Язык программирования

Описание

Уваров Александр Федорович
2011 г. Москва

Оглавление

Краткая характеристика языка *Alfa*

- Язык Alfa процедурный, типизированный;
- Программа на языке *Alfa* состоит из единственного модуля. Выполнение программы начинается с первого оператора после слова *begin*;
- Предусмотрены константы и переменные вещественного (*real*), целого (*int*), строкового (*string*), булевского (*bool*) типа. Ссылочные переменные файлового (*file*) типа, серверного (*server*) типа, COM (*comobj*) типа, процедурного (*proc*, *func*) типа;
- Предусмотрены одномерные массивы перечисленных типов;
- Переменные серверного типа (*server*) представляют собой специальный объект сервер, с помощью методов которого, можно удаленно выполнять различные действия на сервере или рабочей станции. Можно запускать и останавливать службы и процессы, определять их текущее состояние, перезагружать сервер, изменять *IP* адрес, определять доступность любого компьютера или сервера в сети, работать с файлами, папками, общими ресурсами;
- Предусмотрена работа с внешними серверами автоматизации (COM объектами);
- Предусмотрены стандартные процедуры и функции для работы с числами, строками, массивами, папками и файлами, а также функции для отправки сообщений электронной почты по протоколу *SMTP*, функции передачи и приёма файлов по протоколу *FTP*, функции для работы с *INI* файлами, функции для работы с COM объектами;
- Предусмотрен ввод и вывод на консоль, чтение и запись информации в текстовый файл;
- Предусмотрено выполнение кода написанного на других языках программирования *JScript*, *VBScript*;
- В записи программы разрешены комментарии, которые могут быть вложенными;
- Большие и малые буквы различаются;
- Язык программирования *Alfa* предназначен исключительно для специфических задач удаленного администрирования серверов и компьютеров и не претендует ни на что более.

Синтаксис языка *Alfa*

Для описания синтаксиса языка *Alfa* используются Расширенные Бэкуса-Наура формы (РБНФ). Ниже приведено описание языка в РБНФ.

```
Alfa =
  "program" Ident ";"
  Declarations
  "begin"
    Operations
    { FuncDeclare | ProcDeclare }
  "end" "." .

Declarations =
```

```
  { "const"
     { ConstDeclare ";" }
    |
    "var"
     { VarDeclare ";" }
  }
  {
    ServDeclare
  } .

ConstDeclare = ConstName "=" ConstExpr .

ConstExpr<out ObjVar Var> = [Sign] ( IntDef | RealDef ) | StringChar | StringScript |
                            true | false .

VarDeclare = [ "@" ] Ident ":" Type  [ "=" ( ConstExpr | ListParam ) ] .

ServDeclare =
  "declare" "@" ServName ":" "server"
    Ident "=" StringChar ";"
    Ident "=" StringChar ";"
    Ident "=" StringChar ";"
    Ident "=" StringChar ";"
    Ident "=" int ";"
    Ident "=" int ";"
  "end" ";" .

ProcDeclare =
  "procedure" Ident FormalParameters ";"
  { "const"
     { ConstDeclare ";" }
    |
    "var"
     { VarDeclare ";" }
  }
  "begin"
    Operations
  "end" ";" .

FuncDeclare =
  "function" Ident FormalParameters ":" Type ";"
  { "const"
      { ConstDeclare ";" }
    |
    "var"
      { VarDeclare ";" }
  }
  "begin"
    Operations
  "end" ";" .

FormalParameters = "(" [ FormalPar { ";" FormalPar } ] ")" | "()" .

FormalPar = [ "var" ] ( "@" ServName | Ident ) ":" Type .

Type = "real" | "int" | "bool" | "string" | "file" | "server" | "comobj" | "proc" |
       "func" | "array" [ Expr ] of Type .

Operations = [ Operation { Operation } ] .

Operation = Assign | If | Switch | For | While | Loop | ReadLn | Write |
            WriteLn | Exit .

Assign = Variable ( ":=" Expr | ListParam ) ";" .

Exit = exit ";" .
```

```
If =
  "if" Expr "then"
    Operations
  ["else"
     Operations]
  "endif" ";" .

Switch =
  "switch" Expr
    CaseSection
    { CaseSection }
    [ "default" ":" Operations]
  "endswitch" ";" .

CaseSection =
  "case" Expr ":"
    Operations
  "endcase" ";" .

For =
  "for" Ident ":=" Expr "to" Expr
    Operations
  "endfor" ";" .

While =
  "while" Expr
    Operations
  "endwhile" ";" .

Loop =
  "loop"
    Operations
  "endloop" ";" .

ReadLn = "readln" [ ListParam ] ";" .

WriteLn = "writeln" [ ListParam ] ";" .

Write = "write" [ ListParam ] ";" .

ListParam = "(" [ Param { "," Param } ] ")" | "()" .

Expr = SimpleExpr [ RelatOper SimpleExpr ] .

SimpleExpr = [ Sign ] Term { AddOper Term } .

Term = Factor { MulOper Factor } .

Factor = ConstUnSigned | Variable [ ListParam ] | "not" Factor | "(" Expr ")" .

RelatOper = "=" | "<>" | "<" | "<=" | ">" | ">=" .

AddOper = "+" | "-" | "or" | "xor" .

MulOper = "*" | "/" | "and" | "shr" | "shl" .

ConstUnSigned = RealDef | IntDef | StringChar | StringScript | true | false | nil.

Variable = [ "@" ] Ident [( "." Ident | "->" Ident | "[" Expr "]" )] .

Param = Expr .

IntDef = intCon .

RealDef = realCon .

ConstName = Ident .
```

```
ServName = IdentServ .

StringScript = markerbeg {ANY} markerend .

StringChar = stringLit .

Sign = "-" | "+" .

Ident = ident .

IdentServ = ident .

ident  = letter {letter | digit} .

intCon =
  (
    digit {digit} | digit {digit}
    |
    ("0x" | "0X") hexDigit {hexDigit}
  ) .

realCon =
  "." digit {digit} [("e" | "E") ["+" | "-"] digit {digit}]
  |
  digit {digit}
  (
    "." digit {digit} [("e" | "E" ) ["+" | "-"] digit {digit}]
    |
    ("e" | "E") ["+" | "-"] digit {digit}
  ) .

StringChar = stringLit .

stringLit   = '"' { stringCh | "\"\"" } '"' .

newLine = cr + eol .

stringCh = ANY - '"' - newLine .

letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_" .

digit = "0123456789" .

hexDigit = digit + "ABCDEFabcdef" .

markerbeg = "<!--" .

markerend = "-->" .
```

Словарь и представление

Для представления терминальных символов предусматривается использование набора знаков ASCII. Слова языка – это идентификаторы, числа, строки, операции и разделители. Должны соблюдаться следующие лексические правила. Пробелы и концы строк не должны встречаться внутри слов (исключая комментарии и пробелы в символьных строках). Пробелы и концы строк игнорируются, если они не существенны для отделения двух последовательных слов. Заглавные и строчные буквы считаются различными.

- *Идентификаторы* – последовательность букв и цифр. Первый символ должен быть буквой.

```
ident  = letter {letter | digit} .
```

Примеры: x Scan GetMessage firstLetter

- *Числа* – целые или вещественные (без знака) константы. Если константа начинается с символов “0x” или “0X” она является шестнадцатиричной, иначе – десятичной. Вещественное число всегда содержит десятичную точку. Оно может содержать десятичный порядок. Буква E означает «умножить на десять в степени». Вещественное число относится к типу *real*.

Примеры: 12 144 0x0D 12.3 4.765E5 .045e-6

- *Строки* – последовательности символов, заключенные в двойные (") кавычки. Открывающая кавычка должна быть такой же, как и закрывающая, если символ двойные (") кавычки входит в состав строки он должен быть удвоен. Число символов в строке называется ее длиной. Строка длины 1 может использоваться везде, где допускается символьная константа, и наоборот.

```
stringLit   = '"' { stringCh | "\"\"" } '"' .
```

Примеры: ”Просто строка символов” ”Don’t worry!” "Тестовая ""строка"""

Предусмотрено альтернативное представление любой последовательности символов “как есть” включая управляющие символы и символы кавычек.

```
StringScript = "<!--" {ANY} "-->" .
```

Такая последовательность начинается с символов `"<!--"` и заканчивается символами `"-->"`.

Пример:

```
<!--
Function Main()
   Dim fso, fldr
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set fldr = fso.CreateFolder("C:\MyTest1")
   Main = "Выполнено"
End Function
-->;
```

- *Операции* и *разделители* – это специальные символы, пары символов или зарезервированные слова, перечисленные ниже. Зарезервированные слова не могут использоваться как идентификаторы.

  *+ - * / . -> ( ) [] := < > <= >= <> and or not real int string bool nil file server comobj proc func array of begin end if else endif switch endswitch case endcase default for to endfor loop endloop while endwhile var const program procedure function declare readln write writeln exit result*

- *Комментарии* могут быть двух видов. Для комментария, который занимает не более одной строки, можно использовать символы //. Если же текст необходимо внести комментарий, расположенный на нескольких строках, то текст такого комментария нужно помещать внутри блока /* … */.

Типы данных

Alfa поддерживает типы данных, которые приведенные в следующей таблице.

Тип данных	Диапазон
int	-2147483648 .. 2147483647
real	-1.79769313486232e308 .. 1.79769313486232e308
string	Строка символов Юникода
bool	true или false
server	Объект
file	Объект
comobj	Объект
proc	Объект
func	Объект

Объявления

Каждый идентификатор, встречающийся в программе, должен быть обявлен. Объявления задают некоторые постоянные свойства идентификатора, например, является он константой, переменной, процедурой или функцией.

