ВЫСОКИЕ ТЕХНОЛОГИИ
КАЛЛИСТО

# Введение в ИСР Cool Manager

**Июль 2007**

# Оглавление

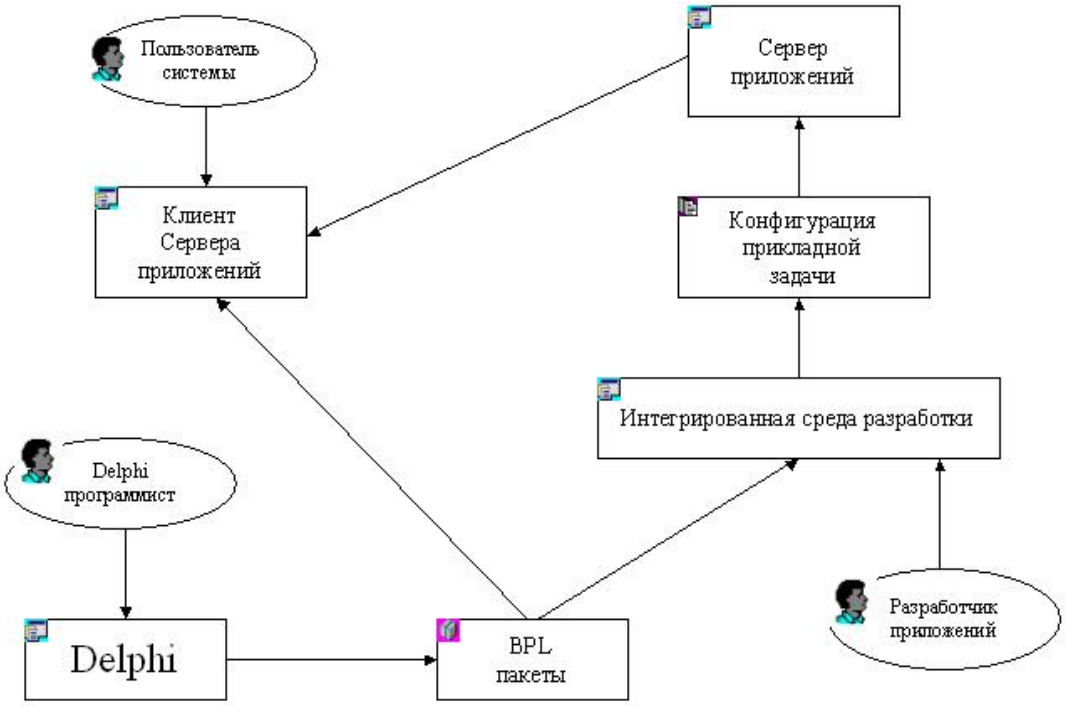
# Глава I

## Описание системы

# 1 Описание системы

## 1.1 Введение

Cool Manager - это инструмент, предназначенный для создания систем автоматизации учета. Cool Manager целиком написан на Delphi, даже интегрированная среда разработки очень похожа на IDE Delphi. Поэтому, программисты имеющие опыт работы с Delphi или Borland C++ Builder могут начать работу с системой практически сразу, без какого либо предварительного обучения.
Может возникнуть вопрос - зачем изобретать велосипед, если уже есть Delphi? Ответ очень прост - Cool Manager проблемно-ориентированная среда, предназначенная для создания СИСТЕМ АВТОМАТИЗАЦИИ УЧЕТА. Delphi таковой не является по определению. Разработка системы автоматизации на Delphi с нуля требует огромных трудозатрат и наличие команды высокопрофессиональных разработчиков. Cool Manager - это как раз тот инструмент, который позволит Вам достичь желаемого результата в реальные сроки, используя при этом всю мощь возможностей Delphi.




Логическая схема системы Cool Manager.

**Список логических модулей изображенных на схеме и принятых для них сокращений.**

Delphi IDE Delphi как инструментарий для разработки ПО.

| | |
|---|---|
| Пакеты | Стандартные BPL пакеты Delphi содержащие исходный код визуальных и невизуальных компонент, используемых дизайнером форм IDE Delphi. Вообще-то информация содержащаяся в пакетах имеет более широкое применение, этот вопрос будет рассмотрен более подробно в Руководстве программиста. |
| ИСР | Интегрированная среда разработки. Приложение используемое для создания конфигурации прикладной задачи. |
| КЗ | Конфигурация прикладной задачи. Совокупность модулей конфигурации, созданных с помощью ИСР. |
| AppServer | Сервер приложений. Используется для доступа к конфигурации с рабочей станции пользователя. Установка соединения с сервером производится по TCP/IP протоколу. |
| AppClient | Запускается на рабочем месте пользователя системы. Приложение используется для установки соединения с AppServer и выполнения запрошенной конфигурации. |

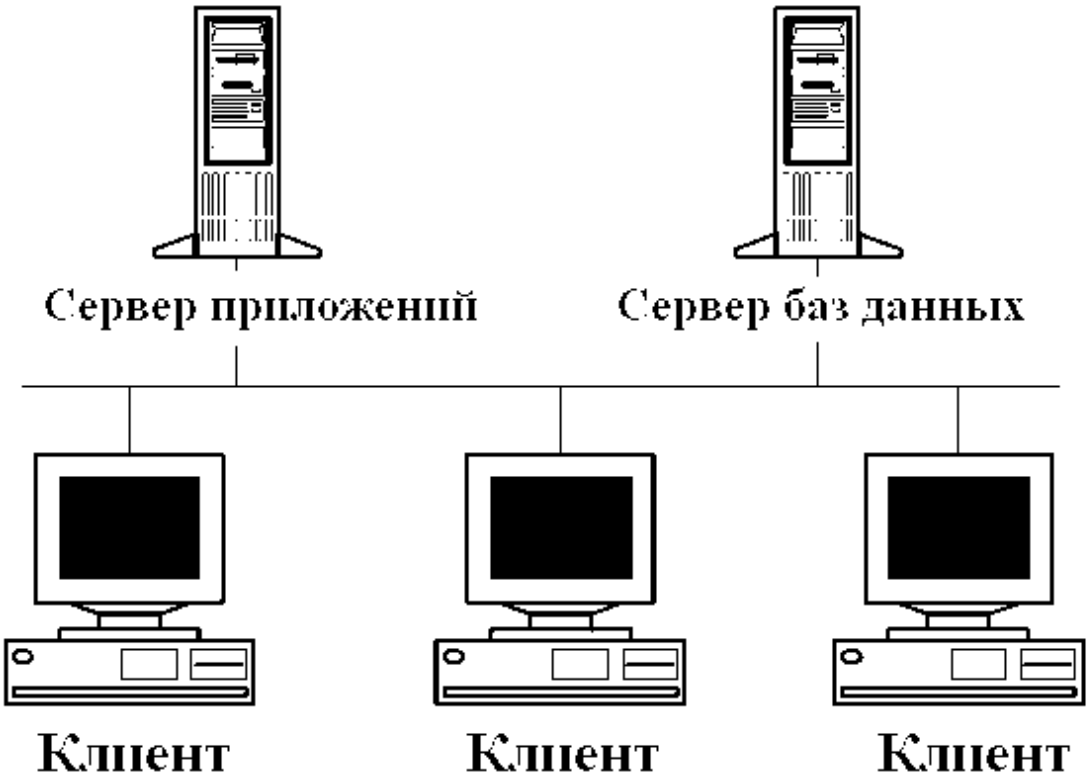
Ядром системы является интегрированная среда разработки (ИСР). С помощью ИСР можно создать полностью законченное пользовательское приложение. ИСР содержит дизайнер форм, Pascal ориентированный скриптовый язык, мощный редактор скриптов, конфигуратор метаданных и многое другое. Главной особенностью ИСР является полностью открытая архитектура. Можно загружать в ИСР стандартные VCL пакеты IDE Delphi, тем самым расширяя инструментарий используемый при конструировании визуальных форм (загруженные компоненты отражаются в палитре компонентов, так же как в Delphi). Кроме того, можно создавать и загружать пользовательские пакеты, содержащие проблемно-ориентированные классы для построения систем автоматизации. С помощью таких пакетов, можно подключить к скриптовому языку новые типы, классы, процедуры и функции. Пакет так же может содержать описание новых классов метаданных, которые становятся доступными в конфигураторе ИСР. Работая с Cool Manager, разработчик может использовать всю мощь и многообразие возможностей заложенных в Delphi, создавать любой Delphi код, помещать его в пакеты, загружать созданные пакеты в ИСР, тем самым расширяя её функциональность до любого требуемого уровня.
Совокупность загруженных в ИСР пакетов и созданных с её помощью модулей образуют конфигурацию прикладной задачи (КЗ). КЗ хранится на отдельном компьютере, конечные пользователи получают доступ к ней посредством сервера приложений. Для этого, на рабочей станции, запускается специальная программа - просмотрщик, которая устанавливает соединение с сервером приложений. После того, как пользователь проходит авторизацию на сервере и указывает с какой конфигурацией собирается работать, сервер приложений начинает передавать на рабочую станцию модули конфигурации, к которым обращается пользователь в процессе работы. Установка соединения с сервером приложений производится посредствой TCP/IP соединения, поэтому не важно где физически находится сервер приложений, на той же рабочей станции, в локальной сети или на компьютере расположенном в другом городе.

См. далее: Архитектура приложений

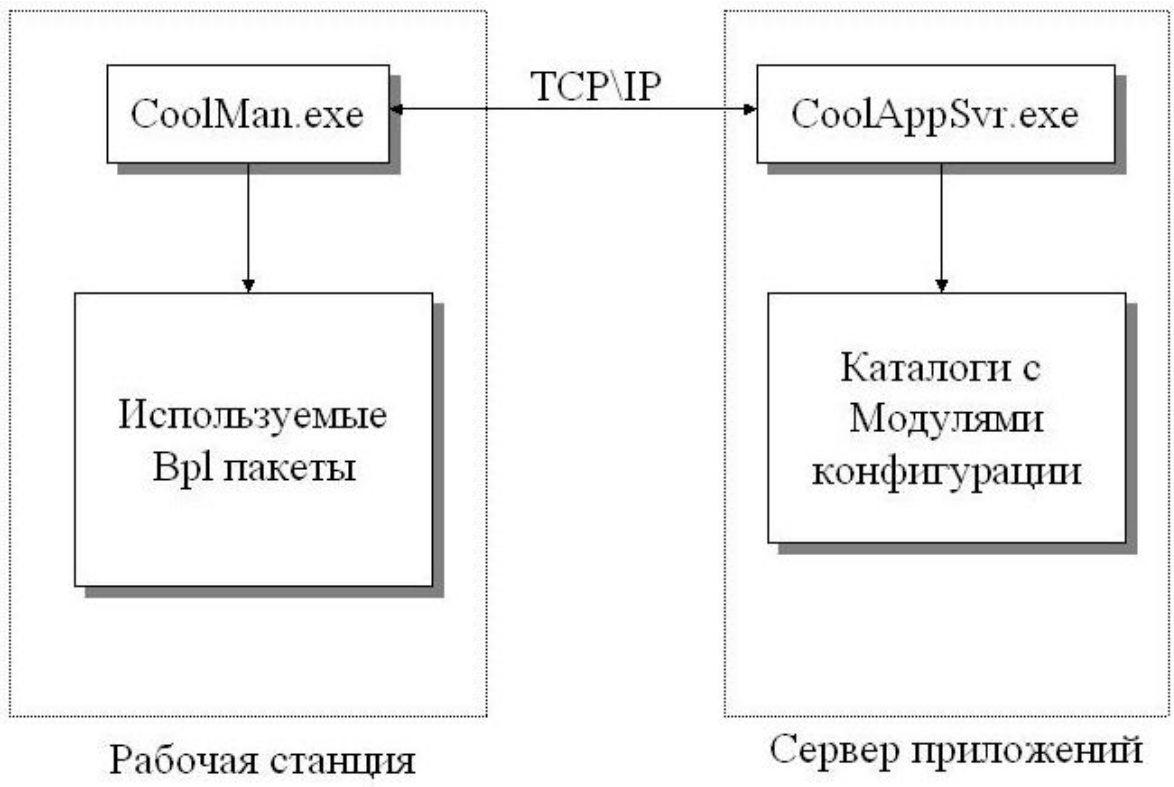
## 1.2 Архитектура приложений

### Основные компоненты приложения:

- **AppClient** - клиентская часть системы, реализующая графический пользовательский интерфейс к функциям и данным системы. В свою очередь, состоит из набора динамически загружаемых bpl пакетов и исполняемого модуля CoolMan.exe
- **Конфигурация** - библиотека приложения, содержит набор модулей конфигурации реализующих бизнес логику системы.
- **База данных** - реляционная СУБД под управлением InterBase, либо одного из его клонов (Yaffil, FireBird).
- **Сервер приложений** - позволяет клиенту получить доступ к модулям конфигурации. Исполняемый модуль сервера приложений называется CoolAppSrv.exe



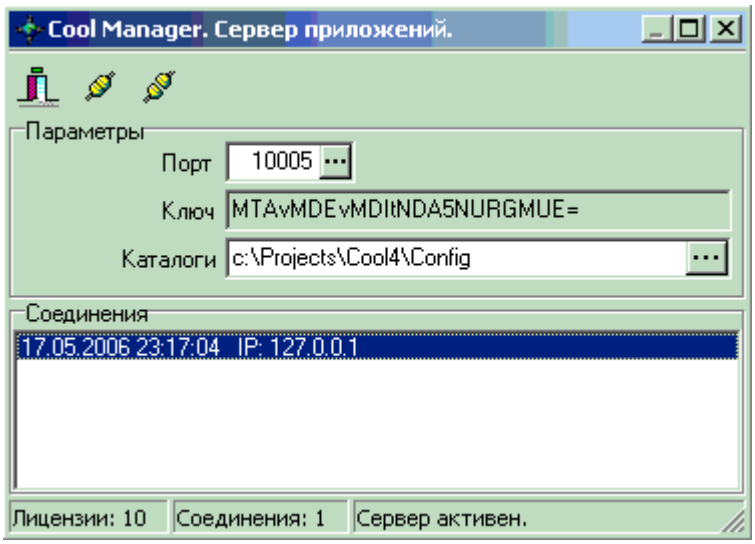

Взаимодействие клиента с сервером приложений:




На рабочей станции запускается клиент CoolMan.exe. Клиент устанавливает TCP соединение с сервером приложений, далее, когда клиенту требуется какой-либо файл конфигурации, CoolMan.exe обращается к серверу приложений и передает ему имя искомого файла. Сервер приложений производит поиск нужного файла и если находит, то возвращает его. Т.к. соединение с сервером производится по TCP протоколу, сервер приложений может находиться на том же компьютере с которого запускается программа просмотрщик, на другом компьютере в локальной сети или вообще на другом сервере доступном через InterNet. Принцип действия сервера приложений напоминает работу стандартного Web приложения: сперва браузер отправляет Web серверу запрос, тот в ответ возвращает сформированную HTML страницу, получив её браузер преобразует текстовое описание в визуальную форму. Только в отличии от браузера, просмотрщик CoolMan.exe отображает не HTML страницу, а полноценное Windows приложение.

Сервер приложений выполняет ещё одну важную функцию - является сервером защиты. Если сервер приложений не зарегистрован, приложение выполняется в демо режиме. В демо режиме максимально допустимое время работы с сервером две минуты, после этого соединение принудительно разрывается. При регистрации сервера приложений генерируется ключевой файл инициализации. Ключевой файл привязывается к компьютеру, на котором запускается сервер приложений. Когда число соединений превышает число рабочих лицензий, все "сверхнормативные" соединения обслуживаются в демо режиме.

## Сервер приложений.




На панели управления расположены три кнопки:

 - выход

 - активировать сервер

 - остановить сервер

- **Порт** - номер порта сервера.
- **Ключ** - ключ активации, высылается разработчику, на его основе генерируется файл инициализации.
- **Каталоги** - список каталогов, в которых расположены файлы конфигураций.

**Список соединений** - отражает все текущие подключения к серверу, для каждого соединения указывается дата и время установки соединения, а так же IP адрес рабочей станции.

В строке статуса отображаются:
- **Лицензии** - показывает число лицензий, приобретенных для данного сервера.
- **Соединения** - текущее число соединений.
- Выводится статус сервера, возможны два варианта: "Сервер активен" и "Сервер остановлен".

## Рабочая станция.

Рабочая станция использует приложение CoolManX.exe (под буквой X подразумевается цифра обозначающая версию Delph). На рабочей станции должны находиться:
- исполняемый файл CoolManX.exe
- BPL пакеты необходимые для запуска приложения
- Файл comps.pal

Всё это должно располагаться в одном каталоге на диске.

При запуске CoolManX.exe, в командной строке, должны быть указаны три параметра:

- TCP/IP адрес компьютера на котором установлен сервер приложений, либо имя компьютера в сети.
- Порт через который следует устанавливать соединение
- Имя файла запускаемой конфигурации

**Пример:**
Порт сервера приложений: 10005
Имя файла конфигурации: prodtorg.cm4

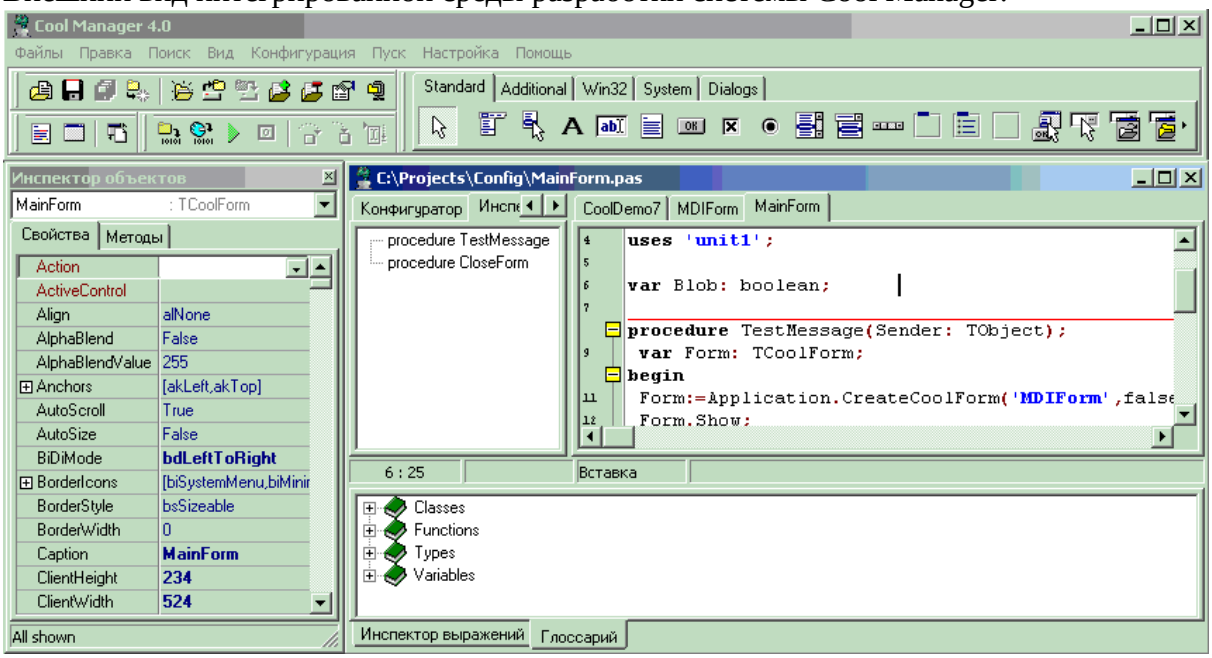
Сервер приложений запущен на рабочей станции:
**CoolMan7 localhost 10005 prodtorg.cm4**
**или**
**CoolMan7 127.0.0.1 10005 prodtorg.cm4**

Сервер приложений запущен на компьютере Server (IP адрес 192.168.0.1):
**CoolMan7 server 10005 prodtorg.cm4**
**или**
**CoolMan7 192.168.0.1 10005 prodtorg.cm4**

См. далее: Интегрированная среда разработки.

## 1.3 Интегрированная среда разработки.

Внешний вид интегрированной среды разработки системы Cool Manager:




Одной из главных причин создания интегрированных сред разработки (ИСР) является желание вынести бизнес-логику работы приложения из скомпилированных .exe файлов на более высокий уровень. Чем шире возможности встроенной ИСР, тем более гибкой и масштабируемой будет система в целом. Каждая конкретная система автоматизации

решает вопрос о реализации ИСР по своему, от создания простеньких редакторов отчетов и форм ввода данных, до полностью логически завершенных сред разработки бизнес-приложений.
Как правило, ИСР содержит следующие инструменты:
- встроенный скриптовый язык. Используется для написания алгоритмов бизнес логики приложения.
- редактор внешних форм. Используется для создания форм с помощью которых пользователь осуществляет ввод данных в систему.
- конструктор отчетов. Позволяет создавать отчеты. Обычно использует редактор внешних форм, для создания диалогов ввода начальных параметров и скриптовый язык для описания алгоритмов формирования отчета.
Фактически, ИСР может содержать и другие, более мощные инструменты для создания бизнес-приложений. Пример: конфигуратор метаданных системы 1С. Конфигуратор позволяет представить все метаданные, используемые приложением, в виде древовидной структуры с разбивкой на логические группы, в зависимости от типа объектов и их назначения.
Очень важным инструментом любой ИСР является скриптовый язык. Основная задача скриптового языка - предоставление доступа к метаданным бизнес-приложения в простом и удобном для разработчика виде. Кроме того, скриптовый язык должен изолировать разработчика от программирования рутинных операций доступа к файлам, базам данных и т.д. Например, если в языке определен объект "Справочник", разработчик должен иметь к нему доступ посредством простых команд: Найти, Удалить, Добавить и т.д. не заботясь о том в каком виде данные справочника хранятся в базе, какой сервер используется и какие операции требуется выполнить для извлечения данных из базы. Вся эта работа должна происходить "за кадром" и не требовать написания дополнительного программного кода.

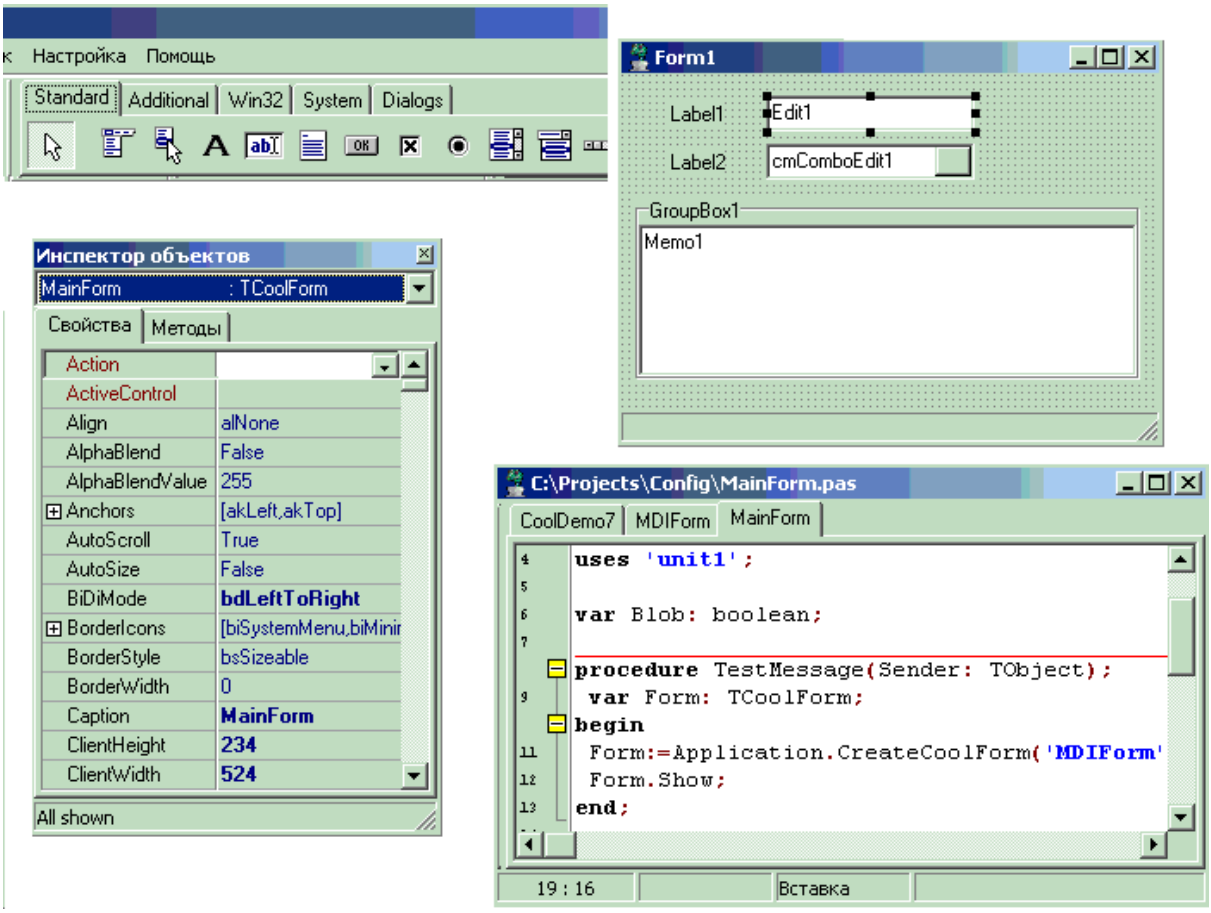
**Возможности ИСР системы CoolManager.**

- Динамическая загрузка BPL пакетов созданных в Delphi.
- Палитра компонентов аналогичная палитре IDE Delphi, возможность загрузки в ИСР стандартных VCL пакетов Delphi.
- Дизайнер форм аналогичный используемому в IDE Delphi + связанный с дизайнером редактор скриптов.
- Скриптовый язык с мультиязыковой поддрежкой синтаксиса. Можно писать скрипты на Pascal, C++, JScript, Visual Basic
- Открытая структура скриптового языка. Загружаемые в ИСР пакеты могут добавлять в язык новые типы, классы, процедуры, функции и переменные.
- Возможность вызова из скриптов стандартных Delphi форм, находящихся в подгружаемых BPL пакетах.
- Каждая форма, созданная в ИСР имеет связанный с ней скрипт. Процедуры и функции, описанные в скрипте, могут быть определены в качестве обработчиков событий для формы и расположенных на ней компонентов.
- Встроенный конфигуратор метаданных. Объекты метаданных в конфигураторе представлены в виде древовидной структуры.
- Конфигуратор метаданных имеет открытую архитектуру, загружаемые в ИСР пакеты могут содержать описание новых классов метаданных.

- Встроенный в ИСР отладчик скриптов, позволяет запускать конфигурацию в режиме отладки, с возможностью трассировки текста скриптов и просмотра текущего значения переменных и выражений.
- Конфигурация может быть разбита на независимые логические блоки, которые сохраняются в отдельных файлах. Это позволяет использовать одни и те же блоки в различных конфигурациях.

