ВЫСОКИЕ ТЕХНОЛОГИИ

КАЛЛИСТО

Руководство разработчика

Март 2007

Оглавление

Глава I

Введение

1 Введение

Руководство разработчика - является вводным курсом в интегрированную среду разработки (**ИСР**) системы **Cool Manager**. Руководство написано в виде обучающего пособия, и адресовано специалистам по сопровождению систем автоматизации. Материал построен на основе обучающих примеров, в процессе рассмотрения которых, сообщается дополнительная информация об интегрированной среде разработки, особенностях встроенного языка системы и принципах построения создаваемых конфигураций.

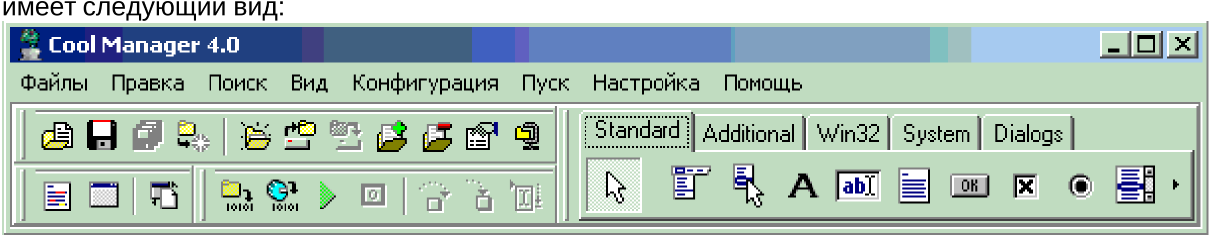
Предполагается, что при завершении знакомства с данным руководством, Вы будете обладать достаточными знаниями для создания собственных конфигураций в **ИСР Cool Manager**. Структура и инструменты **ИСР Cool Manager** очень похожи на IDE Delphi, поэтому, знакомство с Delphi (пусть даже очень поверхностное) значительно ускорит процесс обучения.

Перед прочтением данного руководства, рекомендуем прочитать "**Введение в систему**" (CoolDsgn.pdf), в этом документе излагаются базовые принципы построения системы и содержится краткий обзор основных её элементов.

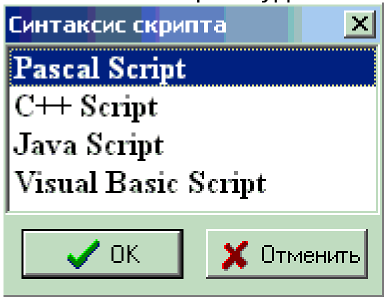
Руководство программиста является логическим продолжением руководства разработчика и знакомит Вас с тем, как создавать и подключать пакеты, созданные в Delphi, к Вашим конфигурациям. **ИСР Cool Manager** построена таким образом, что позволяет использовать всю мощь VCL Delphi в ваших проектах. Вы можете загружать стандартные VCL пакеты Delphi и использовать их в **ИСР** при создании форм, можете добавлять новые функции, классы и объекты к встроенному языку системы, создавать новые метаклассы для конфигуратора **ИСР** и использовать их при построении собственных бизнес-приложений. Для использования вышеперечисленных, а так же ряда других возможностей Вам достаточно создать стандартный **bpl** пакет в Delphi и загрузить его в **ИСР Cool Manager**.

1.1 Быстрый старт

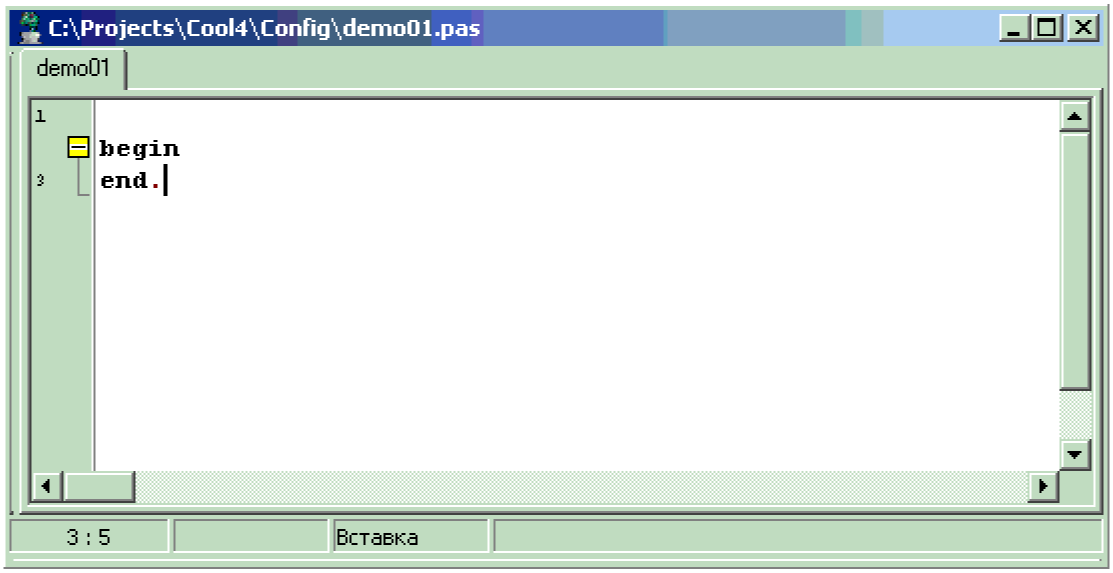
Запустите приложение CoolConfig**X**.exe (под буквой X подразумевается цифра обозначающая версию Delph). Это приложение находится в подкаталоге \BIN. При первом запуске приложение имеет следующий вид:



Это главная форма интегрированной среды. Теперь создадим новую конфигурацию, для этого выберем пункт меню **Конфигурация\Создать конфигурацию**. На экран будет выведено диалоговое окно для ввода имени файла конфигурации. Назовем конфигурацию **test** (по умолчанию файлы конфигурации имеют расширение **cm4**) и сохраним его в подкаталоге \CONFIG. На экран будет выведено диалоговое окно для выбора языка программирования:



выберете Pascal Script и нажмите **Enter**. На экране появится редактор скриптов с текстом главного скрипта конфигурации:

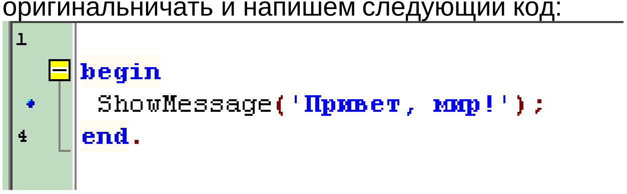


```
begin
end.
```

При этом в каталоге \CONFIG будут созданы два файла:

- test.cm4 - главный модуль конфигурации. Файл сохраняется в XML формате и содержит все глобальные настройки конфигурации.
- test.pas - главный скрипт конфигурации. Он то и выводится автоматически на экран после создания новой конфигурации.

Пока текст нашего скрипта предельно лаконичен - begin end и точка в конце. Это главный исполняемый модуль, содержащийся здесь код выполняется сразу же после запуска конфигурации.
Обычно все обучающие пособия по языкам программирования начинаются с простой программы выдающей на экран текстовое сообщение, что то вроде "Привет, мир!". Не будем оригинальничать и напишем следующий код:



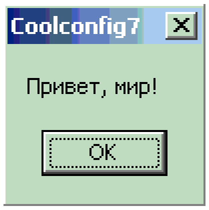
```
begin
 ShowMessage('Привет, мир!');
end.
```

Нажмем клавишу <**F9**> для синтаксического анализа введенного кода. Никаких сообщений на экран выведено не будет, следовательно наш скрипт не содержит синтаксических ошибок.

*Примечание. В Delphi клавиша <**F9**> используется для компиляции и запуска скомпилированного приложения, пара клавиш <**Ctrl-F9**> служит для компиляции текущего приложения. Так как при работе в ИСР Cool Manager пользоваться функцией синтаксического анализа Вы будете значительно чаще, чем запуском конфигурации на выполнение, более краткая команда <**F9**> - используется для компиляции (точнее просто для синтаксического анализа скрипта), а пара клавиш <**Ctrl-F9**> для запуска конфигурации на выполнение.*

Обратите внимание на синюю метку напротив третьей строки, метка выводится на служебной панели слева от текста скрипта. Такими метками обозначаются строки, содержащие исполняемые операторы языка. Метки расставляются после синтаксического анализа (клавиша <**F9**>) и снимаются после внесения любого изменения в скрипт.

Теперь запустим конфигурацию на выполнение. Нажмем пару клавиш <**Ctrl-F9**>, на экране появится стандартное сообщение Windows:

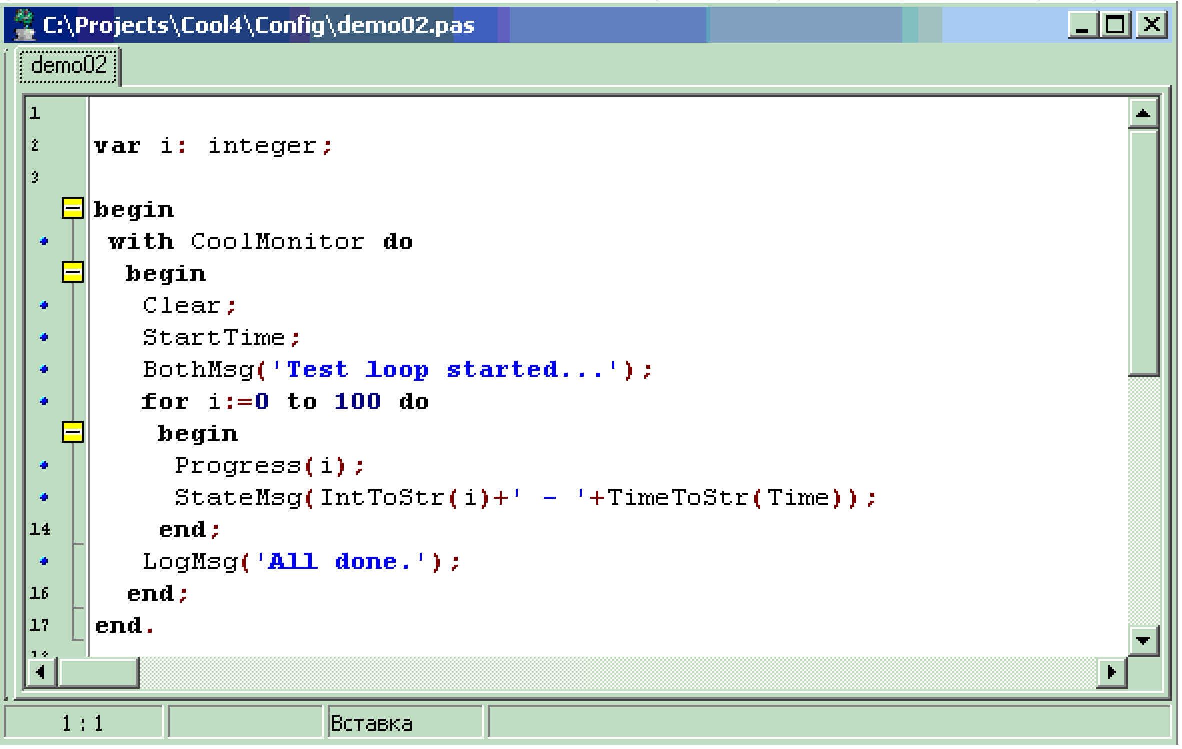


Привет, мир!

После нажатия на кнопку Ок, окно сообщения будет закрыто и выполнение конфигурации завершится, т.к. скрипт не содержит никаких других операторов.

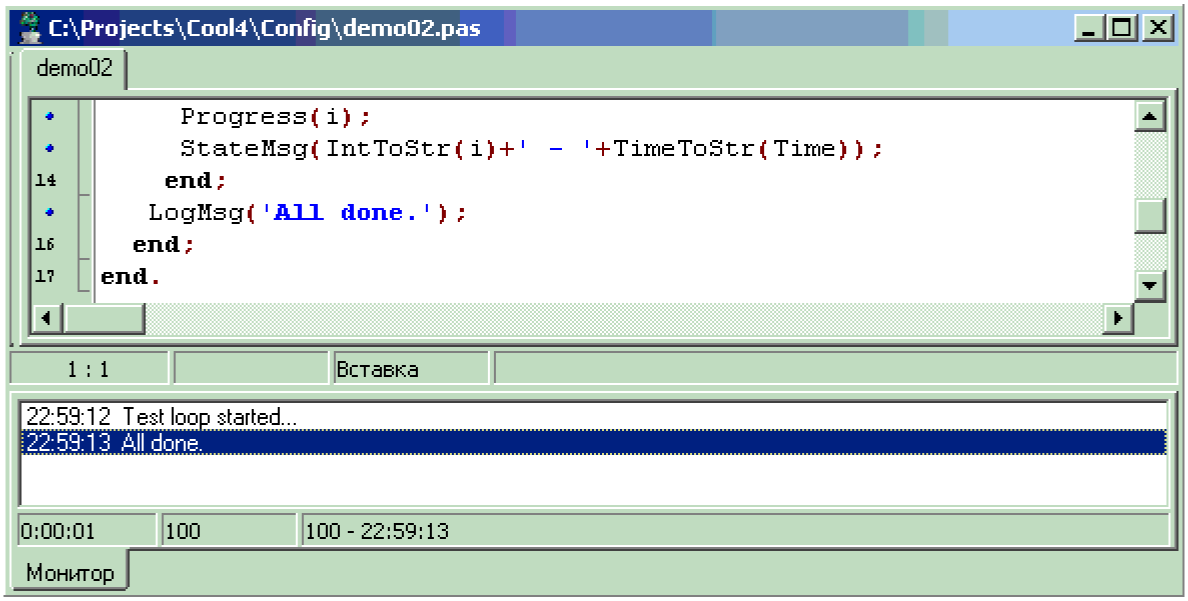
1.2 Работа с монитором сообщений

Рассмотрим более сложный пример. Выберите пункт главного меню ИСР **Конфигурация \Загрузить конфигурацию**. Появится стандартное диалоговое окно загрузки файла, загрузите файл **demo02.cm4** из каталога \CONFIG. Главный скрипт содержит следующие операторы:



```

var i: integer;

begin
 with CoolMonitor do
  begin
   Clear;
   StartTime;
   BothMsg('Test loop started...');
   for i:=0 to 100 do
    begin
     Progress(i);
     StateMsg(IntToStr(i)+' - '+TimeToStr(Time));
    end;
   LogMsg('All done.');
  end;
end.
```

Это демо пример работы с монитором сообщений. Монитор сообщений используется для вывода вспомогательных сообщений и трассировки скриптов при выполнении их в ИСР. Прежде чем описывать возможности и назначение монитора сообщений посмотрим на него в действии. Просто нажмите пару клавиш <Ctrl-F9> и запустите конфигурацию на выполнение. При этом, под редактором скриптов, будет выведено окно монитора сообщений:



```
      Progress(i);
      StateMsg(IntToStr(i)+' - '+TimeToStr(Time));
    end;
   LogMsg('All done.');
  end;
end.
```

```
1 : 1    Вставка
```

```
22:59:12 Test loop started...
22:59:13 All done.
```

```
0:00:01    100    100 - 22:59:13
```

Окно содержит поле для вывода сообщений и строку статуса с тремя панелями. Во время выполнения скрипта, информация на второй и третьей панелях строки статуса очень быстро меняется. После того, как в поле сообщений будет выведено сообщение **All done**, выполнение скрипта завершится.

Итак, зачем нужен монитор сообщений? Если вы обратили внимание, нашу конфигурацию можно отнести к классу "консольных" приложений. Т.е. в процессе работы конфигурации не создаются никакие визуальные формы для ввода и отображения информации. Главный скрипт просто содержит последовательность операторов, которая запускается на выполнения при старте конфигурации. После того, как все операторы выполнены, работа скрипта (и конфигурации в целом) завершается.

На первый взгляд может показаться, что в создании подобного рода "консольных" приложений нет никакого смысла. На самом деле, такие скрипты могут быть очень полезны в целях отладки, кроме того, вы можете написать сложный скрипт, который будет выполнять определенные действия над объектами вашей базы данных. Если изначально планируется запускать скрипт только в рамках ИСР, то создавать визуальные формы нет никакого смысла, но нужен механизм для отображения сообщений при выполнении длительных операций (например, при циклической обработке записей таблицы базы данных). Для этих целей и создан монитор сообщений.

При работе со скриптами доступна глобальная переменная CoolMonitor, это объект класса TCoolMonitor. CoolMonitor по логике использования похожа на глобальную переменную Delphi - Application. Переменная Application класса TApplication автоматически инициализируется в начале выполнения Delphi приложения и позволяет получить доступ к ряду свойств и методов общих для всего приложения в целом. В ходе выполнения приложения нет смысла создавать ещё одну переменную класса TApplication. То же самое можно сказать и о глобальной скриптовой переменной CoolMonitor.

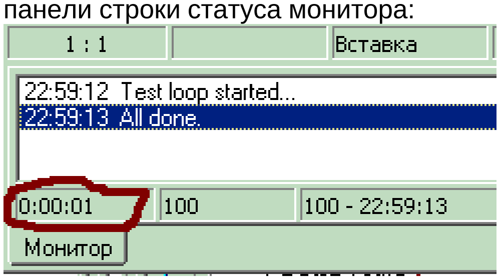
Теперь рассмотрим скрипт конфигурации более подробно. Все операторы скрипта заключены в блок with:

```
with CoolMonitor do
 begin
  ......................
 end;
```

Внутри этого блока происходит обращение к методам объекта CoolMonitor, опишем их в порядке следования.

Первым вызывается метод **Clear** (CoolMonitor.Clear), метод предназначен для очистки монитора сообщений. Если монитор содержал какие либо сообщения, все они будут удалены.

Далее следует метод **StartTime**, вызов метода инициализирует таймер для замера времени текущего процесса. Время прошедшее после вызова метода **StartTime** отражается на первой панели строки статуса монитора:

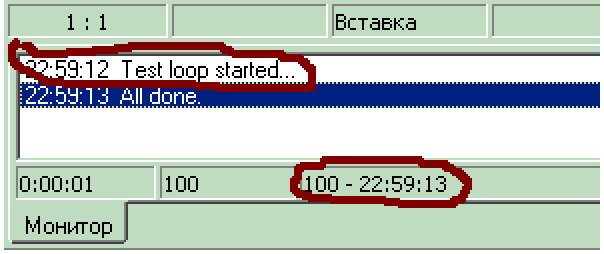


```
22:59:12  Test loop started...
22:59:13  All done.
```

```
0:00:01 | 100 | 100 - 22:59:13
```

Таймер сбрасывается при вызове метода **Clear**.

Далее следует вызов метода **BothMsg**, метод имеет единственный параметр типа строка. Переданная методу строка отображается и в поле сообщений и на третьей панели строки статуса:

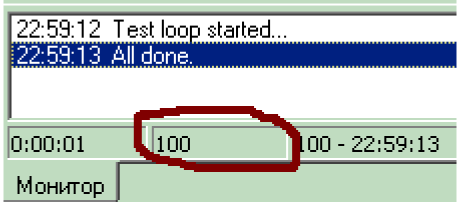


```
22:59:12  Test loop started...
22:59:13  All done.
```

```
0:00:01 | 100 | 100 - 22:59:13
```

Далее следует цикл от 0 до 100. В цикле вызываются два метода:

Метод **Progress** имеет один целочисленный параметр, переданное ему значение отображается на второй панели строки статуса:

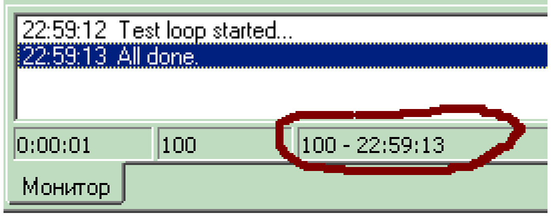


```
22:59:12  Test loop started...
22:59:13  All done.
```

```
0:00:01 | 100 | 100 - 22:59:13
```

Метод служит для отображения прогресса при выполнении длительных циклических операций.

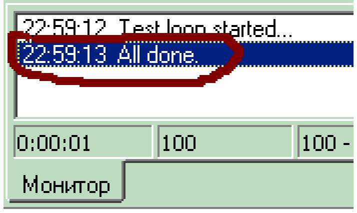
Далее следует вызов метода **StateMsg**, который имеет один строковый параметр. Переданная методу строка отображатся ТОЛЬКО на третьей панели строки статуса:



```
22:59:12  Test loop started...
22:59:13  All done.
```

```
0:00:01 | 100 | 100 - 22:59:13
```

После того, как цикл завершается, следует вызов метода **LogMsg**. Методу передается строковый параметр, эта строка отображается ТОЛЬКО в поле сообщений:



```
22:59:12  Test loop started...
22:59:13  All done.
```

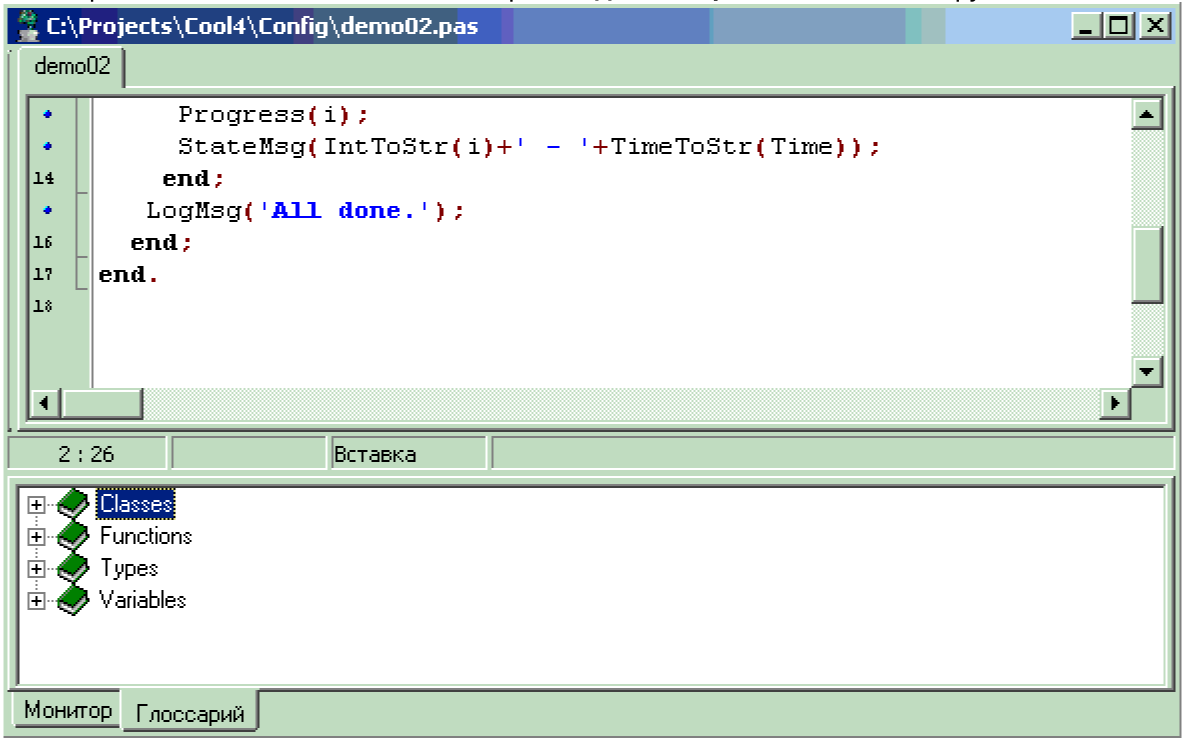
```
0:00:01 | 100 | 100 -
```

Таким образом, используя обращения к методам переменной CoolMonitor можно создать подробный лог выполнения скрипта, вывести прогресс при выполнении длительных циклических операций и произвести замер времени выполнения процесса в целом.