Переменные

Переменная – это именованная область памяти, предназначенная для хранения данных определенного типа. Во время выполнения программы значение переменной можно изменять. Все переменные, используемые в программе, должны быть описаны явным образом. При описании для каждой переменной задаются ее имя и тип. Переменные разделяютя по способу хранения данных на переменные значения и ссылочные переменные. Переменные значения хранят свои значения непосредственно, а ссылочные хранят не сами данные, а ссылку на них (адрес, по которому расположены данные). Сами данные хранятся в хипе. Переменные значения имеют тип: *int*, *real*, *bool*, *string*. Ссылочные переменные имеют тип: *server*, *file*, *comobj*, *proc*, *func*. Ссылочная переменная, которая, не ссылается на данные, имеет специальное значение *nil*. Любой ссылочной переменной можно присвоить значение *nil,* при этом объект, на который ссылается переменная - *удаляется*. Когда происходит присваивание одной ссылочной переменной, другой ссылочной переменной, *присваивается ссылка на эту переменную*, а не её значение.

Константы

Как и переменная, константа также является хранилищем для данных, но, в отличие от переменной, константа задается раз и навсегда и не допускает изменение своего содержимого.

Объявление констант

Объявление константы связывает ее идентификатор с ее значением.

```
ConstDeclare = ConstName "=" ConstExpr .
ConstExpr = [Sign] ( IntDef | RealDef ) | StringScript |
            StringChar | true | false .
```

Примеры объявление констант:

```
const
  E = 0xFFFF;
  N = 100;
  FlagOff = false;
  Str = "строка";
  F = 3.14;
  StrScript =
    <!--
      Function Main()
        Dim fso, fldr
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set fldr = fso.CreateFolder("C:\MyTest1")
        Main = "Выполнено"
      End Function
    -->;
```

Объявление переменных

Объявление переменных дают описание переменных, определяя идентификатор и тип данных для них. Для переменных вещественного *(real)*, целого *(int)*, строкового *(string)*, булевского *(bool),* процедурного *(proc, func)* , серверного *(server)*, COM (*comobj*) типов, при объявлении, можно указать инициализирующее значение. Тип инициализирующего значения должен совпадать с типом переменной. Если инициализирующее значение не указано, переменные инициализируются значениями по умолчанию: *real = 0.0, int = 0, string = "", bool = false, comobj = nil, file = nil, proc = nil, func = nil .*

```
VarDeclare = Ident ":" Type [ "=" ConstDeclare ].
```

Примеры объявление переменных:

```
var
  F : real = 3.14;
  N : int;
  FlagOff : bool = true;
  Str : string;
  TextFile : file;
  @Serv : server = ("Локальный", ".", "", "", 30, 180);
  FS : comobj = "Scripting.FileSystemObject";
  F1 : func = "Cos";
  P1 : proc;
```

Объявление и инициализация массивов

Массив – ограниченная совокупность однотипных величин. Элементы массива имеют одно и то же имя, а различаются порядковым номером (*индексом*). Элементами массива могут быть величины любого предопределенного типа данных. Количество элементов в массиве (*размерность*) не является частью его типа, это количество задается при объявлении массива и не может быть изменено впоследствии, кроме как присваиванием значения функции, которая возвращает массив того же типа или вызовом системной процедуры *ResizeArray().* Размерность *Expr* может задаваться не только константой, но и выражением. Результат вычисления этого выражения должен быть неотрицательным, а его тип должен быть *int*. Элементы массива нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности.

```
ArrayDeclare = Ident ":" "array" Expr "of" Type [ "=" ListParam ] .
```

Примеры объявление массивов:

```
var
  N : array 10 of int;
  FlagArr : array 9 of bool;
  StrArr : array 12 of string;
  RealArr : array 20 of real;
  FileArr : array 5 of file;
  ServArr : array 7 of server;
  COMArr : array 2 of comobj;
  ProcArr : array 9 of proc;
  FuncArr : array 11 of func;
```

Массив можно инициализировать, используя синтаксис объявления массива, за которым следует операция присваивания и разделенный запятыми список значений, заключенный в круглые скобки (количество значений внутри скобок должно совпадать с количеством элементов массива), массивы элементами которых являются *server* и *file* инициализировать значениями нельзя. Массив объектов сервер можно инициализировать, используя системную функцию *CreateServObj().*

Пример инициализации массивов:

```
var
  RealArray : array 4 of real = (1.2, 2.3, 5.7, 9.56);
  IntArray : array 5 of int = (10, 20, 30, 40, 50);
  BoolArray : array 3 of bool = (true, false, true);
  StrArray : array 4 of string = ("один", "два", "три", "четыре");
  COMArray : array 2 of comobj = ("Scripting.FileSystemObject", "WScript.Shell");
  FuncArray : array 2 of func = ("Sin", "Cos");
  ProcArray : array 2 of proc = ("Sin", "Cos");
  @ServArray : array 2 of server = (
    CreateServObj("Сервер приложений 1", "10.33.44.125", "adm1", "pass1", 60, 320),
    CreateServObj("Сервер приложений 2", "10.33.44.126", "adm2", "pass2", 60, 320)
  );
```

Для обращения к элементу массива после имени массива указывается номер элемента в квадратных скобках, например:

```
N[4]   FlagArray[i]
```

С элементами массива можно делать все, что допустимо для переменных того же типа. Массивы одного типа и одинаковой размерности можно присваивать друг другу. При этом происходит присваивание ссылок, а не элементов массива.

Примеры:

1. Присваивание возвращаемому результату функции (result) - массива значений.

```
function Test () : array 2 of string;
var
  A : array 4 of string = ("1", "2", "3", "4");
begin
  result := A; // Ошибка, размерность массива возвращаемого результата, не совпадает
```

```
                // с размерностью присваиваемого массива
end;

function Test () : array of string;
var
  A : array 4 of string = ("1", "2", "3", "4");
begin
  result := A; // Все нормально, возвращаемый результат - открытый массив
end;
```

2. Присваивание массивов.

```
program Test;
var
  Arr : array 1 of string = ("один");
  Arr1 : array 0 of string;

function Test () : array of string;
var
  A : array 4 of string = ("1", "2", "3", "4");
begin
  result := A;
end;

begin
  Arr := Test(); // Все нормально, функция возвращает массив, присваивание возможно
  Arr1 := Arr;   // Ошибка
end.
```

Объявление и инициализация серверных переменных

Объявление серверных переменных возможно двумя способами. Первый способ объявления, осуществляется путем декларации переменной сервера с указанием конкретных значений полей.

```
ServDeclare =
  "declare" "@" ServName ":" "server"
    "Title" "=" StringChar ";"
    "Host" "=" StringChar ";"
    "User" "=" StringChar ";"
    "Password" "=" StringChar ";"
    "IntervalWait" "=" number ";"
    "WaitReboot" "=" number ";"
  "end" ";" .
```

где

Title – заголовок сервера;

Host – IP адрес сервера (Host локального компьютера указывается символом точка. Host = “.”);

User – имя пользователя;

Password – пароль;

IntervalWait – интервал ожидания останова, запуска, службы или процесса;

WaitReboot – интервал ожидания готовности сервера после перезагрузки.

Примеры декларации серверов:

```
declare @CUMR04_TO : server
  Title = "Сервер телеобработки";
  Host = "10.23.248.15";
  User = "adm";
```

```
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

declare @CUMR04A : server
  Title = "Основной сервер приложений";
  Host = "10.23.248.11";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;
```

Второй способ, это объявление серверной переменной, как любой другой переменной, в секции *var*, с указанием типа переменной server. Серверная переменная представляет собой объект. Поля объекта сервера инициализируются значениями по умолчанию. *Title = “”, Host = “.”, User = “”, Password = ””, IntervalWait = 60, WaitReboot = 360.* Имя серверной переменной должно начинаться с символа “*@*”.

```
VarServ = "@" IdentServ ":" "server" [ "=" ListParam ] .
```

Серверную переменную можно инициализировать, используя синтаксис объявления переменной, за которой следует операция присваивания и разделенный запятыми список значений, заключенный в круглые скобки. Количество значений и тип значений внутри скобок должно совпадать с определением полей объекта сервер.

Пример объявления серверных переменных:

```
var
  @SP_Serv : server = ("Тестовый", "10.33.44.128", "adm", "pass", 30, 180);
  @DB_Serv : server;
```

В процессе выполнения программы серверную переменную можно инициализировать новыми значениями. Для этого нужно использовать системную функцию *CreateServObj()*.

```
function CreateServObj(Title: string; Host: string; User: string;
                       Password: string; IntervalWait: int; WaitReboot: int): server;
```

Системная функция *CreateServObj()* создает и возвращает инициализированный объект *server*.

Пример инициализации серверной переменной в процессе выполнения:

```
var
  @SP_Serv : server = ("Тестовый", "10.33.44.128", "adm", "pass", 30, 180);
  @DB_Serv : server;
  Flag : bool;

begin
  …
  @DB_Serv := CreateServObj("Сервер приложений", "10.33.44.125",
                            "adm1", "pass1", 60, 320);
  …
end;
```

При передаче серверной переменной в качестве параметра процедуры или функции, серверная переменная передается по ссылке.

Объявление и инициализация COM переменных

Для того чтобы получить доступ к свойствам или методам внешнего сервера автоматизации нужно определить переменную типа *comobj* и инициализировать переменную соответствующим объектом, т.е. загрузить в память экземпляр нужного COM-объекта и сохранить в переменной ссылку на этот объект. Объект может создаваться несколькими способами:

- объявить COM-переменную с инициализацией, используя синтаксис объявления переменной, за которой следует операция присваивания и в двойных кавычках значение *ProgID*;

  ```
  VarDeclare = Ident ":" "comobj"  "=" "ProgID" .
  ```

  Пример:

  ```
  FS : comobj = "Scripting.FileSystemObject";
  ```

- объявить COM переменную и инициализировать её с помощью системной функции *CreateComObj()*.

  ```
  function CreateComObj (ProgID: string): comobj;
  ```

  Системная функция *CreateComObj()* создает и возвращает объект *comobj*.