См. далее: Конструктор форм

## 1.4 Конструктор форм

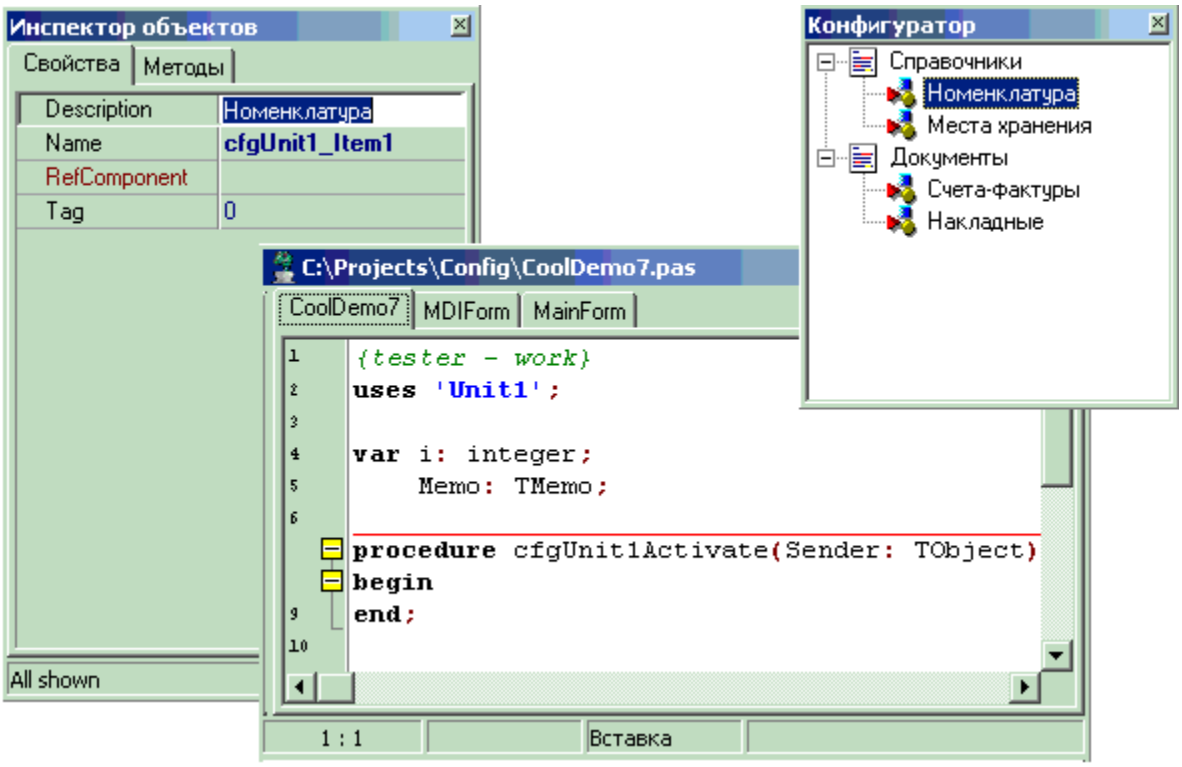



ИСР имеет встроенный конструктор форм, аналогичный дизайнеру форм IDE Delphi. Палитра компонентов содержит набор визуальных и невизуальных компонент, которые можно помещать на редактируемые формы. Инспектор объектов позволяет редактировать опубликованные свойства форм и компонент. Каждая форма состоит из двух файлов: файла формы (.dfm) и файла скрипта связанного с формой (.pas, .cpp, .js, .bas), оба файла должны иметь одинаковые имена. Все создаваемые формы наследуются от класса TCoolForm, это базовый класс для создания всех форм в Cool Manager, так же как TForm в Delphi. При создании новой формы в ИСР не происходит создание нового класса наследованного от TCoolForm, т.к. скриптовый язык ИСР не поддерживает объявления классов. Зато функции и процедуры, объявленные в скрипте, можно использовать в качестве обработчиков событий форм и компонентов. Для этого

используется закладка **Методы** инспектора объектов. Внутри скрипта возможно обращение к компонентам формы по имени, так же как в Delphi. Файл формы может использовать объявления переменных, процедур и функций объявленных в других скриптах, для этого соответствующий скрипт должен быть указан в операторе uses.

См. далее: Использование конфигуратора

## 1.5 Использование конфигуратора



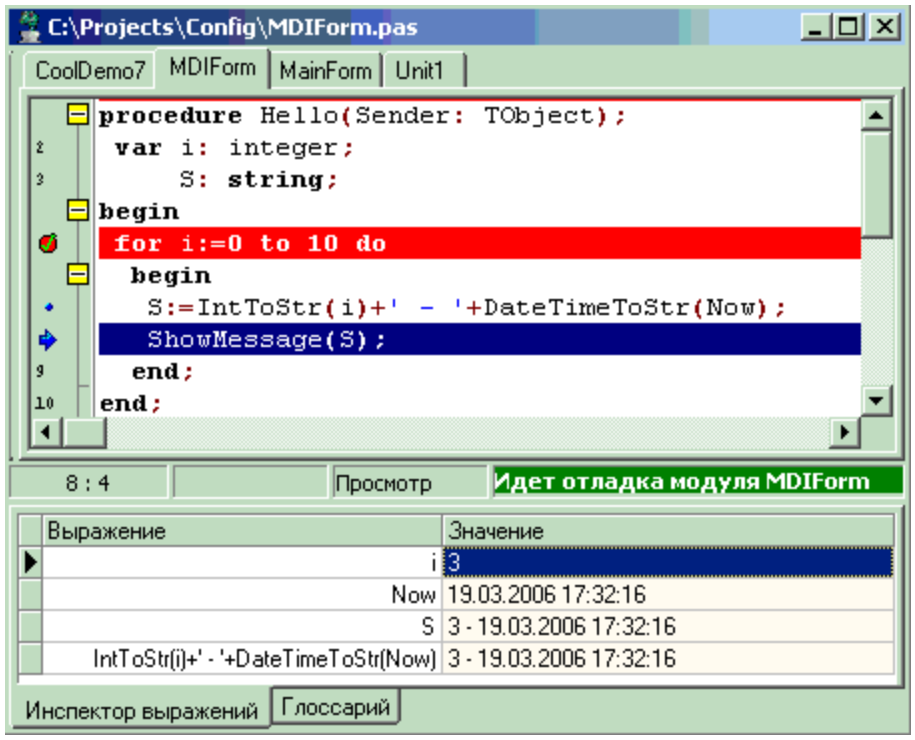

Delphi обладает массой достоинств и содержит множество оригинальных решений, но... к сожалению, в Delphi нет удобного инструмента для визуального построения структур метаданных. Если не совсем понятно, что имеется в виду, вспомните конфигуратор системы 1С предприятие. Самый близкий инструмент по смыслу, который можно использовать для этих целей в Delphi - это модули данных. В Delphi модуль данных - это невизуальный контейнер, в котором традиционно размещаются компоненты доступа к базам данных (а так же при желании, любые другие невизуальные компоненты), кроме того, в модуле данных часто размещают обработчики событий связанных с компонентами модуля. Но модули данных обладают низкой наглядностью, если в контейнере находится 10-15 компонент, то особых сложностей не возникнет, а вот попробуйте поместить в него пару сотен объектов...
В случае работы с метаданными прикладной задачи, просто необходим инструмент, позволяющий представлять всю структуру метаданных в виде логически согласованной древовидной структуры, с возможностью интерактивного редактирования как самих метаданных, так и их свойств. Самое интересное, что создать подобный инструмент средствами Delphi своем не сложно, сама логика построения классов наследованных от TComponent подразумевает создание древовидных структур. Главное, оформить всё

это в удобном для конечного пользователя виде.
ИРС CoolManager содержит встроенный конфигуратор метаданных, позволяющий создавать древовидные структуры. Для редактирования свойств объектов метаданных используется Object Inspector, кроме того, с помощью редактора скриптов можно создавать обработчики событий для метаобъектов конфигурации. Базовым классом для создания метаклассов является:
**TcmUnitComponent = class(TComponent)**
Конфигуратор имеет открытую архитектуру, можно создавать новые классы метаданных, помещать их в пакеты и после загрузки в ИСР использовать при построении дерева конфигурации. Для доступа к конфигурации из скриптового языка определена переменная:
**var CoolConfig: TCoolConfig;**

Конфигурация в Cool Manager играет ту же роль, что и приложение в Delphi. Когда мы создаем новое приложение в ИСР Cool Manager, мы создаем новую конфигурацию. Конфигурация состоит из набора модулей, которые содержат скрипты, описание объектов конфигурации, формы и много другое. По умолчанию файл конфигурации имеет расширение .cm4.

См. далее: Отладка скриптов

## 1.6 Отладка скриптов



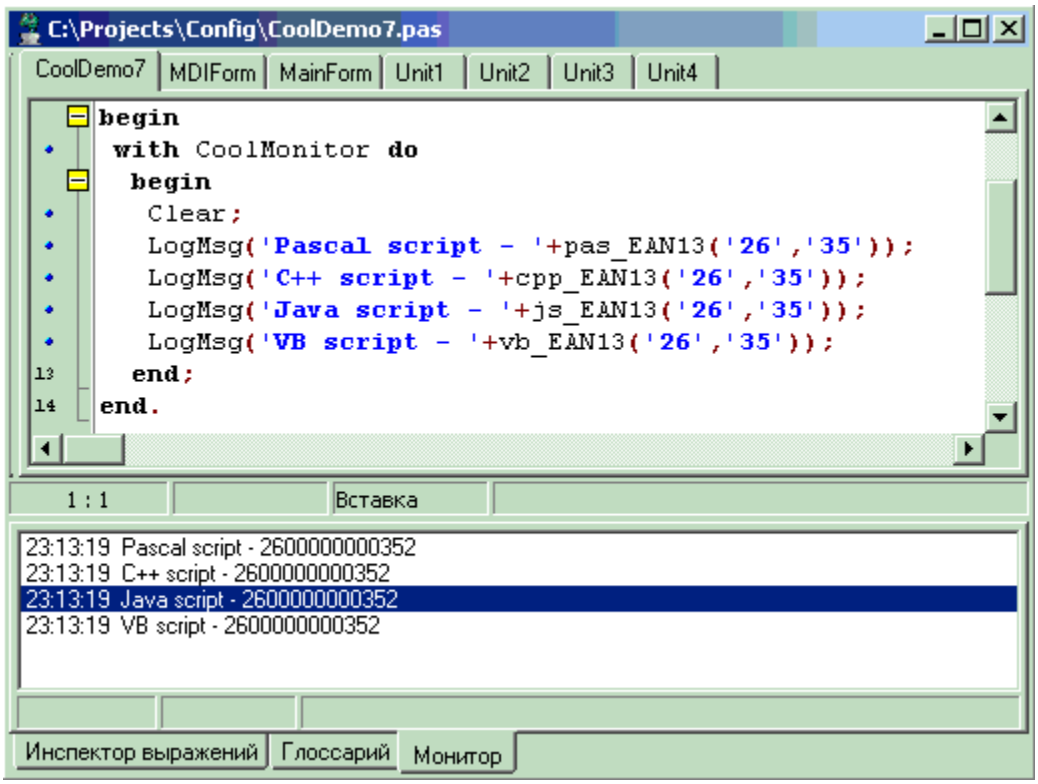

ИСР Cool Manager содержит встроенный отладчик скриптов. Вы можете запустить приложение на выполнение из ИРС, при этом все скрипты будут выполняться в режиме отладки. Это означает, что в тексте скриптов можно устанавливать точки прерывания, выполнять скрипты в пошаговом режиме, просматривать текущие значения

переменных и выражений при помощи инспектора выражений.

Существует три команды выполнения скриптов в пошаговом режиме:
- **Шаг через (F8)**
- **Шаг внутрь (F7)**
- **Идти до курсора (F4)**

См. далее: Монитор сообщений

## 1.7 Монитор сообщений




Монитор сообщений является встроенным инструментом ИСР и предназначен для вывода сообщений в процессе выполнения скриптов. Для доступа к монитору сообщений, используется глобальная переменная скриптов **CoolMonitor**. Монитор может быть полезен на этапе тестирования и отладки. Так же монитор может использоваться для вывода информации при выполнении длительных скриптовых обработок, если обработка выполняется непосредственно из ИСР. Для выполнения некоторых служебных процедур можно создать специальную конфигурацию и запускать её в рамках ИСР. Такая конфигурация не содержит визуальных форм и просто выполняет набор заданных пользователем инструкций. Для вывода сообщений о ходе работы можно использовать монитор сообщений. Для дополнительной информации о командах, поддерживаемых монитором, найдите в глоссарии класс **TCoolMonitor** и нажмите клавишу <**Enter**>.

См. также: Скриптовый язык.

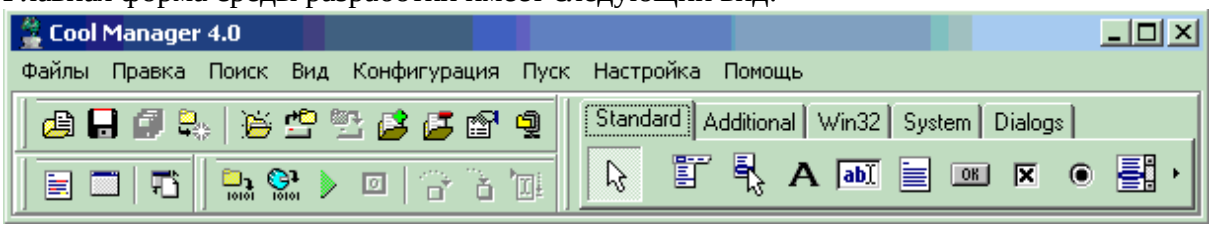
# Глава II

## Среда разработки

# 2 Среда разработки

## 2.1 ИСР Cool Manager

Главная форма среды разработки имеет следующий вид:




Рабочая область, под главным меню приложения, поделена на две части:

- панель инструментов - содержит кнопки для вызова наиболее часто используемых команд главного меню.
- палитра компонентов - содержит набор компонентов используемых конструктором форм.

**Разделы главного меню приложения:**

| | |
|---|---|
| Файлы | Команды для работы с файлами приложения. |
| Правка | Команды редактирования скриптов и форм. |
| Поиск | Команды поиска текста. Можно искать текст в редакторе скриптов, глоссарии и конфигураторе. |
| Вид | Вызов дополнительных инструментов, используемых средой разработки. |
| Конфигурация | Команды для работы с текущей конфигурацией. |
| Пуск | Команды выполнения и отладки текущей конфигурации. |
| Настройка | Команды настройки среды разработки. |
| Помощь | Вызов системы помощи. Работа с сервисом CoolManager Direct. |

## 2.2 Главное меню. Файлы.

Раздел Файлы предназначен для работы с модулями конфигурации.

| | |
|---|---|
| **Открыть** | Загрузить модуль с диска. |
| **Сохранить** | Сохранить изменения, внесенные в активный модуль. |
| **Сохранить как** | Сохранить модуль в другом файле на диске. |
| **Сохранить всё** | Сохранить все сделанные изменения. |
| **Закрыть всё** | Закрыть все редактируемые модули. |
| **Просмотр** | Вызывает диалог для просмотра текста активного скрипта. Текст скрипта показывается в том виде, как он будет выведен на печать, с разбиением на страницы, отступами от края листа и т.д. |
| **Печать** | Печать активного скрипта без предварительного просмотра. |
| **Новая форма** | Создать новую форму. |
| **Новый модуль** | Создать новый скрипт. |

## 2.3 Главное меню. Правка.

Раздел Правка содержит команды, используемые при редактировании скриптов и форм.

**Отменить** Позволяет отменить последнее сделанное изменение. (только для редактора скриптов)

**Вернуть** Позволяет восстановить последнее отмененное изменение. (только для редактора скриптов).

**Копировать** Позволяет скопировать в буфер выделенный фрагмент текста в редакторе скриптов или выделенные компоненты на форме.

**Вырезать** Позволяет вырезать и вставить в буфер выделенный фрагмент текста в редакторе скриптов или выделенные компоненты на форме.

**Вставить** Позволяет вставить из буфера скопированный ранее фрагмент текста в редактор скриптов или набор компонентов на форму.

**Удалить** Удалить выделенный фрагмент текста в редакторе скриптов или выделенные компоненты на форме.

**Выделить всё** Позволяет выделить весь текст в редакторе скриптов или все компоненты на форме.

*Приведенные ниже команды используются только при работе с конструктором форм.*

**Выравнивание по сетке** Выравнивает расположенные на форме компоненты по сетке.

**Переместить вперед** Команда помещает выделенный компонент сверху всех остальных (в верх Z-последовательности, но с учетом того, что все не оконные компоненты располагаются позади оконных). Это полезно, если вы разместили на форме ряд компонентов и они накладываются друг на друга. В этом случае вы можете решить, что какой-то скрытый компонент, лежащий под другими, должен быть наверху. Для этого надо выделить этот компонент и затем выполнить данную команду меню.

**Переместить назад** Эта команда противоположна по смыслу команде **Переместить вперед**. Она помещает выбранный компонент или компоненты позади всех других компонентов (в низ Z-последовательности, но с учетом того, что все не оконные компоненты располагаются позади оконных).

**Выравнивание** При выборе этого пункта меню появляется диалоговое окно Alignment (Выравнивание). Опции этого окна позволяют выбрать ряд вариантов выравнивания компонентов на форме по горизонтали и вертикали.

**Размер** Выбор этого пункта меню позволяет изменять размеры компонента до заданных значений ширины и высоты. Если вы выделили несколько компонентов, данный пункт позволяет увеличить размеры всех этих компонентов по горизонтали, вертикали или в обоих направлениях до размеров наибольшего из выделенных компонентов на странице или сократить их размеры до размера наименьшего из них.

**Масштабирование** Используя команду **Масштабирование**, вы можете пропорционально изменить масштаб всего, расположенного на форме. Все размеры можно увеличивать или уменьшать вплоть до ста раз. В появляющемся диалоговом окне вам надо задать **Масштаб** - масштабирующий коэффициент в %. Например, задав 200 вы увеличите все компоненты в 2 раза.

**Порядок переключения** Позволяет установить последовательность смены активных компонентов, расположенных на форме, при нажатии пользователем клавиши табуляции Tab. Команда **Порядок переключения** высвечивает на экране диалоговое окно редактирования последовательности табуляции. В окне помещен список имен всех компонентов, размещенных на форме. Вы можете изменить их последовательность, выделяя соответствующий элемент и нажимая кнопки со стрелками "вверх" или "вниз". Это удобнее, чем устанавливать свойства каждого компонента вручную.

**Порядок создания** Эта команда позволяет управлять последовательностью, в которой создаются невизуальные компоненты. Эта последовательность может быть важна, если одни из этих компонентов используют свойства других, полагая, что те существуют и инициализированы. Если эти компоненты не создаются в правильной последовательности, то обращение к несуществующему компоненту вызовет генерацию исключения.

**Повернуть компоненты** Раздел позволяет зеркально преобразовать размещение (справа налево) компонентов формы, выбрав из подменю один из двух вариантов: все компоненты формы, дочерние компоненты выделенного оконного элемента. Чаще всего это используется при разработке вариантов приложений, предназначенных для стран Востока.

**Зафиксировать** Зафиксировать компоненты расположенные на форме. Запрещает изменять размер и положение визуальных компонент.

## 2.4 Главное меню. Поиск.

Раздел Поиск используется для поиска заданного текста. Искать можно в редакторе скриптов, глоссарии и конфигураторе.

**Найти** Вызывает диалог для установки параметров поиска. Искать можно в редакторе скриптов, в конфигураторе и глоссарии.

**Искать далее** Ищет следующее вхождение текста согласно установок, сделанных при помощи диалога **Найти**.

*Приведенные ниже команды используются только в редакторе скриптов.*

**Быстрый поиск** Выбрав этот пункт меню начните печатать какое-нибудь слово. Редактор скриптов найдет в тексте первое вхождение печатаемой вами последовательности символов. Это прекрасный инструмент в тех случаях, когда вы только приблизительно знаете, что именно хотите найти.

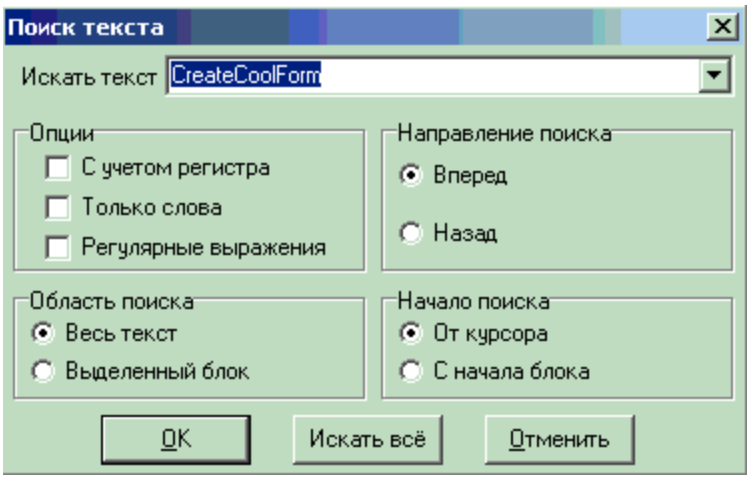
**Заменить** Вызывает диалог поиска и замены текста.

**К следующему слову** Определяет текущее слово под курсором и ищет его вниз по тексту.

**К предыдущему слову** Определяет текущее слово под курсором и ищет его вверх по тексту.

## 2.4.1 Диалоги поиска и замены текста

### 2.4.1.1 Поиск текста в редакторе скриптов




**Искать текст** - искомая последовательность символов.

**Опции**:

- **С учетом регистра** - если включена, при поиске учитывается регистр букв.
- **Только слова** - если влючена, поиск заданной последовательности только как целого слова.
- **Регулярные выражения** - если включена, заданная последовательность интерпритируется как регулярное выражение.

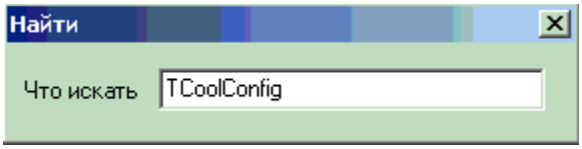
**Направление поиска:**

- **Вперед** - поиск от текущей позиции курсора вниз.
- **Назад** - поиск от текущей позиции курсора вверх.

**Область поиска:**
- **Весь текст** - глобальный поиск по всему тексту.
- **Выделенный блок** - поиск только в выделенном блоке.

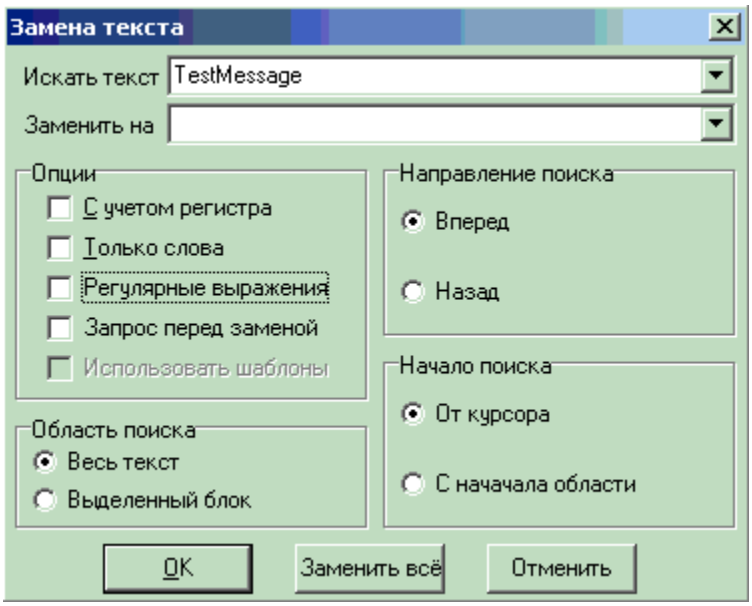
**Начало поиска:**
- **От курсора** - начинать поиск от текущей позиции курсора.
- **С начала блока** - начинать поиск от начала выделенного блока.

#### 2.4.1.2 Поиск текста в глоссарии и конфигураторе




Диалог для ввода искомого текста. Используется для поиска в конфигураторе и глоссарии. Поиск происходит от текущей позиции далее по дереву объектов. Прописные и строчные символы игнорируются.

#### 2.4.1.3 Поиск и замена текста




**Искать текст** - искомая последовательность символов.
**Заменить на** - последовательность символов, которой следует заменить найденную.

**Опции**:
- **С учетом регистра** - если включена, при поиске учитывается регистр букв.
- **Только слова** - если влючена, поиск заданной последовательности только как целого слова.
- **Регулярные выражения** - если включена, заданная последовательность интерпритируется как регулярное выражение.

- **Запрос перед заменой** - перед заменой выводить диалоговое окно с запросом.

**Направление поиска:**
- **Вперед** - поиск от текущей позиции курсора вниз.
- **Назад** - поиск от текущей позиции курсора вверх.

**Область поиска:**
- **Весь текст** - глобальный поиск по всему тексту.
- **Выделенный блок** - поиск только в выделенном блоке.

**Начало поиска:**
- **От курсора** - начинать поиск от текущей позиции курсора.
- **С начала блока** - начинать поиск от начала выделенного блока.

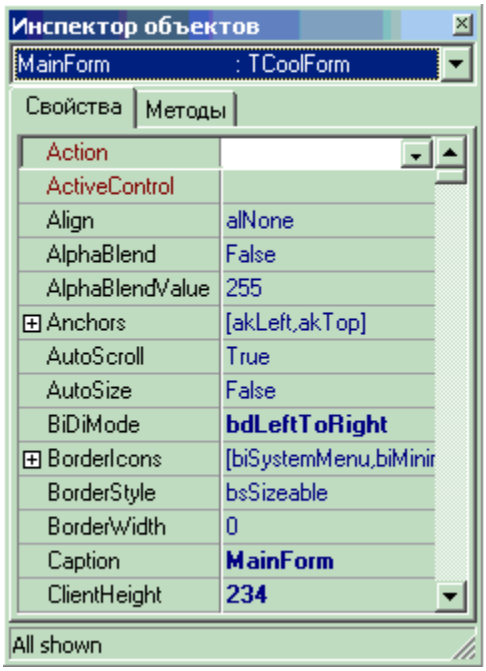
## 2.5 Главное меню. Вид.

