа VT_DATE содержится пустая дата и непустое время, результатом будет тип Time.

Пустым временем считается время "0:0:0,0", пустой датой – дата 30.12.1899.

Поддержка стандартных коллекций

Object RSL позволяет использовать стандартные коллекции объектов автоматизации. Для создания таких коллекций и доступа к их содержимому существует специальный стандарт, в значительной степени облегчающий их использование.

Так, например, в приложении Excel есть коллекция рабочих книг. Пусть переменная ***ob*** содержит ссылку на объект ***Excel.Application***. Для создания коллекции язык RSL позволяет создать объект-перечислитель при помощи предопределенного метода ***CreateEnum***, который автоматически добавляется к каждому ActiveX-объекту.

Пример:

```
En = ob.Workbooks.CreateEnum;
While (En.Next )
    MsgBox (En.Item.Name)
End;
```

В этой программе для навигации по коллекции используется метод ***Next*** объекта-перечислителя. Когда коллекция просмотрена до конца, этот метод возвращает значение FALSE. Ссылка на очередной элемент хранится в свойстве ***Item***. Еще один метод объекта-перечислителя — ***Reset***. После его вызова можно всю итерацию повторить снова.

В качестве стандартной коллекции может выступать объект ***Tarray***.

Пример:

```
Ar = Tarray;
Ar (0) = "Element 1";
Ar (1) = "Elemrnt 2";
En = Ar.createEnum;
While (En.next)
   Println (En.Item)
End;
```

Обращение к свойствам объектов с параметрами

Кроме обычных свойств, объекты автоматизации могут содержать свойства с параметрами, к которым можно обращаться из Object RSL. Рассмотрим пример:

```
ob = ActiveX ("XArray.XArray");
ob.ReDim (0,10,0,20);  /* У массива будет две размерности 0-10 и 0-
20 */
i = 0;
while (i < 10)
  ob.Value (i,0) =  "String "+i;
   i = i + 1
end;
```

Объект ***XArray.Xarray*** реализует двумерный массив. Элементы этого массива хранятся в свойстве ***Value***, а индексы элементов определяются параметрами этого свойства.

При поддержке свойств с параметрами может потребоваться внести изменения в существующие программы. Предположим, что есть класс Object RSL ***MyClass***:

```
Class MyClass
   Var prop = Tarray;
End;
Ob = MyClass;
```

Раньше присвоить значения элементам массива можно было следующим образом:

```
Ob.prop (0) = "Строка1";
Ob.prop (1) = "Строка2";
```

Теперь же в такой записи значения 0 и 1 будут истолкованы как параметры свойства ***prop***, и после выполнения приведенных инструкций это свойство будет содержать не массив из двух элементов, а значение ***Строка2***. Чтобы все-таки присвоить значения элементам массива, следует сделать следующую запись:

```
Ob.prop ()(0) = "Строка1";
Ob.prop ()(1) = "Строка2";
```

Для разрешения неоднозначности при доступе к элементам массива также можно использовать квадратные скобки.

```
Ob.prop [0] = "Строка1";
Ob.prop [1] = "Строка2";
```

Используя свойство ***Value***, с помощью которого можно обращаться к элементам массива, приведенные выше инструкции можно переписать иначе:

```
Ob.prop.Value (0) = "Строка";
Ob.prop.Value (1) = "Строка2";
```

Доступ к объектам Object RSL из других языков

Все объекты классов языка Object RSL являются объектами автоматизации и, следовательно, могут быть доступны из других языков программирования. Рассмотрим пример обращения к объекту Object RSL из Visual BASIC.

Объект Object RSL может быть передан в процедуру Visual BASIC, например, в качестве параметра. Предположим, что на языке Visual BASIC реализован простой сервер автоматизации, к которому можно обратиться, указав его идентификатор ***VbServer.VbClass***. Пусть этот сервер экспортирует следующую процедуру:

```
Public Sub TetRSL (ob As Object)
  ob.Test "From Basic", " Parm1 ", "Parm2"
End Sub
```

Теперь на языке Object RSL создадим два объекта: один — класса RSL, другой — класса Visual BASIC. Вызов метода ***Test*** класса Object RSL в языке Visual BASIC осуществляется следующим образом:

```
import rslx;
class   TestRSLClass
   macro Test (p1,p2,p3)
        MsgBox ("Вызван метод Test с параметрами: ", p1, " ", p2, " ",
p3)
    end;
end;
vb = ActiveX ("VbServer.VbClass");
rsl  = TestRSLClass;
vb.TestRSL (rsl);
```

Обработка событий

Объекты автоматизации могут иметь не только свойства и методы, но и события, то есть способы уведомления клиента объекта о происходящих в объекте изменениях. Object RSL позволяет обрабатывать эти события, то есть принимать сообщения от объектов автоматизации и выполнять необходимые действия. Для этого предусмотрены следующие классы: ***TRslEvHandler*** и ***RslTimer***.

Класс TRslEvHandler

Класс ***TRslEvHandler*** предназначен для обработки событий от ActiveX-объектов. Он может использоваться несколькими способами: от него можно унаследовать класс - обработчик событий, можно создать экземпляр класса ***TRslEvHandler*** и передать в качестве параметра конструктора объект - обработчик событий и т.п. Можно также создать экземпляр класса ***TRslEvHandler*** и не передавать объект-обработчик, в этом случае для обработки событий будут использоваться глобальные макропроцедуры.

Чтобы объект класса ***TRslEvHandler*** мог обрабатывать события от некоторого ActiveX-объекта, этот ActiveX-объект должен быть добавлен во внутреннюю коллекцию объекта класса ***TRslEvHandler*** путем присвоения его свойству ***EvSource***.

Нет необходимости создавать глобальный экземпляр класса ***TRslEvHandler***, RSL автоматически создает его с именем ***RslEvHandler***.

Для создания экземпляра этого класса необходимо вызвать конструктор ***TRslEvHandler***:

TRslEvSourse ([handlerObj] [, doAddRef])

Параметры конструктора задают следующие значения:

- **handlerObj** – объект-обработчик событий, методы которого будут использоваться для обработки событий. Если параметр не задан или задано значение NULL, то в качестве объекта-обработчика выступает вся RSL-программа, а для обработки событий будут использоваться глобальные макропроцедуры.
- **doAddRef** – значение типа Bool. Если параметр равен TRUE, то конструктор класса автоматически инкрементирует счетчик ссылок на объект handlerObj. В противном случае инкремент выполняться не будет. Это полезно в том случае, когда объект класса ***TRslEvHandler*** создается как свойство класса обработчика. Если параметр не задан, то по умолчанию его значение принимается равным TRUE.

Свойства класса TRslEvHandler

EvSource

Это свойство обеспечивает доступ к внутренней коллекции объектов - источников событий. Чтобы поместить объект источник во внутреннюю коллекцию, необходимо присвоить объект-источник этому свойству.

Возможны два способа использования этого свойства. Рассмотрим первый способ:

```
EvSource ( pref [, intfName ] ) = ob
```

Параметры при этом задают следующие значения:

- **pref** – уникальная строка для этого объекта класса ***TRslEvHandler***, используемая при формировании имени объекта - источника событий во внутренней коллекции. Имя метода для обработки события образуется в формате **pref_**<имя события>.
- **intfName** – имя интерфейса событий для объекта-источника в библиотеки типов. Этот параметр является необязательным. Его необходимо указывать только для тех объектов, для которых RSL не может получить эту информацию автоматически, например, для объектов Microsoft Excel. В некоторых случаях требуется еще явно указать имя библиотеки типов при помощи свойства ***TypeLib***.
- **ob** – ActiveX-объект, который является источником событий.

Второй способ более эффективен, так как вообще не требует использования библиотеки типов, но для его применения необходимо вручную задать имена обработчиков и соответствующие им идентификаторы событий:

```
EvSource ( pref , GUID, numProc ) = ob
```

Параметр ***pref*** используется так же, как при первом способе. Остальные параметры задают следующие значения:

- **GUID** – задает уникальный идентификатор интерфейса событий в виде строки, например "{00024413-0000-0000-C000-000000000046}".
- **numProc** – количество процедур обработчиков, которые будут добавлены при помощи метода ***SetHandler***.

Этот способ также позволяет динамически подключать и отключать обработчики событий во время выполнения RSL-программы. Идентификаторы событий можно узнать из документации на используемый ActiveX-объект или из анализа библиотеки типов.

Для того чтобы изъять объект источник событий из коллекции и тем самым прекратить обработку событий, необходимо присвоить свойству ***EvSource*** с соответствующим параметром ***pref*** значение NULL. Это, например, необходимо сделать перед вызовом метода ***Quit*** объекта ***Exel.Application***.

TypeLib

Свойству ***TypeLib*** можно присвоить имя файла, содержащего библиотеку типа для объектов источников. Это необходимо делать только в том случае, если RSL не может получить необходимую информацию автоматически.

Пример:

```
TypeLib = fileName
```

Методы класса TRslEvHandler

SetHandler

Метод устанавливает связь между событием и методом обработки.

Имя для метода обработки может быть выбрано произвольно, но при его формировании удобнее следовать правилу, принятому при использовании первого варианта работы со свойством ***EvSource***.

Рассмотрим пример использования метода:

SetHandler (pref, proc, id)

При использовании этого метода применяются следующие параметры:

- **pref** –строка с именем объекта - источника событий во внутренней коллекции. Объект с этим именем должен быть предварительно присвоен свойству ***EvSource*** с использованием второго способа.
- **proc** – строка с именем метода обработчика события в объекте обработчике событий.
- **id** – целочисленный идентификатор события в объекте-источнике.

RemHandler

Метод разрывает связь события и процедуры обработчика, установленной при помощи метода ***SetHandler***.

Рассмотрим пример использования метода:

RemHandler (pref, id)

Параметры, используемые в этом методе, аналогичны параметрам метода ***SetHandler***.

raise

Этот метод применяется в тех случаях, когда источниками событий являются не только внешние объекты автоматизации, но и объекты Object RSL. Для этого нужно создать класс Object RSL – наследник стандартного класса ***TRslEvHandler***. В этом случае событие произойдет в результате вызова метода ***raise***, который в качестве первого параметра принимает имя события. Остальные параметры передаются как есть в процедуру обработки события.

Пример:

```
/*Приведем пример класса, который порождает событие onEvent1
при вызове метода Test*/
import rslx;
class (TRslEvHandler)  Source
   macro Test
         raise ("onEvent1", "Это параметр события");
   end;
end;
/* Объект-обработчик выглядит так же, как в примере с Excel */
class (TRslEvHandler)  Handler  ( ob )
   EvSource ("OB") = ob;
```

```
    macro OB_onEvent1 ( par )
        MsgBox  ("Событие OnEvent1 с параметром: ", par)
    end;
end;
ob = Source;
ev = Handler (ob);
ob.Test
```

Примеры использования класса TRslEvHandler

В примерах будем обрабатывать событие NewWorkBook с идентификатором 1565 приложения Microsoft Excel.

Пример 1:

```
/* Используем глобальный объект RslEvHandler, создаваемый
автоматически. */
import rslx;
ob = ActiveX ("Excel.Application");
ob.Visible = true;
RslEvHandler.evSource ("Excel", "Application") = ob;
ob.Workbooks.add;
macro Excel_NewWorkBook (wb)
  println ("New Book: ",wb.Name)
End;
MsgBox ("OK");
RslEvHandler.evSource ("Excel") = NULL;
ob.Quit;
```

Пример 2:

```
/* Создадим специальный класс для обработки событий и
отдельный экземпляр класса TRslEvHandler. */
import rslx;
class THandler
  macro Excel_NewWorkBook (wb)
    println ("New Book: ",wb.Name)
  End;
end;
handler = TRslEvHandler (THandler);
ob = ActiveX ("Excel.Application");
ob.Visible = true;
handler.evSource ("Excel", "Application") = ob;
ob.Workbooks.add;
MsgBox ("OK");
handler.evSource ("Excel") = NULL;
ob.Quit;
```

Пример 3:

```
/* Создадим специальный класс для обработки событий и
отдельный экземпляр класса TRslEvHandler как свойство класса
обработчика. Чтобы не было циклических ссылок, вторым
параметром конструктору TRslEvHandler передаем false. */
import rslx;
class THandler
  evObj = TRslEvHandler (this, false);
  macro Excel_NewWorkBook (wb)
    println ("New Book: ",wb.Name)
  End;
end;
handler = THandler;
ob = ActiveX ("Excel.Application");
ob.Visible = true;
handler.evObj.evSource ("Excel", "Application") = ob;
ob.Workbooks.add;
MsgBox ("OK");
handler.evObj.evSource ("Excel") = NULL;
ob.Quit;
```

Пример 4:

```
/* Создадим специальный класс для обработки событий, наследуя
его от класса TRslEvHandler. */
import rslx;
class (TRslEvHandler) THandler
  macro Excel_NewWorkBook (wb)
    println ("New Book: ",wb.Name)
  End;
end;
handler = THandler;
ob = ActiveX ("Excel.Application");
ob.Visible = true;
handler.evSource ("Excel", "Application") = ob;
ob.Workbooks.add;
MsgBox ("OK");
handler.evSource ("Excel") = NULL;
ob.Quit;
```

Пример 5:

```
/* Создадим специальный класс для обработки событий, наследуя
его от класса TRslEvHandler. В отличие от примера 4, применяем
вторую форму использования evSource, не использующую
обращений к библиотеки типов. Такая форма может применяться
и для остальных примеров. */
import rslx;
class (TRslEvHandler) THandler
```

```
  macro Excel_NewWorkBook (wb)
    println ("New Book: ",wb.Name)
  End;
end;
handler = THandler;
ob = ActiveX ("Excel.Application");
ob.Visible = true;
handler.evSource ("Excel", "{00024413-0000-0000-C000-
000000000046}", 1) = ob;
handler.SetHandler ("Excel", "Excel_NewWorkBook", 1565);
ob.Workbooks.add;
MsgBox ("OK");
handler.evObj.evSource ("Excel") = NULL;
ob.Quit;
```

Пример 6:

```
/* Рассмотрим пример  использования Object RSL в качестве
скрипта на HTML-странице. Здесь обрабатывается событие
onClick, закрепленное за кнопкой, которая размещена на HTML-
странице. Как и в случае с Excel, для связи источника с
обработчиком событий используется глобальная переменная
RslEvHandler. */
<HTML>
<BODY>
<P>  Демонстрация обработки сообщения onClick </P>
<BUTTON ID="CmdButton" STYLE="margin:20%;padding:40">
Press Me
</BUTTON>
<SCRIPT LANGUAGE="RSLScript">
macro CmdButton_onClick
  MsgBox ("Нажата кнопка ",event.srcElement.id );
end;
RSLEvHandler.EvSource ("CmdButton") = CmdButton;
</SCRIPT>
</BODY>
</HTML>
```

В качестве обработчиков событий от элементов объектной модели Internet explorer можно также использовать ссылки на процедуру и ссылки на метод объекта.

Пример:

```
<HTML>
<BODY>
<P>  Демонстрация обработки сообщения onClick </P>
<BUTTON ID="CmdButton1" STYLE="margin:10%;padding:40%">
Button1
</BUTTON>
<BUTTON ID="CmdButton2" STYLE="margin:10%;padding:40%">
```

```
Button2
</BUTTON>
<SCRIPT LANGUAGE="RSLScript">
class HandlerClass
 macro OnClick
   MsgBox ("Нажата кнопка ",event.srcElement.id );
 end;
end;
var handler = HandlerClass;
macro RslTestOnClick
   MsgBox ("Нажата кнопка ",event.srcElement.id );
end;
/* Используем в качестве обработчика глобальную процедуру */
CmdButton1.onclick = @RslTestOnClick;
/* Используем в качестве обработчика  метод объекта */
CmdButton2.onclick = R2M(handler,"OnClick");
</SCRIPT>
</BODY>
</HTML>
```

Класс RslTimer

Класс ***RslTimer*** предназначен для вызова пользовательских обработчиков событий через заданные интервалы времени, т.е. позволяет использовать таймеры. Для работы таймера необходимо, чтобы поток управления находился в состоянии обработки пользовательского ввода.

> ***Внимание!***
> *Методы класса, в которых используются таймеры, не выполняются в RSL модулях, исполняемых на терминале при помощи RSCOM сервера rcwhost.d32.*

Точность вызова таймера составляет 1 секунду. В случае необходимости большей точности срабатывания таймера, следует использовать системный таймер. Таймер, время которого равно нулю, вызывается сразу перед входом потока в состояние ожидания пользовательского ввода. Такой таймер предназначен только для однократного срабатывания, его следует удалить в обработчике таймера при первом срабатывании.

Класс ***RslTimer*** вызывается без параметров:

```
RslTimer ()
```

Методы класса *RslTimer*

Класс ***RslTimer*** содержит следующие методы:

SetTimer (timeout:Integer, id:Integer, handler:Variant [, isSys:Bool]):Bool

Метод позволяет установить новый таймер.

Параметры:

timeout – время в миллисекундах, через которое должен срабатывать таймер.

id – произвольный идентификатор задаваемого таймера. Введенное значение передаётся в пользовательский обработчик и в метод ***RemTimer***.

handler – пользовательский обработчик. В качестве обработчика могут быть заданы: имя глобальной или локальной процедуры, переменная типа V_PROC или V_R2M.

isSys – признак использования системного таймера. Возможные значения:

- TRUE – системный таймер используется.
- FALSE – системный таймер не используется.

Примечание.
Указанный признак поддерживается только на платформе Win32.

Возвращаемое значение:

Метод возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

RemTimer (id:Integer):Bool

Метод позволяет удалить таймер с идентификатором ***id***. Допускается вызывать этот метод из обработчика таймера.

Пример 1:

```
class TestTimer
  var count = 1;
  var t = RslTimer ();
  t.setTimer (2000, 1, R2M(this, "TimerProc"));
  t.setTimer (5000, 2, R2M(this, "TimerProc"));
  macro TimerProc (ind)
    message ("Timer id = ", ind, ", count = ", count);
    count = count + 1;
        if (count == 10)
      t.remTimer (1);
    end;
  end;
end;
var ob = TestTimer;
MsgBox (1);
```

Пример 2:

```
class (TForm) MyForm (owner:object, nm:string)
  InitTForm(owner,nm);
  setTemplate("alpha4.lbr", "FRM001");
    var Button1 = TControl(this, "Button1");
  var Label1 = TControl(this, "Label1");
  var myTimer = RslTimer;
  Button1.addHandler (1, R2M(this, "OnClick"));
```

```
    macro MyTimerProc (id)
        Label1.Text = “Текст установлен из таймера”;
        myTimer.remTimer (1);
    end;
    macro OnClick (Sender: Object)
         myTimer.setTimer (0, 1, R2M(this, "MyTimerProc"));
    end;
end;
```

Передача параметров

Как известно, в Object RSL параметры передаются процедурам по значению. Тем не менее, вызванная процедура может изменить значение фактического параметра в процедуре, ее вызвавшей, — это делается при помощи стандартной процедуры ***SetParm*** (см. стр. 180). Что же касается методов объектов автоматизации, то они могут принимать параметры по ссылке. Чтобы передать параметры иным способом, в Object RSL предусмотрены специальные модификаторы (атрибуты):

- ***:i*** – этот атрибут используется для передачи параметров из Object RSL по ссылке;
- ***:iv*** – этот атрибут используется для передачи параметров из Object RSL как ссылок на вариант, так как некоторые методы ActiveX-объектов могут принимать параметры как ссылки на вариант, благодаря чему эти методы могут менять не только значение параметра, но и его тип;
- ***:v*** – этот атрибут используется для передачи параметров по значению в метод ActiveX-объекта. Необходимость в этом атрибуте возникает в Visual RSL, который имеет специальную настройку, согласно которой все параметры передаются методам ActiveX-объектов по умолчанию по ссылке.

Пример:

```
/* Пусть метод объекта автоматизации представляет процедуру
на языке Visual Basic: */
Sub BasicProc (ByRef par1 As String, ByRef par2 As Variant)
        Par1 = "New string";
        Par2 = 10;
End sub

/* При вызове этого метода необходимо передать параметры по
ссылке: переменная Val1 передается как ссылка на строку, а
переменная Val2 - как ссылка на вариант: */
Val1 = "Old string 1";
Val2 = "Old string 2";
Ob.BasicProc (Val1:i, Val2:iv)
```

Рассмотрим порядок передачи методам объектов автоматизации в качестве параметров Object RSL объектов типа ***Tarray***. По умолчанию объекты типа ***Tarray*** передаются, как и любые другие объекты классов Object RSL. Так, в программе на языке Visual Basic массив ***Tarray*** будет выглядеть как объект, у которого есть свойства ***size*** и ***value***. Однако стандарт автоматизации предусматривает для массивов свой тип данных – SAFEARRAY. Поэтому

при необходимости можно сделать так, чтобы при передаче из Object RSL объекта типа ***Tarray*** этот объект автоматически конвертировался в тип данных SAFEARRAY. Для этого нужно будет присвоить свойству ***CvtToSafeArray*** объекта ***Tarray*** значение TRUE.

Пример:

```
Ar = Tarray;
Ar.value (0) = "value for array";
Ar. CvtToSafeArray = true;
```

При передаче в качестве параметра методам объектов автоматизации массив ***Ar*** будет автоматически преобразовываться в тип SAFEARRAY.

Поддержка модулей платформы .NET

В языке RSL реализована поддержка модулей платформы .NET.

Для создания объектов CLR-классов служит процедура ***ClrObj***.

ClrObj (asName:string, className:string) : object;

Процедура создаёт объект класса ***className*** в сборке с именем ***asName***. Закрытые сборки ищутся в каталоге запуска приложения и в подкаталоге с именем CLR.

Доступ к объектам, созданным с помощью этой процедуры, будет только локальным.

Если необходимо создавать или запускать CLR-объект не только локально, но и удаленно, следует воспользоваться RSCOM-технологией: вместо встроенной процедуры ***ClrObj*** использовать стандартный RSCOM-сервер ***Rsclr*** (см. стр. 162).

Объекты, созданные процедурой ***ClrObj***, можно использовать так же, как и ActiveX-объекты, созданные процедурой ***ActiveX***. Например, можно таким же образом обрабатывать события от CLR-объектов.

Пример:

```
ob = ClrObj ("ClassLib1","ClassLib1.Class1");
RslEvHandler.EvSource ("Test") = ob;
ob.Prop1 = "Data for Prop1";
println (ob.Prop1);
v = 10;
println (ob.Method1 (1,@v));
println (v);
ob.Prop1 = "Data for Prop1";
RslEvHandler.EvSource ("Test") = NULL;
macro Test_TestEvent (src, prm)
  println ("Test Event");
end;
macro Test_TestEvent2 (src, prm)
  println ("Test Event2");
end;
// В примере используется сборка с именем ClassLib1. Исходный
текст на/ языке C#: //
```

```
using System;
using System.Runtime.InteropServices;
using System.Reflection;
[assembly: AssemblyVersion("1.0.*")]
namespace ClassLib1
{
        [ClassInterfaceAttribute(ClassInterfaceType.AutoDual)]
        public class TestEventArgs : EventArgs
        {
                private readonly string data;
                public TestEventArgs (string parm)
                { this.data = parm; }
                public string EventText
                {
                        get { return data;}
    }
  }
  public delegate void TestEventHandler(object sender, TestEventArgs e);
  [GuidAttribute("3444502B-0E87-414b-8279-D20FB40FA087") ]
  [InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
  public interface Class1Events
  {
    void TestEvent(object sender, TestEventArgs e);
    void TestEvent2(object sender, TestEventArgs e);
  }
        [GuidAttribute("F94F09D3-EDE7-49e1-8B91-A22689153599") ]
        public interface Class1Intf
        {
                int Method1(int p1, ref short p2);
                string Prop1 { get; set; }
        }
  [ClassInterfaceAttribute(ClassInterfaceType.None)]
  [ComSourceInterfacesAttribute("ClassLib1.Class1Events,ClassLib1")]
        public class Class1 : Class1Intf
        {
    public event TestEventHandler TestEvent;
    public event TestEventHandler TestEvent2;
    protected virtual void OnTestEvent (TestEventArgs e)
    {
      if (TestEvent != null)
      {
        TestEvent (this, e);
      }
    }
    protected virtual void OnTestEvent2 (TestEventArgs e)
    {
      if (TestEvent2 != null)
      {TestEvent2 (this, e);}
    }
    public Class1()
          {
          }
    public int Method1(int p1, ref short p2)
    {
      TestEventArgs e = new TestEventArgs ("This is a data for
TestEvent");
```

```
        OnTestEvent(e);
        short v = p2;
        p2 = 100;
        return p1 + v;
      }
      public string Prop1
      {
        get { return fld1;  }
        set
        {
          fld1 = value;
          TestEventArgs e = new TestEventArgs ("This is a data for
TestEvent2");
          OnTestEvent2(e);
        }
      }
      private string fld1;
    }
}
```

Обработка ошибок, возникающих во время выполнения программы

В Object RSL можно осуществить "перехват" любой ошибки, возникающей во время выполнения, не допустив аварийного завершения RSL-программы. Для этого любая макропроцедура RSL или метод класса может иметь обработчик ошибок. Записывается это следующим образом:

```
Macro Test
  <инструкции RSL>
OnError
  <инструкции для обработки ошибок>
End;
```

Если какая-либо инструкция в теле макропроцедуры ***Test*** или инструкция в процедурах, вызванных из ***Test***, генерирует ошибку во время выполнения, то управление передается на первую инструкцию для обработки ошибок после ключевого слова ***OnError***, и аварийного завершения RSL-программы не происходит.

Если обработчику ошибок нужна информация о произошедшей ошибке, после ключевого слова ***OnError*** в скобках необходимо указать имя переменной (эта переменная может не декларироваться в теле макропроцедуры при помощи ключевого слова VAR), которая после возникновения сбоя получит ссылку на специальный объект класса ***TrslError***, содержащий информацию о этой ошибке.

Объект класса ***TrslError*** имеет следующие свойства:

- ***Code*** – код ошибки.
- ***Message*** – строка, описывающая ошибку.
- ***Module*** – название модуля RSL, вызвавшего ошибку.
- ***Line*** – строка модуля, в которой произошла ошибка.
- ***AxCode*** – код ошибки ActiveX-объекта.
- ***AxMes*** – строка с информацией об ошибке ActiveX-объекта.

Пример:

```
Macro Test2
<инструкции RSL>
OnError ( er )
  MsgBox (er.Message,"| Модуль: ",er.Module,"| строка: ",er.Line)
End;
```

Пример:

```
macro  Demo
   ob = ActiveX ("Test.TestClass");
onError (er)
   println (er.message);
   if (er.AxCode)
      println (er.AxMes);
end
end  /* Demo */
```

В обработчике ***onError*** можно получить информацию об ошибке пользователя при помощи обращения к полю ***err***.

Пример:

```
onError (erObj)
  if (IsEqClass ("MyError", erObj.err))
     println ("Пользовательская ошибка с кодом ", erObj.err.erCode);
  else
     println (erObj.message);
```

Чтобы в обработчике заново сгенерировать ту же самую ошибку для передачи ее другому обработчику в цепочке вызовов макропроцедур или обработчику RSL, необходимо вызвать стандартную RSL-процедуру ***RunError*** без параметров.

В макромодуле, как и в макропроцедуре, можно реализовать обработчик ошибок. Ключевое слово ***end*** при этом необязательно.

Пример:

```
Import rslx;
Macro Test
end;
OnError (er)
  MsgBox (er.Message)
End    /* End может отсутствовать */
```

Передача управления обработчику ***onError*** происходит также при прерывании пользователем RSL-программы по [Ctrl+Break] и при завершении программы процедурой ***Exit***. В этом случае в обработчик передается специальный код ошибки: при нажатии [Ctrl+Break] равный 17, при вызове процедуры ***Exit*** равный 0. При этом в самой программе ошибка при выполнении программы не генерируется.

Автоматическое создание объектов Object RSL

Объекты классов создаются автоматически при первом обращении к декларированной переменной типа класса Object RSL.

Пример:

```
Class TestClass
    Var prop = 10;
End;
Var Ob:TestClass; /* Декларируем переменную типа TestClass */
Println (Ob.prop);  /* Объект автоматически создается */
```

Конструкция WITH

Конструкция WITH применяется для обращения к конкретным экземплярам класса.

Синтаксическая форма конструкции WITH выглядит следующим образом:

```
WITH (идентификатор)
  <список инструкций>
END
```

Идентификатор – это имя переменной, содержащей ссылку на объект RSL-класса.

Список инструкций выполняется для указанного объекта.

Пример:

```
With ( this )
Println ("----------------");
Отчет;
Println ("----------------");
End;
```

В приведенном выше примере для объекта ***this*** выполняется метод ***Отчет*** и печатаются две пунктирные линии.

Организация ввода/вывода

Ввод данных возможен двумя способами:

- интерактивным – ввод данных с клавиатуры;
- из таблиц базы данных, текстовых и файлов формата DBF (см. стр. 110).

Вывод осуществляется:

- На стандартное устройство – файл, в который выводят процедуры ***print, println*** и инструкция вывода. Имя этого файла определяется программой, вызывающей RSL-программу.
- В текстовые файлы, таблицы в базе данных и DBF-файлы.

Спецификаторы форматирования

Спецификаторы форматирования определяют вид, в котором данные выводятся процедурами ***print, println*** и инструкцией вывода. Эти спецификаторы позволяют форматировать как строку в памяти (при помощи процедуры ***string***), так и данные, отображаемые процедурой ***message***.

Список спецификаторов форматирования указывается сразу же после выражения, к которому он относится. Он начинается с двоеточия и не должен иметь пропусков. Спецификаторы в этом списке отделяются друг от друга двоеточиями.

Список спецификаторов форматирования включает в себя:

- число, указывающее ширину поля вывода. Если значение выражения будет меньше заданной ширины, то оно будет дополнено пробелами. Если ширина поля вывода не указана этим спецификатором или специальными символами управляющей строки, а также если задана ширина, равная нулю, значение выражения выводится полностью;
- число, задающее количество цифр после десятичной точки для данных типа Double. Если это значение равно нулю, дробная часть не выводится;
- один или несколько символов форматирования, которые могут задаваться в любом порядке.

При форматировании данных используются следующие символы форматирования:

- **"l"** – выравнивание по левому краю поля вывода; такое выравнивание по умолчанию выполняется при выводе символьных строк.
- **"r"** – выравнивание по правому краю поля вывода; такое выравнивание по умолчанию выполняется при выводе чисел.
- **"c"** – выравнивание по центру поля вывода.
- **"a"** – при выводе чисел применение апострофов для отделения тысяч и так далее.

- **"e"** – выводит значения переменных, содержащих пустые значения, в виде пустой строки. По умолчанию для таких переменных выводится слово ***Undefined***.
- **"m"** – для типа Money вывод значения прописью; для типа Date вывод значения с названием месяца словом; для типа Time вывод сотых долей секунды; для остальных типов данных этот спецификатор игнорируется.
- **"z"** – не выводит нулевые значения переменных типа Integer, Money и Double, а также значений типа V_UNDEF; спецификатор игнорируется, если не определена ширина поля вывода.
- **"f"** – форматирует строку по шаблону, установленному в прикладной системе и соответствующему типу форматируемого параметра. С его помощью, например, форматируются поля типа SNR, используемые для хранения и редактирования номеров лицевых счетов, а также денежные суммы, которые форматируются в "финансовом" формате. Даты с этим спецификатором выводятся с ведущим нулем.
- **"o"** – при использовании с числовыми полями заполняет поле вывода слева нулями, а не пробелами.
- **"s"** – позволяет передать в методы автоматизации COM ссылку на тип Short. Так как язык RSL не поддерживает типа данных Short, по умолчанию передаётся ссылка на Integer.

  Примечание.
  *Для передачи ссылки на **variant** применяется модификатор **"v"**.*

Пример:

```
Var testVal = 10;

Obj.TestMethod (@TestVal:s);  // Передача ссылки на short

Obj.TestMethod (@TestVal:v);  // Передача ссылки на variant
```

- **"x"** – используется при вводе целых чисел в шестнадцатеричном формате.

Для управления выводом значений, длина которых превышает длину поля вывода, дополнительно используются следующие символы форматирования:

- **"t"** – обрезать значение по ширине поля вывода; это действие по умолчанию выполняется при выводе слишком длинных символьных строк.
- **"d"** – вывести в поле вывода символы "*"; так выводятся по умолчанию слишком длинные числа.
- **"w"** – вывести продолжение на следующей строке или строках; это действие выполняется только для слишком длинных значений строкового типа. При этом формат строк продолжения дублируется, строки повторяются до тех пор, пока самое длинное из значений не будет выведено полностью.

  Примечание.
  *Применение спецификатора **"w"** к целочисленному выражению приводит к выводу числа в шестнадцатеричном формате.*

- **"*"** – позволяет задать ширину поля вывода и количество цифр после десятичной точки в виде переменных. Указанный символ используется вместо константного значения спецификатора, при этом значение спецификатора следует указать следующим параметром в списке параметров.

Пример 1:

```
print (a + b:10:5:c)

// В этом примере для значения выражения a+b указаны
//следующие спецификаторы форматирования: ширина поля
//вывода, равная 10 символам, вывод 5 знаков после запятой,
//печать значения переменной по центру поля вывода.
```

Пример 2:

```
start = time;

i = 1000000;

while (i)

  i = i -1

end;

println (“Время выполнения: “, time – start : m)

// Вывод:

Время выполнения:  0:00:06.87

println(date:f); // 1ого сентября 2005 года будет выведено:
01.09.2005
```

Пример 3:

```
val = 134;

println (val:x);
```

Пример 4:

```
Инструкции с константными спецификаторами:

println (123.123456789:20:5);

[#] ("Это длинная строка для вывода на нескольких строках":w:6);

могут быть записаны так:

println (123.123456789:*:*,  20, 5);

[#] ("Это длинная строка для вывода на нескольких строках":w:*,
6);
```

Формирование отчетов с использованием шаблонов

При подготовке отчетов на языке Object RSL используются стандартные классы ***TRepForm*** (см. стр. 70) и ***TPattFieldR*** (см. стр. 73).

Стандартный класс TRepForm

Класс ***TRepForm*** предназначен для формирования отчетов на языке Object RSL с использованием шаблонов, подготовленных в отдельных текстовых файлах. Для использования этого класса в макропрограмму необходимо импортировать DLM-модуль *prnfrm.d32* при помощи директивы ***Import***:

```
Import prnfrm;
```

Файл шаблонов представляет собой текстовый файл, содержащий одну или несколько форм отчета. По умолчанию для этого файла используется расширение **.tpl*. Файл ищется во всех каталогах, указанных для поиска макрофайлов.

Любая форма, входящая в файл шаблонов, имеют следующий вид:

```
<FORM:FormName>
    Набор полей формы
</FORM>
```

Здесь теги *<FORM:FormName>* и *</FORM>* отделяют одну форму от другой в файле шаблона и задают, соответственно, начало и конец формы, а *FormName* означает имя формы. Если в файле шаблона присутствует только одна форма, то теги *<FORM:FormName>* и *</FORM>* в файл включать не нужно.

Полем является последовательность символов ‘#’. Непосредственно перед ним можно задать атрибуты в следующем формате:

```
<!--fieldName:format:pN-->################
```

Атрибуты поля необязательны. К ним относятся:

- ***fieldName*** – имя поля; если задан этот атрибут, то поле является именованным;
- ***format*** – один из следующих атрибутов форматирования содержимого поля:
  - ***c*** – выравнивание по центру поля;
  - ***l*** – выравнивание по левому краю поля;
  - ***r*** – выравнивание по правому краю поля.
- ***pN*** – атрибут, связывающий с полем некоторое число N. Данный атрибут предназначен для вывода данных во внешние источники, например, если данные выводятся в MS Exel, то этот атрибут задает номер колонки таблицы.

Пример:

```
/* Создадим файл шаблона test.tpl, в котором содержатся две
формы TEST1 и TEST2:*/
<FORM:TEST1>
    Test Report
-----------------------------------------
|--<!--id:r:p1-->################--|--<!--Name:c:p3--
>################--|
```

```
-----------------------------------------
</FORM>
<FORM:TEST2>
<!—Sighature1:r:p1-->###############
<!—Signature2:c:p3-->################
</FORM>
```

Для удобства редактирования формы отчета с именованными полями встроенный редактор системы RS-Bank V.6 имеет режим скрытия имен полей, который вызывается с помощью клавиши [F12] (см. раздел "Использование встроенного текстового редактора" Руководства "Общие документы. Стандартные операции").

В процессе формирования отчета необходимые значения в поля формы помещаются при помощи класса ***TRepForm***.

Конструктор класса ***TRepForm*** выглядит следующим образом:

```
TRepForm (fileName:string [, useUnnamed:Bool, formName:String,
delSpace:Bool])
```

Конструктор загружает форму с именем ***formName*** из файла с шаблонами форм с именем ***fileName***. Имя формы следует указывать с учетом регистра. Если файл шаблона содержит только одну форму, то параметр ***formName*** не задается.

Если второй необязательный параметр ***useUnnamed*** задан и равен TRUE, то в качестве полей в панели могут быть использованы неименованные поля. В противном случае неименованные поля игнорируются. По умолчанию этот параметр принимает значение FALSE.

Если параметр ***delSpace*** задан и равен TRUE, то из текста шаблона удаляются все лишние пробельные символы (пробелы, табуляции, переводы каретки).

Класс ***TRepForm*** имеет следующие свойства:

- ***Value*** **-** свойство, позволяющее читать и писать в поле шаблона с именем name или индексом id. Свойство используется следующим образом:

  ```
  Value ( name : string | id : integer )
  ```

- ***Index*** – свойство возвращает индекс поля с именем ***name***. Если поля с указанным именем не существует, возвращает –1. Свойство используется следующим образом:

  ```
  Index ( name : string )
  ```

- ***Field*** – свойство возвращает ссылку на объект класса ***TPattFieldR*** (см. стр. 73), содержащий информацию о поле, которое задается именем ***name*** или индексом ***id***. Свойство используется следующим образом:

  ```
  Field ( name : string | id : integer )
  ```

Класс ***TRepForm*** также имеет метод ***Write***, который используется следующим образом:

```
Write ( [ outObj : object ] )
```

Если параметр ***outObj*** не задан, то метод выводит шаблон формы с заполненными полями в стандартный выходной поток. Если требуется, чтобы формы выводились каким-либо иным способом, то следует создать класс выходного объекта и передать объект этого класса в качестве параметра ***outObj*** методу ***Write***. Объект ***outObj***, в свою очередь, должен иметь два метода с предопределенными именами:

- ***newLine*** – метод предназначен для отделения одной формы от другой;
- ***writeFields*** – метод предназначен для вывода полей формы. Метод используется следующим образом:

  writeFields (name:string, col:integer, data:variant, flags:integer, w:integer, p:integer)

  Параметры метода:

  - ***name*** – имя поля;
  - ***col*** – число, заданное в поле с помощью атрибута ***p***;
  - ***data*** – данные в поле;
  - ***flags*** – флажки;
  - ***w*** – ширина поля;
  - ***p*** – число знаков после точки.

Пример.

```
/* Создадим файл шаблона test.tpl следующего содержания: */
<FORM:TEST1>
      Test Report
-----------------------------------------
|--<!--id:r:p1-->################--|--<!--Name:c:p3-->################--|
-----------------------------------------
</FORM>
/* Пример RSL-программы, использующей формирование отчета при помощи файла шаблона test.tpl. */
import prnfrm;
class TOutObj
  macro newLine
    println;
  end;
  macro writeField (name:string, col:integer, data:variant, flags:integer, w:integer, p:integer)
    println ("name = ",name," col = ",col," data = ", data)
  end
end;
var ob = TRepForm ("test",false,"TEST1");
var out = TOutObj;
ob.value("id") = "123";
```

```
ob.value("name") = "Name1";
ob.Write;
ob.Write(out);
ob.Write(out);
```

Стандартный класс TPattFieldR

Экземпляры класса ***TPattFieldR*** создаются автоматически для каждого поля, заданного в файле шаблоне. Каждый объект содержит информацию для одного поля. Доступ к экземплярам класса осуществляется посредством свойства ***Field*** класса ***TRepForm***.

Свойства класса

Класс ***TPattFieldR*** обладает следующими свойствами:

- ***Value*** – свойство позволяет читать и писать значение в связанное с объектом поле.
- ***Name*** – свойство содержит имя поля и доступно только для чтения.
- ***Attr*** – свойство позволяет читать и устанавливать новые значения для атрибутов этого поля. Возможны следующие значения этого свойства:
  - **1** – выравнивание по левому краю;
  - **2** – выравнивание по правому краю;
  - **3** – выравнивание по центру.

Поддержка интерактивного режима

В состав языка RSL входят процедуры, позволяющие организовать интерактивный режим работы, значительно облегчающий процесс общения с компьютером.

Пользователь имеет возможность создавать меню, предназначенные для выбора конкретного варианта из заранее заданного списка, производить запрос на ввод данных любого типа.

С помощью языка RSL пользователь также имеет возможность организовывать скроллинг по файлам.

Кроме этого, для ввода и корректировки данных можно разработать специальные диалоговые окна. Процедуры ввода данных описаны в разделе, посвященном встроенным процедурам. В этом разделе приведено описание пользовательских меню и диалоговых окон.

Меню

Если разрабатываемая программа имеет несколько вариантов выполнения, то самый простой способ организовать ветвление – дать пользователю возможность выбора варианта из меню.

В Object RSL предусмотрена возможность создавать как вертикальное, так и горизонтальное меню.

Вертикальное меню

Для создания вертикального меню необходимо в программе создать массив строк с вариантами выбора и вызвать процедуру ***menu***.

Пример:

```
array m;
m(0) = “Первый вариант решения”;
m(1) = “Другой вариант”;
m(2) = “Еще один вариант”;

вариант = Menu (m, “Какой вариант желаете выбрать?”);

if (вариант == 0)
  println (“Выбран первый вариант”);
elif (вариант == 1)
  println (“Выбран другой вариант”);
elif (вариант == 2)
  println (“Выбран еще один вариант”);
end;
```

Процедура ***menu*** возвращает индекс выбранного элемента меню.

Горизонтальное меню

Горизонтальное меню создается и сохраняется в специальной библиотеке.

Описание горизонтального меню начинается с ключевого слова ***MENU***, после которого указывается имя меню и комментарий. Далее в блоках ***POPUP-END*** необходимо описать ниспадающее меню. Ключевое слово ***END*** заканчивает описание.

Имя и комментарий для ниспадающего меню указываются после ключевого слова ***POPUP***. Далее внутри блока ***POPUP-END*** следует описать пункты меню ***ITEM*** и разделители пунктов ***DELIM***. Для каждого пункта необходимо задать:

- имя – название пункта меню на экране;
- целочисленный идентификатор – код, используемый для обработки выбранного пункта меню;

  Примечание.

  Целочисленный идентификатор не должен быть равен 256, так как этот код используется для обработки нажатия клавиши ESC.

- комментарий – текст, отображаемый в статус-строке при выборе пункта меню.

Пример:

```
MENU Demo, "Демонстрационное меню"
        POPUP "~К~оманды", "Команды демонстрационного меню"
                ITEM "~П~ечать", 102, "Печать отчета"
                DELIM
                ITEM "~В~ыход", 101, "Окончание работы"
        END
END
```

Для вызова горизонтального меню из RSL-программы используется процедура ***RunMenu***, которая подробно описана в подразделе "Обработка меню" (см. стр. 184).

Диалоговые окна

Если необходимо предоставить пользователю возможность просматривать и корректировать сложную совокупность данных, то лучше всего для этого воспользоваться диалоговыми окнами.

Форма диалогового окна сохраняется в файле-библиотеке с расширением .LBR.

Для языка RSL диалоговое окно выглядит как структура. Чтение значений из полей этой структуры позволяет получать значения, введенные пользователем в поля диалоговой панели. И наоборот, запись в поля структуры новых значений позволяет определить содержимое полей диалоговой панели.

Чтобы в программе можно было работать с диалоговым окном, необходимо определить идентификатор диалога:

```
Record dlg (demo) dialog;
```

Идентификатор ***dlg*** связан с объектом типа RECORD, имеющим название ***demo*** и задающим структуру диалогового окна из библиотеки диалогов. Теперь, при помощи процедуры ***RunDialog*** можно начать работу с диалогом:

```
RunDialog (dlg)
```

Имеется возможность одновременной работы с диалоговыми панелями, хранящимися в разных файлах библиотек ресурсов. Для этого при определении диалога указывается путь к альтернативному файлу библиотеки диалогов.

Пример:

```
Record dlg  (demo, “mydir\\mylib.lbr”) dialog;
```

Идентификатор диалога может также являться объектом класса ***TRecHandler*** или его наследником. Чтобы создать такой идентификатор, необходимо создать экземпляр класса ***TRecHandler***, в качестве первого параметра которого передано имя диалоговой панели из библиотеки диалогов, в качестве второго – имя LBR-файла с диалоговыми панелями, а в качестве третьего – значение TRUE. Если второй параметр не задан, то используется имя библиотеки по умолчанию. Далее следует задать начальные значения полям диалогового окна. Все сопутствующие процедуры (***RunDialog***, ***UpdateFields*** и т.п.) при этом могут принимать ссылку на объект типа ***TRecHandler*** или его наследника.

Пример:

```
dlg = TRecHandler ( "Test", NULL, TRUE );
dlg.rec.name1 = val1;
dlg.rec.name2 = val2;
```

```
…
RunDialog ( dlg );
```

При работе с диалоговым окном определены клавиши перемещения курсора, а также следующие функциональные клавиши:

[Esc] – Завершение работы с диалоговым окном без сохранения введенных пользователем значений.