  Пример:

  ```
  var
    FS : comobj;
    D : comobj
    Flag : bool;
  begin
    …
    FS := CreateComObj("Scripting.FileSystemObject");
    …
  end;
  ```

В любом случае при явной инициализации или вызове системной процедуры используется программный идентификатор объекта (*ProgID*), заключенный в двойные кавычки. Перед точкой в *ProgID* стоит имя библиотеки типов (*type library*) для объекта, которая может существовать как в виде отдельного файла с расширением *tlb*, так и в виде части файла с исполняемым кодом объекта (библиотека типов, содержащая сведения о COM-объекте, регистрируется в системном реестре при установке приложения, использующего этот объект). После точки в *ProgID* указывается имя класса, содержащего свойства и методы, доступные для использования другими приложениями. Выполняя инициализацию или процедуру *CreateComObj()*, программа через *ProgID* получает из системного реестра путь к файлам нужной библиотеки типов. Затем с помощью этой библиотеки в память загружается экземпляр запрашиваемого объекта, и его интерфейсы становятся доступными для использования в программе. Ссылка на созданный объект сохраняется в переменной типа *comobj*; в дальнейшем, используя эту переменную, получаем доступ к свойствам и методам объекта, а также к его вложенным объектам (если они имеются).

Пример:

Обращение к методу объекта:

```
D := FS->GetDrive(C:);
```

Обращение к свойству объекта:

```
D.SerialNumber;
```

При передаче COM переменной в качестве параметра процедуры или функции, COM переменная передается по ссылке.

Выражения

Выражения – конструкции, задающие правила вычисления по значениям констант и текущим значениям переменных других значений путем применения операций и процедур-функций. Выражения состоят из операндов и операций. Круглые скобки могут использоваться для группировки операций и операндов.

Операции

В выражениях синтаксически разделяются четыре класса операций с разными приоритетами (*порядком выполнения*). Операция – имеет самый высокий приоритет, далее следуют операции типа умножения, операции типа сложения и отношения. Операции одного приоритета выполняются слева направо.

```
Expr = SimpleExpr [ RelatOper SimpleExpr ] .

SimpleExpr = [ Sign ] Term { AddOper Term } .

Term = Factor { MulOper Factor } .

Factor = ConstUnSigned | Variable [ ListParam ] | "not" Factor | "(" Expr ")" .

RelatOper = "=" | "<>" | "<" | "<=" | ">" | ">=" .

AddOper = "+" | "-" | "or" | "xor".

MulOper = "*" | "/" | "and" | "shl"| "shr".

ConstUnSigned = RealDef | IntDef | StringScript | StringChar |
                true | false | nil.
```

Логические операции

not	Отрицание
and	Логическая конъюнкция
or	Логическая дизъюнкция
xor	Исключающее ИЛИ

Результат *not x* равен *true*, если операнд *x* равен *false*. В противном случае, результат равен *false*.

Результат *x and y* равен *true*, если оба операнда *x* и *y* равны *true*. В противном случае, результат равен *false*.

Результат *x or y* равен *true*, если хотя бы один операнд *x* или *y* равен *true*. В противном случае, результат равен *false*.

Результат *x xor y* равен *true*, если *x* равен *true*, а *y* равен *false*, либо если *x* равен *false*, а *y* равен *true*. В противном случае, результат равен *false*.

Логические операции применимы к операндам типа bool и дают результат типа bool.

Побитовые операции

not	Дополнение до 1 (унитарный оператор НЕ)
and	Побитовое И
or	Побитовое ИЛИ
xor	Побитовое исключающее ИЛИ
shr	Побитовый сдвиг вправо
shl	Побитовый сдвиг влево

Побитовые операторы воздействуют на отдельные двоичные разряды (биты) своих операндов, они служат для проверки, установки или сдвига двоичных разрядов, состовляющих целое значение.

Побитовые операции работают с целочисленными значениями типа *int* и в качестве результата возвращают целочисленные значения типа *int*.

Пример:

```
var
  i : int;
  t : int;
  val : int;
begin
  // применение поразрядного оператора shl
  val := 1;
  for i := 0 to 7
    t := 128;
    while (t > 0)
      if (val and t) <> 0 then
        write("1 ");
      endif;
      if (val and t) = 0 then
        write("0 ");
      endif;
      t := t / 2;
    endwhile;
    writeln("");
    val := val shl 1;
  endfor;

// 0 0 0 0 0 0 0 1
// 0 0 0 0 0 0 1 0
// 0 0 0 0 0 1 0 0
// 0 0 0 0 1 0 0 0
// 0 0 0 1 0 0 0 0
// 0 0 1 0 0 0 0 0
```

```
// 0 1 0 0 0 0 0 0
// 1 0 0 0 0 0 0 0
```

Арифметические операции

+	Сумма
-	Разность
*	Произведение
/	Деление

Операции +, -, * и / применены к операндам числового типа *real* или *int*. При использовании в качестве одноместной операции ”-” обозначает перемену знака, а ”+” – тождественную операцию.

Отношения

=	Равно
<>	Не равно
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно

Отношения дают результат типа *bool*. Отношения =, <>, <, <=, >, и >= применимы к числовым типам *real* или *int*. Отношения =, <> применимы к типам *bool* и *string*.

Присваивание

Присваивание заменяет текущее значение переменной новым значением, определяемым выражением. Выражение должно быть совместимым по присваиванию с переменной.

```
Assign = Variable ( ":=" Expr  | ListParam ) ";" .
```

Примеры присваиваний:

```
I := 0;
p := I + r;
r := K[I];
f := Sum(a, b);
@Serv := @CUMR04A;
@Serv.Title := "Основной сервер приложений";
```

Последовательность операторов

Последовательность операторов, разделенных точкой с запятой, означает поочередное выполнение действий, заданных составляющими операторами.

Оператор IF

```
If =
  "if" Expr "then"
     Operations
  ["else"
     Operations]
  "endif" ";" .
```

Оператор *if* задает условное выполнение входящих в него последовательностей операторов. Перед выполнением проверяется логическое выражение *Expr*, если оно окажется равным *true*, выполняется последовательность операторов после слова *then* иначе после слова *else*, если оно имеется.

Пример:

```
if not(Error = "") then
  writeln("Ошибка: ", Error);
else
  writeln("Служба: " + "'" + NameService + "'" + " - запущена!");
endif;
```

Оператор SWITCH

```
Switch =
  "switch" Expr
    CaseSection
    { CaseSection }
    ["default" ":" Operations]
  "endswitch" ";" .

CaseSection =
  "case" Expr ":"
    Operations
  "endcase" ";" .
```

Оператор *switch* обеспечивает многовариантное ветвление. Оператор switch работает следующим образом. Значение выражения *switch Expr* последовательно сравнивается с выражениями *case Expr* заданного списка. При обнаружении совпадения для одного из условий сравнения выполняется последовательность операторов связанная с этим условием. Элемент выражения *Expr* оператора *switch* должен иметь тип *int* или *string*. *Case* выражение должно иметь тип совместимый с *switch* выражением. При этом никакие два *case*-выражения в одной *switch*-инструкции не могут иметь идентичных значений. Последовательность операторов *default*-ветви выполняется в том случае, если ни одно из заданных *case*-выражений не совпадает с результатом вычислений *switch*-выражения. Ветвь *default* необязательна. Если она отсутствует, то при несовпадении результата выражения ни с одним *case*-выражением никакое действие выполнено не будет.

Пример:

```
for I := 0 to 9
  switch I
    case 0:
      writeln("I равно нулю.");
    endcase;
    case 1:
      writeln("I равно единице.");
    endcase;
    case 2:
      writeln("I равно двум.");
    endcase;
    case 3:
      writeln("I равно трем.");
    endcase;
    case 4:
```

```
      writeln("I равно четырем.");
    endcase;
    default:
      writeln("I равно или больше пяти.");
  endswitch;
endfor;
```

Оператор WHILE

Оператор *while* задает повторное выполнение последовательности операторов, пока логическое выражение *Expr* (условие) остается равным *true*. Условие проверяется перед каждым выполнением последовательности операторов.

```
While =
  "while" Expr
     Operations
  "endwhile" ";" .
```

Пример:

```
while (i > 0)
  i := i / 2;
  k := k + 1;
endwhile;
```

Оператор FOR

Оператор *for* определяет повторное выполнение последовательности операторов фиксированное число раз для прогрессии значений целой переменной *Variable*, называемой управляющей переменной оператора *for*. Если начальное значение управляющей переменной меньше значения выражения *Expr* (стоящего после ключевого слова *to*) цикл вначале выполняет последовательность операторов, а затем увеличивает значение управляющей переменной на 1. Когда значение управляющей переменной становится равным значению конечного выражения, цикл *for* еще раз выполняет последовательность операторов, а затем завершается.

```
For =
  "for" Ident ":=" Expr "to" Expr
     Operations
  "endfor" ";" .
```

Пример:

```
for i := 0 to 0
  writeln("Hello");
endfor;
```

Оператор LOOP

Оператор *loop* определяет повторное выполнение последовательности операторов. Он завершается после выполнения оператора выхода *exit* внутри этой последовательности.

```
Loop =
  "loop"
     Operations
  "endloop" ";" .
```

Пример:

```
loop
  ReadInt(i);
  if (i < 0) then
    exit;
  endif;
  WriteInt(i);
endloop;
```

Оператор READLN

Определены две формы представления оператора, первая позволяет считывать данные вводимые с клавиатуры, а вторая считывать строковые данные из текстового файла, связанного с файловой переменной *F*.

(1) readln([P1][,P2]);
(2) readln(F:file, P1);

Аргументы:

- *F* – переменная файлового типа, характеризующая текстовый файл;
- *P1, P2* – параметры ввода, представляющий собой перечисленные через запятую выражения или переменные.

Первая форма ввода (ввод данных с клавиатуры).

readln();

На экран консоли выводится строка сообщения “*Press any key …*” и ожидается нажатие любой клавиши.

readln(P1);

На экран выводится символ «?» и считываются данные в соответствующую переменную, определенную параметром *P1*. При считывании строки, содержащей число, оператор *readln* пропускает любые пробелы, символы табуляции, предшествующие числовой строке. После этого числовая строка конвертируется в число и назначается соответствующей переменной, определенной параметром *P1*. Если, во время конвертации данных произошла ошибка, на консоль выдается сообщение "*Error, invalid number*!" и ввод данных повторяется.

readln(P1, P2);

На экран выводится строка определенная параметром *P1*, и считываются данные в соответствующую переменную, определенную параметром *P2*. Параметр *P1* может быть символьной переменной или строковым выражением. Если при вводе данных произошла ошибка, на консоль выдается сообщение об ошибке и ввод повторяется.