1.3 Оперативная справка по скриптовому языку.

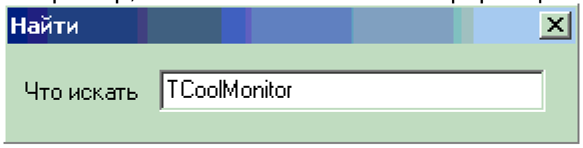
При работе с редактором скриптов Вы можете получить оперативную подсказку по используемым в скрипте переменным, функциям и классам. Например, при работе со скриптом, рассмотренным в предыдущем разделе, достаточно установить курсор на идентификатор **CoolMonitor** или идентификатор любого его метода и нажать пару клавиш <**Ctrl-F1**>. На экран будет выведено окно оперативной системы помощи с описанием текущей переменной или метода.

Встроенный скриптовый язык имеет открытую архитектуру, т.е. можно создавать в Delphi bpl пакеты содержащие описания классов, переменных, процедур и функций. При загрузке такого пакета в ИСР Cool Manager, все сделанные в нем объявления автоматически становятся доступными для использования в скриптах текущей конфигурации. Как просмотреть полный список объектов доступных в данный момент? Для этих целей служит Глоссарий, вызвать Глоссарий можно из главного меню, выбрав **Вид\Глоссарий**, либо нажав пару клавиш <**Alt-G**>:



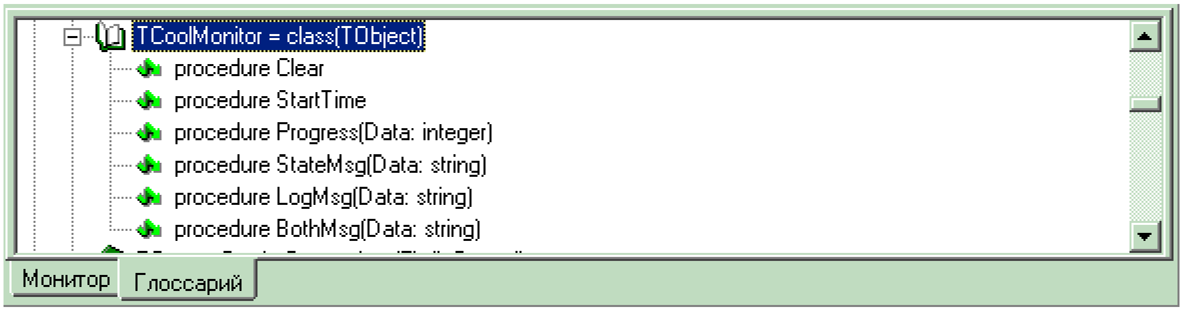
```
     Progress(i);
     StateMsg(IntToStr(i)+' - '+TimeToStr(Time));
    end;
   LogMsg('All done.');
  end;
end.
```

Все объекты, доступные в скрипте, представлены в виде древовидной структуры. Для поиска нужной информации следует нажать пару клавиш <**Ctrl-F**> и ввести ключевое слово для поиска. Например, если мы хотим найти информацию о классе TCoolMonitor, следует ввести:



Что искать: TCoolMonitor

и нажать клавишу <**Enter**>:

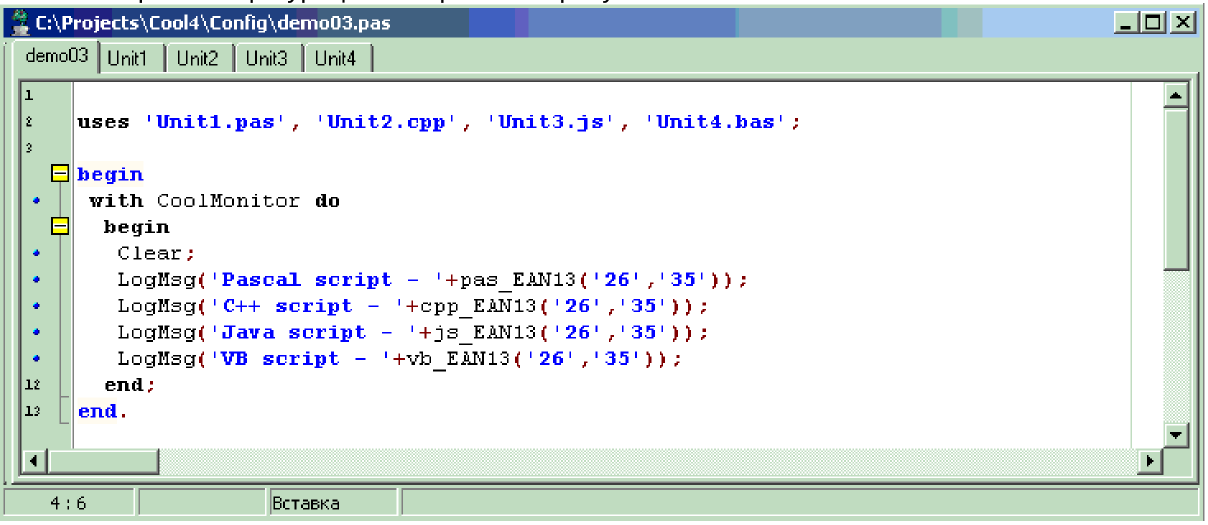


```
TCoolMonitor = class(TObject)
    procedure Clear
    procedure StartTime
    procedure Progress(Data: integer)
    procedure StateMsg(Data: string)
    procedure LogMsg(Data: string)
    procedure BothMsg(Data: string)
```

Для того, что бы получить оперативную подсказку по классу, методу, свойству, процедуре и т.д. достаточно установить полосу подсветки на нужную строку и нажать клавишу <**Enter**>, если раздел существует, на экран будет выведено окно оперативной системы помощи с описанием текущего объекта.

1.4 Мультиязычная архитектура скриптов

Встроенный скриптовый язык ИСР Cool Manager имеет мультиязычную архитектуру. В настоящее время поддерживаются PascalScript, C++Script, JavaScript и VisualBasic script. Конфигурация может содержать скрипты написанные на любом из вышеперечисленных языков программирования. Более подробная информация о структуре скриптового языка приведена в "Ведении в систему" (CoolDsgn.pdf). Рассмотрим пример конфигурации, содержащей скрипты написанные на разных языках. Загрузите из каталога \CONFIG конфигурацию **demo03.cm4**. Главный скрипт конфигурации изображен на рисунке:



```

uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';

begin
 with CoolMonitor do
  begin
   Clear;
   LogMsg('Pascal script - '+pas_EAN13('26','35'));
   LogMsg('C++ script - '+cpp_EAN13('26','35'));
   LogMsg('Java script - '+js_EAN13('26','35'));
   LogMsg('VB script - '+vb_EAN13('26','35'));
  end;
end.
```

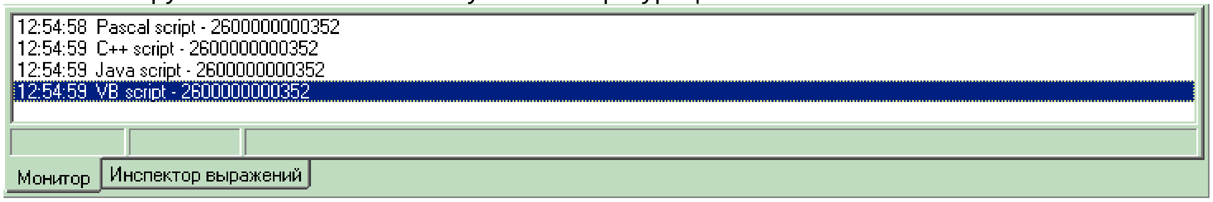
Скрипт начинается с оператора uses после которого указано четыре дополнительных скрипта подключаемых к главному скрипту конфигурации. Подробно, об операторе uses, будет рассказано в следующем разделе, пока только скажем, что главный скрипт использует другие скрипты, которые так же расположены в каталоге \CONFIG.

Несколько слов о рассматриваемом нами демо примере. В конфигурации реализована функция расчета контрольной цифры штрих-кода EAN13. Штрих-код EAN13 состоит из 12 цифр и одной контрольной (тринадцатой). После считывания штрих-кода, программа может самостоятельно рассчитать контрольную цифру по стандартному алгоритму и сравнить её со считанной сканером (проверка корректности считанных данных). Наша конфигурация содержит функцию, которой передаются два строковых параметра, префикс штрих кода и целочисленный код. Функция формирует стандартный код EAN13 и возвращает его в виде строки.
Например, если мы передадим функции два параметра - '26' и '35', будет сформирован следующий штрих-код - '260000000035X', где X обозначает контрольную цифру, сгенерированную для данного штрих кода.

В текущей конфигурации, для того, что бы продемонстрировать работу со скриптами написанными на разных языках, функция генерации штрих кода реализована в четырех вариантах:

- скрипт Unit1.pas содержит функцию pas_EAN13 на PascalScript
- скрипт Unit2.cpp содержит функцию cpp_EAN13 на C++Script
- скрипт Unit3.js содержит функцию js_EAN13 на JavaScript
- скрипт Unit4.bas содержит функцию vb_EAN13 на VBScript

В главном модуле конфигурации вызываются последовательно все четыре функции с одними же и теми параметрами. Для вывода возвращаемых ими данных используется CoolMonitor. Нажмите пару клавиш <Ctrl-F9> и запустите конфигурацию на выполнение:



```
12:54:58 Pascal script - 2600000000352
12:54:59 C++ script - 2600000000352
12:54:59 Java script - 2600000000352
12:54:59 VB script - 2600000000352
```

Как видите, все четыре функции сгенерировали один и тот же код, чего и следовало ожидать.

Для того, что бы указать на каком языке написан скрипт, в первой строке скрипта должна находиться директива #language <ЯзыкСкрипта>, возможны следующие варианты:

- #language PascalScript
- #language C++Script
- #language JScript
- #language BasicScript

По умолчанию используется PascalScript, поэтому в скриптах написанных на PascalScript директива #language может отсутствовать.

1.5 Оператор uses

При написании скриптов можно обращаться к процедурам и переменным описанным в других скриптах. Для этого имя файла скрипта, содержащего вызываемые переменные и процедуры, должно быть указано в операторе uses. В отличии от Delphi имя скрипта указывается в скобках, как текстовая переменная. Кроме имени файла, можно так же указывать расширение файла. Для примера рассмотрим оператор uses конфигурации описанной в предыдущем разделе:
uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';
оператор uses используется потому, что скрипт написан на Паскале. Для других скриптовых языков понадобилось бы написать следующие операторы:

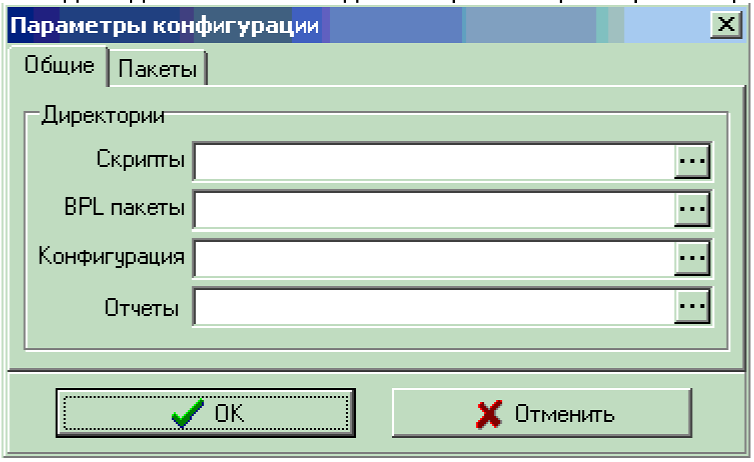
- C++Script: **#INCLUDE "Unit1.pas", "Unit2.cpp", "Unit3.js", "Unit4.bas";**
- JavaScript**: IMPORT "Unit1.pas", "Unit2.cpp", "Unit3.js", "Unit4.bas";**
- VBScript**: IMPORTS "Unit1.pas", "Unit2.cpp", "Unit3.js", "Unit4.bas";**

Порядок поиска скрипта указанного в операторе uses.

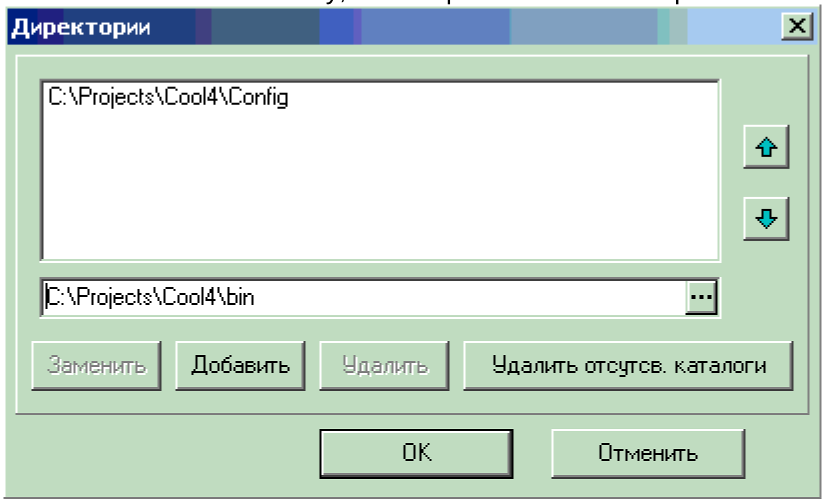
- если не указано расширение файла, производится поиск файлов с ращирениями: .pas, .cpp, .js, .bas. Поиск производится в указанной последовательности. Возвращается первый найденный файл.
- производится поиск в папке, из которой был запущен файл конфигурации (*.cm4)
- если файл не найден, получаем список каталогов, определенных для текущей конфигурации (будет описано ниже), производится поиск нужного файла во всех указанных каталогах. Просмотр каталогов производится в том порядке, как они перечислены в списке.

Список каталогов для поиска файлов.

На уровне конфигурации можно указать список каталогов, в которых следует искать скрипты и файлы конфигурации. Для задания списка каталогов, следует выбрать пункт главного меню **Конфигурация\Параметры конфигурации**, либо нажать <**Shift-Ctrl-F11**>. На экран будет выведено диалоговое окно для настройки параметров конфигурации:



На закладке Общие находится четыре поля ввода для указания списка каталогов. Ввод и выбор каталогов аналогичен тому, как это реализовано в Delphi:



C:\Projects\Cool4\Config

C:\Projects\Cool4\bin

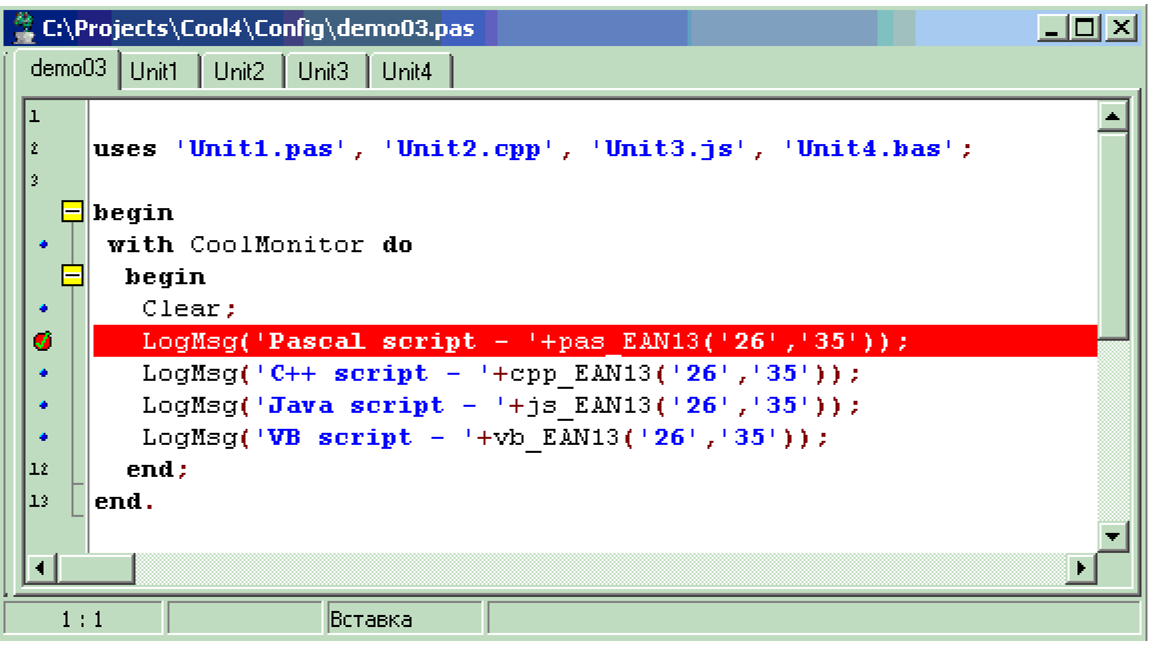
Можно указать списки каталогов для поиска следующих видов файлов:

- **Скрипты** - список каталогов, где следует производить поиск скриптов и dfm-файлов конфигурации.
- **BPL пакеты** - список каталогов, где следует производить поиск скомпилированных bpl пакетов, используемых конфигурацией.
- **Конфигурация** - список каталогов, где следует производить поиск файлов конфигурации. Файлы конфигурации обычно имеют расширение cmu.
- **Отчеты** - список каталогов, где следует производить поиск шаблонов отчетов.

*Примечание. При работе со скриптами запрещены циклические ссылки в операторе **uses**. Пример. Имеем три скрипта: unit1, unit2, unit3. Предположим, unit1ссылается на unit2, а unit2 на unit3. В этом случае, в операторе **uses** unit3, нельзя ссылаться не на Unit1, не на Unit2, т.к. это приведет к созданию циклической ссылки модулей друг на друга.*

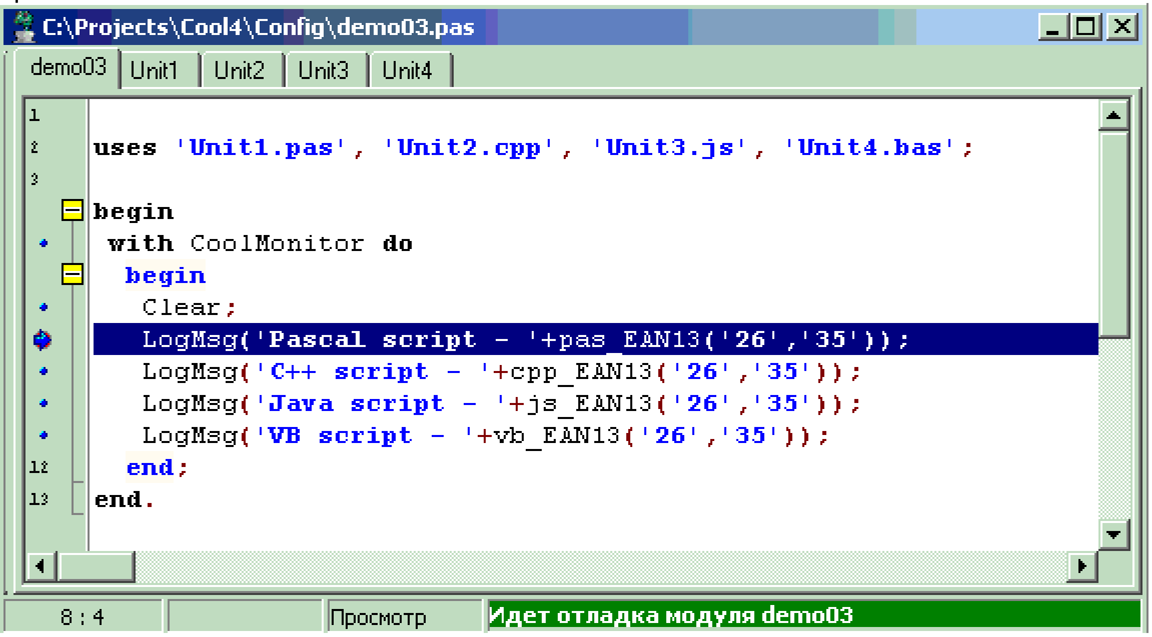
1.6 Работа с отладчиком скриптов.