Примечание.
*Для того чтобы отменить закрытие диалогового окна при нажатии клавиши [Esc], необходимо выполнить обработку сообщения **DLG_KEY**. Процедура обработки сообщений для кода указанной клавиши возвращает значение **CM_IGNORE**.*

[F9] – Завершение работы с диалоговым окном и сохранение введенных значений.

Примечание.
*Для того чтобы отменить закрытие диалогового окна при нажатии клавиши [F9], необходимо выполнить обработку сообщения **DLG_KEY**. Процедура обработки сообщений для кода указанной клавиши возвращает значение **CM_IGNORE**.*

[Enter] – Завершение редактирования текущего поля и перевод фокуса ввода в следующее поле. Если эта клавиша нажимается в последнем поле редактирования, то это эквивалентно нажатию на [F9].

Процедура обработки сообщений

Если необходимо специальным образом обрабатывать нажатия каких-либо клавиш в диалоговой панели, проверять корректность введенной информации и так далее, для разрешения подобных задач пользователь может написать специальную процедуру обработки сообщений. Эта процедура вызывается после того, как пользователь произвел какие-либо действия с диалоговым окном. Обрабатывая эти сообщения, можно влиять на функционирование диалогового окна.

Следует отметить, что процедура ***RunDialog*** генерирует сообщения только в том случае, если пользователем определена специальная процедура для их обработки.

В этом примере процедура обработки сообщений диалогового окна называется ***EvProc***. Ее имя или ссылку на нее необходимо указать вторым параметром при вызове ***RunDialog***:

```
RunDialog (dlg, "EvProc")
RunDialog (dlg,@EvProc)
```

Эта процедура должна быть определена где-либо в тексте программы следующим образом:

```
macro EvProc (obj, cmd, id, key)
```

Параметры этой процедуры имеют следующие значения:

obj – источник данных (тип Object). В качестве параметра может быть указан:

- для процедуры обработки сообщений окна прокрутки – источник, переданный в процедуру ***RunScroll*** или ***AddScroll***;
- для процедуры обработки сообщений диалоговой панели – объект класса ***TRecHandler*** или производный от него.

cmd – код сообщения (каждый код имеет предопределенную константу); тип Integer.

id – номер текущего поля в диалоговой панели или номер колонки в области прокрутки (тип Integer).

key – код нажатой клавиши (тип Integer) появляется в том случае, если код сообщения ***cmd*** равен **DLG_KEY** или **DLG_MOUSE** (см. далее).

Возвращаемое значение процедуры зависит от типа сообщения. Если процедура не обрабатывает сообщение, она возвращает **CM_DEFAULT** или не возвращает ничего, что равносильно значению **CM_DEFAULT**.

Список обрабатываемых сообщений

В разделе представлено описание обрабатываемых значений процедур **RunDialog** (см. стр. 77) и **RunScroll** (см. стр. 78).

Список обрабатываемых сообщений процедуры RunDialog

Список включает следующие сообщения:

- **DLG_PREINIT** – это первое посылаемое сообщение. Указанное сообщение посылается сразу после создания диалоговой панели. При его обработке процедурой ***AddScroll*** создается область прокрутки данных.
- **DLG_INREC** – это сообщение посылается при установке фокуса ввода в текущее поле.
- **DLG_KEY –** это сообщение посылается в том случае, если пользователь нажал клавишу на клавиатуре.
- **DLG_OUTREC** – это сообщение посылается при переносе фокуса из текущего поля. Если необходимо запретить перенос фокуса, из обработчика возвращается значение **CM_CANCEL**.
- **DLG_TIMER –** это сообщение посылается по истечении интервала времени, установленного при последнем вызове процедуры ***SetTimer***.
- **DLG_DESTROY –** это сообщение посылается непосредственно перед удалением диалогового окна с экрана. Указанное сообщение не посылается в процедуру обработки сообщений скроллинга, встроенного в диалоговую панель с помощью процедуры ***AddScroll***. В параметре ***key*** процедуры обработки сообщений передается статус завершения диалогового окна:
  - **0** – закрытие диалогового окна без сохранения изменений; по умолчанию соответствует нажатию клавиши [Esc]. Процедуры ***RunDialog*** и ***RunScroll*** возвращают значение FALSE.
  - **1** – закрытие диалогового окна с сохранением изменений или выбор требуемой записи из окна прокрутки. Для диалогового окна по умолчанию соответствует нажатию клавиши [F9] или возвращаемому значению **CM_SAVE**. Для скроллинга, запущенного с помощью процедуры ***RunScroll***, соответствует значению **CM_SELECT**, возвращаемому при обработке сообщения **DLG_KEY**. Процедуры ***RunDialog*** и ***RunScroll*** возвращают значение TRUE.

- **DLG_MOUSE –** это сообщение посылается, если пользователь использует в своей работе мышь.

  Процедура обработки сообщений получит следующие значения параметра ***key***:

  - **0** – нажатие левой кнопки мыши;
  - **2** – отпускание левой кнопки мыши;
  - **7** – двойной щелчок кнопкой мыши.

- **DLG_SWITCH** – это сообщение посылается один раз в пользовательский обработчик диалоговой панели, созданной с помощью процедуры ***RunDialog***, при переключении между панелью и скроллингом. Если процедура ***EvProc*** указана в качестве значения параметра ***proc*** в процедуре ***RunDialog*** при обработке сообщения **DLG_SWITCH**, то она может вернуть значение **CM_IGNORE** и, тем самым, отменить переключение.

Указанные ниже сообщения посылаются в процедуру обработки после нажатия клавиши, зарегистрированной вызовом процедуры **AddMultiAction**:

- **DLG_MSELSTART –** это сообщение посылается один раз в начале обработки выбранных записей. При его обработке процедурой **GetMultiCount** возвращается количество выделенных записей.
- **DLG_MSEL –** это сообщение посылается для каждой выделенной записи. Обрабатываемая запись становится текущей в объекте ***RsdRecordSet***, переданном в процедурах ***RunScroll*** или ***AddScroll***.
- **DLG_MSELEND –** это сообщение посылается один раз в конце обработки выделенных записей. При его обработке возможно удаление индикатора выполнения.

Список обрабатываемых сообщений процедуры RunScroll

Список включает следующие сообщения:

- **DLG_INIT –** это сообщение посылается после отправки **DLG_PREINIT**, непосредственно перед отображением диалога на экране. Указанное сообщение не посылается в процедуру обработки сообщений скроллинга, встроенного в диалоговую панель с помощью процедуры ***AddScroll***. При его обработке можно произвести необходимую инициализацию и установить фокус ввода в нужное поле. Последняя операция выполняется с помощью процедуры ***SetFocus***.
- **DLG_INLOOP** – это сообщение используется для отслеживания переключения циклов выборки сообщений и посылается при входе диалоговой панели и области прокрутки в цикл выборки сообщений. Указанное отслеживание необходимо для диалоговых панелей со встроенной областью прокрутки. Такие панели состоят из композиции панелей, поэтому для них используется два разных цикла выборки сообщений: цикл выборки сообщений панели и области прокрутки. При переходе фокуса ввода между полями диалоговой панели и встроенной областью прокрутки происходит переключение используемых циклов выборки сообщений и используемых процедур обработки сообщений.

- **DLG_OUTLOOP** – это сообщение используется для отслеживания переключения циклов выборки сообщений и посылается при выходе диалоговой панели и области прокрутки из цикла выборки сообщений. См. описание значения **DLG_INLOOP.**
- **DLG_BUTTON** – это сообщение посылается, если пользователь щелкнул кнопкой мыши на экранной кнопке.
- **DLG_REMFOCUS** – это сообщение посылается перед переносом фокуса из поля, указанного в обработчике.
- **DLG_SETFOCUS** – это сообщение посылается непосредственно перед установкой фокуса ввода в текущее поле.
- **DLG_SAVE** – это сообщение посылается перед запросом на выход из диалога с сохранением внесенных изменений. Указанное действие происходит в случае нажатия клавиши [F9] или в случае, когда обработчик сообщений **DLG_KEY**, **DLG_TIMER** или **DLG_BUTTON** вернул значение **CM_SAVE**.
- **DLG_SWITCH** – это сообщение посылается один раз в пользовательский обработчик скроллинга, созданного с помощью процедуры ***RunScroll***, при переключении между скроллингом и панелью. Если процедура ***EvProc*** указана в качестве значения параметра ***proc*** в процедуре ***RunScroll*** при обработке сообщения **DLG_SWITCH**, то она может вернуть значение **CM_IGNORE** и, тем самым, отменить переключение.

Значение, возвращаемое процедурой обработки сообщений

Значение, возвращаемое процедурой ***EvProc***, зависит от того, какое из сообщений обрабатывается, и влияет на дальнейшую работу с диалоговым окном:

- При обработке сообщения **DLG_PREINIT** процедура ***EvProc*** может вернуть значение **CM_UPDATE_ADDSCROLL** – обновить скроллинг, добавленный с помощью процедуры ***AddScroll***. В этом случае автоматически закрываются все ресурсы, созданные процедурой ***AddScroll***, затем заново вызывается обработчик диалоговых окон с событием **DLG_PREINIT**, и выполняется процедура ***AddScroll***. Следует иметь в виду, что при выводе найденного значения необходимо закрыть и создать заново объект класса **RecordSet**. Обновление скроллинга выполняется при нажатии указанной клавиши.

Пример:

Примерный вид RSL-кода (обработчик для RunDialog):

```
var rs;
...
/*Обработка нажатия клавиш в диалоговом окне*/
macro MsgProc (dlgPanel, cmd, id, key)
...
var CM_FLAG = CM_DEFAULT;
/*возращяемое значение по умолчанию*/
   if  (cmd == DLG_PREINIT)
```

```
      ...
      rs = RsdRecordSet(RsdCommand(str), RSDVAL_CLIENT,
RSDVAL_STATIC );
      ...
      AddScroll ( ..., rs, ... );
    end;
    if( key == K_F5)
      rs.Close();   //закрываем рекордсет
      rs = NULL;    //на всякий случай, разрушаем объект
      CM_FLAG = CM_UPDATE_ADDSCROLL;  //обновить
скроллинг - полная перезагрузка, с вызовом AddScroll на
событии DLG_PREINIT
    end;
  return CM_FLAG;
end;
```

- При обработке сообщения **DLG_INIT** процедурой ***EvProc*** может возвращаться одно из следующих значений:
  - **CM_CANCEL** – завершить работу с диалоговым окном; процедура ***RunScroll*** возвращает значение FALSE.
  - **CM_DEFAULT** – установить фокус ввода на первое поле панели.
  - **CM_IGNORE –** запретить автоматическую установку фокуса ввода; в этом случае фокус ввода следует установить в нужное поле с помощью процедуры ***SetFocus***.
- При обработке сообщения **DLG_KEY** процедура ***EvProc*** может вернуть одно из следующих значений:
  - **CM_CANCEL** – завершить работу с диалогом, не сохраняя введенные данные.
  - **CM_IGNORE** – игнорировать нажатие клавиши или экранной кнопки (это выполняется по умолчанию для кнопки).
  - **CM_INSERT** – выполнить вставку новой записи в область прокрутки. Указанное значение можно использовать для области прокрутки, встроенной в диалоговую панель, т.к. в этом случае вставка записи в область прокрутки по клавише [F9] недоступна – в диалоговой панели эта клавиша используется для выхода из панели с сохранением внесенных изменений.
  - **CM_SAVE** – завершить работу с диалогом и сохранить введенные данные.
  - **CM_SELECT** – закрытие окна прокрутки, запущенного процедурой RunScroll, с выбором текущей записи.
- При обработке сообщений **DLG_OUTREC, DLG_REMFOCUS, DLG_SAVE** может возвращаться значение **CM_CANCEL** – запретить перенос фокуса, а также выйти из диалогового окна (для сообщения **DLG_SAVE**).
- При обработке сообщения **DLG_BUTTON** процедура ***EvProc*** может вернуть одно из следующих значений:

- **CM_CANCEL** – завершить работу с диалоговым окном без сохранения изменений.
- **CM_SAVE** – завершить работу с диалоговым окном и сохранить выполненные изменения.
- **CM_UPDATE_ADDSCROLL** – обновить скроллинг, добавленный с помощью процедуры ***AddScroll***.

♦ При обработке сообщения **DLG_TIMER** процедура ***EvProc*** может вернуть одно из следующих значений:

- **CM_CANCEL** – завершить работу с диалоговым окном без сохранения изменений.
- **CM_SAVE** – завершить работу с диалоговым окном и сохранить выполненные изменения.

♦ При обработке сообщения **DLG_MSELSTART** процедура ***EvProc*** возвращает одно из следующих значений:

- **CM_MSEL_STOP_KEEP** – прервать обработку выделенных записей, сохранив выделение текущей записи.
- **CM_MSEL_STOP_CLEARALL** – прервать обработку выделенных записей и снять выделение со всех записей.

♦ При обработке сообщения **DLG_MSEL** процедура ***EvProc*** может вернуть одно из следующих значений:

- **CM_DEFAULT** – продолжить обработку выделенных записей, сохранив выделение текущей записи.
- **CM_MSEL_CONT_CLEAR** – продолжить обработку выделенных записей, сняв выделение с текущей записи.
- **CM_MSEL_STOP_KEEP**.
- **CM_MSEL_STOP_CLEAR** – прервать обработку выделенных записей, выделение с текущей записи снять.
- **CM_MSEL_STOP_CLEARALL**.

♦ Если процедура ***EvProc*** не обрабатывает никакого сообщения, она должна вернуть **CM_DEFAULT**, то есть выполнить действия по умолчанию.

Если при обработке сообщений полям структуры диалога присваивается новое значение, то для того, чтобы измененные значения отобразились на экране, необходимо вызвать стандартную процедуру ***UpdateFields***.

Для установки фокуса ввода в нужное поле применяется процедура ***SetFocus***. Эта процедура посылает сообщение **DLG_REMFOCUS** в процедуру обработки сообщений от диалоговой панели.

Пример программы для обработки сообщений диалогового окна:

```
macro EvenMacro(dlg, cmd, id, key)
/* Установим фокус ввода на поле Dat */
  if (cmd == DLG_INIT)
    SetFocus (dlg, “Dat”)
    return CM_IGNORE;
```

```
/* Нажата клавиша F3 */
  elif (cmd == DLG_KEY)
    if (key == 317)
      dlg.Int = 500;
      dlg.Lab = “NewLabel”;
      dlg.Str =
        “Это новая строка”;
      UpdateFields (dlg);
/* Нажата клавиша Enter */
    elif ( (key == 13) and
         (id == 6 ) )
      SetFocus (dlg, 0);
      return CM_IGNORE;
    end;
/* Нажата экранная кнопка */
  elif (cmd == DLG_BUTTON)
    /* Выйти с сохранением */
    if (trim (fldname (dlg, id)) == “~O~k”)
      return CM_SAVE;
    /* Выйти без сохранения */
    elif (trim (fldname (dlg, id)) == “~C~ancel”)
      return CM_CANCEL;
   end;
 end;
/* Проверка правильности ввода данных */
 if (cmd == DLG_REMFOCUS)
    if ( (id == 1) and (dlg.Int < 100) )
       MsgBox (“Значение должно быть больше 100”);
       return CM_CANCEL;
    end;
  elif (cmd==DLG_SETFOCUS)
   message (“Focus installed”);
  elif (cmd == DLG_SAVE)
    if (dlg.Int < 100)
      SetFocus (dlg, 1);
       MsgBox (“Значение
       должно быть |>100”));
       return CM_CANCEL;
    end;
  elif (cmd==DLG_DESTROY)
   message (“Dialog is destroyed”);
  end;
  return CM_DEFAULT;
end
```

Пример работы с диалоговым окном в объектно-ориентированном стиле:

```
const K_F3 = 317, K_F4 = 318;
macro MyDlg (dlg,cmd,id ,key)
  return dlg.MsgHandler (cmd,id,key)
end;
class (TRecHandler) TestDialogBase (resName)
  InitTRecHandler (resName,NULL,true);
  macro onF3
    MsgBox ("Default F3 handler");
  end;
  macro onF4
    MsgBox ("Default F4 handler ")
```

```
  end;
  macro MsgHandler (cmd,id,key)
   if (cmd == DLG_KEY)
     if (key == K_F3)    onF3;
     elif (key == K_F4)  onF4;
     end;
   end;
  end;
  macro Run
    RunDialog (this,@MyDlg);
  end;
end;
class (TestDialogBase) TestDialog (par)
  InitTestDialogBase ("Test");
  var prop = par;
  rec.f1 = "Test string";
  rec.f2 = "Other Test string";
  macro onF3
    rec.f1 = prop;
    rec.f2 = "22222222222";
    UpdateFields (this,1);
    MsgBox ("New F3 handler");
    prop = "New prop Value"
  end;
  macro MsgHandler (cmd,id,key)
    trace ("Cmd = ", cmd);
    return MsgHandler (cmd,id,key);
  end;
end;
dlg = TestDialog ("Value for f1 field");
dlg.Run
```

Скроллинг

В качестве источника данных скроллинга могут выступать следующие объекты:

- объект класса ***RsdRecordSet*** из стандартной библиотеки RSD;
- объект класса, производного от инструментального класса ***ToolsDataAdapter***.

С помощью языка RSL пользователь может организовать просмотр на экране Btrieve-файла с возможностью "прокручивания" его вперед и назад.

Для решения такой задачи служит макромодуль ***RslScr***, в котором определены специальные процедуры и переменные, с помощью которых организовывается работа со скроллингом. Подробное описание модуля ***RslScr*** приведено на стр. 190.

Основным компонентом модуля является специальная процедура обработки сообщения от диалоговой панели ***ScrollMes***. Эта процедура обрабатывает нажатия клавиш [PgUp], [PgDown] и клавиш управления курсором для выполнения прокрутки записей из файла.

Для корректной работы процедуре ***ScrollMes*** необходима инициализирующая информация:

- файл, по которому производится скроллинг;
- соответствие полей диалоговой панели полям в файле.

Эта информация задается при помощи процедуры ***SetScroll***.

SetScroll (FILE, DlgName, FileName, …)

Параметр ***FILE*** задает идентификатор файла, по которому будет производиться прокрутка. ***DlgName*** – имя первой колонки в диалоговой панели. ***FileName*** – имя поля в фале ***FILE***, данные из которого должны выводиться в колонку с именем ***DlgName***. Если имя поля в файле совпадает с именем колонки, можно указать вместо имени ***FileName*** значение NULL. Пары ***"DlgName-FileName"*** должны быть указаны для каждой колонки в скроллинге. Процедура ***SetScroll*** должна вызываться перед вызовом ***RunDialog***.

Область прокрутки в диалоговой панели должна быть создана как совокупность колонок. Каждая колонка образуется из набора редактируемых полей. Названия полей в колонке должны быть одинаковыми за исключением окончания. Окончанием в названии поля должен быть номер строки в колонке. Первая строка должна иметь номер 0. Например: ***ColName0*** – имя первой строки в колонке, ***ColName1*** – имя второй строки в колонке и т.д.

Под именем колонки подразумевается имя поля без окончания. В этом случае именем колонки будет ***ColName***. Именно это имя колонки передается процедуре ***SetScroll***.

Таким образом, для создания скроллинга на языке RSL необходимо:

1. С помощью редактора ресурсов создать диалоговую панель.
2. Используя директиву ***import***, импортировать в RSL- программу модуль ***RslScr***.
3. Непосредственно перед вызовом процедуры ***RunDialog*** вызвать макропроцедуру ***SetScroll*** для задания соответствия имен полей в файле и диалоговой панели.
4. Вызвать процедуру ***RunDialog*** и задать процедуру обработки сообщений.
5. В процедуре обработки сообщений вызывать предопределенную процедуру ***ScrollMes*** для всех необработанных сообщений.

Пример.

*Область прокрутки в диалоговой панели состоит из двух колонок с названиями **Blnc** и **NameB**. В этом случае вызов процедуры **SetScroll** может быть таким:*

```
SetScroll ( ff,
            "Blnc",     "Balance",
            "NameB",  "Name_Part")
```

После вызова процедуры ***SetScroll*** запуск на выполнение диалоговой панели с областью прокрутки производится обращением к стандартной процедуре

RunDialog. При этом если не указывается пользовательская процедура обработки сообщений от диалоговой панели, то необходимо указать имя стандартной процедуры ***ScrollMes***. Если создается пользовательская процедура обработки сообщений, то в ней необходимо для всех необработанных сообщений вызывать стандартную процедуру ***ScrollMes***.

Стандартная процедура ***ScrollMes*** обрабатывает нажатие на клавишу [Enter]. При нажатии на эту клавишу диалоговая панель закрывается, выбранная запись в области прокрутки делается текущей записью в файле и процедура ***RunDialog*** возвращает TRUE.

Пользователю необходимо быть внимательным в том случае, когда необходимо отобразить на экране диалог с областью прокрутки из процедуры обработки сообщений от диалоговой панели, так же содержащей область прокрутки. Так как макромодуль ***RslScr*** содержит ряд глобальных переменных, то рекурсивный вызов может уничтожить данные для текущей диалоговой панели. Чтобы избежать этого, нужно сохранять значения глобальных переменных.

Пример:

```
/* Данный пример содержит фрагмент макропрограммы, в
котором рекурсивно используется диалоговая панель с областью
скроллинга. */

var SaveApos,SaveNrec,SaveNfields,SaveScrff , InUse = FALSE;

array saveDlgFields;
array SaveFileFields;

SaveApos = Apos (0);
SaveNrec = Nrec;
SaveNfields = Nfields;
SaveScrff = Scrff;
SaveDlgFields = DlgFields;

SaveFileFields = FileFields;

SetScroll (Scrff,
      "Blnc",      "Balance",
      "NameB",      "Name_Part"
      );

RunDialog (new,"ScrollMes")

Nrec = SaveNrec;
Nfields = SaveNfields;
Scrff = SaveScrff;
DlgFields  = SaveDlgFields;
FileFields = SaveFileFields;
FillDown (dl,SaveApos);
```

Пример.

Данный пример представляет собой текст законченной макропрограммы, демонстрирующей использование области скроллинга в диалоговой панели. Кроме этого, в нем демонстрируется техника замены процедур, которая позволяет организовать отбор записей для показа в области скроллинга.

Так как код из макромодуля *rslscr.mac* для навигации по файлу использует только процедуры ***next***, ***prev***, ***rewind***, то, подменяя эти процедуры на специальные, обеспечивается навигация только по заданному подмножеству записей.

В этом примере по нажатию клавиши [F5] производится запрос максимального и минимального номера счета для отображения в области скроллинга. Предполагается, что в редакторе диалоговых панелей созданы основная панель ***BLNC***, содержащая колонки с именами ***Blnc***, ***NameB*** и ***Num*** и вспомогательная панель ***RBdlg*** для задания интервала счетов. В примере так же показано, как можно использовать процедуру ***UserFill***, которая заменяется на ***MyFill***. Процедура ***MyFill*** заполняет колонку ***Num***, используя первые два символа из колонки ***Blnc***. Колонки ***Blnc*** и ***NameB*** заполняются данными из таблицы с именем "*balance.dbt*" во внутреннем словаре.

```
/*
   Пример использования скроллинга по файлу с
   отбором записей из заданного интервала
*/
import rslscr;  /*  Поддержка скроллинга  */
file ff (balance) key 0;  /* Рабочий файл */
/*  Отображать на экране будем только заданный
   интервал счетов                    */
var  fromB   = "",  /* Минимальное значение  */
     toB   = "";    /* Максимальное значение */
/* ----------------------------------------------------
   Процедуры MyNext, MyPrev, MyRewind будут
   использоваться вместо стандартных next,
   prev,rewind для обеспечения отбора записей
   из заданного интервала
  ----------------------------------------------------*/
var  first = TRUE;
macro MyNext (ff)
  /* Проверка на превышение максимального значения */
  macro Test
    if (toB and (ff.Balance > toB)) return FALSE end;
    return TRUE
  end;
  if (first and fromB)
    first = FALSE;
/* Установка минимального значения для первой записи */
    ff.Balance = fromB;
    if (getGE (ff))   return Test    end
  else
    if (next (ff))  return Test    end
  end;
  return FALSE
end;
macro MyPrev (ff)
/* Проверка достижения минимального значения  */
  macro Test
    if (fromB and (ff.Balance < fromB)) return FALSE;  end;
    return TRUE;
  end;
  if (first and toB)
    first = FALSE;
/* Установка максимального значения для последней записи */
    ff.Balance = toB;
    if (getLE (ff))   return Test    end;
  else
    if (prev (ff))  return Test    end;
```

```
   end;
   return FALSE
end;
macro MyRewind (ff)
  rewind (ff);
  first = TRUE
end;
macro MyFill (dlg)
   var str,i = 0;
   while (i < Nrec)
     str = dlg (FldIndex (dlg,"Blnc" + i));
     dlg (FldIndex (dlg,"Num" + i)) = substr (str,1,2);
     i = i + 1
   end
end;
/* Выполняем замену стандартных процедур next,prev,rewind */
ReplaceMacro ("next","MyNext");
ReplaceMacro ("prev","MyPrev");
ReplaceMacro ("rewind","MyRewind");
ReplaceMacro ("UserFill","MyFill");
record dlg (blnc) dialog;  /* Основной диалог */
record rb (RBdlg) dialog; /* Панель ввода интервала счетов */
const F4 = 318,F5 = 319;
macro MyMesProc (dl,cmd,id,key)
  var row;
  if (cmd == DLG_KEY)
    if (key == F4)
      /*  Сделаем выбранную запись текущей в файле */
      row = FindRow (dl,id);
      if ((row != -1) and GetDirect (Scrff,Apos (row)))
        MsgBox ("Выбран счет " + Scrff.Balance);
      end;
      return CM_IGNORE
    elif (key == F5)
       /* Запрос нового интервала счетов */
       rb.fromB = fromB;
       rb.toB   = toB;
       if (RunDialog (rb))
         fromB = rb.fromB;
         toB   = rb.toB;
         /* Выводим на экран новый интервал счетов */
         rewind (Scrff);
         if (next (Scrff))
           FillDown (dl,GetPos (Scrff))
         end;
       end;
    end;
  end;
  return ScrollMes (dl,cmd,id,key)
end;
SetScroll (ff,
  "Blnc",       "Balance",
  "NameB",      "Name_Part"
);
dlg.Comment = "Тест прокрутки";
dlg.CurDate = date;
```

```
if (RunDialog (dlg,"MyMesProc"))
  println (ff.Balance,"   ",ff.Name_Part);
  println (dlg.Comment,"   ",dlg.Curdate)
end;
```

Кроме перечисленных выше клавиш, используемых при работе со скроллингом, применяются следующие:

[Esc] – Закрытие окна прокрутки без сохранения введенных пользователем значений.

Примечание.
*Для того чтобы отменить закрытие окна прокрутки при нажатии клавиши [Esc], необходимо выполнить обработку сообщения **DLG_KEY**. Процедура обработки сообщений для кода указанной клавиши возвращает значение **CM_IGNORE**.*

[F9] – Вставка новой записи.

Создание источников данных с помощью класса ToolsDataAdapter

Инструментальный класс ***ToolsDataAdapter*** используется для создания произвольных источников данных, не зависящих от библиотеки RSD, при этом процесс создания осуществляется посредством процедуры ***RunScroll***, для которой в качестве источников данных могут выступать следующие объекты:

- объект класса, производного от инструментального класса ***ToolsDataAdapter***.
- объект класса ***RsdRecordset*** из стандартной библиотеки RSD.

Пример инициализации класса ToolsDataAdapter:

```
class ToolsDataAdapter
// Производный класс вызывает этот метод для задания
//объекта, хранящего текущую запись набора данных.
   macro setCurrentRecord (rec: TRecHandler)
   end;
   // Методы, переопределяемые в производном классе:
// Метод возвращает массив с дополнительной информацией о
//визуальных атрибутах колонок:
   macro getColumnsInfo: TArray
   end;
// Методы, предназначенные для получения информации,
//необходимой для отображения окна с информацией об ошибке,
//произошедшей в источнике данных:
  macro getLastStatus: integer
  end;
  macro getFileName: string
  end;
// Методы, используемые для навигации по набору данных:
  macro moveFirst: bool
  end;
  macro moveLast: bool
  end;
  macro moveNext: bool
  end;
  macro movePrev: bool
```

```
    end;
    macro moveToBookmark (bmk): bool
    end;
  // Метод, позволяющий получить закладку для текущей записи:
    macro getBookmark: variant
    end;
  // Методы для модификации данных:
    macro RecordInsert: bool
    end;
    macro RecordUpdate: bool
    end;
    macro RecordDelete: bool
    end;
  end;
```

Примером использования класса ***ToolsDataAdapter*** может служить реализация следующих производных классов, предназначенных для создания источников данных для Btrieve-файла, который представлен классом ***TBFile:***

- ***BtrAdapterBase –*** базовый класс для ***BtrAdapter*** и ***BtrAdapterEx.***
- ***BtrAdapter –*** позволяет только просматривать содержимое существующих полей объекта класса ***TBFile***. В качестве текущей записи класс ***BtrAdapter*** использует непосредственно объект класса ***TBFile***.
- ***BtrAdapterEx*** – позволяет создавать вычисляемые колонки. Класс ***BtrAdapterEx*** создает вспомогательный объект класса ***TRecHandler***, который содержит дополнительные колонки и используется для работы с текущей записью. Между объектами ***TBFile*** и ***TRecHandler*** выполняется обмен информацией.

Для объекта класса ***TBFile***, непосредственно переданного в классы ***BtrAdapter***, ***BtrAdapterEx***, в скроллинг выводятся все поля. Имеется возможность управлять перечнем отображаемых полей. С этой целью необходимо:

- Создать класс, производный от классов ***BtrAdapter*** или ***BtrAdapterEx***, и переопределить в нем метод ***prepareColumns*** указанных классов.
- Каждое поле (колонку), которое должно отображаться в скроллинге, необходимо определить с помощью метода ***AddColumn***, вызванного в методе ***prepareColumns***.
- Для класса с вычисляемыми полями, производного от ***BtrAdapterEx***, следует переопределить метод ***calculate***. В указанном методе необходимо указать правило задания значения вычисляемым колонкам. Значения должны быть помещены в текущую запись, которая хранится в свойстве ***curRecord*** базового класса.

Пример реализации классов BtrAdapterBase, BtrAdapter и BtrAdapterEx:

```
/*Содержимое макрофайла btrdata.mac*/
class (ToolsDataAdapter) BtrAdapterBase (bf)
  InitToolsDataAdapter;
  var bfile = bf;
```

```
  var curRecord;
  var cols;
  macro getColumnsInfo
    return cols
  end;
  macro getLastStatus
    return status;
  end;
  macro getFileName
    return bfile.fileName;
  end;
  macro AddColumn (name, head, width, kind, dec)
   if (cols == null)
      cols = TArray;
   end;
   var cind = cols.size;
   cols.value (cind)       = name;
   cols.value (cind + 1)   = head;
   cols.value (cind + 2)   = width;
   cols.value (cind + 3 )  = kind;
   cols.value (cind + 4 )  = dec;
   cols.value (cind + 5 )  = 0
  end;
  macro prepareColumns
  end;
  macro initCurRecord
    prepareColumns;
    curRecord = bfile
  end;
  macro initAdapter
    initCurRecord;
    setCurrentRecord (curRecord)
  end;
  initAdapter;
end;
class (BtrAdapterBase) BtrAdapter (bf)
  InitBtrAdapterBase (bf);
  macro moveFirst: bool
    bfile.rewind;
    return bfile.Next;
  end;
  macro moveLast: bool
    bfile.rewind;
    return bfile.Prev
  end;
  macro moveNext: bool
    return bfile.Next ()
  end;
```

```
  macro movePrev: bool
    return bfile.Prev ()
  end;
  macro moveToBookmark (bmk): bool
    return bfile.getDirect (bmk)
  end;
  macro getBookmark: variant
    return bfile.getPos ()
  end;
  macro RecordInsert: bool
    return bfile.insert (null, true);
  end;
  macro RecordUpdate: bool
    return bfile.update (null, true);
  end;
  macro RecordDelete: bool
    return bfile.Delete;
  end;
end;
class (BtrAdapterBase) BtrAdapterEx (bf)
  InitBtrAdapterBase (bf);
  var fields;
  var undefTypePresent = false;
  macro AddColumn (name, head, width, kind, dec, type, size, decPoint)
   AddColumn (name, head, width, kind, dec);
   if (fields == null)
     fields = TArray;
   end;
   var find = fields.size;
   fields.value (find)     = name;
   fields.value (find + 1) = type;
   fields.value (find + 2) = size;
   fields.value (find + 3) = decPoint;
   fields.value (find + 4) = width;
   if (type == null)
     undefTypePresent = true
   end
  end;
  macro initCurRecord
    macro setFromMeta (ind, ar)
      for (var i, 0, ar.size / 5 - 1)
        if (fields.value (ind * 5) == ar.value (i * 5))
          fields.value (ind * 5 + 1) = ar.value (i * 5 + 1);
          fields.value (ind * 5 + 2) = ar.value (i * 5 + 2);
          fields.value (ind * 5 + 3) = ar.value (i * 5 + 3);
          if (fields.value (ind * 5 + 4) == null)
            fields.value (ind * 5 + 4) = ar.value (i * 5 + 4)
          end;
```

```
            return
          end
        end;
        RunError ("Не найдено поле " + fields.value (ind * 5));
      end;
      prepareColumns;
      if ((fields == null) or (fields.size == 0))
        curRecord = TRecHandler (bfile.tblName, bfile.GetFldInfo, false);
        return
      end;
      if (undefTypePresent)
        var ar = bfile.GetFldInfo;
        for (var i, 0, fields.size / 5 - 1)
          if (fields.value (i * 5 + 1) == null)
            setFromMeta (i, ar)
          end
        end
      end;
      curRecord = TRecHandler (bfile.tblName, fields, false);
    end;
    macro calculate
    end;
    macro moveFirst: bool
      bfile.rewind;
      var res = bfile.Next;
      if (res)
        copy (curRecord, bfile);
        calculate
      end;
      return res
    end;
    macro moveLast: bool
      bfile.rewind;
      var res = bfile.Prev;
      if (res)
        copy (curRecord, bfile);
        calculate
      end;
      return res
    end;
    macro moveNext: bool
      var res = bfile.Next ();
      if (res)
        copy (curRecord, bfile);
        calculate
      end;
      return res
    end;
```

```
    macro movePrev: bool
      var res = bfile.Prev ();
      if (res)
        copy (curRecord, bfile);
        calculate
      end;
      return res
    end;
    macro moveToBookmark (bmk): bool
      var res = bfile.getDirect (bmk);
      if (res)
        copy (curRecord, bfile);
        calculate
      end;
      return res
    end;
    macro getBookmark
      return bfile.getPos ()
    end;
    macro RecordInsert: bool
      copy (bfile, curRecord);
      return bfile.insert (null, true);
    end;
    macro RecordUpdate: bool
      copy (bfile, curRecord);
      return bfile.update (null, true);
    end;
    macro RecordDelete: bool
      return bfile.Delete;
    end;
  end;
```

Классы ***BtrAdapter, BtrAdapterEx*** и ***BtrAdapterBase*** не входят в состав дистрибутива, но на основе данного примера возможна реализация других пользовательских источников данных для процедуры ***RunScroll***.

AddColumn (name:String, head:String, width:Integer, kind:Integer , dec:Integer, type:Integer, size:Integer, decPoint:Integer)

Метод предназначен для задания колонок, отображаемых в области прокрутки, а также их атрибутов и вычисляемых полей.

Параметры:

name – программное имя колонки. Указанное имя должно соответствовать имени поля в объекте класса ***TBfile***, что необходимо для установления связи колонки с полем. Для задания вычисляемой колонки необходимо, чтобы заданное имя не соответствовало ни одному полю в объекте класса ***TBfile***.

head – отображаемое наименование колонки. Если параметр не задан, используется программное имя колонки, заданное параметром ***name***.

width – начальная ширина колонки (в символах).

kind – тип колонки.

dec – отображаемое количество знаков после точки (для числовых типов данных).

type – тип данных колонки. В качестве значения параметра указывается любой необъектный тип данных RSL. Параметр задается только для вычисляемых колонок.

size – размер поля типа V_STRING в символах. Параметр задается только для вычисляемых колонок.

decPoint – хранимое количество знаков после точки для типа Numeric.

Пример 1:

```
import btrdata, rsd;
class (BtrAdapter) BtrAdapterUser (bf)
  InitBtrAdapter (bf);
  macro prepareColumns
    //        name,      head,    width,  kind,  dec
    AddColumn ("iNumPlan",  "№",        2,     0,    0 );
    AddColumn ("Balance",   null,       10,    0,    0 );
    AddColumn ("Name_Part", null,       80,    0,    0 );
  end;
end;
```

Пример 2:

```
class (BtrAdapterEx) BtrAdapterCalc (bf)
  InitBtrAdapterEx (bf);
  macro prepareColumns
    //        name,      head,    width, kind, dec,     type,   size, decPoint
    AddColumn ("iNumPlan",  "№",        2,     0,    0 );
    AddColumn ("Balance",   null,       10,    0,    0 );
    AddColumn ("Calc",      "Вычислено", 20,    2,    0,    V_STRING, 100,
0);
    AddColumn ("Name_Part", null,       80,    0,    0 );
  end;
  var flag = 0;
  macro calculate
    curRecord.rec.Calc = string (curRecord.rec.Balance) + "  MyCalc " +
flag
  end;
end;
```

Пример 3:

```
cn = RsdConnection("DSN=Northwind;USER ID=sa;PASSWORD=sa");
cmd = RsdCommand (cn,"select * from Customers");
//Создаём источник данных одним из способов/
//ob = RSDRecordset(cmd , RSDVAL_CLIENT, RSDVAL_STATIC );
//ob = BtrAdapter (tbfile ("balance", "w"));
//ob = BtrAdapterUser (tbfile ("balance", "w"));
//ob = BtrAdapterEx (tbfile ("balance", "w"));
ob = BtrAdapterCalc (tbfile ("balance", "w"));
macro SMsgProc (ob, cmd, id, key)
  var a;
  if (cmd == DLG_KEY)
    if (key == 13)
```

```
            return CM_SELECT
          end;
        end;
      end;
      if (RunScroll (ob, 0, null, "Test", @SMsgProc, "Заголовок", "Status",
      false, -1, -1, 60, 30))
        println ("Запись выбрана")
      end;
```

Пример 4:

```
      dlg = TRecHandler ("test2", null, true);
      macro MsgProcScroll (dlg, cmd, id, key)
        if (cmd == DLG_KEY)
          if (key == 319)  // F5
            return CM_INSERT
          end;
        end;
      end;
      macro MsgProc (dlg, cmd, id, key)
        if (cmd == DLG_PREINIT)
           AddScroll (dlg, ob, null, null, @MsgProcScroll, false,  false);
        end;
      end;
      if (RunDialog (dlg, @MsgProc))
         println ("OK")
      end;
```

Работа с файлами и таблицами баз данных

Для работы с файлами и таблицами баз данных в Object RSL предусмотрены следующие конструкции:

- стандартный класс ***Tbfile*** (см. стр. 98);
- стандартный класс ***TRecHandler*** (см. стр. 102);
- конструкции FILE и RECORD (см. стр. 105).

Примечание.
*Для работы с таблицами баз данных предпочтительнее использовать классы **Tbfile** и **TRecHandler**. Конструкция FILE в настоящее время поддерживается для совместимости с ранее разработанными программами.*

Особенности при использовании СУБД Oracle

При работе с прикладными программами, разработанными в компании R-Style Softlab, следует обратить особое внимание на отличие в текстах макропрограмм имен таблиц базы данных и их полей от реальных имен таблиц и полей, видимых при работе в среде СУБД Oracle. Это обусловлено необходимостью обеспечения совместимости с разработанным ранее программным кодом.

В прикладной программе, помимо стандартного описания базы данных, предоставляемого СУБД Oracle, используется внутренний системный словарь, в котором хранится описание структур и ключей всех таблиц данных в специальном формате. С помощью этого словаря устанавливается соответствие между именами реальных таблиц и их полей и именами таблиц и полей, которые используются в текстах макропрограмм.

Обращение к таблицам базы данных в рамках прикладной программы должно осуществляться в соответствии с описанием таблиц во внутреннем словаре. Поэтому при декларировании объектов, представляющих собой ссылки на таблицы базы данных, и передаче их в RSL-процедуру нужно указывать имена таблиц во внутреннем формате, то есть как во внутреннем словаре:

<имя таблицы во внутреннем словаре>.dbt

Реальное имя таблицы в базе данных имеет формат:

d<имя таблицы во внутреннем словаре>_dbt

Имена полей таблиц баз данных, описанные во внутреннем словаре и используемые при написании RSL-программ, в реальных таблицах базы данных имеют вид:

t_<имя поля во внутреннем словаре>

Использование стандартного класса Tbfile

Стандартный класс ***Tbfile*** предназначен для работы с таблицами баз данных и представляет собой объектную альтернативу стандартной конструкции языка FILE.

Синтаксис конструктора ***Tbfile*** имеет следующий вид:

TBfile (dicName:string [, attrStr:String, keyNum:Integer ,btrFlName: String, altDicName:String, cloneName:String]):Object

Параметры конструктора ***Tbfile*** задают следующие значения:

- ***DicName*** – имя внутреннего словаря базы данных. Параметр указывается в том случае, если необходимо использовать словарь, отличный от используемого по умолчанию.
- ***AttrStr*** – строка атрибутов открытия таблицы. В качестве параметра ***AttrStr*** могут быть заданы следующие символы:
  - **R** – открыть таблицу только для чтения (принимается по умолчанию).
  - **W** – открыть таблицу для чтения и записи.
  - **E –** открыть таблицу для чтения и записи в монопольном режиме.
  - **A** – открыть таблицу в ускоренном режиме; данный атрибут используется только для временных таблиц.
  - **C** – открыть таблицу; если требуемая таблица не существует, то она будет создана.
  - **C+** – создать временную таблицу; если таблица с таким именем существует, то она будет замещена вновь созданной.
  - **O** – выполнить клонирование; если клонирование производится не в открытую таблицу, имя требуемой таблицы необходимо задать в параметре ***cloneName***.
  - **P+** – создать постоянную таблицу; если таблица с таким именем существует, то она будет замещена вновь созданной.

  Если этот параметр не задан, таблица открывается только для чтения.

  Для открытия таблицы с переменной частью записи, используемой как поле типа BLOB, дополнительно к перечисленным атрибутам используется символ "**B**".
- ***KeyNum*** – текущий индекс в таблице. Если параметр не указан, текущим индексом является 0. Для организации доступа к записям таблицы в последовательности, задаваемой первичным ключом, необходимо передать в качестве этого параметра значение -1.
- ***btrFlName*** – имя исходной таблицы. Параметр используется при выполнении операции клонирования. Если параметр не задан, используется имя, переданное в параметре ***DicName***. В случае отсутствия расширения в указанном имени, оно считается равным *.dbt*.
- ***altDicName*** – имя альтернативного словаря.
- ***cloneName*** – таблица для записи результата клонирования. Возможны следующие случаи:

- если в имени таблицы не указано расширение, оно считается равным *.dbt*;
- если имя таблицы не задано, используется имя исходной таблицы;
- если таблица с указанным именем не существует, она создается в первом каталоге из списка каталогов поиска таблиц (*.dbt*).

Для создания экземпляра класса необходимо вызвать конструктор ***Tbfile***. В процессе работы конструктора, помимо других действий, таблица (файл) открывается и к ней применяется метод ***rewind***.

Примечание.
При выполнении операции клонирования конструктор возвращает объект, связанный с клонированным файлом. Дальнейшая работа производится с указанным файлом.

Пример:

```
/*Таблица  account открывается только для чтения по нулевому
индексу*/

Ob = Tbfile ("Account.dbt");

/* Таблица account открывается для записи по второму индексу */

Ob = Tbfile ("Account.dbt", "W", 2);

/* Открывается временная таблица test на основании описания
таблицы "person.dbt" из внутреннего словаря */

Ob = TBFile ("person","C+", 0, "test");

/* Открывается постоянная таблица test на основании описания
таблицы "person.dbt" из внутреннего словаря*/

Ob = TBFile ("person","P+", 0, "test");
```

Методы класса Tbfile

Для объектов класса ***TbFile*** определены следующие методы:

AddFilter (cond:String) – с помощью этого метода можно наложить ограничения на выбираемые записи. Параметром является строка, которая добавляется ко всем генерируемым при работе с таблицей SQL-запросам. Строка добавляется без изменений в условие WHERE, поэтому она должна удовлетворять всем требованиям синтаксиса SQL.

DropFilter ():Bool – снимает ограничения, наложенные методом ***AddFilter***. При отсутствии ограничений, наложенных методом ***AddFilter***, метод возвращает значение FALSE.

Пример:

```
file = TBfile ("partyown");
file.AddFilter("t_partykind=3");
.....
file.DropFilter();
```

GetFldInfo ():TArray – метод возвращает объект класса TArray с метаинформацией о полях файла. Размер массива равен количеству полей, умноженному на пять. Таким образом, для каждого поля используются пять последовательных элементов массива, которые содержат (по порядку) следующую информацию:

- имя поля;
- тип;
- размер;
- количество знаков после точки (для числовых полей);
- 0 (зарезервирован).

GetKeyInfo ():TArray – метод возвращает объект класса TArray с метаинформацией о ключах файла. Размер массива равен количеству сегментов ключей, умноженному на пять. Следовательно, для каждого сегмента используются пять последовательных элементов массива, которые содержат:

- номер ключа;
- номер сегмента в ключе;
- номер поля, которому принадлежит сегмент;
- порядок сортировки: 1 – по возрастанию; 0 – по убыванию;
- 0 (зарезервирован).