Вторая форма ввода (ввод данных из текстового файла)

readln(F:file, P1);

Позволяет считать строку из текстового файла в символьную переменную определенную параметром *P1*, выполняя перевод указателя в начало следующей строки после каждого признака конца строки. Если при вводе данных из текстового файла произошла ошибка, на консоль выдается сообщение об ошибке и программа завершает свою работу.

Оператор WRITE

Определены две формы представления оператора, первая позволяет выводить данные на консоль, а вторая выводить данные в текстовый файл, связанный с файловой переменной *F*.

write([var F: file] [P1, P2, …, Pn]);

Аргументы:

- *F* – переменная файлового типа, характеризующая текстовый файл;
- *P1* … *Pn* – список вывода, представляющий собой перечисленные через запятую выражения или переменные, которые содержат значения передаваемые в файл или на консоль.

Оператор WRITELN

Определены две формы представления оператора, первая позволяет выводить данные на консоль, а вторая выводить данные в текстовый файл, связанный с файловой переменной *F*, заканчивающиеся признаком конца строки.

writeln([var F: file] [P1, P2, …, Pn]);

Аргументы:

- *F* – переменная файлового типа, характеризующая текстовый файл;
- *P1* … *Pn* – список вывода, представляющий собой перечисленные через запятую выражения или переменные, которые содержат значения передаваемые в файл или на консоль.

Оператор EXIT

Оператор *exit* предназначен для прерывания циклов, выхода из тела процедур и функций, а также из тела программы (завершение программы).

Объявления процедур и функций

Объявление процедуры состоит из заголовка процедуры и тела процедуры. Заголовок определяет имя процедуры и формальные параметры. Тело содержит объявления и операторы между ключевыми словами *begin* и *end*.

```
ProcDeclare =
  procedure Ident FormalParameters ";"
  { const
    { ConstDeclare ";" }
  |
    var
    {  VarDeclare ";" }
  }
  begin
    Operations
  end ";" .
```

Пример:

```
procedure Hello();
begin
  writeln("Hello");
end;
```

Имеются два вида процедур: собственно процедуры и процедуры-функции. Главное отличие между процедурами и функциями состоит в том, что функции имеют возвращаемое значение. При вызове функции она выполняется и возвращает значение вызывающему приложению.

```
FuncDeclare =
  function Ident FormalParameters ":" Type ";"
  { const
    { ConstDeclare ";" }
  |
    var
    {  VarDeclare ";" }
  }
  begin
    Operations
  end ";" .
```

При создании функций необходимо определить, по меньшей мере, два элемента: имя функции и тип данных результата. Например, следующая функция возвращает строковое значение:

```
function ReturnString() : string;
begin
  …
end;
```

Заголовок функции указывает *Alfa*, что функция возвращает строку, но в действительности она может не возвращать никаких значений. Чтобы возвратить значение из функции, это значение нужно присвоить специальной переменной. Переменная *result* – специальная переменная, которая неявно создается в каждой функции. Таким образом, чтобы функция возвращала строку “*Alfa*” потребуется записать следующее:

```
function ReturnString() : string;
begin
  result := "Alfa";
end;
```

Формальные параметры процедур и функций

Формальные параметры – идентификаторы, объявленные в списке формальных параметров процедуры или функции. Им соответствуют фактические параметры, которые задаются при вызове процедуры или функции. Подстановка фактических параметров вместо формальных происходит при вызове процедуры или функции. Имеются два вида параметров: параметры-значения и параметры-переменные, обозначаемые в списке формальных параметров отсутствием или наличием ключевого слова *var*. Параметры-значения - это локальные переменные, которым в качестве начального присваивается значение соответствующего фактического параметра. Параметры-переменные соответствуют фактическим параметрам, которые являются переменными, и означают эти переменные. Параметры-значения передают по значению, а параметры-переменные – по ссылке. Различие между этими способами передачи состоит в том,

что параметры-значения получают копии значений, а параметры-переменные – ссылку на переменные. При использовании параметров-переменных программа работает непосредственно с исходными значениями, а не с их копиями.

Область действия формального параметра простирается от его объявления до конца процедуры или функции, в котором он объявлен. Функция без параметров должна иметь пустой список параметров. Она должна вызываться обозначением функции, чей список фактических параметров также пуст.

```
FormalParameters = "(" [ FormalPar { ";" FormalPar } ] ")" | "()" .

FormalPar = [ var ] Ident ":" Type .
```

Параметры массивов и открытых массивов

В функции и процедуры можно передавать в качестве параметров массивы двумя способами. Первый способ передача параметра массива с указанием размерности массива.

```
procedure NameProc (NameArray: array Len of Type)
```

Пример:

```
procedure OutStr (ArrStr: array 10 of string);
begin
  …
end;
```

Процедура OutStr может принимать в качестве параметра, строковый массив, состоящий только из 10 строк. Если нужно работать с массивами других размеров, необходимо определить параметр типа открытого массива.

Второй способ передачи параметра, как параметр открытого массива (т.е. без указания размерности).

```
procedure NameProc (NameArray: array of Type);
```

Пример:

```
procedure OutStr (ArrStr: array of string);
begin
  …
end;
```

Параметры открытого массива позволяют передавать процедуре или функции массивы различных размеров. Для определения действительного размера массива нужно использовать стандартную функцию *Len()*.

Пример:

```
procedure OutStr (ArrStr: array of string);
var
  I : int;
begin
  for I := 0 to Len(ArrStr) - 1
    …
```

```
  endfor;
  …
end;
```

Процедурные типы. Параметры – функции и параметры – процедуры.

Определены два процедурных типа: тип - процедура (*proc*) и тип – функция (*func*). Основное назначение процедурных типов - это передача функций и процедур в качестве фактических параметров другим процедурам и функциям.

Для того чтобы воспользоваться процедурным типом, небходимо создать процедурную переменную и присвоить в качестве значения имя соответствующей процедуры (функции). После такого присваивания имя переменной становится синонимом имени соответствующей процедуры (фунции). В качестве значения допускается присваивание имен стандартных процедур и функций. Переменные процедурного типа могут быть параметрами значениями и параметрами переменными в пользовательских процедурах и функциях.

Пример:

```
var
  // объявление процедурной переменной F, указывающей на функцию B()
  F : func = "B";
  // объявление процедурной переменной P, указывающей на процедуру A()
  P : proc = "A";

// процедура A()
procedure A();
begin
  …
end;

// функция B()
function B() : bool;
begin
  …
end;

// процедура C() - в качестве формального параметра передается параметр-функция Func
procedure C(Func : func);
var
  Flag : bool;
begin
  // вызов функции
  Flag := Func();
  …
end;

// процедура D() - в качестве формального параметра передается параметр-процедура Proc
function D(var Proc : proc): bool;
begin
  // вызов процедуры
  Proc();
  …
end;

begin
  C(F);
```

```
  D(P);
end.
```

Процедурные переменные можно сравнивать на равество и неравенство. Процедурные переменные равны, если они одного типа и ссылаются на одно и то же имя процедуры (функции). Процедурные переменные по присваиванию совместимы со строковым типом *string*. Процедурные переменные применяются для создания универсальных процедур и функций и обеспечения механизма обратного вызова. В качестве простейшего примера универсальной процедуры приведена процедура вывода таблицы значений функции, в процедуру передается диапазон значений аргумента, шаг его изменений и вид вычисляемой функции. Этот пример приведен далее.

```
program CalcFunc;
var
  F : func;

procedure Table(F : func; X : real; B : real);
begin
  writeln(" ----- X ----- Y -----");
  while (X <= B)
    writeln(Format("| {0,8:0.000} | {1,8:0.000} |", X, F(X)));
    X := X + 1.0;
  endwhile;
  writeln("----------------------");
end;

begin
  writeln(" Таблица функции Sin");
  F := "Sin";
  Table(F, -2.0, 2.0);
  writeln(" Таблица функции Cos");
  F := "Cos";
  Table(F, -2.0, 2.0);
  readln();
end.

// Таблица функции Sin
//  ----- X ----- Y -----
// |   -2,000 |   -0,909 |
// |   -1,000 |   -0,841 |
// |    0,000 |    0,000 |
// |    1,000 |    0,841 |
// |    2,000 |    0,909 |
// ----------------------
// Таблица функции Cos
//  ----- X ----- Y -----
// |   -2,000 |   -0,416 |
// |   -1,000 |    0,540 |
// |    0,000 |    1,000 |
// |    1,000 |    0,540 |
// |    2,000 |   -0,416 |
// ----------------------
```

Область видимости констант и переменных

Константы, объявленные за пределами функций и процедур, являются глобальными, общедоступными константами, они видны в теле пользовательских процедур и функций. Переменные в отличие от констант, всегда локальны. Переменные, объявленные в программе, не видны в теле пользовательских процедур и функций и наоборот, переменные, объявленные в пользовательских процедурах и функциях, не видны в теле программы.

Работа с COM-объектами

Для работы с COM-объектами служат переменные типа *comobj*. Для получения доступа к объекту сервера автоматизации небходимо:

- объявить и инициализировать COM-переменную для получения ссылки на объект сервера автоматизации;
- вызвать необходимые методы или свойства объекта.