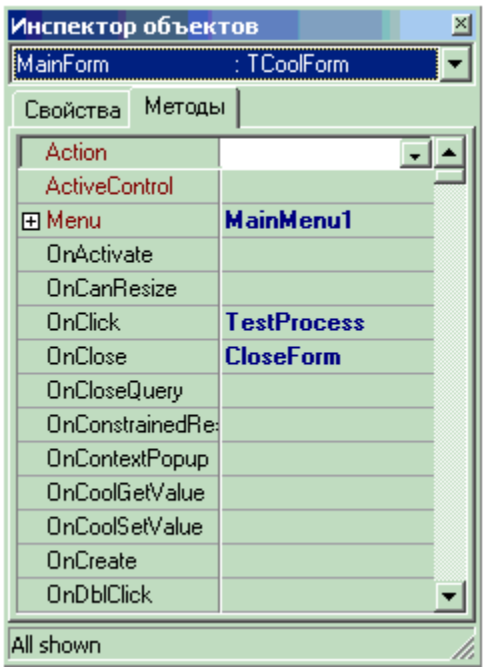
Раздел Вид предназначен для вызова инструментальных средств, используемых в процессе создания конфигурации.

| | |
|---|---|
| **Инспектор объектов** | Вызывает инспектор объектов предназначенный для редактирования свойств форм и их компонентов, а так же компонентов включенных в дерево конфигурации. |
| **Палитра выравнивания** | Вызывает палитру выравнивания.Опции этого окна позволяют выбрать ряд вариантов выравнивания компонентов на форме по горизонтали и вертикали |
| **Редактор скриптов** | Вызывает форму редактора скриптов. |
| **Инспектор скриптов** | Вызывает форму инспектора скриптов. |
| **Закрыть закладку** | Позволяет закрыть текущую закладку в редакторе скриптов. При этом скрипт и связанная с ним форма (если редактируется скрипт формы) удаляются из среды разработки. |
| **Переключить форму/скрипт** | Производит последовательное переключение между активной формой и её скриптом. |
| **Конфигуратор** | Вызывает конфигуратор. |
| **Глоссарий** | Вызывает глоссарий. |
| **Конструктор дистрибутивов** | Вызывает конструктор дистрибутивов. |

## 2.6 Инструменты среды разработки

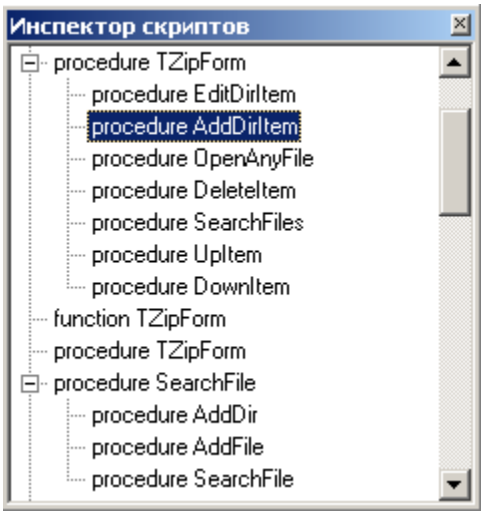
### 2.6.1 Инспектор объектов




Инспектор объектов позволяет редактировать опубликованные свойства и события компонент. Инспектор используется совместно с конструктором форм, а так же для редактирования компонентов конфигурации. Закладка "Методы" инспектора объектов тесно связана и радактором скриптов. Так двойной щелчок по полю имени соответствующего метода приводит к созданию шаблона метода в редакторе скриптов, нажатие на кнопку справа от поля ввода имени метода позволяет выбрать нужный метод из выпадающего списка.

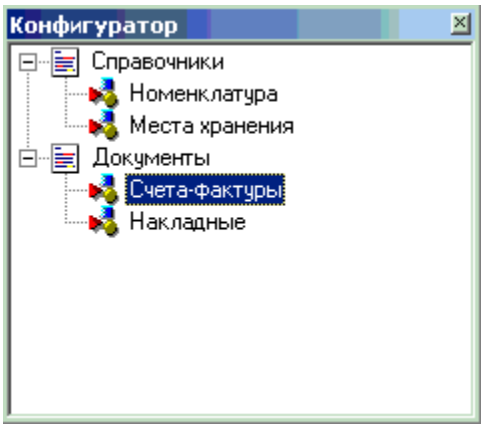
### 2.6.2 Инспектор скриптов




Инспектор скриптов содержит список процедур и функций скрипта. При смене

активного скрипта содержимое инспектора автоматически обновляется. Список процедур и функций представлен в виде дерева с отражением вложенности процедур и функций. Помимо просмотра, инспектор предназначен для быстрого позиционирования на нужный объект. Следует дважды щелкнут по имени процедуры в списке, для того что бы перейти к ней в редакторе скриптов.

### 2.6.3 Конфигуратор




Конфигуратор служит для создания дерева метаобъектов конфигурации. Для редактирования свойств метаобъектов и задания обработчиков событий используется стандартный инспектор объектов. Когда вы щелкаете правой кнопкой мыши по объекту в дереве конфигурации, появляется Popup меню, которое содержит список всех классов метаданных, которые могут быть созданы как подчиненные текущему объекту. Для того, что бы создать новый объект в дереве, просто выбирите имя нужного класса из выпадающего меню. Конфигуратор имеет открытую архитектуру, т.е. в ИСР можно добавлять новые метаклассы, которые будут доступны для использования в конфигураторе. Подробнее, о том как создавать собственные метаклассы, смотрите Руководство программиста.

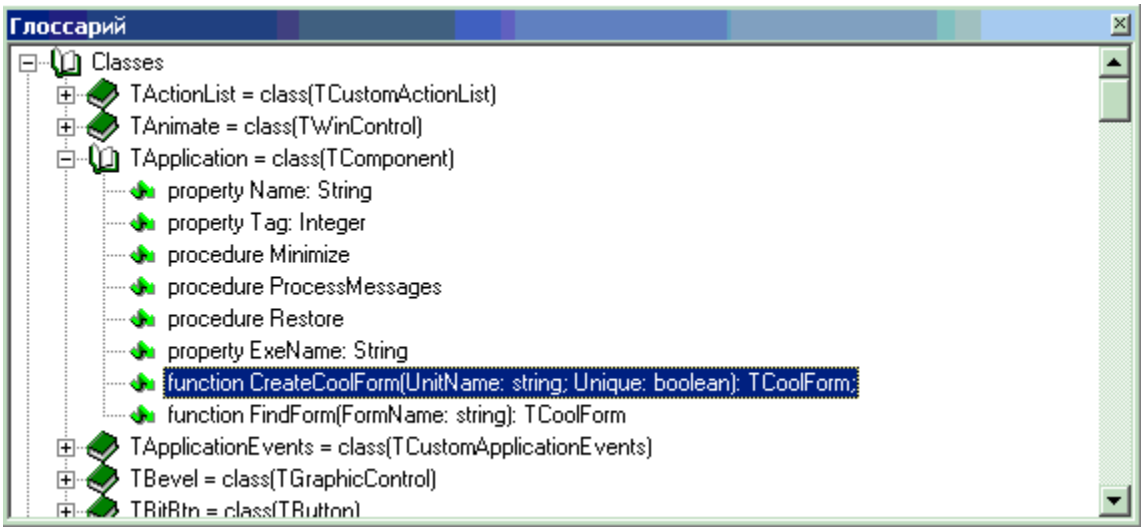
Дополнительные команды конфигуратора:

Delete — Удалить текущий объект из конфигурации. Удаляется сам объект и все подчиненные ему объекты. С помощью этой команды нельзя удалить модули конфигурации (объекты класса TcmComponentUnit). Для того, что бы убрать модуль из конфигурации, следует воспользоваться соответствующим пунктом меню **Конфигурация**.

Shift - Up — Переместить объект на одну позицию вверх по списку.

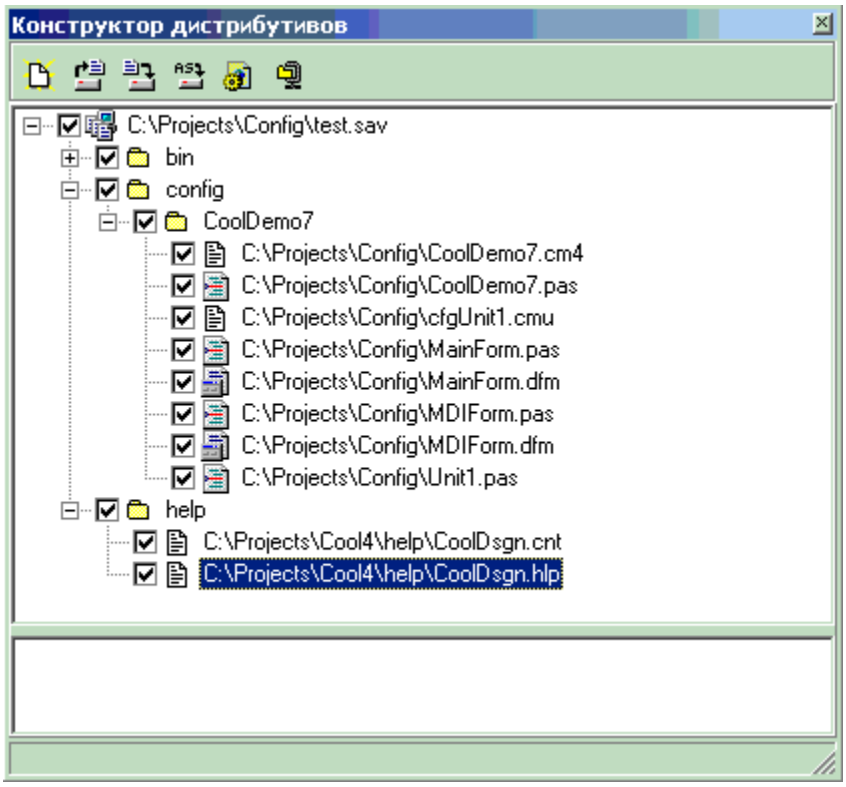
Shift - Down — Переместить текущий объект на одну позицию вниз по списку.

### 2.6.4 Глоссарий




Глоссарий содержит список классов, процедур, функций, типов и переменных доступных в скриптовом языке. Для удобства просмотра, информация в глоссарии представлена в виде дерева. Для поиска нужного объекта в глоссарии по его имени воспользуйтесь командой поиска **<Ctrl - F>**. Для получения оперативной справки по выбранному элементу нажмите клавишу **<Enter>**.

### 2.6.5 Конструктор дистрибутивов




После того, как приложение создано, требуется создать дистрибутив программы для передачи заказчику. Конструктор дистрибутивов призван максимально упростить эту

задачу. С помощью данного инструмента вы можете сформировать список файлов, которые следует передать заказчику и на его основе создать ZIP архив. Список дистрибутива имеет древовидную структуру, узлами которой являются каталоги, а элементами файлы. На панели управления расположен набор кнопок для вызова команд, доступных при работе с конструктором.

Список команд конструктора (команды перечислены в той же последовательности, в которой они расположены на панели управления окна конструктора):

| | |
|---|---|
| Новый | Создать новый дистрибутив |
| Загрузить | Загрузить файл с описанием дистрибутива. |
| Сохранить | Сохранить описание дистрибутива на диске. |
| Сохранить как | Сохранить описание дистрибутива под другим именем. |
| Формирование дистрибутива | Автоматическое формирование дистрибутива. В дистрибутив могут быть добавлены используемые приложением exe и bpl файлы, а так же модули конфигурации. |
| Создание дистрибутива | Создает дистрибутив приложения в виде ZIP архива. |

Любой элемент дерева дистрибутива имеет checkbox, если флажок не установлен, элемент не будет включаться в дистрибутив. Если не установлен флажок для каталога, в дистрибутив не будет включен ни сам каталог, ни его подкаталоги. Для установки\снятия флажка воспользуйтесь клавишей Пробел.

Щелчок правой кнопкой мыши по элементу каталога вызывает Popup меню элемента, оно содержит следующие команды:

| | |
|---|---|
| Создать подраздел **Insert** | Создать подкаталог у текущего каталога |
| Переименовать **Enter** | Переименовать текущий каталог |
| Добавить файл | Выводится диалог загрузки файла, если выбрать файл, он будет помещен в текущий каталог. |
| Поиск файлов | Можно задать шаблон поиска файлов и каталог в котором следует искать файлы, все найденные файлы будут добавлены в текущий каталог. |
| Удалить **Delete** | Удалить текущий элемент. |
| На одну позицию вверх **Ctrl - Up** | Перемещает элемент на подну позицию вверх по списку. |
| На одну позицию вниз **Ctrl-Down** | Перемещает элемент на одну позицию вниз по списку. |

## 2.7 Главное меню. Конфигурация.

Раздел Конфигурация содержит команды для редактирования конфигурации.

| | |
|---|---|
| **Создать конфигурацию** | Используется для создания новой конфигурации. |
| **Загрузить конфигурацию** | Вызывает диалог для загрузки конфигурации. |
| **Сохранить конфигурацию** | Сохраняет изменения внесенные в конфигурацию. |
| **Добавить модуль в конфигурацию** | Вызывает диалог для создания нового модуля конфигурации. |

| | |
|---|---|
| **Удалить модуль из конфигурации** | Вызывает диалог для удаления модуля из конфигурации. |
| **Показать скрипт конфигурации** | Загружает в редактор скриптов главный скрипт конфигурации. |
| **Параметры конфигурации** | Вызывает диалог для настройки параметров конфигурации.<br>В форме диалога можно задать список каталогов для поиска следующих объектов:<br>• скрипты<br>• BPL пакеты<br>• модули конфигурации<br>• шаблоны отчетов. |

см. также: Модули конфигурации.

## 2.8 Главное меню. Пуск.

Раздел Пуск содержит команды предназначенные для выполнения и отладки конфигурации.

| | |
|---|---|
| **Компиляция** | Компилировать текущий скрипт в редакторе скриптов. |
| **Компиляция конфигурации** | Компилировать скрипт конфигурации. |
| **Выполнить** | Запустить конфигурацию на выполнение. |
| **Прервать** | Прервать выполнение конфигурации. |

*Приведенные ниже команды предназначены для отладки скриптов конфигурации. Используются для трассировки скриптов и просмотра текущего значения переменных и выражений.*

| | |
|---|---|
| **Шаг через** | Перейти к следующему оператору. |
| **Шаг внуть** | Если текущий оператор - скриптовая процедура или функция перейти к её первому оператору. |
| **Идти до курсора** | Идти до строки в которой установлен курсор. |
| **Инспектор выражений** | Вызывает инспектор выражений. |
| **Монитор сообщений** | Вызывает монитор сообщений. |

## 2.9 Главное меню. Настройки.

Раздел Настройка содержит команды для настройки и конфигурирования среды разработки.

**Опции дизайнера** Вызывает диалог для настройки дизайнера форм.

*Настройка параметров редактора скриптов*

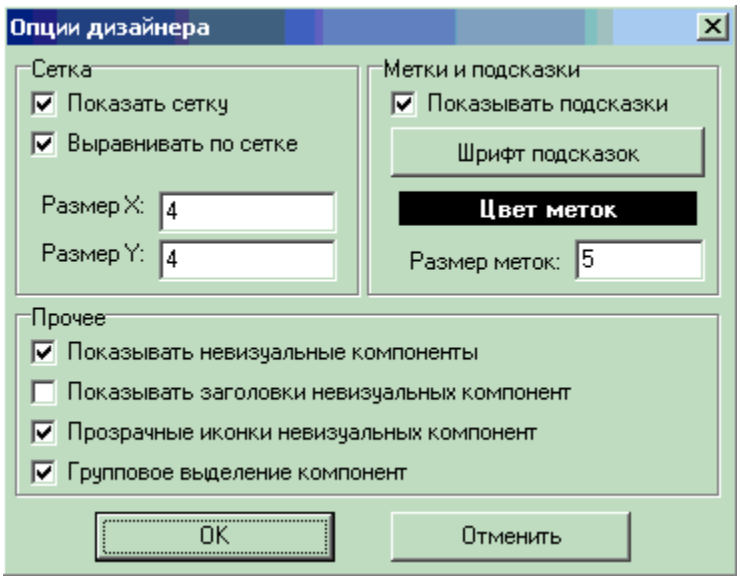
| | |
|---|---|
| **Опции редактора** | Вызывает диалог настройки параметров текстового редактора. |
| **Стили** | Вызывает диалог настройки стилей (подсветка синтаксиса). |
| **Шаблоны** | Вызывает диалог для задания шаблонов языковых конструкций. |
| **Клавиши управления** | Вызывает диалог для настройки клавиш управления текстового редактора. |

*Работа с пакетами*

| | |
|---|---|
| **Установка пакетов** | Вызывает диалог для загрузки и выгрузки bpl пакетов. |
| **Настройка палитры** | Вызывает диалог настройки палитры компонентов. |

## 2.10 Диалоги настройки интегрированной среды

### 2.10.1 Параметры конструктора форм




**Сетка**

- **Показать сетку** - показывать сетку у формы в режиме дизайнера.
- **Выравнивать по сетке** - при вставке и перемещении компоненты будут автоматически выравниваться по сетке.
- **Размер X** - шаг сетки по горизонтали в пикселах
- **Размер Y** - шаг сетки по вертикали в пикселах.

**Метки и подсказки**

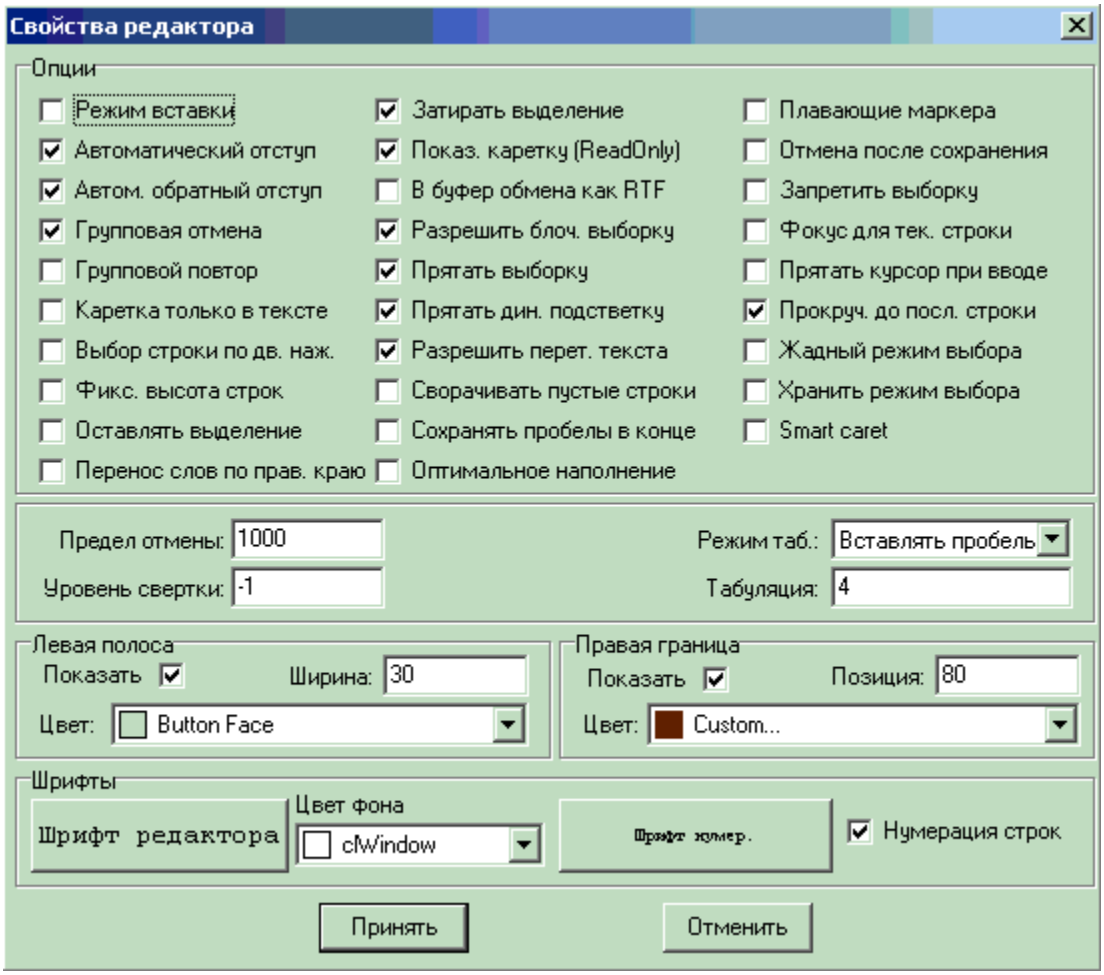
- **Показать подсказки** - показывать подсказки при перемещении компонент
- **Шрифт подсказок** - установка шрифта подсказок
- **Цвет меток** - устанавливается цвет меток, которыми отмечается выбранный компонент на форме.
- **Размер меток** - задается размер меток.

**Прочее**

- **Показывать невизуальные компоненты** - выводит на форме значки невизуальных компонент.
- **Показывать заголовки невизуальных компонент** - выводит под значком невизуального компонента его название.
- **Прозначные иконки невизуальных компонент** - дополнительный режим отображения невизуальных компонент.
- **Групповое выделение компонент** - позволяет выделить группу компонент, иначе можно выделить только один компонент.

Настройки конструктора форм сохраняются в файле DsgnOpt.ecm

### 2.10.2 Параметры редактора скриптов




**Опции**

Опции используются для установок режима работы редактора скриптов. Включите флажок опции, что бы активировать её.

- **Режим вставки** - По умолачнию, при открытии редактора устанавливается режим вставки текста, установите опцию, что бы активировался режим замены.
- **Автоматический отступ** - при переходе на новую строку нажатием клавиши Enter. Новая строка автоматически дополняется тем же количеством начальных пробелов, что и предыдущая строка.
- **Автом. обратный отступ** - Установка числа начальных пробелов в строке при нажатии клавиши BackSpace. Если число пробелов в текущей строке больше чем в предыдущей, отступ выравнивется по предыдущей строке. Работает только когда курсор установлен на первый значимый символ строки (не пробел).
- **Групповая отмена** - откат внесенных изменений группами операций одного типа. Т.е. если в редакторе было напечатана последовательность символов, то при включенной опции будет отменена вся строка, а при отключенной будет происходить посимвольная отмена.
- **Групповой повтор** - групповая отмена отката. Работает аналогично Групповой

отмене.

- **Каретка только в тексте** - возможно перемещение курсора только по введенному тексту (как в стандартном блокноте Windows)
- **Выбор строки по дв.наж**. - при двойном щелчке по тексту выделяет строку целиком, если отключено, выделяет только слово под курсором.
- **Фикс. высота строк** - при работе с редактором можно задавать различные стили для отображения блоков текста (см. Подсветка синтаксиса), поэтому, строки могут иметь различную высоту (в зависимости от размера используемых в стиле шрифтов). Если опция включена, все строки в редакторе будут иметь фиксированную высоту, иначе высота строк может варьироваться в зависимости от используемых стилей.
- **Оставлять выделение** - если выделить блок текста и начать вводить символы с клавиатуры выделенный блок не удаляется. Работает только при выключенной опции **Затирать выделение**.
- **Перенос слов по правому краю** - если для редактора установлен режим переноса текста по словам, то при установленной опции перенос производится при достижении правой границы текста (см. ниже), иначе перенос производится при достижении края клиентской области редактора.
- **Затирать выделение** - если опция включена, то выделенный блок затирается при вводе символов с клавиатуры, иначе блок остается.
- **Показ. каретку (Read only)** - показывать каретку когда для редактора установлен режим чтения, иначе каретка скрывается.
- **В буфер обмена как RTF** - копировать выделенный текст в буфер в формате RTF.
- **Разрешить блочную выборку** - разрешает выделять вертикальные блоки текста. Для того, что бы выделить вертикальный блок текста следует воспользоваться клавишами управления курсором, удерживая нажатыми пару клавиш: <Alt> - <Shift>.
- **Прятать выборку** - скрывать выделенный блок текста, когда форма редактора теряет фокус ввода.
- **Прятать дин. подсветку** - отменять подсветку текущего диапозона, когда окно теряет фокус ввода.
- **Разрешить перет. текста** - разрешает выполнять над выделенным блоком текста операции drag&drop.
- Сворачивать пустые строки - разрешает скрывать пусты строки, следующие за свернутым диапозоном.
- **Сохранять пробелы в конце** - разрешает сохранять пробелы в конце строки.
- **Оптимальное наполнение** - оптимальное наполнение строки символами при использовании различных стилей.
- **Плавающие маркера** - если в тексте установлены маркера, позволяет им перемещаться вместе с текстом при редактировании. Так же эта опция позволяет сохранять текущую позицию в тексте.
- **Отмена после сохранения** - если опция включена, то операции отмены можно выполнять после сохранения текста.
- **Запретить выборку** - запрещает выделение блоков текста.
- **Фокус для тек. строки** - рисует прямоугольник вокруг активной строки, когда редактор получает фокус ввода.
- **Прятать курсор при вводе** - если указатель мыши в тексте, то при вводе текста он

будет скрыт.

- **Прокручивать до последней строки** - если установлено, то прокрутка текста завершается на последней линии текста, иначе в конце страницы. Когда эта опция выключена и размер текста меньше клиентской области редактора, вертикальная полоса прокрутки скрыта.
- **Жадный режим выбора** - при выделении вертикального блока текста захватывает один дополнительный символ (как в редакторе IDE Delphi)
- **Хранить режим выбора** - Сохраняет режим выделения при перемещении по тексту редатора.
- **Smart caret** - Каретка перемещается между строками к ближайшей позиции (например, когда шрифт не регулярный, каретка не прыгает из стороны в сторону).

**Левая полоса.**

- **Показать** - выводит с левого края редактора служебную полосу.
- **Ширина** - ширина полосы.
- **Цвет** - цвет полосы.

**Правая граница.**

- **Показать** - рисует правую границу в редакторе.
- **Позиция** - число столбцов, после которых следует граница.
- **Цвет** - цвет линии границы.

**Другие опции.**

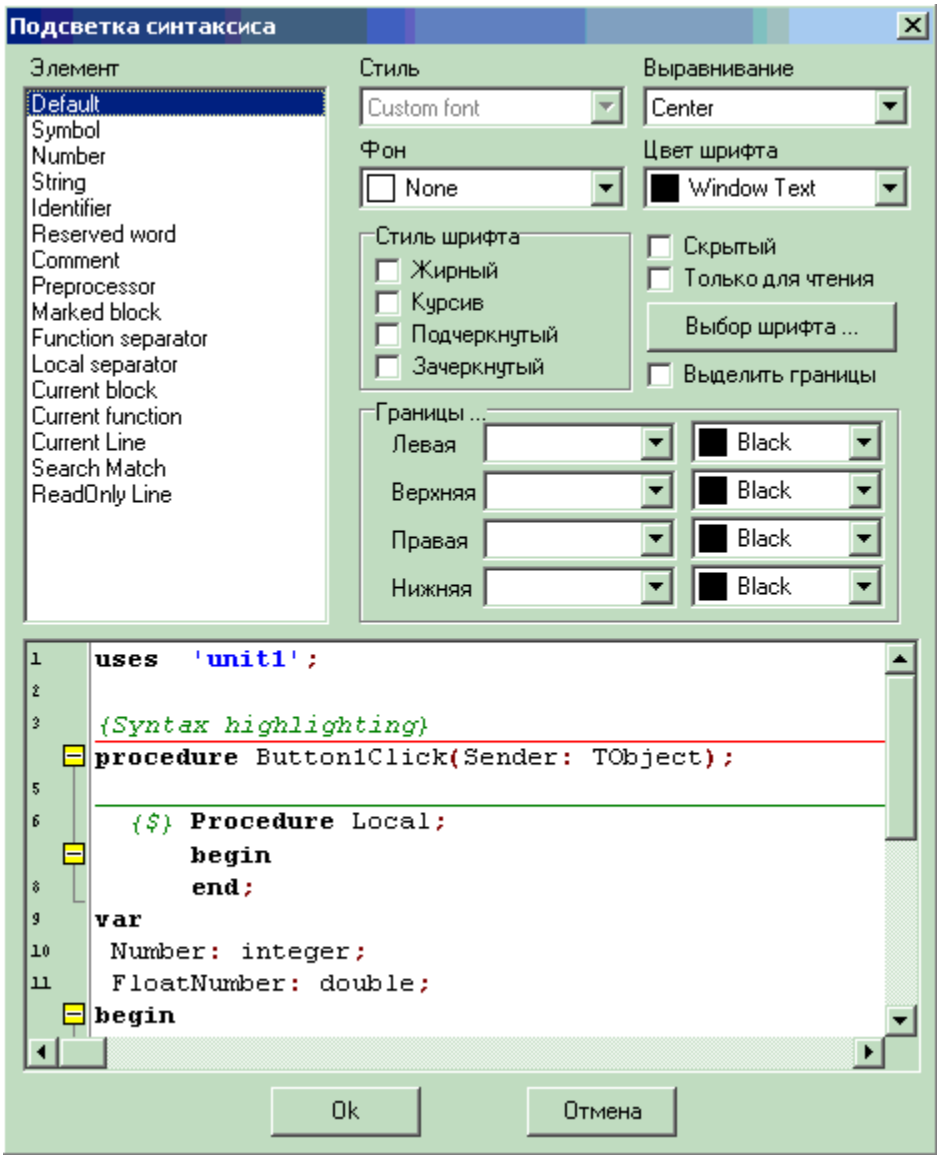
- **Предел отмены** - максимальное число команд сохраняемое в бувере отмены.
- **Уровень свертки** - в текущей версии редактора не используется.
- **Режим табуляции** - устанавливает режим табуляции.
- **Табуляция** - число символов на один знак табуляции.