Пример:

```
var f = tbfile ("balance");
macro PrintInfo (a)
  var sz = a.size / 5;
  var i;
  for (i, 0, sz - 1)
    [######################## | ###################### |
################### | #######]
      (a [i * 5],          a [i * 5 + 1],        a [i * 5 + 2],    a [i * 5 + 3])
  end
end;
var fldInfo = f.GetFldInfo;
printInfo (fldInfo);
[----------------------------------------];
var keyInfo = f.GetKeyInfo;
printInfo (keyInfo);
```

Кроме этих методов, для объектов класса ***Tbfile*** также реализованы методы, аналогичные процедурам для работы с таблицами базы данных:

♦ **next**	♦ **prev**	♦ **getpos**	♦ **getdirect**
♦ **insert**	♦ **update**	♦ **delete**	♦ **getEQ**
♦ **getGE**	♦ **getLE**	♦ **getGT**	♦ **getLT**
♦ **rewind**	♦ **fldnumber**	♦ **fldindex**	♦ **fldname**
♦ **fldoffset**	♦ **Nrecords**	♦ **RecSize**	♦ **VarSize**
♦ **Clear**	♦ **PackVarBuff**	♦ **UnPackVarBuff**	♦ **WriteBlob**
♦ **ReadBlob**	♦ **FileName**		

Все перечисленные процедуры подробно описаны в разделе "Файлы и структуры" (см. стр. 183).

Свойства класса Tbfile

Объект класса ***Tbfile*** имеет следующие свойства:

Rec – ссылка на объект типа RECORD, при обращении к которому осуществляется доступ к полям записи.

KeyNum – текущий индекс в таблице.

OpenMode – код открытия файла. Возможные значения:

- **0** (Normal) – разрешено чтение и запись.
- **-1** (Accelerated) – ускоренный режим открытия Btrieve-файлов. Может использоваться только для работы с временными файлами.
- **-2** (ReadOnly) – разрешено только чтение.
- **-4** (Exclusive) – монопольный доступ к файлу. Поддерживается только для работы с Btrieve файлами.

TblName – имя таблицы.

DicName – имя внутреннего словаря базы данных.

cnvMode – режим автоматической трансляции строковых полей между кодировками ANSI и OEM. Свойство может принимать следующие значения:

- **0** – автоматическая трансляция не выполняется;
- **1** – в базе данных кодировка OEM, в программе – ANSI;
- **2** – в базе данных кодировка ANSI, в программе – OEM.

Пример:

```
/*Пример доступа к полям записи таблицы: */

Ob.rec.Account = "711034";

Ob.rec.Rest = $12345.67;
```

Объекты класса ***Tbfile*** имеют свойство по умолчанию ***item***. Параметр этого свойства указывает имя поля или номер поля записи таблицы в соответствии со структурой, описанной во внутреннем словаре.

Пример:

```
Ob = Tbfile (“account”);

Ob.item (“Sum”) = $123.45;
Ob.item (1) = “Comment”);

Так как item является свойством по умолчанию, то можно
записать и так:

Ob. (“Sum”) = $123.45;
Ob. (1) = “Comment”);
```

Использование стандартного класса TRecHandler

Стандартный класс ***TRecHandler*** представляет собой объектную альтернативу стандартной конструкции языка RECORD.

Объект класса может быть создан следующими способами:

- Структура полей объекта может быть загружена из словаря базы данных или из библиотеки ресурсов (файл LBR).
- Структура может быть задана с помощью конструктора класса.

Конструктор класса может быть вызван одним из следующих способов:

TRecHandler (strName: String, fldInfo: TArray [, shared:Bool]):Object

Конструктор используется для создания объекта класса ***TRecHandler***.

Параметры:

strName – уникальное имя структуры. Если структура с указанным именем уже существует, параметр ***fldInfo*** игнорируется.

fldInfo – массив, задающий поля структуры. Каждое поле структуры определяется пятью последовательными элементами указанного массива, которые содержат следующие сведения:

- Имя поля.
- Тип поля – могут быть заданы константы V_INTEGER, V_MONEY, V_NUMERIC, V_DOUBLE, V_STRING, V_DATE,V_TIME. Если передано другое значение, оно изменяется на V_INTEGER, поле получает размер 2 байта.
- Размер поля – может быть задан только для полей типа V_INTEGER, V_DOUBLE и V_STRING. Для поля типа V_INTEGER размер поля может быть 1, 2, 4 байта. Если передано другое значение, используется размер поля 2. Для поля типа V_DOUBLE – 4 или 8 (по умолчанию). Для поля с типом V_STRING задаётся размер символьной строки с учётом терминирующего нуль-символа. Для остальных типов полей размер поля определяется, исходя из типа поля; при этом переданное значение игнорируется.
- Количество знаков после точки – может быть задано только для полей типа V_NUMERIC и V_MONEY. Для остальных типов полей переданное значение игнорируется.
- Ширина поля (в символах) для вывода информации. Если значение равно -1, поле не отображается.

shared – признак того, что поиск структуры ***strName*** начинается в кэше. Возможные значения:

- TRUE – поиск выполняется в кэше (значение по умолчанию). Если структура найдена, параметр ***fldInfo*** игнорируется.
- FALSE – создается новая структура, в т.ч. и в случае наличия в кэше структуры с тем же именем.

TRecHandler (strName:String [, dicName:String] , isLbr:Bool):Object

Конструктор создает объект класса ***TRecHandler***.

Параметры:

strName – имя структуры в словаре или панели в библиотеке ресурсов.

dicName – имя словаря базы данных или имя библиотеки ресурсов. Если имя не задано, используется словарь или библиотека, установленные в системе по умолчанию.

isLbr – признак, позволяющий определить, наименование какого элемента передано в параметре **strName**: панели или структуры. Возможные значения параметра:

- TRUE – параметр ***strName*** содержит имя панели в библиотеке ресурсов.
- FALSE (или не задан) – параметр ***strName*** указывает на имя структуры в словаре.

Пример:

```
macro AddField (ar, name, tp, sz, dec)
  ar [ar.size] = name;
  ar [ar.size] = tp;
  ar [ar.size] = sz;
  ar [ar.size] = dec;
  ar [ar.size] = 0;
end;

ar = TArray;
AddField (ar, "field1", V_NUMERIC, 0, 10);
AddField (ar, "field2", V_INTEGER, 4, 0);
AddField (ar, "field4", V_STRING, 40, 0);
ob = TRecHandler ("test", ar);
ob.rec.field1 = 12345.782345678;
ob.rec.field2 = 12345.782345678;
ob.rec.field4 = 12345.782345678;
printprops (ob.rec);
println (ob.item("field1"):0:10);
println (ob.item("field2"):0:10);
println (ob.item("field4"):0:10);
```

В отличие от класса ***Tbfile*** (см. стр. 98), класс ***TRecHandler*** не связан непосредственно с таблицей базы данных, однако его объекты могут использоваться для хранения в памяти копии записи из таблицы.

Пример:

```
/* Создается экземпляр класса Tbfile. В качестве параметра
конструктору класса передается таблица, описанная во
внутреннем словаре как  account, открываемая для чтения по
нулевому индексу */;
Ob = Tbfile (“account.dbt”);
```

```
/* Выполняется обращение к методу next */;
Ob.next;
/* Создается экземпляр класса TRecHandler. В качестве
параметра конструктору класса передается структура таблицы,
описанной во внутреннем словаре как account.dbt */;
Rec = TRecHandler (“account.dbt”);
/* Создается копия записи таблицы account.dbt */;
Copy (rec,ob);
```

Объекты ***TRecHandler*** могут использоваться совместно с объектами ***Tbfile***.

Можно "наложить" объект класса ***TRecHandler*** на постоянную или переменную часть записи таблицы. Наложение структуры на запись таблицы предоставляет возможность считывать данные из записи, а так же модифицировать ее данные непосредственно в таблице посредством обращения к полям структуры.

Объекты класса ***TRecHandler*** имеют свойство ***cnvMode***, которое задает режим автоматической трансляции строковых полей между кодировками ANSI и OEM. Свойство может принимать следующие значения:

- **0** – автоматическая трансляция не выполняется;
- **1** – в базе данных кодировка OEM, в программе – ANSI;
- **2** – в базе данных кодировка ANSI, в программе – OEM.

Объекты ***TRecHandler*** имеют метод ***SetRecordAddr***, который позволяет связать существующий объект типа ***TRecHandler*** с объектом ***Tbfile***.

```
SetRecordAddr ( file, ind, offs, isFix );
```

Параметр ***file*** задает ссылку на объект ***Tbfile*** или его наследника. Остальные параметры аналогичны параметрам конструктора.

В результате использования этого метода все процедуры, работающие с конструкцией FILE (***next***, ***prev***, ***SetRecordAddr*** и т.д.), могут принимать в качестве параметра ссылку на объект ***TBfile*** или его наследника. Функции ***fldname***, ***fldindex*** и др., принимающие ссылку на конструкцию RECORD, могут принимать ссылку на объект ***TRecHandler*** или его наследника.

Пример:

```
class (TBfile) MyTb
  InitTBfile ("client.dbt","w",0);
end;
ob = MyTb;
class (TRecHandler) MyRec
  InitTRecHandler ("client.dbt");
end;
obr = MyRec;
obr.SetRecordAddr (ob,0,0,true);
ob.next;
println (obr.rec.name_client);
```

Использование конструкций FILE и RECORD

Конструкции FILE и RECORD используются для объявления объектов типа FILE, DBFFILE, TXTFILE и RECORD.

Эти определения практически одинаковы. Различие между ними заключается в следующем: конструкция FILE по умолчанию определяет объект типа FILE, а конструкция RECORD – объект типа RECORD.

Определение имеет следующий синтаксис:

[local | private] FILE идентификатор (имя объекта [, имя словаря]) [список параметров объекта]

[local | private] RECORD идентификатор (имя объекта [, имя библиотеки диалогов]) [список параметров объекта]

Идентификатор – это имя, через которое можно ссылаться на определяемый объект. В выражении идентификатор преобразуется к переменной типа, характеризующегося кодом V_FILE, V_STRUC, V_TXTFILE или V_DBFFILE, в зависимости от типа определяемого объекта. Благодаря этому, идентификатор объекта можно присваивать переменным и передавать в качестве параметра процедурам.

В круглых скобках указывается:

- **Для таблиц базы данных** (объект типа FILE) – имя таблицы во внутреннем словаре системы с описанием базы данных.
- **Для текстовых и файлов формата DBF** (объекты типа TXTFILE и DBFFILE) – физическое имя файла. Если оно отсутствует, необходимо явно вызвать процедуру ***open*** с указанием имени файла.
- **Для структур** (объект типа RECORD) – имя структуры во внутреннем словаре системы или имя диалоговой панели в библиотеке диалогов (если указан спецификатор DIALOG). Если пользователь использует несколько библиотек диалогов, то после запятой необходимо указать имя библиотеки, в которой описана диалоговая панель. В случае использования одной библиотеки данный параметр можно не указывать.

Если имя объекта включает недопустимые в идентификаторе символы, например, точку (см. стр. 9), его необходимо заключить в кавычки. Таким образом, если необходимо явно указать расширение файла, его имя необходимо заключить в кавычки.

Список параметров определения не обязателен. Он может содержать любое количество разделенных пробелами следующих спецификаторов:

- **SORT <number>** – определение номера ключевого индекса таблицы (файла) (только для объектов типа FILE), где **number** - номер ключа.
- **KEY <number>** – конструкция, эквивалентная спецификатору **SORT**.
- **WRITE** – режим открытия таблицы (файла) на запись.
- **APPEND** – режим открытия текстового файла в режиме дозаписи.

- **MEM** – спецификатор структуры (умолчание для конструкции RECORD).
- **TXT** – спецификатор текстового файла.
- **DBF** – спецификатор файла формата DBF.
- **DIALOG** – параметр, задающий структуру диалогового окна.
- **BTR** – определяет объект типа FILE (умолчание для конструкции FILE).

Так как конструкции FILE и RECORD определяют одинаковые объекты, то справедливы следующие определения:

file id (name) равносильно record id (name) btr;

record id (name) равносильно file id (name) mem;

По умолчанию файлы и таблицы базы данных открываются в режиме только чтения. Чтобы в них можно было вносить изменения, их необходимо открывать в режиме для записи, указав спецификатор WRITE.

Если в таблице базы данных отсутствует номер ключевого индекса, заданный с помощью параметра SORT NUMBER, то используется ключ с номером 0.

Примеры:

1) Пример определения объекта типа FILE для доступа по записи к таблице, описанной во внутреннем словаре как account.dbt:

```
File Счета ("account.dbt") write;
```

или

```
Record Счета ("account") btr write;
```

2) Пример определения объекта типа DBFFILE для доступа к файлу формата DBF demo.dbf:

```
File demo (“demo.dbf”) dbf;
```

или

```
Record demo (“demo.dbf”) dbf;
```

3) Пример определения объекта типа TXTFILE для доступа к текстовому файлу demo.txt:

```
File tfile (“demo.txt”) txt;
```

или

```
Record tfile (“demo.txt”) txt;
```

4) Пример определения объекта типа TXTFILE для доступа по добавлению записей к текстовому файлу demo.txt:

```
File ff (“demo”) txt append;
```

или

```
Record tfile (“demo”) txt append;
```

5) Пример определения объекта типа RECORD, имеющего структуру таблицы, описанной во внутреннем словаре как account.dbt:

```
File rec (account) mem;
```

или

```
Record rec (account);
```

Доступ к полям объекта осуществляется по именам полей, описанных в словаре данных, или по индексу поля (поля нумеруются, начиная с нуля).

Пример:

```
Номер = aa.Account;
Имя   = Клиенты.Name;

Номер = aa(0);
Имя   = Клиенты(10);
```

Для работы с таблицами базы данных можно использовать все процедуры, подробно описанные в разделе "Файлы и структуры" (см. стр. 192).

Для таблиц базы данных возможен поиск записей по ключу при помощи процедур:

- getEQ;
- getGT;
- getGE;
- getLT;
- getLE.

Данные процедуры выполняют поиск записи таблицы, значение ключа для которой:

- равно указанному;
- строго больше указанного;
- больше или равно указанному;
- строго меньше указанного;
- меньше или равно указанному

соответственно перечисленным выше процедурам.

Чтобы задать значение ключа для поиска, необходимо присвоить нужные значения тем полям записи таблицы, которые соответствуют сегментам текущего ключа перед вызовом этих процедур.

Чтобы определить номер ключа, по которому открывается таблица, необходимо при определении таблицы указать спецификатор SORT. По умолчанию используется ключ с номером 0. Для доступа к записям таблицы в последовательности, задаваемой первичным ключом, значение ключа должно быть равно –1.

Пример:

```
/* Пример выборки записей из таблицы с использованием
оптимизации по ключу. Предполагается, что таблица arhdoc
имеет ключ с номером 0 по полю Date_Carry. Программа выбирает

записи, попадающие в заданный диапазон дат.*/

/* Отбор записей по ключу */
file from ("arhdoc.dbt") sort 0;
var  fromdate,    /* Минимальное значение  диапазона*/
     todate;      /* Максимальное значение диапазона*/
var  first = TRUE;
```

```
macro getarh (ff)
  macro Test  /* Проверка на превышение максимального значения */
    if (ff.Date_Carry > todate) first = TRUE;  return FALSE;  end;
    return TRUE;
  end;
  if (first)
    first = FALSE;
    /* Установка минимального значения для поиска по ключу
первой записи */
    ff.Date_Carry = fromdate;
    if (getGE (ff))   return Test;    end;
  else
    if (next (ff))  return Test;    end;
  end
  return FALSE;
end;

/* Установка минимального и максимального значения для отбора
*/
fromdate = date (1,1,1994);
todate   = date (1,4,1994);
var ndoc = 0;
while (getarh (from))
   ndoc = ndoc + 1;
   message (from.Date_Carry,"   ",ndoc);
end;
println (ndoc);
/* Задаем новый интервал */
fromdate = date (1,6,1994);
todate   = date (1,7,1994);
/*  Обрабатываем новую выборку */
while (getarh (from))
  message (from.Date_Carry,"   ",ndoc);
end;
```

Работа с текстовыми файлами

Перед обработкой данных текстового файла необходимо определить его в RSL-программе, используя определение ***FILE***:

Пример:

```
FILE aa ("c:\\out\\data.rep") txt;
```

Данная строка определяет идентификатор aa для текстового файла с именем "*c:\out\data.rep*", доступного только для чтения.

Чтобы определить файл для записи, используется следующее определение:

Пример:

```
FILE aa ("c:\\out\\data.rep") txt write;
```

В этом случае файл создается автоматически. Если файл уже существовал, то он будет перезаписан.

Если необходимо осуществить дозапись в текстовый файл, его необходимо открывать с атрибутом ***append***.

Пример:

```
FILE aa ("MyFile") txt append;
```

Для работы с текстовыми файлами применяются перечисленные ниже процедуры:

- open;
- close;
- next;
- insert;
- rewind.

Все они подробно описаны в разделе "Файлы и структуры" (см. стр. 192).

Можно явно открыть и закрыть файл при помощи процедуры ***open*** и ***close***, соответственно.

Пример:

```
if (open (aa))
  ....
close (aa);
end
```

Текстовый файл автоматически открывается при первой необходимости и закрывается при выходе из области видимости идентификатора файла. Поэтому явно вызывать процедуры ***open*** и ***close*** нет необходимости.

Процедура ***next*** позволяет читать файл строка за строкой. Доступ к прочитанной информации осуществляется через фиксированное имя поля ***str***. В приведенном примере читается файл, и его строки выводятся в выходной поток:

Пример:

```
while (next (aa))
   println (aa.str)
end;
```

Если строки текстового файла состоят из полей, разделенных символами-разделителями, то к ним можно обращаться по индексу. Первое поле имеет индекс 0. Набор символов-разделителей можно задать при помощи процедуры ***setdelim***. По умолчанию разделителями являются символы пробел и табуляция.

Пример:

```
f1 = aa(0);
f2 = aa(1);
f3 = aa(5);
```

Запись в файл осуществляется строка за строкой в конец файла при помощи процедуры ***insert***. Данные для записи определяются при помощи поля с предопределенным именем ***str*** или поля с индексом 0. Допускается передать в качестве второго параметра процедуры ***insert*** непосредственно строку для записи в файл.

Пример:

```
aa.str = "Данные для записи";
insert (aa);
```

или

```
aa(0)  = "Данные для записи";
insert (aa);
```

или

```
insert (aa,"Данные для записи");
```

Последовательность процедур ***rewind*** и ***next*** переустанавливает текущую позицию на начало текстового файла. Она полезна в том случае, если необходимо повторно прочитать файл.

При описании файла можно опустить физическое имя файла (**но не круглые скобки!**). В этом случае необходимо явно вызвать процедуру ***open*** с указанием имени файла:

Пример:

```
File aa () txt;
open (aa,"data.txt");
```

Максимальная длина буфера у конструкции TXT равна 1024 символа. Если необходимо считывать более длинные строки, то необходимо указать максимальный размер строки следующим образом:

```
file aa () txt 2048;
```

В этом примере задается максимальный размер строки, равный 2048 байт. Размер одного поля в строке, тем не менее, не должен превышать 1024 символа.

Работа с файлами в формате DBF

При работе с файлами формата DBF применяются те же принципы, что и для таблиц базы данных. Прежде всего, необходимо определить файл, используя определение ***FILE***:

Пример:

```
FILE aa ("c:\\out\\data.dbf") dbf;
```

Данная строка определяет идентификатор ***aa*** для DBF-файла, доступного только для чтения, с именем *“c:\out\data.dbf”*.

Чтобы определить DBF-файл для записи, используется следующее определение:

```
FILE aa ("c:\\out\\data.dbf") dbf write;
```

Для работы с DBF-файлами применяются перечисленные ниже процедуры:

- **open**
- **close**
- **create**
- **next**
- **insert**
- **update**
- **delete**
- **rewind**
- **getpos**
- **getdirect**
- **fldnumber**
- **fldindex**
- **fldname**
- **clone**

Индексы для DBF-файлов не поддерживаются. Файл читается всегда в физической последовательности. Процедура ***insert*** добавляет новую запись всегда в конец файла. Процедура ***getpos*** возвращает номер записи в DBF-файле.

Если необходимо прочитать запись с конкретным номером, то можно использовать процедуру ***getdirect*** без предварительного вызова ***getpos***. В этом случае вторым параметром для ***getdirect*** необходимо передать номер интересующей записи в DBF-файле. (Первая запись имеет номер 1).

Файл читается запись за записью при помощи процедуры ***next***. Доступ к полям осуществляется по их именам или индексам, аналогично таблицам базы данных.

Данный пример читает весь файл и выводит строки в выходной поток:

Пример:

```
while (next (aa))
  println (aa.name,"  ",aa.id)
end
```

DBF-файл автоматически открывается при первой необходимости и закрывается при выходе из области видимости идентификатора файла. Поэтому явно вызывать процедуры ***open*** и ***close*** нет необходимости.

При описании файла можно опустить физическое имя файла (**но не круглые скобки!**):

```
File aa () dbf;
```

В этом случае необходимо явно вызвать процедуру ***open*** с указанием имени файла:

```
open (aa,"data.dbf");
```

Работа с записями таблиц как с записями переменной длины

В RSL предусмотрено два способа организации доступа к записям таблиц базы данных, имеющим переменную длину записи.

Первый способ подразумевает, что размер переменной части записи ограничивается пределом 64 Kb. В этом случае переменная часть записи считывается в буфер памяти вместе с постоянной частью. Доступ к данным переменной части осуществляется посредством наложения определенной пользователем структуры на переменную часть записи. Для этого используется процедура ***SetRecordAddr***. Работа этой процедуры подробно описана в разделе "Встроенные процедуры" (см. стр. 192)

Процедура ***SetRecordAddr*** позволяет наложить пользовательскую структуру и на постоянную часть записи. Это дает возможность иметь в одном файле записи с разной структурой, но для этого необходимо предусмотреть какой-либо способ идентификации типа записи.

Размер переменной части записи 64Kb соответствует ее теоретическому пределу. Как правило, процедуры системы RS-Bank V.6 работают непосредственно с записями такого размера, но иногда размер переменной части записи необходимо уменьшить. С этой целью ее "упаковывают". Для работы с "упакованными" записями в RSL предусмотрено две процедуры:

- ***PackVarBuff.***
- ***UnPackVarBuff.***

Примеры их использования приведены при описании данных процедур (см. стр. 192).

Второй способ предполагает, что переменная часть записи рассматривается как специальное поле типа BLOB ("двоичный большой объект"), которое не имеет ограничений на размер записываемой информации.

Обычные операции чтения и записи таблицы распространяются только на постоянную часть, а для доступа к полю BLOB применяются специальные процедуры:

- ***WriteBlob.***
- ***ReadBlob.***

Примеры их использования приведены при описании данных процедур (см. стр. 192).

Доступ к источникам данных с помощью библиотеки RSD

Библиотека RSD предназначена для универсального доступа из программ на языке RSL к источникам данных, поддерживающим SQL.

Данный раздел содержит:

- описание библиотеки RSD (см. стр. 112);
- технологию использования RSD в программах на языке RSL (см. стр. 123).

Описание библиотеки RSD

Физически библиотека RSD имеет несколько уровней. Каждый последующий уровень базируется на предыдущем (большую часть функциональности делегирует нижележащему уровню):

- ***Системный уровень***, включающий в себя:
  - ***драйвер источника данных***, который обеспечивает низкоуровневый доступ к сервисам библиотеки; его реализация специфична для источника данных, в настоящее время имеется драйвер для ODBC;
  - ***драйвер универсального набора записей*** (Recordset), который обеспечивает унифицированный навигационный доступ к данным.

Системный уровень реализован в виде *RDDrvO.dll* (драйвер ODBC) и *RDRSet.dll* (драйвер набора записей).

- ***Объектно-ориентированный интерфейс доступа к данным***, не зависимый от конкретного источника. Данный уровень представляет собой C++-классы и реализован в виде *RsdC.dll*.
- ***Интерфейс для доступа к данным из языка RSL***. Данный уровень представляет собой RSL-классы и реализован как DLM *Rsd.d32*.

Объектно-логическая модель библиотеки представлена на схеме, изображенной на *Рис. 1*.

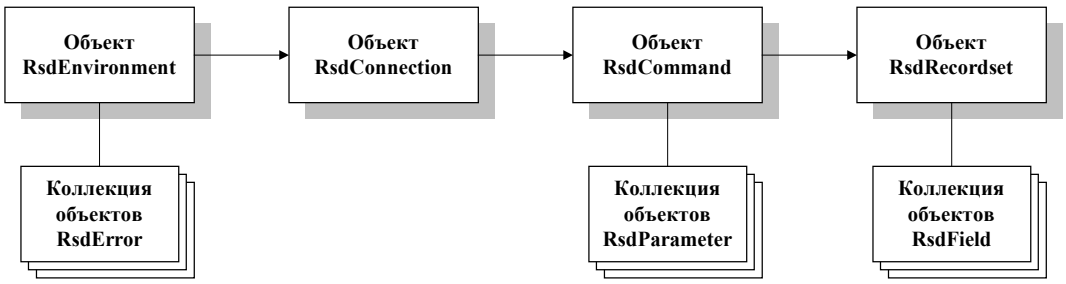

Рис. 1. Объектно-логическая модель библиотеки RSD.

Рассмотрим подробнее каждый из элементов библиотеки.

Класс RsdEnvironment

При использовании RSD первым создается объект окружения ***RsdEnvironment***. Он обеспечивает загрузку нужного драйвера, создание окружения для функционирования других компонент и обработку ошибок.

Конструктор класса вызывается следующим образом:

RsdEnvironment ([driver], [library])

При вызове конструктора используются параметры:

driver – имя интерфейса. По умолчанию используется имя "*RDDrvO*".

library – имя файла драйвера ODBC. По умолчанию используется имя "*RDDrvO.dll*".

В одной программе может быть создано несколько окружений с разными параметрами. Доступ к коллекции ошибок, возникающих при работе объекта, производится с помощью индексированного свойства ***Error***.

В программах на языке RSL нет необходимости явно создавать окружение. Вместо этого можно использовать объект ***RslDefEnv***, который создается по умолчанию и подключается к драйверу для ODBC.

В этом и других объектах библиотеки используется "ленивая" активизация, то есть выполнение функций драйвера откладывается до момента реальной необходимости. То есть окружение фактически открывается только тогда, когда явно вызывается его метод ***Open***, или когда открывается один из объектов, использующих это окружение (это может быть команда при вызове метода ***Execute***).

Свойства класса

Для класса ***RsdEnvironment*** реализованы следующие свойства:

Driver – имя интерфейса (RW|RO). Свойство аналогично одноименному параметру конструктора.

Library – имя файла драйвера ODBC (RW|RO). Свойство аналогично одноименному параметру конструктора.

ErrorCount – количество ошибок в коллекции (RO).

Error (index):RsdError – свойство предоставляет доступ к ошибке с заданным номером из коллекции ошибок (RO).

Здесь и далее в описании свойств классов используются следующие обозначения:

- RO – значение свойства нельзя модифицировать.
- RW – значение свойства можно модифицировать.
- RW|RO – значение свойства можно модифицировать, когда объект не открыт, и нельзя модифицировать для открытого объекта (при попытке модификации генерируется ошибка).

Методы класса

Для класса ***RsdEnvironment*** реализованы следующие методы:

Open ([driver], [library]) – метод открывает окружение. При его вызове можно использовать те же параметры, что и в конструкторе.

Close () – метод закрывает окружение.

ClearErrors () – метод очищает коллекцию ошибок.

Класс RsdConnection

Объект ***RsdConnection*** создает соединение с конкретным источником данных. В одном окружении может существовать несколько соединений с разными параметрами. Каждое соединение характеризуется именем источника данных и (не обязательно) именем пользователя и паролем.

Существует также предопределенный объект класса ***RsdConnection*** с именем ***RslDefCon***.

В качестве строки соединения допустимо указать имя источника данных либо передать полную спецификацию в виде: "*DSN=dsnname; USER ID=username; PASSWORD=userpasswd*". Например:

```
Import RSD;
Env = RsdEnvironment ( “RDDrvO” ); /* явно создается окружение для драйвера ODBC */
Con1 = RsdConnection ( Env, “MyDatabase” ); /* соединение в окружении Env */
Con2 = RsdConnection ( Env, “DSN=MyDB2;USER ID=usr2” ); /* другое соединение в окружении Env, со своими параметрами */
Con3 = RsdConnection ( “MyDatabase”, ”user”, ”secret”); /* используется окружение по умолчанию */
Con4 = RsdConnenction ( “DSN=MyDatabase;USER ID=user;PASSWORD=secret”); /* полная спецификация соединения; используется предопределенное окружение */
```

Конструктор класса может быть вызван следующими способами:

RsdConnection (env, [constr], [user], [password])

RsdConnection (env, [fullconstr])

RsdConnection ([constr,] [user,] [password])

RsdConnection ([fullconstr])

При вызове конструктора используются параметры:

env:RsdEnvironment – объект окружения.

constr – строка соединения, задающая имя DSN.

user – имя пользователя базы данных, осуществляющего соединение.

password – пароль пользователя, осуществляющего соединение.

fullconstr – строка вида "*dsn=database;user id=usr;password=pwd*" или "*custom=customconstr*". Во втором случае строка "*customconstr*" не анализируется, а напрямую передается источнику данных.

Так как свойства объектов RSD устанавливаются не только в конструкторе, то для соединения можно задать другое имя пользователя следующим образом:

```
Con.User = “user2”;
```

Чтобы установить или получить окружение для объекта ***RsdConnection***, можно использовать его свойство ***Environment***.

Объект ***RsdConnection*** осуществляет транзакции. По умолчанию соединение работает в режиме "Autocommit", то есть каждое выполнение команды, использующей это соединение, завершается сохранением изменений в базе данных. В то же время в классе ***RsdConnection*** предусмотрены методы ***BeginTrans***, ***CommitTrans*** и ***RollbackTrans***, с помощью которых можно явно управлять транзакциями.

Свойства класса

Для класса ***RsdConnection*** реализованы следующие свойства:

Environment – имя окружения (RW|RO).

ConString – строка соединения (RW|RO).

User – имя пользователя, осуществляющего соединение (RW|RO).

Password – пароль пользователя, осуществляющего соединение (RW|RO).

Методы класса

Для класса ***RsdConnection*** реализованы следующие методы:

Open () – метод открывает соединение.

Close () – метод закрывает открытое соединение.

BeginTrans () – метод начинает транзакцию.

CommitTrans () – метод завершает транзакцию с сохранением изменений в базе данных.

IsInTrans () – метод возвращает TRUE, если вызывается внутри транзакции, FALSE – в противном случае.

RollbackTrans () – метод выполняет "откат" транзакции.

Класс RsdCommand

Объект класса ***RsdCommand*** представляет собой SQL-запрос к источнику данных. Этот объект инициализируется соединением либо текстом команды и выполняется методом ***Execute***. К одному соединению может быть подключено несколько объектов команды ***RsdCommand***.

Конструктор класса может быть вызван одним из следующих способов:

RsdCommand ([cmdText:String], [cmdType:Integer], [cursorType:Integer])

RsdCommand ([con:Variant], [cmdText:String], [cmdType:Integer], [cursorType:Integer])

RsdCommand ([constring|fullconstring:String], [cmdText:String], [cmdType:Integer], [cursorType:Integer])

Параметры:

con – соединение.

constring – строка соединения, задающая имя DSN.

cmdText – текст команды.

cmdType – тип команды, передаваемой в параметре ***cmdText***. Тип команды задается одной из следующих констант:

- **RsdCmdStoreProc** – в параметре ***cmdText*** передается имя хранимой процедуры.
- **RsdCmdTable** – в параметре ***cmdText*** передается имя таблицы базы данных.
- **RsdCmdText** – в параметре ***cmdText*** передается текст SQL-запроса.

cursorType – тип курсора, который будет создан при выполнении команды. Тип курсора задается одной из констант:

- **RSDVAL_STATIC** – статический курсор, который загружает данные, полученные при выполнении команды, в область памяти целиком.
- **RSDVAL_DYNAMIC** – динамический курсор, который загружает данные в область памяти постранично. Размер страницы устанавливается в свойстве ***BlockSize***. Этот тип курсора рекомендуется использовать при работе с большими объемами данных.
- **RSDVAL_FORVARD_ONLY** – курсор удерживает только одну запись и поддерживает перемещение вперед по набору данных. Этот тип курсора рекомендуется использовать, если запрос создается с целью простого чтения данных без необходимости их дальнейших изменений.
- **RSDVAL_KEYSET_DRIVEN** – курсор, поддерживаемый MS SQL-сервером, который представляет собой разновидность динамического курсора. Такой курсор сохраняет ключ для набора данных и перемещается по нему динамически. Этот тип курсора рекомендуется использовать при работе с большими объемами данных для экономии серверных ресурсов.

fullconstr – строка вида "*dsn=database; user id=usr; password=pwd*" или "*custom=customconstr*". Во втором случае строка "*customconstr*" не анализируется, а напрямую передается источнику данных.

В целях оптимизации команда производит локальное кэширование данных, им можно управлять с помощью свойства ***BlockSize***.

Для задания параметров команды имеется коллекция объектов ***RsdParameter***. Доступ к параметрам команды осуществляется по имени или по номеру (первый параметр в коллекции имеет номер 0).

Пример:

```
Import RSD;

. . . /* здесь определен объект соединения Con */

Cmd1 = RsdCommand( Con,“INSERT INTO clients(id, name)
VALUES(1,’Петров’)” );      /* простая команда, использующая
соединение Con */

Cmd2 = RsdCommand( Con, “INSERT INTO clients(id, name)
VALUES(?,?)” );          /* параметризованная команда, тоже
использующая соединение Con. Для ее успешного выполнения надо
задать значения параметров */

Cmd2.AddParam(“id”, RSDBP_IN, 174 );

Cmd2.AddParam(“name”, RSDBP_IN,‘Петров-Сидоров’ );

Cmd2.Execute; /* выполнение команды */

/* демонстрация различных способов задания параметров */

Cmd2.Value(“id”) = 175; /* использование индексированного
свойства Value */

Cmd2.Param(1).Value = ‘Сидоров’; /* использование коллекции
параметров; можно использовать альтернативный способ
Cmd2.Param(“name”).Value = ‘...’ */

Cmd2.Execute; /* выполнение команды уже с другими значениями
параметров */

Cmd3 = RsdCommand( “MyDb2”, “UPDATE clients SET id=id+1000
WHERE id<100” );     /* задание строки соединения для команды.
Будет создан отдельный объект класса RsdConnection в
предопределенном окружении */

Cmd4 = RsdCommand( “SELECT * FROM clients” );   /*использование
подключения по умолчанию */

Cmd4.CmdText = “SELECT name FROM employees”; /* изменяем
текст команды */

Cmd1.Connection = RslDefCon;     /* теперь Cmd1 использует
предопределенное соединение */
```

Свойства класса

Для класса ***RsdCommand*** реализованы следующие свойства:

Connection – соединение (RW|RO); тип Variant. Свойство аналогично параметру ***con*** конструктора класса (RW|RO).

CmdText – текст команды; тип String. Свойство аналогично одноименному параметру конструктора класса (RW|RO).

ParamCount – количество параметров команды (RO); тип Integer.

Param:RsdParameter (index|name:Integer, String) – параметр команды, заданный индексом ***index*** или именем ***name*** (RO); тип Variant.

Value (index|name:Integer, String) – свойство, обеспечивающее непосредственный доступ к значению параметра, заданного индексом ***index*** или именем ***name*** (RW); тип Variant.

CursorType – тип курсора; тип Variant. Свойство аналогично одноименному параметру конструктора класса (RW|RO).

BlockSize – размер блока данных, которое команда прочитывает за одно обращение к серверу (RW|RO); тип Integer.

NullConversion – признак необходимости конвертирования в NULL специальных значений, используемых в базе данных ИБС RS-Bank V.6 (chr(1), ’01.01.01’); тип Bool.

Методы класса

Для класса ***RsdCommand*** реализованы следующие методы:

Execute ([parm1:Variant [, parm2:Variant]…]) – метод выполняет команду с именованными параметрами ***param1***, ***param2*** и т.д.

AddParam (name:String [, dir:Integer] [, val:Variant], [len:Integer]) – метод добавляет в команду именованный параметр.

Параметры:

- **name** – наименование параметра.
- **dir** – характеристика параметра, которая задается одной из следующих констант:
  - **RSDBP_IN** – входящий параметр.
  - **RSDBP_OUT** – параметр может использоваться для возвращения значения.
  - **RSDBP_RETVAL** – параметр используется для возвращения значения хранимой процедуры, указанной в SQL-запросе.
- **val** – начальное значение параметра. Если параметр ***dir*** принимает значения **RSDBP_RETVAL** или **RSDBP_OUT**, то в параметре ***val*** должен задаваться тип возвращаемого значения (одна из констант V_*).
- **len –** длина строкового параметра. Значение параметра может быть указано в случае, если параметр ***dir*** принимает значения **RSDBP_OUT** или **RSDBP_RETVAL**.

Пример:

```
cmd.addParam("p1", RSDBP_OUT, V_INTEGER)
```

Для строковых параметров также может быть указана длина буфера.

Пример:

```
cmd.addParam("p2", RSDBP_RETVAL, V_STRING, 100)
```

DeleteParam (index|name:Integer, String) – метод удаляет параметр, заданный номером ***index*** либо именем ***name***.

RefreshParams () – для команды с типом ***RsdCmdStoredProc*** метод заполняет коллекцию параметров команды в соответствии с аргументами хранимой процедуры.

Close – метод закрывает команду.

Класс RsdRecordset

Если в результате выполнения команды создается набор записей, возможен навигационный доступ к нему через объект ***RsdRecordset***.

При создании объекта ***RsdRecordset*** имеется возможность непосредственно задать подключение и текст команды, в этом случае объект команды создается неявно. Также вместо подключения можно задать строку соединения. Таким образом, конструктор класса может быть вызван одним из следующих способов:

RsdRecordset ([cmd], [CursorLocation], [CursorType])

RsdRecordset ([cmdText], [CursorLocation], [CursorType])

RsdRecordset ([constring|fullconstring], [cmdText], [CursorLocation], [CursorType])

Конструктор вызывается с параметрами:

cmd:RsdCommand – объект класса ***RsdCommand.*** Параметр передается по ссылке.

cmdText – текст команды. Параметр аналогичен одноименному параметру конструктора класса ***RsdCommand***.

constring – строка соединения, задающая имя DSN.

fullconstring – строка вида "*dsn=database; user id=usr; password=pwd*" или "*custom=customconstr*". Во втором случае строка "*customconstr*" не анализируется, а напрямую передается источнику данных.

CursorLocation – местоположение курсора, которое задается одной из следующих констант:

- **RSDVAL_SERVER** – курсор создается на сервере.
- **RSDVAL_CLIENT** – курсор создается в памяти клиентского приложения.
- **RSDVAL_CLIENT_IF_NEEDED** – если сервер поддерживает серверные курсоры, то курсор создается на сервере, если нет, то в памяти клиентского приложения.

CursorType – тип курсора. Данный параметр аналогичен одноименному параметру конструктора класса ***RsdCommand***.

Примеры создания и инициализации объекта ***RsdRecordset***:

```
Import Rsd;
/* здесь определяются объекты соединения Con и команды Cmd1,
Cmd2 */
Rs1 = RsdRecordset( Cmd1 ); /* создаем объект для получения
набора записей из команды Cmd1 */
```

```
Rs2 = RsdRecordset( Cmd1 );   /* еще один набор, подключенный к
команде Cmd1. По объектам Rs1 и Rs2 можно перемещаться
независимо */
Rs1.Command = Cmd2; /* заново подключаемся к команде Cmd2 */
Rs3 = RsdRecordset( Con, “SELECT * FROM clients” );
/* инициализируем соединением и текстом команды */
Rs4 = RsdRecordset (“MyDatabase”, “SELECT name FROM
employees”);    /* неявно создаются объекты соединения и команды
*/
Rs5 = RsdRecordset( “DSN=MyDb2;USER ID=scott;PASSWORD=tiger”,
“SELECT id, name FROM clients ORDER BY name” );  /* строку
соединения можно специфицировать полностью */
```

Для доступа к значениям и атрибутам полей имеется коллекция объектов ***RsdField***.

Для работы с набором данных объект ***RsdRecordset*** реализует простую навигационную модель – по данным он перемещается с помощью методов ***MoveNext***, ***MovePrev***, ***MoveFirst***, ***MoveLast***. Для получения данных используется либо коллекция полей, либо индексированное свойство ***Value***.

Простейшая процедура получения данных выглядит так:

```
Import RSD;
. . . /* используем объект Rs5 из предыдущего примера */
while( Rs5.MoveNext ) /* итерация по всем записям */
  [###### ####################] ( Rs5.Fld(0).Value,
Rs5.Fld(“name”).Value );      /* используется коллекция полей */
end;
while( Rs5.MovePrev ) /* можно перемещаться и в обратном
направлении */
  [###### ######################] ( Rs5.Value(“id”), Rs5.Value(1) );
/* используется индексированное свойство Value */
end;
```

Свойства класса

Для класса ***RsdRecordset*** реализованы следующие свойства:

Command:RsdCommand – объект команды, с помощью которого получен набор данных (RW|RO).

CursorLocation – местоположение курсора. Свойство аналогично одноименному параметру конструктора класса (RW|RO).

CursorType – тип курсора (RW|RO). Свойство аналогично одноименному параметру конструктора класса ***RsdCommand***.

BookMark – закладка текущего элемента набора данных.

EOF:Bool – признак достижения конца набора данных. Если свойство равно TRUE, то признак установлен (RO).

BOF:Bool – признак выхода за начало набора данных. Если свойство равно TRUE, то признак установлен (RO).

FldCount – количество полей в наборе данных (RO).

Fld (index|name):RsdField – свойство, обеспечивающее доступ к полю с номером ***index,*** либо с именем ***name*** (RO).

RecCount – количество записей в наборе данных (RO). Значение, возвращаемое свойством, зависит от типа курсора:

- если курсор статический, то значение свойства равно количеству записей;
- если курсор динамический, то значение равно –1.

Value (index|name [, nullflag] [, type]) – свойство, обеспечивающее доступ к значению поля с номером ***index,*** либо с именем ***name*** (RW). Свойство также имеет параметры:

- **nullflag** – переменная, в которой возвращается признак нулевого значения SQL, если поле содержит такое значение;
- **type** – код типа данных RSL (константа с префиксом V_...), задающий желаемый тип значения поля.

PageSize – число записей в странице локального кэша, используемого при загрузке данных в область памяти.

MaxPages – максимальное количество страниц локального кэша, загружаемое в память (RW|RO).

Методы класса

Для класса ***RsdRecordset*** реализованы следующие методы:

Open () – метод открывает набор данных.

Close () – метод закрывает набор данных.

Edit ()– начало редактирования текущей записи набора данных.

AddNew () – вставка новой записи в набор данных.

Update () – сохранение изменений в данных.

CancelEdit () – отмена изменений в данных. Метод ***CancelEdit*** применяется для выхода из режима редактирования после неудачного выполнения команды ***Update***.

Delete () – удаление записи из набора данных.

Move (NumRec, MoveDirect) – переход к заданному элементу набора данных. Метод вызывается с параметрами:

- **NumRec** – закладка, либо номер элемента, к которому нужно перейти;
- **MoveDirect** – направление перехода, задаваемое одной из констант:
  - **RELATIVE** – переход на ***NumRec*** элементов относительно текущего элемента.
  - **ABSOLUTE** – переход на ***NumRec*** элементов относительно начала набора данных.
  - **BOOKMARK** – переход к закладке, заданной в параметре ***NumRec***.

MoveNext ():Bool – переход к следующему элементу набора данных.

MovePrev ():Bool – переход к предыдущему элементу набора данных.

MoveFirst ():Bool – переход к первому элементу набора данных.

MoveLast ():Bool – переход к последнему элементу набора данных.

Методы ***MoveNext***, ***MovePrev***, ***MoveFirst*** и ***MoveLast*** возвращают значение TRUE при успешном завершении и FALSE, если произошла ошибка.

Если в процессе редактирования или ввода новой записи возникла ошибка, то процесс останавливается в том же состоянии (ввода или редактирования). Следовательно, следует:

- либо попытаться завершить операцию успешно, тогда выход из режима ввода или редактирования будет выполнен автоматически;
- либо явно выйти из этих режимов – для чего воспользоваться методом ***CancelEdit***.

Пример:

```
Inserter.AddNew( );
Inserter.Value( "MyField" ) = "10";
if (not UpdateCatch( Inserter ))
Inserter.Value( "MyField" ) = "11";
UpdateCatch( Inserter );
// Либо отменить
// Inserter.CancelEdit;
end;
```

Класс RsdError

Класс ***RsdError*** предназначен для доступа к SQL-ошибкам, возникающим в процессе работы с базой данных.

Для класса реализованы следующие свойства:

Descr – описание ошибки (RO).

Code – код ошибки (RO).

Source – тип объекта, в котором произошла ошибка (RO). Свойство может принимать значения:

- **Connection** – ошибка в соединении.
- **Command** – ошибка при выполнении команды.
- **Recordset** – ошибка при работе с набором данных.

SQLState – код состояния SQL, установленный после возникновения ошибки (RO).

NativeError – код ошибки драйвера (RO).

Класс RsdField

Класс ***RsdField*** предназначен для доступа к полям данных, полученных из базы в результате выполнения команд.