Пример:

```
program TestCom;
var
 // создаем объект FileSystemObject
 FS : comobj = "Scripting.FileSystemObject";
 DR : comobj;
 Space : real;
begin
  // получаем объект Drive диска C:
  DR := FS->GetDrive("C:");
  // определяем свободное место на диске C:
  Space := DR.AvailableSpace;
  // выводим на консоль
  writeln("Свободное место на диске C: ",
          Format("{0:##.#}", Space/Pow(1024.0, 3.0)), "Гб");
  // запрос на завершение работы
  readln();
end.
```

Пример работы с WMI:

```
program TestWMI;
var
  Locator : comobj = "WbemScripting.SWbemLocator";
  Service : comobj;

function Connect (Locator : comobj) : comobj;
const
  wbemPrivilegeCreatePermanent = 15;
var
  Service : comobj;
  Privileges : comobj;
  LastError : comobj;
  Prop : comobj;
  Item : comobj;
begin
  Service := Locator->ConnectServer(".", "root\CIMV2");
  Privileges := Service.Security_;
  Privileges := Privileges.Privileges;
  Privileges := Privileges->Add(wbemPrivilegeCreatePermanent, true);
  result := Service;
end;

function RunProcess(Service : comobj; CommandLine : string) : int;
const
  SW_SHOW = 1;
```

```
var
  Startup : comobj;
  Config : comobj;
  Process : comobj;
  ProcessID : int;
  nResult : int;
begin
  Startup := Service->Get("Win32_ProcessStartup");
  Config := Startup->SpawnInstance_();
  Config.ShowWindow := 1;
  // получаем об'ект Win32_Process
  Process := Service->Get("Win32_Process");
  // запускаем процесс
  nResult := Process->Create(CommandLine, nil, Config, ProcessID);
  result := nResult;
end;

begin
  // осуществляем соединение с локальным компьтером
  Service := Connect(Locator);
  // запускаем блокнот
  RunProcess(Service, "notepad.exe");
  // запрос на завершение работы
  readln();
end.
```

Во многих серверах автоматизации широко используются коллекции объектов. Для получения доступа к коллекции объектов сервера автоматизации необходимо:

- объявить и инициализировать COM-переменную для получения ссылки на объект сервера автоматизации;
- получить объект коллекции;
- создать объект перечислителя коллекции с помощью системной функции *Enumerator()*;
- в цикле, используя объект перечислителя с помощью системной функции *MoveNext()*, получить элементы коллекции.

Пример:

```
program TestCom1;
var
 // создаем объект FileSystemObject
 FS : comobj = "Scripting.FileSystemObject";
 // объект перечислителя
 Enum : comobj;
 // объект элемента коллекции
 D : comobj;
begin
  writeln("Список устройств");
  // создаем объект перечислителя коллекции
  Enum := Enumerator(FS.Drives);
  // получаем элемент коллекции
  while(MoveNext(Enum, D))
    // выводим значение элемента коллекции
```

```
    writeln("  Устройство: ", D.DriveLetter);
  endwhile;
  // запрос на завершение работы
  readln();
end.
```

Пример работы с WMI:

```
program TestWMILogfile;
var
  sBegDate : string;
  sEndDate : string;
  oService : comobj;

function Connect() : comobj;
const
  wbemPrivilegeCreatePermanent = 15;
var
  oLocator : comobj = "WbemScripting.SWbemLocator";
  oService : comobj;
  oPrivileges : comobj;
begin
  oService := oLocator->ConnectServer(".", "root\CIMV2");
  oPrivileges := oService.Security_;
  oPrivileges := oPrivileges.Privileges;
  oPrivileges := oPrivileges->Add(wbemPrivilegeCreatePermanent, true);
  result := oService;
end;

procedure EventLog(oService : comobj; sBegDate : string; sEndDate : string);
const
  wbemFlagReturnImmediately = 16;
var
  Str : string;
  oEnum : comobj;
  oElem : comobj;
  oBDate : comobj = "WbemScripting.SWbemDateTime";
  oEDate : comobj = "WbemScripting.SWbemDateTime";
begin
  oBDate->SetVarDate(DateTimeToOLE(sBegDate), true);
  oEDate->SetVarDate(DateTimeToOLE(sEndDate), true);
  Str := "Select * from Win32_NTLogEvent Where TimeWritten >= '" +
         oBDate.Value + "' AND TimeWritten <= '" + oEDate.Value + "'" +
         " AND Logfile = 'Application'" + " AND EventType = 2";
  writeln(Str);
  oEnum := Enumerator(
    oService->ExecQuery(Str, "WQL", wbemFlagReturnImmediately));
  while (MoveNext(oEnum, oElem))
    writeln("-----------------------------------------------------------------");
    writeln("Дата:          ", WMIToDateTime(oElem.TimeWritten));
    writeln("Сервер:        ", oElem.ComputerName);
    writeln("Журнал:        ", oElem.Logfile);
    writeln("Источник:      ", oElem.SourceName);
    writeln("Тип:           ", oElem.Type);
    if not IsNull(oElem.User) then
      writeln("Пользователь: ", oElem.User);
    else
      writeln("Пользователь: ", "Н/Д");
```

```
    endif;
    if not IsNull(oElem.Message) then
      writeln("Описание:     ", oElem.Message);
    else
      writeln("Описание:     ", "Нет");
    endif;
  endwhile;
end;

begin
  sBegDate := GetDate(Date() + " " + Time());
  sEndDate := Date() + " " + Time();
  oService := Connect();
  EventLog(oService, sBegDate, sEndDate);
  readln();
end.
```

Работа с текстовыми файлами

Для получения доступа к текстовым файлам служат переменные типа *file*.

```
var
  myFile: file;
```

При передаче переменной файлового типа в качестве параметра процедуры или функции, файловая переменная передается по ссылке.

Прежде чем можно будет приступить к работе с файлом, с помощью процедуры *OpenFile* его нужно открыть для записи или для чтения.

Запись в текстовый файл

Чтобы подготовить файл к записи, необходимо использовать процедуру *OpenFile*, которая в зависимости от параметра *Append* перезаписывает существующий файл или добавляет данные в существующий файл и наконец, если файл не существует, создается новый файл.

```
OpenFile(myFile, "c:\dataprog\data.txt", true, false);
```

Когда файл открыт и готов к записи, для записи текста в текстовый файл нужно использовать оператор *writeln* или *write*. При выполнении записи в текстовый файл первым параметром, переданным *writeln* или *write*, должна быть переменная файла.

```
writeln(myFile, "cave canem");
```

По завершению работы с файлом его всегда следует закрывать, чтобы обеспечить корректное сохранение файла на диске и освободить любую память, занятую в процессе записи. Для закрытия файла служит процедура *CloseFile*, принимающая единственный параметр – файл, который нужно закрыть.

```
CloseFile(myFile);
```

Чтение из текстового файла

Чтобы подготовить файл к чтению, необходимо использовать процедуру *OpenFile*, которая открывает существующий файл для чтения.

```
OpenFile(myFile, "c:\dataprog\data.txt", false, false);
```

Когда файл открыт и готов к чтению, для чтения данных из текстового файла нужно использовать оператор *readln.* При выполнении чтения данных из текстового файла первым параметром, переданным *readln*, должна быть переменная файла, а вторым – строковая переменная, которая будет временно хранить значение, считанное из файла.

```
readln(myFile, line);
```

По завершению работы с файлом его нужно закрыть.

```
CloseFile(myFile);
```

Код завершения программы

Чтобы возвратить значение из программы, это значение нужно присвоить специальной переменной. Переменная *result* – специальная переменная, которая неявно создается в программе и имеет тип *int*. Программа возвращает значение 0, в случае успешного выполнения. В случае любой ошибки периода выполнения программа возвращает значение равное 1 и завершает свою работу.
Для уведомления системы об ошибке во время выполнения можно возвращать ненулевые пользовательские значения.

Объект сервер

После определения переменных серверного типа или декларации серверов, создаются объекты серверов. Каждый объект сервер имеет предопределенный набор свойств и методов. Свойства объекта сервер доступны для чтения и записи.

Обращение к свойству объекта:

```
@ServName.Title;
```

Обращение к методу объекта:

```
@ServName->Connect();
```

Свойства объекта сервер

Title: string
Описание хоста;

Host: string
Имя хоста (IP адрес сервера);

User: string
Имя пользователя.

Password: string
Пароль.

IntervalWait: int
Интервал ожидания останова, запуска службы или процесса.

WaitReboot: int

Интервал ожидания готовности сервера после перезагрузки.

Методы объекта сервер

Методы для соединения с сервером

function Connect (): bool;

Осуществляет соединение с сервером.

Возвращаемый результат:

true – соединение прошло успешно или *false*, если произошла ошибка.

Примечание:

Метод Connect обязан быть вызван (один раз), перед вызовом других методов.

Пример:

```
writeln("Соединение с сервером: ", @CUMR02APL.Host);
if not @CUMR02APL->Connect() then
  writeln("Ошибка: ", @CUMR02APL->Get_MessErr());
  exit;
endif;
```

function Connect1 (Host: string): bool;

Осуществляет соединение с сервером.

Параметры:

Host – имя хоста (имя компьютера или IP адрес);

Возвращаемый результат:

true – соединение прошло успешно или *false*, если произошла ошибка.

Примечание:

Метод *Connect1* обязан быть вызван (один раз), перед вызовом других методов.

Пример:

```
Flag := @Serv->Connect1("10.45.168.234");
Flag := @Serv->Connect1("Test_Serv");
```

Методы для определения доступности компьютера

function PingStatus (Host: string): bool;

Определение доступности компьютера по его имени или IP адресу.

Параметры:

Host – имя хоста (имя компьютера или IP адрес);

Возвращаемый результат:

true – удаленный компьютер доступен или *false*, удаленный компьютер не доступен.

Пример:

```
Flag := @Serv->PingStatus("10.45.168.234");
Flag := @Serv->PingStatus("Test_Serv");
```

function WaitForEnableComputer (Host: string; WaitTime: int): bool;

Ожидание доступности компьютера по его имени или IP адресу.

Параметры:

Host – имя хоста (имя компьютера или IP адрес);
WaitTime - время ожидания (в секундах).

Возвращаемый результат:

true – удаленный компьютер доступен или *false*, удаленный компьютер не доступен.

Пример:

```
Flag := @Serv->WaitForEnableComputer("10.45.168.234", 60);
Flag := @Serv->WaitForEnableComputer("Test_Serv", 60);
```

Методы для работы с IP адресами

function SearchNetworkAdapter (IP: string): bool;

Проверка наличия заданного IP адреса (адресов)

Параметры:

IP – IP адрес (адреса);

Возвращаемый результат:

true – заданный IP адрес (адреса) найден или *false*, когда не найден .

Пример:

```
Flag := @Serv->SearchNetworkAdapter(“10.46.167.204”);
```

function ReplaceIP (OldIP: string; NewIP: string; SubNetMask: string): string;

Изменение IP адреса

Параметры:

OldIP - старый IP адрес;
NewIP - новый IP адрес;
SubNetMask - маска подсети.