Шрифты.

- **Шрифт редактора** - устанавливает шрифт редактора по умолчанию.
- **Цвет фона** - цвет фона редактора.
- **Шрифт нумер**. - шрифт нумератора строк
- **Нумерация строк** - включить режим нумерации строк.

Настройки редактора сохраняются в файле EditOpt.ecm

### 2.10.3 Подсветка синтаксиса




Диалог предназначен для настройки стилей используемых в редакторе скриптов. Так как скриптовый язык ИРС поддерживает четыре языка (Pascal, C++, Java и Visual Basic), стили для каждого из них настраиваются отдельно.

Список **Элемент** содержит список определенных для редактора лексем:

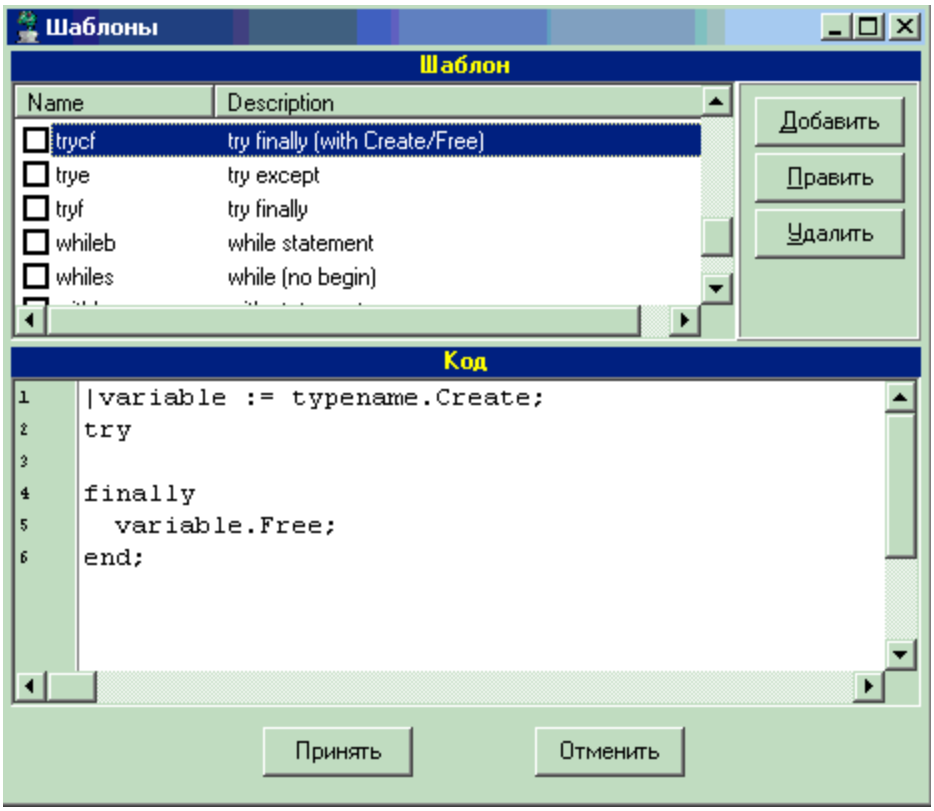
- **Default** - стиль по умолчанию
- **Symbol** - символы
- **Number** - числа
- **String** - строковые константы
- **Identifier** - идентификаторы
- **Reserved word** - зарезервированные слова
- **Comment** - коментарии
- **Preprocessor** - символы препроцессора

- **Marked block** - помеченный блок
- **Function separator** - линия разделитель функций
- **Local separator** - линия разделитель вложенных функций
- **Current block** - текущий диапазон
- **Current function** - текущая функция
- **Current line** - текущая строка в редакторе
- **Search Match** - используется для выделения найденных подстрок при операциях поиска
- **Read only line** - стиль линий в режиме только для чтения

Для каждого элемента можно задать шрифт, цвет фона, способ выравнивания, границы. Диалог также содержит текстовый редактор для визуального отображения сделанных настроек.

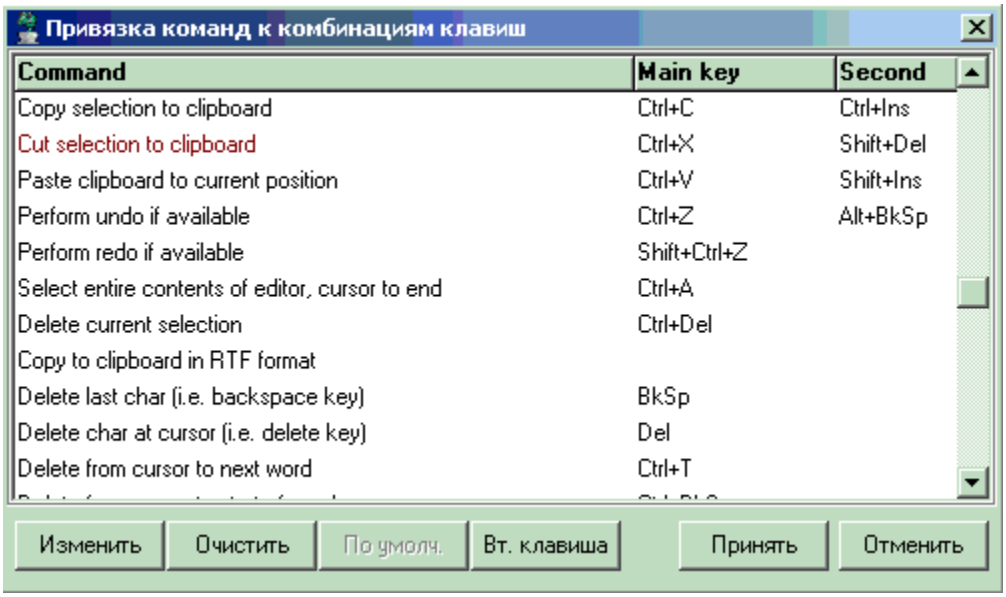
Настройка стилей сохраняется в файле - Cool.lxl

### 2.10.4 Настройка шаблонов языка



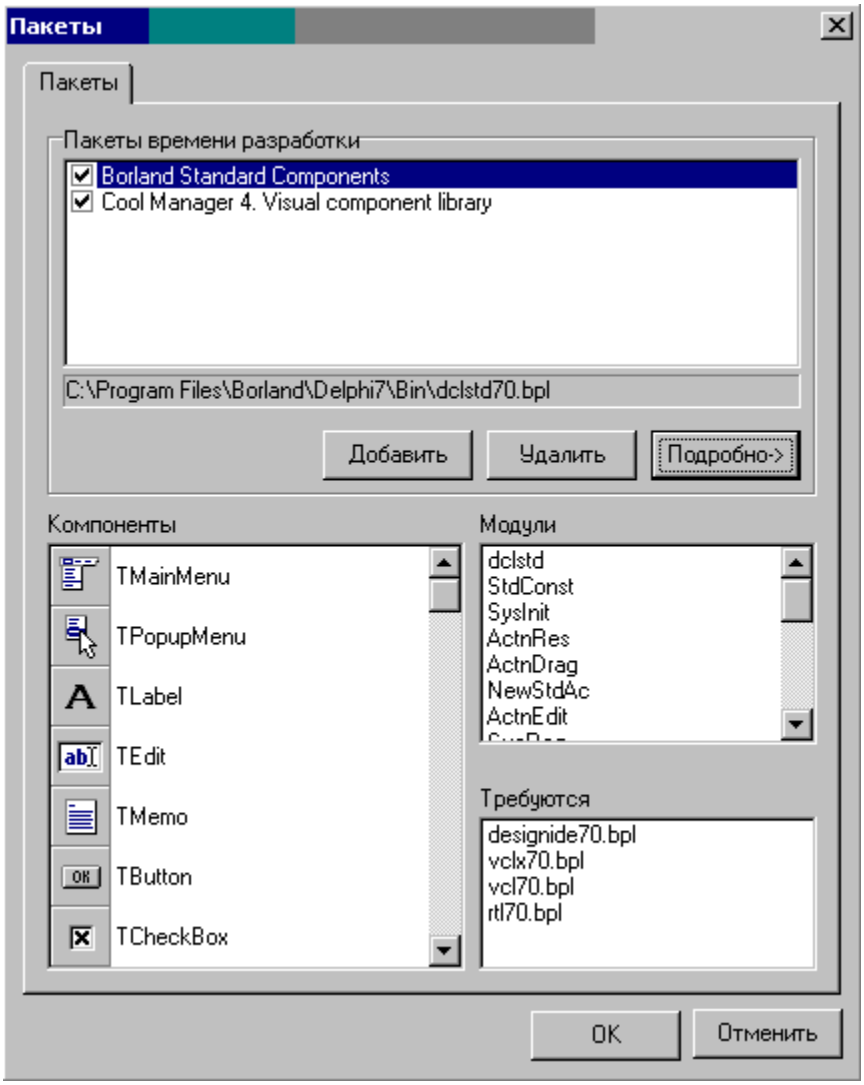

Диалог позволяет задать шаблоны для редактора скриптов. Для использования определенных шаблонов следует ввести идентификатор определенный в столбце Name и нажать пару клавиш <Ctrl - J>, идентификатор будет заменен кодом шаблона. Так как скриптовый язык ИРС поддерживает четыре языка (Pascal, C++, Java и Visual Basic), шаблоны для каждого из них настраиваются отдельно.

### 2.10.5 Настройка горячих клавиш редактора



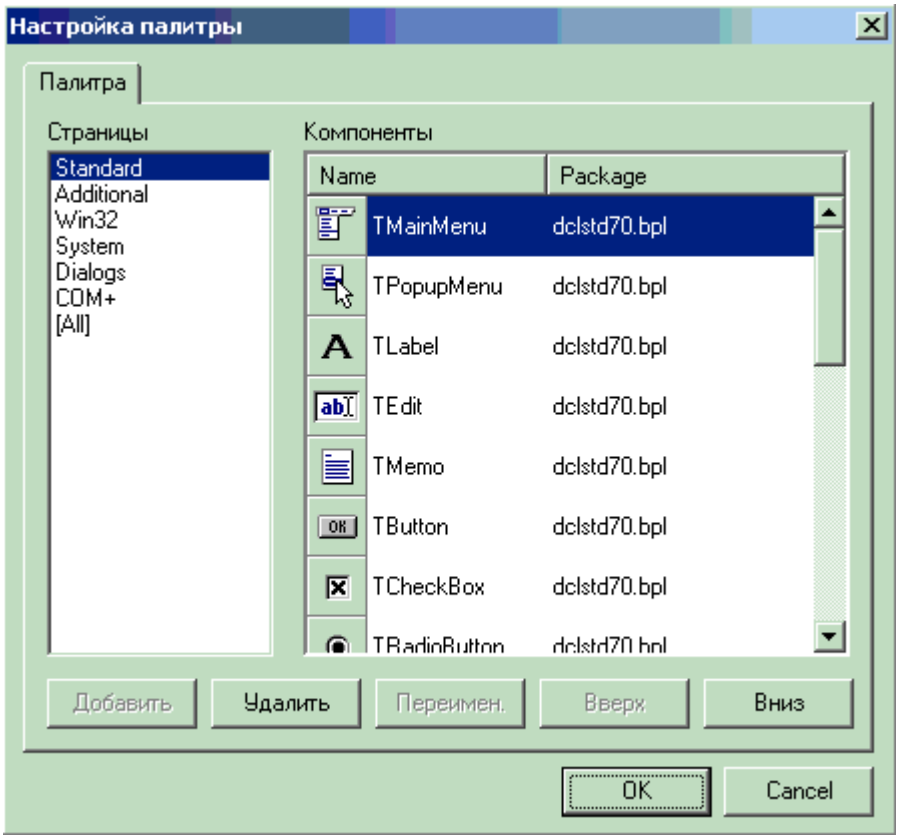

Диалог используется для настройки комбинаций клавиш и привязки их к командам редактора. Для каждой команды может определено два сочетания клавиш - основная и дополнительная.

### 2.10.6 Загрузчик пакетов




Диалог используется для загрузки VCL пакетов ИСР системы Cool Manager. Для того, что бы добавить новый пакет следует нажать кнопку **Добавить** и в окне выбора файла указать нужный пакет. Для удаления пакета из ИСР следует нажать кнопку **Удалить**. Все компоненты зарегистрированные для отражения в палитре компонентов, будут добавлены в палитру, кроме того, они будут подключены к скриптовому языку. По умолчанию регистрируются все published свойства компонент, для подключения к языку public свойств и методов, а так же для регистрации компонент не подключенных к палитре, следует написать дополнительный код. Подробнее о регистрации классов, типов, переменных, процедур и функций в скриптовом языке см. Руководство программиста.

### 2.10.7 Настройка палитры компонент



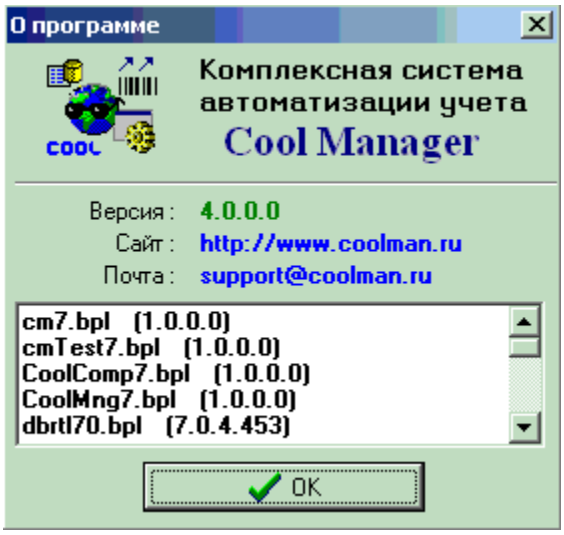

Используется для настройки внешнего вида палитры компонент. Список страниц содержит все страницы определенные в палитре, можно создавать новые страницы и удалять существующие. Список Компоненты содержит список компонент текущей страницы. Компоненты можно скрывать, менять местами и переносить на другие страницы палитры.

## 2.11 Главное меню. Помощь.

Раздел Помощь содержит команды для обращения к системе помощи и он-лайн поддержки зарегистрированных пользователей.

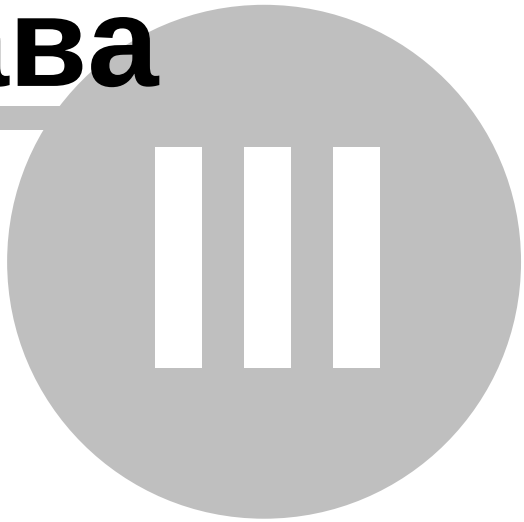
| | |
|---|---|
| **О программе** | Вызывает диалог с информацией о текущей версии программы и используемых библиотеках. |
| **Введение в систему** | Описание принципов построения системы CoolManager |
| **Система помощи** | Справка по среде разработки |
| **Справка по языку** | Справка по скриптовому языку системы |
| **CoolManager Direct** | Доступ к серверу поддержки зарегистрированных пользователей |

## 2.12 Диалог "О программе"




Диалог "О программе" показывает текущую версию ИСР, а так же список загруженных в ИСР пакетов с указанием их версии.

# Глава



## Скриптовый язык

# 3 Скриптовый язык

## 3.1 Возможности языка

В качестве скриптового языка системы Cool Manager используется FastScript www.fast-report.com.

**Реализованные особенности**

- мультиязычная архитектура, позволяющая использовать множество языков (в настоящее время - PascalScript, C++Script, JScript и BasicScript).
- стандартный языковой набор: переменные, константы, процедуры, функции (с возможностью вложенности) с переменными/постоянными/умалчиваемыми параметрами, все стандартные операторы и объявления (включая case, try/finally/except, with), типы (целый, дробный, логический, символьный, строковый, многомерные массивы, множество, variant), классы (с методами, событиями, свойствами, индексами и свойствами по умолчанию).
- проверка совместимости типов.
- доступ к любому объекту вашего приложения. Стандартные библиотеки для доступа к базовым классам, контролам, формам и БД. Легко расширяемая архитектура библиотеки.
- Компактность - 90-150Кб в зависимости от используемых модулей.

**Нереализованные особенности**

- Отсутствуют объявления типов (records, classes) в скрипте; нет записей (records), указателей (pointers), множеств (sets) (однако возможно использование оператора 'IN' - "a in ['a'..'c','d']"), нет типа shortstrings, нет безусловного перехода (GOTO).
- C++Script: нет восьмеричных констант; нет 'break' в операторе SWITCH (SWITCH работает подобно Pascal CASE); операторы '++' и '--' возможны только после переменных, т.е. '++i' не будет работать; операторы '--', '++' и '=' ничего не возвращают, т.е. 'if(i++)' не будет работать; все идентификаторы не чувствительны к регистру; Константа NULL это Null из Pascal- используйте nil вместо NULL.
- JScript и BasicScript: см. синтаксические диаграммы.

**Классы.**
Нельзя объявить класс в скрипте, но можно использовать внешние классы, объявленные в подключаемых bpl пакетах. Возможно получить доступ к любому свойству (простому, индексному или умалчиваемому) или методу. По умолчанию все published свойства объектов доступны из скрипта. Public свойства и методы нуждаются в написании дополнительного кода инициализации. О том как подключать к скриптовому языку новые классы, процедуры, типы, переменные - см. **Руководство программиста**.

См. также:
Структура скрипта
Типы данных

Стандартные_функции
Синтаксис Pascal
Синтаксис C++
Синтаксис JScript
Синтаксис Visual Basic

## 3.2 Структура скрипта

**Структура PascalScript**

```
#language PascalScript // опционально
program MyProgram; // опционально
uses 'unit1.pas', 'unit2.pas'; // раздел uses должен быть перед любыми другими разделами
var // раздел var
i, j: Integer;
const // раздел const
pi = 3.14159;

procedure p1; // процедуры и функции
 var i: Integer;

 procedure p2; // вложенная процедура
  begin
  end;

begin
end;

begin // главный исполняемый модуль.
end.
```

**Структура C++Script:**

```
#language C++Script // опционально
#include "unit1.cpp", "unit2.cpp"
// раздел include - должен быть перед любым другим разделом
int i, j = 0; // раздел переменных - может быть в любом месте
#DEFINE pi = 3.14159 // раздел констант
void p1() // функции
{ // вложенных процедур нет
}
{ // главная исполняемая функция.
}
```

**Структура JScript:**

```
#language JScript // опционально
import "unit1.js", "unit2.js"
// раздел import - должен быть перед любым другим разделом
var i, j = 0; // раздел переменных - может быть в любом месте
function p1() // функции
{ //
}
// главная исполняемая функция.
p1();
for (i = 0; i < 10; i++) j++;
```

**Структура BasicScript:**

```
#language BasicScript // опционально
imports "unit1.vb", "unit2.vb"
// раздел imports - должен быть перед любым другим разделом
dim i, j = 0 // раздел переменных - может быть в любом месте
function p1() // функции
{ //
}
// главная исполняемая функция.
for i = 0 to 10
p1()
next
```

## 3.3 Типы данных

FastScript работает с типом Variant и основан на нём. Тем не менее, вы можете использовать следующие предопределённые типы в ваших скриптах:

| | |
|---|---|
| цеолчисленные | **Byte**<br>**Word**<br>**Integer**<br>**Longint**<br>**Cardinal**<br>**TColor** |
| логический | **boolean** |
| расширенный (с плавающей запятой) | **real**<br>**Single**<br>**Double**<br>**Extended**<br>**Currency**<br>**TDate**<br>**TTime**<br>**TDateTime** |
| символьный | **Char** |

| | |
|---|---|
| строковый | **string** |
| Variant (вариантный тип) | **variant** |
| | **pointer** |
| массив | **array** |

Соответствие некоторых типов C++Script стандартным типам:
**int, long = Integer**
**void = Integer**
**bool = Boolean**
**float = Extended**

JScript не имеет описаний типов - все типы являются Variant. BasicScript может использовать описание типов (напр. dim i as Integer), а может опускать тип или даже объявление переменной. В этом случае она считается типом Variant. Не все из этих типов могут быть неявно приведены один к другому. Как и в Object Pascal, вы не можете привести Extended или String к Integer. Только один тип - Variant - может быть присвоен любому типу и получить значение от любого типа.Помимо встроенных типов, вы можете использовать перечислимые типы, объявленные в вашем приложении или в дополнительных модулях.

## 3.4 Использование директивы uses

Вы можете разбивать большой скрипт на модули, подобно тому, как это делается в Object Pascal. Для использования модуля служит директива "uses". Вот пример ее применения:

*Файл unit1.pas:*
```
uses 'unit2.pas';
begin
 Unit2Proc('Hello!');
end.
```

*Файл unit2.pas:*
```
procedure Unit2Proc(s: String);
begin
 ShowMessage(s);
end;

begin
ShowMessage('initialization of unit2...');
end.
```

Как видно, отличие от Object Pascal заключается в том, что мы указываем в uses имя файла с расширением в одинарных кавычках. Подключаемый файл должен иметь такую же структуру, как и основной. Код, заключенный в основной блок begin..end, будет выполнен при подключении модуля - это аналог секции initialization в Pascal. В

этом примере нельзя использовать unit1 из unit2 - это вызовет бесконечный цикл при попытке откомпилировать такой скрипт. Подобные ссылки невозможны, т.к. в скриптовом языке нет аналогов паскалевским конструкциям interface/implementation. Пользуясь директивой #language, можно писать многоязычные скрипты. Так, один модуль может быть написан на PascalScript, другой - на C++Script:

*Файл unit1.pas:*

```
uses 'unit2.pas';
begin
 Unit2Proc('Hello from PascalScript!');
end.
```

*Файл unit2.pas:*

```
#language C++Script
void Unit2Proc(string s)
{
ShowMessage(s);
}
{
ShowMessage("unit2 initialization, C++Script");
}
```

Директива #language должна быть первой строкой в файле.

## 3.5 Перечисления и множества

FastScript поддерживает перечисления. Вы можете написать в скрипте:
Form1.BorderStyle := bsDialog;

Множества не поддерживаются. Тем не менее, вы можете оперировать элементами множества:

| В FastScript | В Delphi |
|---|---|
| Font.Style := fsBold; | Font.Style := [fsBold] в Delphi |
| Font.Style := fsBold + fsItalic; | Font.Style := [fsBold, fsItalic] |
| Font.Style := 0; | Font.Style := [] |

## 3.6 Массивы

FastScript поддерживает все типы массивов: статические (одномерные, многомерные), динамические, вариантные. Вот пример скрипта, использующего три массива целых чисел, объявленных разным способом:

```
var
  ar1: array[0..2] of Integer;
  ar2: array of Integer;
```

```
  ar3: Variant;

SetLength(ar2, 3);
ar3 := VarArrayCreate([0, 2], varInteger);
ar1[0] := 1;
ar2[0] := 1;
ar3[0] := 1;
```

## 3.7 Оператор присваивания

Оператор присваивания записывается в виде:

<переменная> := <выражение>;

где <переменная> - переменная или типизированная константа любого типа, объект и т.д., а <выражение> - любое допустимое выражение, совместимое по типу с переменной в левой части оператора. Оператор вычисляет значение выражения, записанного как правый операнд операции присваивания :=, и присваивает полученное значение переменной в левой части оператора.

Например, оператор

I := 3;

присваивает переменной I значение 3. Оператор

I := I + 1;

увеличивает значение переменной I на 1.

## 3.8 Оператор with..do

Оператор with...do используется для сокращения записи при обращении к полям записей или к свойствам и методам объекта. В этих случаях применение with...do позволяет избежать повторных ссылок на объект в последующих операторах. Оператор with...do может записываться следующим образом:

with <объект> do <оператор>;

В операторе, следующем за ключевым словом do, можно для полей, свойств и методов объекта, указанного как <объект>, не включать ссылки на этот объект. При этом каждый идентификатор в операторе, который совпадает с именем поля, свойства, метода объекта, трактуется как относящийся к этому объекту и к нему неявно добавляется ссылка на этот объект.

Если объект в операторе with...do определяется как элемент индексированного массива, то соответствующая операция индексации выполняется только один раз до начала

выполнения оператора, следующего за ключевым словом do. Никакие присваивания внутри этого оператора не влияют на интерпретацию понятия <объект>, принятого в конструкции with...do.

Возможна другая форма оператора with...do:

with <объект 1>, <объект 2>, ..., <объект n> do <оператор>;

Она соответствует множеству вложенных друг в друга конструкций with:

```
with <объект 1> do
 with <объект 2> do
  ...
   with <объект n> do
    <оператор>;
```

## 3.9 Оператор if

Оператор if предназначен для выполнения тех или иных действий в зависимости от истинности или ложности некоторого условия. Условие задается выражением, имеющим результат булева типа.
Оператор имеет две формы: if...then и if...then...else.

Оператор выбора if...then записывается в форме:

if <условие> then <оператор>;

Если условие возвращает true, то указанный в конструкции if оператор выполняется. В противном случае управление сразу передается следующему за конструкцией if оператору. Например, в результате выполнения операторов

```
C := A;
if B > A then C := B;
```

переменная C станет равна максимальному из чисел A и B, поскольку оператор C := B будет выполнен только при B > A.

Оператор выбора if...then...else записывается в форме:

if <условие> then <оператор1> else <оператор2>;

Если условие возвращает true, то выполняется первый из указанных операторов, в противном случае выполняется второй оператор. Обратите внимание, что в конце первого оператора перед ключевым словом else точка с запятой не ставится.

При вложенных конструкциях if могут возникнуть неоднозначности в понимании того,

к какой из вложенных конструкций if относится элемент else. Интерпритатор всегда считает, что else относится к последней из конструкций if, в которой не было раздела else.