ИСР CoolManager имеет встроенный отладчик скриптов, с помощью которого можно производить пошаговое выполнение скриптов, просмотр текущего значения переменных и выражений. Работа с отладчиком начинается с расстановки точек прерывания в тексте скрипта, для этого достаточно щелкнуть указателем мыши по служебному поля слева от текста скрипта:



```

uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';

begin
 with CoolMonitor do
  begin
   Clear;
   LogMsg('Pascal script - '+pas_EAN13('26','35'));
   LogMsg('C++ script - '+cpp_EAN13('26','35'));
   LogMsg('Java script - '+js_EAN13('26','35'));
   LogMsg('VB script - '+vb_EAN13('26','35'));
  end;
end.
```

На рисунке изображено окно редактора скриптов с главным скриптом конфигурации \CONFIG \demo03.pas. Точка прерывания установлена в 8 строке скрипта. Если теперь запустить скрипт на выполнение, то при достижении точки прерывания, выполнение скрипта будет приостановлено:



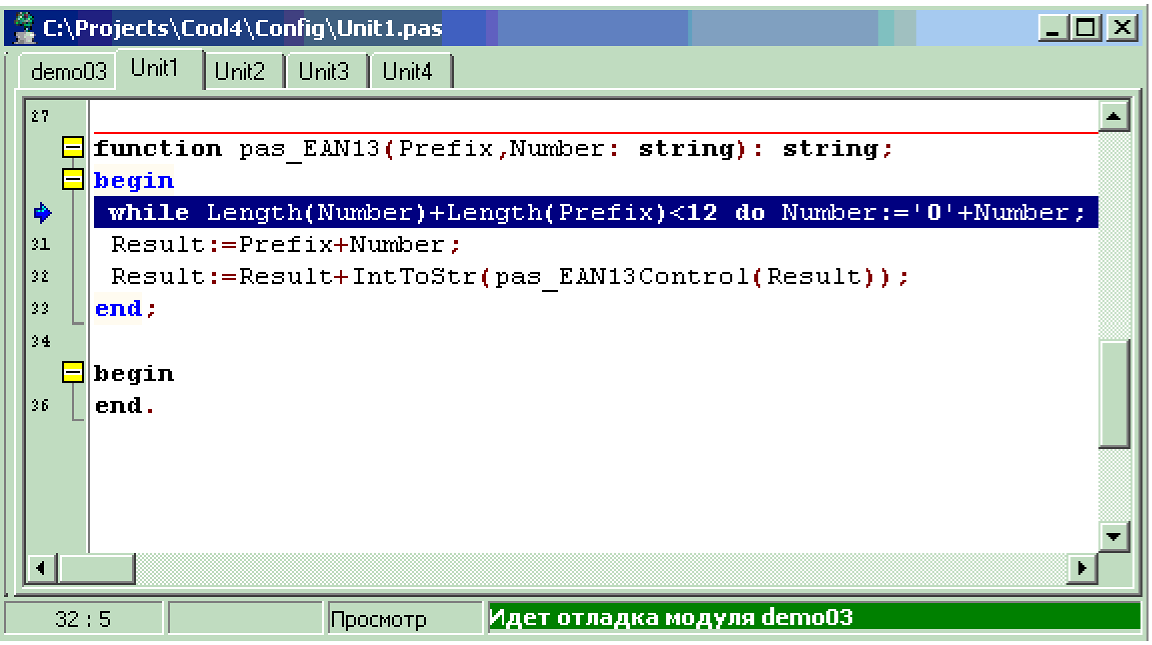
```

uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';

begin
 with CoolMonitor do
  begin
   Clear;
   LogMsg('Pascal script - '+pas_EAN13('26','35'));
   LogMsg('C++ script - '+cpp_EAN13('26','35'));
   LogMsg('Java script - '+js_EAN13('26','35'));
   LogMsg('VB script - '+vb_EAN13('26','35'));
  end;
end.
```

Идет отладка модуля demo03

Для выполнения скрипта в пошаговом режиме, доступно три команды:

- <**F8**> - шаг через, перейти к следующей строке текущего скрипта.
- <**F7**> - шагнуть внутрь, если текущий оператор содержит вызов процедуры или функции, перейти к ней.
- <**F4**> - идти до курсора. Скрипт выполняется до строки, в которой установлен курсор.

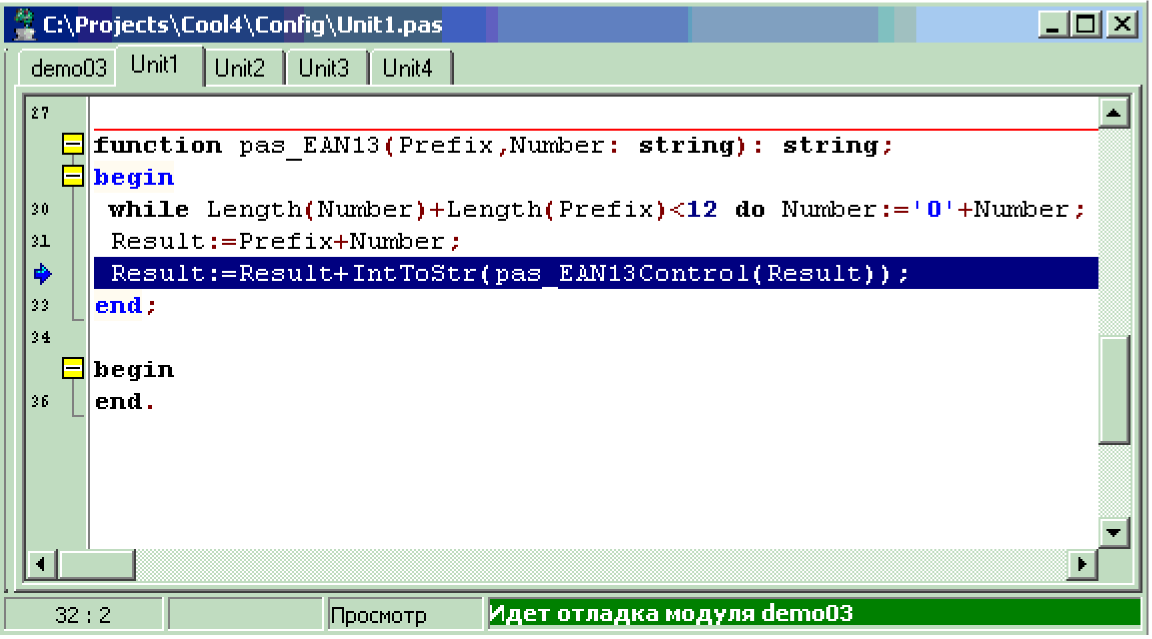
Нажмем клавишу <**F7**>, мы перейдем к первому оператору функции pas_EAN13:



```
27

   function pas_EAN13(Prefix,Number: string): string;
   begin
    while Length(Number)+Length(Prefix)<12 do Number:='0'+Number;
31  Result:=Prefix+Number;
32  Result:=Result+IntToStr(pas_EAN13Control(Result));
33 end;
34
   begin
36 end.
```

32 : 5 | Просмотр | Идет отладка модуля demo03

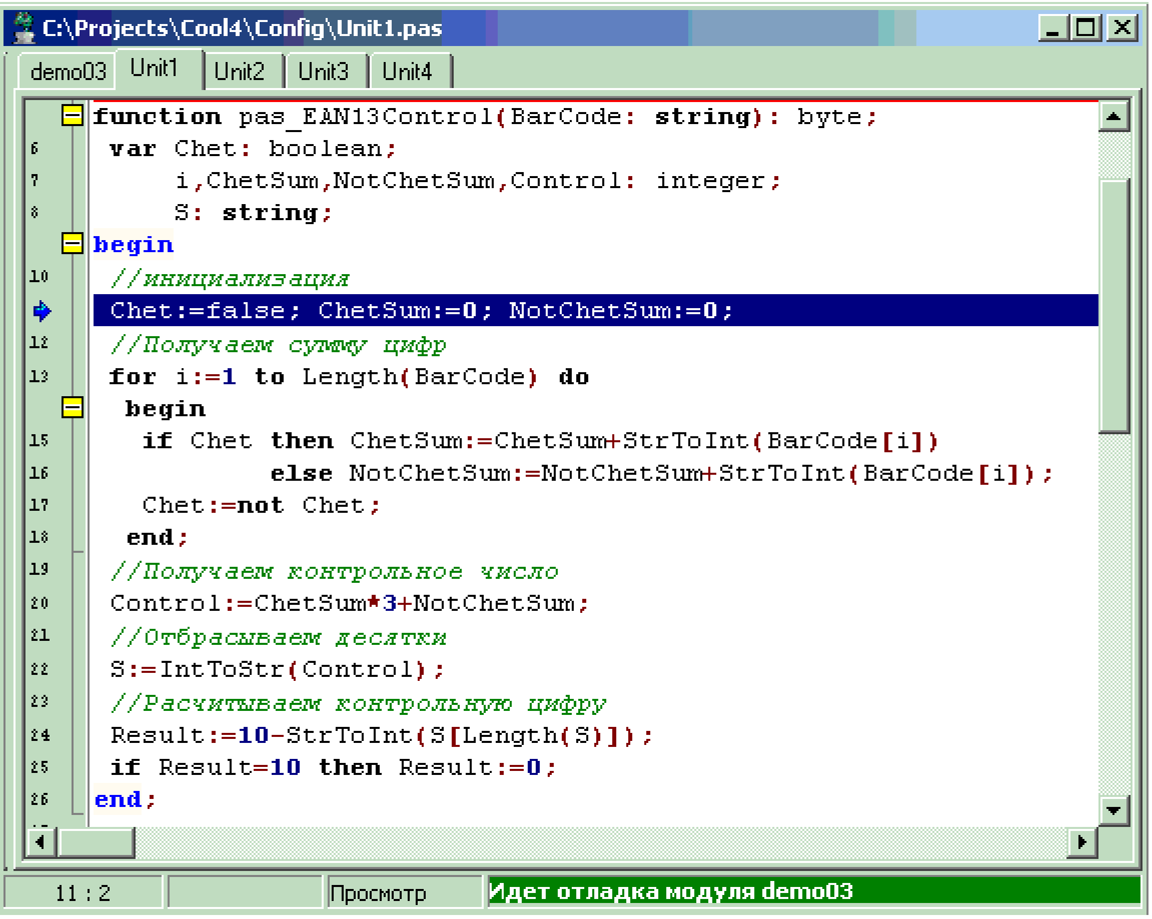
Установим курсор в строку 32 и нажмем клавишу <**F4**>:



```
27

   function pas_EAN13(Prefix,Number: string): string;
   begin
30  while Length(Number)+Length(Prefix)<12 do Number:='0'+Number;
31  Result:=Prefix+Number;
    Result:=Result+IntToStr(pas_EAN13Control(Result));
33 end;
34
   begin
36 end.
```

32 : 2 | Просмотр | Идет отладка модуля demo03

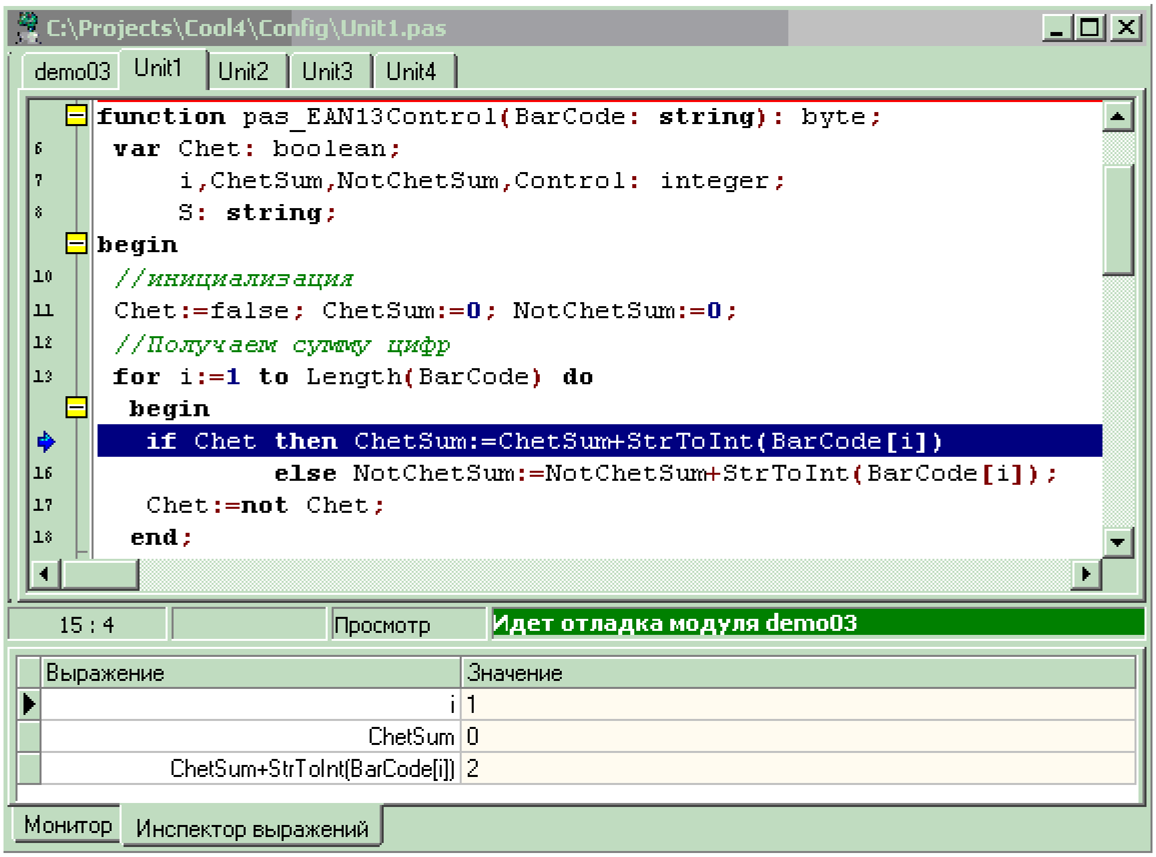
Если опять нажать <**F7**>, мы перейдем к первому оператору функции pas_EAN13Control:



```
function pas_EAN13Control(BarCode: string): byte;
 var Chet: boolean;
     i,ChetSum,NotChetSum,Control: integer;
     S: string;
begin
 //инициализация
 Chet:=false; ChetSum:=0; NotChetSum:=0;
 //Получаем сумму цифр
 for i:=1 to Length(BarCode) do
  begin
   if Chet then ChetSum:=ChetSum+StrToInt(BarCode[i])
           else NotChetSum:=NotChetSum+StrToInt(BarCode[i]);
   Chet:=not Chet;
  end;
 //Получаем контрольное число
 Control:=ChetSum*3+NotChetSum;
 //Отбрасываем десятки
 S:=IntToStr(Control);
 //Расчитываем контрольную цифру
 Result:=10-StrToInt(S[Length(S)]);
 if Result=10 then Result:=0;
end;
```

Идет отладка модуля demo03

В процессе отладки можно использовать инспектор выражений для просмотра значений переменных и выражений. Для вызова инспектора выражений следует нажать пару клавиш <**Ctrl-F5**>:



```
function pas_EAN13Control(BarCode: string): byte;
 var Chet: boolean;
     i,ChetSum,NotChetSum,Control: integer;
     S: string;
begin
 //инициализация
 Chet:=false; ChetSum:=0; NotChetSum:=0;
 //Получаем сумму цифр
 for i:=1 to Length(BarCode) do
  begin
   if Chet then ChetSum:=ChetSum+StrToInt(BarCode[i])
           else NotChetSum:=NotChetSum+StrToInt(BarCode[i]);
   Chet:=not Chet;
  end;
```

15 : 4 | Просмотр | Идет отладка модуля demo03

Выражение	Значение
i	1
ChetSum	0
ChetSum+StrToInt(BarCode[i])	2

Для того, что бы прервать выполнение конфигурации, следует нажать пару клавиш <**Ctrl-F2**>.

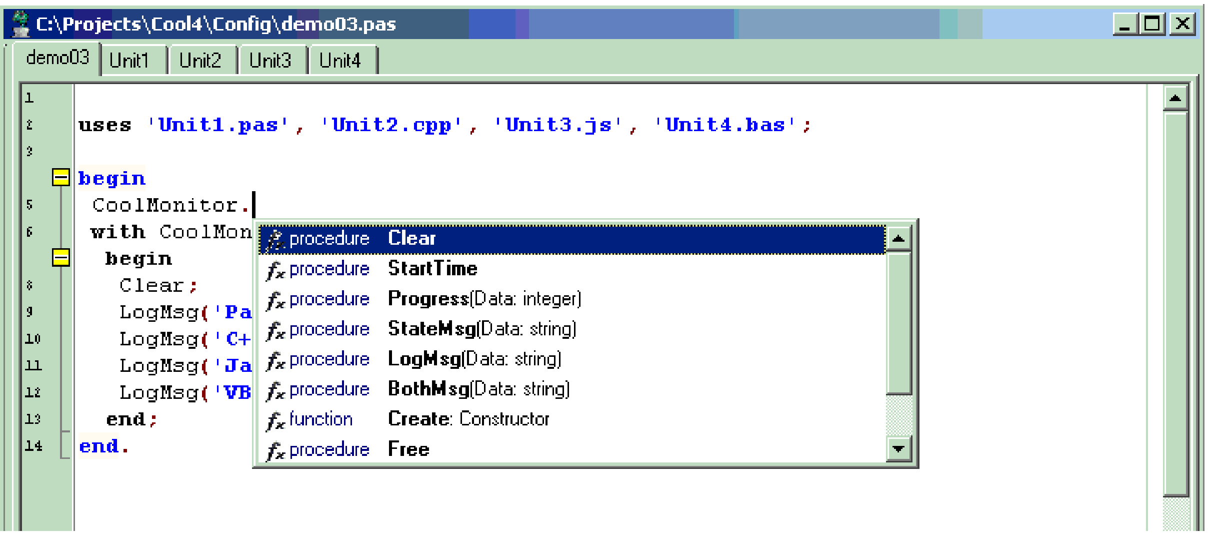
Примечание: Нельзя использовать отладчик для трассировки модальных форм (вызов метода TCoolForm.ShowModal). Для отладки форму следует выводить в немодальном режиме (методом TCoolForm.Show).

1.7 Инструменты для создания скриптов.

ИСР CoolManager предоставляет разработчику обширный арсенал дополнительных средств используемых при создании скриптов. В текущем разделе описываются некоторые наиболее интересные и полезные инструменты.

Сервисы CompletePopup и ParamPopup.

Редактор скриптов предоставляет разработчику дополнительные сервисы призванные максимально облегчить и ускорить процесс написания скриптов. Один из таких сервисов **CompletePopup** - интерактивная подсказка по свойствам и методам классов. Сервис активизируется автоматически в тот момент, когда разработчик должен указать имя свойства или метода класса:



```
uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';

begin
 CoolMonitor.
 with CoolMon
  begin
   Clear;
   LogMsg('Pa
   LogMsg('C+
   LogMsg('Ja
   LogMsg('VB
  end;
end.
```

```
procedure  Clear
procedure  StartTime
procedure  Progress(Data: integer)
procedure  StateMsg(Data: string)
procedure  LogMsg(Data: string)
procedure  BothMsg(Data: string)
function   Create: Constructor
procedure  Free
```

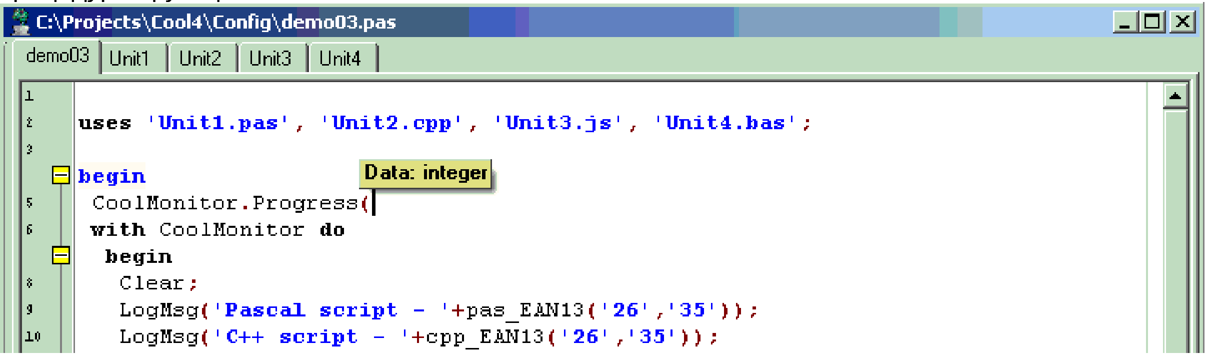
на экран выводится Popup список содержащий описание всех свойств и методов текущего объекта, можно выбрать нужное свойство из списка и нажать <**Enter**>, это приведет к вставке имени свойства или метода в текст скрипта.

Следует иметь ввиду, что поиск информации для **CompletePopup** ведется по дереву синтаксического разбра скрипта, само дерево формируется при синтаксическом анализе скрипта (<**F9**>), поэтому, если вы напишете следующий код:

```
var    List:    TStrings;
begin
    List.
end.
```

То после того, как вы поставите точку за идентификатором List, **CompletePopup** список для свойств и методов объекта List сформирован не будет. Т.к. синтаксическое дерево разбора не содержит никаких сведений о данной переменной. Для того, что бы **CompletePopup** работал корректно, следует произвести синтаксический анализ после объявления переменной List (нажать клавишу <**F9**>).