Возвращаемый результат:

“*OK*” - успешное завершение, перезагрузка не требуется;
“*Reboot*” - успешное завершение, требуется перезагрузка;
“*Error*” - ошибка.

Пример:

```
Status := @Serv->ReplaceIP(IP, NewIP, SubNetMask);
switch Status
  case "Reboot":
    writeln("Перезагрузка сервера: " + @Serv.Title);
    if not @Serv->Reboot() then
      writeln("Ошибка при перезагрузке сервера: " + @Serv.Title);
      result := false;
    endif;
  endcase;
  case "Error":
    writeln("Ошибка при замене IP на сервере: " + @Serv.Title + "!");
    writeln("Требуется срочное вмешательство и замена вручную!");
    result := false;
```

```
  endcase;
endswitch;
```

Методы для работы с процессами

function RunProcess (CommandLine: string; CurrentDirectory: string): bool;

Запуск процесса.

Параметры:

CommandLine – командная строка запуска процесса;

CurrentDirectory - текущий каталог.

Возвращаемый результат:

true – запуск процесса прошел успешно или *false*, ошибка при запуске процесса.

Пример:

```
// копирование файла настроек сервера подсистемы КСАРМ ВЧД
Flag := @Res->RunProcess("c:\cittrans\tov\osn\scl_05a_Reserve_to_Main.bat", "");
```

function TerminateProcess (NameProcess: string): bool;

Завершение процесса.

Параметры:

NameProcess – имя процесса.

Возвращаемый результат:

true – завершение процесса прошло успешно или *false*, ошибка при завершении процесса.

Пример:

```
Flag := @Main->TerminateProcess("accounts.exe");
```

function TerminateProcess1 (IdProcess: int): bool;

Завершение процесса.

Параметры:

IdProcess – ID процесса.

Возвращаемый результат:

true – завершение процесса прошло успешно или *false*, ошибка при завершении процесса.

Пример:

```
Flag := @Main->TerminateProcess(445);
```

function ProcessIsRunning (NameProcess: string): bool;

Определение, выполняется ли процесс.

Параметры:

NameProcess – имя процесса (имя выполняемого файла с расширением).

Возвращаемый результат:

true – процесс выполняется или *false*, процесс не выполняется.

Пример:

```
Flag := @Main->ProcessIsRunning("accounts.exe");
```

function WaitForProcess (NameProcess: string; WaitTime: int): bool;

Ожидание завершения процесса.

Параметры:

NameProcess – имя процесса (имя выполняемого файла с расширением).

WaitTime - время ожидания (в секундах).

Возвращаемый результат:

true – процесс завершен или *false*, процесс не завершен.

Примечание:

По истечению времени ожидания заданный процесс принудительно завершается. Интервал времени определяется свойством *IntervalWait*.

Пример:

```
Flag := @Main->WaitForProcess("accounts.exe", 30);
```

function WaitRunningProcess (NameProcess: string; WaitTime: int): bool;

Ожидание запуска процесса

Параметры:

NameProcess – имя процесса (имя выполняемого файла с расширением).

WaitTime - время ожидания (в секундах).

Возвращаемый результат:

true – процесс запущен или false, процесс не запущен.

Пример:

```
Flag := @Res->WaitRunningProcess("accounts.exe", 30);
```

function GetListProcess (NameProcess: string; Delim: string; FullForm: bool): array of string;

Получить список информации о запущенных процессах по заданным условиям.

Параметры:

NameProcess – имя процесса (имя выполняемого файла с расширением), можно задавать начальную часть имени, в этом случае будет выдаваться информация по всем запущенным процессам при совпадении части имени. Если NameProcess = “” то выдается информация по всем запущенным процессам;

Delim – строка с символом разделителем полей данных*;*

FullForm – признак выдачи результатов, если true - полная форма, иначе краткая.

Возвращаемый результат:

Строковый массив с результатом запроса. Если процесс по заданному условию, не найден, то возвращается массив нулевой длины. В строке массива представлена информация о процессе. Формат строки массива представления информации о запущенном процессе зависит от признака выдачи результатов.

Полная форма (*FullForm* = *true*)). Формат элемента массива:

<*Name*><Delim><*CreationDate*><Delim><*ProcessId*><Delim><*ParentProcessId*><Delim>
<*CommandLine*><Delim><*ExecutablePath*><Delim><*Priority*><Delim><*HandleCount*><Delim>
<*WorkigSetSize*><Delim><*PeakWorkingSetSize*><Delim><*PageFaults*><Delim>
<*PageFileUsage*><Delim><*PeakPageFileUsage*><Delim><*PeakPageFaults*><Delim>
<*VirtualSize*><Delim><*PeakVirtualSize*><Delim><*ReadTransferCount*><Delim>
<*WriteTransferCount*><Delim><*OtherTransferCount*>

Краткая форма (*FullForm* = *false*)). Формат элемента массива:

<*Name*><Delim><*ProcessId*>

где:

Delim – строка символов разделителей полей;

Name – имя запущенного процесса;

CreationDate – дата и время создания процесса в формате "`dd.MM.yyyy HH:mm:ss`";

ProcessId - числовой идентификатор, используемый для обозначения процесса во время выполнения;

ParentProcessId – числовой идентификатор процесса создавший данный процесс;

CommandLine – командная строка запуска процесса;

ExecutablePath – выполняемый путь процесса;

Priority - ранг, определяющий порядок, в котором потоки процесса обрабатываются процессором.

HandleCount – общее количество дескрипторов, открытых в настоящее время процессом (равно общему количеству дескрипторов, открытых каждым потоком в процессе);

WorkigSetSize - текущий рабочий набор страниц памяти, занятых процессом (в Кбайт). Текущий рабочий набор — это число страниц, резидентных в настоящий момент в памяти;

PeakWorkingSetSize - объем физической памяти, используемой процессом (в Кбайт) с момента его запуска;

PageFaults - прерывание, происходящее при попытке программы прочитать или записать данные в область виртуальной памяти, имеющую пометку отсутствует. Число обращений к диску для загрузки данных, не найденных в ОЗУ. Это значение суммируется с момента запуска процесса;

PageFileUsage - скрытый файл на жестком диске, используемый Windows для хранения частей программ и файлов данных, не помещающихся в оперативной памяти. Файл подкачки и физическая (оперативная) память составляют виртуальную память. По мере необходимости Windows перемещает данные из файла подкачки в оперативную память (для их использования программой) и обратно (для освобождения места для новых

данных). Файл подкачки называется также файлом виртуальной памяти. Обьем (в Кбайт) использования файла подкачки данным процессом;

PeakPageFileUsage - максимальный объем (в Кбайт) использования файла подкачки процессом с момента его запуска;

VirtualSize - объем виртуальной памяти (в Кбайт адресного пространства), переданной процессу;

PeakVirtualSize - максимальный объем виртуальной памяти (в Кбайт адресного пространства), переданной процессу с момента его запуска;

ReadTransferCount - это число байтов, прочитанных в ходе операций ввода/вывода, созданных процессом, включая ввод/вывод в файлах, в сети и в устройствах. Прочитанные байты, направленные в дескрипторы CONSOLE (объекты ввода в консоль), не учитываются;

WriteTransferCount - это число байтов, записанных в ходе операций ввода/вывода, созданных процессом, включая ввод/вывод в файлах, в сети и в устройствах. Записанные байты, направленные в дескрипторы CONSOLE (объекты ввода в консоль), не учитываются;

OtherTransferCount - это число байтов, переданных в ходе операций ввода/вывода, созданных процессом и не являющихся ни операциями чтения, ни операциями записи, включая ввод/вывод в файлах, в сети и в устройствах. В качестве примера операции такого типа можно привести функцию управления. Прочие байты, направленные в дескрипторы CONSOLE (объекты ввода в консоль), не учитываются.

Пример:

```
program Process;
var
  Process : array 0 of string;
  i : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Process := @UAF->GetListProcess("bds", true);
  writeln("Процессы запущенные: ");
  for i := 0 to Len(Process) - 1
    writeln("Process[", i, "] = ", Process[i]);
  endfor;
end.

// Процессы запущенные:
//Process[0] = bds.exe, 27.12. 2010 12:18:25,3324,2088,"C:\Program
Files\Borland\BDS\4.0\Bin\bds.exe" -pDelphi,C:\Program
Files\Borland\BDS\4.0\Bin\bds.exe,8,591,9336,106912,80909,87792,107536,372796,374972,1
1824616,447184,409444

program Process1;
var
  Process : array 0 of string;
  i : int;
  Flag : bool;
```

```
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Process := @UAF->GetListProcess("bds", ",", false);
  writeln("Процессы запущенные: ");
  for i := 0 to Len(Process) - 1
    writeln("Process[", i, "] = ", Process[i]);
  endfor;
end.

// Процессы запущенные:
//Process[0] = bds.exe,3324

program Process2;
var
  Process : array 0 of string;
  Sep : array 1 of string = ("|");
  Info : array 0 of string;
  i : int;
  j : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Process := @UAF->GetListProcess("services.exe", Sep, true);
  writeln("Процессы запущенные: ");
  for i := 0 to Len(Process) - 1
    Info := Split(Process[i], Sep);
    for j := 0 to Len(Info) - 1
      switch j
        case 0:
          writeln("------------------------------------------------------------");
          writeln("Процесс: ", Info[0]);
          writeln("------------------------------------------------------------");
        endcase;
        case 1:
          writeln("Дата создания процесса: ", Info[1]);
        endcase;
        case 2:
          writeln("Идентификатор процесса: ", Info[2]);
        endcase;
        case 3:
          writeln("Код (ID) создавшего процесса: ", Info[3]);
        endcase;
        case 4:
          writeln("Командная строка: ", Info[4]);
        endcase;
        case 5:
          writeln("Выполняемый путь: ", Info[5]);
        endcase;
        case 6:
          writeln("Базовый приоритет: ", Info[6]);
        endcase;
        case 7:
          writeln("Дескрипторов кол: ", Info[7]);
        endcase;
        case 8:
          writeln("Память КБ: ", Info[8]);
        endcase;
```

```
        case 9:
          writeln("Память (пик) КБ: ", Info[9]);
        endcase;
        case 10:
          writeln("Ошибок страницы кол: ", Info[10]);
        endcase;
        case 11:
          writeln("Файл подкачки КБ: ", Info[11]);
        endcase;
        case 12:
          writeln("Файла подкачки (пик) КБ: ", Info[12]);
        endcase;
        case 13:
          writeln("Виртуальная память КБ: ", Info[13]);
        endcase;
        case 14:
          writeln("Виртуальная память (пик) КБ: ", Info[14]);
        endcase;
        case 15:
          writeln("Прочитанных данных байт: ", Info[15]);
        endcase;
        case 16:
          writeln("Записанных данных байт: ", Info[16]);
        endcase;
        case 17:
          writeln("Прочих байт: ", Info[17]);
        endcase;
      endswitch;
    endfor;
  endfor;
end.

//Процессы запущенные:
//-------------------------------------------------------------
//Процесс: services.exe
//-------------------------------------------------------------
//Дата создания процесса: 28.12.2010 08:52:10
//Идентификатор процесса: 712
//Код (ID) создавшего процесса: 668
//Командная строка: C:\WINDOWS\system32\services.exe
//Выполняемый путь: C:\WINDOWS\system32\services.exe
//Базовый приоритет: 9
//Дескрипторов кол: 356
//Память КБ: 4412
//Память (пик) КБ: 6752
//Ошибок страницы кол: 6225
//Файл подкачки КБ: 4044
//Файла подкачки (пик) КБ: 4564
//Виртуальная память КБ: 30224
//Виртуальная память (пик) КБ: 55592
//Прочитанных данных байт: 8492416
//Записанных данных байт: 8304715
//Прочих байт: 171068
```