Свойства класса

Для класса реализованы следующие свойства:

Name – имя поля (RO).

Value – значение поля (RO).

BlobFilename – имя файла, в котором будет осуществляться чтение/запись содержимого поля типа BLOB (RW).

NullVal – значение, которое будет возвращаться в RSL в качестве нулевого, если в поле содержится нулевое значение SQL (RW).

Методы класса

Для класса реализованы следующие методы:

Read (Var, Count) – чтение ***Count*** байтов из поля типа BLOB и запись в переменную ***Var***.

Read (RecHandler) – чтение значения поля типа BLOB и запись его в структуру типа ***RecordHandler***.

Write (Var, Count) – запись ***Count*** байтов из переменной ***Var*** в поле типа BLOB.

Write (RecHandler) – запись содержимого структуры типа ***RecHandler*** в поле типа BLOB.

При работе с полями типа BLOB следует использовать либо свойство ***BlobFilename***, либо методы ***Read*** и ***Write***.

Класс RsdParameter

Класс ***RsdParameter*** предназначен для доступа к именованным параметрам, используемым при задании SQL-запросов в объекте класса ***RsdCommand***.

Для класса реализованы следующие свойства:

Name – наименование параметра (RO).

Type – тип параметра (RO).

Value – значение параметра (RW).

Direction – характеристика параметра (RW). Свойство аналогично параметру ***dir*** метода ***AddParam*** класса ***RsdCommand***.

Использование RSD в программах на RSL

Типичная схема использования RSD в RSL состоит из следующих шагов:

1. ***Создание объекта окружения*** (необязательный шаг). Если окружение не создается явно, библиотека конструирует нужный объект при первой необходимости. В этой реализации окружение по умолчанию использует ODBC-драйвер.
2. ***Создание объекта соединения*** (необязательный шаг). При отсутствии явно созданного соединения порождается объект со строкой связи RSTest.
3. ***Создание объекта команды*** (необязательный шаг).
4. ***Создание объекта набора записей***, если в результате выполнения команды предполагается получить набор данных.

Таким образом, минимальная программа на RSL с использованием RSD может состоять из строк:

```
import RSD;
cmd = RsdCommand("INSERT INTO T(ID, NAME) VALUES(101,
'RSLTest1')");
```

Обработка ошибок производится самими классами RSL. Фатальные ошибки приводят к аварийному останову. Успешность выполнения метода RSD-класса можно проверить с помощью коллекции ошибок объекта окружения. Также можно применить стандартную технику обработки ошибок:

```
onerror(e)
        println( e.message );
        i = 0;
/* env – объект окружения. Создан в программе ранее или
является окружением по умолчанию: RslDefEnv */
        while( i < env.ErrorCount )
                println( env.error(i).descr );
                i = i + 1;
        end;
```

Рассмотрим примеры использования библиотеки RSD. В примерах будет предполагаться, что в системе создан DSN RSTest и существует таблица, созданная с помощью запроса SQL:

```
CREATE TABLE T (ID INTEGER, NAME VARCHAR(20) )
```

1. Выполнение простой команды (вставка в таблицу).

```
// Явное создание всех объектов.
import RSD;
env = RsdEnvironment( "RDDrvO", "RDDrvO.dll" );   /* создание
окружения */
con = RsdConnection( env, "RSTest" ); /* создание соединения */
cmd = RsdCommand( con, "INSERT INTO T VALUES(99,'RSLTEST')" );
        /* создание команды */
cmd.execute;            /* выполнение команды */
/* блок обработки ошибок: */
i = 0;
while( i<env.ErrorCount )
[error № ##: ########################################](i,
env.error(i).descr);
i=i+1;
end;

// Краткий вариант
import RSD;
cmd = RsdCommand( "RSTest", "INSERT INTO T
VALUES(99,'RSLTEST')" );
cmd.execute;
env = cmd.connection.environment;
/* получение объекта окружения по умолчанию/
```

```
/* блок обработки ошибок: */
i = 0;
while( i<env.ErrorCount )
[error № ##: ########################################](i,
env.error(i).descr);
i=i+1;
end;
/* В последующих примерах предложение import и обработка
ошибок будут опущены.*/
```

2. Выполнение параметризованной команды.

```
cmd = RsdCommand( "DSN=RSTest;USER
ID=username;PASSWORD=passwd",
  "INSERT INTO T VALUES(?, 'testrsl2')" );
cmd.addParam( "id" );
cmd.value("id") = 45;
/* можно было объединить в одну строку:
cmd.addParam("id") = 45 или cmd.addParam("id", NULL, 45) !!! */
/* эквивалентные формы: cmd.value(0)=45,
cmd.param("id").value=45,... */
cmd.execute;
```

3. Выполнение запроса с получением результатов (самая краткая форма).

```
rs = RsdRecordset("DSN=RSTest;USER
ID=username;PASSWORD=passwd", "SELECT ID, NAME FROM T" );
while( rs.moveNext )
        [id: #####  name: ################](rs.value(0), rs.value(1) );
/* эквивалентные формы: rs.fld(0).value, rs.value("id"), ... */
end;
```

Следует отметить, что при задании строки соединения в объектах ***RsdCommand*** и ***RsdRecordset*** для каждого вновь создаваемого объекта будет неявно создаваться новый объект ***RsdConnection***, что может привести к снижению производительности. При неоднократном использовании одного и того же соединения лучшим выходом будет создать один объект ***RsdConnection*** и использовать его во всех командах.

Обработка ошибок

Все процедуры, работающие с таблицами базы данных, при возникновении критических ошибок аварийно завершают выполнение RSL-программы. В остальных случаях они возвращают статус завершения:

- TRUE – в случае успешного завершения.
- FALSE – в случае неуспешного завершения.

Если процедура завершилась неуспешно, для таблиц можно использовать процедуру ***status*** для получения информации об ошибке:

Пример.

```
var ercode,ertext;
if (not next (ff))
    ercode = status (ertext);
    println ("Ошибка (",ercode,"): ",ertext)
end
```

Данный пример иллюстрирует вывод в стандартный выходной поток сообщения об ошибке в случае ее возникновения при выполнении процедуры **next**.

Другой способ обработки ошибок обращения к таблицам данных предполагает определение в тексте программы процедуры с именем ***BtrError***. Формат определения процедуры должен быть следующим:

```
macro BtrError (oper, kod, name)
```

Данная процедура будет вызываться неявно в случае возникновения ошибки, связанной с выполнением операций. Процедура вызывается для всех ошибок, за исключением ошибки "Конец файла" (**BEeof**).

При вызове процедуры ***BtrError***:

- в первом параметре передается код операции;
- во втором – код ошибки;
- в третьем – имя таблицы в базе данных, при обработке которой произошла ошибка.

Процедура ***BtrError*** генерирует следующие коды ошибок:

- 1 – "Код операции".
- 2 – "Ввод/Вывод".
- 3 – "Файл не открыт".
- 4 – "Значение ключа не найдено".
- 5 – "Значение ключа дублируется".
- 6 – "Неверный номер ключа".
- 7 – "Изменился номер ключа".
- 8 – "Неверная позиция".
- 9 – "Конец файла".
- 12 – "Файл не найден".
- 20 – "Менеджер записей не активен".
- 21 – "Мал буфер ключа".
- 22 – "Некорректная длина буфера".
- 28 – "Неверная длина записи".
- 29 – "Неверная длина ключа".
- 36 – "Менеджер не может обрабатывать транзакции".
- 37 – "Транзакция уже начата".
- 39 – "Транзакция не была начата".

- 41 – "Контекстно недопустимая операция".
- 43 – "Неверный номер записи".
- 46 – "Обновление файла, доступного для чтения".
- 59 – "Файл уже существует".
- 80 – "Запись конкурентно изменена".
- 84 – "Запись захвачена".
- 85 – "Файл захвачен".

Процедура должна возвращать значение типа Integer. От возвращаемого значения зависит дальнейшая работа системной процедуры, вызвавшей ошибку:

- **0** – процедура вернет FALSE, указывая на неуспешное завершение (это эквивалентно действию, выполняемому по умолчанию);
- **1** – ошибка игнорируется, процедура вернет TRUE, указывая на успешное завершение;
- **2** – процедура вернет FALSE, указывая на неуспешное завершение, и на экран будет выдано сообщение об ошибке;
- **3** – процедура вернет FALSE, указывая на неуспешное завершение, и выполнение программы будет прервано.

В RSL предусмотрена специальная конструкция ***OnError*** для обработки ошибок, которая позволяет не допустить аварийного завершения RSL-программы в случае возникновения во время ее выполнения ошибки.

Синтаксическая форма конструктора ***OnError*** имеет следующий вид:

```
OnError ['(' ссылка на объект класса TrslError ')']
<инструкции для обработки ошибок>
```

Если любая инструкция в теле макропроцедуры генерирует ошибку времени выполнения, аварийного завершения программы не происходит, а последовательно выполняются инструкции для обработки ошибок, указанные после ключевого слова ***OnError***.

Если в обработчике ошибок нужна информация о случившейся ошибке, то в качестве параметра ***OnError*** необходимо указать имя переменной, которая после возникновения ошибки получит ссылку на специальный объект класса ***TrslError***, содержащий информацию об ошибке.

Примечание.
Переменная, которая в случае сбоя получает ссылку на объект класса TrslError, не должна декларироваться в теле макропроцедуры при помощи ключевого слова var.

Пример:

```
OnError (er)
  MsgBox (er.Message,"| Модуль:",er.Module,"| строка:",er.Line);
```

Объект класса ***TrslError*** содержит следующие свойства:

- ***Code*** – код ошибки.
- ***Message*** – строка, описывающая ошибку.
- ***Module*** – название модуля RSL, вызвавшего ошибку.
- ***Line*** – номер строки модуля, в которой произошла ошибка.

Чтобы в обработчике принудительно сгенерировать некоторую ошибку для передачи ее другому обработчику в цепочке вызова макропроцедур или обработчику RSL по умолчанию, необходимо вызвать стандартную процедуру RSL ***RunError*** без параметров.

В случае, когда необходимо получить сведения об ошибках, возникших при выполнении операции с базой данных, в качестве параметра ***OnError*** следует указать наименование объекта класса ***TDbError***.

Объект класса ***TDbError*** содержит следующие свойства:

- **Stat** – код ошибки СУБД. Соответствует кодам ошибок Btrieve.
- **Oper** – код операции, вызвавшей ошибку. Соответствует операциям Btrieve.
- **Table** – имя таблицы, над которой выполнялась операция.
- **Message** – описание ошибки.
- **ExtMessage** – расширенное описание ошибки.

Пример:

```
onError (er)
  // Printprops (er);
  if (IsEqClass ("TDbError", er.err))
    println ("Код ошибки:      ", er.err.stat);
    println ("Номер операции:  ", er.err.oper);
    println ("Имя таблицы:     ", er.err.table);
    println ("Название ошибки: ", er.err.message);
    println ("Дополнительно:   ", er.err.extMessage);
  end
```

Обработка транзакций

Обработка транзакций в RSL распределена между макросом, написанным пользователем на языке RSL, и вызывающим модулем, написанным на языке C или C++.

Для выполнения транзакции на языке RSL служат процедуры ***ProcessTrn*** и ***LoopInTrn*** (см. стр. 192).

Внимание!
*Следует отметить следующие особенности работы с процедурой **ProcessTrn**:*

- *Процедура **ProcessTrn** всегда выполняется в конкурентном режиме.*
- *Макропроцедуры, выполняемые в рамках транзакции должны иметь минимальное время их выполнения. Иначе это снижает общее быстродействие системы.*

Если во время выполнения транзакции таблица базы данных или запись, к которым необходим доступ, оказываются захваченными другим пользователем, RSL автоматически выполняет серию повторов операции через определенные промежутки времени. Количество повторов и время задержки задаются в файле rsreq.ini:

- OPREPCOUNT – количество повторов операции в случае конкурентного захвата записей.
- OPSLEEPTIME – время задержки между повторами.

Действие процедур выполнения транзакции в случае захвата таблиц или записей может быть изменено с помощью процедуры ***LooplnTrn***. Если эта процедура вызвана перед процедурой выполнения транзакции с параметром, равным TRUE, то в случае блокировки файла или записи процедура выполнения транзакции прервет выполнение транзакции и выполнит ее заново. Количество повторов и время задержки между повторами задается при помощи параметров REPCOUNT и SLEEPTIME.

Вложенные транзакции не поддерживаются.

Пример:

```
/*  В этом примере в транзакции обрабатывается справочник
клиентов. Если отсутствует хотя бы один клиент, выполняется
откат транзакции  */

file acc ("account.dbt");
file cl  ("client.dbt") write;

if (ProcessTrn (NULL,"MyTrn",cl,acc))
  println ("Transaction OK !")
else
  println ("Transaction Aborted")
end;

macro UpdateClient (client)
   cl.Client = client;
   if (getEQ (cl))
     cl.szShortName = cl.Name_Client;
   else
     println ("Клиент ",client," отсутствует в справочнике !");
     AbortTrn
   end
end;

macro MyTrn
  while (next (acc))
    UpdateClient (acc.Client)
  end

end

/* Пример демонстрирует использование процедуры LoopInTrn: */

ob = TBfile ("client.dbt","w",0);

macro TrnProc
  ob.next;
   .......
  ob.update;
end;

LoopInTrn (true);

if (not ProcessTrn (0,"Test"))
  println ("Ошибка в транзакции ", status)

end;
```

Работа с текстовыми и двоичными файлами

В настоящей главе приведено описание стандартных классов языка RSL, используемых для работы с текстовыми и двоичными файлами.

Использование класса TStream

Класс **TStream** используется для работы с двоичными файлами и RSCOM-объектами, поддерживающими интерфейс IrsStream.

Конструктор класса может быть вызван одним из следующих способов:

TStream (filename:String [, openmode:String])

Создаёт объект ***TStream***, связанный с дисковым файлом, заданным именем **filename**.

При вызове конструктора используются параметры:

- **filename** – имя файла.

  Если не задан абсолютный путь, файл создаётся в каталоге заданным параметром настройки TXTFILE.

  Если параметр не задан или равен ***null***, создаётся объект в памяти без связи с физическим файлом.

- **openmode** – текстовая строка, задающая режим открытия.

  Если параметр не задан, используется режим только для чтения. Строка может содержать один из символов:

  - ***R*** – открыть файл на чтение. Файл должен существовать.
  - ***W*** – открыть файл на чтение и запись. Файл должен существовать.
  - ***C*** – создать новый файл и открыть его на запись. Если файл уже существует, он перезаписывается.
  - ***A*** – открыть файл на дозапись. Если файл не существует, он создаётся.

TStream (strm:Object)

Создаёт объект **TStream** для RSCOM-объекта с интерфейсом IRsStream, заданного параметром **strm**. При помощи данного конструктора, можно создать объект для работы с полями типа BLOB источников данных RSD.

Свойства класса

Для класса ***TStream*** реализованы следующие свойства:

Name – свойство возвращает имя файла, если объект был сконструирован при помощи первого варианта конструктора. В противном случае значение свойства – null.

Stream – свойство возвращает RSCOM-объект с интерфейсом IRsStream. Этот объект может быть передан RSCOM-методам, ожидающим объект с данным интерфейсом, например, источникам данных.

Методы класса

Для класса ***TStream*** реализованы следующие методы:

Write (from:Variant [, tp:Integer, sz:Integer, decPoint:Integer]):Bool

Метод записывает данные из переменной **from** в поток. Метод вызывается с параметрами:

- **from** – переменная RSL, данные которой записываются в поток.
- **tp** – тип данных, которые необходимо записать в поток.

  Если значение переменной имеет тип, отличный от заданного, то выполняется автоматическое преобразование типа значения в параметре **from** к типу **tp**. Если параметр **tp** не задан, то тип определяется по типу значения в параметре **from**.

  Поддерживаются следующие типы данных: V_INTEGER, V_DOUBLE, V_NUMERIC, V_MONEY, V_STRING, V_DATE, V_TIME.
- **sz** – размер данных для записи. Для разных типов данных этот параметр может принимать разные значения:
  - *V_INTEGER* – 1, 2, 4. По умолчанию: 4.
  - *V_DOUBLE* – 4, 8. По умолчанию: 8.
  - *V_NUMERIC* – 16.
  - *V_STRING* – максимальное количество символов строки для записи. По умолчанию записывается вся строка.

  Для остальных типов данных размер игнорируется.
- **decPoint** – количество знаков после точки. По умолчанию – 4.

  Используется только для типа данных V_NUMERIC. Поскольку этот формат не хранит позицию десятичной точки, её следует задать явно.

Read (out:@Variant [, tp:Integer, sz:Integer, decPoint:Integer]):Bool

Метод считывает из потока данные типа, заданного параметром **tp**. Вызывается с параметрами:

- **out** – переменная, в которой возвращается прочитанный из потока результат.
- **tp** – тип данных, которые необходимо считать из потока.

  Если тип данных не задан, то по умолчанию используется V_INTEGER.

  Поддерживаются следующие типы данных: V_INTEGER, V_DOUBLE, V_NUMERIC, V_MONEY, V_STRING, V_DATE, V_TIME.
- **sz** – размер данных для чтения.

  Для разных типов данных этот параметр может принимать разные значения. Возможные значения аналогичны значениям, представленным для метода **Write**.
- **decPoint** – количество знаков после точки. По умолчанию – 4.

  Используется только для типа данных V_NUMERIC. Поскольку этот формат не хранит позицию десятичной точки, её следует задать явно.

При достижении конца файла метод возвращает – ***false***.

Write2 (from:Object):Bool

Метод записывает в поток данные из объекта типа TRecHandler, заданного параметром **from**.

Примечание.
Язык RSL позволяет определять структуру полей записи объекта TRecHandler непосредственно в коде RSL без необходимости определять структуру в словаре базы данных.

Read2 (to:Object):Bool

Метод считывает из потока данные и помещает их в объект типа TRecHandler, заданный параметром **to**.

WriteVal (from:Variant):Bool

Метод записывает в поток значение переменной, переданной в качестве параметра **from**. В отличие от метода ***Write***, этот метод сначала записывает тип переменной RSL, а затем – значение переменной. В результате метод ***ReadVal*** восстанавливает из потока значение переменной без указания дополнительной информации о типе.

Методы ***WriteVal*** и ***ReadVal*** удобно использовать для сохранения в потоке значений переменных RSL и последующего их восстановления.

Поддерживаются следующие типы данных: V_INTEGER, V_DOUBLE, V_NUMERIC, V_MONEY, V_STRING, V_DATE, V_TIME.

ReadVal (to:@Variant):Bool

Метод считывает из потока значение переменной, сохранённой методом ***WriteVal***, и помещает значение в переменную, заданную параметром **to**.

Copy (from:Object [, numBytes:Integer):Bool

Метод копирует данные из объекта **from**, которым может быть либо объект класса ***TStream***, либо RSCOM-объект с интерфейсом IRsStream.

Параметр **numBytes** позволяет задать явно количество байт, которое необходимо скопировать из объекта **from**. Если он не задан, данные копируются из объекта **from**, начиная с текущей позиции и до конца потока.

SetPos (pos:Integer [, from:Integer])

Метод устанавливает позицию в потоке в соответствии с параметрами:

- **pos** – позиция в потоке, которую необходимо установить. Интерпретация этого параметра зависит от параметра **from**.
- **from** – параметр определяет как следует интерпретировать значение из параметра **pos**. Если параметр **from** равен:
  - ***0***, **pos** задаёт смещение от начала потока;
  - ***1***, **pos** задаёт смещение от текущей позиции в потоке;
  - ***2***, **pos** задаёт смещение от конца потока.

  Параметр **pos** может принимать отрицательные значения для **from**, равные ***-1*** и ***-2***. Это означает смещение в сторону начала потока.

GetPos:Integer

Метод возвращает текущую позицию в потоке. Позиция представлена смещением от начала потока.

GetSize:Integer

Метод возвращает размер потока.

SetSize (sz:Integer)

Метод устанавливает размер потока равным значению, заданному параметром **sz**.

Flush

Метод сбрасывает на диск не сохраненные изменения.

Использование класса TStreamDoc

Класс **TStreamDoc** применяется для работы с текстовыми файлами в различных кодировках. Этот класс выполняет трансляцию содержимого файла. Он автоматически конвертирует символы между кодировкой, используемой в программе и кодировкой символов в файле.

Конструктор класса может быть вызван одним из следующих способов:

TStreamDoc (filename:String [, openmode:String , encode:String, eolType:Integer])

Создаёт объект ***TStreamDoc***, связанный с дисковым файлом, заданным именем **filename**. При вызове конструктора используются параметры:

- **filename** – имя файла. Если не задан абсолютный путь, файл создаётся в каталоге, заданном параметром настройки *TXTFILE*.

  Если данный параметр не задан, или равен ***null***, создаётся объект в памяти без связи с физическим файлом.

- **openmode** – текстовая строка, задающая режим открытия. Если параметр не задан, используется режим только для чтения. Строка может содержать один из символов:
  - ***R*** – открыть файл на чтение. Файл должен существовать.
  - ***C*** – создать новый файл и открыть его на запись. Если файл уже существует, он перезаписывается.
  - ***W*** – эквивалентно ***C***.
  - ***A*** – открыть файл на дозапись. Если файл не существует, он создаётся.
- **encode** – текстовая строка, определяющая кодировку символов в файле. Может иметь одно из следующих значений:
  - ***rsoem*** – RSOEM кодировка;
  - ***rsansi*** – RSANSI кодировка;
  - ***lcoem*** – LCOEM кодировка;
  - ***lcansi*** – LCANSI кодировка;
  - ***utf8*** – UTF-8 кодировка;
  - ***utf16le*** – UTF-16LE кодировка;
  - ***utf16be*** – UTF-16BE кодировка.

Если этот параметр не задан, то по умолчанию в качестве кодировки символов используется кодировка, заданная в глобальных настройках прикладной системы (по умолчанию ***rsoem***).

Если открывается существующий Unicode-файл на чтение или дозапись, то его кодировка определяется из сигнатуры в начале файла и значение параметра **encode** игнорируется.

- **eolType** – стиль завершения строки при записи в текстовый документ. Параметр принимает следующие значения:
  - **0** – стандарт DOS и Windows (CR LF) – значение по умолчанию;
  - **1** – стандарт UNIX (LF);
  - **2** – стандарт MAC (CR).

TStreamDoc (strm:Object [, encode:String, eolType:Integer])

Создаёт объект ***TStreamDoc*** для RSCOM-объекта с интерфейсом **IRsStream**, заданным параметром **strm**. При помощи данного конструктора, можно создать объект для работы с полями типа BLOB источников данных RSD.

Параметры **encode** и **eolType** аналогичны одноименным параметрам, указанным в вышеописанном варианте конструктора класса.

Свойства класса

Для класса ***TStreamDoc*** реализованы следующие свойства:

Name – свойство возвращает имя файла, если объект был сконструирован при помощи первого варианта конструктора. В противном случае значение свойства – ***null***.

Stream – свойство возвращает RSCOM-объект с интерфейсом IRsStream. Этот объект может быть передан RSCOM-методам, ожидающим объект с данным интерфейсом, например, источникам данных.

Str – свойство, доступное только для чтения. В этом свойстве сохраняется строка, которая была считана из файла последним вызовом метода ***readLine***.

Методы класса

Для класса ***TStreamDoc*** реализованы следующие методы:

WriteLine (str:String):Bool

Метод записывает в поток строку символов из параметра **str**. В конец строки символов автоматически помещается признак конца строки:

- **CR LF** – стандарт DOS и Windows (используется по умолчанию).
- **LF CR** – встречается в некоторых текстовых файлах, не является стандартным.
- **CR** – стандарт MAC.
- **LF** – стандарт UNIX.

Примечание.
Запись в файл признака конца строки откладывается и реально выполняется только в том случае, если в выходной поток будут добавляться ещё какие-либо данные.

Если в записываемой строке встречается последовательность символов конца строки, то в выходной поток всегда записывается признак конца строки, стиль которого задан параметром ***eolType*** конструктора.

ReadLine (result:@String):Bool

Метод считывает строку из файла и помещает результат в переменную, заданную выходным параметром **result**. Кроме того, считанная строка сохраняется в свойстве **Str** объекта. Символы признака конца строки считываются из потока, но изымаются из результирующей строки.

При достижении конца файла метод возвращает ***false***, иначе – ***true***.

Пример:

```
macro AddField (ar, name, tp, sz, dec)
  ar [ar.size] = name;
  ar [ar.size] = tp;
  ar [ar.size] = sz;
  ar [ar.size] = dec;
  ar [ar.size] = 0;
end;

ar = TArray;
AddField (ar, "field1", V_NUMERIC, 0, 10);
AddField (ar, "field2", V_INTEGER, 4, 0);
AddField (ar, "field4", V_STRING, 40, 0);
macro WriteProc
  ob = TStream ("file1.txt", "c");
   ob.write (23);
  ob.write (1, V_INTEGER, 1);
  ob.write (2, V_INTEGER, 2);
  ob.write (3, V_INTEGER, 4);
    ob.write (78.23);
  ob.write (5, V_DOUBLE, 4);
  ob.write (6, V_DOUBLE, 8);
  ob.write (7, V_DOUBLE, 10);
  ob.write (Numeric (23.78));
  ob.write (8, V_NUMERIC, null, 6);
  ob.write (date);
  ob.write (time);

  r = TRecHandler ("test", ar);
    r.rec.field1 = 12345.782345678;
  r.rec.field2 = 12345.782345678;
  r.rec.field4 = 12345.782345678;

  ob.write2 (r);
```

```
  ob.writeVal (10);
  ob.writeVal (Numeric (23.78));
  ob.writeVal ("Привет");
  ob.writeVal (date);
end;

macro ReadProc
  ob = TStream ("file1.txt", "r");
  ob.read (v, V_INTEGER);
  println (v);
  ob.read (v, V_INTEGER, 1);
  println (v);
  ob.read (v, V_INTEGER, 2);
  println (v);
  ob.read (v, V_INTEGER, 4);
  println (v);
    ob.read (v, V_DOUBLE);
  println (v);
  ob.read (v, V_DOUBLE, 4);
  println (v);
  ob.read (v, V_DOUBLE, 8);
  println (v);
  ob.read (v, V_DOUBLE, 10);
  println (v);
  ob.read (v, V_NUMERIC);
  println (v);
  ob.read (v, V_NUMERIC, null, 6);
  println (v);
  ob.read (v, V_DATE);
  println (v);
  ob.read (v, V_TIME);
  println (v);

  r = TRecHandler ("test", ar);
  ob.read2 (r);
  println (r.item("field1"):0:10);
  println (r.item("field2"):0:10);
  println (r.item("field4"):0:10);
  while (ob.readVal (@v))
    println (v)
  end;
end;

WriteProc;
ReadProc;

macro WriteProcDoc
  ob = TStreamDoc ("unistrm.txt", "c", "utf8");
```

```
  ob.writeLine ("Привет, Мир 1!");
  ob.writeLine ("");
  ob.writeLine ("Привет, Мир 2!");
  ob.writeLine ("Привет, Мир 3!");
end;

macro ReadProcDoc
  ob = TStreamDoc ("unistrm.txt", "r");
  while (ob.readLine (@v))
    println (ob.str)
  end;
end;

WriteProcDoc;
ReadProcDoc;
```

Управление файлами и каталогами

В языке RSL предусмотрены средства для манипулирования файлами и каталогами, расположенными на компьютере, в том числе и в сети двух- и трехуровневой архитектуры. К ним относятся стандартный класс ***TDirList*** (см. стр. 140), а также специальные процедуры управления файлами и каталогами:

- ***CopyFile*** – процедура создает копию файла, причем исходный файл и файл-копия могут находиться как на терминале, так и на сервере.
- ***RenameFile*** – процедура переименовывает файл, причем исходный файл может находиться как на сервере, так и на терминале, а переименованный файл должен располагаться там же, где и исходный файл.
- ***RemoveFile*** – процедура удаляет указанный файл, при этом файл может находиться как на сервере, так и на терминале.
- ***MakeDir*** – процедура создает каталог с указанным именем, при этом каталог может быть создан как на сервере, так и на терминале.
- ***RemoveDir*** – процедура удаляет каталог с заданным именем, причем каталог может быть удален как на сервере, так и на терминале.
- ***GetCurDir*** – процедура возвращает название текущего каталога на сервере.

Более подробно перечисленные процедуры описаны на стр. 215.

Для использования средств управления файлами и каталогами в сети трехуровневой архитектуры на терминале должен быть установлен модуль *rsextt.d32*.

> ***Внимание!***
> *Чтобы эти процедуры были доступны, в макропрограмму пользователя следует явно импортировать модуль* ***rsexts****, т.е. в макропрограмму необходимо включить процедуру импорта:*
>
> *Import rsexts*

Если при использовании указанных средств управления в качестве имен файлов не используются абсолютные пути, то действуют следующие соглашения:

- поиск файла на сервере осуществляется в каталогах текстовых файлов;
- новый файл на сервере создается в первом каталоге из списка каталогов текстовых файлов;
- поиск файла на терминале осуществляется в каталогах, которые указаны в файле *rsextt.ini* в качестве параметра DNDIR;
- новые файлы на терминале создаются в каталоге, указанном в файле *rsextt.ini* в качестве параметра UPDIR.

Чтобы указать, что файл находится на терминале, перед именем файла необходимо указать символ "$". Если прикладная программа запущена в сети двухуровневой архитектуры, то символ "$" игнорируется.

Использование класса TDirList

Класс ***TDirList*** позволяет получать список файлов и каталогов по заданной маске на сервере или на терминале, а так же позволяет удалять файлы по маске на сервере или терминале.

TDirList ([mask:String] [, attr:String])

Конструктор создает новый экземпляр объекта класса. Конструктор может быть вызван с параметрами, в этом случае объект будет наполнен списком файлов, которые удовлетворяют маске поиска:

- ***mask*** – строка с маской файлов для поиска;
- ***attr*** – строка с атрибутами для отбора файлов в список. Строка атрибутов может содержать символы F и D. Если задан только символ F, то производится поиск только файлов, удовлетворяющих маске. Если задан только символ D, то производится поиск только каталогов, удовлетворяющих маске. Если заданы оба символа, то производится поиск файлов, удовлетворяющих маске, и всех каталогов. Последний режим используется по умолчанию.

Методы класса TDirList

Copy (srcMask:String, attr:String, dstDir:String, move:Bool, indic:Bool, header:String):Bool

Метод выполняет копирование файлов с заданной маской, в указанный каталог.

Параметры:

srcMask – маска, в соответствии с которой производится отбор файлов для копирования. В качестве параметра могут быть указаны как локальные, так и удаленные файлы. Для того чтобы отобрать удаленные файлы, необходимо в качестве параметра передать символ "$" в имени файла.

attr – атрибуты для отбора файлов. В настоящей версии системы параметр не используется, процедура копирования выполняется только для файлов, каталоги не копируются.

dstDir – каталог, в который требуется выполнить копирование файлов. Наименование каталога указывается одним из следующих способов:

- указывается непосредственно имя каталога с терминирующим символом "\", например, ***d:\\mydir***;
- задается путь к произвольному файлу в заданном каталоге, например, ***d:\\mydir*.****.

В качестве каталога может быть указан как локальный, так и удаленный каталог. Для того чтобы задать удаленный каталог, необходимо в качестве параметра **srcMask** передать символ "$" в имени файла.

move – признак удаления исходных файлов после успешного копирования. Параметр принимает одно из значений:

- TRUE – удалить файлы после копирования.
- FALSE – не удалять файлы.

indic – признак отображения диалогового окна с индикатором процесса копирования. Параметр принимает одно из значений:

- TRUE – отображать диалоговое окно в процессе копирования файлов.
- FALSE – диалоговое окно не отображать.

Примечание.
Значение указанного параметра игнорируется, если исходный файл и каталог, в который выполняется копирование, находятся на терминале. В этом случае диалоговое окно не отображается.

header – заголовок окна с индикатором процесса копирования.

Возвращаемое значение:

Процедура наполняет объект класса **TdirList** информацией об отобранных по маске файлах и возвращает одно из значений:

- TRUE – если все отобранные по маске файлы были скопированы.
- FALSE – если хотя бы один файл не скопировался.

Количество файлов, отобранных по маске, можно получить, используя свойство ***Count***. Для определения перечня файлов, которые не скопировались, следует воспользоваться свойством ***IsCopy***. Если для процедуры был установлен признак удаления исходных файлов (**move**), с помощью процедуры ***IsDel*** определяется, все ли файлы успешно удалены.

Пример:

```
//Пример процедуры копирования содержимого каталога с
подкаталогами:
//
// src – исходный каталог
// dst – каталог назначения
// Каталоги задаются в виде x:\\mydir\\
//
macro DeepCopy (src, dst)
  var count = 0;
  macro DoCopy
    var i        = 0;
    var errCount  = 0;
    var cp       = TDirList;
    MakeDir (dst);
    if (not cp.Copy (src + "*.*", "f", dst + "*.*"))
      while (i < cp.Count)
        if (not cp.IsCopy (i))
```

```
            errCount = errCount + 1;
            println ("Error copy file: ", src + cp.name (i));
          end;
          i = i + 1
        end;
      end;
      count = cp.Count - errCount;
    end;
    DoCopy;
    var ob = TDirList (src + "*.*", "d");
    var i = 0;
    while (i < ob.count)
      if (substr (ob.name (i), 1, 1) != ".")
        count = count + DeepCopy (src + ob.name (i) + "\\", dst + ob.name
(i) + "\\");
      end;
      i = i + 1;
    end;
    return count;
  end;
  // Пример вызова
  println (DeepCopy ("test\\", "dest\\"));
```

List (mask:String [, attr:String])

Метод наполняет объект списком файлов и/или каталогов, удовлетворяющих заданной маске ***mask***. Параметр ***attr*** принимает такие же значения, как и одноименный параметр конструктора.

Remove (mask:String [, attr:String])

Метод наполняет объект списком файлов и/или каталогов, удовлетворяющих заданной маске ***mask***, и удаляет отобранные файлы. Параметр ***attr*** принимает такие же значения, как и одноименный параметр конструктора.

Sort ([sortBy:Integer] [, dirFirst:Bool])

Метод выполняет сортировку списка файлов. Параметр ***sortBy*** задает порядок сортировки и может принимать следующие значения:

- ***0*** – сортировка по имени фала;
- ***1*** – сортировка по размеру;
- ***2*** – сортировка по дате модификации.

Если параметр ***sortBy*** не задан, то выполняется сортировка по имени.

Если задан параметр ***dirFirst*** и он равен TRUE, то каталоги после сортировки будут находиться в списке перед всеми файлами.

Свойства класса TDirList

Свойства класса ***TDirList*** служат для получения информации из списка файлов или каталогов:

- **Count** – возвращает количество элементов в списке.
- **IsCopy (id: integer)** – возвращает TRUE, если файл с индексом ***id*** успешно скопирован. Параметр ***Id*** принимает значения от ***0*** до значения, возвращаемого свойством ***Count***.
- **Name (ind:integer)** – возвращает имя файла или каталога с индексом ***ind***.
- **Size (ind:integer)** – возвращает размер файла с индексом ***ind***.
- **FTime (ind:integer)** – возвращает время последней модификации файла с индексом ***ind***.
- **FDate (ind:integer)** – возвращает дату последней модификации файла с индексом ***ind***.
- **IsDir (ind:integer)** – возвращает TRUE, если элемент с индексом ***ind*** является каталогом.
- **IsDel (ind:integer)** – возвращает TRUE, если файл с индексом ***ind*** был успешно удален методами ***Remove*** или ***Copy***.

Пример 1:

```
Import rsexts;
…
ob = TDirList ("$*.*","f");
ob.Sort (0);
i = 0;
while (i < ob.Count)
  if (ob.isDir (i))
    dir = "<DIR>"
  else
    dir = ""
  end;
  if (ob.isDel (i))
    del = "Del"
  else
    del = ""
  end;
[ ##############   #########  ########## ###########  #####
###### ]
   (ob.name (i),  ob.size (i), ob.fdate (i), ob.ftime (i) , dir , del);
  i = i + 1
end
```

Пример 2:

```
import rsexts;
ob = TDirList; // ("mac\\*.mac");
ob.Copy ("mac\\*.mac", "", "dest\\", false, false, "Test copy")щ
//ob.Copy ("mac\\*.mac", "", "$dest\\*.*", false, true, "Test copy");
//ob.Copy ("$dest\\*.mac", "", "$dest2\\*.*", true, true, "Test copy");
if (ob.count)
```

```
  for (i, 0, ob.count - 1)
    if (ob.isDir (i))
      dir = "<DIR>"
    else
      dir = ""
    end;
        if (ob.isDel (i))
      del = "Del"
    else
      del = ""
    end;
        if (ob.isCopy (i))
      cp = "Copied"
    else
      cp = ""
    end;
    [ ##################   ############   ############ ###########
###### ##### #######]
      (ob.name (i),  ob.size (i), ob.fdate (i), ob.ftime (i) , dir , del, cp);
  end;
else
  println ("No files")
end;
```

Использование "домашних" каталогов пользователей

В прикладных программах реализовано понятие "домашних" каталогов пользователей. В этих каталогах, в дополнение к существующим каталогам, приложение осуществляет поиск персональных конфигурационных файлов. В "домашнем" каталоге также могут размещаться и персональные файлы данных.

Поиск файлов конфигурации будет происходить в следующем порядке:

1. "домашний" каталог пользователя;
2. текущий каталог прикладной программы;
3. список каталогов, заданных в переменной окружения ***RSCNFG***;
4. каталог запуска прикладной программы.

Переменная ***RSCNFG*** в трехзвенной архитектуре может быть установлена администратором сервера приложений. В двухзвенной архитектуре эта переменная может быть установлена на компьютере пользователя.

В конфигурационных файлах можно ссылаться на имя "домашнего" каталога с помощью конструкции ***%HOME%.*** Имя "домашнего" каталога при этом автоматически подставляется в значениях параметров в файлах конфигурации.

При запуске прикладной программы в двухзвенной архитектуре в качестве "домашнего" каталога используется значение переменной окружения ***RSHOME.*** Если такая переменная не задана, то в качестве "домашнего" используется текущий каталог ".".

Поддержка "домашних" каталогов в сети трехзвенной архитектуры предусмотрена для сервера приложений версии 5.03.105 и выше. "Домашний" каталог при этом задается администратором сервера приложений при настройке сервера приложений следующим образом:

- администратор создает каталоги с именами пользователей, которые присоединяются к серверу приложений;
- администратор при настройке конфигурации сервера приложений присваивает значение параметру ***ROOTDIR*** (см. раздел "Утилита настройки конфигурации сервера приложений" Руководства "Сервер приложений. Администрирование сервера приложений"). Если этот параметр не задан, то вместо него используется значение параметра ***STARTDIR***.

Имя "домашнего" каталога для пользователя образуется из значения параметра ***ROOTDIR*** плюс имя каталога, совпадающее с сетевым именем пользователя, присоединённого к серверу приложений.

В сети трехзвенной архитектуры RSL-программам доступны устанавливаемые сервером приложений переменные окружения:

- **RSAPPROOT** – каталог, заданный в настройках сервера приложений как ROOTDIR.
- **RSAPPFS32** – полный путь к используемой процессом библиотеке fs32cm.dll.
- **RSAPPNAME** – имя сервера приложений, запустившего прикладной процесс.

Для доступа к персональным файлам пользователя предоставляется процедура ***GetSysDir*** с параметром, равным 4.

Также предусмотрено использование процедуры ***GetUserName***. Эта процедура в трехзвенной архитектуре возвращает сетевое имя пользователя, присоединенного к серверу приложений, в двухзвенной архитектуре возвращает имя текущего пользователя.

Использование RSCOM-объектов из программ на языке RSL

RSCOM – распределенная объектная модель, разработанная в компании R-Style Softlab. Эта модель является удобным средством построения повторно используемых модулей. Такими модулями являются компоненты – RSCOM-серверы.

RSCOM-компоненты доступны из разных языков программирования и из различных программ. RSCOM-компоненты могут использоваться напрямую из языка RSL и C++ как в программах компании R-Style Softlab, так и в разработках сторонних разработчиков.

Технология RSCOM обеспечивает удаленный доступ к прикладным компонентам. С помощью RSCOM-компонентов можно из любого приложения, написанного на языке, поддерживающем RSCOM, получать доступ к прикладным объектам на любом сервере приложений R-Style SoftlLab в сети и работать с ними.

Посредством модуля сопряжения с ActiveX модули RSCOM становятся доступными везде, где доступны ActiveX-компоненты.

RSCOM позволяет свободно обмениваться объектами между программой-терминалом, предоставляющей пользовательский интерфейс, и прикладным процессом, используя при этом тот же коммуникационный канал, что используется терминалом и прикладным процессом.

Основным функциональным элементом модели является RSCOM-сервер – специального вида DLL, предоставляющая программам-клиентам RSCOM-объекты. Клиент работает с объектами посредством интерфейсов. Объект может поддерживать множество интерфейсов. RSCOM-серверы загружаются в адресное пространство процесса-хоста. Хост-процессом может являться программа-терминал, любая прикладная программа, разработанная при помощи стандартного инструментария либо специальная программа-контейнер RSCOM-серверов *rcomcnt.exe*.

Клиентом RSCOM-серверов может являться код в прикладном процессе либо код в RSCOM-сервере. Для обеспечения взаимодействия клиентского кода с удаленными RSCOM-объектами необходим маршалинг интерфейсов. Инфраструктура RSCOM обеспечивает маршалинг базовых интерфейсов.

Основное назначение RSCOM-серверов – обеспечение удаленного доступа к RSCOM-объектам. Но использование RSCOM-серверов может быть полезно и для использования локальных RSCOM-объектов. В этом случае RSCOM выполняет рутинную работу по поиску и загрузке RSCOM-серверов. Если RSCOM-сервер планируется использовать только локально, то не нужно обеспечивать маршалинг его интерфейсов.

Для того чтобы RSCOM-объект мог использоваться из Object RSL, этот объект обязан поддерживать диспетчерский интерфейс. Диспетчерский интерфейс является основным интерфейсом в RSCOM.

Поддержка RSCOM в RSL реализована во встроенном модуле RCW.

Для создания RSCOM-объектов служит процедура ***CreateObject*** из модуля RCW. Для создания специфического коммуникационного канала этот модуль содержит специальный класс ***TRslChanel***. Объект, созданный при помощи ***CreateObject***, может использоваться в RSL-коде так же, как и объект любого другого RSL-класса.

Объекты "обёртки" для RSCOM-объектов в RSL поддерживают стандартный метод ***_callHost***:

_callHost (methodName:string, par1, par2, ...) : variant

Метод вызывает процедуру или конструктор класса с именем ***methodName*** либо возвращает значение глобальной переменной с таким именем из экземпляра RSL, которому принадлежит обёртка. Следующие за именем процедуры (класса) параметры ***par1***, ***par2*** и т.п. передаются без изменений в вызываемую процедуру или конструктор класса.

Возвращаемым значением является значение, которое возвращает вызываемая процедура или конструктор класса.

Пример:

```
ob = CreateObject ("rcwhost","TRcwHost","MyInst",false);
ob.GUIMode = false;
rslCls1 = ob.CreateRSLObject ("RslSrv3","TRslClass");
// Вызываем глобальный метод
println (rslCls1._callHost ("TstMac", "Параметр"));
// Устанавливаем и читаем значение глобальной переменной
rslCls1._callHost ("TstVar") = "Проба";
println (rslCls1._callHost ("TstVar"));
// Устанавливаем и читаем значение глобальной переменной через
TRcwHost.
ob.call ("TstVar") = 100;
println (ob.call ("TstVar"));
```

Описание стандартного модуля RCW

В этом разделе дается описание стандартного модуля RCW. Данный модуль реализован вместе с ядром интерпретатора Object RSL в библиотеке *RsScript.dll*. Перед использованием процедур и классов модуля его необходимо импортировать при помощи директивы ***Import***.

Процедуры

CreateMarkObj (p1:Variant, p2:Variant, …):Object

Процедура используется для создания и заполнения объекта **MarkObject** произвольным количеством переменных RSL ***p1***, ***p2***…, переданных в качестве параметров. Созданный объект служит заместителем при реализации иерархических поставщиков данных или для чтения полей типа BLOB по запросу.

ReadMarkObj (mark:Object, p1:@Variant, p2:@Variant, …):Bool

Процедура позволяет определить значение переменных объекта, созданного процедурой ***CreateMarkObj***.

Параметры:

mark – наименование объекта, для которого необходимо определить значения переменных.

p1, p2, … – выходные параметры, в которых сохраняются найденные значения переменных.

Пример:

```
import rcw;

mark = CreateMarkObj (1, Numeric(1234.567), 3.14, "Строка", $5);

println (ReadMarkObj (mark, @p1, @p2, @p3, @p4, @p5));

println (p1, " ", p2, " ", p3, " ", p4, " ", p5);
```

CreateObject (serverName:String, className:String [, objectName:String, (isLocal:Bool | chanel:TRslChanel), par1, par2,...]):Object

Процедура загружает необходимый RSCOM-сервер и создает в нем затребованный RSCOM-объект.

Параметры:

serverName – имя RSCOM-сервера без расширения d32.

className – имя класса RSCOM-объекта.

objectName – имя конкретного экземпляра объекта.

isLocal – признак локальной загрузки RSCOM-сервера: если значение параметра равно TRUE, RSCOM-сервер загружается локально, иначе удаленно.

В качестве четвертого параметра может использоваться либо ***isLocal***, либо ***chanel***. Если этот параметр не задан, то используется параметр ***isLocal***, равный TRUE, то есть загрузка RSCOM-сервера выполняется локально.

chanel – специфический объект коммуникационного канала класса ***TRslChanel***.

par1, par2,... – переменный список параметров, передающихся в метод инициализации объекта.