Существует аналогичный **ParamPopup** сервис для вывода Hint подсказки для параметров процедур и функций:

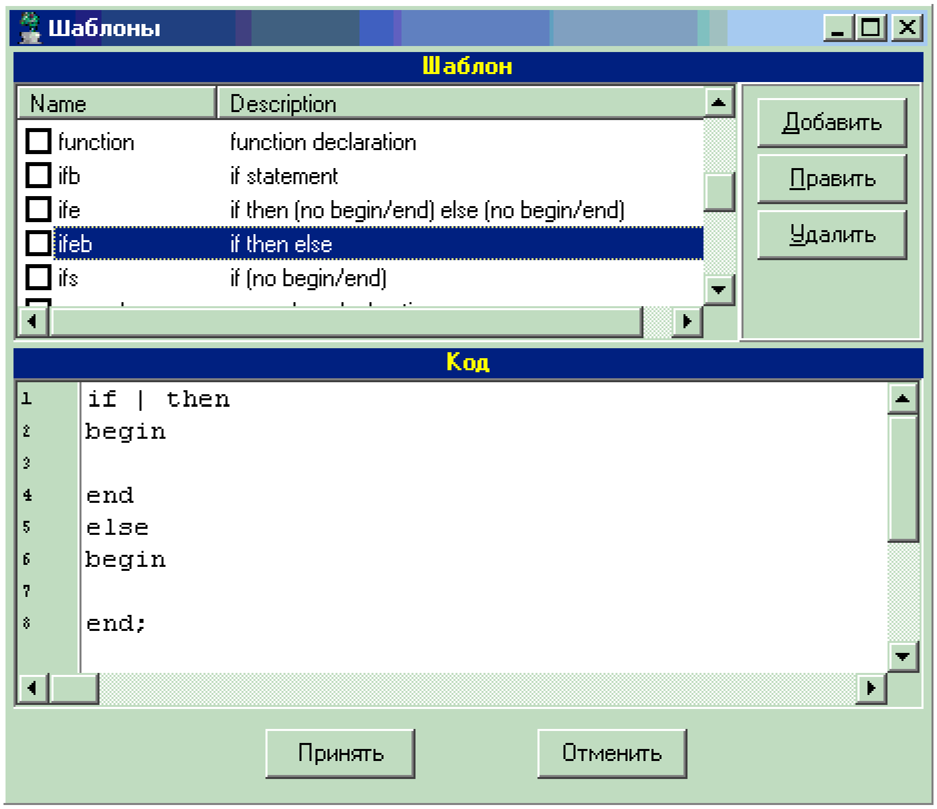


```
uses 'Unit1.pas', 'Unit2.cpp', 'Unit3.js', 'Unit4.bas';

begin
 CoolMonitor.Progress(
 with CoolMonitor do
  begin
   Clear;
   LogMsg('Pascal script - '+pas_EAN13('26','35'));
   LogMsg('C++ script - '+cpp_EAN13('26','35'));
```

ParamPopup работает подобно сервису **CompletePopup**, т.е. информация о параметрах процедур и функций извлекается из дерева синтаксического разбора скрипта. Поэтому, если в тексте скрипта есть объявления процедур и функций, то после их создания, следует провести синтаксический анализ скрипта (нажать <**F9**>), иначе сервис **ParamPopup** для скриптовых процедур и функций работать не будет.

Редактор шаблонов.

Кроме сервисов CompletePopup и ParamPopup, следует упомянуть о редакторе шаблонов конструкций языка. Для вызова редактора шаблонов, следует выбрать пункт главного меню **Настройка\Настройка редактора\Шаблоны\<Шаблон языка>**, на экране появится редактор шаблонов:



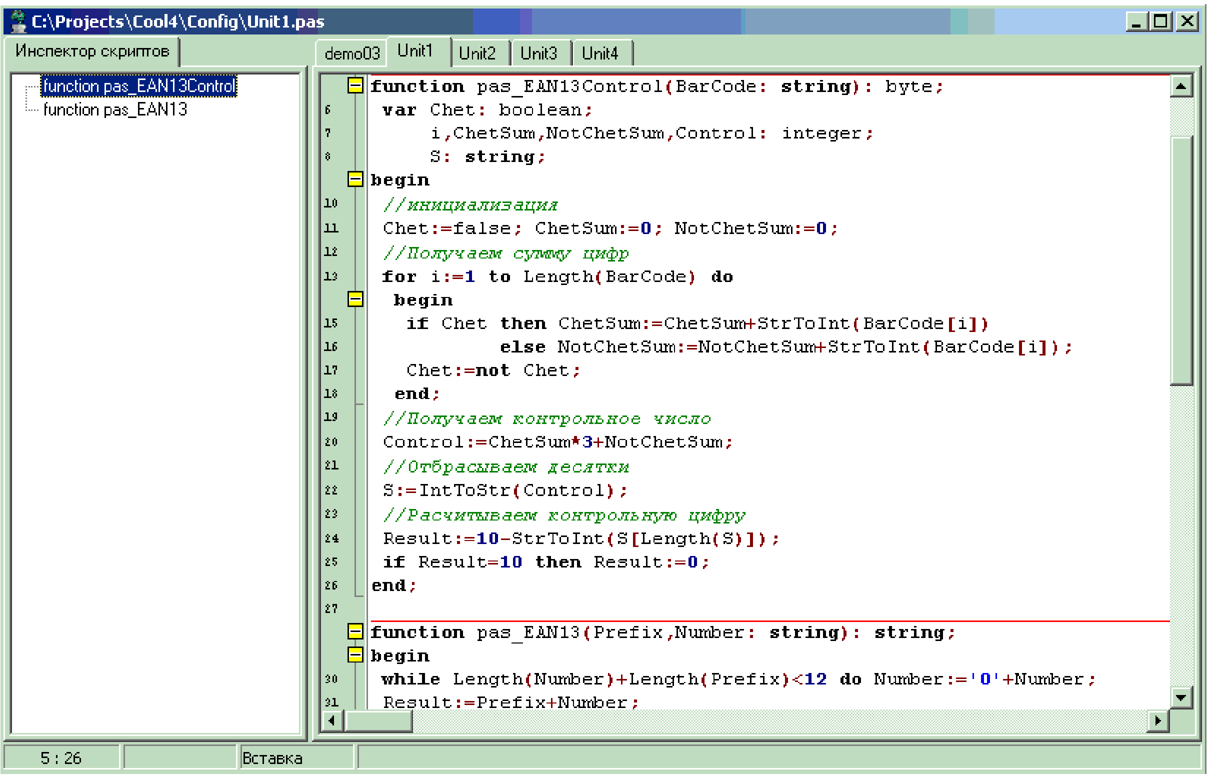
Name	Description
□ function	function declaration
□ ifb	if statement
□ ife	if then (no begin/end) else (no begin/end)
□ ifeb	if then else
□ ifs	if (no begin/end)

```
if | then
begin

end
else
begin

end;
```

В верхней части окна расположен список определенных шаблонов, внизу расположен редактор для ввода текста шаблона. Для вставки шаблона в скрипт, следует ввести имя шаблона и нажать пару клавиш <**Ctrl-J**>, текст шаблона будет вставлен с текущей позиции курсора.

Инспектор скриптов.

Для вызова инспектора скриптов следует выбрать пункт главного меню **Вид\Инспектор скриптов**. Инспектор скриптов содержит список процедур и функций определенных в текущем скрипте:

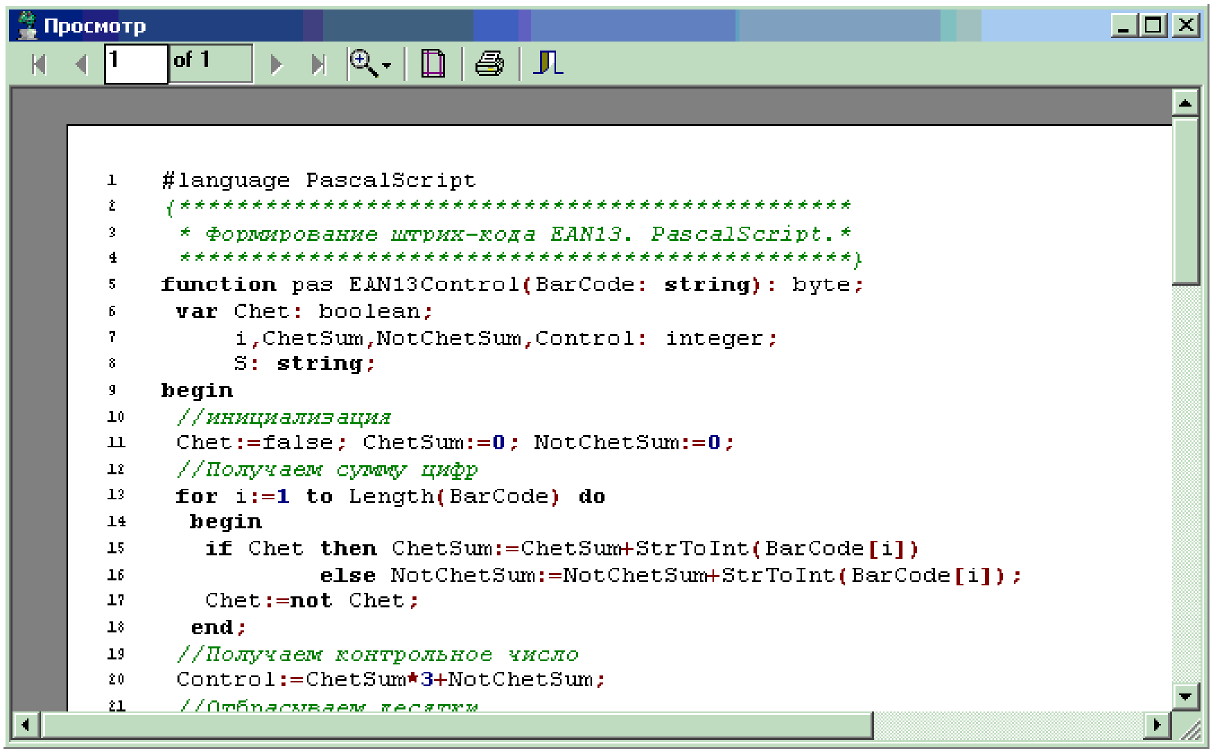


```
function pas_EAN13Control(BarCode: string): byte;
 var Chet: boolean;
     i,ChetSum,NotChetSum,Control: integer;
     S: string;
begin
 //инициализация
 Chet:=false; ChetSum:=0; NotChetSum:=0;
 //Получаем сумму цифр
 for i:=1 to Length(BarCode) do
  begin
   if Chet then ChetSum:=ChetSum+StrToInt(BarCode[i])
           else NotChetSum:=NotChetSum+StrToInt(BarCode[i]);
   Chet:=not Chet;
  end;
 //Получаем контрольное число
 Control:=ChetSum*3+NotChetSum;
 //Отбрасываем десятки
 S:=IntToStr(Control);
 //Расчитываем контрольную цифру
 Result:=10-StrToInt(S[Length(S)]);
 if Result=10 then Result:=0;
end;

function pas_EAN13(Prefix,Number: string): string;
begin
 while Length(Number)+Length(Prefix)<12 do Number:='0'+Number;
 Result:=Prefix+Number;
```

Двойной щелчёк указателем мыши по записи в инспекторе скриптов, приводит к выделению выбранной процедуры или функции в редакторе скриптов. Инспектор скриптов является удобным инструментом для навигации и исследования содержимого скриптов.

Просмотр и печать скриптов.

Для печати скриптов следует использовать пункт главного меню **Файлы/Просмотр..**. На экран будет выведена форма для предварительного просмотра скрипта в том виде, как он будет напечатан:



```
#language PascalScript
{*********************************************
 * Формирование штрих-кода EAN13. PascalScript.*
 *********************************************}
function pas EAN13Control(BarCode: string): byte;
 var Chet: boolean;
     i,ChetSum,NotChetSum,Control: integer;
     S: string;
begin
 //инициализация
 Chet:=false; ChetSum:=0; NotChetSum:=0;
 //Получаем сумму цифр
 for i:=1 to Length(BarCode) do
  begin
   if Chet then ChetSum:=ChetSum+StrToInt(BarCode[i])
           else NotChetSum:=NotChetSum+StrToInt(BarCode[i]);
   Chet:=not Chet;
  end;
 //Получаем контрольное число
 Control:=ChetSum*3+NotChetSum;
 //Отбрасываем десятки
```

Для печати скрипта, следует нажать пару клавиш <**Ctrl-P**>.

Глава II

Работа с визуальными формами

2 Работа с визуальными формами

ИСР CoolManager имеет встроенный конструктор форм, аналогичный дизайнеру форм IDE Delphi. Палитра компонентов содержит набор доступных визуальных и невизуальных компонент, которые можно помещать на формы. Инспектор объектов позволяет редактировать опубликованные свойства форм и компонент. Каждая форма состоит из двух файлов: файла формы (.dfm) и файла скрипта (.pas, .cpp, .js, .bas), оба файла должны иметь одинаковые имена. Все создаваемые формы наследуются от класса TCoolForm, это базовый класс для создания 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. Во время выполнения приложения, создание формы осуществляется при помощи метода CreateCoolForm класса TApplication. Для скриптов определена глобальная переменная Application: TApplication (так же как в Delphi):
Form:=Application.CreateCoolForm('DerectForm', false);
Методу CreateCoolForm передаются два параметра: имя модуля формы и переменная типа boolean, которая определяет способ создания формы. Когда в скрипте происходит вызов метода CreateCoolForm, производится поиск модуля формы с именем, указанным в первом параметре. Если модуль найден производится загрузка файла формы и связанного с ней скрипта. Затем создается форма, скрипт компилируется, после чего форма запускается на выполнение. Второй параметр определяет можно ли создавать несколько экземпляров одной и той же формы. Если второй параметр равен false - можно создавать только один экземпляр, т.е. при вызове CreateCoolForm, производится поиск формы с указанным в первом параметре именем, если такая форма существует - возвращается указатель на найденную форму, иначе форма создается и возвращается указатель на новую форму. Если второй параметр равен true, всегда создается новый экземпляр формы, для него генерируется уникальное имя и возвращается ссылка на созданный экземпляр. При работе с формами следует помнить, что форма фактически загружается в приложение только при вызове метода CreateCoolForm и удаляется после вызова метода Free формы. Это позволяет более экономично использовать оперативную память компьютера при работе приложения, т.к. сложная система автоматизации может использовать не одну сотню различных форм, но файлы и скрипты этих форм не будут загружаться автоматически при запуске приложения.

Примечание: Отличается способ загрузки форм при работе в ИСР и при вызове конфигурации посредством приложения-просмотрщика с рабочей станции пользователя. В первом случае, производится поиск формы во всех каталогах указанных в параметрах конфигурации, если модуль найден, он загружается непосредственно с диска. Во втором случае, происходит обращение к серверу приложений, который, в свою очередь, возвращает приложению-просмотрщику модули формы. Сервер приложений будет описан в четвертой главе.
Так же следует отметить, что при создании нескольких экземпляров одной формы, отладчик будет работать только с первым экземпляром формы.

Хорошим тоном разработки приложений в системе Cool Manager, считается создание всех форм в ИСР. Но при желании к скрипту можно подключить форму содержащюся в bpl пакете (то, как подключить к скриптовому языку форму описанную в bpl пакете, описывается в Руководстве программиста). В этом случае, для создания и показа формы из скрипта, используется стандартный код:

```
procedure  ExecutePackForm;
var  Form:  TPackForm;
begin
         Form:=TPackForm.Create(Application);
```

```
    Form.ShowModal;
     Form.Free;
end;
```

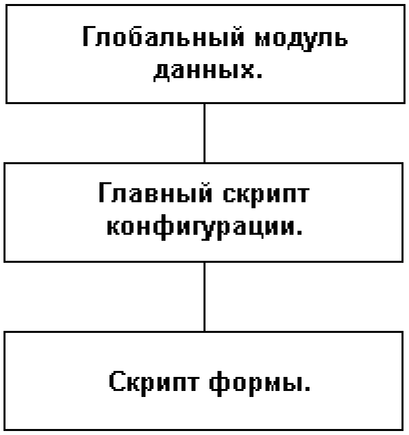
Примечание. Ещё один хороший аргумент в пользу создания форм средствами ИСР: если форма находится в пакете, то пакет автоматически загружается в оперативную память при запуске приложения, большое количество форм в bpl пакетах могут потребовать значительного объема памяти при загрузке, хотя вовсе не факт, что все они будут использоваться в течении текущего сеанса работы. Использование ИСР форм, полностью лишено этого недостатка, т.к. форма загружается в оперативную память только при вызове метода CreateCoolForm. Кроме того, теряется сам смысл использования CoolManager - т.к. изменить форму находящуюся в пакете, невозможно без перекомпиляции самого пакета, кроме того, если изменяется interface раздел модуля содержащего форму, требуется перекомпиляция всех пакетов использующих измененный пакет. Как видите, сложность сопровождения системы при таком подходе, значительно возрастает. Хотя, существуют ситуации, когда использование форм, объявленных в пакете, необходимо. Например, когда вы адаптируете сложное Delphi приложение для работы в системе CoolManager. Разом переместить все используемые приложением формы из пакетов в ИСР не представляется возможным, поэтому на переходном этапе, можно использовать вызов форм из пакетов.

Возможно, кто-то из внимательных читателей приведет ещё один аргумент в защиту вызова форм из bpl пакетов: исполнение скомпилированного кода пакета значительно быстрее, чем выполнение скрипта, т.е. если форма содержит достаточно сложный код, помещение форм в пакеты может дать значительный выигрыш в производительности. Но делать так всё же не следует. Гораздо предпочтительнее создать в Delphi класс, который будет содержать весь код вычислений, затем поместить класс в пакет. В скрипте ИСР создать экземпляр класса и производить вычисления посредством вызова методов класса.

Идеология создания приложений с использованием Cool Manager, подразумевает создание всех визуальных элементов средствами ИСР, но у разработчика всегда есть выбор - какую часть бизнес-логики приложения реализовать на уровне скриптового языка, а какую оформить в виде специальных классов и поместить в пакеты.

2.1 Области видимости объектов скрипта

Конфигурация в Cool Manager играет ту же роль, что и приложение в Delphi. Конфигурация состоит из набора файлов: скриптов, модулей конфигурации, форм и т.д.. По умолчанию файл конфигурации имеет расширение .cm4. Каждая конфигурация содержит главный скрипт. Переменные, процедуры и функции, описанные в главном скрипте, являются глобальными для всех форм конфигурации. Вышесказанное можно проиллюстрировать следующим рисунком:



- **Глобальный модуль данных -** в глобальном модуле происходит регистрация типов, переменных, процедур, функций и классов подключаемых к скриптовому языку из bpl пакетов. Все сделанные здесь объявления доступны для любого скрипта запущенного в рамках системы CoolManager.
- **Главный скрипт конфигурации** - создается для каждой конфигурации, может содержать объявления переменных, констант, процедур и функций. Именно этот скрипт выполняется при запуске конфигурации.
- **Формы** - состоят из модуля формы (dfm файла) и связанного с ним скрипта. Процедуры и функции описанные в скрипте,могут использоваться в качестве обработчиков событий формы и расположенных на ней компонентов. Владельцем скрипта формы является главный скрипт конфигурации. Это обозначает, что все объявления процедур, функций и переменных сделанных в главном скрипте конфигурации (а так же в скриптах перечисленных в операторе uses главного скрипта) доступны в скрипте формы.

Приведем пример. Предположим существует форма Form1, при запуске формы, в событии onCreate, устанавливается текст заголовка окна:

Form1.pas

```
----------------
procedure    Form1_onCreate(Sender:    TObject);
begin
         Self.Caption:=FormTitle;
end;
```

В приведенном скрипте, служебная переменная Self содержит ссылку на форму связанную со скриптом. Эта переменная объявляется автоматически при компиляции скрипта формы. Переменная Self используется для доступа к свойствам и методам формы. Заголовку окна (свойство Caption), присваивается значение глобальной переменной FormTitle.Где может быть описана эта переменная?

Первый вариант - в главном скрипте конфигурации:

Demo.pas

```
---------------
var     FormTitle: string;
          DerectForm:   TCoolForm;

begin
     FormTitle:='Главная    форма    конфигурации';
           DerectForm:=Application.CreateCoolForm('Form1',      true);
    DerectForm.Show;
end.
```

При запуске конфигурации, в первую очередь, выполняется код расположенный в блоке {begin end.} главного скрипта. В нашем примере, устанавливается значение глобальной переменной FormTitle, затем создается и выводится на экран форма Form1. Т.к. владельцем скрипта формы является главный скрпит конфигурации, скрипту Form1.pas доступно объявление глобальной переменной FormTitle.