Методы для работы со службами

function StopService (DisplayNameService: string): bool;

Останов службы.
Параметры:
DisplayNameService – выводимое имя службы.
Возвращаемый результат:
true – служба остановлена или *false*, ошибка при останове службы.
Примечание:
Процесс службы принудительно завершается по истечению заданного интервала времени, определяемый свойством *IntervalWait*.

Пример:

```
// запустить-остановить заданные службы на сервере
function StartStopService (@Serv: server;
                           NameService : arrray 4 of string;
                           Start : bool) : bool;
var
  I : int;
begin
  result := true;
  for I := 0 to Len(NameService) - 1
    if Start then
      result := @Serv->StartService(NameService[I]);
    else
      result := @Serv->StopService(NameService[I]);
    endif;
    OutMess(@Serv->Get_MessErr(), NameService[I], Start);
    if not result then
      exit;
    endif;
  endfor;
end;
```

function StopDepService (DisplayNameService: string): bool;

Останов службы и всех зависимых от неё служб.
Параметры:
DisplayNameService – выводимое имя службы.
Возвращаемый результат:
true – службы остановлены или *false*, ошибка при останове службы.
Примечание:
Процесс службы принудительно завершается по истечению заданного интервала времени, определяемый свойством *IntervalWait*.

Пример:

```
result := @Serv->StopDepService("Диспетчер сетевого DDE");
```

function StartService (DisplayNameService: string): bool ;

Запуск службы.
Параметры:
DisplayNameService – выводимое имя службы.
Возвращаемый результат:
true – служба запущена или *false*, ошибка при запуске службы.

Пример:

```
result := @Serv->StartService("StatClient(основной)");
```

function StartDepService (DisplayNameService: string): bool ;

Запуск службы и всех зависимых от неё служб.
Параметры:
DisplayNameService – выводимое имя службы.
Возвращаемый результат:
true – службы запущены или *false*, ошибка при запуске службы.

Пример:

```
result := @Serv->StartDepService("Диспетчер сетевого DDE");
```

function ControlService (DisplayNameService: string): bool ;

Контроль запуска службы. Проверяется состояние службы, если служба «упала» делается попытка её «поднять».
Параметры:
DisplayNameService – выводимое имя службы.
Возвращаемый результат:
true – служба запущена или *false*, ошибка, невозможно запустить службу.

Пример:

```
var
  I : int;
  // службы на сервере телеобработки
  Main_TO : array 4 of string =
    (
       "Телеобработка",
       "Монитор взаимодействия с Этран",
       "Модуль взаимодействия с АС ЭТД",
       "StatClient"
    );

  procedure OutMess (Mess : string; NameService : string);
  begin
    if not(Mess = "") then
      writeln("Ошибка: ", Mess);
    else
      writeln("Служба: " + "'" + NameService + "'" + " - запущена!");
    endif;
  end;

  procedure ServiceControl (@Serv : server; NameService : string);
  begin
    if not @Serv->ControlService(NameService) then
      OutMess(@Serv->Get_MessErr(), NameService);
    endif;
  end;

  // проверка запуска служб на сервере телеобработки
  for I := 0 to Len(Main_TO) - 1
```

```
    ServiceControl(@CUMR04_TO, Main_TO[I]);
  endfor;
```

function StartMode (DisplayNameService: string): string;

Определение типа запуска службы
Параметры:
DisplayNameService – выводимое имя службы.
Возвращает строку типа запуска службы:
"Boot"
"System"
"Auto"
"Manual"
"Disabled"

Пример:

```
Mode := @Main->StartMode("Телеобработка");
writeln("Тип запуска службы Телеобработка: ", Mode);
```

function ChangeStartMode (DisplayNameService: string, StartMode: string): bool;

Изменение типа запуска службы
Параметры:
DisplayNameService – выводимое имя службы;
StartMode – тип запуска службы, может принимать следующие значения:
"Boot"
"System"
"Auto"
"Manual"
"Disabled"
Возвращаемый результат:
true – изменение прошло успешно или *false*, ошибка.

Пример:

```
Flag := @Main->ChangeStartMode("Телеобработка", "Manual");
```

function StateService (DisplayNameService: string): string;

Определение текущего состояния службы
Параметры:
DisplayNameService – выводимое имя службы.
Возвращает строку текущего состояния службы:
"Stopped"
"Start Pending"
"Stop Pending"
"Running"
"Continue Pending"
"Pause Pending"
"Paused"
"Unknown"

Пример:

```
State := @Main->StateService("Телеобработка");
writeln("Текущее состояние службы Телеобработка: ", State);
```

function StatusService (DisplayNameService: string): string;

Определение статуса службы
Параметры:

DisplayNameService – выводимое имя службы.

Возвращает строку статуса службы:

"OK"
"Degraded"
"Pred Fail"
"Error"
"Starting"
"Stopping"
"Service"
"Unknown"

Пример:

```
Status := @Main->StatusService("Телеобработка");
writeln("Статус службы Телеобработка:", Status);
```

function WaitForStateService (DisplayNameService: string; State: string; WaitTime: int): bool;

Ожидание заданного состояния службы.

Параметры:

DisplayNameService – выводимое имя службы.

State – состояние службы;

WaitTime - время ожидания (в секундах).

Возвращаемый результат:

true – служба находится в заданном состоянии или *false*, служба не находится в заданном состоянии.

Пример:

```
if @Main->WaitForStateService("Телеобработка", "Stopped", 60) then
  writeln("Служба Телеобработка перешла в состояние – остановлена!");
endif;
```

function GetListService (DisplayNameService: string; State: string;
Delim: string; FullForm: bool): array of string;

Получить список служб по заданным условиям.

Параметры:

DisplayNameService – выводимое имя службы (или часть имени), если DisplayNameService = "", то выдается информация по всем службам;

State – состояние службы, если *State = "",* то выдается информация по службам с любым состоянием;

Delim – строка символов разделителей полей возвращаемых данных*;*

FullForm - признак выдачи результатов, если true - полная форма, иначе краткая.

Возвращаемый результат*:*

Строковый массив с результатом запроса. Если служба по заданным условиям, не найдена, то возвращается массив нулевой длины. В строке массива представлена информация о службе. Формат строки массива представления информации о службе зависит от признака выдачи результатов.

Полная форма (*FullForm* = *true*). Формат элемента массива:

<*Name*><Delim><*DisplayName*>< Delim ><*PathName*>< Delim ><*ServiceType*><Delim>
<*Status*>< Delim ><*State*>< Delim ><*Started*>< Delim ><*StartMode*>< Delim >
<*StartName*>< Delim ><*AcceptPause*>< Delim ><*AcceptStop*>

Краткая форма (*FullForm* = *false*)). Формат элемента массива:

<*Name*><Delim><*DisplayName*>

где:

Delim – строка символов разделителей полей;
Name – имя службы;
DisplayName – выводимое имя службы;
PathName – полный путь выполняемого файла службы;
ServiceType – тип службы;
Status – текущий статус службы;
State – текущее состояние службы;
Started – служба была запущена;
StartMode – тип запуска службы;
StartName – учетная запись под которой выполняется служба;
AcceptPause – служба может перейти в состояние - пауза;
AcceptStop – служба может быть остановлена.