Обязательным параметром является только имя сервера. Кроме того, обязательно должен быть задан хотя бы один из двух параметров: ***className*** или ***objectName***. Если задан параметр ***className***, но не задан ***objectName***, всегда будет создаваться новый экземпляр объекта заданного класса. Если задан ***objectName*** и не задан ***className***, объект с заданным именем должен уже существовать. В противном случае во время выполнения программы будет сгенерирована ошибка. Если заданы оба параметра, то если объект с заданным именем существует, он возвращается клиенту, иначе создается новый экземпляр.

Метод инициализации объекта – это диспетчерский метод с фиксированным идентификатором RSDISP_DEFAULT, вызываемым по уровню RSCOM_SYS_LEVEL.

Возвращаемое значение:

Процедура возвращает экземпляр созданного объекта. При невозможности создания объекта во время выполнения программы генерируется ошибка.

GetNamedChanel (name:String):Object

Процедура проверяет, существует ли канал с именем ***name***, и если канал существует, возвращает объект класса ***TRslChanel*** для него. В противном случае возвращается значение NULL.

ClrRmtStubs ()

Процедура принудительно удаляет все ненужные удалённые объекты. По умолчанию освобождение ненужных удаленных объектов происходит во время следующей транзакции по коммуникационному каналу, при помощи которого эти объекты были созданы.

PrintObject (object:Object, [CaseSensitive:Bool])

Процедура выводит в стандартный выходной поток информацию об именах всех свойств и методов объекта, переданного ей в качестве параметра.

Параметры:

object – объект, имена свойств и методов которого возвращает процедура.

CaseSensitive – признак возврата информации о свойствах и методах объекта в программу в исходном виде. Параметр может принимать одно из следующих значений:

- ***TRUE*** – процедура возвращает информацию без изменения регистра.
- ***FALSE*** – в верхнем регистре.

По умолчанию, если значение параметра не задано, оно считается равным ***FALSE***.

Пример:

```
import rcw;
class TTT
  var p1 = 5;
  macro Method1
  end;
end;
ob = TTT;
PrintObject (ob);
```

LockRcwHost (…)

Процедура позволяет принудительно увеличить счетчик ссылок для объекта хоста.

Внимание!
Если программа RSL выполняется не под управлением RSCOM объекта класса ***TRcwHost****, вызов процедуры не приводит к требуемому результату.*

UnlockRcwHost (…)

Процедура позволяет принудительно уменьшить счетчик ссылок для объекта хоста.

Внимание!
Если программа RSL выполняется не под управлением RSCOM объекта класса ***TRcwHost****, вызов процедуры не приводит к требуемому результату.*

Класс TRslChanel

Объекты класса ***TRslChanel*** предоставляют RSL-программе коммуникационный канал для связи с произвольным сервером приложений. Использование такого канала обеспечивает возможность загружать RSCOM-серверы в процессы на альтернативных серверах приложений.

Конструктор класса выглядит следующим образом:

```
TRslChanel ([server:String, protocol:Intger, termNumber:Integer,
pipeName:String,keyPath:String, user:String, domain:String,
password:String, port:Integer, ip:String, spx:String, NBpref:String,
lana:Integer, hostApp:String, name:String]): Object
```

Параметры:

Все параметры конструктора являются необязательными:

server – имя сервера, к которому необходимо подключиться. Для локального сервера необходимо указать ".". Это значение используется, если параметр не задан.

protocol – код используемого протокола. Могут применяться следующие значения:

- 1 – именованный канал (named pipe);
- 2 – TCP/IP;
- 4 – SPX;
- 8 – NetBIOS.

Если этот параметр не задан, по умолчанию используется значение 1 (named pipe).

termNumber – номер используемого терминала. Номер терминала определяет имя используемого файла ключей, который задает режим шифрования и паковки передаваемых по сети пакетов.

pipeName – имя именованного канала. Задавать этот параметр необходимо только в том случае, если сервер использует имя канала, отличное от принятого по умолчанию ***RsAppServ***.

keyPath – имя каталога, в котором необходимо искать файл ключей. Если параметр не задан, то ключи ищутся в текущем каталоге и каталоге запуска клиентской программы.

userName – имя пользователя, с правами которого необходимо подсоединяться к серверу приложений.

domain – домен пользователя.

password – пароль пользователя.

port – порт для протоколов TCP/IP и SPX.

ip – IP-адрес сервера для протокола TCP/IP. Если этот параметр не задан, то ip-адрес определяется по имени сервера ***serverName***.

spx – SPX-адрес сервера для протокола SPX.

NBpref – трехсимвольный NetBIOS-префикс, используемый на сервере для протокола NetBIOS.

lana – номер сетевого адаптера для протокола NetBIOS.

hostApp – задает имя процесса-хоста, в адресное пространство которого загружается RSCOM-сервер. Если этот параметр не задан, используется специальный процесс *rcomcnt.exe*.

name – имя коммуникационного канала.

Созданный объект является не подключенным к серверу приложений. Для выполнения подключения можно явно вызвать метод ***Connect***. Если неподключенный объект передать в процедуру ***CreateObject***, то подключение выполнится автоматически. После создания объекта до его подключения к серверу приложений можно задавать значения любым свойствам объекта.

Методы класса TRslChanel

Connect:Bool

Метод выполняет соединение с сервером приложений с установленными параметрами соединения.

LoadConfig (iniFile:String):Bool

Метод загружает параметры соединения из ini-файла, заданного параметром ***iniFile***.

Свойства класса TRslChanel

Данный класс имеет свойство ***Name***. Если для него задано значение, то новый канал создаётся только в том случае, если канал с таким именем ещё не существует. Иначе объект просто связывается с уже существующим каналом.

Помимо ***Name***, класс обладает следующим набором свойств, каждое из которых соответствует назначению одноименного параметра конструктора:

- **Protocol**
- **Server**
- **KeyPath**
- **PipeName**
- **Ip**
- **Spx**
- **NBPref**
- **Lana**

- **Port**
- **TermNumber**
- **User**
- **Domain**
- **Password**
- **HostApp**
- **Name**

Класс TRcwSite

Стандартный класс ***TRcwSite*** позволяет обрабатывать интерактивные сообщения от RSCOM-объекта ***RcwHost***. Стандартная реализация вызывает соответствующий метод в текущем экземпляре RSL. В производном классе можно переопределить реализацию базовых методов.

Пример:

```
import rcw;
class (TRcwSite) MySite
  macro OnTrace (mes)
    MsgBox ("OnTrace in MySite: ",mes);
    _extObj.OnTrace (mes);
  end;
end;
ob1 = CreateObject
("rcwhost","TRcwHost","MyInst",false,MySite,"RslSrv2");
```

Обработка ошибок RSCOM в языке RSL

Для обработки ошибок RSCOM в RSL необходимо использовать стандартную конструкцию ***onError***. Стандартный объект ошибки, передающийся в ***onError***, содержит дополнительное поле ***err*** с информацией о прикладной ошибке. Данное поле заполняется объектом класса ***TRsComError*** при возникновении ошибки RSCOM. Объект ***TRsComError*** содержит свойство ***count*** с количеством доступных RSCOM ошибок и индексированные свойства:

- **Message (n)** - текст n-ой ошибки RSCOM.
- **Code (n)** - код n-ой ошибки.
- **Level (n)** - прикладной уровень n-ой ошибки.

Пример:

```
onError (er)
  var count, i;
  if (IsEqClass ("TRsComErr", er.err))
    count = er.err.Count;
    i = 0;
    while (i < count)
      println ("*** ", er.err.Message (i)," (Code = ",er.err.Code (i),
            ", Level = ",er.err.Level (i),")");
      i = i + 1
    end
  end;
```

Обработка сообщений от RSCOM-объектов

RSCOM-объекты могут генерировать события. Их обработка в RSL выполняется точно так же, как и обработка событий от ActiveX- или RSL-объектов, но с некоторыми отличиями. Для обработки событий необходимо присвоить объект-источник событий параметризированному свойству ***EvSource*** объекта класса ***TRslEvHandler***:

EvSource (pref:String [,num:Integer])

Первый параметр задает строковый префикс, идентифицирующий источник событий. Второй, необязательный параметр для RSCOM-объектов задаёт начальный размер таблицы соответствий событий и методов обработки. При необходимости эта таблица динамически увеличивается.

При получении события от RSCOM-объекта автоматически осуществляется привязка к обработчику. Имя обработчика образуется по правилу:

<строковый префикс>+'_'+<имя события>

Если обработчик не найден, то для событий RSCOM-объектов метод ***OnUnbind*** не вызывается.

Для явной установки имени обработчика применяется метод ***SetHandler***:

SetHandler (pref:String, handler:String, (id:Integer | name:String):Bool

Параметр ***Pref*** задаёт префикс для идентификации источника событий. Параметр ***Handler*** задаёт имя метода обработки события. Последний параметр может задавать или целочисленный номер события, которое будет обрабатываться, или его имя.

Примечание.
Задание имени события поддерживается только для RSCOM-объектов.

Для удаления обработчика применяется метод ***RemHandler***:

RemHandler (pref:String, (id:Integer | name:String):Bool

После установки связи между объектом-источником и обработчиком событий от источника передаются все без исключения события, даже те, для которых нет обработчика. В трехзвенной архитектуре это может привести к неоправданному увеличению трафика. Для решения проблемы можно применить метод ***PinEvents***:

PinEvents (pref:String)

Метод разрешает передачу от источника событий только тех событий, для которых установлены обработчики методом ***SetHandler***.

Примеры использования

В примерах используется RSCOM-сервер *proba3.d32*. В нем создается объект класса ***TDemoClass***. Объект поддерживает метод ***GetCompName***, возвращающий имя компьютера, на котором исполняется RSCOM-сервер.

Пример 1:

```
/* В этом примере RSCOM-сервер загружается локально.  */
import rcw;
ob = CreateObject ("proba3","TDemoClass");
println (ob.GetCompName);
```

Пример 2:

```
/* В этом примере RSCOM-сервер загружается удаленно.  */
import rcw;
ob = CreateObject ("proba3","TDemoClass", null,false);
println (ob.GetCompName);
```

Пример 3:

```
// В этом примере RSCOM-сервер загружается удаленно с
//использованием установленного коммуникационного канала к
серверу //TECHMASTER.
import rcw;
cn = TRslChanel ("TECHMASTER”);
ob = CreateObject ("proba3","TDemoClass",null,cn);
println (ob.GetCompName);
```

Пример 4:

```
/* В этом примере RSCOM-сервер загружается удаленно с
использованием установленного коммуникационного канала к
серверу TECHMASTER. Параметры соединения задаются при
помощи свойств объекта класса TRslChanel.*/
import rcw;
cn = TRslChanel;
cn.server = “TECHMASTER”;
cn.protocol = 2;
if (cn.connect)
  ob = CreateObject ("proba3","TDemoClass",null,cn);
  println (ob.GetCompName);
end;
```

Пример 4:

```
/* В этом примере демонстрируется обработка ошибок RSCOM
средствами RSL */
import rcw;
var ob = CreateObject ("rcwhost","TRcwHost","MyInst",false);
ob.AddModule ("RslSrv_");
ob.Execute;
onError (er)
var count, i;
println (er.message);
if (IsEqClass ("TRsComErr", er.err))
  count = er.err.Count;
  i = 0;
  while (i < count)
    println ("*** ", er.err.Message (i)," (Code = ",
```

```
            er.err.Code (i),", Level = ",er.err.Level (i),")");
        i = i + 1
      end
    end
```

Пример 5:

```
/* В этом примере демонстрируется обработка событий RSCOM-
объектов */
import rcw;
macro Test_OnGetCompName (p1,p2,p3)
  println ("OnGetCompName ",p1,"  ",p2,"  ",p3)
end;
ob = CreateObject ("proba3","TDemoClass",null,true);
RslEvHandler.EvSource ("Test") = ob;
println ("Local computer name is: ", ob.GetCompName);
```

Пример 6:

```
/* В этом примере демонстрируется обработка событий RSCOM-
объектов при помощи задания обработчика методом SetHandler*/
import rcw;
macro HandleOnGetCompName (p1,p2,p3)
  println ("OnGetCompName ",p1,"  ",p2,"  ",p3)
end;
ob = CreateObject ("proba3","TDemoClass",null,true);
RslEvHandler.EvSource ("Test") = ob;
RslEvHandler.SetHandler
("Test","HandleOnGetCompName","OnGetCompName");
RslEvHandler.PinEvents ("Test");
println ("Local computer name is: ", ob.GetCompName);
```

Стандартные RSCOM-серверы

В этом разделе описываются стандартные RSCOM-серверы *rsax.d32*, *rcwhost.d32* и *rsclr.d32*. Данные серверы могут использоваться как клиентами из кода C++, так и из кода Object RSL. Загружаться они могут любым из трех возможных способов.

Данные модули входят в состав RSCOM SDK.

RSCOM-сервер rsax.d32

RSCOM-сервер *rsax.d32* предоставляет в распоряжение клиентам доступ к ActiveX-объектам. В отличие от стандартной процедуры RSL ***ActiveX***, использование RSCOM-сервера ***rsax*** позволяет создавать ActiveX-объекты на удаленных компьютерах, в том числе клиентам на платформах, не поддерживающих технологии Microsoft ActiveX и COM.

Использование сервера ***rsax*** заключается в следующем. При помощи процедуры Object RSL ***CreateObject*** необходимо создать RSCOM-объект класса ***TRsAxServer***, выполняющий роль "фабрики" ActiveX-объектов. Объект этого класса содержит метод ***CreateComObject***, позволяющий создать необходимый ActiveX-объект.

Экземпляр класса ***TRsAxServer*** остается в памяти до тех пор, пока не будут удалены все созданные им ActiveX-объекты. RSCOM-сервер *rsax.d32* версии 3.03.018.0 и старше всегда создаёт только один экземпляр класса ***TRsAxServer*** с именем ***RsAxDefaultInst***.

Так как единственный объект класса ***TRsAxServer*** может обслуживать все запросы от клиентов, то при создании можно дать ему уникальное имя. При необходимости повторного создания объекта класса ***TRsAxServer*** будет использоваться один и тот же экземпляр, что позволит экономить оперативную память.

Конструктор класса не имеет параметров.

Методы класса TRsAxServer

CreateComObject (progID:String [, useActive:Bool] [, evId]):Object

Метод создает ActiveX-объект с программным идентификатором ***progID***. Если второй необязательный параметр равен TRUE, метод пытается вернуть ссылку на существующий активный ActiveX-объект. Если такого объекта нет, создается новый экземпляр.

Параметр ***evId*** задает уникальный идентификатор интерфейса в виде строки.

CreateComObject (progID:String , moniker:Integer):Object

Применение указанного варианта метода позволяет создавать ActiveX-объекты с использованием моникеров. В параметре ***progID*** указывается строковое представление моникера. Значение параметра ***moniker*** всегда должно быть равно 1.

Пример:

```
/* Пример создает удаленный объект Excel.Application и делает
окно Excel видимым */

import rcw;

var ob = CreateObject ("rsax","TRsAxServer","RsAxServer",false);

var ex = ob.CreateComObject ("Excel.Application");

ex.visible = true;

MsgBox (“Excel стартован”);
```

Cast (obj:Object, intf:String, typeLib:String, verHi:Integer, verLow:Integer):Object

Метод позволяет запросить интерфейс требуемого объекта.

Параметры:

obj – объект, интерфейс которого требуется запросить.

intf – уникальный идентификатор или имя запрашиваемого интерфейса. Если заданный интерфейс поддерживается объектом, метод создаёт диспетчерский объект, реализующий заданный интерфейс. Интерфейс может быть дуальным или произвольным COM интерфейсом. Важно, чтобы указанный интерфейс был совместимым с COM автоматизацией и был описан в библиотеке типов ***typeLib***.

typeLib – уникальный идентификатор библиотеки.

verHi – старший номер версии библиотеки.

verLow – младший номер версии библиотеки.

Cast (obj:Object, intf:String, progId:String):Object

Указанный вариант метода позволяет автоматически определить уникальный идентификатор библиотеки типов и её версию по заданному программному идентификатору ***progId*** любого COM класса из этой библиотеки типов. Параметры ***obj***, ***intf*** аналогичны описанным выше.

Пример:

```
/* Пример создает удаленный объект Excel.Application и делает
окно Excel видимым */
import rcw;
var objRSAX = CreateObject ("rsax", "TRsAxServer", "RsAxServer",
true);
var rpt = objRSAX.CreateComObject ("FastReport.TfrxReport");
rpt.PrepareReport(true);
// Три варианта получения одного и того же интерфейса
var expIntf = objRSAX.Cast (rpt, "{4859EE7C-4132-4B7B-87EE-
0BA8CAE4A1A3}",
      "{D3C6FB9B-9EDF-48F3-9A02-6D8320EAA9F5}", 4, 0);
var expIntf2 = objRSAX.Cast (rpt, "{4859EE7C-4132-4B7B-87EE-
0BA8CAE4A1A3}",
      "FastReport.TfrxReport");
 var expIntf3 = objRSAX.Cast (rpt, "IfrxBuiltinExports",
      "FastReport.TfrxReport");
expIntf.ExportToPDF ("D:\\user\\test.pdf", true, true, true);
expIntf2.ExportToPDF ("D:\\user\\test2.pdf", true, true, true);
```

RSCOM-сервер rcwhost.d32

RSCOM-сервер *rcwhost.d32* предоставляет функциональность интерпретатора Object RSL. Сервер позволяет:

- создать локально или удаленно экземпляр интерпретатора;
- загружать в него RSL- и DLM-модули;
- исполнять загруженные модули;
- вызывать процедуры;
- создавать экземпляры RSL-классов.

Rcwhost.d32 позволяет передавать в качестве параметров и возвращать как результат кроме скалярных типов данных так же объекты RSL-классов. Такая функциональность реализуется классом ***TRcwHost***.

RSCOM-сервер *rcwhost.d32* считывает свои настройки из собственного настроечного файла *rcwhost.ini*. В этом файле можно указать следующие значения параметров:

- **DATADIC** – имя словаря базы данных.

- **DATAPATH** – список каталогов с таблицами баз данных.
- **INCPATH** – список каталогов для поиска mac- и dlm-файлов.
- **WORKDIR** – каталог для записи выходных файлов.
- **DBFPATH** – список каталогов для поиска db
- **LOGDIR** – каталог, в котором создаются файлы протоколирования ошибок.
- **SQLMODE** – признак используемого режима ("Yes" – SQL, "No" – Btrieve). По умолчанию принимается Btrieve.

В списках каталоги могут разделяться символом ';'. В именах каталогов можно применять макроподстановку %EXEDIR%, которая заменяется именем каталога запуска прикладной программы.

Файл *rcwhost.ini* ищется по правилам поиска любых других ini-файлов.

Пример файла rcehost.ini:

```
DATADIC   = bank.def
DATAPATH  = d:\dbfile
INCPATH   = %EXEDIR%\ext;%EXEDIR%\sample;%EXEDIR%\mac
WORKDIR   = %EXEDIR%\out
DBFPATH   =
TXTPATH   =
```

В языке RSL предусмотрена специальная процедура ***UnderRCWHost***, с помощью которой можно определить, исполняется ли RSL-файл модулем *rcwhost.d32* или нет. Если процедура возвращает значение True, то файл исполняется данным модулем, в противном случае возвращается значение False.

Использование класса TRcwHostслучае возвращается значение False.

Использование класса TRcwHost

Возможны два варианта вызова конструктора класса ***TRcwHost***. Параметры обоих являются необязательными.

1 вариант: **TRcwHost ([site:TRcwSite, module:String])**

Конструктор устанавливает сайт ***site*** (см. метод ***AttachSite***) и добавляет модуль с именем ***module***.

2 вариант: **TRcwHost ([module:String])**

Конструктор добавляет модуль с именем ***module***.

Свойства класса TRcwHost

Класс ***TRcwHost*** имеет следующие свойства:

- **Version** – номер версии модуля;
- **SQLMode:bool** – признак, определяющий, является ли данный модуль SQL-модулем. Если признак равен TRUE, то является.

Методы класса TRcwHost

AttachSite (site:TRcwSite)

Метод устанавливает для объекта ***TRcwHost*** объект типа ***TRcwSite*** или производный от него.

Объект ***TRcwSite*** используется для вызова команд, реализация которых зависит от приложения-хоста.

DetachSite

Удаляет ранее установленный сайт.

AddModule (moduleName:String):Bool

Метод загружает в объект интерпретатора RSL-модуль с именем ***moduleName***. В качестве имени модуля может быть указано имя встроенного модуля, например, ***system***, имя mac-файла или имя DLM-модуля. Указанный модуль загружается только в том случае, если он не был загружен ранее.

Если модуль был успешно добавлен, процедура возвращает TRUE, иначе FALSE.

Execute:Bool

Метод исполняет загруженные в объект интерпретатора модули. После исполнения загруженных модулей можно вызывать процедуры и создавать объекты RSL-классов при помощи методов ***Call***, ***RunProc***, ***GetVar***, ***SetVar***.

В случае успешного исполнения процедура возвращает TRUE, иначе FALSE.

Stop

Процедура деинициализирует экземпляр интерпретатора. Можно повторно вызывать метод ***Execute*** для инициализации RSL.

TestExist (moduleName:String):Bool

Процедура проверяет, загружен ли уже в память модуль с именем ***moduleName***. Если загружен, возвращает TRUE, иначе – FALSE.

Call (methodName:String, par1, par2, ...) : Variant

Метод вызывает процедуру или конструктор класса с именем ***methodName*** либо возвращает значение глобальной переменной с таким именем. Следующие за именем процедуры (класса) параметры ***par1***, ***par2*** и т.п. передаются без изменений в вызываемую процедуру или конструктор класса.

Возвращаемым значением является значение, которое возвращает вызываемая процедура или конструктор класса.

Пример использования:

В приведенном примере создается удаленный RSCOM-объект класса TRcwHost, представляющий экземпляр интерпретатора RSL. Объекту присваивается имя MyInst. В созданный объект загружается RSL модуль RslSrv.mac, содержащий RSL класс TRslClass. Далее создаем удаленный экземпляр этого класса и работаем с его методами и свойствами. Содержимое модуля RslSrv.mac:

```
/*Этот класс расположен в модуле RslSrv.mac. Мы будем
работать с удаленным экземпляром этого класса. */
class TRslClass
  var prop1 = "Property1";
  macro Method1 (p1,p2)
     prop1 = p1;
     return p1 + p2
  end;
end;
import rcw;
/* Код из модуля TestRcw.mac. Демонстрирует работу с
удаленным RSL классом */
ob = CreateObject ("rcwhost","TRcwHost","MyInst",false);
ob.AddModule ("RslSrv");
ob.Execute;
rslCls = ob.Call ("TRslClass");
println (rslCls.prop1);
println (rslCls.Method1 (10, 20));
println (rslCls.prop1);
// Вывод программы:
Property1
30
10
```

Run

Метод аналогичен методу ***Call***, но не позволяет получать значения глобальных переменных. Вызывает только методы и конструкторы классов.

GetVar (varName:String):Variant

Метод возвращает значение глобальной переменной RSL с именем ***varName***.

SetVar (varName:String, value:Variant)

Метод присваивает новое значение ***value*** глобальной переменной RSL с именем ***varName***.

CreateRSLObject (moduleName:String, className:String [, par1, par2, …]):Object

Метод создаёт объект RSL-класса с именем ***className*** из модуля ***moduleName***. Модуль ***moduleName*** загружается только в том случае, если он ещё не был загружен. Параметры ***par1***, ***par2*** и т.д. задают параметры, передаваемые конструктору.

Метод не требует предварительного вызова метода ***Execute***. При необходимости он автоматически вызывается. Данный метод похож на аналогичные методы создания объектов из модулей *rsax.d32*, *rsclr.d32*.

Пример:

```
import rcw;
ob = CreateObject ("rcwhost","TRcwHost","MyInst",false);
```

rslCls = ob.CreateRSLObject ("RslSrv","TRslClaSetDefFile (name:String)

Процедура задаёт имя файла словаря базы данных.

SetDbtDir (name:String)

Процедура задаёт список каталогов для поиска dbt-файлов.

SetDbfDir (name:String)

Процедура задаёт список каталогов для поиска dbf-файлов.

SetTxtDir (name:String)

Процедура задаёт список каталогов для поиска текстовых файлов.

SetMacDir (name:String)

Процедура задаёт список каталогов для поиска mac- и dlm-файлов.

SetOutDir (name:String)

Процедура задаёт имя каталога для выходных файлов.

RSCOM-сервер rsclr.d32

RSCOM-сервер *rsclr.d32* предоставляет клиентам доступ к объектам классов из сборок платформы .NET. Для его использования на компьютере должен быть установлен Microsoft .NET Framework.

При помощи процедуры Object RSL ***CreateObject*** необходимо создать RSCOM-объект класса ***TClrHost***. Объект этого класса содержит метод ***CreateCLRObject***, позволяющий создать необходимый объект из сборки .NET. RSCOM-сервер *rsclr.d32* создаёт только один экземпляр класса ***TClrHost*** с именем ***RsClrDefaultInst***.

Конструктор класса не имеет параметров.

Методы класса TClrHost

CreateCLRObject (assembly:String, className:String):Object

Метод создаёт объект класса с именем ***className*** из сборки с именем ***assembly***. Имя класса должно быть задано с указанием пространства имён. Приватные сборки ищутся в каталоге запуска приложения и в подкаталоге CLR.

Пример:

```
В примере используется .NET сборка ClassLib2.dll, написанная на
C#. Исходный текст:
using System;
using System.Reflection;
[assembly: AssemblyVersion("1.0.*")]
namespace ClassLib2
{
  public class Class1
```

```
  {
    public Class1()
     {
     }
    public int Method1(int p1, ref short p2)
    {
      short v = p2;
      p2 = 100;
      return p1 + v;
    }
    public string Prop1
    {
      get { return fld1;  }
      set { fld1 = value; }
    }
    private string fld1;
  }
}
// На RSL можно использовать объект следующим образом:
import rcw;
ob = CreateObject ("rsclr","TClrHost",null,true);
clr = ob.CreateClrObject ("ClassLib2","ClassLib2.Class1");
println (clr);
val = 2;
println (clr.Method1 (1,@val));
println (val);
clr.Prop1 = "Data for Prop1";
println (clr.Prop1);
```

Встроенные процедуры

В этом разделе представлен список всех стандартных процедур RSL. Они охватывают довольно широкий диапазон обработки данных:

- типы и значения переменных;
- работа со строками;
- параметры процедур;
- обработка ошибок и прочее.

Как правило, в результате успешной работы эти процедуры возвращают TRUE, в против\ном случае – FALSE.

При описании процедур в квадратные скобки заключаются необязательные параметры.

Стандартные процедуры ввода данных с клавиатуры

Процедуры этой группы присваивают переменной, имя которой передано в качестве ***первого параметра***, значение, введенное пользователем с клавиатуры. Значение, которое эта переменная имела до вызова процедуры, используется как значение по умолчанию.

Второй параметр процедуры, имеющий тип String, задает текст приглашения к вводу. Если он отсутствует, то по умолчанию принимается следующий текст: "Введите значение: ".

Третий параметр задает ширину поля ввода. Если он не задан, то используется значение по умолчанию, указанное при описании каждой процедуры.

Четвертый параметр имеет тип Bool. Если он равен TRUE, то вводимая информация отображается на экране звездочками.

Пятый параметр имеет тип Integer и задает количество знаков после десятичной точки.

Шестой параметр процедуры задает заголовок окна.

Седьмой параметр содержит строку статуса.

В случае успешной работы эти процедуры возвращают TRUE. Если пользователь закрыл окно ввода с помощью клавиши [Esc], возвращается FALSE, при этом значение переменной не изменяется.

Пример:

```
aa = "Иван";                          /* Присвоено значение переменной, которая
                                         будет использована как первый параметр
                                         процедуры */
if (GetString (aa,"Как вас            /* На экране появится запрос: "Как вас зовут?
зовут?"));                               Иван". */
println ("Вас зовут: ",aa);           /* Печать измененного или прежнего значения
end;                                     после ввода ответа */
```

GetInt (id [, prompt, len [, hide]])

Процедура присваивает введенное пользователем значение переменной типа Integer с именем ***id***. Пользователь может ввести в поле значение из диапазона от -2147483648 до 2147483647. По умолчанию ширина поля ввода равна 11 символам.

GetDouble (id [, prompt, len [, hide [, pos]]])

Процедура присваивает введенное пользователем значение переменной типа Double с именем ***id***. По умолчанию ширина поля ввода равна 24 символам.

GetMoney (id [, prompt, len [, hide [, pos]]])

Процедура присваивает введенное пользователем значение переменной типа Money с именем ***id***. По умолчанию ширина поля ввода равна 18 символам; количество знаков после десятичной точки равно 2.

GetNumeric (id [, prompt, len [, hide [, pos]]])

Процедура присваивает введенное пользователем значение переменной типа Numeric с именем ***id***. По умолчанию количество знаков после десятичной точки равно 4.

GetString (id [, prompt, len [, hide]])

Процедура присваивает введенное пользователем значение переменной типа String с именем **id**. По умолчанию ширина поля ввода равна 70 символам.

GetStringR (id [, prompt, len [, hide]])

Процедура используется для ввода числовых строк (например, номеров лицевых счетов). Она присваивает введенное пользователем значение типа String переменной с именем ***id***. Кроме этого, она выравнивает введенное значение по правому краю поля ввода. По умолчанию ширина поля ввода равна 9 символам.

GetTime(id [, prompt [, hide]])

Процедура присваивает введенное пользователем значение переменной типа Time с именем ***id***. Ширина поля ввода всегда равна 10 символам.

GetDate (id [, prompt [, hide]])

Процедура присваивает введенное пользователем значение переменной типа Date с именем ***id***. Ширина поля ввода всегда равна 10 символам.

GetTRUE (id [, prompt])

Процедура присваивает введенное пользователем значение переменной типа Bool с именем ***id***.

Если пользователь утвердительно ответил на запрос, то переменной ***id*** присваивается значение TRUE, и процедура возвращает TRUE. Если в качестве параметра процедуры ***id*** используется константа (TRUE или FALSE), ответ пользователя можно определить только по возвращаемому значению. Ширина поля ввода игнорируется.

Пример:

```
if (GetTRUE (TRUE,"Хотите напечатать отчет?"))
  OutReport;
end;
```

Стандартные процедуры вывода

Print (...)

Процедура выводит в отформатированном виде значения переменных, переданных в качестве параметров. Количество параметров не ограничено.

При записи параметров могут быть указаны спецификаторы форматирования. Если список параметров пуст, эта процедура не выполняет никаких действий.

PrintGlobs (…)

Процедура выводит в стандартный выходной поток информацию обо всех глобальных переменных. Для каждой переменной указывается название, имя модуля, в котором определена переменная, тип и значение.

PrintFiles (…)

Для каждого модуля текущего экземпляра RSL процедура выводит в стандартный выходной поток следующую информацию:

- имя модуля (Module name);
- тип модуля (Type);
- имя файла (File name).

В качестве типа модуля могут выступать следующие значения:

- ***Internal*** – модуль реализован в ядре RSL или непосредственно в использующем его RSL-приложнении. В этом случае имя файла не указывается.
- ***MAC*** – модуль реализован на языке RSL. Указанный тип модуля может загружаться из файла базы данных *btrmac.ddf*. В этом случае в качестве имени файла указывается имя модуля, затем – фраза "*from btrmac.ddf*" (например, *lib from btrmac.ddf*). Если загрузка файла производится не из указанной базы данных, выводится полное имя файла (например, *mac\privinc.mac*).
- ***RSM*** – прекомпилированный модуль. В качестве имени файла указывается имя соответствующего MAC-файла (например, *d:\user\rsl\0.mac*).
- ***DLM*** – DLM-модуль. Указывается полный путь к файлу, реализующему указанный модуль (например, *D:\user\rsl\rsd.D32*).

PrintLn (...)

Действия этой процедуры аналогичны действиям процедуры ***print***. Отличие заключается в том, что после вывода значений переменных выполняется возврат каретки.

PrintLocs (…)

Процедура выводит информацию о локальных переменных процедуры, из которой вызвана **PrintLocs**. Если процедура вызвана непосредственно из кода инициализации модуля, дополнительно указывается информация о глобальных переменных указанного модуля. Для каждой переменной указывается название, тип и значение.

PrintModule (modName:String)

Процедура для заданного модуля ***modName*** выводит информацию о типе модуля и имени файла, из которого загружен указанный модуль.

PrintProps (object)

Процедура выводит в стандартный выходной поток имя, тип и значение каждого свойства объекта ***object***.

PrintRefs (ob: Object)

Процедура выводит в стандартный выходной поток информацию об объектах, ссылающихся на объект ***ob***.

PrintStack (…)

Процедура выводит информацию о стеке вызова процедур для контекста, в котором вызвана **PrintStack**:

- выводится название всех процедур в стеке;
- предоставляется информация о локальных переменных каждой процедуры;
- для каждой локальной переменной указывается название, тип и значение.

PrintSymModule (symbolName:String)

Процедура выводит в стандартный выходной поток RSL информацию о переменной, процедуре или классе, имя которого задано параметром ***symbolName***. Указывается вид сущности (процедура, переменная или класс) и информация о модуле, в котором объявлена указанная RSL-сущность.

Message (...)

Данная процедура принимает произвольное количество параметров и выводит в нижнюю строку экрана сформированную из них отформатированную строку. При записи параметров могут быть указаны спецификаторы форматирования.

Если список параметров пуст, эта процедура не выполняет никаких действий.

SetOutput ([string] [, bool])

Процедура устанавливает новый файл для вывода, имя которого задается параметром ***string***.

Если параметр ***bool***:

- отсутствует или равен ***FALSE***, то все данные из файла удаляются;
- равен ***TRUE***, то новая информация дописывается в конец файла, заданного параметром ***string***.

После выполнения этой процедуры весь вывод процедур ***Print, Println*** или инструкции вывода направляется в указанный файл.

Для того чтобы вернуть вывод в стандартный файл, необходимо задать процедуру со следующими параметрами:

```
SetOutput ( NULL, TRUE)
```

Пример.

```
oldFile = SetOutput (“My.out”);
[  Этот текст выводится в файл My.out ];
SetOutput (NULL,TRUE);
[  Этот текст выводится в стандартный файл вывода];
```

Процедура ***SetOutput*** возвращает имя файла, в который выполнялся вывод до ее вызова.

SetColumn (integer)

Процедура формирует отчет с переменным числом колонок.

Параметр ***integer*** устанавливает относительный номер колонки, в которую будет осуществляться весь вывод процедур ***Print, Println*** и инструкции вывода.

Процедура ***SetColumn*** формирует отчет и сохраняет его в памяти компьютера. Для того чтобы информация попала в выходной файл, необходимо вызвать процедуру **FlushColumn.**

FlushColumn

Процедура выводит информацию из сформированного при помощи процедуры ***SetColumn*** отчета в файл вывода.

Пример:

```
macro PrintHead (n)
  if (n == 0)
     ch = "|"; ch2 = "|"
  else
     ch = "-"; ch2 = ""
  end;
[#-----------|
 #   ##      |
 #-----------|] (ch,ch2,n,ch)
end;
macro PrintCol (n,str)
  if (n == 0)
     [| ## ###### |] (n,str:w)
  else
     [  ## ###### |] (n,str:w)
  end;
end;
NumColumn = 5;
i = 0;
while (i < NumColumn)
  SetColumn (i);
  PrintHead (i);
  i = i + 1;
end;
FlushColumn;
i = 0;
while (i < NumColumn)
```

```
  SetColumn (i);
  PrintCol (i,"Эту очень длинную строку мы будем выводить во все
колонки !!!");
  i = i + 1;
end;
SetColumn (4);
[ 4 колонка ];
SetColumn (2);
[ 2 колонка ];
FlushColumn;
```

ClearColumn

Процедура удаляет все данные из буфера печати.

Данная процедура работает совместно с процедурами ***SetColumn*** и ***FlushColumn***.

SetDefPrec (integer)

Процедура устанавливает количество отображаемых по умолчанию знаков после точки для вывода значений типа Double и DoubleL.

Установленное значение будет действовать до следующего вызова этой процедуры.

SetOutHandler (NameMacro)

Процедура устанавливает обработчик стандартного вывода.

Имя или ссылка на макропроцедуру обработки вывода передаются в качестве параметра ***NameMacro***. Для установки стандартного обработчика необходимо вызвать ***SetOutHandler*** без параметров.

Примеры:

```
1) /* Пример asp-файла, в котором все вызовы print
перенаправляются в  Response.Write. */
<%@ Language=RSLScript %>
<html>
<head>
<title>New Page 1</title>
</head>
<body>
<%
Macro MyOutProc (str)
  Response.Write (str+"<br>")
End;
SetOutHandler (@MyOutProc);
print (Request.Form ("Name").item);
print (Request.Form ("Address").item);
print (Request.Form ("Info").item);
print (Request.Form ("Password").item);
%>
</body>
```

```
</html>
2) /* Пример перенаправления стандартного вывода в пайп. */
file ff () write txt;
Macro MyOutProc (str)
  insert (ff,str,strlen (str), true)
End;
pipeName = "\\\\" + getEnv ("COMPUTERNAME") + "\\pipe\\brmon.";
if (open (ff,pipeName))
  SetOutHandler (@MyOutProc);
  println ("Brmon.mac executed");
end;
```

GetPRNInfo ([escSeq:String, banner:String, frmFeed:Bool]):String

Процедура возвращает имя используемого устройства печати.

В параметре ***escSeq*** возвращается используемая ESC-последовательность.

В параметре ***banner*** возвращается используемый файл заголовка.

В параметре ***frmFeed*** возвращается признак перевода формата.

SetPRNInfo (prnName:String [, escSeq:String, banner:String, frmFeed:Bool])

Процедура устанавливает новое имя используемого принтера ***prnName***, ESC-последовательность ***escSeq***, используемый файл заголовка ***banner*** и признак перевода формата ***frmFeed***.

Преобразование типов значений переменных

Процедуры этой группы выполняют преобразование типа переменной.

Каждая из процедур получает в качестве параметра некоторую величину и возвращает уже преобразованное значение. Если процедура не может выполнить преобразование, она возвращает нулевое значение соответствующего типа.

ValType (val)

Процедура возвращает код типа переменной, переданной через параметр ***val.*** В зависимости от типа данных возвращаемым значением может быть одно из следующих значений:

- V_UNDEF
- V_INTEGER
- V_MONEY
- V_DECIMAL
- V_DOUBLE
- V_STRING
- V_BOOL
- V_DATE
- V_TIME
- V_DTTM
- V_FILE
- V_STRUC
- V_ARRAY
- V_TXTFILE
- V_DBFFILE
- V_PROC
- V_R2M
- V_MEMADDR

Пример:

```
if (ValType (MyVar) == V_UNDEF)
        MsgBox ("Значение переменной MyVar не
        определено")
end
```

Double (val)

Процедура возвращает преобразованное в тип Double значение типа Integer, Money или String, переданное через параметр ***val***.

DoubleL (val)

Процедура возвращает преобразованное в тип DoubleL значение типа Integer, Money, Double или String, переданное через параметр ***val***.

Int (val)

Процедура возвращает преобразованное в тип Integer значение типа Double, String, Date и Time, переданное через параметр ***val***.

После преобразования числа с плавающей точкой процедура возвращает целую часть числа, дробная часть отбрасывается. Для преобразования Double в Integer с округлением используется процедура **Round()**.

При преобразовании даты в целое число процедура возвращает значение, равное количеству дней от начала нашей эры до заданной параметром ***val*** даты.

При преобразовании времени в целое число процедура возвращает значение, равное количеству сотых долей секунды во времени, заданном параметром ***val***.

String (val {, val})

Данная процедура принимает произвольное количество параметров и возвращает сформированную из них отформатированную строку.

При записи параметров, так же, как и в процедурах ***print*** и ***println***, могут быть указаны специальные символы форматирования.

Money (val)

Процедура возвращает преобразованное в тип Money значение параметра типа String, Integer, Double или DoubleL.

MkStr (val1, val2)

Процедура возвращает строку типа String конкретной длины, состоящую из заданного символа-заполнителя.

Вид символа-заполнителя определяется первым параметром ***val1***, который может быть числом или строкой. Если это число имеет тип Integer, то оно будет расценено как код символа-заполнителя. Если параметр ***val1*** представляет собой строку (тип String), то в качестве заполнителя используется ее первый символ.

Второй параметр ***val2*** задает длину возвращаемой строки.

Floor (val)

Процедура возвращает наибольшее целое число, меньшее или равное заданному параметру. Данная процедура применяется для значений типа Double, DoubleL или Money.

Asize (val [, newsize])

Процедура возвращает текущий размер массива ***val***. Если указан второй параметр, то старый массив удаляется, и распределяется пустой массив размером ***newsize***.

Date [(day, mon, year)]

Процедура возвращает переменную типа Date.

Можно передать от одного до трех параметров типа Integer. Первый задает число, второй – месяц, третий – год. Переданные значения заменят соответствующие им части текущей даты. Если нужно изменить только год и оставить текущий день и месяц, необходимо передать в качестве первого и второго параметров значение любого типа, отличного от Integer.

Процедуре также можно передать в качестве первого параметра строку вида "dd.mm.yy" или "dd.mm.yyyy", в результате переданное значение будет преобразовано к соответствующей дате.

Процедура также может принимать в качестве параметра значения типа Date и Dttm, которые будут преобразованы в переменную типа Date.

Если при вызове процедуры параметры не указаны, то процедура возвращает одно из значений:

- текущую дату – если в файле ***rsreq.ini*** параметр TESTMODE = 0;
- значение "31.12.2099" – если параметр TESTMODE = 1.

Пример:

```
aa = date;                  /*Переменная aa равна текущей дате */
aa = date ("10.03.1994");   /* Переменная aa равна 10 марта 1994 года */
aa = date (10);             /* 10 число текущего месяца и года */
aa = date (10,3);           /* 10 число 3 месяца текущего года */
aa = date ("",3,1994);      /* Текущий день марта1994 года */
```

DateSplit (date, day, mon, year)

Процедура получает в качестве первого параметра ***date*** переменную типа Date и возвращает через параметры ***day, mon, year***, имеющие тип Integer, соответственно, день, месяц и год для заданной даты.

Если нужно получить только какую-нибудь одну составляющую даты, пропущенные компоненты заменяются нулевыми значениями, а конец списка опускается.

Пример:

```
datesplit (date, NULL, mon);
```

Time [(hour, min, sec, msec)]

Процедура возвращает переменную типа Time.

Можно передать от одного до четырех параметров типа Integer. Первый задает часы, второй – минуты, третий – секунды, четвертый – сотые доли секунды. Переданные значения заменят соответствующие им части текущего времени. Передача не всех параметров осуществляется так же, как для процедуры ***Date***.

Процедуре также можно передать в качестве первого параметра строку вида "hh:mm:ss.nn", "hh:mm:ss" или "hh:mm", в результате переданное значение будет преобразовано к соответствующему времени. Если время задается строкой вида "hh:mm:ss.nn", то процедуре можно передать второй строковый параметр, задающий символ-разделитель между секундами и сотыми долями секунды. По умолчанию используется символ ".".

Процедура может принимать в качестве параметра значения типа Time и Dttm, которые будут преобразованы в переменную типа Time.

Если при вызове процедуры параметры не указаны, то процедура возвращает одно из значений:

- текущую дату – если в файле ***rsreq.ini*** параметр TESTMODE = 0;
- значение "31.12.2099" – если параметр TESTMODE = 1.

TimeSplit (time, hour, min, sec)

Процедура принимает в качестве первого (входного) параметра переменную типа Time и возвращает в выходных параметрах ***hour***, ***min***, ***sec*** типа Integer компоненты заданного времени: часы, минуты и секунды, соответственно.

Допускается указывать в выходных параметрах только интересующие пользователя компоненты времени.

DtTm (date, time)

Процедура принимает в качестве первого параметра переменную типа Date, в качестве второго - переменную типа Time и возвращает величину типа Dttm, состоящую из двух компонент: даты и времени.

DtTmSplit (d, dt, tm)

Процедура получает в качестве первого параметра величину ***d*** типа Dttm и возвращает две ее составляющие: дату и время через параметры ***dt*** и ***tm***.

В качестве первого параметра также можно передать значение типа Date или Time, в этом случае процедура вернет дату либо время в соответствующем параметре.

RubToStr (money [, rub, kop [, full]], kol)

Процедура принимает в качестве первого параметра переменную типа Money. Второй и третий параметры процедуры необязательны. Если они заданы, то процедура инициализирует переменную ***rub*** строкой, содержащей количество рублей прописью, а переменную ***kop*** – строкой, содержащей количество копеек цифрами. Параметр ***kol*** задает количество знаков после десятичной точки; по умолчанию значение параметра равно 2.

Процедуре можно указать на необходимость вывода слов рублей и копеек в несокращенном виде. Признаком этого является значение типа Bool, равное TRUE и переданное в качестве второго или четвертого параметра.

Текст, выдаваемый этой процедурой, можно изменить с помощью соответствующих настроек в файле *locale.ini*. Эти настройки используются в том случае, если национальная валюта отлична от рубля.

Процедура возвращает строку, содержащую сумму прописью.