Можно сделать объявление глобальной переменной в отдельном скрипте и подключить его к к главному скрипту в операторе uses:

```
Global.pas
---------------
var    FormTitle: string;

begin
end.
```

```
Demo.pas
---------------
uses 'Global';

var   DerectForm:    TCoolForm;
begin
    FormTitle:='Главная    форма    конфигурации';
            DerectForm:=Application.CreateCoolForm('Form1',       true);
    DerectForm.Show;
end.
```

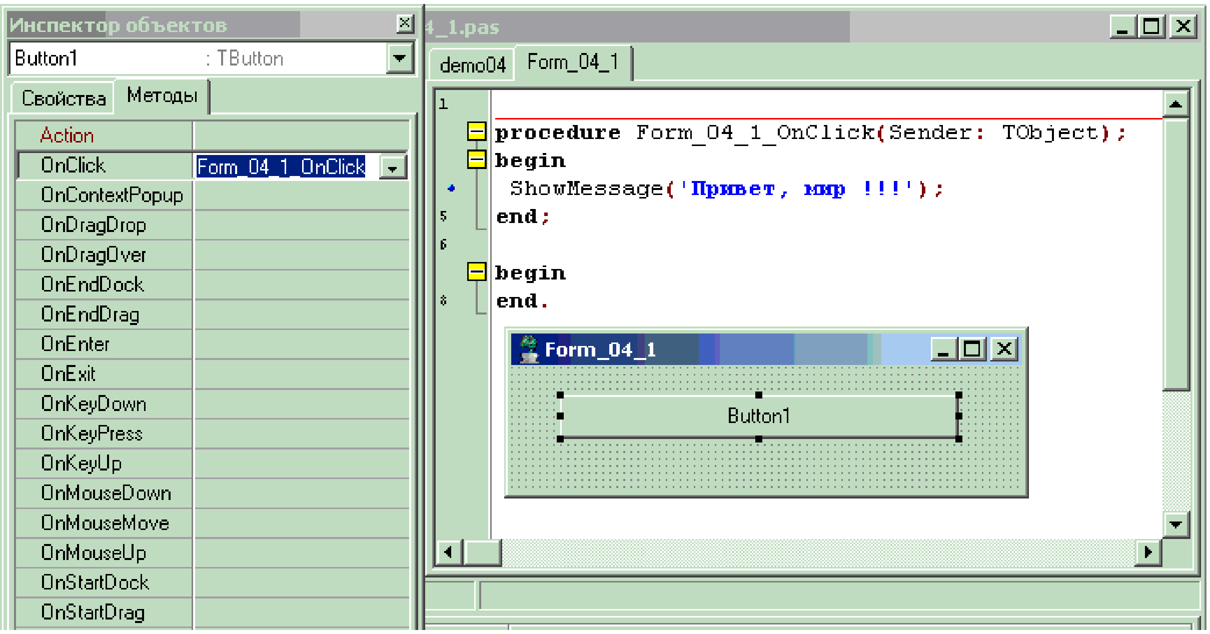
В этом случае объявление переменной FormTitle так же будет доступно скрипту Form1.pas.
Но если добавить в Form1 следующий оператор uses:

```
Form1.pas
----------------
uses 'Global';

procedure    Form1_onCreate(Sender:      TObject);
begin
         Self.Caption:=FormTitle;
end;
```

наш пример будет работать некорректно. Дело в том, что при компиляции скрипта, в него включаются все объявления сделанные в скриптах перечисленных в операторе uses. Это обозначает, что в скрипте Form1.pas создается локальное объявление переменной FormTitle, которое никак не связано с объявлением FormTitle главного скрипта конфигурации. Т.к. объявления оператора uses скрипта обладают более высоким приоритетом, чем объявления сделанные в главном скрипте конфигурации, наш скрипт будет "видеть" локальную копию переменной FormTitle. В принципе, нет никаких причин, для того, что бы объявлять в операторе uses скрипты, уже подключенные к главному модулю конфигурации. Т.к. все объявления сделанные в этих скриптах и так "видны" скрипту формы, поэтому повторное объявление скриптов в операторе uses форм является явно избыточным.

2.2 Конфигурация с визуальной формой

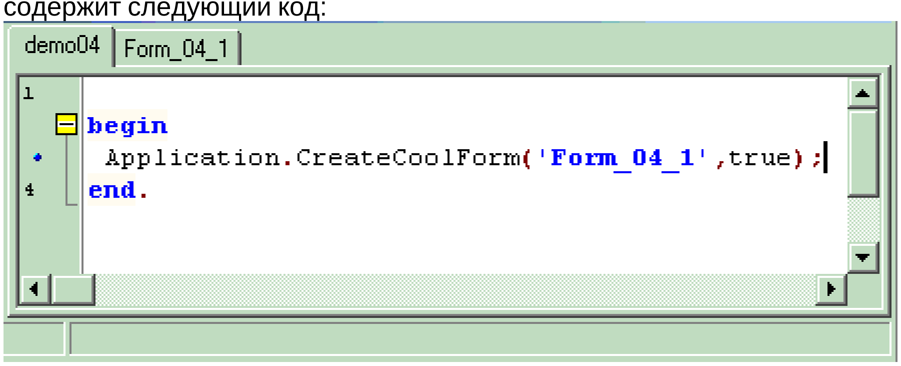
Теперь рассмотрим пример конфигурации содержащей визуальную форму, загрузите демо пример \CONFIG\demo04.cm4. Конфигурация содержит одну форму с расположенной на ней кнопкой:



OnClick: Form_04_1_OnClick

```
procedure Form_04_1_OnClick(Sender: TObject);
begin
 ShowMessage('Привет, мир !!!');
end;

begin
end.
```

Скрипт формы содержит одну процедуру, которая используется в качестве обработчика события OnClick кнопки Button1. Как видно из приведенного рисунка, при нажатии на кнопку, на экран выводится форма с сообщением 'Привет, мир !!!'. Главный модуль конфигурации содержит следующий код:



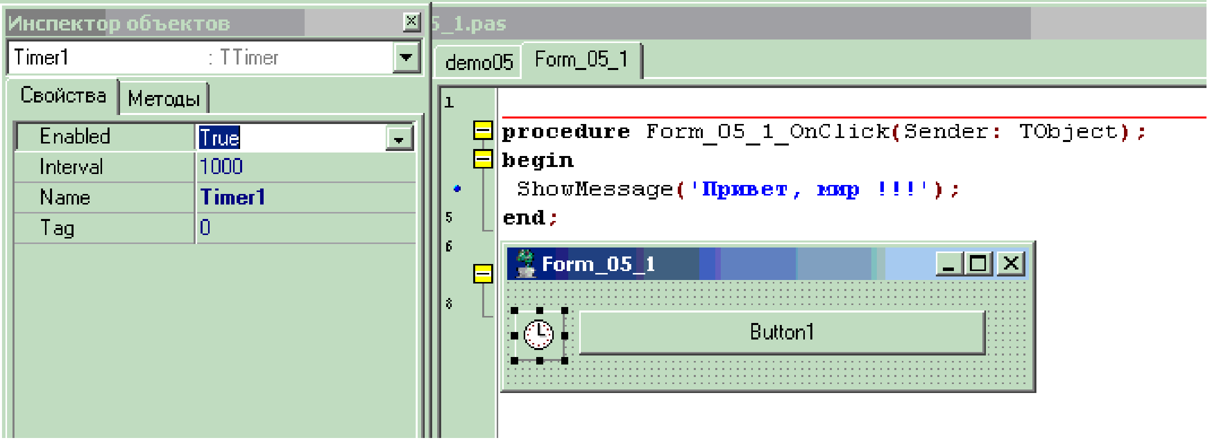
```
begin
 Application.CreateCoolForm('Form_04_1',true);
end.
```

При запуске конфигурации на выполнение, вызывается метод Application.CreateCoolForm, которому передается имя формы. Т.е. на экран будет выведена описанная выше форма, нажатие на кнопку Button1 будет приводить к генерации сообщения 'Привет мир !!!'. При закрытии формы выполнение конфигурации завершается и управление передается ИСР.

Здесь важно отметить следующий факт: первая форма, созданная посредством вызова метода CreateCoolForm автоматически становится главной формой конфигурации и обрабатывает основной поток сообщений Windows, закрытие главной формы приводит к завершению выполнения конфигурации. Для доступа к главной форме конфигурации можно использовать функцию класса TCoolConfig.MainForm. В скриптах доступна глобальная переменная CoolConfig: TCoolConfig (будет подробно описана в следующей главе). Объект конфигурации создается автоматически при загрузке конфигурации в ИСР, в то же самое время инициализируется глобальная переменная CoolConfig. Как известно, в Delphi, для доступа к главной форме приложения используется свойство класса TApplication.MainForm. Между TApplication.MainForm и TCoolConfig.MainForm существует очень важное различие. Дело в том, что при работе в ИСР, запуск конфигурации на выполнение только иммитируется. На самом деле все формы конфигурации создаются в рамках приложения CoolDsgn, поэтому, TApplication.MainForm при работе в ИСР всегда будет указывать на главную форму приложения CoolDsgn. В то же время TCoolConfig.MainForm указывает на главную форму конфигурации. Если при работе главного модуля конфигурации не создается ни одной визуальной формы посредством вызова метода Application.CreateCoolForm, тогда CoolConfig.MainForm=nil. При запуске конфигурации на выполнения посредством приложения-просмотрщика CoolMan (будет подробно описано далее), Application.MainForm всегда равна CoolConfig.MainForm. Во избежание путаницы и для того, что бы корректно получить ссылку на главную форму конфигурации, всегда следует пользоваться функцией CoolConfig.MainForm.

2.3 Создание обработчиков событий

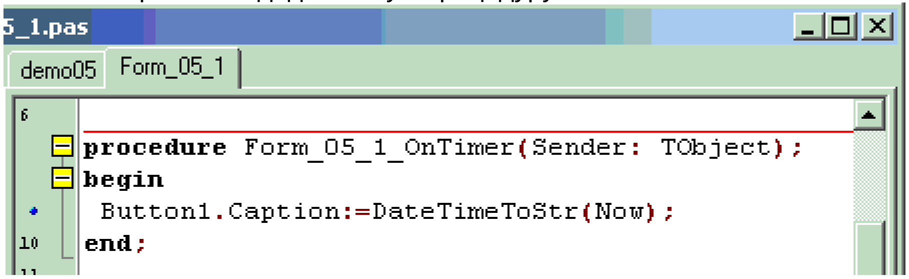
Модернизируем демо пример рассмотренный в предыдущем разделе. На форму положим невизуальный компонент TTimer со страницы System палитры компонент:



Enabled	True
Interval	1000
Name	Timer1
Tag	0

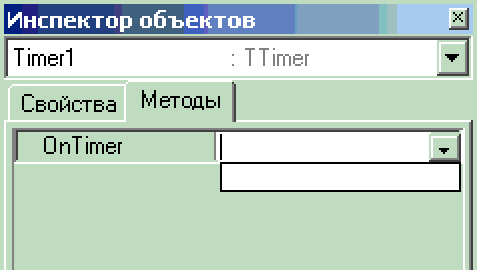
```
procedure Form_05_1_OnClick(Sender: TObject);
begin
 ShowMessage('Привет, мир !!!');
end;
```

Для тех, кто не знаком с набором стандартных компонент Delphi, поясним: компонент TTimer используется для установки таймера в вашем приложении. Свойство Interval определяет период срабатывания таймера в миллисекундах (т.е. 1 секунда = 1000 миллисекунд), свойство Enabled=true, делает таймер активным. При каждом срабатывании таймера инициируется событие OnTimer, его то мы и должны задать.
В тексте скрипта создадим новую процедуру:

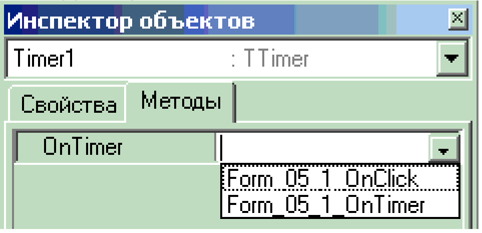


```
procedure Form_05_1_OnTimer(Sender: TObject);
begin
 Button1.Caption:=DateTimeToStr(Now);
end;
```

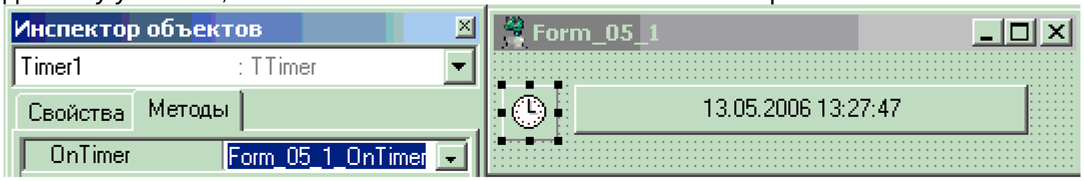
Это обработчик события OnTimer для компонента Timer1. Логика работы процедуры очень проста, при каждом срабатывании таймера, на кнопке Button1 выводятся текущие дата и время. Теперь следует установить обработчик события. Перейдем на закладку Методы инспектора объектов и нажмем кнопку слева от поля ввода имени метода:



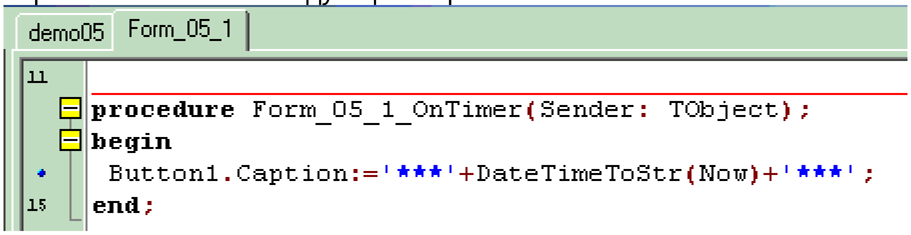
Как видите, выпадающий список не содержит никаких имен методов. Это происходит потому, что список имен формируется посредством просмотра дерева синтаксического разбора скрипта формы. После того, как мы создали в редакторе новую процедуру, следует провести синтаксический разбор скрипта (клавиша <**F9**>), после этого процедура станет доступна в инспекторе объектов. Нажмите <**F9**> и попробуйте снова выбрать подходящий метод из выпадающего списка:



При формировании списка скриптовых процедур для текущего обработчика события, у каждой скриптовой процедуры проверяется список параметров и их типов на предмет соответствия параметрам устанавливаемого метода. В нашем случае метод для обработчика OnTimer имеет всего один параметр типа TObject, наш скрипт содержит две процедуры удовлетворяющие данному условию, они то и появились в списке. Установим обработчик:

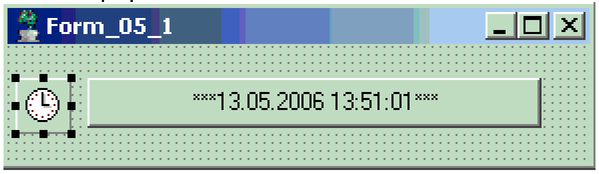


Обратите внимание, на кнопке Button1 сразу же отображается текущие дата и время, причем время обновляется, т.е. таймер работает. Установите свойство Timer1.Enabled=false, часы остановятся, вновь установите Timer1.Enabled=true, время вновь начинает обновляться. С одной стороны в таком поведении нет ничего странного, мы просто отключаем таймер, поэтому обработчик события OnTimer больше не вызывается, затем снова активируем таймер и он начинает вызывать обработчик через установленный интервал времени (Interval=1000). Но если Вы работали в Delphi, то сразу же заметите существенное отличие от среды разработки Delphi, дело в том, что в Delphi установленные обработчики событий срабатывают только в run-time (приложение следует откомпилировать и запустить на выполнение). В нашем случае мы находимся в ИСР CoolManager, но обработчик уже действует. С точки зрения ИСР такое поведение вполне закономерно, перед тем, как установить обработчик, мы выполнили синтаксический разбор скрипта (<**F9**>). Хочу напомнить, что в рамках системы CoolManager это действие аналогично процессу компиляции в Delphi, следовательно скрипт "скомпилирован" и готов к исполнению. Далее, свойство Timer1.Enabled=true, следовательно таймер активен, поэтому и выполняется связанная с обработчиком скриптовая процедура. Модифицируйте обработчик события следующим образом:



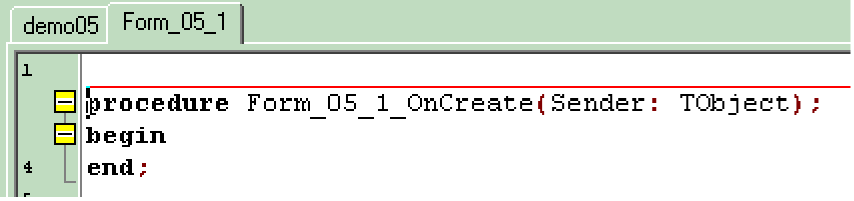
```
procedure Form_05_1_OnTimer(Sender: TObject);
begin
 Button1.Caption:='***'+DateTimeToStr(Now)+'***';
end;
```

Затем нажмите клавишу <**F9**>, обратите внимание на то, как изменится надпись выводимая на кнопке формы:



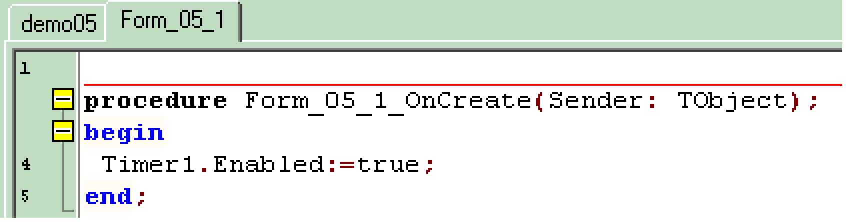
Как видите, внесенные в скрипт изменения тут же были приняты к исполнению.

Теперь остановим таймер (в инспекторе объектов установим свойство Timer1.Enabled=false) и сделаем активной форму, просто щелкнем по ней мышью или выбирем Form_05_1 в списке компонентов инспектора объектов. Перейдем на закладку Методы и дважды щелкнем по полю ввода обработчика OnCreate (это событие вызывается при создании формы). В редакторе скриптов появится шаблон процедуры связанной с обработчиком события onCreate:



```
procedure Form_05_1_OnCreate(Sender: TObject);
begin
end;
```

Напишем следующий код:



```
procedure Form_05_1_OnCreate(Sender: TObject);
begin
 Timer1.Enabled:=true;
end;
```

Как было сказано выше, событие OnCreate происходит сразу же после создания формы, написанный нами код активирует таймер Timer1.

Примечание. Для того, что бы выполнить какие либо действия при создании формы, можно вынести соответствующий код в главный модуль скрипта:

begin
Timer1.Enabled:=true;
end.

И создание обработчика OnCreate и вынесение кода в главный модуль скрипта, являются альтернативными путями, для решения одной и той же задачи - выполнения кода инициализации при создании формы. Каким способом пользоваться - дело вкуса.

Теперь запустим конфигурацию на выполнение, на экране появится форма с кнопкой, часы идут, следовательно, таймер активен. Закроем форму - это приведет к завершению сеанса отладки.

Описанная в данном разделе конфигурация находится в \CONFIG\Demo05.cm4.

2.4 Обмен данными между формами

Пришло время поговорить о принципе "автономности" форм системы CoolManager. В предыдущих разделах было показано, что для создания и вызова формы достаточно написать следующий код:

```
var  Form:  TCoolForm;
begin
        Form:=Application.CreateCoolForm('Form1',      true);
end.
```

При этом, скрипт содержащий подобный код, не "знает" ничего о форме с именем Form1. При вызове метода CreateCoolForm управление передается ИСР CoolManager, которая должна определить местонахождение файла формы, загрузить его и связанные с формой скрипты, создать форму, скомпилировать скрипты и вернуть указатель на созданную форму. Другими словами, при выполнении, конфигурация состоит из набора форм, загружаемых в память приложения вызовом метода CreateCoolForm. В отличие от Delphi все формы являются экземплярами класса TCoolForm, следовательно не содержат никаких сведений о расположенных на них компонентах и определенных в скриптах форм переменных, процедурах и функциях. Каким же образом формы могут обмениваться данными друг с другом? Для этого класс TCoolForm имеет свойство Variables:

```
property     Variables[VarName:  string]  :     variant;
```

и объявляет два события:

```
procedure     OnCoolGetValue(const  VarName:  string;  var    VarData:      Variant);
procedure     OnCoolSetValue(const  VarName:  string;     VarData:      Variant);
```

вот пример иллюстрирующий обращение к свойству Variables формы:

```
var  Form:    TCoolForm;
       i:      integer;
        List:      TStrings;
begin
        Form:=Application.CreateCoolForm('Form1',       true);
         List:=TStringList.Create;
     Form.Variables['Strings']:=List;
   Form.ShowModal;
       i:=Form.Variables['Count']  ;
     List.Free;
   Form.Free;
end;
```

Никакой скрытой логики данный пример не содержит, просто он показывает как следует обращаться к свойству Variables класса TCoolForm. В примере предполагается следующая последовательность действий:

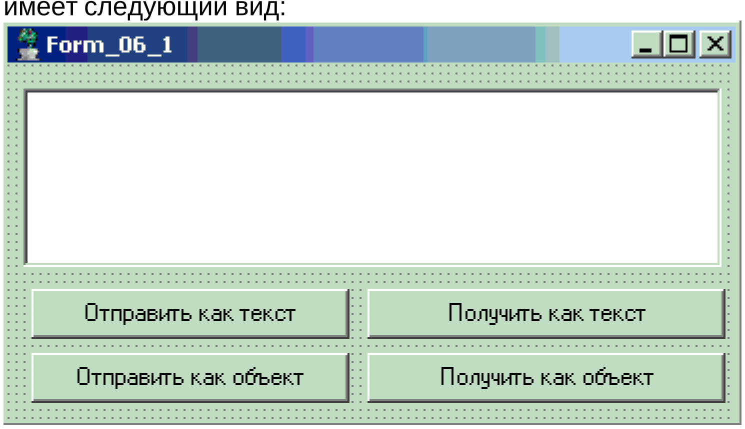
- Создается форма Form
- Создается коллекция строк List
- Форме через свойство Variables передается ссылка на объект List.
- Форма запускается на выполнение в модальном режиме
- Получаем целое число посредством вызова свойства Variables
- Уничтожаем коллекцию строк
- Уничтожаем форму

Механизм обмена данными с помощью свойства Variables напоминает работу с OLE объектами. Существует спецификация, описывающая список переменных доступных при работе с формой. Установка и получение имен переменных производится посредством свойства Variables, при этом в квадратных скобках указывается имя переменной.
Для того, что бы определить какие переменные может получать и передавать форма определены события:

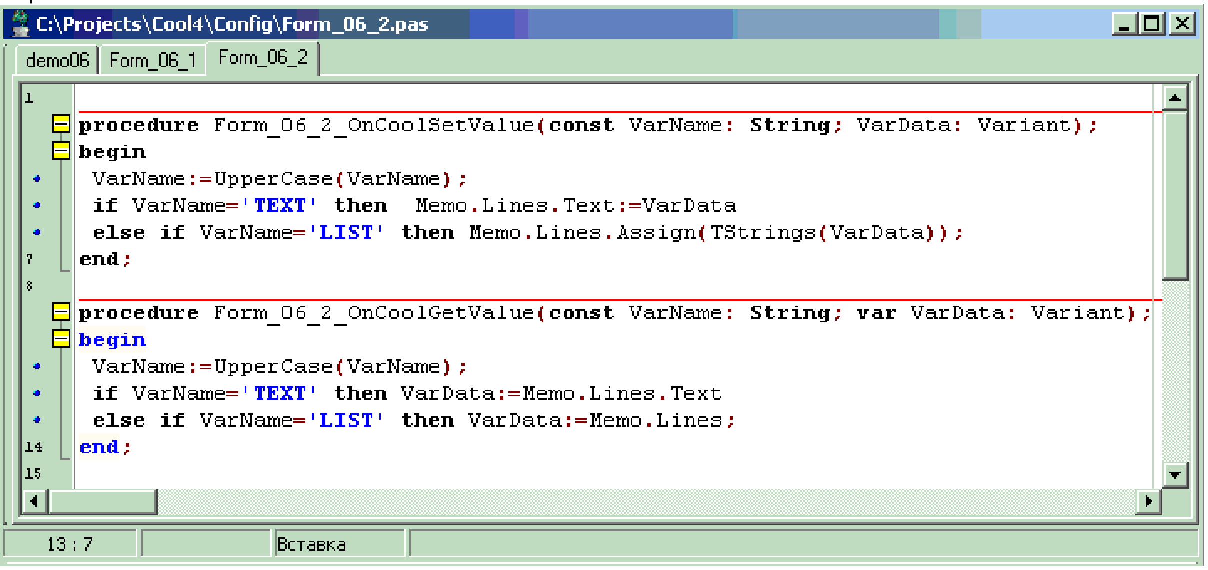
- **OnCoolGetValue** - первый параметр имя переменной, вторым параметром возвращается текущее значение переменной.
- **OnCoolSetValue** - первый параметр имя переменной, во втором параметре передается текущее значение переменной.