Например, в конструкции

```
if <условие1> then if <условие2> then <оператор1>
            else <оператор2>;
```

else будет отнесено интерпритатором ко второй конструкции if, т.е. <оператор2> будет выполняться в случае, если первое условие истинно, а второе ложно. Иначе говоря, вся конструкция будет прочитана как

```
if <условие1>
 then begin
  if <условие2> then <оператор1> else <оператор2>
 end;
```

Если же вы хотите отнести else к первому if, это надо записать в явном виде с помощью операторных скобок begin...end:

```
if <условие1>
 then begin
   if <условие2> then <оператор1>
  end
 else <оператор2>;
```

## 3.10 Оператор case

Оператор множественного выбора case позволяет провести анализ значения некоторого выражения и в зависимости от его значения выполнять те или иные действия. В общем случае формат записи оператора case следующий:

```
case <выражение> of
 <список значений 1>: <оператор 1>;
  ...
 <список значений n>: <оператор n>;
else
 <оператор>
end;
```

В этой конструкции выражение должно иметь порядковый тип. Поэтому, например, нельзя использовать выражения, возвращающие действительные числа или строки.

Списки значений могут содержать одно или несколько разделенных запятыми возможных значений константных выражений. После списка ставится двоеточие ":", а

затем пишется оператор (может писаться составной оператор), который должен выполняться, если выражение приняло одно из перечисленных в списке значений. После выполнения этого оператора работа структуры case завершается и управление передается следующему за этой конструкцией оператору. Операторы всех последующих разделов не выполняются.

Если значение выражения не соответствует ни одному из перечисленных во всех списках, то выполняется оператор, следующий после ключевого слова else. Впрочем, раздел else не обязательно должен включаться в структуру case. В этом случае, если в списках не нашлось соответствующего значения выражения, то ни один оператор не будет выполнен.

Списки могут содержать константы и константные выражения, которые совместимы по типу с объявленным выражением. Допустимо использование ограниченных типов. Не допустимо использование переменных и многих функций. В списках не допускается повторения одних и тех же значений, поскольку в этом случае выбор был бы неоднозначным.

## 3.11 Оператор for

Оператор for обеспечивает циклическое повторение некоторого оператора - тела цикла заданное число раз. Повторение цикла определяется некоторой управляющей переменной (счетчиком), которая увеличивается или уменьшается на единицу при каждом выполнении тела цикла. Повторение завершается, когда управляющая переменная достигает заданного значения.
Оператор for записывается в одной из следующих форм:

for <счетчик>:=<начальное значение> to <конечное значение>
  do <оператор>;

или

for <счетчик>:=<начальное значение> downto <конечное значение>
  do <оператор>;

<счетчик> - локальная управляющая переменная порядкового типа. В начале выполнения оператора for ей присваивается <начальное значение>. После каждого очередного выполнения тела цикла <оператор> ее значение увеличивается (в первой форме с to) или уменьшается (во второй форме с downto) на единицу. Когда значение управляющей переменной достигает значения <конечное значение>, тело цикла выполняется последний раз, после чего управление передается оператору, следующему за структурой for. <начальное значение> и <конечное значение> являются выражениями, совместимыми по типу с управляющей переменной.

Если заданные начальное и конечное значения равны друг другу, тело цикла выполняется только один раз. Если в форме с to начальное значение больше конечного

или в форме с downto начальное значение меньше конечного, то тело цикла не выполняется ни разу.

Внутри цикла значение управляющей переменной может использоваться в выражениях. Однако, после окончания выполнения структуры for значение управляющей переменной не определено.

Досрочное прерывание цикла или его текущей итерации может осуществляться так же, как и в других операторах цикла.

## 3.12 Оператор repeat..until

Структура repeat...until используется для организации циклического выполнения совокупности операторов, называемой телом цикла, до тех пор, пока не выполнится некоторое условие. Синтаксис управляющей структуры repeat...until:

```
repeat
  <операторы тела цикла>
until <выражение условия>;
```

Точка с запятой после последнего оператора тела цикла (перед ключевым словом until) может опускаться.

Структура работает следующим образом. Выполняются операторы тела цикла. Затем вычисляется <выражение условия>, которое должно возвращать результат булева типа. Если выражение возвращает false, то повторяется выполнение операторов тела цикла и после этого снова вычисляется выражение. Такое циклическое повторение цикла продолжается до тех пор, пока проверяемое выражение не вернет true. После этого цикл завершается и управление передается оператору, следующему за структурой repeat...until.

Поскольку проверка выражения осуществляется после выполнения операторов тела цикла, то эти операторы заведомо будут выполнены хотя бы один раз, даже если выражение сразу истинно.

Досрочное прерывание цикла или его текущей итерации может осуществляться так же, как и в других операторах цикла.

## 3.13 Оператор while..do

Структура while...do используется для организации циклического выполнения оператора, называемого телом цикла, пока выполняется некоторое условие. Синтаксис управляющей структуры while...do:

```
while <выражение условия> do <оператор>;
```

Структура работает следующим образом. Сначала вычисляется <выражение условия>, которое должно возвращать результат булева типа. Если выражение возвращает true, то выполняется оператор тела цикла, после чего опять вычисляется выражение, определяющее условие. Такое циклическое повторение выполнения оператора и проверки условия продолжается до тех пор, пока условие не вернет false. После этого цикл завершается и управление передается оператору, следующему за структурой while...do.

Поскольку проверка выражения осуществляется перед выполнением оператора тела цикла, то, если условие сразу ложно, оператор не будет выполнен ни одного раза. В этом основное отличие структуры while...do от структуры repeat...until, в котором тело цикла заведомо выполняется хотя бы один раз.

Досрочное прерывание цикла или его текущей итерации может осуществляться так же, как и в других операторах цикла.

## 3.14 Прерывание циклов

В некоторых случаях желательно прервать повторение цикла, проанализировав какие-то условия внутри его цикла. Это может потребоваться в тех случаях, когда проверки условия окончания цикла громоздкие, требуют многоэтапного сравнения и сопоставления каких-то данных и все эти проверки просто невозможно разместить в выражении условия операторов for, или while.
Для этого можно использовать процедуру Break. Она прерывает выполнение тела любого цикла for, repeat или while и передает управление следующему за циклом выполняемому оператору.

Для прерывания циклов, размещенных в процедурах или функциях, можно воспользоваться процедурой Exit. В отличие от процедуры Break, процедура Exit прервет не только выполнение цикла, но и выполнение той процедуры или функции, в которой расположен цикл.
Прервать выполнение цикла, а заодно - и блока, в котором расположен цикл, можно также генерацией какого-то исключения.

Описанные способы прерывали выполнение цикла. Имеется еще процедура Continue, которая прерывает только выполнение текущей итерации, текущего выполнения тела цикла и передает управление на следующую итерацию.

## 3.15 Процедуры и функции

Процедуры и функции представляют собой программные блоки, которые могут вызываться из разных частей программы. При вызове в них передаются некоторые переменные, константы, выражения, являющиеся аргументами, которые в самих процедурах и функциях воспринимаются как формальные параметры. При этом

функции возвращают значение определенного типа, которое замещает в вызвавшем выражении имя вызванной функции.
Например, оператор

I:=5*F(X);

вызывает функцию F с аргументом X, умножает возвращенное ею значение на 5 и присваивает результат переменной I.

Допускается также вызов функции, не использующий возвращаемого ею значения. Например:

F(X);

В этом случае возвращаемое функцией значение игнорируется.

Функция описывается следующим образом:

function <имя функции>(<список параметров>): <тип возвращаемого значения>;
<объявления локальных переменных, типов, констант, описания вложенных функций и процедур>
begin
 <операторы тела функции>
end;

Первая строка, содержащая имя функции, список параметров и тип возвращаемого значения, называется заголовком или объявлением функции.

Список параметров и объявления локальных элементов и вложенных процедур не обязательны. Если список параметров отсутствует, то скобки после имени функции не ставятся.

В простом случае список параметров - это перечисление имен формальных параметров с указанием их типов. Например, объявление:

function FSum(X1, X2: real; A: integer): real;

объявляет функцию с именем FSum, с тремя параметрами X1, X2 и A, из которых первые два типа real, а последний - integer. Тип возвращаемого результата - real. Имена параметров X1, X2 и A - локальные, т.е. они имеют значение только внутри данной функции и никак не связаны с именами аргументов, переданных при вызове функции. Значения этих параметров в начале выполнения функции равны значениям аргументов на момент вызова функции.

Тело функции пишется по тем же правилам, что и любой код программы. При этом

надо учитывать области видимости различных элементов программы. В теле функции видны объявленные в ней локальные элементы (переменные, функции и т.п.) и глобальные элементы.

Возвращаемое значение в теле функции может присваиваться или имени функции, или специальной предопределенной переменной Result. Например, тело приведенной выше функции FSum может иметь вид:

```
begin
 Fsum:= A*(X1+X2);
end;
```

или

```
begin
 Result:= A*(X1+X2);
end;
```

Результат работы обеих приведенных выше функций будет одинаковым. Но переменная Result имеет одно принципиальное отличие от имени функции. Имя функции - это не переменная. Оно может фигурировать только в левой части оператора присваивания, и не может входить ни в какие выражения. А Result - это обычная переменная. Поэтому, например, приведенный выше код можно было бы переписать следующим образом:

```
begin
 Result := (X1+X2);
 Result:= Result * A;
end;
```

С именем функции это сделать невозможно, так как оно не может встречаться в правой части оператора присваивания.

В Pascal Script существует предопределенная процедура exit, которая обеспечивает выход из функции или процедуры в любом месте тела программы. Например, приведенный выше текст мог бы быть записан так:

```
begin
 Result := (X1+X2);
 if (A = 1) then exit;
 Result:= Result * A;
end;
```

Прервать выполнение функции или процедуры можно также генерацией какого-то исключения.
Объявление процедуры практически ничем не отличается от объявления функции,

кроме того, что для процедуры не указывается возвращаемое значение:

procedure <имя процедуры>(<список параметров>);
<объявления локальных переменных, типов, констант, описания вложенных функций и процедур>
begin
 <операторы тела процедуры>
end;

Например:

procedure Pr1(S: string);
begin
 Self.Label1.Caption:=S;
end;

Вызов этой процедуры может иметь вид:

Pr1('Привет !');

## 3.16 Передача параметров в процедуры и функции

Список параметров, передаваемый в процедуры и функции, состоит из имен параметров и указаний их типа. Например, в объявлении:

procedure Pr(X1, X2: real; A: integer);

указано три параметра X1, X2, A и определены их типы. Вызов такой процедуры может иметь вид:

Pr(Y, X2, 5);

Это только один из способов передачи параметров в процедуру, называемый передачей по значению. Работает он так. В момент вызова функции в памяти создаются временные переменные с именами X1, X2, A и в них копируются значения аргументов Y, X2 и константы 5. На этом связь между аргументами и переменными X1, X2, A разрывается. Вы можете изменять внутри процедуры значения X1, X2 и A, но это никак не отразится на значениях аргументов. Аргументы при этом надежно защищены от непреднамеренного изменения своих значений вызванной процедурой или функцией.

Недостатком передачи параметров по значению является невозможность из функций и процедур изменять значения некоторых аргументов, что во многих случаях очень желательно.

Возможны и другие способы передачи параметров: как переменных (иногда это называется передачей по ссылке), как констант и как выходных величин. Эти способы

осуществляются указанием перед именем соответствующего параметра спецификатора var, const или out.

При передаче параметра по ссылке перед его именем в заголовке процедуры должно быть указано ключевое слово var. Например:

procedure Pr(var X1: real; X2: real; A: integer);

В этом случае не происходит копирования значения аргумента в локальную, временную переменную в процедуре. Процедура реально работает не с параметром, а со ссылкой - указателем на место хранения аргумента в памяти. Соответственно, в приведенном примере любые изменения параметра X1, произведенные в процедуре, в действительности относятся не к этому параметру, а к тому аргументу Y, который передан при вызове процедуры. Таким образом, ключевое слово var позволяет возвращать информацию из процедуры в вызвавшую его внешнюю процедуру.

Передача параметра как константы осуществляется заданием перед его именем ключевого слова const. Например:

procedure Prc(const X1:real; X2: real; A: integer);

Такая передача параметра почти идентична по своим последствиям обычной передаче по значению. Как и в том случае, значение аргумента невозможно изменить, изменяя параметр процедуры. Но есть одно существенное отличие: если при передаче параметра по значению вы можете в тексте процедуры изменять значение соответствующей ему локальной переменной, то при передаче параметра как константы это невозможно. При попытке в теле процедуры изменить значение параметра, переданного как константа, интерпритатор выдаст сообщение об ошибке.

Передача параметра как выходного осуществляется заданием перед его именем спецификатора out. Например:

procedure Prc(out X1:real; X2: real; A: integer);

Передача параметра как выходного очень похожа на передачу по ссылке. Отличие в том, что при этом не гарантируется передача в процедуру начального значения этого параметра. При этом память, занимаемая аргументом, в момент обращения к процедуре очищается. Таким образом, параметр, переданный как выходной, указывает на некоторое место в памяти, являющееся просто контейнером, куда процедура должна занести соответствующее выходное значение.

Реальный тип аргумента должен соответствовать типу, принятому в процедуре, но не обязательно точно. Например, если параметр - массив, то размер массива, определенный в процедуре, может превышать размер аргумента. Если же получается несоответствие типов (например, аргумент - real, а в процедуре он описан как integer), то никаких сообщений об ошибках и никаких исключений генерироваться не будет, но

результаты вычислений будут неверными.

Для параметров в объявлениях функций и процедур могут задаваться значения по умолчанию. Значение по умолчанию - это значение параметра в случае, если он не передан в вызове функции или процедуры. Таким образом, введение параметров со значениями по умолчанию равносильно разрешению передавать при вызове функции не все необходимые аргументы.
Значения по умолчанию задаются добавлением в конце объявления параметра знака равенства "=", после которого записывается константное выражение. Пусть, например, вы хотите написать функцию, которая рассчитывает суммарную силу, действующую на тело объемом V с плотностью P, погруженное в жидкость (например, воду) с плотностью PH2O. Как известно, формула, выражающая эту суммарную силу, направленную вверх (если ответ будет отрицательным, значит сила направлена вниз - тело тонет), следующая: F = G * V * (P - PH2O), где G - ускорение свободного падения.

Функцию, определяющую эту силу, можно описать следующим образом:

```
function Arh(V:double = 1; P:double = 0.5;
          PH2O:double = 1; G:double = 9.81): double;
begin
 Arh := G * V * (PH2O - P);
end;
```

Здесь всем параметрам даны значения по умолчанию. Объем V по умолчанию принят равным 1 м3, плотность тела P по умолчанию равна 0.5 т/м3 (плотность некоторых пород дерева), плотность воды PH2O принята по умолчанию равной 1 т/м3, а ускорение свободного падения G принято равным 9,81 м/с2.

Если при вызове функции параметр по умолчанию не указан, то в функцию автоматически передается его значение по умолчанию. Например, если вызвать приведенную функцию оператором

```
F := Arh();
```

то значение F будет равно силе при значениях всех параметров по умолчанию.

Аргументы по умолчанию должны быть самыми правыми (последними) аргументами в списке параметров функции. Если вызывается функция с двумя или более параметрами по умолчанию и если пропущенный параметр не является самым правым в списке, то все параметры справа от пропущенного тоже пропускаются.

Например, вызов той же функции оператором

```
F := Arh(2);
```

позволяет рассчитать силу, действующую на тело объемом 2 м3 при значениях всех остальных параметров по умолчанию. Вызов функции оператором

F := Arh(2,2.6);

позволяет рассчитать силу, действующую на алюминиевое (плотность 2.6 т/м3) тело объемом 2 м3 при значениях остальных параметров по умолчанию. Аналогично, задав при вызове три параметра, можно рассчитать силу, действующую на тело, погруженное в жидкость другой плотности, а задав все четыре параметра можно определить силу, действующую на тело при эксперименте, проводящемся не на уровне моря (при этом изменится ускорение свободного падения).

Этот пример показывает, что последними в списке параметров со значениями по умолчанию надо указывать те параметры, значения которых в реальных задачах чаще всего остаются равными заданным по умолчанию.

Пропускать при вызове можно только некоторое число последних параметров в списке. Например, нельзя вызвать функцию таким образом:

F := Arh(2,,1.1);  // Ошибочный вызов

Значения по умолчанию могут задаваться не всем, а только некоторым параметрам функции или процедуры. В этом случае действует правило, согласно которому все параметры со значениями по умолчанию должны размещаться в конце списка в объявлении функции или процедуры. Иначе говоря, если какой-то параметр имеет значение по умолчанию, то и все последующие параметры в списке должны иметь значения по умолчанию.

## 3.17 Стандартные функции

В скрипте можно использовать богатейший набор стандартных функций.

**Преобразование типов**

function IntToStr(i: Integer): String
Перевод целого в строку

function FloatToStr(e: Extended): String
Перевод числа с плавающей запятой в строку

function DateToStr(e: Extended): String
Перевод даты в строку

function TimeToStr(e: Extended): String
Перевод времени в строку

function DateTimeToStr(e: Extended): String
Перевод даты и времени в строку

function VarToStr(v: Variant): String
Перевод variant в строку

function StrToInt(s: String): Integer
Перевод строки в целое

function StrToFloat(s: String): Extended
Перевод строки в число с плавающей запятой

function StrToDate(s: String): Extended
Перевод строки в дату

function StrToTime(s: String): Extended
Перевод строки во время

function StrToDateTime(s: String): Extended
Перевод строки в дату и время

**Форматирование**

function Format(Fmt: String; Args: array): String
Форматирование

function FormatFloat(Fmt: String; Value: Extended): String
Форматирование числа с плавающей запятой

function FormatDateTime(Fmt: String; DateTime: TDateTime): String
Форматирование даты и времени

function FormatMaskText(EditMask: string; Value: string): string
Форматирование строки по маске

**Дата/время**

function EncodeDate(Year, Month, Day: Word): TDateTime
перевод года, месяца и дня в формат даты

procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word)
Перевод даты в года, месяц и день

function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime
Перевод часов, минут и секунд в формат времени

procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word)
Перевод времени в часы, минуты и секунды

function Date: TDateTime
Текущая дата

function Time: TDateTime
Текущее время

function Now: TDateTime
Текущие дата и время

function DayOfWeek(aDate: DateTime): Integer
День недели

function IsLeapYear(Year: Word): Boolean
Високосный год

function DaysInMonth(nYear, nMonth: Integer): Integer
Дней в месяце

**Строковые функции**

function Length(s: String): Integer
Длина строки

function Copy(s: String; from, count: Integer): String
Возвращает подстроку из строки с заданной позиции заданной длины

function Pos(substr, s: String): Integer
Позиция подстроки в строке

procedure Delete(var s: String; from, count: Integer)
Удаляет подстроку из строки с заданной позиции заданной длины

procedure Insert(s: String; var s2: String; pos: Integer)
Добавляет первую строку ко второй строке

function Uppercase(s: String): String
Перевод строки в верхний регистр

function Lowercase(s: String): String
Перевод строки в нижний регистр

function Trim(s: String): String
Удаляет окружающие пробелы из строки

function NameCase(s: String): String
Перевод первого символа в верхний регистр

function CompareText(s, s1: String): Integer
Сравнение строк

function Chr(i: Integer): Char
Возвращает символ с заданным номером

function Ord(ch: Char): Integer
Возвращает номер заданного символа

procedure SetLength(var S: String; L: Integer)
Устанавливает длину строки

**Математические функции**

function Round(e: Extended): Integer
Округление до ближайшего

function Trunc(e: Extended): Integer
Округление до меньшего

function Int(e: Extended): Integer
Возвращает целую часть

function Frac(X: Extended): Extended
Возвращает дробную часть

function Sqrt(e: Extended): Extended
Возвращает квадратный корень

function Abs(e: Extended): Extended
Возвращает модуль числа

function Sin(e: Extended): Extended
Синус

function Cos(e: Extended): Extended
Косинус

function ArcTan(X: Extended): Extended
Арктангенс

function Tan(X: Extended): Extended

Тангенс

function Exp(X: Extended): Extended
Экспонента

function Ln(X: Extended): Extended
Натуральный логарифм

function Pi: Extended
Число Пи

**Другие**

procedure Inc(var i: Integer; incr: Integer = 1)
Инкремент

procedure Dec(var i: Integer; decr: Integer = 1)
Декремент

procedure RaiseException(Param: String)
Генерация исключения

function GetLastExcept: string;
Описание последнего сгенерированного исключения.

procedure ShowMessage(Msg: Variant)
Вывод сообщения

procedure Randomize
Инициализация генератора псевдослучайных чисел

function Random: Extended
Генерация псевдослучайного числа

function ValidInt(cInt: String): Boolean
Проверка валидности целого в строке

function ValidFloat(cFlt: String): Boolean
Проверка валидности цикла с плавающей запятой в строкеValidDate(cDate: String): Boolean
Проверка валидности даты в строке

function CreateOleObject(ClassName: String): Variant
Создание OLE-объекта

function VarArrayCreate(Bounds: Array; Typ: Integer): Variant

Создание динамического массива

Как видите, некоторые функции и процедуры содержат параметры по умолчанию. Вы можете вызывать их так же, как и в Delphi:
Inc(a);
Inc(b, 2);
Вы можете подключить свои собственные процедуры и функции к скрипту - подробнее смотрите "Руководство программиста".

## 3.18 Обработка исключений

Обработка исключительных ситуаций производится с использованием блоков:

```
try
<операторы>
except
<операторы>
end
```

или

```
try
<операторы>
finally
<операторы>
end
```

При работе блока try..except: если при выполнении операторов раздела try генерируется исключение, выполнение этого раздела прерывается и управление передается операторам раздела except.
Блок try...finally позволяет защитить код, записанный в finally разделе, от исключений, которые в силу различных ошибок могут генерироваться операторами в разделе try. Операторы раздела finally будут выполняться независимо от того, было или не было сгенерировано исключение при выполнении операторов раздела try. Если где-то в разделе try произошла ошибка и сгенерировано исключение, то выполнение блока try прерывается и управление немедленно передается разделу finally. Даже если при выполнении операторов раздела finally случилась ошибка, операторы этого раздела все-таки выполняются до конца.

Обычно в раздел finally помещают операторы "зачистки", необходимые для того, чтобы "прибраться" в приложении: освободить динамически выделенную память, закрыть ненужные файлы, удалить временные файлы и т.д. Если не помещать такие операторы в защищенный раздел finally, то из-за прерывания вследствие возможных ошибок они могут оказаться не выполненными, что приведет к "утечке ресурсов".
Блоки try..finally и try..except могут использоваться совместно. Например:

```
try     {начало блока try...except}
................
  try    {начало блока try...finally}
  ................
  finally
  ................
  end;   {конец блока try...finally}
except
................
```

```
end;    {конец блока try...except}
```

Очень часто требуется сгенерировать исключение программно. Для этих целей используется процедура RaiseException. Процедуре передается один строковый параметр с описанием сгенерированного исключения.
Например: RaiseException('Abort').

Если в блоке try...except требуется получить описание исключения, следует воспользоваться функцией GetLastExcep:

```
try
//операторы сгенерировавшие исключение
except
 ShowMessage('Ошибка:'+GetLastExcept);
end;
```

## 3.19 Синтаксис Pascal

**Принятые обозначения.**

- служебные слова выделяются жирным шрифтом. Пример: **uses case**
- символы указаны в кавычках. Примеры: ':'  ';'  '<='
- имена синтаксических конструкций выделяются голубым цветом
- синтаксические структуры "по умолчанию" выделяются красным цветов.
- конструкции, которые могут встречаться в тексте 0 или 1 раз заключены в квадратные скобки.
- конструкции, которые могут встречаться в тексте 1 или более раз заключены в в круглые скобки.

Program = [**PROGRAM** Ident ';'] [UsesClause] Block '.'