Пример:

```
/* Вариант 1 */
Var a = 121.11, b, c;
println(RubToStr(a*100.00, b, c)); /* Результат : Сто двадцать один
руб. 11 коп. */
/* Вариант 2 */
Var a = 121.11, b = TRUE, c;
println(RubToStr(a*100.00, b, c)); /* Результат : Сто двадцать один
рубль 11 копеек */
/* Вариант 3  */
Var str, rub, kop;
str = rubtostr ($12.34, true);
str = rubtostr ($12.34, rub, kop, true);
rub = true;
str = rubtostr ($12.34,rub,kop);
/*Все три варианта распечатают названия полностью.*/
```

RubToStrAlt (money, [rub, kop,] kol)

Процедура принимает в качестве первого параметра переменную типа Money. Второй и третий параметры процедуры необязательны. Если они заданы, то процедура инициализирует переменную ***rub*** строкой, содержащей количество рублей прописью, а переменную ***kop*** – строкой, содержащей количество копеек цифрами. Параметр ***kol*** задает количество знаков после десятичной точки; по умолчанию значение параметра равно 2.

Текст, выдаваемый этой процедурой, можно изменить с помощью соответствующих настроек в файле *locale.ini*. Эти настройки используются в том случае, если национальная валюта отлична от рубля.

Процедура возвращает строку, содержащую сумму прописью. Рубли и копейки указываются с учетом падежа: рублей (рубля), копеек (копейки).

CurToStrAlt (money [, rub, kop], ISO, kol)

Процедура принимает в качестве первого параметра переменную типа Money, в качестве предпоследнего параметра – цифровой ISO-код валюты. Второй и третий параметры процедуры не обязательны. Если они заданы, то процедура инициализирует переменную ***rub*** строкой, содержащей количество основных денежных единиц прописью, а переменную ***kop*** – строкой, содержащей количество дробных денежных единиц прописью.

Параметр ***kol*** задает количество знаков после десятичной точки; по умолчанию значение параметра равно 2.

Процедура возвращает строку, содержащую сумму прописью. Основные и дробные денежные единицы указываются сокращенно.

Для работы процедуры необходим файл *locale.ini*, содержащий цифровые ISO-коды валют с соответствующими им падежными формами основных и дробных денежных единиц для перевода их численного представления в прописное.

MonName (mon)

Процедура возвращает название месяца, номер которого (от 1 до 12) передан через параметр ***mon*** типа Integer.

NumToStr (val, n1:String, n2:String, n3:String, isMan:Bool, prec:Integer) :String

Процедура возвращает значение числа ***val*** прописью.

Параметры ***n1***, ***n2*** и ***n3*** задают три возможных вида сущности, количество которой задается параметром ***val***.

Параметр ***isMan*** имеет значение TRUE для сущности мужского рода и FALSE для сущности женского рода. По умолчанию параметр принимает значение FALSE.

Параметр ***prec*** задает максимальное количество знаков после десятичной точки. Может быть задано от 0 до 5 знаков. По умолчанию используется значение 2.

Пример.

```
println (NumToStr (125.45,"болт", "болта", "болтов", true,3) );
/* Выводит: */
сто двадцать пять целых сорок пять сотых болта
```

SetAutoMoneyFloor (auto:Bool):Bool

Процедура устанавливает режим автоматического отбрасывания десятых и сотых долей копеек в денежных типах при выполнении операций над ними, если параметр ***Auto*** равен TRUE. В противном случае десятые и сотые доли не отбрасываются. По умолчанию в системе установлен режим отбрасывания десятых и сотых долей.

Процедура возвращает старое значение режима.

Round (val:Variant [, pos:Integer] , [round:Integer]):Money

Процедура предназначена для округления введенного значения.

Параметры:

val – значение типа **Money**, для которого следует выполнить округление. Если переданное значение имеет тип **Numeric**, необходимо указать значение параметра ***round***.

pos – количество знаков после запятой. Возможные значения:

- **0** – округление до рублей;
- **2** – округление до копеек.

round – характер округления. Возможные значения:

- **1** – округление в большую сторону;
- **2** – округление в меньшую сторону.

Параметр указывается только, если в параметре ***val*** указано значение типа **Numeric**.

Процедуры для работы с типом DoubleL

Раздел содержит описание процедур, предназначенных для работы с типом данных DoubleL (long double). Данные указанного типа могут присутствовать в базе данных RS-Bank Pervasive; их невозможно представить в стандартном для RSL формате описания базы данных (в файлах *ddf*). При конвертировании словаря *def* в *ddf*, все поля типа long double (10 байт) преобразуются в три целочисленных поля.

MakeDouble (p1:Integer, p2:Integer, p3:Integer):Double

Процедура служит для получения значения типа Double из трёх указанных полей (**p1** ,**p2**, **p3**).

MakeMoney (p1:Integer, p2:Integer, p3:Integer):Money

Процедура служит для получения значения типа Money из трёх указанных полей (**p1** ,**p2**, **p3**).

SplitMoney (mn:Money, p1:@Integer, p2:@Integer, p3:@Integer)

Указанная процедура является обратной процедуре **MakeMoney** и предназначена для преобразования величины типа Money в три целых числа, возвращаемых в параметры **p1** ,**p2**, **p3**.

Работа со строками

Процедуры этой группы выполняют некоторые специфические действия над переменными типа String.

При работе с символьными строками считается, что первый символ находится в позиции с номером 1.

StrLen (string)

Процедура возвращает длину строки символов, заданной в качестве параметра.

Index (srcStr:String, fnd:String [, startPos:Integer]):Integer

Процедура возвращает номер позиции строки ***fnd*** в строке ***srcStr*** или 0, если строка ***fnd*** не найдена в ***srcStr***.

Параметр ***startPos*** задаёт номер позиции в строке ***srcStr***, начиная с которой необходимо выполнить поиск. Минимальный номер позиции равен **1**. Если параметр не задан, поиск выполняется с начала строки ***srcStr***.

StrBrk (string1, string2)

Процедура возвращает позицию первого из символов в строке ***string1***, который содержится также и в строке ***string2***. Если в строке ***string1*** нет ни одного из символов строки ***string2***, процедура возвращает **0**.

StrIsNumber(str:String):Bool

Процедура предназначена для выполнения проверки, определяющей, являются ли все символы строки цифрами.

Параметры:

str – строка символов.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – если все символы в строке являются цифрами;
- ***FALSE*** – если в качестве значения параметра ***str*** указана пустая строка или хотя бы один из символов строки не является цифрой.

SubStr (string, integer1 [, integer2])

Процедура возвращает часть строки ***string***, которая начинается с позиции ***integer1***. Третий параметр не обязателен. Если он указан, то извлекается подстрока длиной ***integer2***. В противном случае извлекается подстрока с позиции ***integer1*** до конца строки ***string***. При неправильно заданных параметрах возвращается пустая строка.

StrSet (string1, integer, string2)

Процедура записывает строку ***string2*** в строку ***string1***, начиная с позиции строки ***string1***, определенной вторым параметром типа Integer.

Trim (string)

Процедура удаляет начальные и конечные пробелы из строки типа String, заданной в качестве параметра, и возвращает результат.

StrSplit (string, array, len [, len_first] [, minseg])

Процедура разбивает длинную строку, заданную в качестве первого параметра типа String, на несколько сегментов. Результат помещается в одномерный массив, определенный вторым параметром.

Длина каждого сегмента задается третьим параметром ***len***, имеющим тип Integer. Если к тому же задан параметр ***len_first***, он используется как длина первого сегмента массива.

Указав последний параметр процедуры ***minseg***, можно задать минимальное число сегментов. В том случае, если реальное число созданных сегментов меньше заданного минимального, лишние элементы массива ***array*** инициализируются пустыми строками.

StrSplit2 (source:String, len:Integer [, len_first:Integer] [, minseg:Integer]) : TArray

Процедура аналогочна процедуре ***StrSplit*** и возвращает объект типа ***TArray***, содержащий полученные строки.

StrSplit2 (str:String, len:Integer, isFix:Bool, minseg:Integer):TArray

Использование указанного варианта процедуры позволяет разбить строку на сегменты фиксированного размера без переноса по словам. Для этого следует в качестве параметра ***isFix*** передать значение TRUE.

Пример:

```
ar = StrSplit2("Иванов Иван Иванович", 8, true);
for (a, ar)
  println (a);
end;
```

Результат:

```
Иванов И
ван Иван
ович
```

StrUpr (string [,len])

Процедура возвращает копию строки ***string,*** преобразованную в символы верхнего регистра.

Необязательный параметр ***len*** задает количество символов, которые необходимо преобразовать.

Если параметр ***len*** не указан, все символы строки ***string*** будут преобразованы в символы верхнего регистра.

Пример:

```
TestLine = "Эта Строка Набрана В Разных Регистрах";
println ("Нормальная строка – ",TestLine);
println ("В верхнем регистре – ",StrUpr (TestLine));
```

StrLwr (string [, len])

Процедура возвращает копию строки ***string,*** преобразованную в символы нижнего регистра.

Необязательный параметр ***len*** задает количество символов, которые необходимо преобразовать.

Если параметр ***len*** не указан, все символы строки ***string*** будут преобразованы в символы нижнего регистра.

Пример:

```
TestLine = "Эта Строка Набрана В Разных Регистрах";
println ("Нормальная строка – ",TestLine);
println ("В нижнем регистре – ",StrLwr (TestLine));
```

CodeFor (string)

Процедура возвращает код ASCII для первого символа в строке ***string***.

Пример:

```
code = CodeFor (“A”);
```

StrFor (number)

Процедура возвращает строку, состоящую из одного символа, код ASCII которого равен ***number***.

Пример:

```
str  = StrFor (65);
```

StrSubst (sourse, strToFind, strToReplace)

Процедура ищет в строке ***source*** подстроки ***strToFind*** и заменяет их строками ***strToReplace***. Возвращаемым значением является результирующая строка.

Пример:

```
Str = StrSubst (oldStr, "\n", " ")
```

ToOEM (string [, mode])

Процедура конвертирует строку, заданную параметром ***string***, из ANSI в OEM. Если значение параметра ***mode*** равно TRUE, выполнение конвертирования осуществляется в любом режиме. Если параметр ***mode*** не задан, конвертирование строк в режиме Unicode выполняется, только если в файле *rsreq.ini* задан параметр FORCE_OEM_ANSI_CVT = Yes. Возвращаемым значением является преобразованная строка.

ToANSI (string [, mode])

Процедура конвертирует строку, заданную параметром ***string***, из OEM в ANSI. Если значение параметра ***mode*** равно TRUE, выполнение конвертирования осуществляется в любом режиме. Если параметр ***mode*** не задан, конвертирование строк в режиме Unicode выполняется, только если в файле *rsreq.ini* задан параметр FORCE_OEM_ANSI_CVT = Yes. Возвращаемым значением является преобразованная строка.

Пример:

```
s = ToANSI (oemstr, true)
```

Параметры процедур

Процедуры этой группы предоставляют дополнительные возможности для работы со списком параметров процедур языка RSL. Примеры их использования описаны в разделе "Передача параметров" (см. стр. 41). Первый в списке параметр имеет номер 0.

GetParm (num, var)

Процедура присваивает переменной ***var*** значение фактического параметра вызывающей процедуры с номером ***num***.

В случае успешного выполнения возвращается TRUE. При неудаче возвращается FALSE, и переменная ***var*** получает значение неопределенного типа V_UNDEF.

Параметр процедуры ***var*** не должен быть описан как константа.

Примечание.
При использовании этой процедуры в методе класса следует помнить, что в метод неявно первым параметром с номером 0 передается ссылка на объект (this). Таким образом, первый параметр метода имеет номер 1.

IsOutParm (num:Integer):Bool

Процедура определяет, является ли заданный параметр выходным или только входным.

Параметры:

num – параметр, который требуется проверить.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – переданный параметр является выходным (и входным).
- FALSE – переданный параметр является только входным.

Пример:

```
Macro Test (a, b c)
  If (IsRef (0))
     SetParm (0, “new value”)
  End
End;
```

SetParm (num, expression)

Процедура присваивает фактическому параметру вызывающей процедуры с номером ***num*** значение, заданное в параметре ***expression***.

В случае успешного выполнения возвращается TRUE. При неудаче возвращается FALSE, и фактический параметр процедуры получает значение неопределенного типа V_UNDEF.

Фактический параметр в вызывающей процедуре должен быть описан как переменная. Если же он является константой, его значение не изменится, при этом система не сгенерирует сообщения об ошибке.

Parmcount ()

Процедура возвращает количество переданных RSL-процедуре параметров.

Математические процедуры

В языке RSL предусмотрены процедуры для вычисления стандартных математических функций.

Exp (number)

Log (number)

Log10 (number)

Pow (number, number)

Sqrt (number)

Abs (number)

Min (number, number)

Процедура вычисляет минимальное из двух чисел.

Max (number, number)

Процедура вычисляет максимальное из двух чисел.

Параметры:

В качестве параметров процедуры по умолчанию используются значения типа Double. Если этим процедурам передать значения любого другого типа, то они будут преобразованы в тип Double. Исключение составляет значение типа DoubleL (long double), которое остается без изменений.

Возвращаемое значение:

Все математические процедуры возвращают значение типа Double. Однако, если в качестве параметра передано значение типа DoubleL, то результатом процедуры также будет значение типа DoubleL.

mod (val1:Variant, val2:Variant):Variant

Процедура возвращает остаток от деления величины ***val1*** на величину ***val2***.

Параметры:

В качестве параметров процедуры используются значения типа Double и Integer.

Внешние программы

При работе с языком RSL можно использовать внешние программы, написанные на любых других языках программирования. Внешняя программа должна быть представлена в виде файла с расширением *.exe* или *.com*. Для ее вызова применяется специальная встроенная процедура:

Run (prog, parm, init, finish:String])

Процедура предназначена для запуска внешней программы.

Параметры:

Примечание.
*Параметры **parm**, **init**, **finish** могут быть пропущены, вместо них следует указать **Null**, например, run ("myprog.com",NULL,NULL,"Нажмите любую клавишу").*

prog – имя внешней программы. При записи параметра расширение *".exe"* в имени внешней программы можно не указывать. Поиск программы производится сначала в текущем каталоге, потом в каталогах, указанных в переменной среды PATH. Для обработки командных файлов необходимо запустить программу *command.com* и передать ей в качестве параметра имя файла:

Пример:

```
run (GetEnv("COMSPEC"),"/c " + "my.bat");
```

parm – командная строка, передаваемая во внешнюю программу, указанную в параметре ***prog***.

init – сообщение, которое необходимо вывести на экран в момент запуска программы. Параметр может включать следующие символы:

Примечание.

*Если строка сообщения начинается не с символов "<" или ">" (например, "Строка для отображения>c:\\mydir\\myout.txt"), все символы строки **до** указанных символов отображаются в консольном режиме перед запуском программы.*

- ">" – Символ используется для перенаправления стандартного вывода в требуемый файл; имя файла задается после указанного символа. В случае, когда необходимо использовать стандартный вывод программы в выходной файл RSL, имя файла не указывается.

Пример:

">c:\\mydir\\myout.txt" – выход программы будет произведен в указанный файл

"Begin>" – выход программы направлен в стандартный выход RSL.

- "<" – Символ используется для перенаправления стандартного ввода, имя файла задается после указанного символа.

Пример:

“<c:\\mydir\\myin.txt".

- ">>" – Символ дозаписи. Если в качестве параметра передать строку типа ">><имя файла>", будет произведена дозапись в указанный файл.

Пример:

*run (GetEnv("COMSPEC"),”/c dir *.*”, “>>e:\remote.out”);*

Имеется возможность одновременно задать имена файлов для перенаправления ввода и вывода. В этом случае порядок указания файлов не имеет значения.

Пример:

">c:\\mydir\\myout.txt<c:\\mydir\\myin.txt"

"> <c:\\mydir\\myin.txt"

"Строка для отображения>c:\\mydir\\myout.txt <c:\\mydir\\myin.txt"

finish – сообщение, которое необходимо вывести на экран после завершения программы. При использовании указанного параметра перед возвратом в интерпретатор RSL будет сделана пауза.

Возвращаемое значение:

Процедура возвращает одно из значений:

- код возврата внешней программы – в случае успешного выполнения;
- **-1**, если указанная программа не найдена.

Пример:

//Пример запуска процедуры xcopy с перенаправленным входным файлом, содержащим символ ответа на запрос ‘y’:

run ("xcopy", "/F d:\\user\\cmd.mac d:\\user\\cmd2.", "> < d:\\user\\yes.txt", "Press a key");*

Результат в выходном файле RSL:

Overwrite D:\user\cmd2.mac (Yes/No/All)? y

D:\user\cmd.mac -> D:\user\cmd2.mac

1 File(s) copied

Удаленный запуск макропрограмм

При работе в трехзвенной архитектуре "Клиент-сервер" пользователь может из макропрограмм, выполняющихся на сервере приложений, запускать RSL-программы на терминале. Это осуществляется посредством специальной процедуры **CallRemoteRsl**, входящей в модуль *rsexts*.

Внимание!
Для того, чтобы RSL-программа запускалась на терминале необходимо в макрофайле явно импортировать модуль rsexts в макрофайле.

CallRemoteRsl (fileName [, procName [, parm1, parm2, ...]])

Параметры:

fileName – строка с именем макрофайла, который должен быть выполнен на терминале.

procName – необязательное имя процедуры из файла ***fileName***, которая должна быть вызвана.

parm1, **parm2 –** параметры, передаваемые процедуре ***procName***.

Возвращаемое значение:

Возвращаемым значением процедуры ***CallRemoteRsl*** является значение, возвращаемое процедурой ***procName***.

Параметры, передаваемые процедуре ***procName***, и ее возвращаемое значение должны быть одного из простых типов данных RSL. Не допускаются параметры и возвращаемое значение объектного типа (FILE, RECORD, OBJECT).

Пример:

```
/* Макрофайл на сервере: */
import rsexts;
println (CallRemoteRsl ("remote.mac" ,"Main", $1.33L ,200 ));
/* Макрофайл на терминале (с именем remote.mac): */
MsgBox ("This is remote.mac");
macro Main (p1,p2)
    MsgBox ("Macro Main called with ", p1,"  ",p2);
    return "Main Return value"
end;
```

В случае возникновения ошибок компиляции или выполнения на терминале серверная часть генерирует ошибку, возникшую во время выполнения.

Следует иметь в виду, что для макрофайла на терминале организуется отдельный файл вывода, таким образом, результаты работы инструкций вывода и др. попадут не в файл вывода вызывающего файла на сервере, а в файл вывода на терминале.

Обработка меню

Menu (array [, prompt] [, head] [, x] [, y] [, n])

Процедура формирует на экране меню для выбора варианта.

Текст каждой строки меню необходимо определить в элементах массива ***array***, который является первым параметром и имеет тип V_ARRAY.

Во втором параметре ***prompt*** задается сообщение, выводимое в нижнюю строку экрана вместе с меню.

Параметр ***head*** задает текст, выводимый в заголовок окна. Если этот параметр не задан, в качестве заголовка окна используется строка: "Меню".

Следующие два параметра ***x*** и ***y*** задают координату на экране левого верхнего угла окна меню. Если эти параметры не заданы, окно меню располагается по центру экрана.

Параметр ***n*** определяет номер текущего активного пункта меню.

Процедура возвращает номер элемента массива ARRAY, соответствующего выбранному варианту.

Примеры:

1) Пример иллюстрирует формирование меню в заданной позиции экрана с заголовком и подсказкой по умолчанию:

```
array  a;
a (0) = “Первый пункт”;
Выбор = Menu (a, NULL, NULL, 4, 6)
```

2) Пример иллюстрирует формирование меню, имеющее заголовок "Заголовок" и подсказку "Подсказка", расположенное по центру экрана. В этом случае активным станет третий пункт экранного меню (нумерация пунктов начинается с нуля):

```
Array mn;

/* Формируем пункты меню */

mn = "Первый пункт";

mn = "Второй пункт";

mn = "Третий пункт";

Menu (mn, "Подсказка", "Заголовок", NULL, NULL, 2)
```

RunMenu (menuName, procName | procAddr [, lbrName])

Процедура активизирует выполнение меню, созданного в специальной библиотеке ресурсов (см. стр. 73).

Имя меню в библиотеке ресурсов определяется параметром ***menuName***.

Во втором параметре задается строка с именем процедуры обработки команд меню ***procName*** или ссылка на данную процедуру ***procAddr***.

С помощью третьего параметра при необходимости можно указать имя альтернативной библиотеки ресурсов.

Пример:

Предположим, что в библиотеке создано меню Demo, имеющее два пункта:

- *выход, с идентификатором 101;*
- *печать отчета, с идентификатором 102.*

Код обработки такого меню выглядит следующим образом:

```
Macro HandleMenu (id)
        If (id == 101) /* Завершить работу меню */
                Return TRUE;
        Elif (id == 102) /* Напечатать отчет */
```

```
                RunReport
            End;
        Return FALSE;
        /* Возвращать FALSE для необрабатываемых идентификаторов */
        End;
        RunMenu ("Demo", @HandleMenu);
```

Процедура обработки вызывается автоматически, когда пользователь выбирает пункт меню. В качестве параметра эта процедура получает целочисленный идентификатор выбранного пункта ***id***, определенный при создании меню (см. стр. 73). Если в ответ она возвращает значение ***TRUE***, то это же значение возвращает и процедура ***RunMenu***, и обработка меню завершается.

При нажатии клавиши [Esc] RSL вызывает процедуру обработки команд меню с идентификатором, равным 256. Если она вернет NULL или не вернет никакого значения, то обработка меню завершится, и ***RunMenu*** возвратит значение ***FALSE***. Если же завершение обработки меню по клавише ESC нежелательно, то для идентификатора, равного 256, или для всех необрабатываемых идентификаторов (см. пример), нужно возвращать значение ***FALSE***.

Обработка диалоговых окон

AddScroll (dlg:TRecHandler, data:Object [, numCol:Integer, colArray:TArray , proc:Variant, rdOnly:Bool, focused:Bool], [NumBar:Integer])

Процедура позволяет создать область прокрутки диалоговой панели. Используется при обработке события **DLG_PREINIT**.

Параметры:

dlg – объект класса ***TRecHandler*** или производный от него. Указанный объект должен содержать описание диалоговой панели.

data – источник информации для окна прокрутки: объект класса ***RsdRecordset*** или класса, производного от ***ToolsDataAdapter***. Объект класса ***RsdRecordset*** должен реализовывать статический клиентский курсор.

numCol – количество колонок, отображаемых в области прокрутки, информация для которых содержится в массиве ***colInfo***. Если значение параметра равно **0**, колонки создаются автоматически для каждого поля в источнике данных ***data***. В этом случае в качестве заголовка колонки используется название поля.

colInfo – массив, задающий атрибуты колонок области прокрутки. Для каждой колонки атрибуты записываются в 6 последовательных элементов массива следующим образом:

- **0** – имя поля в источнике данных, обязательный атрибут;
- **1** – заголовок колонки, если элемент не задан, в качестве заголовка используется название поля;

- **2** – начальная ширина колонки, если элемент не задан, используется автоматическое вычисление ширины колонки;
- **3** – тип поля, возможные значения:
  - **1** – редактируемое (значение по умолчанию);
  - **2** – нередактируемое;
  - **5** – прокручиваемое.
- **4** – количество знаков после точки для числовых полей;
- **5** – зарезервировано.

proc – используемый обработчик сообщений. Если обработчик не требуется, указывается значение ***null***. В качестве обработчика могут быть указаны: имя RSL процедуры, ссылка на RSL процедуру, ссылка на метод RSL объекта, полученная при помощи метода ***R2M***.

rdOnly – признак, определяющий возможность редактирования сведений в области прокрутки. Возможные значения:

- ***TRUE*** – редактирование запрещено, независимо от типа колонки.
- ***FALS***E – редактирование разрешено.

focused – признак, позволяющий управлять фокусом ввода. Возможные значения:

- ***TRUE*** – при отображении диалоговой панели фокус ввода устанавливается на область прокрутки.
- ***FALSE*** – фокус ввода устанавливается на первое поле или поле, заданное при обработке сообщения **DLG_INIT**.

NumBar – номер элемента типа Bar, расположенного на панели, в который необходимо добавить скроллинг. По умолчанию значение параметра равно "0". Нумерация элементов типа Bar и порядок работы инструмента с этими элементами зависит от того, в какой последовательности элементы добавлялись на панель.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – при успешном выполнении процедуры.
- ***FALSE*** – в случае возникновения ошибок.

RunDialog(dlg:TRecHandler [, proc:Variant], [SwitchKey:Integer]):Bool

Процедура предназначена для отображения на экране диалогового окна ***dlg***.

Примечание.
*Для программного закрытия диалоговой панели, вызванной указанной процедурой, необходимо при обработке сообщений **DLG_KEY**, **DLG_BUTTON** процедурой **evProc** вернуть значение **CM_SAVE** или **CM_CANCEL**.*

Параметры:

dlg – объект класса ***TRecHandler*** или производный от него объект, представляющий собой диалоговое окно.

proc – используемый обработчик сообщений. Если обработчик не требуется, указывается значение ***null***. В качестве обработчика могут быть указаны: имя RSL процедуры, ссылка на RSL процедуру, ссылка на метод RSL объекта, полученная при помощи метода ***R2M***.

SwitchKey – код клавиши, по которой будет выполняться переключение между панелью и скроллингом. Если параметр не передан, то переключение между панелью и скроллингом выполняется только с помощью мыши.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – в случае, если пользователь сохранил данные в диалоговом окне.
- ***FALSE*** – при возникновении ошибок.

Примечание.
Во время выполнения процедуры RunScroll не осуществляется журнализация возникающих изменений.

Пример:

```
record id (name) dialog;
RunDialog (id);

[Имя  ###############
Адрес  ###############]
(id.Name, id.Address) //диалог имеет два поля: Name и Address
```

DisableValidation (dlg:Variant, id:Integer)

Процедура отключает проверку корректности даты в поле ***id*** диалогового окна ***dlg***. Процедуру можно вызвать при обработке сообщения DLG_INIT.

UpdateFields (dlg:TRecHandler [, id:Integer])

Процедура используется для обновления значения поля с ***id*** диалоговой панели ***dlg***. Если параметр ***id*** не задан, процедура выполняет обновление данных во всех полях диалоговой панели.

SetFocus (dlg:TRecHandler, id:Integer)

Процедура устанавливает фокус ввода в поле ***id*** диалоговой панели ***dlg***.

MsgBox (mes)

Процедура выводит на экран диалоговое окно с сообщением, заданным параметром ***mes***. В качестве параметра могут быть переданы данные любого типа.

Для того, чтобы сообщение занимало несколько строк, необходимо разделить его текст на части при помощи вертикальной черты "|". Удаление сообщения с экрана производится при помощи клавиши [Esc].

Пример:

```
MsgBox ("Это первая строка|Это вторая строка");
```

DisableFields (dlg:TRecHandler [, id:Integer])

Процедура делает поле ***id*** диалоговой панели ***dlg*** недоступным для корректировки. Если параметр ***id*** не задан, процедура вводит запрет на редактирование всех полей указанной диалоговой панели.

EnableFields (dlg:TRecHandler [, id:Integer])

Процедура отменяет запрет на редактирование поля ***id*** диалоговой панели ***dlg***, установленный процедурой ***DisableFields***. Если параметр ***id*** не задан, процедура разрешает редактирование всех полей указанной диалоговой панели.

SetTimer (dlg:TRecHandler [, timeOut:Integer, set:Bool])

Процедура устанавливает таймер для диалоговой панели ***dlg***. После установки таймера обработчику сообщений этой панели через одинаковые промежутки времени, заданные в миллисекундах параметром ***timeOut,*** будет посылаться сообщение **DLG_TIMER**. По умолчанию этот параметр принимается равным 3000 (3 секунды).

Чтобы отключить ранее установленный таймер, необходимо вызвать процедуру ***SetTimer***, передав ей в качестве параметра ***set*** значение FALSE. В этом случае параметр ***timeOut*** игнорируется.

При успешном выполнении процедуры она возвращает значение TRUE, в противном случае – FALSE.

Пример:

В диалоговой панели timer с интервалом в 1 секунду инкрементируется поле Counter:

```
record tm (timer) dialog;
Macro MesProc (dlg,cmd,id)
if (cmd == DLG_INIT)
SetTimer (dlg,1000); /* Устанавливаем таймер на 1сек. */
Elif (cmd == DLG_TIMER)
/* Увеличиваем счетчик и обновляем поля на экране */
dlg,Counter = dlg.Counter + 1;
UpdateFields (dlg);
end;
return CM_DEFAULT
end;
RunDialog (tm, @MecProc);
```

MsgBoxEx (val:Variant [, flags:Integer, defInd:Integer, title:String, statLn:String]):Integer

Процедура выводит диалоговое окно с сообщением, задаваемым параметром ***val***. Параметр ***val*** может иметь любой тип, он автоматически преобразуется в строку.

Параметр ***flags*** представляет собой сумму констант, определяющих набор кнопок, которые будут отображаться в диалоговом окне. При этом используются следующие константы:

- MB_OK – кнопка "ОК".
- MB_YES – кнопка "Да".
- MB_NO – кнопка "Нет".

- MB_CANCEL – кнопка "Отмена".
- MB_ERROR – константа, задающая цвет фона, применяемый для сообщений об ошибках.

Если параметр равен 0, то в диалоговом окне не будет ни одной кнопки. Если параметр не задан, то по умолчанию он принимает значение MB_OK, то есть в диалоговом окне будет присутствовать кнопка "OK".

Параметр ***defInd*** определяет кнопку, выбранную по умолчанию. Возможны следующие значения:

- IND_OK.
- IND_YES.
- IND_NO.
- IND_CANCEL.

В параметре ***title*** задается заголовок диалогового окна.

В параметре ***statLn*** указывается текст, отображаемый в статус-строке.

Процедура возвращает код выбранной кнопки. При этом используются те же значения, что и для параметра ***defInd***. В случае ошибки выполнения процедура возвращает код IND_ERROR.

Обработка скроллинга

Макропроцедуры модуля RslScr

Раздел содержит описание переменных и макропроцедур обработки скроллинга, определенных в макромодуле ***RslScr***.

Переменные

В макромодуле определены следующие глобальные переменные:

LastUsed – индекс последней заполненной строки в скроллинге.

Nrec – количество строк в скроллинге.

Nfields – количество колонок в скроллинге.

Scrff – ссылка на скроллируемый файл.

Apos – массив физических адресов записей для каждой строки скроллинга.

DlgFields – массив имен колонок скроллинга.

FileFields – массив номеров полей в файле для каждой колонки скроллинга.

Макропроцедуры

SetScroll (FILE, { DlgName, FileName } …)

Данная процедура устанавливает соответствие имен полей в файле и в диалоговой панели. Эта процедура должна вызываться перед вызовом RunDialog.

Параметры:

FILE – идентификатор файла, по которому будет производиться скроллирование.

DlgName – имя колонки в диалоговой панели.

FileName – имя поля в файле ***FILE***, данные из которого должны выводиться в колонку с именем ***DlgName***. Если имя поля в файле совпадает с именем колонки, то в качестве этого параметра можно указать значение NULL.

Пары ***"DlgName – FileName"*** должны быть указаны для каждой колонки в скроллинге.

FindRow (dlg, id)

Процедура возвращает номер строки скроллинга в диалоге ***dlg*** для поля с номером ***id***. Если поле ***id*** не принадлежит области скроллинга, то процедура возвращает значение -1.

FindCol (dlg, id)

Процедура возвращает номер колонки скроллинга в диалоге ***dlg*** для поля с номером ***id***. Если поле ***id*** не принадлежит области скроллинга, то процедура возвращает значение -1.

FillDown (dlg, pos)

Процедура заполняет область скроллинга в диалоге ***dlg*** данными из файла. Первая строка скроллинга получает данные из записи файла с физическим адресом ***pos***.

FillUp (dlg, pos)

Процедура заполняет область скроллинга в диалоге ***dlg*** данными из файла. Последняя строка скроллинга получает данные из записи файла с физическим адресом ***pos***.

SetDlgFields (dlg, row)

Процедура заполняет строку скроллинга с номером ***row*** данными из файла для диалога ***dlg***.

ScrollMes (dlg, cmd, id, key)

Стандартная процедура для обработки сообщений диалога с областью скроллинга.

Эта процедура должна вызываться для всех необработанных программой пользователя сообщений.

UserFill (dlg)

Эта процедура вызывается каждый раз перед перерисовкой области скроллинга. Оригинальная процедура не выполняет никаких действий. Она может быть заменена при помощи процедуры ***ReplaceMacro*** на процедуру, выполняющую нужные пользователю действия.

Макропроцедуры модуля rslx

Раздел содержит описание макропроцедур обработки скроллинга, определенных в макромодуле ***rslx***.

AddMultiAction (rs:Object, key:Integer):Bool

Процедура позволяет зарегистрировать код клавиши **key** для запуска перебора выбранных записей в скроллинге. Процедуру необходимо использовать при обработке сообщения **DLG_INIT**. Для скроллинга, реализованного с помощью процедуры ***AddScroll***, процедуру ***AddMultiAction*** необходимо использовать при обработке сообщения **DLG_INLOOP**. Для скроллингов, встроенных в диалоговую панель с помощью процедуры ***RunScroll***, также возможно выполнение процедуры ***AddMultiAction*** при обработке сообщения **DLG_INLOOP**.

Примечание.

Если необходимо зарегистрировать несколько клавиш, указанную процедуру следует вызвать требуемое количество раз.

Параметры:

rs – объект класса, указанного в процедурах ***RunScroll***, ***AddScroll*** или ***TRecHandler***.

key – код клавиши, который требуется использовать для запуска перебора записей.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

GetMultiCount (rs:Object):Integer

Процедура возвращает количество выбранных записей в скроллинге; вызывается только при обработке сообщений **DLG_MSEL…**.

Параметры:

rs – объект класса, указанного в процедурах ***RunScroll***, ***AddScroll*** или ***TRecHandler***.

Возвращаемое значение:

Процедура возвращает одно из значений:

- количество выбранных записей – в случае успешного выполнения;
- **-1** – если количество записей не известно (в случае использования инвертированного выбора).

GoToScroll (obj:Object)

Процедура позиционирует окно прокрутки на текущую запись в источнике данных ***obj***. Указанная процедура может использоваться только в процедурах обработки сообщений от окна прокрутки или диалоговой панели.

Параметры:

obj – объект класса, указанного в процедурах ***RunScroll***, ***AddScroll*** или ***TRecHandler***, данные которого будут отображаться в окне прокрутки.

Пример:

```
macro Proc_RunScroll (rs, cmd, id, key, numscrol)
...
 if(cmd == DLG_INIT)
    rs.movenext();
    rs.movenext();
    rs.movenext();
    GoToScroll();
 end;
....
end;
```

RunScroll (data:Object, numCol:Integer, colInfo:TArray, uniqName, proc:Variant, head:String, stLine:String, rdOnly:Bool, x:Integer, y:Integer, cx:Integer, cy:Integer):Bool

Процедура производит запуск окна прокрутки наборов данных RSD; может быть использована для обработки нескольких выделенных строк скроллинга.

Примечание.
Сортировка записей полученного окна прокрутки выполняется при помощи щелчка левой кнопки мыши по заголовку колонки, записи которой необходимо упорядочить. При первом щелчке выполняется сортировка по возрастанию, при повторном – по убыванию.

Для программного закрытия окна прокрутки, запущенного указанной процедурой, и выбора текущей записи необходимо при обработке сообщения **DLG_KEY** процедурой обработки ***evProc*** для требуемой клавиши вернуть значение **CM_SELECT**.

Примечание.
Во время выполнения процедуры RunScroll не осуществляется журнализация возникающих изменений.

Параметры:

data – источник данных для окна прокрутки (объект класса ***RsdRecordset*** или класса, производного от инструментального класса ***ToolsDataAdapter***). Объект класса ***RsdRecordset*** должен реализовывать статический клиентский курсор.

numCol – количество отображаемых колонок в области прокрутки, сведения для которых содержатся в массиве ***colInfo***. Если значение параметра равно **0**, колонки создаются для каждого поля в источнике данных **data**. При этом в качестве заголовка колонки используется название поля.

colInfo – массив, задающий атрибуты колонок области прокрутки. Для каждой колонки атрибуты записываются в 6 последовательных элементах массива:

- **0** – имя поля в рекордсете, обязательный атрибут;
- **1** – заголовок колонки, если заголовок не задан, используется название поля;
- **2** – начальная ширина колонки, если ширина не задана, используется автоматическое вычисление ширины колонки;

Примечание.
Нужно учесть, что в системе RS-Bank V.6 значение этого элемента массива, заданное пользователем в макросе, который реализует пользовательский интерфейс, увеличивается в любых скроллингах.

- **3** – тип поля, возможные значения:
  - **1** – редактируемое (значение по умолчанию);
  - **2** – нередактируемое;
  - **5** – прокручиваемое.
- **4** – количество знаков после точки (для числовых полей);
- **5** – зарезервировано.

Примечание.
Если

uniqName – уникальное наименование окна просмотра. Указанное наименование используется для сохранения и восстановления изменений, которые вносятся во внешний вид колонок: ширины, позиции на экране, видимости. Атрибуты колонок сохраняются в ветке *COMMON\TOOLS\COLUMNWIDTH* пользовательского реестра. В результате при повторном вызове процедуры ***RunScroll*** с заданным параметром **uniqName** колонки скроллинга будут иметь атрибуты, сохраненные в настройке реестра. Атрибуты колонок, передаваемые в процедуру ***RunScroll*** из макроса, при этом не учитываются. Если наименование окна просмотра не задано, все визуальные атрибуты колонок принимают значения, заданные в макросе.

proc – процедура обработки сообщений окна просмотра (имя глобальной процедуры, ссылка на процедуру, ссылка на метод RSL класса). Если обработчик не требуется, используется значение ***Null***. Список обрабатываемых значений описан в разделе "Список обрабатываемых сообщений процедуры RunScroll" (см. стр. 78).

head– строка, задающая заголовок окна просмотра.

stLine – строка, задающая содержимое статусной строки.

rdOnly – признак "только для чтения". Возможные значения:

- TRUE (или не задано) – данные окна просмотра недоступны для корректировки.
- FALSE – данные окна просмотра доступны для корректировки.

x, y – координаты левого верхнего угла окна просмотра:

- **x** – расположение окна по горизонтали, если параметр равен **-1**, выполняется центрирование окна по горизонтали;
- **y** – расположение окна по вертикали, если параметр равен **-1**, выполняется центрирование окна по вертикали.

cx – ширина окна просмотра.

cy – высота окна просмотра.

Примечание.
Если параметры не заданы, окно просмотра принимает максимально возможный размер в пределах клиентской области окна.

Возвращаемое значение

Процедура возвращает одно из значений:

- TRUE – в случае, когда процедура обработки сообщений в ответ на сообщение **DLG_KEY** возвращает значение **CM_SELECT**. При этом выбранная в окне прокрутки запись становится текущей и в источнике.
- FALSE – в остальных случаях (например, если обработчик сообщения вернул значение **CM_CANCEL**; при этом выполняется закрытие окна прокрутки).

Примеры:

```
import RSD,rcw;
// Создаём клиентский статический курсор
cn = RsdConnection("Northwind");
cmd = RsdCommand (cn,"select * from Customers");
rs = RSDRecordset(cmd , RSDVAL_CLIENT, RSDVAL_STATIC );
// Пример 1.  Просматриваем все поля набора данных данные
RunScroll (rs);
// Пример 2. Просматриваем только заданные поля.
// Вспомогательная процедура подготовки информации с
атрибутами полей
macro AddCol (ar,ind, fld, head, width, rdonly)
  ar.value (ind * 6)     = fld;
  ar.value (ind * 6 + 1) = head;
  ar.value (ind * 6 + 2) = width;
  ar.value (ind * 6 + 3 ) = 0;   // fldType
  ar.value (ind * 6 + 4 ) = -1;  // decPoint
  ar.value (ind * 6 + 5 ) = 0;   // reserv
end;

col = TArray;
AddCol (col, 0, "CompanyName", "Компания", null, true);
AddCol (col, 1, "ContactName", "Контактное лицо",    null, true);
AddCol (col, 2, "Address", null,    null, true);
RunScroll (rs, 3, col);

// Пример 3. Обрабатываем событие DLG_KEY.
macro EvProc (rs, cmd, id, key )
   if ((cmd == DLG_KEY) and (key == 13))
       return CM_SELECT;
   end;
end;
if (RunScroll (rs,null,null,null,@EvProc))
     println (“Выбрана запись: “, rs.Value (0));
end;

// Пример 4. Использование вспомогательного класса для
обработки в ОО стиле.
class RslScroll
   private var colInfo;
```

```
  private macro numCol
    if (colInfo)
      return colInfo.size / 6
    end
  end;
  macro AddCol (ind, fld, head, width, fldType,decPoint)
    if (not colInfo)
      colInfo = TArray;
    end;
    colInfo.value (ind * 6)     = fld;
    colInfo.value (ind * 6 + 1) = head;
    colInfo.value (ind * 6 + 2) = width;
    colInfo.value (ind * 6 + 3 ) = fldType;
    colInfo.value (ind * 6 + 4 ) = decPoint;
    colInfo.value (ind * 6 + 5 ) = 0;   // reserv
  end;
  macro EvProc (rs, cmd, id, key)
  end;
  macro Run (rs,name,Head, StLn, rdOnly)
    return RunScroll (rs, numCol, colInfo,name, R2M(this,"EvProc"),
Head, StLn, rdOnly);
  end;
end;
// Производный класс задаёт колонки и переопределяет процедуру
обработки сообщений
class (RslScroll) MyScroll
  InitRslScroll;
  macro EvProc (rs, cmd, id, key)
     println ("--- ","Cmd = ", CmdName (cmd), ", FldId = ", id, ", Key =  ",
key);
  end;
  AddCol ( 0, "CompanyName", "Компания", null, true);
  AddCol ( 1, "ContactName", "Контактное лицо",    null, true);
  AddCol ( 2, "Address", null,    null, true);
end;
MyScroll.Run (rs);
OnError(e)
 println (e.message);
 if (IsEqClass ("TRsdError", e.err))
    i=0;
    while( i< e.err.environment.errorcount )
      println(e.err.environment.error(i).descr);
      printprops (e.err.environment.error(i));
      i=i+1;
    end;
    e.err.environment.ClearErrors;
 end;
```

UpdateFields (obj:Object)

Процедура выполняет обновление полей текущей записи на экране. В качестве параметра ***obj*** используется объект-источник данных. При этом, в отличие от процедуры ***UpdateScroll***, данные из хранилища не запрашиваются. Выводятся текущие значения текущей записи, которые могли быть изменены в коде.

UpdateScroll (obj:Object, mode:Integer)

Процедура выполняет обновление окна прокрутки набора данных, заданного параметром ***obj***, в соответствии с режимом обновления ***mode***. Процедура может быть вызвана только в контексте процедуры обработки сообщений от окна прокрутки или диалоговой панели.

Параметры:

obj – объект класса, указанного в процедурах ***RunScroll***, ***AddScroll*** или ***TRecHandler***, данные которого будут отображаться в окне прокрутки.

mode – режим обновления. Параметр принимает одно из значений:

- **2** – обновление только текущей записи набора (значение по умолчанию);
- **3** – обновление всей области прокрутки;
- **4** – обновление страницы данных и окна прокрутки, начиная с текущей записи;
- **5** – обновление страницы данных и области прокрутки.

Контроль ввода записи в скроллинг

Для контроля вводимой в скроллинг записи используются следующие процедуры:

GetScrollFieldValue(numfld:Integer, value:Undef):Bool

Процедура позволяет получить значение поля текущей строки скроллинга.

Параметры:

numfld – идентификационный номер поля.

value – возвращаемый параметр, в который передается значение поля, при этом тип значения соответствует типу поля текущей строки скроллинга.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – при успешном выполнении процедуры.
- ***FALSE*** – в случае возникновения ошибки.

SetScrollFieldValue(numfld:Integer, newvalue:Undef, [strlen:Integer]):Bool

Процедура задает новое значение поля текущей строки скроллинга.

Параметры:

numfld – идентификационный номер поля.

newvalue – новое значение поля. Тип значения должен соответствовать типу поля текущей строки скроллинга.

strlen – длина строки. Параметр используется только для полей строкового формата. Значение параметра не должно превышать максимального размера строкового поля источника данных, по которому построен скроллинг.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – при успешном выполнении процедуры.
- ***FALSE*** – в случае возникновения ошибки.

Пример:

```
macro EvProc_RunScroll(rs, cmd, id, key, numscrol)
var getval;
  var CM_FLAF = CM_DEFAULT; /*возвращаемое значение по
умолчанию*/
   if (key == EXITFIELD)
     if (GetScrollFieldValue(0, getval) AND (getval == 12345))
      SetScrollFieldValue(1, "Новое значение", 20);
      SetScrollFieldValue(2, $12345.12);
     end;
     if (GetScrollFieldValue(0, getval) AND (getval == 0))
       msgbox("Значение должно быть ненулевым!");
      CM_FLAG = CM_CANCEL;
    end;
end;
  return CM_FLAG;
end;
```

IsScrollEditMode():Bool

Процедура определяет, установлен ли режим ввода/ редактирования записи непосредственно в скроллинге, реализованном с помощью процедур ***RunScroll*** или ***AddScroll***.

Возвращаемое значение:

Процедура возвращает одно из значений:

- ***TRUE*** – в случае, если установлен режим ввода/ редактирования записи непосредственно в скроллинге.
- ***FALSE*** – режим просмотра скроллинга.