При попытке прочитать значение индексированного свойства Variables происходит вызов метода **OnCoolGetValue**, а при попытке установит значение, вызывается метод **OnCoolSetValue**.

Пример передачи и чтения данных посредством свойства Variables содержится в демо конфигурации \CONFIG\Demo06.cm4. Конфигурация содержит две формы, главная форма имеет следующий вид:



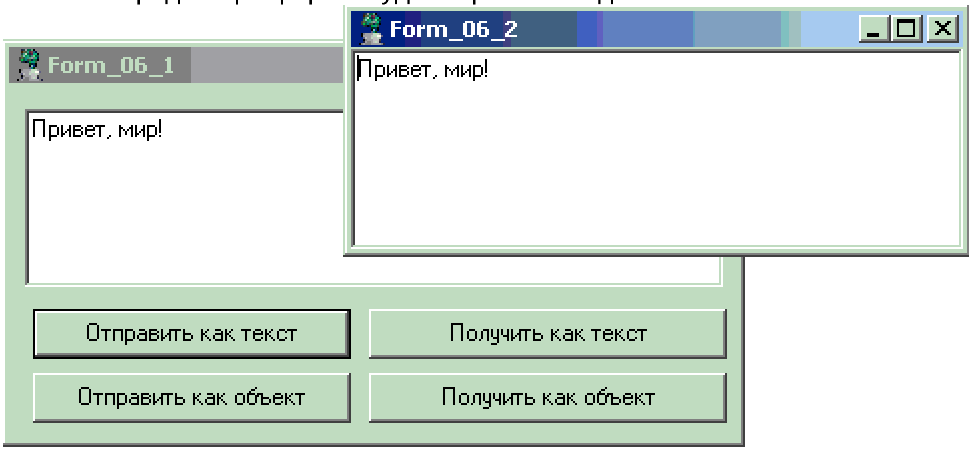
На форме расположен текстовый редактор и четыре кнопки. Вторая форма называется **Form_06_2**, и содержит только поле текстового редактора. У формы Form_06_2 заданы обработчики событий **OnCoolGetValue** и **OnCoolSetValue**:



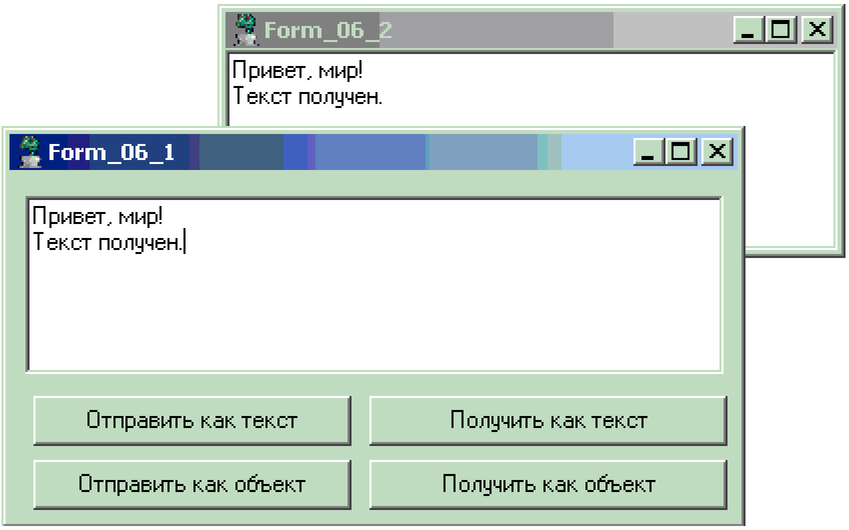
```
procedure Form_06_2_OnCoolSetValue(const VarName: String; VarData: Variant);
begin
 VarName:=UpperCase(VarName);
 if VarName='TEXT' then  Memo.Lines.Text:=VarData
 else if VarName='LIST' then Memo.Lines.Assign(TStrings(VarData));
end;

procedure Form_06_2_OnCoolGetValue(const VarName: String; var VarData: Variant);
begin
 VarName:=UpperCase(VarName);
 if VarName='TEXT' then VarData:=Memo.Lines.Text
 else if VarName='LIST' then VarData:=Memo.Lines;
end;
```

Таким образом для формы определены две переменные Text и List доступные как для чтения, так и для записи. Обе переменные предоставляют доступ к списку строк компонента Memo расположенного на форме, только Text это текстовая строка, а List ссылка на объект класса TStrings.

Запустите конфигурацию на выполнение, введите в текстовом редакторе строку 'Привет, мир!' и нажмите кнопку 'Отправить как текст'. На экран будет выведена форма Form_06_2, а в текстовом редакторе формы будет отражен введенный нами текст:



Добавьте в текстовый редактор формы Form_06_2 строку 'Текст получен.', и нажмите кнопку 'Получить как текст'. Введенный текст будет отражен в редакторе Form_05_1:



Привет, мир!
Текст получен.

Привет, мир!
Текст получен.

Передача и прием текста осуществляются через свойство Variables формы Form_06_2. Кнопки 'Отправить как объект' и 'Получить как объект' выполняют аналогичные операции, только в этом случае через свойство Variables передается не строка текста, а ссылка на объект класса TStrings.

2.5 Создание конфигурации в MDI стиле.

В завершении этой главы приведем еще один пример. Загрузите конфигурацию \CONFIG \Demo07.cm4. Конфигурация содержит две формы:

- **Form_07_1** - главная форма конфигурации
- **Form_07_2** - форма текстового редактора

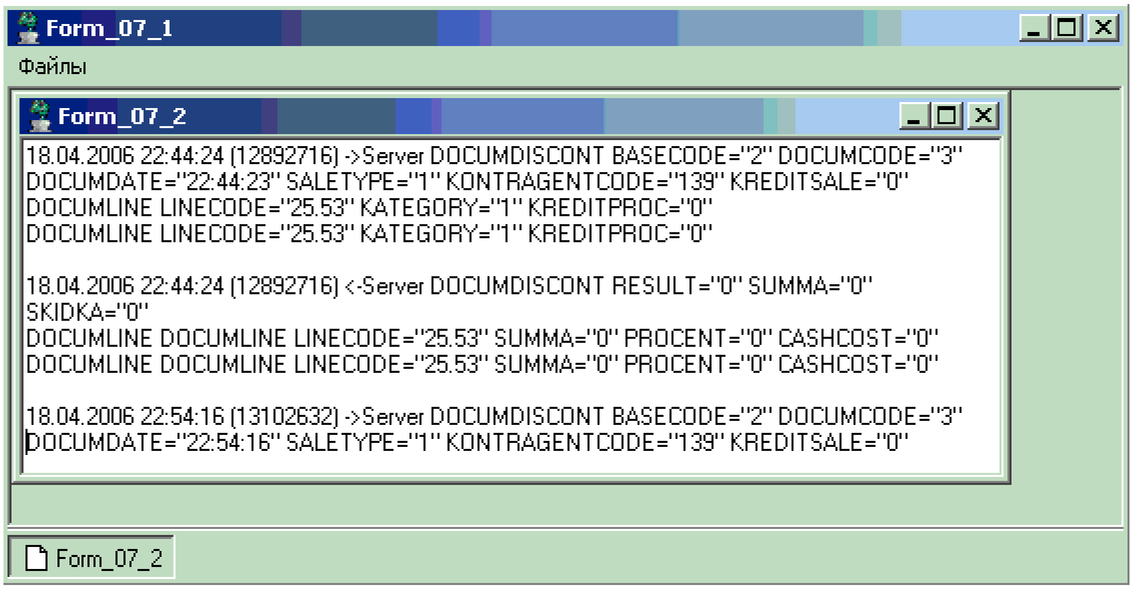
Основное отличие этого примера от рассмотренных ранее заключается в том, что конфигурация реализует MDI приложение. Для тех, кто не знаком с этим термином, поясним, в рамках IDE Delphi можно создавать два типа приложений:

- SDI - Single document interface
- MDI - Multi document interface

MDI приложение состоит из главного окна, все остальные окна являются MDI клиентами, т.е. могут перемещаться и отображаться только в пределах главной MDI формы.

Примечание. Вообще-то API Windows позволяет иметь в одном приложении любое количество MDI окон владельцев и связанных с ними клиентов. Почему в рамках IDE Delphi реализован вариант приложения только с одним MDI владельцем не совсем понятно, но такое ограничение существует и это следует иметь ввиду.

Так уж исторически сложилось, что именно MDI интерфейс пользуется большой популярностью среди разработчиков прикладных систем автоматизации. Наш пример предельно прост. На главной форме расположено меню с двумя пунктами и панель задач для переключения между MDI клиентами приложения. Запустите конфигурацию на выполнение, выберите пункт меню **Файлы\Открыть**. На экран будет выведено диалоговое окно для выбора файла, выберите какой-нибудь файл и нажмите кнопку "**Открыть**". На экран будет выведена MDI форма текстового редактора с содержимым выбранного файла:



```
18.04.2006 22:44:24 (12892716) ->Server DOCUMDISCONT BASECODE="2" DOCUMCODE="3"
DOCUMDATE="22:44:23" SALETYPE="1" KONTRAGENTCODE="139" KREDITSALE="0"
DOCUMLINE LINECODE="25.53" KATEGORY="1" KREDITPROC="0"
DOCUMLINE LINECODE="25.53" KATEGORY="1" KREDITPROC="0"

18.04.2006 22:44:24 (12892716) <-Server DOCUMDISCONT RESULT="0" SUMMA="0"
SKIDKA="0"
DOCUMLINE DOCUMLINE LINECODE="25.53" SUMMA="0" PROCENT="0" CASHCOST="0"
DOCUMLINE DOCUMLINE LINECODE="25.53" SUMMA="0" PROCENT="0" CASHCOST="0"

18.04.2006 22:54:16 (13102632) ->Server DOCUMDISCONT BASECODE="2" DOCUMCODE="3"
DOCUMDATE="22:54:16" SALETYPE="1" KONTRAGENTCODE="139" KREDITSALE="0"
```

На панели задач отразится кнопка для выбора MDI формы. Посредством кнопок расположенных на панели задач можно переключаться между MDI формами приложения. Как видите пример достаточно прост, но вот его реализация содержит несколько интересных ньансов, которые показывают как следует создавать приложения в ИСР CoolManager.
Ниже приводятся скрипты главной формы и формы редактора.

Form_07_1.pas

```
-----------------------
//главная MDI форма. Свойство  формы FormStyle=fsMDIForm
procedure Form_07_1_OnCoolGetValue(const VarName: String; var VarData: Variant);
begin
 if UpperCase(VarName)='TASKBAR' then VarData:=cmTaskBar1;
end;

procedure MenuClose(Sender: TObject);
begin
    Self.Close;
end;

procedure ShowFile(FileName: string) ;
 var Form: TCoolForm;
       List: TStrings;
begin
       Form:=Application.CreateCoolForm('Form_07_2',false);
      List:=Form.Variables['List'] ;
       List.LoadFromFile(FileName);
  Form.Show;
end;

procedure OpenFile(Sender: TObject);
begin
 if OpenDialog1.Execute then ShowFile(OpenDialog1.FileName);
end;

begin
end.
```

Form_07_2.pas

```
-----------------------
//MDI клиент. Свойство  формы FormStyle=fsMDIChild
procedure Form_07_2_OnDestroy(Sender: TObject);
begin
     GetTaskBar.Remove(Self);
end;
```

```
function GetTaskBar: TcmTaskBar;
 var Form: TCoolForm;
begin
      Form:=Application.FindForm('Form_07_1') ;
      Result:=Form.Variables['TaskBar'] ;
end;

procedure Form_07_2_OnClose(Sender: TObject; var Action: TCloseAction);
begin
    Action:=caFree;
end;

procedure Form_07_2_OnCoolGetValue(const VarName: String; var VarData: Variant);
begin
 if UpperCase(VarName)='LIST' then VarData:=Memo.Lines;
end;

begin
     GetTaskBar.Add(Self,0) ;
end.
```

С пунктом меню главной формы **Файл\Открыть** связана процедура OpenFile, она содержит следующий код:

```
 if OpenDialog1.Execute then ShowFile(OpenDialog1.FileName);
```

Используется компонент класса TOpenDialog для выбора имени файла, если файл был выбран, вызывается процедура ShowFile и ей передается имя файла.

В процедуре ShowFile создается новая форма текстового редактора:

```
Form:=Application.CreateCoolForm('Form_07_2',false);
```

Обратите внимание, второй параметр метода равен false, следовательно, каждый раз при вызове метода будет создаваться новая форма текстового редактора.

Далее получаем ссылку на список строк компонента Memo созданной формы:

```
List:=Form.Variables['List'] ;
```

Ссылку получаем через свойство Variables, в скрипте Form_07_2 определен обработчик события OnCoolGetValue, в котором и производится возврат ссылки.

Затем используя метод списка LoadFromFile в форму редактора загружается выбранный файл:

```
List.LoadFromFile(FileName);
```

Рассмотрим код написанный для отражения кнопки окна на панели задач.

Главный модуль скрипта Form_07_2 содержит следующий оператор:

```
GetTaskBar.Add(Self,0) ;
```

Используется функция GetTaskBar для того, что бы получить ссылку на панель задач главной формы, затем вызывается метод панели задач Add. Методу передаются два параметра: ссылка на текущую форму и номер картинки связанной с кнопкой. Обратите внимание на то, как передается ссылка на текущую форму. В каждом скрипте формы определена переменная Self: TCoolForm, которая и указывает на форму с которой связан скрипт.

Рассмотрим подробнее функцию GetTaskBar. Вначале, мы получаем ссылку на главную форму приложения:

```
Form:=Application.FindForm('Form_07_1') ;
```

Затем получаем ссылку на панель задач:

```
Result:=Form.Variables['TaskBar'] ;
```

Ссылка передается через свойство Variables, в скрипте Form_07_1 определен обработчик события OnCoolGetValue для возврата ссылки на панель задач.

В скрипте Form_07_2 также определен обработчик события OnDestroy, который вызывается при уничтожении формы:

```
GetTaskBar.Remove(Self);
```

в обработчике вызвается метод Remove панели задач, ему передается ссылка на форму. Метод удаляет с панели задач кнопку, связанную с указанной формой.

И наконец, определен обработчик события OnClose для Form_07_2:

```
Action:=caFree;
```

присваивание параметру Action значения caFree указывает на то, что после закрытия форма должна быть удалена из памяти.

Глава III

Работа с пакетами

3 Работа с пакетами

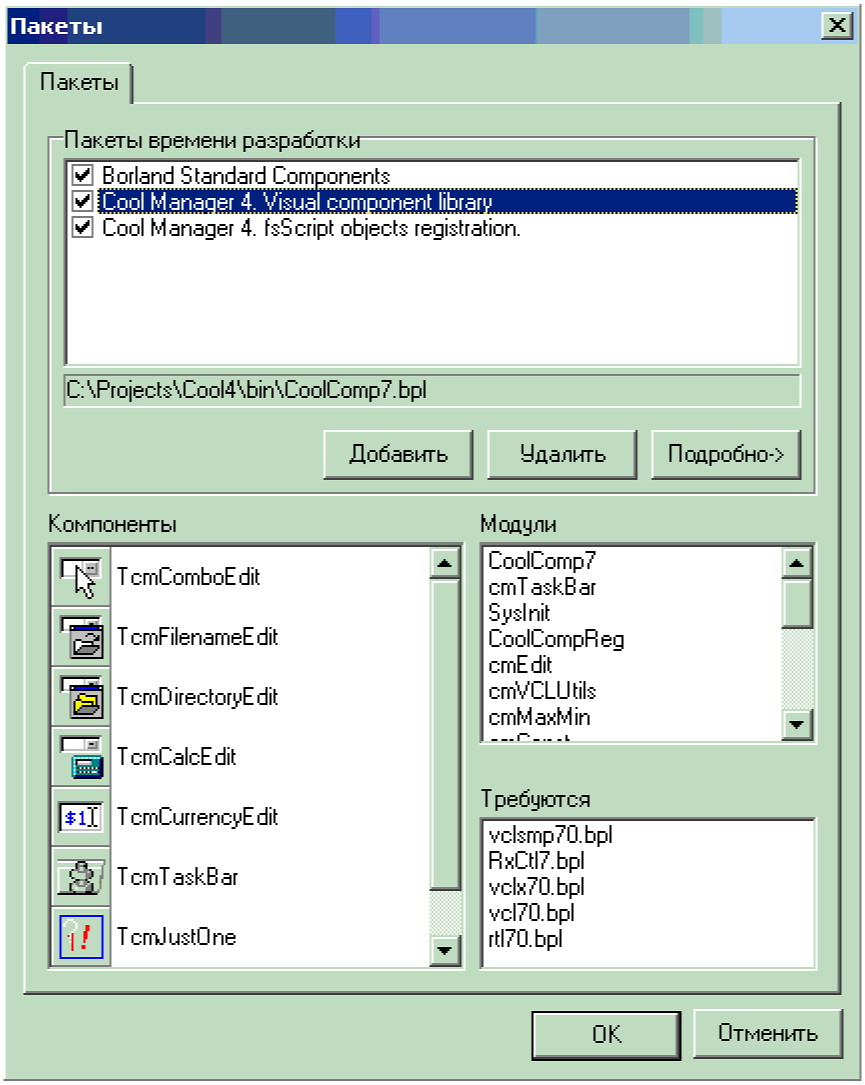
Как было сказано ранее, ИСР CoolManager позволяет загружать bpl пакеты во время работы приложения. Это стандартные пакеты загружаемые в среду Delphi и содержащие библиотеки классов и компонент. Причем, так же как в Delphi, после загрузки пакета содержащиеся в нем компоненты появляются в палитре компонентов и могут использоваться при создании визуальных форм. Кроме этого, пакеты могут содержать расширения для ИСР CoolManager. Поэтому, можно разделить пакеты на две группы:

- **стандартные** - стандартные Delphi пакеты, содержат библиотеки компонент и классов.
- **служебные** - содержат информацию для расширения возможностей встроенного языка ИСР и классы для использования в конфигураторе системы.

Разделение пакетов на группы довольно условное, в принципе любой пакет может содержать как библиотеки компонент, так и расширения для ИСР CoolManager. В отличие от IDE Delphi список загруженных в ИСР пакетов и настройки палитры компонентов хранятся в XML файле, а не в реестре Windows. Файл с описанием настроек палитры компонентов называется comps.pal и располагается в каталоге, из которого запускается ИСР CoolManager.

3.1 Подключение пакетов

Для загрузки пакетов среды используется пункт меню **Настройка\Установка пакетов**. Менеджер пакетов ИСР CoolManager имеет следующий вид:



☑ Borland Standard Components
☑ Cool Manager 4. Visual component library
☑ Cool Manager 4. fsScript objects registration.

C:\Projects\Cool4\bin\CoolComp7.bpl

TcmComboEdit
TcmFilenameEdit
TcmDirectoryEdit
TcmCalcEdit
TcmCurrencyEdit
TcmTaskBar
TcmJustOne

CoolComp7
cmTaskBar
SysInit
CoolCompReg
cmEdit
cmVCLUtils
cmMaxMin

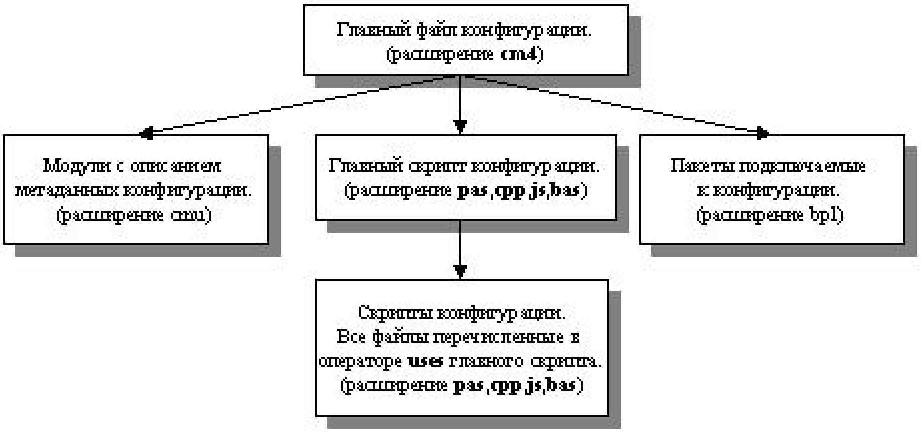
vclsmp70.bpl
RxCtl7.bpl
vclx70.bpl
vcl70.bpl
rtl70.bpl

Вверху формы расположен список загруженных пакетов. Для того, что бы загрузить новый пакет, следует нажать кнопку "Добавить", для выгрузки пакета следует нажать кнопку "Удалить". Если снять метку в списке, то пакет будет выгружен из ИСР, но запись о нем останется в списке. Внизу формы расположен список компонент содержащихся в пакете, а так же списки модулей и используемых bpl пакетов. Пакеты указанные в списке будут автоматически загружать при каждом запуске ИСР CoolManager.