UsesClause -> **USES** [String [(',' String)] ';'

Block = [DeclSection] CompoundStmt

DeclSection = (ConstSection | VarSection | ProcedureDeclSection)

ConstSection = **CONST** (ConstantDecl)

ConstantDecl = Ident '=' Expression ';'

VarSection = **VAR** (VarList) ';'

VarList = Ident (',' Ident) ':' TypeIdent [InitValue]

TypeIdent = Ident | Array

Array = **ARRAY** '[' ArrayDim [( ',' ArrayDim)] ']' **OF** Ident

ArrayDim = Expression '..' Expression | Expression

InitValue = '=' Expression

Expression = SimpleExpression [RelOp SimpleExpression]

SimpleExpression = ['-'] Term [ AddOp Term]

Term -> Factor [MulOp Factor]

Factor -> Designator | UnsignedNumber | String | '(' Expression ')' | **NOT** Factor | '[' SetConstructor ']'

SetConstructor = SetNode [',' SetNode]

SetNode -> Expression ['..' Expression]

RelOp = '>' | '<' | '<=' | '>=' | '<>' | '=' | **IN** | **IS**

AddOp = '+' | '-' | **OR** | **XOR**

MulOp = '*' | '/' | **DIV** | **MOD** | **AND** | **SHL** | **SHR**

Designator = ['@'] Ident ( '.' Ident | '[' ExprList ']' | '(' ExprList')' )

ExprList = Expression [',' Expression]

Statement = [SimpleStatement | StructStmt]

StmtList = Statement [';' Statement]

SimpleStatement = Designator | Designator ':=' Expression | **BREAK** | **CONTINUE** | **EXIT**

StructStmt = CompoundStmt | ConditionalStmt | LoopStmt | TryStmt | WithStmt

CompoundStmt = **BEGIN** StmtList **END**

ConditionalStmt = IfStmt | CaseStmt

IfStmt = **IF** Expression **THEN** Statement [**ELSE** Statement]

CaseStmt = **CASE** Expression **OF** CaseSelector [ ';' CaseSelector ] [**ELSE** Statement] [';'] **END**

CaseSelector = SetConstructor ':' Statement

LoopStmt = RepeatStmt | WhileStmt | ForStmt

RepeatStmt = **REPEAT** StmtList **UNTIL** Expression

WhileStmt = **WHILE** Expression **DO** Statement

ForStmt = **FOR** Ident ':=' Expression ToDownto Expression **DO** Statement

ToDownto = **TO** | **DOWNTO**

TryStmt = **TRY** StmtList FinallyExcept StmtList **END**

FinallyExcept = **FINALLY** | **EXCEPT**

WithStmt = **WITH** Designator [',' Designator] **DO** Statement

ProcedureDeclSection = ProcedureDecl | FunctionDecl

ProcedureDecl = ProcedureHeading ';' Block ';'

ProcedureHeading = **PROCEDURE** Ident [FormalParameters]

FunctionDecl = FunctionHeading ';' Block ';'

FunctionHeading = **FUNCTION** Ident [FormalParameters] ':' Ident

FormalParameters = '(' FormalParam [';' FormalParam] ')'

FormalParm -> [**VAR** | **CONST**] VarList

## 3.20 Синтаксис C++

Program = [UsesClause] (DeclSection) CompoundStmt

UsesClause = '#' **INCLUDE** String [(',' String)]

DeclSection = ConstSection | ProcedureDeclSection | VarStmt

ConstSection = '#' **DEFINE** ConstantDecl

ConstantDecl = Ident Expression

VarStmt = Ident Ident [Array] [InitValue]

ArrayDef = '[' ArrayDim [(',' ArrayDim)] ']'

ArrayDim = Expression

InitValue = '=' Expression

Expression = SimpleExpression [RelOp SimpleExpression]

SimpleExpression = ['-'] Term [AddOp Term]

Term = Factor [MulOp Factor]

Factor = Designator | UnsignedNumber | String | '(' Expression ')' | '!' Factor | '[' SetConstructor ']' | NewOperator

SetConstructor = SetNode [(',' SetNode)]

SetNode = Expression ['..' Expression]

NewOperator = **NEW** Designator

RelOp = '>' | '<' | '<=' | '>=' | '!=' | '==' | **IN** | **IS**

AddOp = '+' | '-' | '||' | '^' |

MulOp = '*' | '/' | '%' | '&&' | '<<' | '>>'

Designator = ['&'] Ident [( '.' Ident | '[' ExprList ']' | '(' ExprList')' )]

ExprList = Expression [',' Excpression]

Statement = SimpleStatement ';' | StructStmt | EmptyStmt

EmptyStmt = ';'

StmtList = (Statement)

SimpleStatement -> DeleteStmt | AssignStmt | VarStmt | CallStmt | ReturnStmt | BreakSmpt

BreakSmtp = **BREAK** | **CONTINUE** | **EXIT**

DeleteStmt = **DELETE** Designator

AssignStmt = Designator [ '+' | '-' | '*' | '/' ] '=' Expression

CallStmt = Designator [ '++' | '--' ]

ReturnStmt = **RETURN** [Expression]

StructStmt = CompoundStmt | ConditionalStmt | LoopStmt | TryStmt

CompoundStmt = '{' [StmtList] '}'

ConditionalStmt = IfStmt | CaseStmt

IfStmt = **IF** '(' Expression ')' Statement [**ELSE** Statement]

CaseStmt = **SWITCH** '(' Expression ')' '{' (CaseSelector)... [**DEFAULT** ':' Statement] '}'

CaseSelector = **CASE** SetConstructor ':' Statement

LoopStmt = RepeatStmt | WhileStmt | ForStmt

RepeatStmt = **DO** Statement [';'] **WHILE** '(' Expression ')'  ';'

WhileStmt = **WHILE** '(' Expression ')' Statement

ForStmt = **FOR** '(' ForStmtItem ';' Expression ';' ForStmtItem ')' Statement

ForStmtItem = AssignStmt | VarStmt | CallStmt | Empty

TryStmt = **TRY** CompoundStmt  CompoundStmt

FinallyExcept = **FINALLY** | **EXCEPT**

FunctionDecl = FunctionHeading CompoundStmt

FunctionHeading = Ident Ident [FormalParameters]

FormalParameters = '(' [FormalParam [( ';' FormalParam)] ')'

FormalParam = TypeIdent (['&'] Ident [InitValue] [',' FormalParam])

## 3.21 Синтаксис JScript

Program = Statements

Statements = (Statement)

Block = '{' Statements '}'

ImportStmt = **IMPORT** String [(',' String)]

VarStmt = **VAR** VarDecl [( ',' VarStmt)]

VarDecl = Ident [Array] [InitValue]

Array = '[' ArrayDim [(',' ArrayDim)] ']'

ArrayDim = Expression

InitValue = '=' Expression

Expression = SimpleExpression [RelOp SimpleExpression]

SimpleExpression = ['-'] Term [AddOp Term]

Term = Factor [MulOp Factor]

Factor = Designator | UnsignedNumber | String | '(' Expression ')' | '!' Factor | NewOperator | '<' FRString '>'

SetConstructor = SetNode [(',' SetNode)]

SetNode = Expression ['..' Expression]

NewOperator = **NEW** Designator

RelOp = '>' | '<' | '<=' | '>=' | '!=' | '==' | **IN** | **IS**

AddOp = '+' | '-' | '||' | '^'

MulOp = '*' | '/' | '%' | '&&' | '<<' | '>>'

Designator = ['&'] Ident [( '.' Ident | '[' ExprList ']' | '(' [ExprList] ')' )]

ExprList = Expression [ ',' Expression ]

Statement = StatmentVar [';']

StatemrntVar = AssignStmt | CallStmt | BreakStmt | ContinueStmt | DeleteStmt | DoWhileStmt | ForStmt | FunctionStmt | IfStmt | ImportStmt | ReturnStmt | SwitchStmt | VarStmt | WhileStmt | WithStmt | Block

BreakStmt = **BREAK**

ContinueStmt = **CONTINUE**

DeleteStmt = **DELETE** Designator

AssignStmt = Designator ['+'|'-'|'*'|'/'] '=' Expression

CallStmt = Designator ['+"+'|'-"-']

ReturnStmt = **RETURN** [Expression]

IfStmt = **IF** '(' Expression ')' Statement [**ELSE** Statement]

SwitchStmt = **SWITCH** '(' Expression ')' '{' (CaseSelector) [**DEFAULT** ':' Statement] '}'

CaseSelector = **CASE** SetConstructor ':' Statement

DoWhileStmt = **DO** Statement [';'] **WHILE** '(' Expression ')' ';'

WhileStmt = **WHILE** '(' Expression ')' Statement

ForStmt = **FOR** '(' ForStmtItem ';' Expression ';' ForStmtItem ')' Statement

ForStmtItem = AssignStmt | CallStmt | VarStmt | Empty

TryStmt = **TRY** CompoundStmt FinallyExcept CompoundStmt

FinallyExcept = **FINALLY** | **EXCEPT**

FunctionStmt = FunctionHeading Block

FunctionHeading = **FUNCTION** Ident FormalParameters

FormalParameters = '(' FormalParam [( ',' FormalParam)] ')'

FormalParam = ['&'] Ident

WithStmt = **WITH** '(' Designator ')' Statement

## 3.22 Синтаксис Visual Basic

Program = Statements

Statements = (**EOL** | StatementList **EOL**)

StatementList = Statement [( ':' Statement )]

ImportStmt = **IMPORTS** String [( ',' String )]

DimStmt = **DIM** VarDecl [( ',' VarDecl )]

VarDecl = Ident [Array] [AsClause] [InitValue]

AsClause = **AS** Ident

Array = '[' ArrayDim [( ',' ArrayDim )] ']'

ArrayDim = Expression

InitValue = '=' Expression

Expression = SimpleExpression [RelOp SimpleExpression]

SimpleExpression = ['-'] Term [AddOp Term]

Term = Factor [MulOp Factor]

Factor = Designator | UnsignedNumber | String | '(' Expression ')' | **NOT** Factor | NewOperator | '<' FRString '>'

SetConstructor -> SetNode [( ',' SetNode )]

SetNode = Expression ['..' Expression]

NewOperator = **NEW** Designator

RelOp = '>' | '<' | '<=' | '>=' | '<>' | '=' | IN | IS

AddOp = '+' | '-' | '&' | OR | XOR

MulOp = '*' | '/' | '\' | MOD | AND

Designator = [**ADDRESSOF**] Ident [( '.' Ident | '[' ExprList ']' | '(' [ExprList] ')' )]

ExprList = Expression ([ ',' Excpression ])

Statement = BreakStmt | CaseStmt | ContinueStmt | DeleteStmt | DimStmt | DoStmt | ExitStmt | ForStmt | FuncStmt | IfStmt | ImportStmt | ProcStmt | ReturnStmt | SetStmt | TryStmt | WhileStmt | WithStmt | AssignStmt | CallStmt

BreakStmt = **BREAK**

ContinueStmt = **CONTINUE**

ExitStmt = **EXIT**

DeleteStmt = **DELETE** Designator

SetStmt = **SET** AssignStmt

AssignStmt = Designator ['+'|'-'|'*'|'/'] '=' Expression

CallStmt = Designator ['+"+'|'-"-']

ReturnStmt = **RETURN** [Expression]

IfStmt = **IF** Expression **THEN** ThenStmt

ThenStmt = **EOL** [Statements] [ElseIfStmt | ElseStmt] **END IF** | StatementList

ElseIfStmt = **ELSEIF** Expression **THEN** ElseIfStmtBlock

ElseIfStmtBlock = **EOL** [Statements] [ElseIfStmt | ElseStmt] | Statement

ElseStmt = **ELSE** ElseStmtBlock

ElseStmtBlock = **EOL** Statements | Statement

CaseStmt = **SELECT CASE** Expression **EOL** (CaseSelector) [**CASE ELSE** ':' Statements] **END SELECT**

CaseSelector = **CASE** SetConstructor ':' Statements

DoStmt = **DO** [Statements] **LOOP** UntilWhile Expression

UntilWhile = **UNTIL** | **WHILE**

WhileStmt = **WHILE** Expression [Statements] **WEND**

ForStmt = **FOR** Ident '=' Expression **TO** Expression [**STEP** Expression] **EOL** [Statements] **NEXT**

TryStmt = **TRY** Statements FinallyCatch [Statements] **END TRY**

FinallyCatch = **FINALLY** | **CATCH**

WithStmt = **WITH** Designator **EOL** Statements **END WITH**

ProcStmt = **SUB** Ident [FormalParameters] **EOL** [Statements] **END SUB**

FuncStmt = **FUNCTION** Ident [FormalParameters] [AsClause] **EOL** [Statements] **END FUNCTION**

FormalParameters = '(' FormalParam [( ',' FormalParam )] ')'

FormalParm = [**BYREF** | **BYVAL**] VarList

# Глава IV

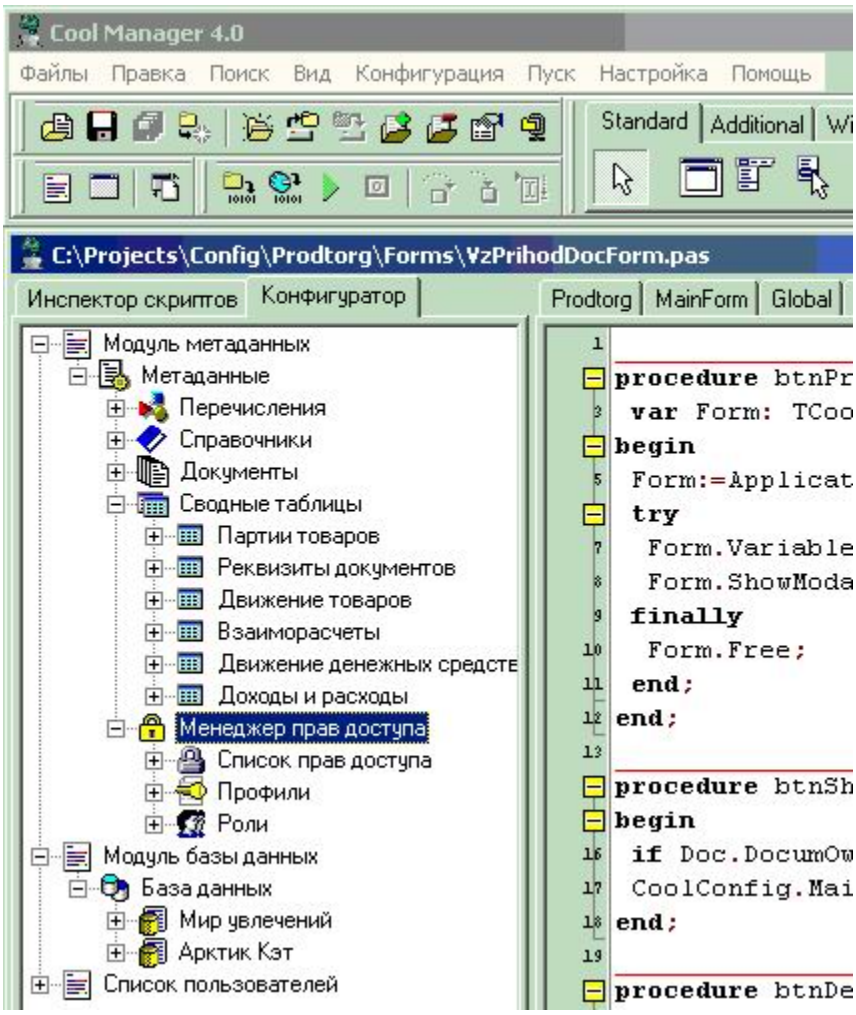
## Библиотека CoolLibrary

# 4 Библиотека CoolLibrary

## 4.1 Описание библиотеки CoolLibrary

Помимо интегрированной среды разработки ИСР, в стандартный комплект поставки системы CoolManager включена проблемно ориентированная библиотека классов CoolLibrary. CoolLibrary содержит инструментарий, необходимый разработчику для построения бизнес-приложений любой степени сложности. Библиотека реализована в виде отдельного bpl пакета (CoolLib**X**.bpl). ИСР CoolManager и библиотека CoolLibrary являются "платформой" для разработки конфигурации прикладной задачи - "бизнес-приложения". Платформа представляет собой контейнер, в котором функционирует бизнес-приложение. С одной стороны платформу можно считать фундаментом для разработки приложений, с другой - средой исполнения. Кроме того, платформа содержит инструментарий, необходимый для разработки, администрирования и поддержки приложений. Разработанное с помощью платформы бизнес-приложение является самостоятельной сущностью и может выступать в качестве отдельного программного продукта, но полностью опирается на технологии платформы. Важно, что платформа предоставляет разработчику определенную модель, изолирующую его от понятий и подробностей более низкоуровневых технологий и хранилищ баз данных. Платформа, так же, позволяет разработчику использовать одну систему типов данных и для взаимодействия с БД, и для реализации бизнес-логики, и для построения интерфейсных решений.
Одним из существенных преимуществ четкого разграничения между платформой и бизнес-приложением является высокий уровень адаптируемости типовых решений. Следует заметить, что именно для систем автоматизации учета особенно важна возможность эффективного изменения готового решения разработчиком, не учавствовавшим в его создании. В индустрии разработки бизнес-приложений, существенная часть разработчиков не создает программы "с чистого листа", а дорабатывает и развивает типовые решения. Это обстоятельство определяет особые требования к обеспечению наглядности и простоте понимания разработчиком существующих решений и максимально учитывается во всех механизмах платформы.
Выделение бизнес-приложения как самостоятельного элемента, фактически, позволяет создать замкнутый технологический цикл разработки, распространения и поддержки разнообразных прикладных систем.

Библиотека CoolLibrary содержит набор метаклассов для создания конфигурации прикладной задачи:




а так-же набор компонентов используемых при разработке визуальных форм:


## 4.2 Метаданные прикладной системы

В ИСР Cool Manager прикладное решение не пишется в прямом смысле на языке программирования. Язык программирования, конечно, используется, но только там, где это действительно необходимо. В основе бизнес-приложения лежат метаданные. Они представляют собой структурированное декларативное его описание. Метаданные образуют иерархию объектов, из которых формируются все составные части прикладной системы и которые опеределяют все аспекты её поведения. Фактически, при работе бизнес-приложения платформа "проигрывает" (интерпритирует) метаданные, обеспечивая всю необходимую функциональность.

Метаданными описываются структуры данных, состав типов, связи между объектами, особенности их поведения, разграничение прав доступа и т.д. В метаданных,

фактически, сосредоточены сведения не только о том, "что хранить в базе данных",но и о том, "зачем" хранится та или иная информация, какова её роль в системе, и как связаны между собой информационные массивы.
Что дает такой подход к построению бизнес-приложения? Во-первых, при описании метаданных широко используется визуальное редактирование. Это позволяет свести существенную часть разработки к визуальному проектированию, не требующему написания программного кода. Кроме того, описывая прикладное решение в терминах метаданных, разработчик "сообщает" платформе много очень полезной информации, которую та может эффективно использовать в самых различных целях. На основе метаданных система автоматичесики "выстраивает" большую часть механизмов и объектов, обеспечивающих функционирование прикладного решения.
Для создания конфигурации бизнес-приложения используется конфигуратор метаданных. Он позволяет создать взаимосвязанную модель метаданных, представленную в виде древовидного иерархического списка. Конфигуратор системы Cool Manager имеет открытую архитектуру, это означает, что все метаклассы используемые при построении конфигурации подключаются к ИСР динамически, в момент загрузки bpl пакетов, в которых они описаны. Так, например, пакет CoolLib предоставляет в распоряжение разработчика большой набор проблемно-ориентированных метаклассов. Все метаклассы содержащиеся в пакете можно разделить на четыре группы:

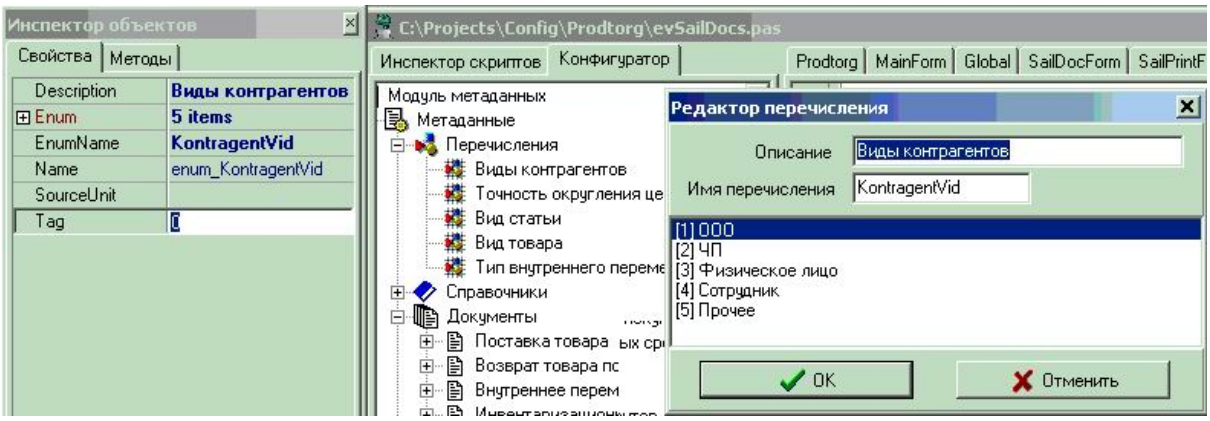
- Метаклассы для описания объектов используемых бизнес-приложением
- Метаклассы для описания распределенной структуры БД бизнес-приложения
- Метаклассы для описания системы авторизации пользователей
- Метаклассы для организации обмена данными между БД территориально удаленных объектов

## 4.3 Метаклассы объектов

При описании объектов прикладной задачи разработчик может оперировать следующими метаклассами:
- перечисление
- справочник
- документ
- сводная таблица

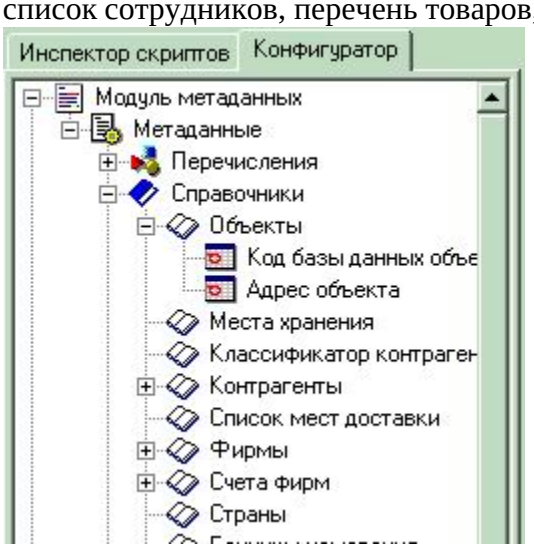
### Перечисление.

Перечисления служат для описания перечислимых типов данных. Перечисления используются в качестве типов реквизитов справочников, документов и сводных таблиц:

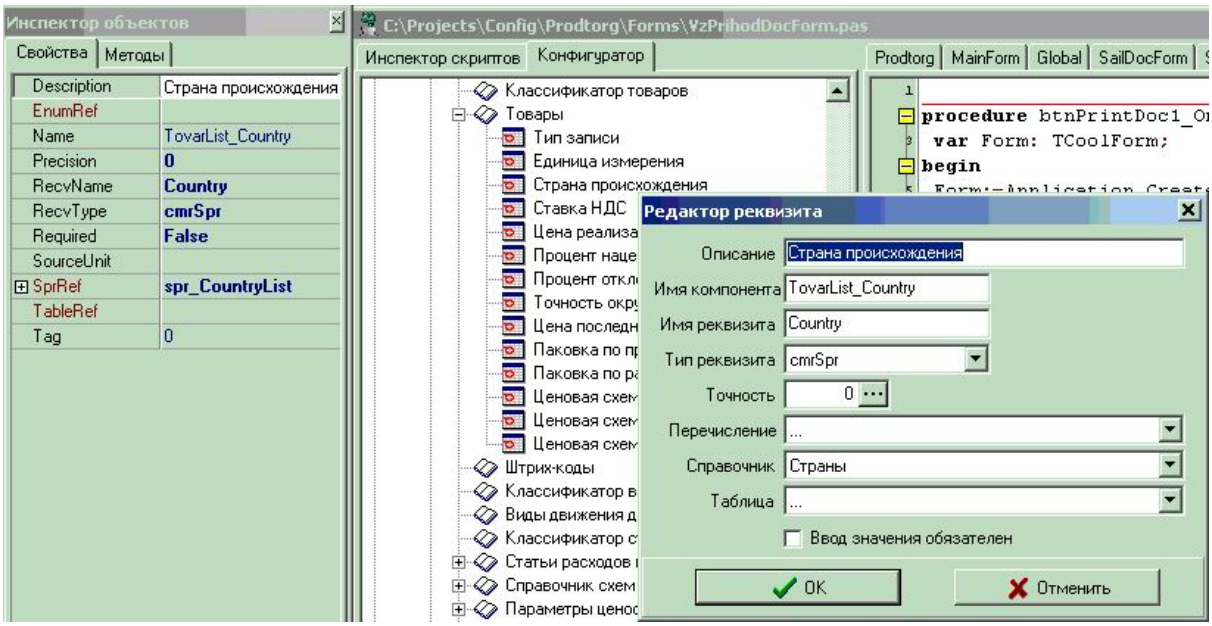



## Справочник.

Объекты метакласса "Справочник" позволяют хранить в информационной базе данные, имеющие одинаковую структуру и списочный характер. Это может быть, например, список сотрудников, перечень товаров, список поставщиков или покупателей.

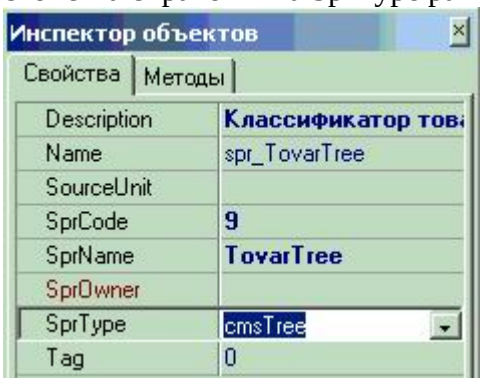



Каждый элемент справочника характеризуется кодом и наименованием. Система поддерживает режим автоматической нумерации элементов, при котором она самостоятельно может генерировать код для нового элемента справочника. Помимо кода и наименования, каждый элемент справочника, как правило, содержит некоторую дополнительную информацию, которая подробно описывает этот элемент. Например, для товара это может быть информация об артикуле, номере государственной таможенной декларации, стране происхождения и т.п. Набор такой информации является одинаковым для всех элементов конкретного справочника, и для ее хранения служат реквизиты справочника:

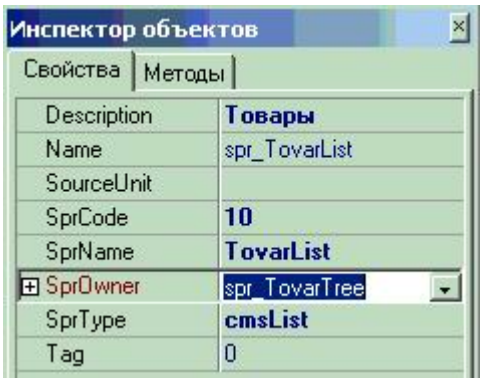



Для редактирования свойств реквизита может использоваться стандартный Object Inspector или специализированный редактор реквизита. Такой подход является общим практически для всех метаобъектов представленных в конфигураторе.

Справочники могут поддерживать иерархическое расположение элементов. Например, в справочнике "Классификатор товаров" могут быть созданы группы: Бытовая техника, Обувь, Продукты и т.д., в которых располагаются элементы, относящиеся к этим группам. Кроме того группы справочника могут включать в себя и другие группы, создавая тем самым многоуровневую иерархическую структуру. Для того, чтобы определить справочник с иерархической структурой, следует установить значение свойства справочника SprType равным cmsTree:

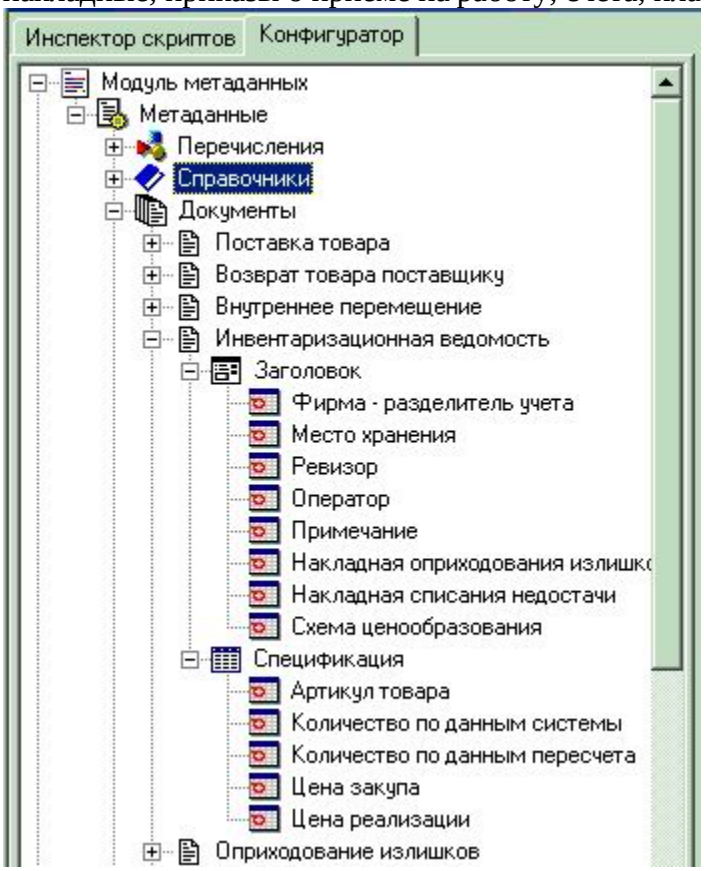



Разные справочники могут находиться в состоянии подчинения, т.е. элементы одного справочника могут быть подчинены элементам или группам другого справочника. Например, элементы справочника "Товары" подчинены соотвествующим записям в справочнике "Классификатор товаров". Поэтому, справочник "Классификатор товаров" является владельцем справочника "Товары":




## Документ.

Объекты метакласса Документ позволяют хранить в базе данных информацию о совершенных хозяйственных операциях. Это могут быть, например, приходные накладные, приказы о приеме на работу, счета, платежные поручения и т.д.