Пример проверки дублирования записей при вводе/ редактировании записи в скроллинг:

```
var InsertMode = false;
macro EvProc_RunScroll(rs, cmd, id, key, numscrol)
var getval;
 var CM_FLAF = CM_DEFAULT; /*возвращаемое значение по
умолчанию*/
…
 //вводим признак режима ввода
 if ((cmd == DLG_KEY) AND (key == K_F9))
  InsertMode = true;
 end;
 if ((cmd == DLG_OUTREC) AND IsScrollEditMode()) //событие –
выход из записи в режиме ввода/ редактирования
  if (GetScrollFieldValue(0, getval))
   if (FindRecord(getval, InsertMode)) //FindRecord –
пользовательская функция. реализующая поиск записи в
источнике данных, по которому построен скроллинг, с учетом
InsertMode(режима ввода)
     CM_FLAG = CM_CANCEL;
   end;
  end;
 //сбрасываем признак режима ввода
 if (CM_FLAG != CM_CANCEL)
  InsertMode = false;
 end;
end;
 return CM_FLAG;
end;
```

Работа с отладчиком RSL

В разделе содержится описание процедур, используемых для работы с отладчиком RSL.

В отладчике предусмотрена возможность выполнения интерактивных выражений (RSL кода). Вызов диалогового окна, предназначенного для ввода требуемого RSL кода, производится из отладчика с помощью клавиши [Ctrl+I]. Список введенных выражений сохраняется в файле для повторного использования: *<Application Data>\R-Style Softlab\<Product Name>\RSL Debugger\InterExp.dbg*.

Trace (...)

Процедура печатает свои параметры в окно трассировки отладчика RSL.

DebugBreak

Если в системе установлен отладчик RSL, то эта процедура вызывает прерывание исполнения RSL программы и активизацию отладчика на следующей после ***DebugBreak*** инструкции.

Файлы и структуры

Как уже говорилось ранее, язык RSL является составной частью информационной системы, разработанной программистами R-Style Softlab. При его работе используется внутренний словарь системы, который поставляется в дистрибутиве. Доступ к таблицам базы данных и структурам, описанным в этом словаре, осуществляется при помощи набора специальных процедур.

Процедуры этой группы работают со следующими видами файлов:

- с таблицами в базе данных;
- с текстовыми файлами;
- с файлами в формате DBF.

В списке формальных параметров этих процедур в качестве первого параметра обязательно указывается ссылка на таблицу или файл или структуру ***id*** (тип V_FILE или V_RECORD, соответственно). Любая из этих процедур возвращает значение типа Bool, если в ее описании не указано другое значение. При успешном выполнении возвращается TRUE, в противном случае FALSE.

В описании действий, выполняемых процедурами, встречаются термины "Текущая запись" и "Физическая позиция", которые необходимо пояснить.

Текущей считается запись, которая была прочитана (извлечена) и в этот момент находится в буфере данных. Она может отсутствовать:

- при первом обращении к таблице или файлу;
- после переустановки таблицы или файла процедурой ***rewind***;
- в результате сбоя или другой внештатной ситуации.

Физическая позиция – это адрес записи, определяющий ее местоположение в таблице.

ConvertDDF (defName:String, outDir:String):Bool

Процедура позволяет выполнить конвертацию заданного файла словаря def в стандартный ddf формат.

Параметры:

defName – имя словаря def.

outDir – имя каталога, в котором необходимо создать файлы словаря dff (*file.ddf, field.ddf, index.ddf*).

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

CopyTblDef (inDef:String, outDef:String, inStruct:String, outStruct:String):Bool

Процедура используется для заимствования описания структуры таблицы из одного словаря в другой.

Параметры:

inDef – имя словаря, из которого производится заимствование.

outDef – имя словаря, в котором необходимо сохранить заимствованную структуру.

Примечание.
*Если заимствование производится в пределах одного словаря, значения параметров **inDef** и **outDef** совпадают.*

instruct – имя структуры, заимствование которой требуется выполнить.

outStruct – имя структуры, полученной в результате заимствования.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

ExistFile (string [, integer])

Процедура возвращает TRUE, если файл (таблица), имя которого задано параметром ***string***, существует. Параметр ***integer*** определяет, где искать файл:

- **0 –** в каталоге текстовых файлов;
- **1 –** в базе данных;
- **2 –** в каталоге DBF-файлов.

Если параметр ***integer*** не задан, поиск осуществляется в текущем каталоге.

Open (fid, name:String, encode:String, fatal:Bool, eolType:Integer) : Bool

Открытие таблицы базы данных (файла). Следует отметить, что таблица открывается автоматически при первом обращении к ней, поэтому нет необходимости явно вызывать процедуру ***open***. После открытия таблицы для неё автоматически выполняется процедура ***rewind***.

Параметры:

fid – идентификатор таблицы.

name – имя таблицы, отличное от имени во внутреннем словаре. Это позволяет работать с несколькими таблицами, имеющими одинаковую структуру. Если имя таблицы соответствует имени в словаре базы данных, то в качестве параметра можно указать значение NULL.

encode – кодировка символов в таблице (файле). Параметр принимает следующие значения:

- "rsoem" – RSOEM кодировка;

- "rsansi" – RSANSI кодировка;
- "lcoem" – LCOEM кодировка;
- "lcansi" – LCANSI кодировка;
- "utf8" – UTF-8 кодировка;
- "utf16le" – UTF-16LE кодировка;
- "utf16be" – UTF-16BE кодировка.

fatal – признак, определяющий поведение процедуры в случае возникновения ошибки. Возможные значения:

- **TRUE** – процедура аварийно завершает работу RSL-программы, если таблицу невозможно открыть.
- **FALSE** – если таблицу невозможно открыть, процедура возвращает значение FALSE (значение по умолчанию).

eolType – стиль завершения строки при записи в текстовый документ. Параметр принимает следующие значения:

- **0** – стандарт DOS и Windows (CR LF) – значение по умолчанию;
- **1** – стандарт UNIX (LF);
- **2** – стандарт MAC (CR).

Пример:

```
File aa ("balance.dbt");
File bb ("balance.dbt");
File cc () txt;

open (aa,"bal.dbt");
open (bb,"balarch.dbt");
name = datadir + "data.txt";
open (cc,name);
```

Если файл уже открыт, то последующий вызов ***open*** будет проигнорирован до вызова ***close***.

Пример:

```
open (aa,"data.dbf");          /* Этот вызов игнорируется */
(aa,"olddata.dbf");

close (aa);                    /* Этот вызов сработает */
open (aa,"olddata.dbf");
```

Close (id)

Процедура закрывает таблицу (файл). Нужно отметить, что таблица (файл) закрывается автоматически при выходе из контекста программы, поэтому нет необходимости явно вызывать процедуру ***close***. Эта процедура используется при наличии в программе большого количества глобальных таблиц (файлов), так как они закрываются только после выполнения всей RSL-программы.

Примечание.

Внутренние словари базы данных, открытые в процессе работы программы, также закрываются только после выполнения всей RSL-программы.

ClearStructs

Процедура освобождает из памяти структуры всех таблиц базы данных, открытых во время работы RSL-программы.

Create (id [, filename] [, isTable:Bool] [, isP:Bool])

Процедура создает новую таблицу в базе данных или DBF-файл по описанию, которое во внутреннем словаре базы данных имеет идентификатор ***id***.

Если имя новой таблицы (файла) отличается от имени, заданного во внутреннем словаре в качестве указанного идентификатора, его необходимо определить во втором параметре ***filename***.

Процедура не работает с текстовыми файлами. Если в программе определен текстовый файл с доступом на запись, то текстовый файл создается автоматически (если файл уже существовал, то он удаляется и создается новый).

Параметр ***isTable*** используется только при работе с таблицами базы данных. Пользователь может установить два значения данного параметра:

- **TRUE** – при этом процедура ***Create*** аварийно завершает работу RSL-программы, если таблицу невозможно создать.
- **FALSE** – если таблицу невозможно создать, процедура возвращает значение FALSE. То же самое происходит и в том случае, если пользователь не установил никакого значения данного параметра.

Параметр ***isP*** используется только при создании таблиц баз данных. Если этот параметр равен TRUE, то процедура создает постоянную таблицу. Такая таблица должна открываться по тому описанию, по которому она была создана.

Пример:

```
file  f("person.dbt", "bank.def");
if(not Create( f, "aa", True ))
  println("Error create");
end;
```

Clone (id [, filename] [, isTable:Bool] [, isP:Bool])

Процедура позволяет создавать новые или удалять информацию в уже существующих таблицах в базе данных и DBF-файлах.

Если в системе задан список каталогов для поиска DBF-файлов, новый файл создается в первом каталоге из списка, если не задан – в текущем.

Параметры ***filename***, ***bisTable*** и ***isP*** аналогичны соответствующим параметрам процедуры ***Create***.

Процедура не работает с текстовыми файлами.

Пример:

```
File aa (“data.dbf”) dbf write;     /* Открывает файл data.dbf с
                                    идентификатором aa */
clone (aa);                         /* Удаляет информацию в файле data.dbf */
clone (aa,“newdata.dbf”);           /* Создает и открывает новый файл
                                    newdata.dbf со структурой как у data.dbf */
```

Если перед вызовом процедуры ***clone*** файл с заданным идентификатором ***id*** уже был открыт, то после клонирования он автоматически закроется.

ViewFile (id, name [, edit])

Процедура позволяет просматривать текстовые файлы и таблицы в базе данных.

В качестве параметра ***id*** передается идентификатор файла, который нужно просмотреть**.**

Параметр ***name*** позволяет задать строку с именем текстового файла, необходимого для просмотра.

Если в качестве параметра ***edit*** задано значение ***TRUE***, то просматриваемая таблица открывается в режиме корректировки. Параметр ***edit*** для текстового файла игнорируется.

DelFile (string)

Процедура удаляет таблицу баз данных (файл) с именем ***string*** вместе с ее описанием во внутреннем словаре системы.

DropTable (string)

Процедура удаляет таблицу баз данных с именем ***string***, не удаляя ее описания из внутреннего словаря системы.

Next (id [, integer])

Процедура считывает из таблицы или файла следующую за текущей запись или первую запись, если вызывается после процедуры ***rewind***. Если указан параметр ***integer***, процедура считает заданное в нем количество байт, символы конца строки в этом случае обрабатываются как и любые другие символы.

При работе с таблицами базы данных, если текущая запись оказывается последней, процедура возвращает значение FALSE и выполняет ***rewind***. Во всех остальных ошибочных случаях процедура аварийно завершает работу RSL-программы.

Для текстовых файлов процедура читает следующую строку файла.

Prev (id)

Процедура считывает из таблицы или файла предыдущую запись или последнюю, если вызывается после процедуры ***rewind***.

При работе с таблицами базы данных, если текущая запись оказывается последней, процедура возвращает значение FALSE и выполняет ***rewind***. Во всех остальных ошибочных случаях процедура аварийно завершает работу RSL-программы.

Процедура не работает с текстовыми файлами.

ReWind (id)

Процедура переустанавливает таблицу или файл таким образом, что текущая позиция не изменяется, но изменяется поведение вызванных после ***rewind*** процедур ***next*** или ***prev***: ***next*** извлечет первую запись, а ***prev*** – последнюю.

Insert (id [, string | size] [, integer] [, bool])

Процедура помещает в таблицу (файл) новую запись, используя значения полей из буфера данных. Если указан параметр ***integer***, процедура запишет заданное в нем количество байт, символы конца строки в этом случае автоматически не добавляются.

Добавление происходит в соответствии с заданными ключевыми последовательностями.

Второй параметр принимает одно из двух значений:

- для текстовых файлов – строка для записи в файл;
- для таблицы базы данных – размер в байтах переменной части записи, добавляемой в таблицу. Если этот параметр отсутствует, используется размер, который был определен при выполнении операции чтения. Его значение можно узнать при помощи процедуры ***GetVarSize***.

Параметр ***bool*** используется только при работе с таблицами базе данных и имеет тип Bool. Пользователь может установить два значения данного параметра:

- **TRUE** – в этом случае при дублировании ключа процедура ***Insert*** аварийно завершает работу RSL-программы.
- **FALSE** – при этом дублирование ключа не будет считаться ошибкой. Процедура в этом случае возвращает значение FALSE. То же самое происходит и тогда, когда пользователь не установил никакого значения этого параметра.

В текстовые и DBF-файлы записи добавляются только в конец.

Возвращаемое значение:

Процедура по умолчанию возвращает значение FALSE в том случае, если таблица (файл) или запись захвачены или дублируется ключ.

Update (id [, size] [, bool])

Процедура обновляет текущую запись в файле ***id***, используя значения полей из буфера данных.

Параметр ***size*** задается только для таблиц базы данных. Он имеет тип Integer и содержит размер в байтах переменной части записи, изменяемой в таблице. Если этот параметр отсутствует, используется размер, который был определен при выполнении операции чтения. Его значение можно узнать при помощи процедуры ***GetVarSize***.

Параметр ***bool*** аналогичен соответствующему параметру процедуры ***Insert***.

Процедура не работает с текстовыми файлами.

Возвращаемое значение:

Процедура по умолчанию возвращает значение FALSE в том случае, если таблица (файл) или запись захвачены или дублируется ключ.

Delete (id, bool)

Процедура удаляет из таблицы (файла) ***id*** текущую запись.

Параметр ***bool*** аналогичен соответствующему параметру процедуры ***Insert***.

Процедура не работает с текстовыми файлами.

Возвращаемое значение:

Процедура по умолчанию возвращает значение FALSE в том случае, если таблица (файл) или запись захвачены или дублируется ключ.

GetPos (id)

Процедура возвращает физическую позицию текущей записи таблицы (файла). Для DBF-файлов это порядковый номер записи.

Процедура не работает с текстовыми файлами.

GetDirect (id [, recpos])

Процедура извлекает из таблицы (файла) запись, находящуюся в физической позиции, специфицированной вторым необязательным формальным параметром ***recpos*** (для DBF-файлов это порядковый номер записи). Значение ***recpos*** должно быть предварительно получено при помощи процедуры ***getpos***, описанной выше:

```
recpos = getpos (id);
```

Если параметр ***recpos*** не указан, то извлекается запись, для которой последний раз выполнялась процедура ***getpos***.

Пример:

```
File ff (account);          /* Запоминает позицию */
getpos (ff)
getdirect (ff)              /* Извлекает запись   */
```

В случае возникновения ошибки эта процедура возвращает значение FALSE.

Процедура не работает с текстовыми файлами.

GetEQ (id), GetGT (id), GetGE (id), GetLT (id), GetLE (id)

Процедуры этой группы работают только с таблицами базы данных.

Процедуры осуществляют поиск записи таблицы, значение ключа для которой:

- равно указанному;
- больше указанного;
- больше или равно указанному;
- меньше указанного;
- меньше или равно указанному,

соответственно перечисленным выше процедурам.

Данные процедуры возвращают значение FALSE, если искомая запись отсутствует в таблице. Во всех остальных ошибочных ситуациях происходит аварийное завершение RSL-программы.

Для того чтобы задать значение ключа для поиска, необходимо присвоить нужные значения тем полям записи таблицы, которые соответствуют сегментам текущего ключа.

Пример:

```
FILE Счета ("account.dbt") key 0;   /* Описана таблица Счета с нулевым
                                       ключом, который, например, имеет два
                                       сегмента: дату и номер счета */
Счета.Date = date;                  /* Присвоены значения текущей даты полю
Счета.Account = "   010234";        Date и нужного номера счета полю Account
                                    */
getEQ (Счета);                      /* Получена запись с нулевым ключом и
                                       заданными значениями сегментов */
```

SetRecordAddr (recId, fileId [, ind] [, offset] [, fix])

Процедура позволяет наложить структуру, заданную идентификатором ***recId***, на таблицу базы данных с идентификатором ***fileId***. Это обеспечивает доступ к переменной части записи (так как структура переменной части не описывается при описании структуры таблицы), а так же использовать для доступа к постоянной части записи структуру, отличную от той, которая описывает структуру записи таблицы. Последняя возможность позволяет хранить в таблице записи разной структуры (union).

Параметры ***ind*** и ***offset*** типа Integer, а также параметр ***fix*** типа Bool позволяют определить смещение структуры ***recId*** в файле ***fileId***.

По умолчанию значения ***ind*** и ***offset*** считаются равными нулю. Если эти параметры заданы, то смещение вычисляется следующим образом:

```
GetRecordSize (recId) * ind + offset
```

Если значение ***fix*** равно FALSE или не задано, то структура ***recId*** накладывается с вычисленным смещением относительно начала переменной части записи. Если параметр ***fix*** равен TRUE, то структура накладывается с вычисленным смещением относительно начала постоянной части записи.

Пример:

```
file fl ("varlen.dbt") write
/* Таблица с записями переменной длины */
record rec ("vrec.rec");
/* Структура для доступа к переменной части */
macro ShowVarLen (n)
/* Параметр n задает количество структур rec в переменной
части */
  i = 0;
  while ( i < n )
    SetRecordAddr (rec, fl, i);
    println
      (rec.id, " ", rec.name);
    i = i + 1;
```

```
  end;
end;
while ( next (fl) )
  ShowVarLen (file.nRec);
end;
```

Если в переменную часть записи добавляются новые элементы, то для операций ***insert*** и ***update*** необходимо указать новый размер переменной части. В следующем примере считается, что в переменной части находится 5 элементов (структур ***vrec***):

```
file fl ("varlen.dbt") write
/* Файл с записями переменной длины */
record rec ("vrec.rec");
/* Структура для доступа к переменной части */
update (fl,  GetRecordSize (rec) * 5)
```

Наложить структуру можно и на постоянную часть записи:

```
file       acc ("account.dbt") key 1000;
record   dkr ("dkr.str");
println (FldOffset (acc,"D0"));
if (next (acc))
   SetRecordAddr (dkr,acc,0,FldOffset (acc,"D0"),TRUE);
  [ Debet        ##############
    Credit       ##############
    Rest         ############## ]    (dkr.D,dkr.K,dkr.R);
end;
```

PackVarBuff (file [, size])

Процедура "упаковывает" переменную часть записи таблицы базы данных, заданной идентификатором ***file***. Размер переменной части определяется параметром ***size***. Если параметр ***size*** не задан, то используется текущий размер переменной части.

Процедура возвращает новый размер переменной части записи и устанавливает этот размер текущим размером переменной части.

UnPackVarBuff (file)

Процедура "распаковывает" переменную часть записи таблицы базы данных, заданной идентификатором ***file***, и возвращает ее новый размер. Данный размер становится текущим размером переменной части.

GetRecordSize (id)

Процедура возвращает размер в байтах структуры или таблицы базы данных (файла) с идентификатором, заданным параметром ***id***. Это значение вычисляется на основании информации из словаря базы данных.

Пример:

```
file fl ("account.dbt");
record rec ("document.rec");
fsize = GetRecordSize (fl);
rsize = GetRecordSize (rec);
```

Процедура работает только с объектами типа FILE и RECORD.

GetVarSize (id)

Эта процедура возвращает размер в байтах переменной части текущей записи для таблицы базы данных с идентификатором, заданным в параметре ***id***.

FileName (id)

Процедура возвращает имя таблицы базы данных (файла) с идентификатором, заданным в параметре ***id***.

Если в качестве параметра ***id*** передать объект типа ***TRecHandler***, то процедура вернет имя соответствующей структуры во внутреннем словаре системы.

NRecords (id [, par:Bool])

Процедура возвращает количество записей в таблице базы данных (файле) с идентификатором, заданным в параметре ***id***. Если необязательный параметр ***par*** равен FALSE, процедура всегда возвращает -1.

Процедура не работает с текстовыми файлами.

FldNumber (id)

Процедура возвращает количество полей в таблице базы данных (файле) с идентификатором, заданным в параметре ***id***.

Процедура не работает с текстовыми файлами.

FldName (id, number)

Процедура возвращает имя поля с индексом ***number*** для таблицы базы данных (файла) с идентификатором, заданным в первом параметре ***id***. Второй параметр ***number*** представляет собой переменную или константу типа Integer.

Если поле не найдено, возвращается пустая строка.

Процедура не работает с текстовыми файлами.

FldIndex (id, string)

Процедура возвращает индекс поля с именем ***string*** для таблицы базы данных (файла) с идентификатором, заданным в параметре ***id***. Второй параметр ***string*** представляет собой переменную или константу типа String.

Если поле не найдено, процедура возвращает значение, равное минус единице.

Процедура не работает с текстовыми файлами.

FldOffset (id, string | number)

Процедура возвращает смещение в байтах поля с именем ***string*** или номером ***number*** для таблицы базы данных с идентификатором, заданным в параметре ***id***.

Если поле не найдено, процедура возвращает значение, равное 0.

Пример:

```
File ff ("account.dbt");
I = 0;
while (I < FldNumber (ff))
  println (I," – ",FldOffset (ff,I)
end
```

ClearRecord (id)

Процедура обнуляет буфер записи таблицы базы данных (файла), заданной идентификатором ***id***, или объект типа RECORD.

Процедура не работает с текстовыми файлами.

SetBuff (id, addr)

Процедура явно устанавливает адрес памяти, заданный во втором параметре, для объекта типа RECORD с идентификатором ***id***. Второй параметр, представляющий собой переменную с типом, соответствующим константе V_MEMADDR, должен содержать адрес памяти, переданный из программы на языке C.

Copy (id1, id2 [, flag:Integer])

Процедура копирует содержимое буфера ***id2*** в буфер ***id1***. Параметры ***id1*** и ***id2*** представляют собой переменные типа ссылка на таблицу базы данных (V_FILE) или ссылка на структуру (V_RECORD).

В качестве параметров процедуры копирования могут также использоваться объекты стандартных классов ***Tbfile*** и ***TRecHandler***.

Если оба буфера имеют одинаковую структуру, то выполняется побайтное копирование.

Если структура буферов различна, то для каждого поля из буфера ***id2*** будет производиться поиск поля с таким же именем в буфере ***id1***. Если такое поле найдено, и его тип совпадает с типом поля в буфере ***id2***, выполняется побайтное копирование.

Для полей с разным типом предварительно, если возможно, выполняется преобразование. Поля в ***id1***, для которых не обнаружено соответствующих полей в ***id2***, обнуляются.

Процедура позволяет копировать поля типа BLOB.

Параметр ***flag*** задает режим копирования, если буфер представляет собой запись переменной длины, и может принимать следующие значения:

- 0 – процедура скопирует только постоянную часть записи (используется по умолчанию);
- 1 – процедура скопирует постоянную и переменную часть записи;
- 2 – процедура скопирует только переменную часть записи.

Данная процедура позволяет также выполнять копирование буферов DBF-файлов.

CopyBlob ()

Данная процедура является аналогом процедуры ***Copy*** с параметром ***flag***, равным 2.

Если в файле-приёмнике имеется поле типа BLOB, то при копировании записи необходимо сначала скопировать постоянную часть записи без BLOB и вставить новую запись. После этого необходимо скопировать BLOB. Если файл-приёмник имеет просто переменную часть, то необходимо скопировать постоянную и переменную часть записи и после этого вставить запись в файл.

Пример:

```
// Файл-приёмник содержит поле BLOB
f1 = TBfile ("blobdata.dbt");
f1.next;
f2 = TBFile ("blobcopy.dbt","w");
copy (f2,f1);
f2.insert;
CopyBlob (f2,f1);
// Файл-приёмник содержит простую переменную часть
f1 = TBfile ("vardata.dbt","w");
f1.next;
f2 = TBFile ("varcopy.dbt","w");
copy (f2,f1,1);
f2.insert;
```

KeyNum (id [, newkey])

Процедура возвращает текущий номер ключа для таблицы базы данных с идентификатором ***id***.

Если указан второй, необязательный параметр ***newkey***, устанавливается новый номер ключа, равный значению ***newkey***.

SetDelim (symbol, space)

Процедура устанавливает символы (максимальное количество 9) ***symbol***, которые являются разделителями элементов в строках текстового файла. По умолчанию в качестве символов-разделителей используются символ табуляции "\t" и пробел. Если задано значение параметра ***space***, все пробельные разделители обрабатываются аналогично другим разделителям.

Процедура работает только с текстовыми файлами.

Пример:

```
file f ("test.txt")  txt;
SetDelim ("\t", true);
next (f);
println ( f (0) );
println ( f (1) );
println ( f (2) );
println ( f (3) );
println ( f (4) );
```

Status [(parm)]

Процедура позволяет получить код ошибки в случае неудачного завершения процедур, работающих с таблицами в базе данных.

Если процедуре передан необязательный параметр ***parm***, процедура присвоит ему строку, содержащую сообщение об ошибке.

Пример:

```
if (not next (id)
  println (“Cmamyc:”,status);
end;
```

Процедура работает только с таблицами в базе данных.

ProcessTrn (TrnType [, MacroName, file1, file2,...])

Процедура служит для выполнения транзакции при работе с языком RSL.

Параметр ***TrnType*** определяет тип транзакции, то есть указывает, что должно быть выполнено в рамках этой транзакции в вызывающем модуле. Значения, которые может принимать этот параметр, должны быть определены в вызывающем модуле. В рамках прикладных систем, разработанных в компании R-Style Softlab, тип транзакции может быть следующим:

- **1** – транзакция рублевой проводки;
- **2** – транзакция валютной проводки;
- **3** – транзакция проводки валюты без рублей;
- **4** – транзакция проводки вкладчиков;
- **5** – ввод документов, счетов, клиентов, банков;
- **6** – транзакция рублевой проводки "как есть" (то есть без дополнительного заполнения полей рублевого документа);
- **7** – транзакция валютной проводки "как есть";
- **8** – транзакция проводки валюты без рублей "как есть";
- **10** – транзакция ввода параметров сделок.

Если в вызывающем модуле ничего выполнять не надо, то необходимо указать тип транзакции 0 или передать вместо типа транзакции значение NULL.

Если в рамках транзакции необходимо выполнить действия только в вызывающем модуле, то, кроме типа транзакции, больше никаких параметров для ***ProcessTrn*** указывать не нужно.

Параметр ***MacroName*** задает процедуру, которая вызывается для выполнения транзакции на языке RSL. В качестве параметра может быть передано одно из следующих значений:

- текстовая строка с именем глобальной макропроцедуры;
- текстовая строка с именем локальной макропроцедуры;
- ссылка на локальную макропроцедуру;
- ссылка на метод класса, полученную с помощью процедуры ***R2M***.

Макропроцедуры, вызываемые из процедуры ***MacroName***, выполняются в рамках той же транзакции. После имени макропроцедуры указывается список идентификаторов таблиц базы данных, участвующих в транзакции на языке RSL. В рамках транзакции можно обращаться только к тем таблицам, идентификаторы которых указаны в этом списке.

Процедура ***ProcessTrn*** возвращает одно из двух значений:

- **TRUE** – если транзакция завершилась успешно, то есть макропроцедура ***MacroName*** выполнилась до конца, или была вызвана процедура ***Exit***.

- **FALSE** – если транзакция завершилась вызовом процедуры ***AbortTrn***, или пользователь ответил отрицательно на запрос повторения операции над заблокированной таблицей.

Вложенные транзакции не поддерживаются.

LoopInTrn (val)

Процедура изменяет действие процедур выполнения транзакции ***ProcessTrn*** и ***ProcessConTrn*** в случае захвата таблиц базы данных или записей.

Если параметр ***val*** равен TRUE, то процедура выполнения транзакции прерывает транзакцию и после паузы повторно ее начинает, а также повторно вызывает пользовательскую процедуру обработки транзакции.

Количество повторов задается параметром REPCOUNT. Время задержки задается параметрами SLEEPTIME и RANDOMTIME, причем время задержки определяется как сумма SLEEPTIME и случайного числа в диапазоне между 0 и RANDOMTIME. Эти параметры устанавливаются в файле *rsreq.ini*.

После выполнения заданного числа повторов процедура выполнения транзакции выводит диалоговое окно с запросом на продолжение повтора выполнения транзакции.

AbortTrn ()

Процедура завершает выполнение макросов, вызванных для выполнения транзакции, и производит откат всех изменений, внесенных в файлы в рамках транзакции как на RSL, так и в вызывающем модуле на C или C++. Процедура ***ProcessTrn*** в этом случае возвратит FALSE.

Если эта процедура вызвана вне транзакции, то она не выполняет никаких действий.

Процедура используется только при работе с таблицами базы данных.

SelectFile (file [, mask, head], sort, term)

Процедура выводит на экран стандартное диалоговое окно выбора файла (см. Руководство "Общие документы").

Процедура возвращает TRUE, если файл был выбран. При этом параметр ***file*** принимает значение имени выбранного файла.

Параметр ***mask*** содержит строку, определяющую шаблон имени файла, который необходимо выбрать. По умолчанию данный параметр содержит название текущего каталога.

Параметр ***head*** содержит строку, которая будет выведена на экран в качестве заголовка окна выбора файла. По умолчанию в качестве этого параметра передается строка: "Выбор файла".

Параметр ***sort*** определяет порядок сортировки файлов по умолчанию. Возможные значения:

- **0** – по имени;
- **1** – по расширению;
- **2** – по времени;
- **3** – по размеру.

Параметр ***term*** используется при работе в трехзвенной архитектуре и позволяет указать, откуда следует осуществлять выбор файла. Возможные значения:

- TRUE – выбор файла выполняется на терминале.
- FALSE – выбор файла производится на сервере.

Пример:

```
SelectFile (@fl, "*.mac", "Заголовок", 0, true);
```

SelectFolder (folder [, mask, head,] term)

Данная процедура выводит на экран стандартное диалоговое окно выбора подкаталога.

Процедура возвращает TRUE, если подкаталог был выбран. При этом параметр ***folde***r принимает значение имени выбранного подкаталога.

Параметр ***mask*** содержит строку, определяющую шаблон имени подкаталога, который необходимо выбрать. По умолчанию этот параметр содержит название текущего каталога.

Параметр ***head*** содержит строку, которая будет выведена на экран в качестве заголовка окна выбора подкаталога. По умолчанию в качестве этого параметра передается строка: "Выбор каталога".

Параметр ***term*** используется при работе в трехзвенной архитектуре и позволяет указать, откуда следует осуществлять выбор подкаталога. Возможные значения:

- TRUE – выбор подкаталога выполняется на терминале.
- FALSE – выбор подкаталога производится на сервере.

Пример:

```
SelectFolder (@fl, "*.mac", "Заголовок", true);
```

NeedFreeDB ()

Процедура может быть вызвана в любом месте RSL-программы для указания интерпретатору на необходимость закрытия словарей при завершении выполнения RSL-программы.

WriteBlob (file, value)

Процедура записывает значение переменной ***value*** в поле типа BLOB таблицы ***file***.

Перед записью при помощи процедур ***next, prev, get..., insert*** в таблице следует установить требуемую позицию. После этого процедуру можно вызывать для каждой записываемой в поле BLOB переменной.

ReadBlob (file, value)

Процедура считывает из поля типа BLOB таблицы ***file*** очередное значение и присваивает его переменной ***value***. Если информация этого поля прочитана до конца, процедура возвращает значение FALSE, иначе – TRUE.

Перед выполнением процедуры необходимо установить позицию в таблице при помощи процедур ***next, prev, get*** и т.п. Далее процедуру ***ReadBlob*** можно вызывать до тех пор, пока она не вернет значение FALSE.

Пример:

```
f1 = TBfile ("blobdata.dbt");
v1 = TRecHandler ("var.str");
f1.next;
while (f1.ReadBlob (v1))
 [ ########   ########################] (v1.rec.f1, v1.rec.f2);
end;
```

Управление файлами и каталогами

В этом разделе содержится описание процедур, с помощью которых осуществляется управление файлами и каталогами.

Большая часть перечисленных процедур входит в состав модуля ***rsexts***, предназначенного для управления файлами и каталогами. В этот модуль также входит стандартный класс ***TDirList*** (см. стр. 140).

CopyFile (src:String, dst:String [, ind:Bool [, indHeading:String]]):Bool

Процедура копирует файл ***src*** в файл ***dst***. Исходный файл и файл назначения могут находиться как на терминале, так и на сервере. Если необязательный параметр ***ind*** равен TRUE, то во время копирования отображается индикатор прогресса. В параметре ***indHeading*** можно задать строку, которая будет отображаться в заголовке окна индикатора. Индикатор прогресса не отображается, если оба файла находятся на терминале.

Примечание.

*Процедура **CopyFile** может обрабатывать файлы без указания расширения.*

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Процедура возвращает TRUE при успешном копировании и FALSE в противном случае.

Пример:

```
/* Копируем файл с сервера на терминал */
if (not CopyFile ("cvt.c","$rslDir\\cvt.c"))
  println ("Error Copy file");
end;
```

RenameFile (src:String, dst:String) :Bool

Процедура переименовывает файл ***src*** в файл с именем ***dst***. Исходный файл может находиться как на сервере, так и на терминале. При этом файл назначения должен располагаться там же, где и исходный файл.

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Процедура возвращает TRUE при успешном переименовании и FALSE в противном случае.

Пример:

```
if (not RenameFile ("$rslDir\\cvt.c", "rslDir\\cvt.c2"))
  println ("Error rename file");
end;
```

RemoveFile (src:String):Bool

Процедура удаляет файл, заданный параметром ***src***. Файл может находиться как на сервере, так и на терминале.

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Процедура возвращает TRUE при успешном удалении и FALSE в противном случае.

Пример:

```
if (not RemoveFile ("$rsldir\\cvt.c2"))
  println ("Error remove file");
end;
```

MakeDir (name:String):Bool

Процедура создает каталог, имя которого задается параметром ***name***. Каталог может быть создан как на сервере, так и на терминале.

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Процедура возвращает TRUE при успешном создании каталога и FALSE в противном случае.

Пример:

```
/* Создаем каталог rslDir на терминале */
if (not MakeDir ("$rslDir"))
  println ("Error crating dir");
end;
```

RemoveDir (name:String):Bool

Процедура удаляет каталог, имя которого задается параметром ***name***. Каталог может быть удален как на сервере, так и на терминале. Удаляемый каталог должен быть пустым.

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Процедура возвращает TRUE при успешном удалении и FALSE в противном случае.

Пример:

```
if (not RemoveDir ("$rslDir"))
  println ("Error remove dir");
end;
```

GetCurDir ([isRemote:Bool]):String

Процедура возвращает название текущего каталога на сервере. Если задан параметр ***isRemote*** и его значение равно TRUE, то возвращается название текущего каталога на терминале.

Чтобы эта процедура была доступна в макропрограмме пользователя, следует явно импортировать модуль ***rsexts***.

Пример:

```
/* Печатаем название текущего каталога на терминале */
println (GetCurDir (true));
```

SplitFile (pathName:String [, name:String [, ext:String]]):String

Процедура разделяет полное имя файла на составляющие: каталог, имя файла, расширение имени файла.

Параметры:

pathName – строка, содержащая полное имя файла.

name – возвращаемое имя файла.

ext – возвращаемое расширение файла.

Возвращаемое значение:

Возвращаемым значением процедуры является имя каталога - строка типа String.

MergeFile (dirName:String, name:String [, ext:String]):String

Процедура возвращает полное имя файла, составленное из компонент.

Параметры:

dirName – имя каталога.

name – имя файла.

ext – расширение имени файла.

FindPath (name:String [, dirList:String [, defExt:String [, curDir:Bool]]]):String

Процедура ищет файл с именем ***name*** в указанном списке каталогов. По умолчанию процедура также осуществляет поиск файла в текущем каталоге.

Параметры:

name – имя файла, который нужно найти.

dirList – список каталогов для поиска, разделенных символом ";". Если этот параметр не задан, то поиск осуществляется в списке каталогов для поиска макрофайлов.

defExt – имя расширения файла по умолчанию, если оно не задано в параметре ***name***. Если параметр не задан, то по умолчанию принимается MAC.

curDir – если этот параметр принимает значение FALSE, то поиск в текущем каталоге не производится.

Возвращаемое значение:

Процедура возвращает найденное имя файла либо пустую строку в случае неудачи.

GetSysDir ([ndir:Integer]):String

Процедура возвращает используемые интерпретатором списки каталогов для поиска файлов с расширениями MAC, DBF и TXT.

Параметры:

ndir – тип списка каталогов, который необходимо вернуть. Параметр может принимать следующие значения:

- **0** – каталог txt-файлов;
- **2** – каталог dbf-файлов;
- **3** – каталог mac-файлов;
- **4** – "домашний" каталог.

Если параметр не задан, то возвращается список каталогов поиска mac-файлов.

GetIniFileValue (iniFileName:String, keyName:String):String

Процедура ищет в файле с именем ***iniFileName*** строку, переданную в качестве параметра ***keyName***, и возвращает найденную строку.

При считывании строки для поиска производится замена специального имени "%HOME" на имя домашнего каталога пользователя. Файл настроек ищется в текущем каталоге приложения, в домашнем каталоге пользователя и в списках каталогов, заданных в переменной окружения RSCNFG.

GetFileInfo (src:String [, dt:@Date, tm:@Time, size:@Integer, path:@String]):Bool

Процедура возвращает информацию о файле с именем ***src***. Файл может располагаться на сервере или на терминале. Если файл находится на терминале, то в этом случае его имя должно начинаться с символа ‘$’.

Параметры:

src – имя файла.

Dt – возвращаемая дата модификации файла.

Tm – возвращаемое время модификации файла.

Size – возвращаемый размер файла.

Path – возвращаемый путь к файлу.

Возвращаемое значение:

Если заданный файл существует, то возвращается значение TRUE. Если файл не существует, то возвращается значение FALSE.

Классы и объекты

В Object RSL предусмотрено несколько вспомогательных процедур, предназначенных для поддержки работы с классами и объектами.

ActiveX (mon:String, const=1):Object

Процедура предназначена для создания объекта класса ActiveX с помощью строкового моникера ***mon***.

Пример:

```
var strComputer = ".";
var str = "winmgmts:{impersonationLevel=impersonate}!\\\\" +
strComputer + "\\root\\cimv2";
var wmiSrv = ActiveX (str, 1);
var colServices = wmiSrv.ExecQuery ("Select * from Win32_Service");
var ob;
for (ob, colServices)
  [####################  ###############]
     (ob.Name,        ob.State);
end;
```

ClassKind (obj:Object [, mask:Integer):Integer

Процедура позволяет определить вид указанного объекта.

Параметры:

obj – объект, вид которого требуется определить.

mask – маска поиска.

Возвращаемое значение:

Возвращаемое значение процедуры образуется как результат логической операции "И" для вида класса и значения маски. Процедура возвращает найденный вид объекта. Возможны следующие варианты:

- **1** – обычный класс RSL или DLM;
- **2** – обёртка над RSCOM-объектом для RSL объекта из другого экземпляра RSL;
- **4** – обёртка над RSCOM объектом, отличным от случая 2.

GenClassName (obj)

Процедура возвращает строку с именем класса для объекта ***obj***.

Если параметр ***obj*** не является ссылкой на объект, то возвращается значение типа V_UNDEF.

GenAttach (id, methodName, macroName | macroAddr)

Процедура позволяет заменить метод ***methodName*** объекта, заданного идентификатором ***id***, глобальной макропроцедурой ***macroName*** или ссылкой на нее ***macroAddr***.

Если после выполнения этой процедуры заменяемый метод вызывается из заменяющей процедуры, то рекурсивный вызов не происходит, а вызывается метод предыдущего уровня замены. Это означает, что при вызове метода ***methodName*** из процедуры ***macroName*** будет выполнен метод ***methodName***, а не заменяющая его процедура ***macroName***.

GenObject (className [, parm1, parm2, ...])

Данная процедура позволяет создавать объекты, имена классов которых задаются при помощи параметров.

Первый параметр – строка, задающая имя класса, к которому относятся объекты. Далее следуют необязательные параметры, передаваемые конструктору (инициализатору) объекта.

Процедура возвращает ссылку на созданный объект. Если объект заданного класса не может быть создан, во время выполнения программы произойдет ошибка.

Доступ к свойствам объектов осуществляется аналогично доступу к полям файлов:

```
Ob = GenObject ("MyClass");
Ob.prop1 = "New value"
```

Вызов методов осуществляется следующим образом:

```
Ob.method1 (par1,par2)
```

GenRun (ob, methodName [, par1, par2, ...])

Процедура вызывает метод с именем ***methodName*** для объекта ***ob*** и передает ему параметры ***par1, par2*** и т.д.

Процедура возвращает значение, возвращаемое вызываемым методом.

```
Val = GenRun (Ob, "Method2", 10)
```

GenSetProp (ob, propName, val)

Данная процедура устанавливает для свойства с именем ***propName*** объекта ***ob*** новое значение, равное ***val***.

GenGetProp (ob, propName)

Процедура возвращает значение свойства с именем ***propName*** объекта ***ob***.

GetObjProps (obj:Object, [CaseSensitive:Bool]):TArray

Процедура возвращает массив с именами свойств объекта ***obj***.

Параметры:

obj – объект, свойства которого возвращает процедура.

CaseSensitive – признак возврата информации о свойствах объекта в программу в исходном виде. Параметр может принимать одно из следующих значений:

- ***TRUE*** – процедура возвращает информацию без изменения регистра.
- ***FALSE*** – в верхнем регистре.

По умолчанию, если значение параметра не задано, оно считается равным ***FALSE***.

GetObjMethods (obj:Object, [CaseSensitive:Bool]):TArray

Процедура возвращает массив с именами методов объекта ***obj***.

Параметры:

obj – объект, имена методов которого возвращает процедура.

CaseSensitive – признак возврата информации о методах объекта в программу в исходном виде. Параметр может принимать одно из следующих значений:

- ***TRUE*** – процедура возвращает информацию без изменения регистра.
- ***FALSE*** – в верхнем регистре.

По умолчанию, если значение параметра не задано, оно считается равным ***FALSE***.

Пример:

```
import rcw;
testObj = MyClass;
[Properties:];
obj = GetObjProps (testObj);
for (i, obj)
  println (i)
end;
[Methods:];
obj = GetObjMethods (testObj);
for (i, obj)
  println (i)
end;
```

IsEqClass (className, obj)

Процедура проверяет принадлежность объекта, заданного параметром ***obj***, классу объектов с именем ***className***.

Процедура возвращает TRUE, если объект принадлежит классу, и FALSE – в противном случае.

Пример:

```
/* Определяем класс AA */
Class AA
End;
/* Наследуем класс BB от класса AA */
Class (AA) BB
End;
If (IsEqClass (“AA”,BB))
   Println (“Объект класса BB является и объектом класса AA”)
End;
```

GenPropID (obj, propName)

Процедура возвращает индекс свойства с именем ***propName*** для объекта ***obj***.

Используя полученный индекс, можно получить доступ к свойству при помощи выражения:

```
A = obj (n);
```

или

```
a = obj [n];
```

R2M (obj:Object, name:String):MethodRef

Процедура возвращает ссылку на метод с именем ***name*** объекта ***obj***.

CallR2M (oPtr:MethodRef [, par1, par2,...]):Variant

Процедура вызывает метод ***oPtr***, заданный в виде ссылки типа ***MethodRef***.

Перед использованием этой процедуры необходимо убедиться, что объект, для которого вызывается метод, существует. Вызов метода для несуществующего объекта приведет к разрушению программы.

Пример:

```
class Test
  macro Method1 (par)
    return “Ret Val ” + par
  end;
end;

var obj = Test;
var ptr = R2M (obj,”Method1”);

println (CallR2M (ptr, “Test Param”);
```

GenNumProps (obj:Object):Integer

Процедура возвращает количество свойств у объекта ***obj***.

Пример:

```
/* Пример процедуры, распечатывающей свойства объекта,
переданного в качестве параметра. */

macro Show (ob:object)

  var n = GenNumProps (ob);

  var i = 0;

  while (i < n)

    println (ob [i]);

    i = i + 1;

  end;

end;
```

Примечание.
Приведенный в примере способ перебора свойств объекта при помощи оператора [] можно использовать не для любых объектов, так как объект может содержать свойства с произвольными идентификаторами (например, объект класса TArray).

ClrRmtOnRelease (proxy:Object):Bool

Процедура при удалении прокси-объекта ***proxy*** автоматически удалит связанный с ним удаленный реальный RSCOM-объект.

Примечание.
Данная процедура поддерживается сервером приложений версии не ниже 5.06.118.

GetNamedChanel (name:String):Object

Процедура проверяет, существует ли коммуникационный канал с именем ***name***. Если существует, то возвращается объект класса ***TRslChannel*** для этого канала. В противном случае возвращается значение NULL.

Индикаторы выполнения процессов

InitProgress (maxRecord:Integer [, msg:String, head:String])

Процедура инициализирует и выводит на экран индикатор выполнения циклического процесса.

В первом параметре необходимо указать максимальное значение, которое может принимать счетчик отображаемого цикла. Каждый раз при переходе к следующему значению счетчика закрашенная полоска индикатора увеличит свою длину.

Остальные параметры процедуры не являются обязательными. Во втором параметре ***msg*** задается сообщение, выводимое в нижнюю строку экрана вместе с индикатором. Третий параметр процедуры должен содержать текст заголовка индикатора выполнения.

Пример:

```
InitProgress (1000);
var i = 0;
while ( i < 1000 )
    i = i + 1;
    UseProgress (i);
end;
RemProgress (i)
```

UseProgress (record)

Процедура приводит изображение на экране в соответствие текущему значению индикатора выполнения циклического процесса.

Единственный параметр ***record*** задает текущее значение счетчика цикла.

RemProgress

Данная процедура удаляет с экрана окно индикатора выполнения, выведенное процедурой ***InitProgress***.

BegAction ([tm:Integer, text:String, canClose:Bool])

Процедура выводит на экран асинхронный индикатор, предназначенный для отображения занятости приложения. Эта процедура должна использоваться совместно с процедурой ***EndAction***.

Процедура работает только в режиме RS-Forms и EasyWin.