Примечание. При загруке пакета, в скриптовом языке будут доступны только компоненты отраженные в палитре и их published свойства. Для того что бы сделать доступными в скриптах public свойства и методы компонентов, классы, процедуры и функции содержащиеся в пакете, нужно написать дополнительный код для их регистрации. Этот вопрос будет подробно рассмотрен в руководстве программиста.

3.2 Работа с конфигуратором

Пришло время более подробно рассмотреть структуру конфигурации, создаваемой при помощи ИСР CoolManager. Приведенный ниже рисунок отображает все элементы конфигурации и существующие между ними взаимосвязи:


Главный файл конфигурации.
(расширение cm4)
Модули с описанием
метаданных конфигурации.
(расширение cm1)
Главный скрипт конфигурации.
(расширение pas ,cpp js,bas)
Пакеты подключаемые
к конфигурации.
(расширение bpl)
Скрипты конфигурации.
Все файлы перечисленные в
операторе uses главного скрипта.
(расширение pas ,cpp js,bas)

В предыдущих разделах мы рассматривали скрипты и пакеты подключаемые к конфигурации, а так же их функциональное назначение. В этом разделе речь пойдет о метаданных конфигурации. Любая система автоматизации содержит информацию о структуре используемых ею данных, например, информация о справочниках, документах, регистрах, подключаемом оборудовании и т.д. Понятно, что такого рода данные должны быть хорошо структурированы и представлены в удобном для просмотра и поиска информации виде. Классическим примером является конфигуратор системы 1С Предприятие, очень мощный и удобный инструмент, позволяющий максимально упростить работу с метаданными прикладной задачи.

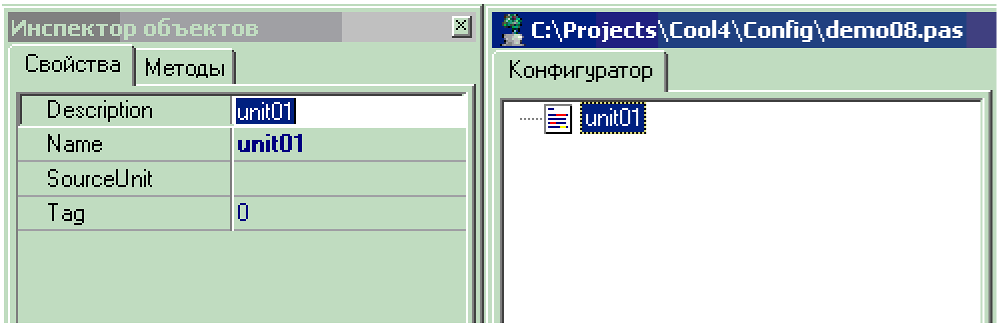
Конфигурация, создаваемая в CoolManager, так же может содержать описание метаданных, а ИСР имеет встроенный инструментарий для просмотра и редактирования метаданных. Для доступа к метаданным конфигурации, на уровне скриптовых языков, определен специальный класс TCoolConfig. При запуске конфигурации на выполнение, автоматически создается глобальная переменная CoolConfig: TCoolConfig, доступная всем скриптам конфигурации.

В основе метаданных ИСР CoolManager лежит стандартная модель компонентов Delphi. Все классы метаданных наследуются от базового класса:

```
TcmUnitComponent = class(TComponent)
```

Метаданные конфигурации представлены в виде иерархической древовидной структуры, с четко регламинтированным местоположением каждого доступного класса . Классы метаданных создаются в Delphi и подключаются к конфигурации в виде служебных bpl пакетов (процесс создания подключаемых к конфигуратору классов подробно рассмотрен в Руководстве программиста). Так как все метаклассы наследуются от TComponent, для редактирования свойств и событий объектов, отображаемых в конфигураторе, используется Object Inspector. Внутри конфигурации метаобъекты разбиты на группы, которые сохраняются в отдельных файлах. Это позволяет создавать блоки метаданных, а затем подключать их к другим конфигурациям. Для просмотра и редактирования метаданных в ИСР включен встроенный конфигуратор.

Создайте новую конфигурацию. Для вызова конфигуратора воспользуйтесь пунктом меню **Вид \Конфигуратор**, либо нажмите комбинацию клавиш <**Ctrl-Alt-F11**>. После того, как конфгуратор появится на экране, следует создать новый модуль метаданных. Выберите пункт меню **Конфигурация\Добавить модуль в конфигурацию**, или нажамите пару клавиш <**Shift-F11**>. На экран будет выведено диалоговое окно для задания имени файла модуля, назовите модуль Unit01.cm4 и сохраните его в том же каталоге, что и главный файл конфигурации:



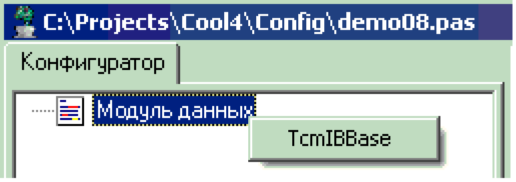
Description	unit01
Name	unit01
SourceUnit	
Tag	0

В окне конфигуратора появится запись с именем модуля, а в инспекторе объектов будут отражены свойства объекта. Все объекты метаданных являются компонентами, поэтому их свойства и методы могут редактироваться так же, как и свойства любого компонента расположенного на форме. Можно сказать, что классы метаданных являются альтернативой использованию невизуальных компонентов располагаемых на форме или в модуле данных. В отличие от невизуальных компонент, компоненты метаданных не связаны с формой. Сейчас конфигурация содержит один объект, который является визуальным представлением модуля метаданных, напомним, что каждый модуль расположен в отдельном файле. Созданный объект является экземпляром класса TcmUnitComponent, базового класса для всех меткаклассов конфигурации. Для редактирования в инспекторе объектов доступны следующие свойства и методы:

- Name - имя компонента (свойство класса TComponent). При создании нового объекта имя генерируется автоматически, все объекты метаданных должны иметь уникальные имена.
- **Tag** - свойство определено в классе TComponent.
- **Description** - произвольный текст связанный с объектом, отображается в конфигураторе. Введите текст '**Модуль данных**'.
- **OnActivate** - событие вызываемое при запуске конфигурации (расположено на закладке Методы).
- **SourceUnit** - по умолчанию, все обработчики событий метаобъектов, создаются в главном скрипте конфигурации. Если требуется назначить для вывода другой скрипт, его имя должно быть указано в этом свойстве. Скрипт, введенный в свойстве SourceUnit, должен быть указан в операторе **uses** главного модуля конфигурации.

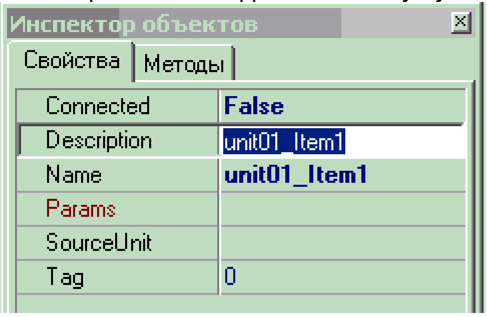
Вот и все, что можно создать в конфигураторе без дополнительных служебных пакетов. В отличии от 1С, конфигуратор ИСР CoolManager не имеет подмножества "зашитых" в приложение метаклассов. Вместо этого, предлагается полностью открытая архитектура. Разработчик сам может решить, какие классы метаданных ему требуются, реализовать их в виде bpl пакетов и подключить к конфигурации, либо воспользоваться разработками других программистов. Классы метаданных могут разрабатываться и распространяться точно так же, как распространяются библиотеки различных VCL компонентов.

Продемонстрируем это на примере. Базовая поставка системы CoolManager включает пакет CoolIB**X**.dpk, содержащий классы для доступа к базам InterBase. Исходые тексты пакета расположены в каталоге \source\ib, скомпилированный bpl пакет расположен в каталоге **\BIN**. Подключим его к нашей конфигурации. Выберем пункт меню **Настройка\Установка компонентов**, нажмем кнопку "**Добавить**" и загрузим пакет CoolIB**X**.bpl (под буквой X подразумевается цифра обозначающая версию Delphi) из каталога **\BIN.**
Вернемся к конфигуратору. Щелкнем правой кнопкой мыши по компоненту Unit01:

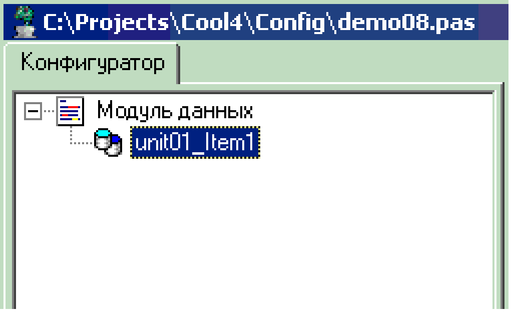


На экране появится PopupMenu со списком метаклассов подчиненными классу

TcmUnitComponent. Здесь требуются дополнительные пояснения. Каждый метакласс, регистрируемый в системе, имеет владельца. Когда мы щелкаем правой клавишей мыши по компоненту в конфигураторе, определяется класс выбранного компонента, затем сканируется список зарегистрированных метаклассов и выбираются все классы подчиненные классу выбранного компонента. Список найденных классов и отображается в PopupMenu конфигуратора. Загруженный нами пакет CoolIB.bpl содержал метакласс TcmIBBase, подчиненный классу TcmUnitComponent, поэтому он и отразился в PopupMenu нашего модуля. Итак, щелкнем по единственному пункту PopupMenu:



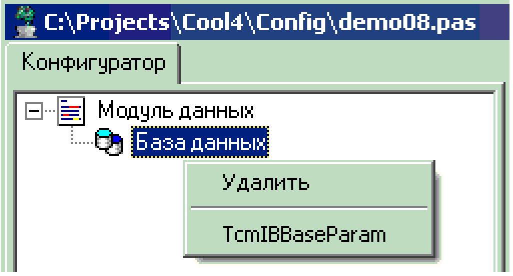
Connected	False
Description	unit01_Item1
Name	unit01_Item1
Params	
SourceUnit	
Tag	0



Забегая вперед скажем, что класс TcmIBBase предназначен для установки соединения с InterBase базой данных. По отношению к классу TcmUnitComponent, от которого он был наследован, TcmIBBase содержит два новых свойства:

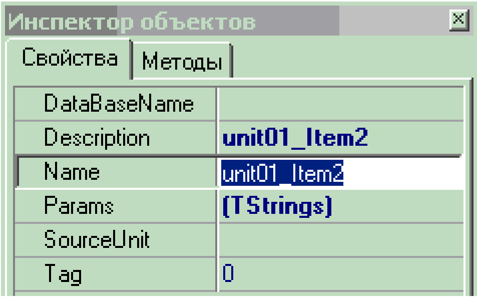
- Connected - свойство для установки\разрыва соединения с базой данных.
- Params - параметры соединения.

Присвоим свойству Description строку '**База данных**'. Теперь щелкнем правой кнопкой мыши по этому компоненту:

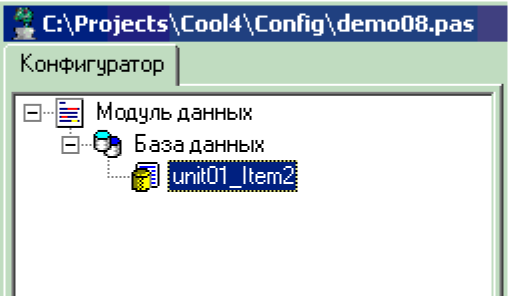


PopupMenu содержит два пункта:

- **Удалить** - удалить объект из конфигурации.
- **TcmIBBaseParam** - компонент подчиненный TcmIBBase. Выберем этот пункт.

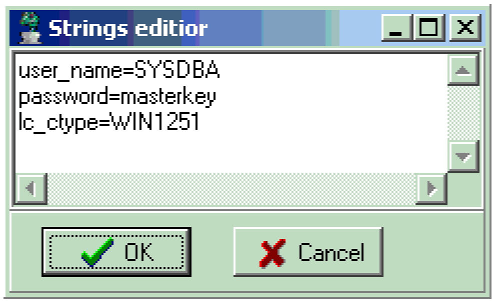


DataBaseName	
Description	unit01_Item2
Name	unit01_Item2
Params	(TStrings)
SourceUnit	
Tag	0



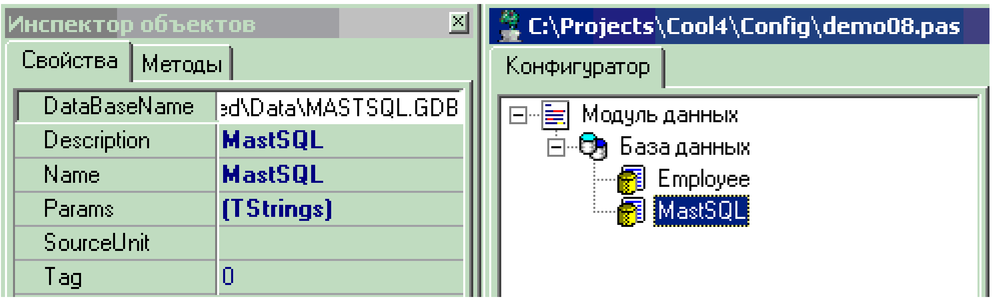
Класс **TcmIBBaseParam** предназначен для настройки параметров соединения с базой данных. Класс наследован от TcmUnitComponent и имеет два уникальных свойства:

Params - список параметров подключения. По умолчанию установлены следующие значения:



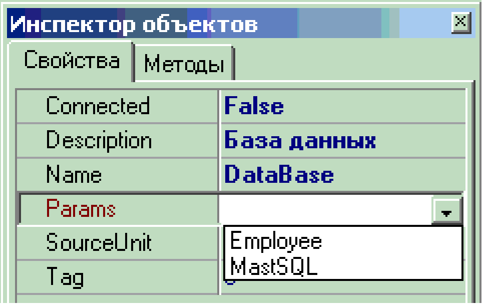
```
user_name=SYSDBA
password=masterkey
lc_ctype=WIN1251
```

если на вашем компьютере пароль для пользователя SYSDBA отличается от установленного по умолчанию (masterkey), следует указать правильное значение.
DataBaseName - это имя базы данных с которой следует устанавливать соединение.
Если на вашем компьютере установлена одна из версий Delphi, и установлена в каталоги по умолчанию на диске C, то в каталоге C:\Program Files\Common Files\Borland Shared\Data\ расположена база данных EMPLOYEE.GDB, введите её имя и полный путь в свойство DataBaseName. В этом же каталоге расположена еще одна база MastSQL.gdb, создайте в конфигураторе ещё один компонент подчиненный TcmIBBase и настройте его для подключения к этой базе:



DataBaseName	ɜd\Data\MASTSQL.GDB
Description	MastSQL
Name	MastSQL
Params	(TStrings)
SourceUnit	
Tag	0

Вернемся к компоненту TcmIBBase, выделите его в конфигураторе и откройте выпадающий список у свойства Params:



Connected	False
Description	База данных
Name	DataBase
Params	
SourceUnit	
Tag	

Как видите, в списке присутствуют оба подчиненных компонента. Выберите базу данных Employee, и щелкните по свойству Connected, если соединение будет успешно установлено, свойство примет значение true, иначе на экране появится сообщение об ошибке. Проверьте таким способом соединения с обоими базами, просто меняя значение свойства Params.

3.3 Класс TCoolConfig

В этом разделе будет раказано о том, как получить доступ к метаданным из скриптового языка. Для скриптов определена глобальная переменная CoolConfig, которая позволяет получить доступ к свойствам конфигурации. Для работы с метаобектам конфигурации TCoolConfig предоставляет следующие методы:

```
property    UnutCount:    integer;
```

Возвращает колчисество модулей метаданных подключенных к конфигурации.

property Units[**Index**: integer]: TcmUnitComponent;
Возвращает ссылку на модуль метаданных по порядковому номеру.

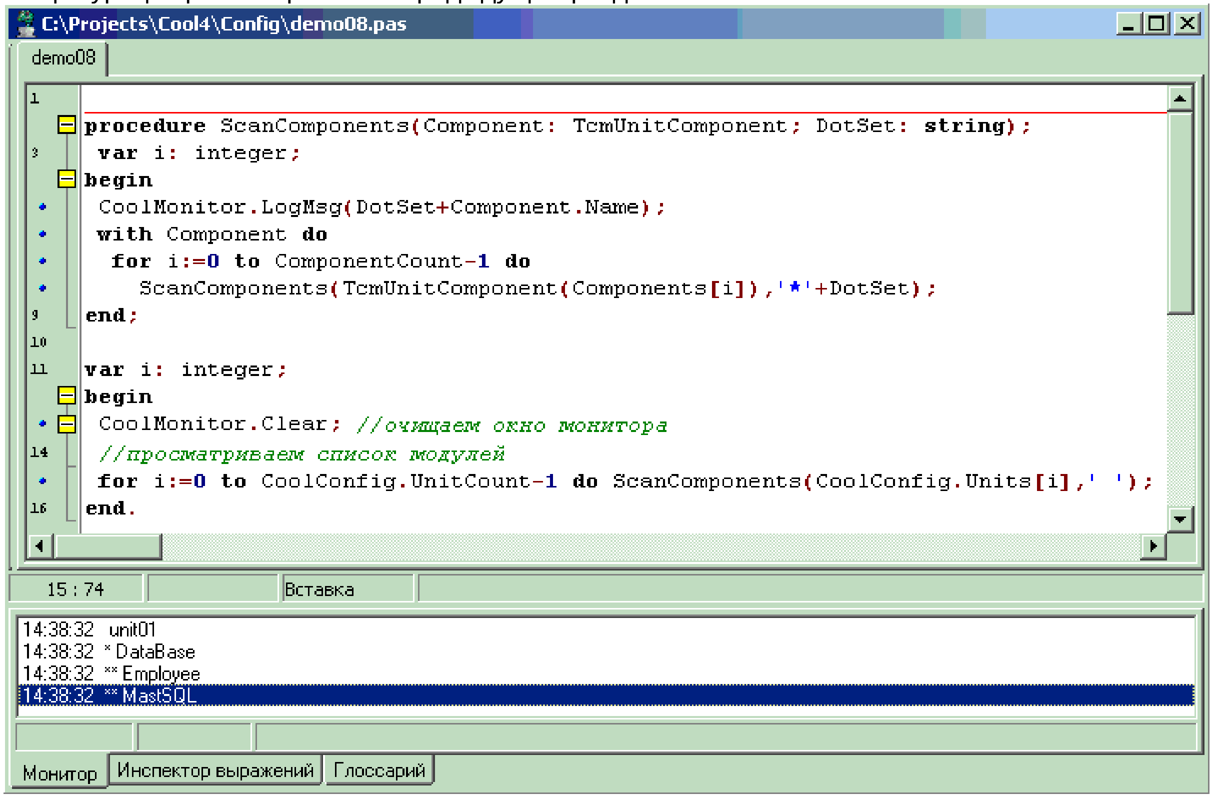
function GetComponentByName(**Name**: **string**): TcmUnitComponent;
Функция возвращает ссылку на метообъект по его имени.

procedure GetByClassName(AClass: **string**; List: TList);
Процедуре передается имя класса, в списке List возвращаются ссылки на все метаобъекты указанного класса.

Так как все метаклассы наследуются от TComponent, для навигации по дереву метаданных можно воспользоваться стандартными свойствами класса TComponent:

- **property** Owner: TComponent - получить ссылку на владельца компонента.
- **property** ComponentCount: integer - количество подчиненных компонентов.
- **property** Components[**Index**: integer]: TComponent - получить подчиненный компонент по порядковому номеру.

Для демонстрации приемов работы с TCoolConfig, модернизируем главный скрипт конфигурации рассмотренной в предыдущем разделе:



```
procedure ScanComponents(Component: TcmUnitComponent; DotSet: string);
 var i: integer;
begin
 CoolMonitor.LogMsg(DotSet+Component.Name);
 with Component do
  for i:=0 to ComponentCount-1 do
    ScanComponents(TcmUnitComponent(Components[i]),'*'+DotSet);
end;

var i: integer;
begin
 CoolMonitor.Clear; //очищаем окно монитора
 //просматриваем список модулей
 for i:=0 to CoolConfig.UnitCount-1 do ScanComponents(CoolConfig.Units[i],' ');
end.
```

```
14:38:32  unit01
14:38:32 * DataBase
14:38:32 ** Employee
14:38:32 ** MastSQL
```

В главном модуле, в цикле просматриваем список модулей определенных в конфигурации и вызываем процедуру ScanComponents. Процедура ScanComponents получает ссылку на компонент конфигурации, вызывает процедуру CoolMonitor.LogMsg для вывода имени компонета в окне монитора сообщений, затем просматривает список всех подчиненных компонентов и рекурсивно вызывает саму себя, передавая ссылку на текущий подчиненный компонент. В результате работы процедуры, на мониторе сообщений, выводятся все компоненты конфигурации. Второй параметр процедуры ScanComponents служит для отображения уровня вложенности текущей компоненты конфигурации.

Описанная конфигурация находится в \CONFIG\Demo08.cm4.

3.4 Классы для работы с базами данных

В разделе "Работа с конфигуратором" была рассмотрена конфигурация использующая служебный пакет CoolIB.bpl, было показано как создавать объекты конфигурации и настраивать их свойства. В этом разделе описываются классы, объявленные в пакете CoolIB.

В базовой поставке система CoolManager содержит спецификации трех абстрактных классов доступа к данным:

TcmTransaction	Класс для работы с транзакциями. Содержит методы для создания, сохранения, отмены и активации транзакций.
TcmCursor	Класс для работы с курсорами. Позволяет на основе SQL запроса формировать выборку данных, передвигаться по списку полученных записей и извлекать значения полей каждой записи.
TcmDataBase	Класс для доступа к базе данных. Позволяет настраивать параметры базы данных, устанавливать соединение с базой данных, выполнять запросы обновления данных.