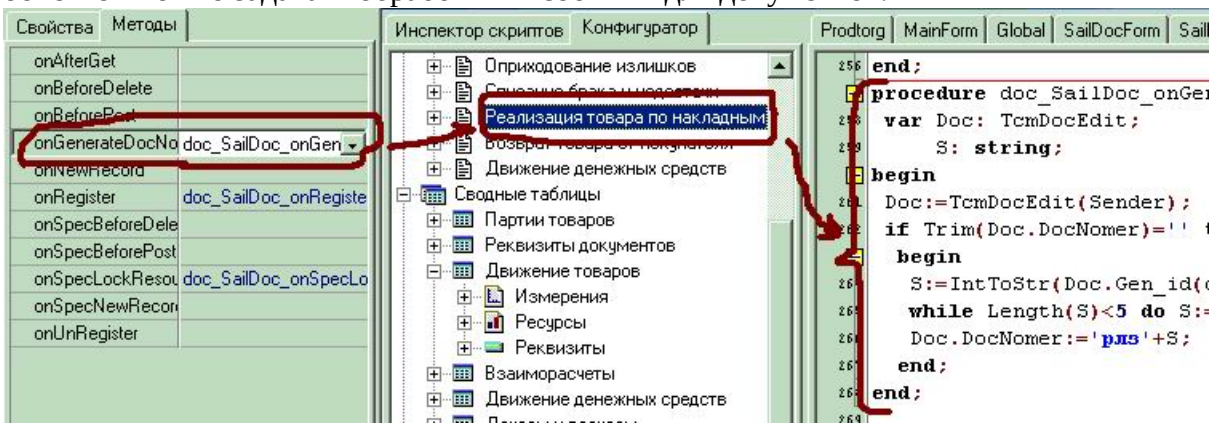



Каждый документ характеризуется кодом, номером, датой и временем. Система поддерживает режим автоматической нумерации документов, при котором она самостоятельно генерирует уникальный код для нового документа.

Помимо номера, даты и времени, каждый документ, как правило, содержит некоторую дополнительную информацию, которая подробно описывает этот документ. Например, для приходной накладной это может быть информация о поставщике товаров, складе, на который приходуются товары и т.п. Набор такой информации является одинаковым

для всех документов конкретного вида, и для ее хранения служит заголовок документа. Кроме этого, каждый документ может содержать некоторый набор информации, которая одинакова по своей структуре, но различна по количеству, для разных документов. Например, для приходной накладной это может быть информация о товарах, поступивших на предприятие (наименование, количество, цена и т.д.). Для хранения подобных данных служит спецификация документа.

Для документа определен ряд событий, которые могут возникать на различных этапах его обработки. Так например, при генерации номера документа вызывается событие onGenerateDocNomer, при проведении документа onRegister и т.д. Используя инспектор объектов можно задавать обработчики событий для документов:



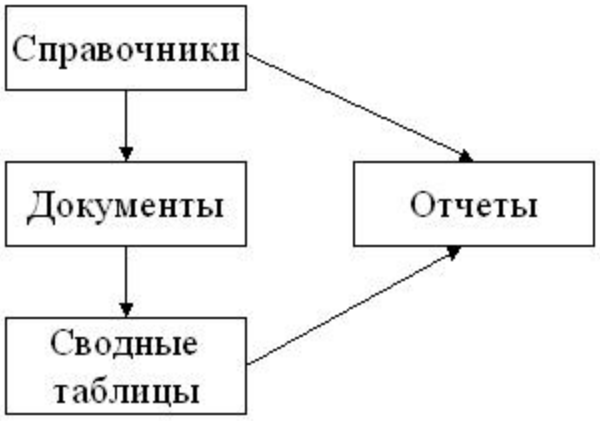

## Сводная таблица.

Сводные таблицы позволяют хранить в базе данных произвольные данные в разрезе нескольких измерений. Так, например, таблица товарных остатков может хранить информацию о движении товаров в разрезе фирм разделителей учета, мест хранения, номенклатурных кодов товара и партий товара:


Измерения таблицы описывают разрезы, в которых хранится информация, а ресурсы непосредственно содержат хранимую информацию. Вместе с каждой записью, находящейся в таблице, можно хранить дополнительную произвольную информацию. Для этого служат реквизиты таблицы.

Таблицы подразделяются на два типа - регистрации и накопительные.
При вставке записи в таблицу регистрации автоматически расчитываются оперативные остатки ресурсов таблицы в разрезе заданных измерений, а так же обороты за период. Накопительные таблицы используются только для хранения информации без автоматического расчета оперативных остатков и оборотов. Тип таблицы определяется при описании таблицы в конфигураторе. Так к накопительным автоматически относятся все таблицы для которых не задано не одного измерения и/или ресурса.
Запись информации в сводные таблицы возможна только в обработчике события onRegister документа, т.е. при его проведении. При распроведении документа, вся информация внесенная в сводные таблицы в момент проведения, автоматически удаляется.

Построение бизнес-приложений с использованием платформы Cool Manager подразумевает ввод информации "от документа". В упрощенном виде принцип работы бизнес-приложения можно изобразить следующей схемой:




Информация, введенная в справочники, используется при создании документов. При проведении документа, вносятся данные в сводные таблицы. Информация сводных таблиц и справочников используется при построении отчетов.

## 4.4 Метаклассы баз данных

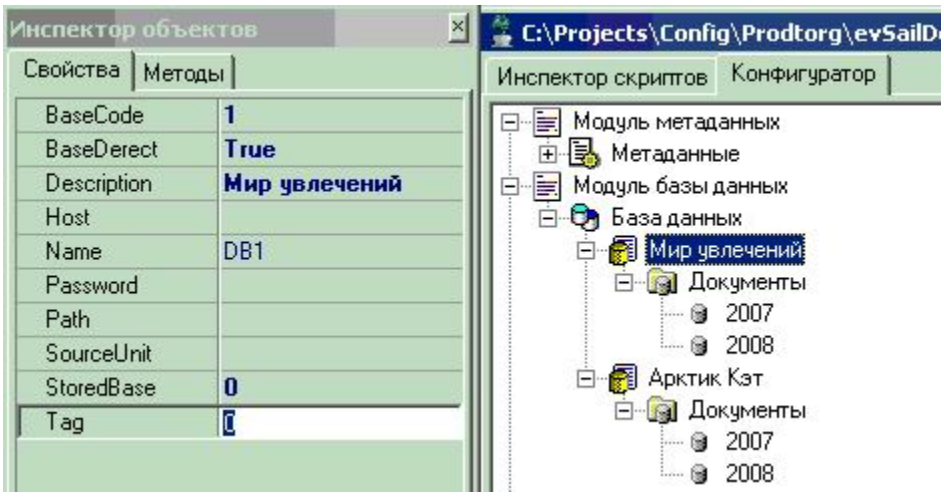
Под термином "база данных" в системе CoolManager подразумевается не отдельная база InterBase сервера (обычно файл с расширением gdb), а набор логически взаимосвязанных баз. База любого объекта (здесь термин "объект" обозначает территориально обособленное подразделение предприятия: офис, магазин, склад и т.д.) состоит из нескольких InterBase баз, различных по своему назначению и структуре хранимых данных. Можно выделить два типа баз используемых каждым объектом:

- База справочников - содержит все справочники системы и историю их редактирования, а так-же оперативные остатки сводных таблиц. Создается одна база на объект.
- Базы документов - содержат документы созданные в системе, а так же движения и обороты сводных таблиц. База документов создается для календарного периода, т.е.

одного года или нескольких лет.

Одним из основных вопросов связанных с обслуживанием баз данных, является проблема физического роста размера файлов БД. Чем интенсивнее работа с базой, тем быстрее растет её размер. Поэтому, как правило, все системы учета содержат специальные утилиты для "обрезки" или "сжатия" файлов БД. Обычно под процедурой "сжатия" подразумевается удаление из базы информации за заданный период. В системе CoolManager применяется совершенно иной подход к решению данной проблемы. Главный прирост объема любой базы дают создаваемые в ней документы и информация вносимая в сводные таблицы. Поэтому, для хранения документов в CoolManager предназначен специальный тип баз - базы документов. Подобная база хранит документооборот за один календарный год или несколько лет. Когда, с точки зрения системного администратора, размер базы достигает критического размера, просто создается база документов для нового периода. Все используемые системой базы описываются в конфигураторе, и доступны в любой момент работы с системой. Платформа сама определяет какую базу следует использовать для извлечения и сохранения данных введенных пользователем. Соединение с нужной базой устанавливается автоматически, в момент первого обращения. С точки зрения разработчика, СУБД системы CoolManager составляет единое целое. Нет никакой необходимости задумываться о том, из какой базы извлекается та или иная информация, всю эту работу ИСР выполняет в автоматическом режиме. Очень важным моментом является то, что физически, различные файлы СУБД InterBase, могут располагаться на разных компьютерах локальной сети. Но с точки зрения платформы, все они представляют единое целое - базу данных системы Cool Manager.
Кроме "главной" базы, СУБД "объекта" может содержать любое количество "зеркал" других "объектов" предприятия. Под "зеркалом" подразумевается копия базы удаленного объекта, данные в которую поступают посредством механизмов репликации встроенных в платформу Cool Manager. Если базы "зеркала" описаны в конфигураторе, то с точки зрения платформы они опять же представляют единое целое с главной базой "объекта". Единственным отличием является то, что информация хранимая в "зеркалах" доступна только для чтения и не подлежит изменению. При передаче информации из базы в "зеркало" формируется файл обмена в XML формате, так называемый "пакет". Далее "пакет" может быть передан любым удобным для персонала предприятия способом - на мобильном носителе информации, по выделенному каналу или посредством электронной почты. Обмен данными может осуществляться в автоматическом режиме, переодичность обмена варьируется в пределах от нескольких дней до нескольких минут.

Пример описания структуры базы данных в конфигураторе:

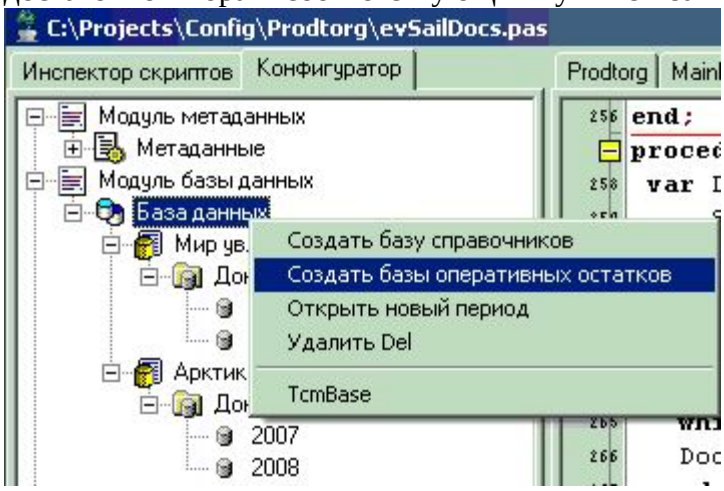



Описывается главная база объекта и все "зеркала" подключенные к системе. Структура таблиц, создаваемых в базе, тесно связана с описанием метаданных (справочников, документов, сводных таблиц).

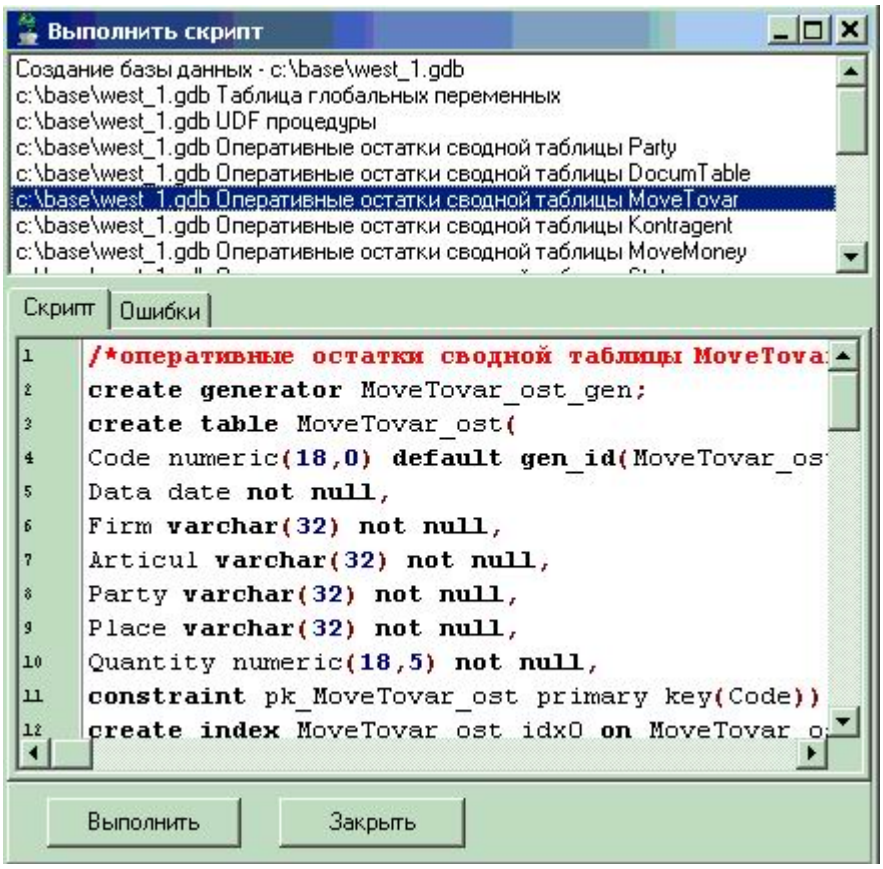
Последовательнось создания новой базы следующая:

- Создается описание объектов бизнес-приложения
- Создается структура модулей базы данных
- Создаются базы данных.

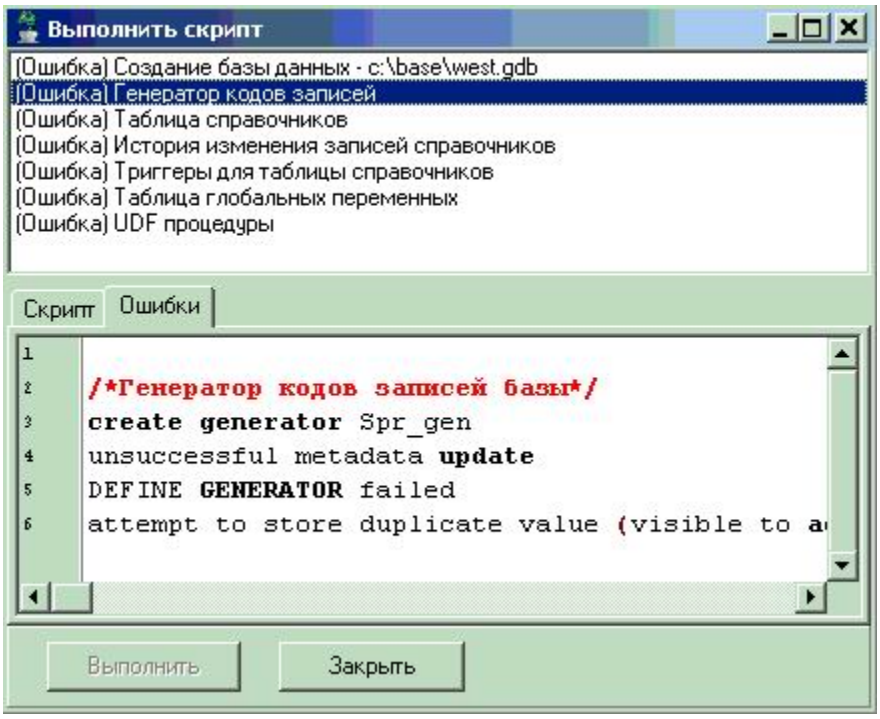
Процесс создания баз данных полностью автоматизирован. Для создания базы достаточно выбрать соответствующий пункт связанного с базой PopupMenu:




После того, как выбран нужный пункт меню, на экран выводится форма с подробным описанием всех создаваемых баз и объектов:




В верхней части формы расположен список выполняемых скриптов. В нижней части текстовый редактор, в котором отображается текст выбранного скрипта. Для создания баз(ы) достаточно нажать кнопку "Выполнить". Если базы(а) успешно созданы, на экран будет выведено соответствующее сообщение. Если при выполнении скриптов произошли какие либо ошибки, то некорректно выполненные скрипты будут помечены в общем списке, а на закладке "Ошибки" можно посмотреть подробную информацию об ошибках, возникших в ходе выполнения скрипта:

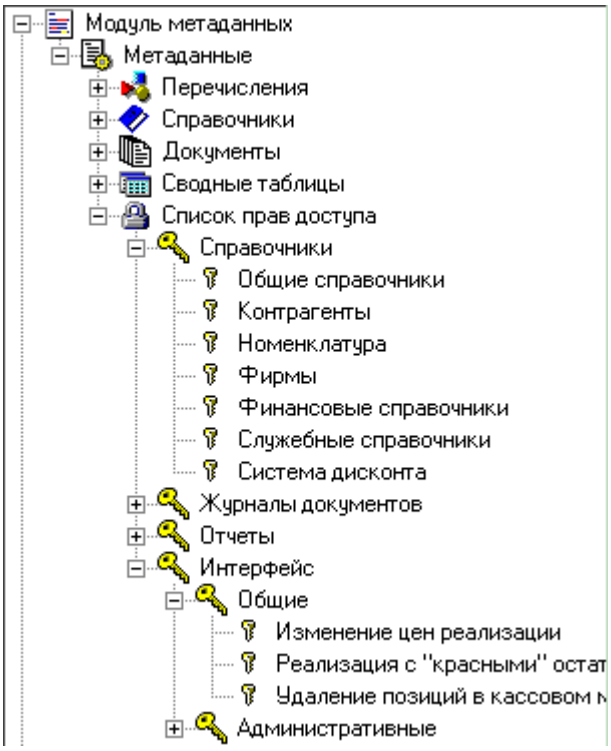



## 4.5 Метаклассы прав доступа

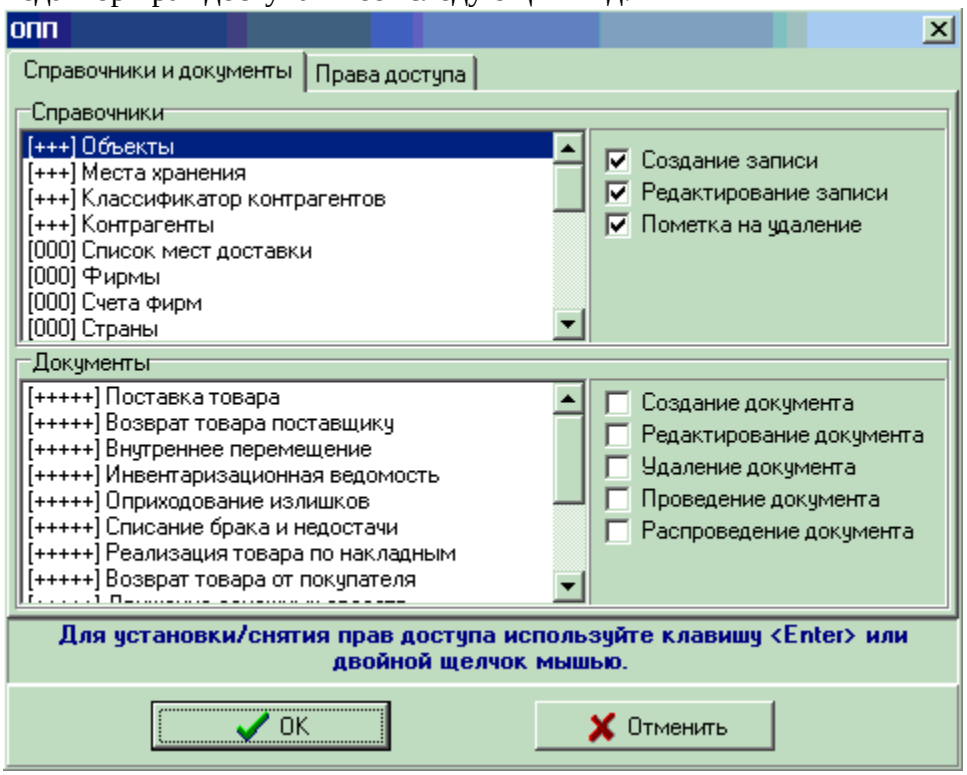
Права доступа к объектам бизнес-приложения, определяемые в конфигурации, делятся на три группы:

- Права доступа к справочникам
- Права доступа к документам
- Права доступа к объектам конфигурации

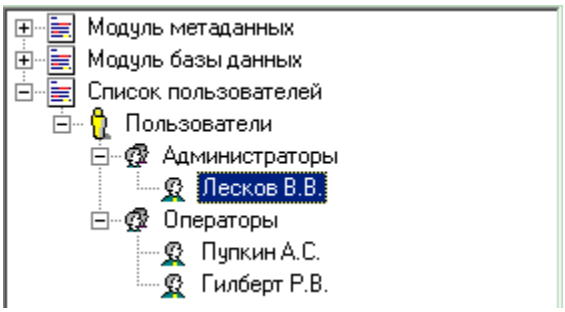
Пример создания списка прав доступа в конфигураторе:



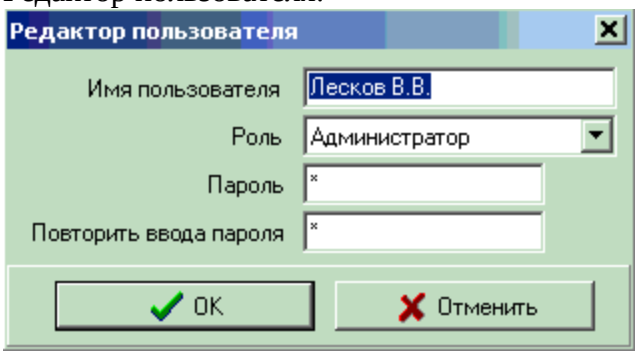

Редактор прав доступа имеет следующий вид:




Пользователи и пароли так-же описываются в конфигураторе:



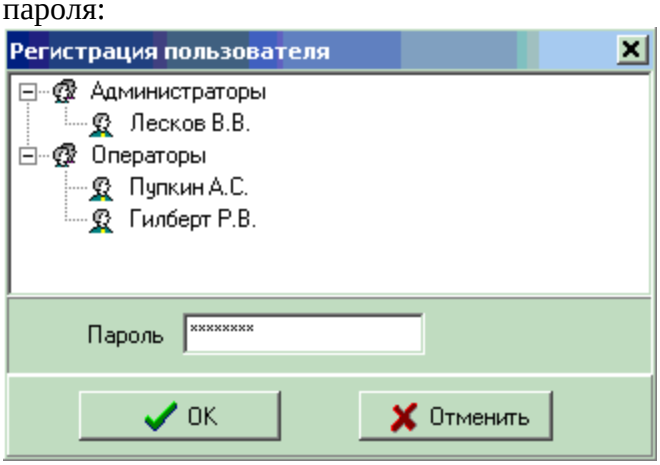

Редактор пользователя:




При создании нового пользователя указываются:

- Имя пользователя - свойство TcmUnitComponent.Description
- Роль - роль пользователя
- Пароль - пароль пользователя

Если бизнес-приложение использует систему авторизации пользователей, то при запуске приложения автоматически выводится диалог для выбора пользователя и пароля:




## 4.6 Метаклассы межбазового обмена

Как было сказано ранее под термином "база данных" в ИСР CoolManager подразумевается совокупность СУБД территориально удаленных объектов предприятия. СУБД объекта состоит из нескольких баз данных работающих под управлением сервера InterBase. Если системе CoolManager требуется получить доступ к

базам территориально удаленных объектов, в конфигуратор добавляется информация о локальных копиях БД других объектов. В локальные копии поступает информация об изменениях сделанных в удаленных базах. Система CoolManager работает как с главной базой объекта, так и с данными хранящимися в локальных копиях удаленных баз. Когда формируется запрос к БД, информация может извлекаться как из главной базы, так и из локальных копий удаленных баз. Но вся эта работа происходит "за кадром", с точки зрения разработчика прикладной задачи, обращение происходит к единой БД. Описанная выше архитектура базы данных декларируется на уровне метаклассов технологической платформы ИСР CoolManager. В связи с этим правила и механизмы межбазового обмена данными так же описываются в виде метаклассов технологической платформы. Можно выделить два типа межбазового обмена:

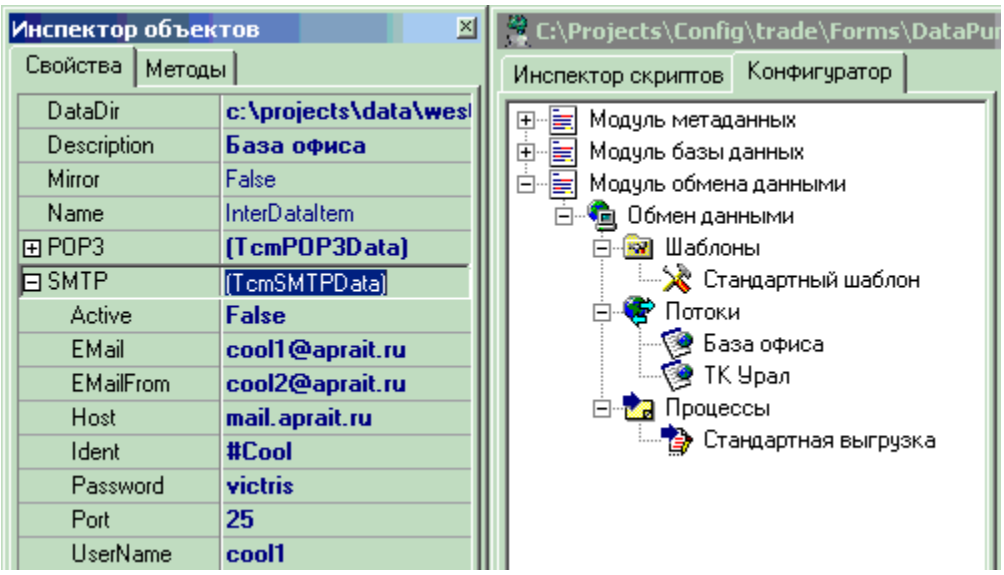
- Обмен данными справочников - справочники могут редактироваться в базе любого объекта, затем внесенные изменения должны быть синхронизированы со всеми удаленными базами.
- Репликация документов - документы созданные в базе объекта передаются во все удаленные копии БД этого объекта. Данные в удаленных копиях БД доступны для просмотра, но не подлежат изменению.

Межбазовый обмен данными носит дискретный характер, изменения передаются в виде XML файлов. Такие файлы обмена называются пакетами. Пакеты могут создаваться в автоматическом режиме через равные промежутки времени. Переодичность создания пакетов определяется исходя из особенностей прикладной задачи и может варьироваться в интервале от нескольких дней до нескольких минут. Между удаленными объектами обмен пакетами производится посредством электронной почты. Процесс межбазового обмена данными состоит из четырех фаз:

- В базе объекта "А" формируется пакет для базы объекта "Б".
- Объект "А" отправляет электронное письмо с пакетом (пакет передается в виде ZIP архива).
- Объект "Б" получает электронное письмо и извлекает из письма пакет.
- Пакет загружается в базу объекта "Б".

Технологическая платформа определяет три типа метаклассов для описания процесса межбазового обмена:

- **Шаблон пакета** - описание данных подлежащих включению в пакет.
- **Поток** - последовательность пакетов, пересылаемых от одной базы к другой.
- **Процесс** - процедура формирования пакетов.