Параметры:

tm – время в миллисекундах, через которое появляется индикатор, если не было вызова процедуры ***EndAction***. Если параметр не задан, его значение принимается равным 2000.

text – текст, выводимый на индикаторе вместо стандартного.

canClose – признак доступности кнопки **"Отменить"**. По умолчанию кнопка доступна. Если этот параметр равен FALSE, кнопка недоступна.

EndAction ([tm:Integer])

Процедура убирает с экрана асинхронный индикатор активности.

Если индикатор был выведен, то параметр ***tm*** задаёт задержку в миллисекундах, по истечении которой индикатор должен быть убран с экрана. По умолчанию параметр имеет значение 200.

Другие процедуры

CheckBits (n1:Integer, n2:Integer):Integer

Процедура в качестве параметров принимает два целых числа и возвращает результат операции побитового "И" для этих чисел.

Пример:

```
println (CheckBits (#ff, #80));
//Выводит:
//128
```

DateShift (inData:Date [, nDay:Integer] [, nMon:Integer] [, nYear:Integer]):Date

Процедура позволяет выполнить смещение заданной даты на требуемое количество дней, месяцев и лет.

Параметры:

inData – дата, смещение которой необходимо выполнить.

nDay – количество дней, на которое требуется выполнить смещение.

nMon – количество месяцев, на которое требуется выполнить смещение.

nYear – количество лет, на которое требуется выполнить смещение.

Возвращаемое значение:

Процедура возвращает дату, полученную в результате необходимого смещения.

Пример:

```
// Вернуть дату, отстоящую от текущей на 5 месяцев:
Var result = DateShift (date, null, 5);
```

ExecMacro (string,.....)

Данная процедура вызывает для выполнения процедуру, заданную в качестве первого параметра ***string***. Это может быть:

- собственно процедура, в этом случае в качестве первого параметра передается ее имя в виде символьной константы или переменной типа String либо передается ссылка на данную процедуру;
- метод объекта, в этом случае в качестве первого параметра передается ссылка на метод объекта.

После первого параметра указываются те значения, которые должны быть переданы вызываемой процедуре в качестве фактических параметров.

Если процедура с именем ***string*** существует, процедура ***Execmacro*** возвращает значение TRUE, в противном случае возвращается FALSE.

Пример:

```
Macro MyMacro (parm)
        PrintIn ("Вызвана процедура MyMacro с параметром", parm)
End;
ExecMacro (@MyMacro, "Это параметр для MyMacro")
```

ExecMacro2 (string, ...)

Процедура эквивалентна процедуре ***ExecMacro***, но возвращает значение, которое возвращает запускаемая процедура.

ExecMacroFile (Module [, ProcName [, Parm1,Parm2, ...]])

Данная процедура производит компиляцию и выполнение макромодуля, заданного строкой ***Module***. Строка ***ProcName*** задает имя макропроцедуры внутри модуля ***Module***, которой передается управление. Список параметров ***Parm1***, ***Parm2*** и т.д. передается процедуре ***ProcName***.

Возвращаемое значение

Процедура ***ExecMacroFile*** возвращает значение, возвращаемое процедурой ***ProcName***. Если процедура ***ProcName*** не задана или не существует, то возвращается значение типа V_UNDEF.

Пример:

```
/*Предположим, у нас имеется модуль add2.mac:*/
println ("This is add.mac");
macro MainProc
 println ("This is MainProc");
i = 0;
```

```
 while (getParm (i,a))
   println (a);
   i = i + 1
 end;

 return "RetFrom MainProc";
end;

/* Мы можем запустить на выполнение модуль add.mac из модуля*/
/* main.mac */

println ("This is main.mac");

println (ExecMacroFile ("add","MainProc",10,20,"par"))

/*Результат выполнения макромодуля main.mac должен выглядеть
так:*/

This is main.mac
This is add.mac
This is MainProc
10
20
par
RetFrom MainProc
```

ExecMacroModule (codeStr:String [, macroName, par1, par2, …])

Процедура аналогична процедуре ***ExecMacroFile***, но принимает код RSL-модуля в виде строки ***codeStr***.

ReplaceMacro (string1, [string2])

Данная процедура позволяет заменить процедуру с именем ***string1*** на глобальную процедуру с именем ***string2***. После этого любой вызов процедуры ***string1*** будет перенаправляться в процедуру с именем ***string2***.

Заменить можно не только макропроцедуру, но и любую стандартную процедуру, такую как **println**, **open**, **next**, **prev** и т.д.

Если заменяемая процедура вызывается из заменяющей, то перенаправление вызова не происходит: из заменяющей процедуры вызывается процедура предыдущего уровня замены. Одна процедура может быть заменена несколько раз.

Если параметр ***string2*** не задан, то это приводит к снятию замены и восстановлению оригинальной процедуры. Если процедура с именем ***string1*** была заменена несколько раз, то для восстановления оригинальной процедуры необходимо столько же раз вызвать ***ReplaceMacro*** без параметра ***string2***.

Пример:

```
/*  Пример замены макропроцедур */
macro  Test
  println ("Это Test")
end;
macro ReplTest
  println ("Это ReplTest");
  Test
end;
macro ReplReplTest
```

```
  println ("Это ReplReplTest");
  Test
end;
/* Выполняем первую замену процедуры Test */
ReplaceMacro ("Test","ReplTest");
println ("Вызываем Test после выполнения первой замены...");
Test;
/* Выполняем вторую замену процедуры Test */
ReplaceMacro ("Test","ReplReplTest");
println ("Вызываем Test после выполнения второй замены...");
Test;
println ("Отменяем одну замену...");
ReplaceMacro ("Test");
Test;
println ("Отменяем вторую замену...");
ReplaceMacro ("Test");
Test;
```

Результат работы этой макропрограммы должен выглядеть так:

```
Вызываем Test после выполнения первой замены...
Это ReplTest
Это Test
Вызываем Test после выполнения второй замены...
Это ReplReplTest
Это ReplTest
Это Test
Отменяем одну замену...
Это ReplTest
Это Test
Отменяем вторую замену...
Это Test
```

ExecExp (string)

Процедура вычисляет выражение, заданное параметром ***string***, и возвращает результат вычисления.

Пример:

```
Val = ExecExp (“10 + 20*5”);
```

GCollect (ncol:@Integer):Integer

Процедура выявляет объекты классов языка RSL и класса **TArray**, на которые нет ссылок в RSL-программе, и пытается освободить такие объекты. При помощи процедуры ***PrintRefs*** для объектов в стандартный выходной поток автоматически выводится информация о ссылках.

Параметры:

ncol – количество объектов без ссылок из RSL-программы, оставшихся после попытки их очистки.

Возвращаемое значение:

Процедура возвращает количество обнаруженных объектов без ссылок из RSL-программы.

GetCallStack ()

Процедура возвращает массив строк с названиями процедур в текущем стеке вызовов.

Пример:

```
macro Proc1
  macro LocalProc
     a = GetCallStack;
     for (e, a)
       println (e)
     end;
  end;
  LocalProc
end;
```

GetEnv (string)

Процедура возвращает значение переменной среды с именем, заданным в виде символьной строки ***string***.

Пример:

```
println ( getenv ("PATH") );
```

GetMemAddrFrom (obj:Variant):MemAddr

Процедура возвращает адрес буфера для объекта, переданного в параметре ***obj***, в виде значения типа MemAddr. Данное значение может использоваться для передачи в некоторые служебные процедуры RS-Bank V.6.

GetUIMode (…)

Процедура возвращает код, обозначающий тип пользовательского интерфейса приложения. Возможны следующие значения:

- **0** – невозможно определить тип интерфейса;
- **1** – консольный интерфейс;
- **2** – EasyWin интерфейс;
- **3** – интерфейс RS-Forms.

InstLoadModule (moduleName:String):Bool

Процедура выполняет динамическую загрузку RSL-модуля в текущий экземпляр интерпретатора RSL. Если модуль с заданным именем уже присутствует в данном экземпляре RSL, процедура не выполняет никаких действий.

Параметры:

moduleName – имя RSL-модуля, загрузку которого требуется выполнить.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

IsWeakRef (obj:Object):Bool

Процедура позволяет определить, является ли объект, указанный в параметре ***obj***, "слабой" ссылкой. Если ссылка "слабая", процедура возвращает TRUE, в противном случае – FALSE.

WriteByte (stream:Object, byte:Integer):Bool

Процедура предназначена для побайтовой записи в поток IRsStream.

Параметры:

stream – поток, для которого требуется выполнить операцию.

byte – номер байта, начиная с которого требуется выполнить запись.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

ReadByte (stream:Object, byte:@Integer):Bool

Процедура предназначена для побайтового чтения потока IRsStream.

Параметры:

stream – поток, для которого требуется выполнить операцию.

byte – номер байта, начиная с которого требуется произвести чтение.

Возвращаемое значение:

Процедура возвращает одно из значений:

- TRUE – в случае успешного выполнения.
- FALSE – в случае возникновения ошибки.

RunError ([mes:String] [, userObj:Variant])

Процедура вызывает аварийное завершение RSL-программы с генерацией ошибки, возникающей во время ее выполнения. Кроме этого, производится откат всех изменений, внесенных в таблицы баз данных в рамках транзакции как на RSL, так и в вызывающем модуле на C или C++.

Параметр ***mes*** задает символьную строку, содержащую текст сообщения об ошибке.

Параметр ***userObj*** задает специфическую информацию пользователя об ошибке. Может использоваться значение любого типа RSL.

Пример:

```
Class MyError (v)
  Var erCode = v;
```

```
End;
RunError (“My Error”, MyError (78));
```

Если при аварийном завершении программы управление передается обработчику ошибок ***onError***, то в самом обработчике можно вызвать процедуру ***RunError*** без параметров. В этом случае будет заново сгенерирована та же самая ошибка, из-за которой программа была аварийно завершена. Этот метод позволяет обработать ошибочную ситуацию и снова прервать выполнение программы.

Пример:

```
macro Test
/* Генерация ошибки при выполнении программы и передача
управления обработчику ошибок */
RunError ("My error");
onError
  println ("My handler");
 // Прерывание работы программы и генерация предыдущей
ошибки
  RunError;
end;
```

Exit ([code:Integer] [, mes:String])

Данная процедура прекращает выполнение макропрограммы. Если эта процедура вызвана в рамках транзакции, то она завершает выполнение макросов, вызванных для выполнения транзакции, и актуализирует все изменения, внесенные в таблицы базы данных в рамках транзакции как на RSL, так и в вызывающем модуле на C или C++. Процедура ***ProcessTrn*** при этом возвращает значение TRUE, а вся RSL-программа не завершается.

Параметр ***code*** задает режим просмотра результата работы программы и может принимать следующие значения:

- **0** – результат работы программы будет показываться в окне просмотра; это значение используется по умолчанию;
- **1** – результат работы программы не будет показываться в окне просмотра;
- **2** – файл вывода не показывается, а сразу распечатывается на принтере.

Параметр ***mes*** задает сообщение, которое выводится в выходной поток.

SetExitFlag (code:Integer)

Процедура позволяет установить режим просмотра результата работы программы, заданный в параметре ***code***, аналогично процедуре ***Exit***, но не завершает выполнение RSL-программы.

MemSize()

Процедура возвращает количество свободной оперативной памяти, доступной для работы программы. При работе под Windows всегда возвращает ноль.

Примечание.
Процедура оставлена в RSL для совместимости с ранее написанными программами.

Version()

Процедура возвращает целое число – номер версии RSL.

CurrentLine ([line])

Процедура возвращает номер текущей строки в выходном файле.

Необязательный параметр ***line*** задает новое значение для счетчика номеров строк.

Пример:

```
If (CurrentLine > 40)
    НапечататьПодвал;
    CurrentLine (0);
end
```

UserNumber()

Процедура возвращает номер пользователя, под которым он зарегистрирован в компьютерной сети. Использование этой процедуры позволяет формировать уникальные имена файлов для разных пользователей.

Пример:

```
MyFileName = "DemoFile." + UserNumber;
SetOutput (MyFileName)
```

Random ([integer])

Процедура возвращает случайное число в диапазоне:

- от ***0*** до значения ***integer-1***, если параметр задан;
- от ***0*** до ***32766***, если параметр ***integer*** не задан.

SetDefMoneyPrec (newVal:Integer):Integer

Процедура используется для установки нового значения количества знаков после запятой; применяется только для денежного типа.

Возвращает предыдущее значения количества знаков после точки.

Параметры:

newVal – новое значение количества знаков после запятой.

Примечание.

Если в качестве денежного типа используется FDecimal, то допустимы только значения 0, 2, 4.

Возвращаемое значение:

Процедура возвращает предыдущее значение количества знаков после запятой.

Пример:

```
cpwin;
var mn1 = $12.3456789;
macro OutPrec (val:money, prec:integer)
  var old = SetDefMoneyPrec (prec);
  var result = string (val);
  SetDefMoneyPrec (old);
  return result
```

```
end;
[ Точность 0  ##################
  Точность 1  ##################
  Точность 2  ##################
  Точность 3  ##################
  Точность 4  ##################
  Точность 5  ##################
  Точность 6  ##################
  Точность 7  ################## ]
(  OutPrec (mn1,0):r,
  OutPrec (mn1,1):r,
  OutPrec (mn1,2):r,
  OutPrec (mn1,3):r,
  OutPrec (mn1,4):r,
  OutPrec (mn1,5):r,
  OutPrec (mn1,6):r,
  OutPrec (mn1,7):r);
```

ShowDictError (show:Bool):Bool

Процедура устанавливает признак вывода диагностических сообщений о несоответствии физического Btrieve файла его описанию в словаре базы данных.

Параметры:

show – признак вывода диагностических сообщений. Параметр принимает следующие значения:

- TRUE – сообщения выводятся (значение по умолчанию).
- FALSE – сообщения не выводятся.

Возвращаемое значение:

Процедура возвращает предыдущее значение признака вывода диагностических сообщений.

ShowRSCOMError (obj:TRsComErr)

Процедура отображает ошибки RSCOM в стандартном диалоговом окне.

StartProg (fileName:String [, cmdLine:String] [, detached:Bool]):Integer

Процедура предназначена для запуска приложения на терминале или на сервере (с ожиданием завершения или без ожидания).

Параметры:

fileName –имя запускаемой программы. Если имя программы начинается с символа "$", то параметр задаёт файл на терминале, иначе – на сервере приложений.

cmdLine – строка с аргументами командной строки для запускаемого приложения.

Detached – признак ожидания завершения. Возможные значения:

- TRUE – процедура не ждёт завершения запущенной программы (значение по умолчанию).
- FALSE – выполнение приостанавливается до завершения запущенной программы.

Возвращаемое значение:

Процедура возвращает одно из значений:

- В режиме ожидания – код завершения запускаемого процесса.
- В режиме выполнения:
  - **0**, если программа успешно стартована;
  - **-1** – в случае возникновения ошибки.

StrongRef (par:Object):Object

Процедура возвращает "сильную" ссылку на объект ***par***. Основное назначение процедуры – получение "сильной" ссылки на объект, для которого имеется только "слабая" ссылка.

Параметры:

par – "сильная" или "слабая" ссылка на объект.

Пример:

```
class  X
end;
ob = X;
weak = WeakRef (ob);
strong = StrongRef (weak);
```

SysGetProperty (key:String):String

Процедура позволяет получить значение глобального свойства ***key***.

SysPutProperty (key:String, val:String):Bool

Процедура позволяет в качестве значения свойства ***key*** установить значение ***val***. Если значение ***val*** раво ***null***, указанное свойство удаляется из коллекции. Процедура возвращает TRUE в случае успешного изменения значения свойства.

Примечание.
*Процедуры **SysGetProperty** и **SysPutProperty** используются для получения доступа к глобальной коллекции свойств приложения. Коллекция свойств доступна из кода на языках C++, RSL, RSCOM и Java. Таким образом, указанная коллекция позволяет выполнять процесс обмена информацией между различными компонентами приложения.*

System (Number:Integer, CodeFor:Integer|Type:String [, CmdArgs:String])

Процедура активизирует выполнение системного модуля, заданного идентификатором ***Number*** и вызванного из активной подсистемы.

Второй параметр задает подсистему: это либо номер (параметр ***CodeFor***), либо буквенный код (параметр ***Type***) подсистемы, из которой модуль вызывается на выполнение. Номера и буквенные коды подсистем

определяются в таблице базы данных, которая во внутреннем словаре имеет имя ***typeac.dbt***.

Необязательный параметр ***CmdArgs*** задает строку параметров для запуска модуля.

Пример:

```
System(16522, CodeFor("Z"));

/* Для подсистемы "Депозитарий" на экран будет выдана панель
подготовки отчета "Отчет об инвентарном исполнении

перации" (системный модуль 16522).*/
```

IsStandAlone

Программный комплекс RS-Bank V.6 может работать в двух- либо трехуровневой архитектуре "клиент-сервер" (см. Руководство пользователя "Установка системы"). Процедура ***IsStandAlone*** позволяет определить, в каком именно режиме функционирует приложение.

Данная процедура возвращает значения:

- TRUE – если программа выполняется в двухуровневой архитектуре.
- FALSE – если программа выполняется в трехуровневой архитектуре.

Пример:

```
if (IsStandAlone)
    println ("StandAlone");
else
    println ("Communication");
end;
```

TestEvent ([pause])

Процедура позволяет узнать, была ли нажата клавиша на клавиатуре или кнопка "мыши", и возвращает:

- код нажатой клавиши – если нажатие было произведено;
- ноль – если нажатий не было.

Необязательный параметр ***pause*** используется для 32-разрядных программ и определяет время в миллисекундах, по прошествии которого процедура вернет ноль, если нажатий клавиши клавиатуры или кнопки "мыши" не произошло. Если до истечения заданного интервала времени будет произведено движение мышью, то процедура выйдет из цикла ожидания раньше времени и вернет ноль.

Внимание!
Следует иметь в виду, что максимально допустимое значение параметра ***pause*** *составляет 1000 миллисекунд (т.е. 1 секунду). При указании большего значения, оно автоматически изменяется на 1000.*

Если в качестве этого параметра указать значение "-1", то процедура будет дожидаться нажатия кнопки "мыши" или любой клавиши клавиатуры. По умолчанию параметр ***pause*** принимается равным нулю.

Процедуру ***TestEvent*** можно использовать для прерывания цикла:

Пример:

```
File ff ("account.dbt");
Const ESC = 27;
Var Done = FALSE;
While ( not Done && next (ff) )
        Println ( ff.account );
        If ( TestEvent == Esc )
                Done = TRUE
        End
End
```

AddEvent (key:Integer)

Процедура добавляет в очередь клавиатурное сообщение с кодом ***key***.

IsGUI

Данная процедура определяет среду выполнения макропрограммы и возвращает следующие значения:

- FALSE – если макропрограмма выполняется консольным приложением Windows NT или программой DOS.
- TRUE – если макропрограмма выполняется в графической среде Visual RSL.

ErrPrint (...)

Процедура печатает свои параметры в стандартный поток ошибок.

ErrBox (str:TArray [,caption:String, flags:Integer]):Integer

Процедура отображает в режиме RSFM стандартное диалоговое окно, отображающее список строк. В консольном режиме строки выводятся в обычном диалоговом окне без кнопок.

Параметры:

str – массив строк типа ***TArray***.

Caption – заголовок окна.

Flags – вид стандартных иконок и выравнивание текста в окне. В качестве значений параметра передаётся сумма следующих значений:

- **16** – окно будет иметь вид стандартного сообщения об ошибке;
- **32** – окно будет выглядеть как запрос пользователю с несколькими вариантами ответа в виде стандартных кнопок;
- **48** – окно будет иметь вид стандартного предупреждения;
- **64** – окно будет иметь вид информационного сообщения;
- **65536** – выравнивание текста по правому краю;
- **131072** – выравнивание текста по центру;

- **2097152** – в окне будет присутствовать стандартная кнопка минимизации окна;
- **4194304** – окно будет выведено на экран без звукового сигнала.

Пример:

```
cpdos;
ob = TArray ();
ob [0] = "Строка1";
ob [1] = "Строка2";
ob [2] = "String3";
ob [3] = "String4";
println (ErrBox (ob,"Заголовок",2097152+131072+64+0));
```

ModuleFileName ([moduleName:String] [, moduleType:@Integer]):String

Процедура возвращает строку с именем файла для заданного модуля.

Парамеры:

moduleName – имя исполняемого модуля. Если параметр не задан, используется имя модуля, из которого была вызвана указанная процедура.

moduleType – тип модуля; возвращаемый параметр. Возможные значения:

- **0** – заданный модуль не найден;
- **1** – макромодуль, загруженный из mac-файла;
- **2** – прекомпилированный модуль, загруженный из rsm-файла;
- **3** – модуль dlm;
- **4** – встроенный модуль;
- **5** – макромодуль, загруженный из базы данных *btrmac.ddf*.

ModuleName ([symbolName:string]):string

Процедура возвращает строку с именем исполняемого модуля, в котором определена переменная, процедура или класс.

Парамеры:

symbolName – имя переменной, процедуры или класса. Если параметр не задан, возвращается имя модуля, из которого был вызван указанный элемент.

CmdArgs:String

Процедура возвращает строку с параметрами, которая задается в командной строке при старте приложения.

Пример:

```
/* Запуск утилиты RSL со строкой параметров "Par1 par2 par3": */
rsl32.exe bank.def 00.mac "Par1 par2 par3"
/* В этом случае процедура CmdArgs возвратит значение: */
/* "Par1 par2 par3" */
```

GetUserName:String

При работе в трехзвенной архитектуре процедура возвращает сетевое имя пользователя, присоединенного к серверу приложений, в двухзвенной архитектуре – имя текущего пользователя.

IsSQL :bool

Процедура возвращает TRUE, если код исполняется в SQL-версии RSL, и FALSE в противном случае.

UnderRCWHost:Bool

Процедура возвращает TRUE, если RSL-файл исполняется модулем *RCWHost.d32*. В противном случае возвращается значение FALSE.

GetLocaleInfo (id:Integer, code:Integer [, isLocal:Bool]):String

Процедура возвращает строку с запрашиваемой региональной настройкой.

Параметры:

id – код языка, для которого необходимо получить настройку. Параметр может принимать следующие значения:

- **0** – язык, установленный для пользователя, работающего с программой;
- **1** – язык системы по умолчанию;
- **2** – русский язык.

code – код региональной настройки, которую требуется получить. Параметр может принимать следующие значения:

- **LOCALE_SDECIMAL** – разделитель дробной части.
- **LOCALE_SMONDECIMALSEP** – разделитель копеек.
- **LOCALE_SDATE** – разделитель даты.
- **LOCALE_STIME** – разделитель времени.
- **LOCALE_SLONGDATE** – формат даты.
- **LOCALE_SSHORTDATE** – короткий формат даты.
- **LOCALE_STIMEFORMAT** – формат времени.

isLocal – признак, определяющий, где необходимо выполнить запрос. Если параметр не задан или задан равным TRUE, запрос выполняется на сервере. Если параметр равен FALSE, то запрос выполняется на терминале.

GetLangId (id:Integer [, isLocal:Bool]):Integer

Процедура возвращает идентификатор языка.

Параметры:

id – код языка, для которого необходимо получить идентификатор. Значения параметра такие же, как и для процедуры ***GetLocaleInfo***.

isLocal – признак, определяющий, где необходимо выполнить запрос. Значения параметра такие же, как и для процедуры ***GetLocaleInfo***.

ZeroValue (valtp:Integer):Variant

Процедура возвращает нулевые значения для типа данных, код которого задан параметром ***valTp***.

Пример:

```
PrintIn (ZeroValue (V_DOUBLE));
```

Средство разработки расширений для языка RSL (DLM SDK)

Язык RSL является прекрасным средством настройки программных комплексов, разрабатываемых компанией R-Style Softlab под конкретные требования пользователей. Он позволяет создавать макропрограммы, которые формируют нестандартные отчеты, проводят анализ или диагностику базы данных, выполняют другие функции. Встречаются ситуации, когда для решения поставленных задач стандартных средств языка не хватает. Для устранения этой проблемы был создан инструмент DLM SDK.

DLM SDK предназначен для разработки на языке C/C++ динамически загружаемых модулей. Использование этого инструмента позволяет дополнить RSL новыми эффективными процедурами и даже классами объектов, используя для этого всю мощь компилируемых языков C/C++.

В состав дистрибутивного комплекта DLM SDK входят:

- библиотеки;
- заголовочные файлы;
- документация;
- примеры использования.

В этом разделе Руководства представлено описание наиболее простых примеров создания DLM-модулей. Полная документация с примерами приведена в дистрибутиве.

Создание и использование DLM-модулей

Двоичные DLM-модули (DLM-ы), созданные при помощи DLM SDK, загружаются в память динамически во время выполнения RSL-программы.

В зависимости от платформ, для которых создаются DLM-модули, компилятор должен удовлетворять следующим требованиям:

- Для создания DLM-ов для защищенного режима необходим компилятор фирмы "Borland" версии 4.02 или выше, а также система Power Pack for DOS.
- Для создания DLM-ов для 32-разрядного защищенного режима DOS и WIN32 необходим компилятор фирмы "Borland" версии 4.0 или выше или компилятор фирмы "Microsoft" Visual C++ версии 5.0.

Создаваемые при помощи DLM SDK модули представляют собой обыкновенные 32-разрядные DLL-библиотеки.

Для того, чтобы одна и та же макропрограмма, созданная с использованием DLM SDK, могла исполняться разными интерпретаторами RSL принято следующее соглашение о выборе имен для DLM-модулей:

- имя файла должно удовлетворять требованиям операционной системы;
- расширение файла должно быть следующим: **d32** – для режимов DPMI32 и WIN32.

Для того, чтобы DLM модуль стал доступен RSL-программе, имя модуля без его расширения необходимо указать в директиве RSL ***import***.

Пример:

```
import demo; //Подключение файла, содержащего код DLM-модуля
```

Рассмотрим пример DLM-модуля, который реализует для языка RSL процедуру с именем *"RlsHello"*. Данная процедура выводит в стандартный выходной поток текстовую строку *"Вызвана процедура RslHello"*.

Пример:

```
#include "rsl\rsldll.h"
static void DLMAPIC RslHello (void)
{print ("Вызвана процедура RslHello\r\n");}
void DLMAPI EXP AddModuleObjects (void)
{AddStdProc (V_UNDEF,"RslHello",RslHello,0);}
```

Проанализируем этот код.

Сначала к программе подключается файл *rsldll.h*, содержащий описание интерфейса с интерпретатором RSL.

Затем вводится определение процедуры *RslHello*, которая будет вызываться из RSL-прграммы. Следует отметить, что все процедуры, которые вызываются из макропрограмм на языке RSL, должны иметь прототип, как приведенная *RslHello*: не принимать параметров, не возвращать значений и иметь тип вызова DLMAPIC. Для передачи параметров и возвращения значений предусмотрен специальный механизм, который подробно рассматривается в подразделе "Передача параметров и возврат значений" (см. стр. 241).

После создания процедуры для RSL ее необходимо зарегистрировать при помощи стандартной функции ***AddStdProc***. Функция ***AddStdProc*** имеет четыре параметра:

- тип возвращаемого значения регистрируемой процедуры;
- строка с именем, которое будет использоваться для вызова процедуры из макропрограммы;
- указатель на функцию, реализующую алгоритм процедуры;
- четвертый параметр зарезервирован и всегда должен быть равен 0.

Функция ***AddStdProc*** вызывается для каждой регистрируемой процедуры из функции ***AddModuleObjects***, которую необходимо определять аналогично определению в примере. Интерпретатор RSL вызывает функцию ***AddModuleObjects*** автоматически при загрузке DLM-модуля.

Теперь необходимо выполнить компиляцию созданного нами модуля.

Предположим, что исходный текст нашего DLM-модуля находится в файле с именем demo.c. Тогда, чтобы осуществить компиляцию модуля для платформы DPMI32, необходимо выполнить команду:

```
bcc32 -WXD -Ih -edemo.32 demo.c lib\rsldlm32.lib\dpmi32x.lib
```

В результате компиляции будет получен модуль *demo.d32*.

Приведем пример использования созданного DLM-модуля:

Пример:

```
import demo;
RslHello;
```

Передача параметров и возврат значений

Для доступа к переданным в макропроцедуру параметрам применяется функция ***GetParm***, имеющая два аргумента:

- первый аргумент задает номер параметра, значение которого необходимо получить;
- второй – возвращает ссылку на значение RSL-переменной.

Если запрашиваемый параметр не существует, функция возвращает значение FALSE. В этом случае второй аргумент инициализируется ссылкой на временное значение неопределенного типа V_UNDEF.

Переменная RSL описывается структурой VALUE. В поле v_type этой структуры хранится тип переменной, а в поле value собственно значение переменной для конкретного типа данных.

Для возврата значений в RSL-программу служит функция ***ReturnVal***, которая задает возвращаемое значение и имеет два параметра:

- тип возвращаемого значения;
- указатель на возвращаемое значение.

Покажем на примере, как передаются параметры и возвращаются значения в макропрограмму, созданную на языке RSL.

Пример:

Разработаем процедуру MultiStrLen, которая принимает в качестве параметров произвольное количество строк и возвращает их длину:

```
#include "rsl\rsldll.h"
#include <string.h>
static void DLMAPIC MultiStrLen (void)
{
VALUE *v;
int i = 0;
long len = 0;
```

```
for ( i = 0; GetParm (i,&v); ++i )
if ( v-> v_type == V_STRING)
len += strlen ( v->value.string);
        Return Val (V_INTEGER,&len);
}
void DLMAPI EXP AddModuleObjects (void)
{
AddStdProc (V_INTEGER,"MultiStrLen",MultiStrLen,0);
}
```

Все параметры в RSL-процедуры передаются по значению. Однако, используя стандартную функцию ***PutParm***, можно изменить значение фактического параметра в программе, вызвавшей эту процедуру.

Пример:

Создадим процедуру RetDemo, которая с помощью первого параметра возвращает значение 200:

```
Static void DLMAPIC RetDemo (void)
{
long val = 200;
PutParm ( 0, V_INTEGER, &val );
}
```

Функция ***PutParm*** отличается от функции ***ReturnVal*** дополнительным первым аргументом, который задает номер устанавливаемого параметра.

Имеется возможность инициализации и деинициализации DLM-модуля. Для этого служат две специальные функции: ***InitExec*** и ***DoneExec***. Функция ***InitExec*** вызывается автоматически интерпретатором RSL перед началом выполнения RSL-программы, а ***DoneExec*** – после выполнения.

Пример:

```
void DLMAPI EXP InitExec (void)
{
//Выполняем инициализацию
.....
}
void DLMAPI EXP DoneExec (void)
{
//Выполняем освобождение занятых ресурсов
.....
}
```

Помимо процедур, вызываемых из RSL-программ, при помощи инструмента DLM SDK можно создавать целые классы объектов со свойствами и методами. Объекты классов в RSL создаются и уничтожаются с помощью механизма поставщиков объектов. В DLM-модуль можно включить код, реализующий поставщик объектов требуемого типа, и, используя функцию ***AddModuleObject***s, зарегистрировать этого поставщика.

Сводка синтаксиса RSL

В этом приложении приведен формальный синтаксис языка RSL на РБНФ.

Все терминалы набраны заглавными буквами.

NUMBER – обозначает константу типа Integer, Double, Money;

NAME – последовательность букв и цифр, начинающаяся с буквы, символ подчеркивания '_' считается буквой.

SPNAME – последовательность любых символов, заключенная в фигурные скобки: {34-23-O}.

STRING – строка символов, заключенная в кавычки. В строке могут быть указаны Esc- символы:

\n – новая строка;

\r – возврат каретки;

\t – символ табуляции;

\f – перевод формата;

\xHH, \XHH – символ, заданный кодом (HH две шестнадцатеричные цифры)

Необязательные конструкции заключены в квадратные скобки. Конструкции, которые могут повторяться произвольное количество раз или отсутствовать, заключены в фигурные скобки. Альтернативы разделены символом '|'.

unit	**:=**	**u_decl_list [errhandler] [END].**
u_decl_list	**:=**	**u_decl { ';' u_decl }.**
errhandler	**:=**	**ONERROR ['(' id ')'] decl_list.**
decl_list	**:=**	**decl { ';' decl }.**
u_decl	**:=**	**import_def | decl.**
import_def	**:=**	**IMPORT file_nm_lst.**
file_nm_lst	**:=**	**file_name { ',' file_name }.**
file_name	**:=**	**id | STRING.**
id	**:=**	**NAME | SPNAME.**
decl	**:=**	**macro_def | var_def | const_def | array_def | file_def | record_def | class_def | with_def | statement.**
macro_def	**:=**	**[attr] MACRO id [form_parm] [tpdecl] decl_list [errhandler] END.**
form_parm	**:=**	**'(' [id_list] ')'.**
id_list	**:=**	**tpid { ',' tpid }.**

tpid := id [tpdecl]

tpdecl := ':' tpname

tpname := id | VARIANT | INTEGER | MONEY | DOUBLE | DOUBLEL | STRING | BOOL | DATE | TIME | OBJECT | PROCREF | DATETIME | R2M | MEMADDR.

var_def := [attr] VAR init_id_list.

init_id_lst := init_var { ',' init_var }.

init_var := tpid ['=' expression].

const_def := [attr] CONST const_lst.

const_lst := init_const { ',' init_const }.

init_const := tpid '=' expression.

array_def := [attr] ARRAY id_list.

file_def := [attr] FILE id '(' file_name [',' file_name] ')' [fparm_lst].

fparm_lst := fparm { fparm }.

fparm := NORMAL | KEY NUMBER | WRITE | MEM | TXT [NUMBER] | BTR | SORT NUMBER | DBF | DIALOG | BLOB.

record_def := [attr] RECORD id '(' file_name [',' file_name] ')' [fparm_lst].

class_def := [attr] CLASS ['(' id ')'] id [form_parm] decl_list [errhandler] END.

with_def := WITH '(' id ')' decl_list END.

attr := PRIVATE | LOCAL .

statement := [loop | ifstmt | retstmt | outstmt | expstmt].

loop := WHILE condition stmt_list END.

ifstmt := IF condition stmt_list
{ ELIF condition stmt_list }
[ELSE stmt_list] END.

condition := '(' expression ')'.

retstmt := RETURN [expression].

outstmt := '[' control_str ']' ['(' [explist] ')'].

control_str := print_ch | format_ch.

print_ch := любой символ, не равный format_ch.

format_ch := '#'.

expstmt := expression.

stmt_list := statement { ';' statement }.

expression	**:=**	**BoolExp { '=' BoolExp }.**
BoolExp	**:=**	**SimplExpr [relop SimplExpr].**
SimplExpr	**:=**	**['+' | '-'] Term { AddOp Term }.**
Term	**:=**	**Factor { MulOp Factor }.**
Factor	**:=**	**NUMBER | STRING | '(' expression ')' | NOT Factor | '@' Factor | qualId.**
qualId	**:=**	**id { '.' id | '(' [expList] ')' }**
expList	**:=**	**exp_form { ',' exp_form }.**
exp_form	**:=**	**expression [fmt_list].**
fmt_list	**:=**	**fmt_spec { fmt_spec }.**
fmt_spec	**:=**	**':' fmt_symbol**
fmt_symbol	**:=**	**NUMBER | 'l' | 'r' | 'c' | 'a' | 't' | 'd' | 'm' | 'w' | 'z' | 'f' | 'i' | 'iv' | 'v'.**
relop	**:=**	**'==' | '!=' | '<' | '<=' | '>' | '>=' .**
AddOp	**:=**	**'+' | '-' | OR .**
MulOp	**:=**	**'*' | '/' | AND .**

Существуют следующие правила:

- Все операции, кроме операции присваивания, имеют левую ассоциативность; операция присваивания правоассоциативна.
- Приоритет группы relop самый низкий, MulOp самый высокий.
- Порядок вычисления операндов не определен для всех операций, кроме OR (логическое ИЛИ) и AND (логическое И). Для операций OR и AND сначала вычисляется левый операнд, далее, если необходимо, вычисляется правый операнд.

Алфавитный указатель

_extObj, 38
_callHost, 148
AbortTrn, 213
Abs, 181
ActiveX, 46, 219
AddColumn, 93
AddEvent, 235
AddMultiAction, 192
AddScroll, 186
APPEND, 105
ARRAY, 43
ArrayRef, 15
Asize, 173
AttachSite, 160
AxCode, 63
AxMes, 63
BegAction, 224
Bool, 14
BREAK, 32
BtFileRef, 15
BTR, 106
BtrError, 126
Call, 160
CallR2M, 222
CallRemoteRsl, 184
Cast, 157
CheckBits, 224
ClassKind, 219
ClearColumn, 170
ClearRecord, 210
ClearStructs, 203
Clone, 203
Close, 202
ClrObj, 61
ClrRmtOnRelease, 223
ClrRmtStubs, 150
CmdArgs, 236
cnvMode, 104
Code, 63
CodeFor, 179
Connect, 152
CONST, 39
CONTINUE, 32
ConvertDDF, 200
Copy, 140, 210
CopyBlob, 210
CopyFile, 215
CopyTblDef, 201
Count, 143
Create, 203
CreateCLRObject, 162
CreateComObject, 157
CreateEnum, 49
CreateMarkObj, 148
CreateObject, 149
CreateRSLObject, 161
CurrentLine, 231
CurToStrAlt, 175
Date, 14, 173
DateShift, 224
DateSplit, 173
DateTime, 14
DBF, 106
DbfFileRef, 15
DebugBreak, 199
Decimal, 14
Delete, 206
DelFile, 204
DetachSite, 160
DIALOG, 106
DisableFields, 188
DisableValidation, 188
Double, 14, 172
doubleL, 172
DoubleL, 14
DropTable, 204
DtTm, 174
DtTmSplit, 174
EnableFields, 189
EndAction, 224
ErrBox, 235
ErrPrint, 235
EvSource, 52
ExecExp, 227
ExecMacro, 225
ExecMacro2, 225
ExecMacroFile, 225
ExecMacroModule, 226
Execute, 160
ExistFile, 201

Exit, 230
Exp, 181
FALSE, 17
FDate, 143
Field, 71
FileName, 209
FillDown, 191
FillUp, 191
FindCol, 191
FindPath, 217
FindRow, 191
FldIndex, 209
FldName, 209
FldNumber, 209
FldOffset, 209
Floor, 173
FlushColumn, 169
FOR, 30
FTime, 143
GCollect, 227
GenAttach, 219
GenClassName, 219
GenGetProp, 220
GenNumProps, 222
GenObject, 219
GenPropID, 221
GenRun, 220
GenSetProp, 220
GetCallStack, 228
GetCurDir, 216
GetDate, 166
GetDirect, 206
GetDouble, 166
GetEnv, 228
GetEQ, 206
GetFileInfo, 218
GetGE, 206
GetGT, 206
GetIniFileValue, 218
GetInt, 166
GetLangId, 237
GetLE, 206
GetLocaleInfo, 237
GetLT, 206
GetMemAddrFrom, 228
GetMoney, 166
GetMultiCount, 192
GetNamedChanel, 150, 223
GetNumeric, 166
GetObjMethods, 220
GetObjProps, 220
GetParm, 180
GetPos, 206
GetPRNInfo, 171
GetRecordSize, 208
GetScrollFieldValue, 197
GetString, 166
GetStringR, 166
GetSysDir, 218
GetTime(, 166
GetTRUE, 166
GetUIMode, 228
GetUserName, 237
GetVar, 161
GetVarSize, 209
GoToScroll, 192
IF, 29
IMPORT, 26
Index, 71, 177
InitProgress, 223
Insert, 205
InstLoadModule, 228
Int, 172
Integer, 14
IsCopy, 143
IsDel, 143
IsEqClass, 221
IsGUI, 235
IsOutParm, 180
IsScrollEditMode, 198
IsSQL, 237
IsStandAlone, 234
IsWeakRef, 229
Item, 50
KEY, 105
KeyNum, 211
Line, 63
List, 142
LoadConfig, 152
local, 25
LockRcwHost, 150
Log, 181
Log10, 181
LoopInTrn, 213
MACRO, 40
MakeDir, 216
MakeDouble, 177
MakeMoney, 177
Max, 181
MEM, 106

MemAddr, 14
MemSize, 230
Menu, 184
MergeFile, 217
Message, 63, 168
MethodRef, 14
Min, 181
MkStr, 172
mod, 182
Module, 63
ModuleFileName, 236
ModuleName, 236
Money, 14, 172
MoneyL, 14
MonName, 176
MsgBox, 188
MsgBoxEx, 189
Name, 143
NeedFreeDB, 214
newLine, 72
Next, 204
NRecords, 209
NULL, 17
Numeric, 14
NumToStr, 176
Object, 16
OnError, 63, 127
Open, 201
Parmcount, 181
Pow, 181
Prev, 204
Print, 167
PrintFiles, 167
PrintGlobs, 167
PrintLn, 167
PrintLocs, 167
PrintModule, 168
PrintObject, 150
PrintProps, 168
PrintRefs, 168
PrintStack, 168
PrintSymModule, 168
private, 24
ProcessTrn, 212
ProcRef, 14
R2M, 222
raise, 54
Random, 231
rcwhost.d32, 158
ReadBlob, 214
ReadByte, 229
ReadMarkObj, 149
RemHandler, 54
Remove, 142
RemoveDir, 216
RemoveFile, 216
RemProgress, 223
RemTimer, 59
RenameFile, 215
ReplaceMacro, 226
Reset, 50
RETURN, 32
ReWind, 205
Round, 176
rsax.d32, 156
RsdCommand, 116
RsdConnection, 114
RsdEnvironment, 113
RsdError, 122
RsdField, 122
RsdParameter, 123
RsdRecordset, 119
RslDefCon, 114
RslDefEnv, 113
RslTimer, 58
RubToStr, 174
RubToStrAlt, 175
Run, 182
RunDialog, 187
RunError, 229
RunMenu, 185
RunScroll, 193
ScrollMes, 191
SelectFile, 213
SelectFolder, 214
SetAutoMoneyFloor, 176
SetBuff, 210
SetColumn, 169
SetDbfDir, 162
SetDbtDir, 162
SetDefFile, 162
SetDefMoneyPrec, 231
SetDefPrec, 170
SetDelim, 211
SetDlgFields, 191
SetExitFlag, 230
SetFocus, 188
SetHandler, 54
SetMacDir, 162

SetOutDir, 162
SetOutHandler, 170
SetOutput, 168
SetParm, 181
SetPRNInfo, 171
SetRecordAddr, 104, 207
SetScroll, 190
SetScrollFieldValue, 197
SetTimer, 58, 189
SetTxtDir, 162
SetVar, 161
ShowDictError, 232
ShowRSCOMError, 232
Size, 143
Sort, 142
SORT, 105
SpecVal, 14
SplitFile, 217
SplitMoney, 177
Sqrt, 181
StartProg, 232
Status, 211
Stop, 160
StrBrk, 177
StrFor, 179
String, 14, 172
StrIsNumber, 178
StrLen, 177
StrLwr, 179
StrongRef, 233
StrSet, 178
StrSplit, 178
StrSplit2, 178
StrSubst, 180
StrucRef, 15
StrUpr, 179
SubStr, 178
SysGetProperty, 233
SysPutProperty, 233
System, 233
TArray, 44
Tbfile, 98
TDbError, 128
TDirList, 140
TestEvent, 234
TestExist, 160
Time, 14, 173
TimeSplit, 174
ToANSI, 180
ToOEM, 180
TPattFieldR, 73
Trace, 199
TRcwHost, 159
TRcwSite, 153
TRecHandler, 102
TRepForm, 70
Trim, 178
TRsAxServer, 156
TRsComError, 153
TRslChanel, 151
TrslError, 63, 127
TRslEvHandler, 52
TRUE, 17
TStream, 131
TStreamDoc, 134
TXT, 106
TxtFileRef, 15
TypeLib, 53
UnderRCWHost, 237
UnlockRcwHost, 151
UnPackVarBuff, 208
Update, 205
UpdateFields, 188, 197
UpdateScroll, 197
UseProgress, 223
UserFill, 191
UserNumber, 231
ValType, 12, 171
Value, 50, 71
VAR, 34
Version, 230
ViewFile, 204
WeakRef, 38
WHILE, 30
Write, 71
WRITE, 105
WriteBlob, 214
WriteByte, 229
writeFields, 72
ZeroValue, 238

Наши программные продукты можно приобрести в Москве:

R-Style Softlab

127549 Москва, ул. Пришвина, 8

Связаться с нами можно по телефонам и электронной почте:

Служба поддержки

т: (495) 796-9311; E-mail: support@softlab.ru

Отдел продаж

т: (495) 796-9310; E-mail: sales@softlab.ru

http://www.softlab.ru/

А также в филиалах:

Брянск т. (4832) 52-4778, 52-4779, ф. (4832) 52-4532;
E-mail: softlab@moon.r-style.ru

Екатеринбург т/ф: (343) 261-3044, 261-6086, (343) 261-6819;
E-mail: obt@ural.r-style.ru

Киев т. (+38 044) 248-8982, 245-0519,
496-3455, факс. (+38 044) 248-8054;
E-mail: sales@rstyle.kiev.ua

Нижний Новгород т. (8312) 44-3517, 44-1622, т/ф. (8312) 44-4328;
E-mail: office@office.r-style.sci-nnov.ru

Новосибирск т. (3832) 266-0197, 266-9001, т/ф. (3832) 266-9508;
E-mail: welcome@sib.r-style.ru

Ростов-на-Дону т. (8632) 252-4813, ф. (8632) 290-8360;
E-mail: support@r-style.donpac.ru

Санкт-Петербург т. (812) 329-3682, т/ф. (812) 329-3686;
E-mail: soft.sale@r-style.spb.ru

Хабаровск т.(4212) 31-5200, ф. (4212) 31-2228;
E-mail: root@fe.r-style.ru