На основе спецификации свойств и методов, описываемых вышеперечисленными классами, создаются классы доступа к конкретному серверу баз данных. Смысл введения подобного промежуточного слоя в том, что бы стандартизировать работу с базами данных на уровне скриптов. Можно перекрыть абстрактные классы доступа и написать реализацию для работы с любым SQL сервером. При создании прикладной задачи, разработчик вызывает методы вышеперечисленных классов, поэтому, не важно какая конкретно библиотека подключена, лишь бы классы доступа к данным были унаследованы от вышеперечисленных.

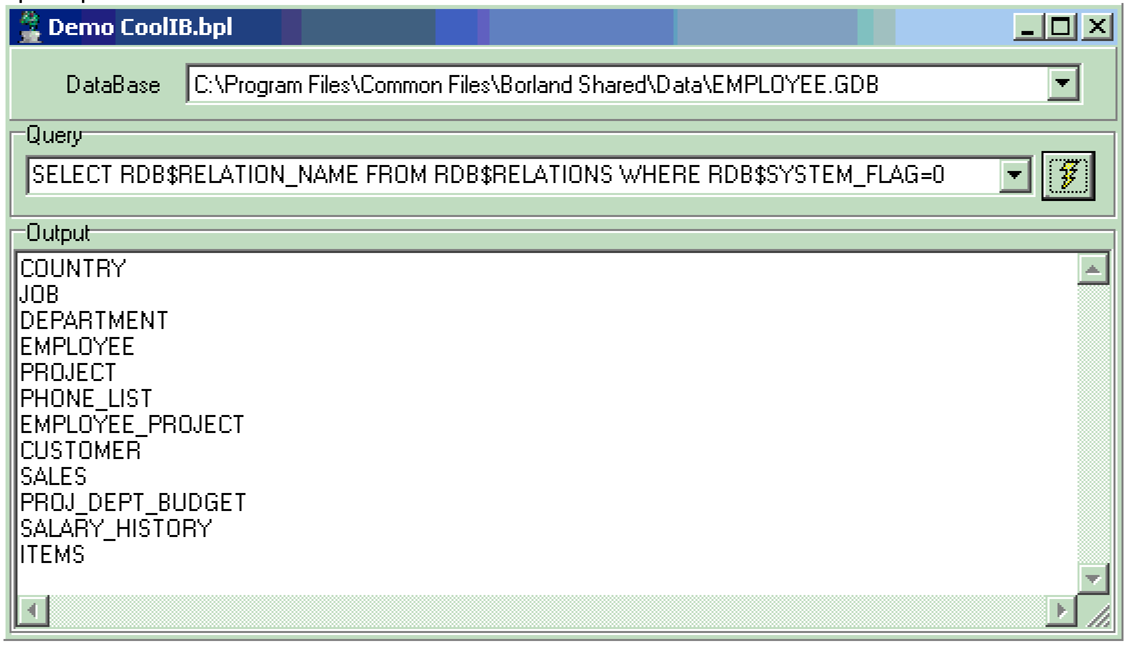
Примером реализации классов доступа к данным, является пакет CoolIB. Пакет используется для доступа к серверу баз данных InterBase. Классы пакета используют IBX компоненты. При подключении пакета к конфигурации становятся доступными следующие классы:

TcmDataBase	Наследуется от TcmDataBase, используется для установки соединения с InterBase сервером. Создан с использованием TIBDataBase.
TcmCursor	Наследуется от TcmCursor. Основан на TIBSql.
TcmTransaction	Наследуется от TcmTransaction. Основан на TIBTransaction.
TcmIBBase	Метакласс конфигурации. Оболочка для визуального представления класса TcmIBDataBase в конфигураторе. Для доступа к объекту базы данных используется метод GetDataBase.
TcmIBBaseParams	Метакласс конфигурации, подчинен TcmIBBase. Используется для хранения параметров базы данных.

Для получения более детальной информации о свойствах и методах вышеперечисленных классов, воспользуйтесь глоссарием ИСР CoolManager.

По своим возможностям, абстрактные классы доступа к данным очень ограничены, например, не поддерживают работу с наборами данных TDataSet. Но если говорить о доступе к базам данных в системе автоматизации учета, то в любом случае, доступ к объектам базы реализуется на более высоком уровне. Разработчик должен оперировать такими понятиями как справочник, документ, регистр, операция и т.д., а не писать запросы для прямого извлечения данных из базы. Но есть ряд задач, где иметь прямой доступ к базе данных всё же необходимо, например, при переносе данных из одной системы в другую. В этом случае упрощенный подход, реализованный в виде абстрактных классов доступа к данным вполне подходит, можно создать несколько реализаций для доступа к различным серверам, но с точки зрения разработчика ничего не изменится - он будет обращаться к одними и тем же классам и свойствам. Конечно, можно использовать богатый набор библиотек доступа к данным включенных в Delphi - IBX, ADO, BDE, dbExpress и т.д., но для упрощенного доступа к базе все их возможности явно излишни, кроме того, для полноценного использования этих библиотек в скриптах, придется создавать дополнительные служебные пакеты. Есть еще одна причина, по которой мы уделяем этому вопросу столько внимания. Абстрактные классы для доступа к базам данных, наглядно иллюстрируют подход, используемый в CoolManager, при создании предметно-ориентированных классов и библиотек. Сперва создается абстрактный класс, представляющий собой некую спецификацию, весь код содержится в классах наследниках.

Для демонстрации работы с классами доступа к базам данных, создан специальный демо-пример: \CONFIG\Demo09.cm4:



C:\Program Files\Common Files\Borland Shared\Data\EMPLOYEE.GDB

SELECT RDB$RELATION_NAME FROM RDB$RELATIONS WHERE RDB$SYSTEM_FLAG=0

```
COUNTRY
JOB
DEPARTMENT
EMPLOYEE
PROJECT
PHONE_LIST
EMPLOYEE_PROJECT
CUSTOMER
SALES
PROJ_DEPT_BUDGET
SALARY_HISTORY
ITEMS
```

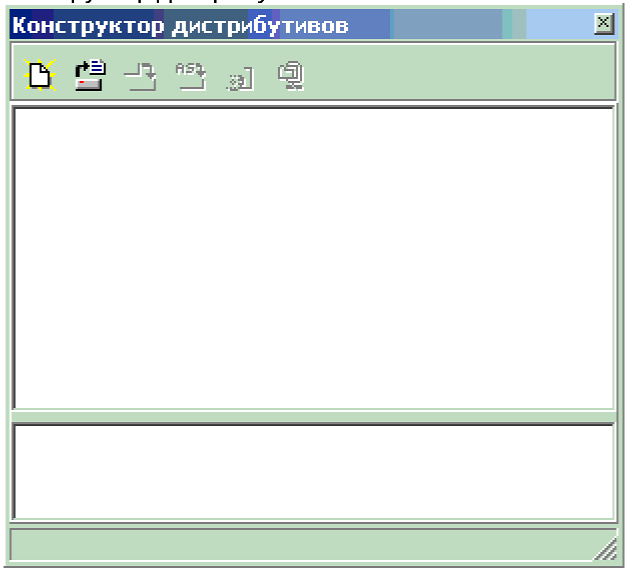
В примере показано, как получить доступ к метаобъекту базы данных описанном в конфигураторе, создать транзакцию и открыть курсор для доступа к базе данных, а так же как работать с данными курсора. Пакет использует модуль метаданных unit01.cmu, создание которого, рассматривалось в разделе посвященному работе с конфигуратором, это наглядный пример того, как модуль данных, созданный в рамках одной конфигурации, может быть использован при создании другой.

3.5 Конструктор дистрибутивов

Одним из важнейших вопросов, связанных с сопровождением прикладных программ, является вопрос создания дистрибутивов. Особенность CoolManager состоит в том, что все приложения системы построены по принципу динамического связывания, т.е. компилируются без bpl пакетов. Об этом вы можете судить по размеру exe файлов расположенных в каталоге \BIN. Так, например, файл ИСР CoolManager (CoolDsgn**X**.exe) имеет размер немного превышающий 400 kb, а размер программы-просмотрщика CoolMan**X**.exe меньше 50 кб. Столь малый размер обусловлен тем, что все bpl пакеты, используемые приложением, загружаются динамически, после его запуска. При распространении приложения, скомпилированного без bpl пакетов, следует включать в состав дистрибутива не только exe файл приложения, но и все используемые bpl пакеты. При этом, возникают дополнительные сложности с отслеживанием всех пакетов используемых приложением, ведь загружаемый пакет, может использовать другие пакеты, которые так же будут загружены, те в свою очередь могут загружать что то ещё и т.д. Пакеты загружаемые при запуске конфигурации, так же могут ссылаться на другие пакеты, например, рассмотренный в предыдущем разделе пакет CoolIB**X**.bpl, использует пакет IBExpress, который автоматически загружается при загрузке пакета CoolIB**X**.bpl. Кроме множества пакетов, используемых приложением, существует проблема с поиском и включением в дистрибутив всех файлов, скриптов и форм конфигурации. Если все файлы находятся в одном каталоге, особенных проблем быть не должно, а если подключаемые скрипты и модули разбросаны по разным каталогам Вашего компьютера или локальной сети? Для решения проблем связанных с поиском всех модулей, необходимых для нормальной работы приложения, в состав ИСР включен конструктор дистрибутивов.

Для вызова конструктора дистрибутивов следует выбрать пункт главного меню **Вид**

\Конструктор дистрибутивов или нажать пару клавиш <**Alt-Z**>. Рассмотрим процесс создания дистрибутива на примере. Загрузите в ИСР конфигурацию \CONFIG\Demo9.cm4 и вызовите конструктор дистрибутивов:



На панели инструментов расположены следующие кнопки:

Новый - создать новый файл дистрибутива.

Загрузить - загрузить файл дистрибутива созданный ранее.

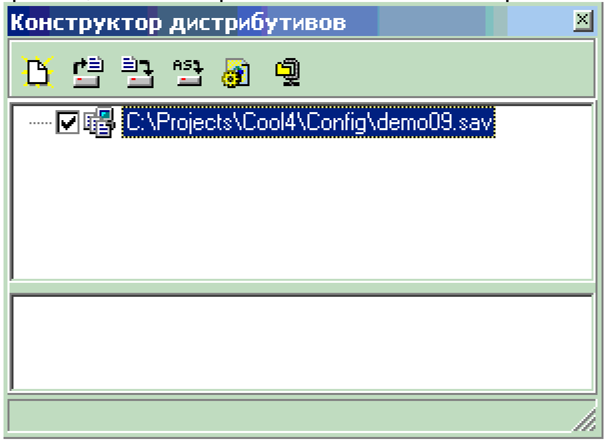
Сохранить - сохранить файл дистрибутива на диске.

Сохранить как - сохранить файл дистрибутива под другим именем.

Формирование дистрибутива - мастер для поиска Bpl пакетов и файлов конфигурации.

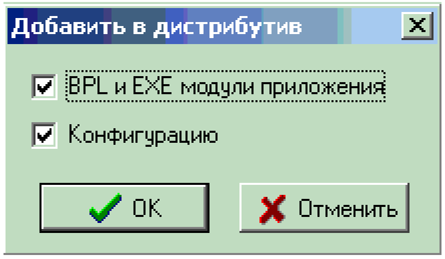
Создание дистрибутива - создание zip файла дистрибутива.

Создадим новый дистрибутив. Нажмите кнопку "**Новый**", появится диалог для ввода имени файла, назовем файл Demo09.sav и сохраним его в каталоге \CONFIG.



☑ C:\Projects\Cool4\Config\demo09.sav

Воспользуемся мастером для поиска пакетов и файлов конфигурации, нажмите кнопу , на экран будет выведено диалоговое окно:

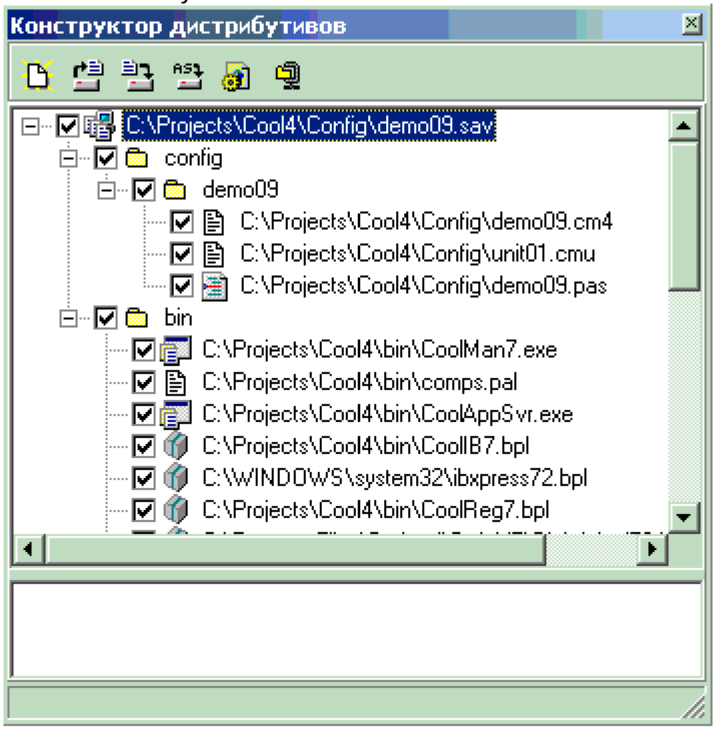

Добавить в дистрибутив

☑ BPL и EXE модули приложения

☑ Конфигурацию

- **BPL и EXE модули приложения** - включить в дистрибутив EXE файлы и BPL модули.
- **Конфигурацию** - добавить в дистрибутив модули и файлы конфигурации.

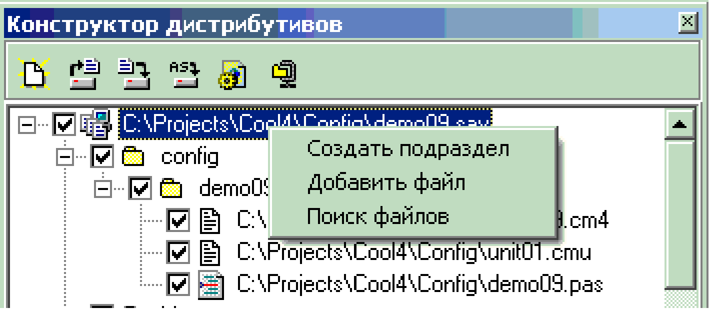
Нажмем кнопку ОК:


Конструктор дистрибутивов

```
☑ C:\Projects\Cool4\Config\demo09.sav
  ☑ config
    ☑ demo09
      ☑ C:\Projects\Cool4\Config\demo09.cm4
      ☑ C:\Projects\Cool4\Config\unit01.cmu
      ☑ C:\Projects\Cool4\Config\demo09.pas
  ☑ bin
    ☑ C:\Projects\Cool4\bin\CoolMan7.exe
    ☑ C:\Projects\Cool4\bin\comps.pal
    ☑ C:\Projects\Cool4\bin\CoolAppSvr.exe
    ☑ C:\Projects\Cool4\bin\CoolIB7.bpl
    ☑ C:\WINDOWS\system32\ibxpress72.bpl
    ☑ C:\Projects\Cool4\bin\CoolReg7.bpl
```

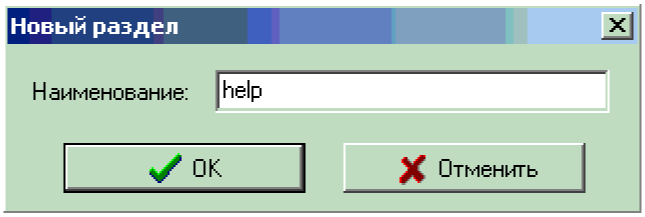
В конструкторе будет сформирована структура каталогов дистрибутива, в папке \BIN расположены исполняемые модули и пакеты, каталог \CONFIG\DEMO09 содержит файлы текущей конфигурации. Если снять галочку рядом с именем файла в конфигураторе, то файл в не будет включаться в дистрибутив при создании **zip** архива.

Добавим в дистрибутив файлы системы помощи. Установим полосу подсветки на корневой элемент списка каталогов и нажмем правую кнопку мыши:


Конструктор дистрибутивов

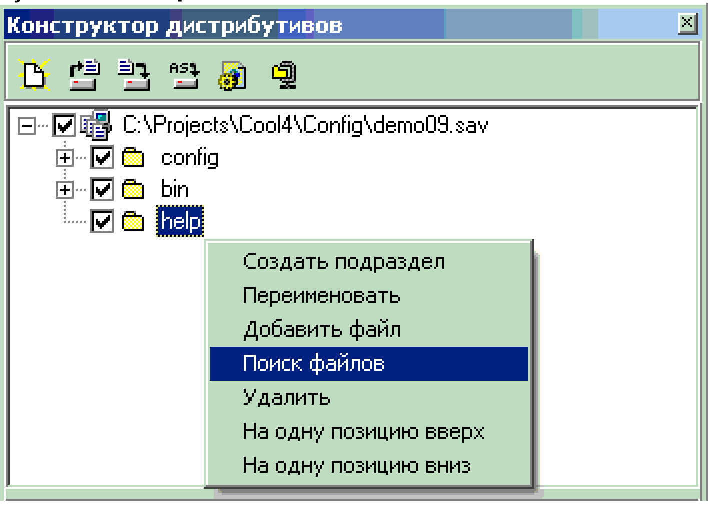
```
☑ C:\Projects\Cool4\Config\demo09.sav
  ☑ config
    ☑ demo09
      ☑ C:\Projects\Cool4\Config\demo09.cm4
      ☑ C:\Projects\Cool4\Config\unit01.cmu
      ☑ C:\Projects\Cool4\Config\demo09.pas
```

Выберите пункт "**Создать прдраздел**", в появившемся диалоговом окне введите имя раздела:



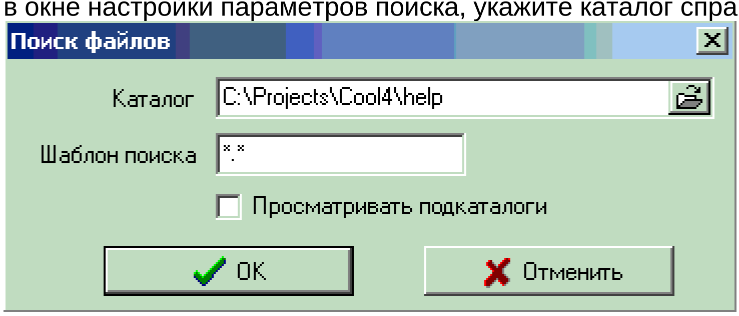
Наименование: help

будет создан новый каталог, щелкните по нему правой кнопкой мыши, в PopupMenu выберите пункт "**Поиск файлов**":



```
☑ C:\Projects\Cool4\Config\demo09.sav
    ☑ config
    ☑ bin
    ☑ help
```

в окне настройки параметров поиска, укажите каталог справочных файлов и шаблон поиска:

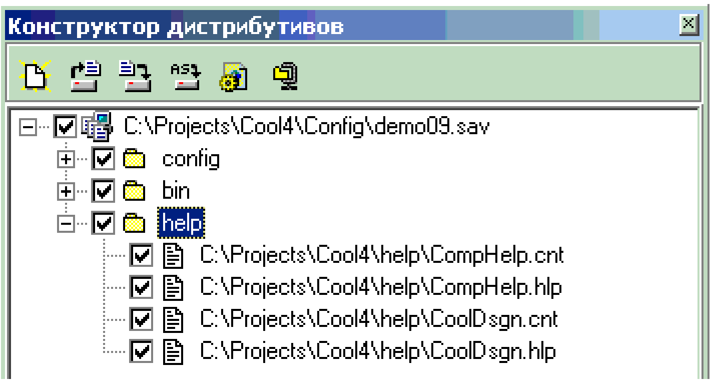


Каталог: C:\Projects\Cool4\help

Шаблон поиска: *.*

☐ Просматривать подкаталоги

Нажмите кнопку ОК. В папку help дистрибутива, будут добавлены все файлы из указанного каталога:



```
☑ C:\Projects\Cool4\Config\demo09.sav
    ☑ config
    ☑ bin
    ☑ help
        ☑ C:\Projects\Cool4\help\CompHelp.cnt
        ☑ C:\Projects\Cool4\help\CompHelp.hlp
        ☑ C:\Projects\Cool4\help\CoolDsgn.cnt
        ☑ C:\Projects\Cool4\help\CoolDsgn.hlp
```

Теперь можно сформировать zip архив. Нажмите кнопку , появится диалог для задания имени файла zip архива, сохраните дистрибутив как \CONFIG\demo08.zip.

Всё, дистрибутив собран и готов к использованию.

Глава IV

Резюме

4 Резюме

Не секрет, что Delphi пользуется огромной популярностью среди российских разработчиков прикладного ПО. CoolManager является инструментом созданным на основе Delphi для повышения производительности и комфортабельности работы разработчика бизнес-приложений. ИСР CoolManager проводит четкое разделение между Delphi программистом и разработчиком прикладных задач. Первый, используя всю мощь возможностей Delphi создает для CoolManager проблемно ориетнированные классы и инструменты в виде bpl пакетов. Второй, используя созданные Delphi программистом пакеты и мобильность ИСР CoolManager, с высокой производительностью пишет и отлаживает код прикладной задачи, который опять же, благодаря усилиям Delphi программиста, должен быть предельно прост и лаконичен.

"Дайте мне точку опоры, и я переверну Землю," - сказал Архимед. Сейчас трудно судить, что он действительно имел в виду. Но даже если воспринимать данное выражение как аллегорию, то в любом случае, для исполнения обещанного, Архимеду был необходим универсальный рычаг. Если под термином "Земля" подразумевать систему автоматизации, в качестве "точки опоры" использовать Delphi, Архимедом называть прикладного программиста, тогда ИСР CoolManager и есть тот универсальный рычаг, который позволит быстро и качественно реализовать задуманное.