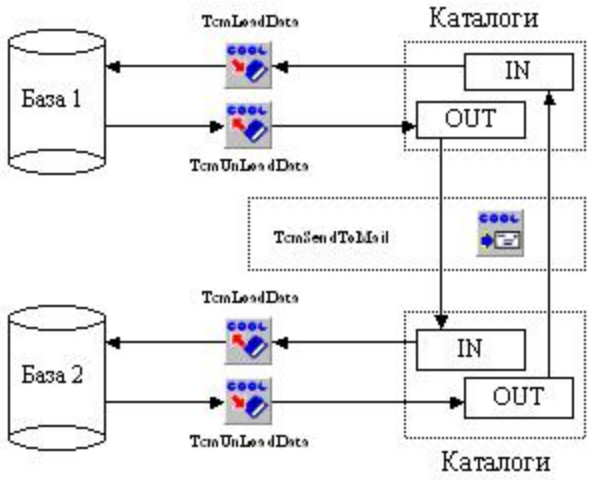

Палитра компонентов содержит три компонента для создания утилит межбазового обмена данными:


TcmLoadData = **class**(TcmSendData) - загрузка пакетов в БД


TcmUnloadData = **class**(TcmSendData) - создание пакетов


TcmSendToMail = (TcmSDProcessor) - обмен пакетами посредством электронной почты

Схема обмена пакетами с использованием компонент межбазового обмена:




Конфигурация "Торговля" содержит приложение SendData.cm4 - утилита для организации автоматического обмена пакетами с удаленными базами.

## 4.7 Управление транзакциями

Библиотека CoolLibrary изолирует разработчика от особенностей реализации сервера InterBase. Нет необходимости знать структуру таблиц базы, строить SQL запросы, явно запускать и завершать транзакции и т.д. Вместо этого библиотека предоставляет ряд компонентов, позволяющих разработчику абстрагироваться от уровня управления базами данных и сосредоточиться на разработке прикладной задачи. Одной из особенностей библиотеки является механизм автоматического управления транзакциями.

Рассмотрим как реализуется работа с транзакциями в стандартных системах управления базами данных:

```
-----------------------------------------------------------------
StartTransaction; //стартуем транзакцию

//выполняем какие либо действия над объектами базы

if Ok then CommitTransaction //если всё хорошо, сохраняем транзакцию
    else RollbackTransaction; //иначе откатываем все сделанные изменения
-----------------------------------------------------------------
```

Ключевым моментом в приведенном фрагменте является то, что программист явно указывает начало транзакции, контролирует результаты работы и программирует действия при сохранении и откате внесенных изменений. Разработчику системы CoolManager никогда не придется писать подобный код. Он вообще может не иметь ни малейшего представления о системе управления транзакциями в InterBase, их типах, уровнях изоляции и о многом другом. Всю рутинную работу за него делают компоненты библиотеки CoolLib. Теперь посмотрим как разработчик может управлять транзакциями используя платформу CoolManager.

При работе в системе CoolManager все транзакции стартуют и завершаются автоматически, без участия разработчика. При этом разработчик всё же должен иметь возможность прервать транзакцию и откатить внесенные изменения. Справочники и документы описанные в конфигураторе, имеют ряд обработчиков событий. Эти события возникают при редактировании записей справочников, проведении и распроведении документов, резервировании ресурсов сводных таблиц и т.д. Каждый обработчик стартует в рамках отдельной транзакции, которая автоматически завершается при выходе из процедуры обработчика. Если разработчику требуется прервать выполнение обработчика и откатить все сделанные изменения, достаточно сгенерировать исключение в теле процедуры.

Пример:

```
-------------------------------------------
//обработчик события проведения документа PrihodDoc
procedure doc_PrihodDoc_onRegister(Sender: TObject);
begin
 //выполняем обновление сводных таблиц базы
 //.............................

 //что то пошло не так, следует прервать проведение документа и откатить все внесенные изменения
 RaiseException('Проведение документа отменено.');

 //другие команды обновления данных
end;
-------------------------------------------
```

Процедура RaiseException генерирует исключение. При этом выполнение процедуры будет прервано, все изменения, сделанные в рамках текущей транзакции будут

отменены. Т.е. фактически выполняется команда RollbackTransaction.

Но иногда требуется явно запустить транзакцию, в рамках которой выполняется некая последовательность команд обновления данных (взаимосвязанное обновление записей справочников, создание и проведение ряда документов и т.д.). Для выполнения таких транзакций служит компонент **TcmTransaction**.

TcmTransaction = **class**(TComponent)

Работа с компонентом достаточно проста. Следует написать обработчик события OnExecute, в котором задается последовательность команд, выполняемых в рамках одной транзакции. Для старта транзакции вызвать метод Execute. Если во время выполнения требуется прервать транзакцию и откатить все сделанные изменения, сгенерировать исключение в теле обработчика события OnExecute.

По своей архитектуре СУБД используемая платформой CoolManager имеет распределенную структуру. Т.е. справочники хранятся в одном файле БД, документы и движения сводных таблиц в других файлах. В рамках одной транзакции может происходить модификация данных расположенных в разных базах, но разработчику не надо задумываться о подобных ньюансах, т.к. в этом случае всю работу за него выполняет платформа Cool Manager.

## 4.8 Редактирование объектов

Библиотека CoolLibrary предоставляет ряд невизуальных компонентов для доступа к объектам базы данных в скриптовом языке. Можно просто расположить нужный компонент на форме, либо создать его динамически в ходе выполнения скрипта. Далее приводятся примеры использования компонент библиотеки CoolLibrary при работе с данными СУБД.

Для редактирования записей справочников используется класс TcmSprEdit, в палитре компонентов он расположен на закладке CoolMan.

TcmSprEdit = class(TcmRecvizitEdit)

Пример работы с компонентом TcmSprEdit:

```
procedure TestWork;
 var Spr: TcmSprEdit;
   SprCode, GlobalVar: string;
begin
 //создаем компонент для редактирования справочника
 Spr:=TcmSprEdit.Create(nil);
 try
  //получаем ссылку на метаобъект базы данных
 Spr.Base:=TcmBases(CoolConfig.GetComponentByName('DataBaseObj'));
  if Spr.Base=nil then RaiseException('Не найдена база данных.');
 Spr.New('ObjectList',''); //новая запись
 Spr.RecordName:='Объект №1';
 Spr.UserName:='None';
 Spr.Post; //сохраняем запись в базе данных
 SprCode:=Spr.Code; //получаем уникальный код записи
```

```
 Spr.SprName:=''; //очищаем объект Spr
 Spr.Get(SprCode); //загружаем запись из базы
 Spr.Delete; //метим запись на удаление
 Spr.Restore; //снимаем пометку на удаление
  //сохраняем код записи в глобальной переменной DerectObject
 Spr.SetGlobalValue('DerectObject',SprCode);
 finally
 Spr.Free; //удаляем компонент
 end;
end;
```

Для редактирования документов используется компонент TcmDocEdit, в палитре компонентов он расположен на закладке CoolMan.

TcmDocEdit = class(TcmRecvizitEdit)

Пример работы с компонентом TcmDocEdit:

```
procedure EditDoc;
 var DocEdit: TcmDocEdit;
begin
 DocEdit:=TcmDocEdit.Create(nil);
 try
  //получаем ссылку на метаобъект базы данных
 DocEdit.Base:=TcmBases(CoolConfig.GetComponentByName('DataBaseObj'));
  if DocEdit.Base=nil then RaiseException('Не найдена база данных.');
 DocEdit.New('PrihodDoc'); //создаем новый документ
 DocEdit.UserName:='Tester'; //имя пользователя
 DocEdit.DocNomer:='1'; //номер документа
 DocEdit.DocDate:=Date; //дата выписки документа
  //устанавливаем реквизиты шапки документа
  DocEdit['Firm'].AsString:='1-1';
  DocEdit['Kontragent'].AsString:='1-3';
  DocEdit['Place'].AsString:='1-5';
 DocEdit.Post; //сохраняем документ
 DocEdit.SpecEdit.New; //создаем строку в спецификации документа
  //устанавливаем реквизиты строки специмфикации
 DocEdit.SpecEdit['Articul'].AsString:='1-7';
 DocEdit.SpecEdit['Quantity'].AsDouble:=20;
 DocEdit.SpecEdit['Summa'].AsCurrency:=100;
 DocEdit.SpecEdit['NDS'].AsCurrency:=152.42;
 DocEdit.SpecEdit.Post; //сохраняем строку спецификации
 DocEdit.RegisterDoc; //регистрируем документ
 DocEdit.UnRegisterDoc; //отменяем регистрацию документа
  //записываем в базу документа глобальную переменную с именем LastDoc
 DocEdit.SetGlobalValue(cmvCurrent,'LastDoc',Code);
 DocEdit.Delete; //удаляем документ
 finally
 DocEdit.Free;
 end;
end;
```

Компонент TcmDocSpecEdit предназначен для редактирования записей спецификации документа. При создании объекта класса TcmDocEdit автоматически создается объект для редактирования спецификации документа. Ссылка на него находится в свойстве TcmDocEdit.SpecEdit.

```
TcmDocSpecEdit= class(TcmRecvizitEdit)
```

Пример работы с компонетом TcmDocSpecEdit:

```
{В цикле удаляем все записи из спецификации документа DocEdit}
procedure ClearDocSpec(DocEdit: TcmDocEdit);
```

```
begin
 with DocEdit.DocSpec do
  if FirstRecord then
   repeat
    Delete;
   until not NextRecord;
end;
```

При проведении документа производится запись информации в сводные таблицы. Для описания процесса проведения следует перекрыть событие on_Register метаобъекта TcmDoc. Событию on_Register передается ссылка на  документ (объект класса TcmDocEdit). Для доступа к сводной таблице следует создать объект класса TcmTableEdit. Создать объект можно при помощи метода TcmDocEdit.GetTableEditObject:

```
function GetTableEditObject(TableName: string): TcmTableEdit;
```

Методу передается имя таблицы, к которой следует получить доступ. Имя таблицы находится в свойстве TcmTable.TableName. Вызывать данный метод можно только при проведении документа (в событии TcmDoc.OnRegister), при вызове этого метода в любом другом месте программного кода будет сгенерировано исключение.
Класс TcmTableEdit позволяет получить доступ к измерениям, ресурсам и реквизитам сводной таблицы:

```
TcmTableEdit = class(TObject)
```

При распроведении документа, все записи из сводных таблиц удаляются автоматически. Запрещено вносить изменения в проведенный документ.

Пример процедуры проведения документа:

```
{Проведение документа поставки товара}
procedure PrihodDoc_onRegister(Sender: TObject);
 var DocumTable,PartyTable,MoveTable: TcmTableEdit;
   Doc: TcmDocEdit;
  PartyCode: string;
begin
 Doc:=TcmDocEdit(Sender); //получаем ссылку на документ

 //вставка записи в регистр документов
 DocumTable:=Doc.GetTableEditObject('DocumTable');
 with DocumTable do
  try
   Recvizit['Firm'].AsString:='1-1';
   Recvizit['Kontragent'].AsString:='1-3';
   Recvizit['Place'].AsString:='1-5';
  Post;
  finally
  DocumTable.Free;
  end;

 //таблицы партий и движения товара
 PartyTable:=Doc.GetTableEditObject('PartyTable');
 MoveTable:=Doc.GetTableEditObject('MoveTable');
 try
  //в цикле просматриваем спецификацию документа
  with Doc.SpecEdit do
  if FirstRecord then
   repeat
    //.............добавляем информацию по партии товара.......................
```

```
 PartyTable.LineCode:=Code;
 PartyTable.Recvizit['PrihodCost'].AsDouble:=Recvizits['Summa'].AsDouble/
                                  Recvizits['Quantity'].AsDouble;
 PartyTable.Recvizit['Kontragent'].AsString:=Doc.Recvizits['Kontragent'].AsString;
 PartyCode:=PartyTable.Post;
  //..........добавляем информацию о движении товара.......................
 MoveTable.LineCode:=Code;
 MoveTable.LineSubCode:=1;
 MoveTable.Scale['Firm'].AsString:=Doc.Recvizits['Firm'].AsString;
 MoveTable.Scale['Place'].AsString:=Doc.Recvizits['Place'].AsString;
 MoveTable.Scale['Articul'].AsString:=Recvizits['Articul'].AsString;
 MoveTable.Scale['Party'].AsString:=PartyCode;
 MoveTable.Resource['Quantity'].AsDouble:=Recvizits['Quantity'].AsDouble;
 MoveTable.Post;
  until not NextRecord;
 finally
 PartyTable.Free;
 MoveTable.Free;
 end;
end;
```

# Глава V

## Приложения

# 5 Приложения

## 5.1 Модули конфигурации

**Конфигурация системы Cool Manager может содержать следующие объекты:**

**Главный модуль конфигурации**

Состоит из двух файлов:

- *Файл конфигурации.* Файл в XML формате (расширение .cm4). Содержит информацию, общую для всей конфигурации (список используемых bpl пакетов, список файлов конфигурации, списки каталогов в которых расположены объекты конфигурации и т.д.)
- *Скрипт конфигурации*. Текстовый файл с расширением .pas, содержит константы, переменные, процедуры и функции доступные для других объектов конфигурации. Скрипт конфигурации может ссылаться на другие скрипты (посредством оператора **uses**).

Любая конфигурация обязательно содержит оба вышеописанных файла.

*Пример:* Предположим мы создали конфигурацию CoolDemo. При этом автоматически будут созданы два файла:

CoolDemo.cm4

CoolDemo.pas

**Файлы конфигурации**

Конфигурация может содержать набор компонентов имеющих древовидную структуру. Дерево компонентов создается при помощи конфигуратора. Компоненты, включенные в конфигурацию, являются объектами предметной области и реализуют бизнес-логику приложения. Дерево компонентов сохраняется в виде текстовых файлов с расширением **cmu.** Конфигурация может содержать любое количество cmu файлов, что позволяет разработчику группировать объекты предметной области удобным для него образом. Кроме того, это позволяет представить конфигурацию в виде набора отдельных логических блоков, которые, в свою очередь, могут использоваться при создании других конфигураций.

**Формы**

Форма представляет собой визуальный элемент (окно) и используется для ввода и отображения информации, выполнения различных расчетов и т.д. Фактически, конфигурация, в момент выполнения, взаимодействует с пользователем посредством набора подключенных к ней форм.

Форма состоит из двух файлов:

- *Файл формы* (расширение dfm). Это текстовый файл, содержащий описание компонентов расположенных на форме.
- *Скрипт* (расширение pas). Содержит переменные, процедуры и функции используемые при работе формы.

**Скрипты**

Скрипты это текстовые файлы с расширением pas. Скрипты широко используются при создании конфигурации и содержат весь программный код. Кроме того, могут быть созданы скрипты содержащие библиотеки процедур и функций общего назначения. Любой скрипт может обращаться к переменным, процедурам, функциям другого скрипта (если этот скрипт указан в операторе **uses**).

**Шаблоны отчетов**

В стандартной поставке, для формирования отчетов, используется библиотека FastReport (*.frf) и экспорт данных в Excel (*.xls). Шаблоны отчетов являются неотемлемой частью конфигурации. При желании, можно подключить к ИСР любые другие генераторы отчетов созданных для IDE Delphi.

## 5.2 Регулярные выражения

### Введение

Регулярные выражения - это широкоиспользуемый способ описания шаблонов для поиска текста и проверки соответствия текста шаблону. Специальные **метасимволы** позволяют определять, например, что Вы ищете подстроку в начале входной строки или определенное число повторений подстроки.

На первый взгляд регулярные выражения выглядят страшновато (ну хорошо, на второй - еще страшнее ;) ).
Однако Вы очень быстро оцените всю их мощь. Они съэкономят Вам многие часы ненужного кодирования, а в некоторых случаях будут и быстрее работать, чем вручную закодированные проверки.

### Простое сравнение

Любой символ совпадает с самим собой, если он не относится к специальным метасимволам описанным чуть ниже.

Последовательность символов совпадает с такой же последовательностью во входной строке, так что шаблон "bluh" совпадет с подстрокой "bluh" во входной строке. Пока все просто, не так ли ?

Если необходимо, чтобы метасимволы или **escape-последовательности** воспринимались как обычные символы, их нужно предварять символом "\", например, метасимвол "^" обычно совпадает с началом строк, однако, если записать его как "\^", то он будет совпадать с символом "^", "\\" совпадает с "\" и т.д.

**Примеры:**
foobar *находит 'foobar'*
\^FooBarPtr *находит '^FooBarPtr'*

**Escape-последовательности**

Любой символ может быть определен с помощью escape последовательности, так же как это делается в языках C или Perl: "\n" означает начало строки, "\t" - табуляцию и т.д.. Вообще, \xnn, где nn это последовательность шестнадцатеричных цифр, означает символ с ASCII-кодом nn. Если необходимо определить двухбайтный (Unicode) символ, используйте формат '\x{nnnn}', где 'nnnn' - одна или более шестнадцатеричных цифр.

\xnn *символ с шестнадцатеричным кодом nn*
\x{nnnn} *символ с шестнадцатеричным кодом nnnn (более одного байта можно задавать только в режиме Unicode)*
\t *табуляция (HT/TAB), можно также \x09*
\n *новая строка (NL), можно также \x0a*
\r *возврат каретки (CR), можно также \x0d*
\f *перевод формата (FF), можно также \x0c*
\a *звонок (BEL), можно также \x07*
\e *escape (ESC), можно также \x1b*

**Примеры:**
foo\x20bar *находит 'foo bar' (обратите внимание на пробел посередине)*
\tfoobar *находит 'foobar' предшествуемый табуляцией*

**Перечни символов**

Вы можете определить перечень, заключив символы в []. Перечень будет совпадать с любым **одним** символом перечисленным в нем.

Если первый символ перечня (сразу после "[") - "^", то такой перечень совпадает с любым символом **не** перечисленным в перечне.

**Примеры:**
foob[aeiou]r *находит 'foobar', 'foober' и т.д. но не 'foobbr', 'foobcr' и т.д.*
foob[^aeiou]r *находит 'foobbr', 'foobcr' и т.д.. но не 'foobar', 'foober' и т.д.*

Внутри перечня символ "-" может быть использован для определения **диапазонов** символов, например a-z представляет все символы между "a" и "z", включительно.

Если Вам необходимо включить в перечень сам символ "-", поместите его в начало или конец перечня или предварите '\'. Если Вам необходимо поместить в перечень сам символ ']', поместите его в самое начало или предварите '\'.

**Примеры:**
[-az] *'a', 'z' и '-'*
[az-] *'a', 'z' и '-'*
[a\-z] *'a', 'z' и '-'*
[a-z] *все 26 малых латинских букв от 'a' до 'z'*
[\n-\x0D] *#10, #11, #12, #13.*
[\d-t] *цифра, '-' или 't'.*
[]-a] *символ из диапазона ']'..'a'.*

**Метасимволы**

Метасимволы - это специальные символы, являющиеся важнейшим понятием в регулярных выражениях. Существует несколько групп метасимволов.

Метасимволы - разделители строк
^ *начало строки*
$ *конец строки*
\A *начало текста*
\Z *конец текста*
. *любой символ в строке*

**Примеры:**
^foobar *находит 'foobar' только если он в начале строки*
foobar$ *находит 'foobar' только если он в конце строки*
^foobar$ *находит 'foobar' только если это единственное слово в строке*
foob.r *находит 'foobar', 'foobbr', 'foob1r' и т.д.*

Метасимвол "^" по умолчанию совпадает только в начале входного текста, а метасимвол "$" - только в конце текста. Внутренние разделители строк, имеющиеся в тексте, не будут совпадать с "^" и "$".
Однако, если Вам необходимо работать с текстом как с многострочным, чтобы "^" совпадал после каждого разделителя строки внутри текста, а "$" - перед каждым разделителем, то Вы можете включить модификатор /m.

Метасимволы \A и \Z аналогичны "^" и "$", но на них не действует модификатор /m, т.е. они всегда совпадают только с началом и концом всего входного текста.

Метасимвол "." по умолчанию совпадает с любым символом, однако, если Вы выключите модификатор /s, то '.' не будет совпадать с разделителями строк.

"^" совпадает с началом входного текста, а также, если включен модификатор /m, с точкой непосредственно следующей после \x0D\x0A, \x0A или \x0D (если Вы используете Unicode, то также \x2028 или \x2029 или \x0B или \x0C или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

"$" совпадает с концом входного текста, а также, если включен модификатор /m, с точкой непосредственно предшествующей \x0D\x0A, \x0A или \x0D (если Вы используете Unicode, то также \x2028 или \x2029 или \x0B или \x0C или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

"." совпадает с любым символом, но если выключен модификаторr /s, то "." не совпадает с \x0D\x0A и \x0A и \x0D (если Вы используете Unicode, то не совпадает также с \x2028 и \x2029 и \x0B и \x0C и \x85).

Обратите внимание, что "^.*$" (шаблон для пустой строки) не совпадает с пустой строкой вида \x0D\x0A, но совпадает с \x0A\x0D.

**Метасимволы - стандартные перечни символов**

\w *буквенно-цифровой символ или "_"*
\W *не \w*
\d *цифровой символ*
\D *не \d*
\s *любой "пробельный" символ (по умолчанию - [ \t\n\r\f])*
\S *не \s*

Стандартные перечни \w, \d и \s можно использовать и внутри **перечней символов**.

**Примеры:**
foob\dr *находит 'foob1r', ''foob6r' и т.д. но не 'foobar', 'foobbr' и т.д.*
foob[\w\s]r *находит 'foobar', 'foob r', 'foobbr' и т.д. но не 'foob1r', 'foob=r' и т.д.*

**Метасимволы - границы слов**

\b *Совпадает на границе слова*
\B *Совпадает не на границе слова*

Граница слова (\b) это точка между двумя символами, один из которых удовлетворяет \w, а другой - \W (в любом порядке), при этом перед началом и после конца строки подразумевается \W.

**Метасимволы - повторения**

После любого элемента регулярного выражения может следовать очень важный тип метасимвола - **повторитель**. Используя их Вы можете определить число допустимых повторений предшествующего символа, метасимвола или подвыражения.

* *ноль или более раз ("жадный"), то же что {0,}*
\+ *один или более раз ("жадный"), то же что {1,}*
? *ноль или один раз ("жадный"), то же что {0,1}*
{n} *точно n раз ("жадный")*
{n,} *не менее n раз ("жадный")*
{n,m} *не менее n но не более m раз ("жадный")*
*? *ноль или более раз ("не жадный"), то же что {0,}?*
+? *один или более раз ("не жадный"), то же что {1,}?*
?? *ноль или один раз ("не жадный"), то же что {0,1}?*
{n}? *точно n раз ("не жадный")*
{n,}? *не менее n раз ("не жадный")*
{n,m}? *не менее n но не более m раз ("не жадный")*

Т.о. {n,m} задает **минимум** n повторов и **максимум** - m. Повторитель {n} эквивалентен {n,n} и задает точно n повторов. Повторитель {n,} задает минимум n повторов. Теоретически величина параметров n и m не ограничена, но рекомендуется не задавать большие значения, поскольку в некоторых ситуациях это может потребовать существенных затрат времени и ОЗУ при обработке такого повторителя в связи с рекурсивным характером работы.

Если фигурные скобки встречаются в "неправильном" месте, где они не могут быть восприняты как повторитель, то они вопринимаются просто как символы.

**Примеры:**
foob.*r *находит 'foobar', 'foobalkjdflkj9r' и 'foobr'*
foob.+r *находит 'foobar', 'foobalkjdflkj9r' но не 'foobr'*
foob.?r *находит 'foobar', 'foobbr' и 'foobr' но не 'foobalkj9r'*
fooba{2}r *находит 'foobaar'*
fooba{2,}r *находит 'foobaar', 'foobaaar', 'foobaaaar' и т.д.*
fooba{2,3}r *находит 'foobaar', или 'foobaaar' но не 'foobaaaar'*

Небольшое пояснение по поводу "жадности". "Жадные" варианты повторителей пытаются захватить как можно большую часть входного текста, в то время как "не жадные" - как можно меньшую. Например, 'b+' как и 'b*' примененные к входной строке 'abbbbc' найдут 'bbbb', в то время как 'b+?' найдет только 'b', а 'b*?' - вообще - пустую строку; 'b{2,3}?' найдет 'bb', в то время как 'b{2,3}' найдет 'bbb'.
Вы можете переключить все повторители в выражении в "не жадный" режим, воспользовавшись модификатором /g.

### Метасимволы - варианты

Вы можете определить перечень **вариантов**, используя метасимвол "|" для их разделения, например "fee|fie|foe" найдет "fee" или "fie" или "foe", (так же как "f(e|i|o)e"). В качестве первого варианта воспринимается все от предыдущего метасимвола "(" или "[" или от начала выражения до первого метасимвола "|", в качестве последнего - все от последнего "|" до конца выражения или до ближайшего метасимвола ")". Обычно, чтобы не запутаться, набор вариантов всегда заключают в скобки, даже если без этого можно было бы обойтись.
Варианты пробуются начиная с первого и попытки завершаются сразу же как удастся подобрать такой при котором совпадет вся последующая часть выражения . Это означает, что варианты не обязательно обеспечат "жадное" поведение. Например, если применить выражение "foo|foot" ко входной строке "barefoot", то будет найдено "foo" так это первый вариант который позволил совпасть всему выражению.
Обратите внимание, что метасимвол "|" воспринимается как обычный символ внутри перечней символов, например, [fee|fie|foe] означает ровно то же самое что и [feio|].

**Примеры:**
foo(bar|foo) *находит 'foobar' или 'foofoo'.*

### Метасимволы - обратные ссылки

Метасимволы от \1 до \9 воспринимаются как обратные ссылки. \<n> совпадает с ранее найденным подвыражением #<n>.

**Примеры:**
(.)\1+ *находит 'aaaa' и 'cc'.*
(.+)\1+ *также находит 'abab' и '123123'*
(["']?)(\d+)\1 *находит "13" (в дв.кавычках), или '4' (в один.кавычках) или 77 (без кавычек) и т.д.*

### Расширения Perl

**(?imsxr-imsxr)**
Позволяет изменять значения модификаторов

**Примеры:**
(?i)Saint-Petersburg *находит 'Saint-petersburg' и 'Saint-Petersburg'*
(?i)Saint-(?-i)Petersburg *находит 'Saint-Petersburg' но не 'Saint-petersburg'*
(?i)(Saint-)?Petersburg *находит 'Saint-petersburg' и 'saint-petersburg'*
((?i)Saint-)?Petersburg *находит 'saint-Petersburg', но не 'saint-petersburg'*

**(?#text)**
Комментарий, просто игнорируется. Обратите внимание, что в комментарии такого вида невозможно поместить символ ")", поскольку он воспринимается как конец комментария.

# Глава VI

## Резюме

# 6 Резюме

Предложенное вашему вниманию "Введение в систему" содержит вводный материал и краткое описание основных возможностей платформы Cool Manager. Для более углубленного изучения системы следует ознакомиться со следующими руководствами:

- **Руководство программиста (Programmer.pdf)** - Содержит описание процесса создания собственных метаклассов для ИСР Cool Manager, а так-же подключения к скриптовому языку новых классов, типов, переменных, процедур и функций.
- **Руководство разработчика (Developer.pdf)** - Обучающее пособие по разработке в ИСР Cool Manager собственных бизнес-приложений.
- **Описание библиотеки CoolLibrary (CoolLib.pdf)** - Справочное руководство по классам и компонентам библиотеки CoolLibrary.

В качестве примера разработки коммерческих бизнес-приложений рекомендуем ознакомиться с конфигурацией "Торговля". Конфигурация предназначена для автоматизации процесса продаж предприятий оптовой и розничной торговли, и может быть использована в качестве "отправной точки" при разработке собственных бизнес-приложений.