Пример:

```
program Services;
var
  Service : array 0 of string;
  i : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Service := @UAF->GetListService("Служба", "Running", ",", false);
  writeln("Службы запущенные: ");
  for i := 0 to Len(Service) - 1
    writeln("Service[", i, "] = ", Service[i]);
  endfor;
end.
//Службы запущенные:
//Service[0] = BITS,Фоновая интеллектуальная служба передачи (BITS)
//Service[1] = ERSvc,Служба регистрации ошибок
//Service[2] = Nla,Служба сетевого расположения (NLA)
//Service[3] = srservice,Служба восстановления системы
//Service[4] = SSDPSRV,Служба обнаружения SSDP
```

```
//Service[5] = W32Time,Служба времени Windows

program Services1;
var
  Service : array 0 of string;
  i : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Service := @UAF->GetListService("Служба регистрации ошибок", "Running", ",” true);
  writeln("Службы запущенные: ");
  for i := 0 to Len(Service) - 1
    writeln("Service[", i, "] = ", Service[i]);
  endfor;
end.

//Службы запущенные:
//ERSvc,Служба регистрации ошибок,C:\WINDOWS\System32\svchost.exe -k netsvcs,Share
Process,OK,Running,true,Auto,LocalSystem,false,true

program Services2;
var
  Service : array 0 of string;
  Sep : array 1 of string = ("|");
  Info : array 0 of string;
  i : int;
  j : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Service := @UAF->GetListService("Служба регистрации ошибок", "Running", Sep, true);
  writeln("Службы запущенные: ");
  for i := 0 to Len(Service) - 1
    Info := Split(Service[i], Sep);
    for j := 0 to Len(Info) - 1
      switch j
        case 0:
          writeln("-------------------------------------------------------------");
          writeln("Служба: ", Info[0]);
          writeln("-------------------------------------------------------------");
        endcase;
        case 1:
          writeln("Выводимое имя: ", Info[1]);
        endcase;
        case 2:
          writeln("Путь: ", Info[2]);
        endcase;
        case 3:
          writeln("Тип: ", Info[3]);
        endcase;
        case 4:
          writeln("Статус: ", Info[4]);
        endcase;
        case 5:
          writeln("Состояние: ", Info[5]);
        endcase;
```

```
        case 6:
          writeln("Запущена: ", Info[6]);
        endcase;
        case 7:
          writeln("Тип запуска: ", Info[7]);
        endcase;
        case 8:
          writeln("Учетная запись: ", Info[8]);
        endcase;
        case 9:
          writeln("Возможность установки паузы: ", Info[9]);
        endcase;
        case 10:
          writeln("Возможность останова: ", Info[10]);
          writeln("");
        endcase;
      endswitch;
    endfor;
  endfor;
end.

//Службы запущенные:
//-------------------------------------------------------
//Служба: ERSvc
//-------------------------------------------------------
//Выводимое имя: Служба регистрации ошибок
//Путь: C:\WINDOWS\System32\svchost.exe -k netsvcs
//Тип: Share Process
//Статус: OK
//Состояние: Running
//Запущена: true
//Тип запуска: Auto
//Учетная запись: LocalSystem
//Возможность установки паузы: false
//Возможность останова: true
```

function GetListDepService (DisplayNameService: string;
Delim: string; Dependent: bool): array of string;

Получить список служб от которых зависит данная служба или список служб которые зависят от данной службы

Параметры:

DisplayNameService – выводимое имя службы;

Delim – строка символов разделителей полей возвращаемых данных*;*

Dependent - признак выдачи результатов, если true – выдаются службы от которых зависит указанная служба, иначе- выдаются службы, которые зависят от указанной службы.

Возвращаемый результат*:*

Строковый массив с результатом запроса. Если служба по заданным условиям, не найдена, то возвращается массив нулевой длины. В строке массива представлена информация о службе.

Формат элемента массива:

<*Name*><Delim><*DisplayName*>

где:

Delim – строка символов разделителей полей;
Name – имя службы;
DisplayName – выводимое имя службы;

Пример:

```
program DepServices;
const
  Name = "Система событий COM+";
  Sep = "|";
var
  Service : array 0 of string;
  Info : array 0 of string;
  i : int;
  Flag : bool;
  @UAF : server;
begin
  Flag := @UAF->Connect();
  Service := @UAF->GetListDepService(Name, Sep, true);
  writeln("Список служб, от которых зависит служба ", "'", Name, "'", ":");
  for i := 0 to Len(Service) - 1
    Info := Split(Service[i], Sep);
    writeln(" ", i+1, ". Имя: ", ApostStr(Info[0]),
            " Выводимое имя: ", ApostStr(Info[1]));
  endfor;
end.

//Список служб, от которых зависит служба 'Система событий COM+':
// 1. Имя: "RpcSs" Выводимое имя: "Удаленный вызов процедур (RPC)"
```

Методы для работы с файлами

function CopyFile (SourceFile: string; DestinFile: string): bool;

Копирование файла (или каталога). Перезапись существующего файла (или каталога) не поддерживается.

Параметры:

SourceFile – имя файла источника;
DestinFile – имя файла приемника.

Возвращаемый результат:

True – файл успешно скопирован или *False*, если произошла ошибка.

Пример:

```
Source := "d:\temp\alfa.atg";
Destin := "d:\test\alfa.atg";
Flag := @UAF->CopyFile(Source, Destin);
if Flag then
  writeln("Копирование файла d:\temp\alfa.atg в d:\test\alfa.atg");
else
  writeln("Ошибка копирование файла d:\temp\alfa.atg в d:\test\alfa.atg");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function RenameFile (OldFile: string; NewFile: string): bool;

Переименование файла (или каталога). Переименование невозможно, если файл (или каталог) *NewFile* находится на другом диске или требуется перезапись существующего файла.

Параметры:

OldFile – старое имя файла;

NewFile – новое имя файла.

Возвращаемый результат:

True – файл успешно переименован или *False*, если произошла ошибка.

Пример:

```
if @UAF->RenameFile("d:\test\alfa.atg", "d:\test\alfa1.atg") then
  writeln("Переименование файла d:\temp\alfa.atg в d:\test\alfa1.atg");
else
  writeln("Ошибка переименования файла d:\temp\alfa.atg в d:\test\alfa1.atg");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function DeleteFile (NameFile: string): bool;

Удаление файла (или каталога).

Параметры:

NameFile – имя файла.

Возвращаемый результат:

True – файл успешно удален или *False*, если произошла ошибка.

Пример:

```
if @UAF->DeleteFile("d:\test\alfa1.atg") then
  writeln("Удаление файла d:\test\alfa1.atg");
else
  writeln("Ошибка удаления файла d:\test\alfa1.atg");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function FileExists (NameFile: string): bool;

Проверка на существование файла.

Параметры:

NameFile – имя файла.

Возвращаемый результат:

True – файл существует или *False*, если отсутствует.

Пример:

```
if @UAF->FileExists("d:\temp\alfa.atg") then
  writeln("Файл d:\temp\alfa.atg - существует");
else
  writeln("Файл d:\temp\alfa.atg - не существует");
endif;
```

function GetListFiles (NameDirectory: string): array of string;

Получить список файлов из заданного каталога.

Параметры:

NameDirectory – имя каталога.

Возвращаемый результат:

Строковый массив имен файлов.

Пример:

```
ArrFiles := @UAF->GetListFiles("d:\temp");
```

Методы для работы с каталогами

function CopyDirectory (SourceDirectory: string;
DestinDirectory: string): bool;

Копирование каталога. Копирование каталога не поддерживается, если требуется перезапись существующих файлов.

Параметры:

SourceDirectory – имя каталога источника;

DestinDirectory – имя каталога приемника.

Возвращаемый результат:

True – каталог успешно скопирован или False, если произошла ошибка.

Пример:

```
if @UAF->CopyDirectory("d:\test", "d:\test1") then
  writeln("Копирование директории d:\test в d:\test1");
else
  writeln("Ошибка копирования директории d:\test в d:\test1");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function RenameDirectory (OldDirectory: string;
NewDirectory: string): bool;

Переименование каталога.

Параметры:

NewDirectory – старое имя каталога;

OldDirectory – новое имя каталога.

Возвращаемый результат:

True – каталог успешно переименован или *False*, если произошла ошибка.

Пример:

```
if @UAF->RenameDirectory("d:\test1", "d:\test2") then
  writeln("Переименование директории d:\test1 в d:\test2");
else
  writeln("Ошибка переименовании директории d:\test1 в d:\test2");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function DeleteDirectory (NameDirectory: string): bool;

Удаление каталога.

Параметры:

NameDirectory – имя каталога.

Возвращаемый результат:

True – каталог успешно удален или *False*, если произошла ошибка.

Пример:

```
if @UAF->DeleteDirectory("d:\test2") then
  writeln("Удаление директории d:\test2");
else
  writeln("Ошибка удаления директории d:\test2");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

function DirectoryExists (NameDirectory: string): bool;

Проверка на существование каталога.

Параметры:

NameDirectory – имя каталога.

True – каталог существует или *False*, если отсутствует.

Пример:

```
if @UAF->DirectoryExists("d:\test") then
  writeln("Директория d:\test - существует");
else
  writeln("Директория d:\test - не существует");
endif;
```

Методы для работы с общими ресурсами

function CreateShare (Path: string;
Name: string;
TypeShare: string;
MaximumAlloved: int;
Description: string;
Password: string): bool;

Создание общего ресурса .

Параметры:

Path – локальный путь к общему ресурсу;

Name – имя общего ресурса;

TypeShare – тип общего ресурса;

Значение	**Описание**
“DiskDrive”	Disk Drive

“PrintQueue”	Print Queue
“Device”	Device
“IPC”	IPC
“DiskDriveAdmin”	Disk Drive Admin
“PrintQueueAdmin”	Print Queue Admin
“DeviceAdmin”	Device Admin
“IPCAdmin”	IPC Admin

MaximumAlloved – максимально возможное количество подключений к общему ресурсу;

Description – описание общего ресурса;

Password – пароль доступа к общему ресурсу.

Возвращаемый результат:

True – общий ресурс успешно создан или *False*, если произошла ошибка.

Пример:

```
if @UAF->CreateShare("d:\temp", "Test_Share", "DiskDriveAdmin", 2,
                     "Папка общего доступа", "admin") then
  writeln("Создание папки общего доступа d\temp");
else
  writeln("Ошибка создания папки общего доступа d:\temp - Test_Share");
  writeln("Ошибка: ", @UAF->Get_MessErr());
  exit;
endif;
```

Ниже приведен пример программы копирования файла с рабочего места в общую папку на сервере.

```
#on_wait
#on_log

program CopyOnNetworkDisk;
const
  LocalName = "z:";
  NameShare = "Test_Share";
  Script =
<!--
var
  Network = new ActiveXObject("WScript.Network");
// Подключение сетевого диска
```

```
function MapDrive(LocalName, RemoveName, User, Password)
{
  try {
    Network.MapNetworkDrive(LocalName, RemoveName,
                            false, User, Password);
  }
  catch (e) {
    return e.description;
  }
  // возвращаем результат
  return "";
}  // MapDrive

// Отключение сетевого диска
function RemoveDrive(LocalName)
{
  try {
    Network.RemoveNetworkDrive(LocalName);
  }
  catch (e) {
    return e.description;
  }
  // возвращаем результат
  return "";
}  // RemoveDrive
-->;

var
  Source : string;
  Destin : string;
  NameProg : string;
  Err : string;
  Flag : bool;
  @CUMR02APL : server = ("Тестовый сервер", "1.7.32.39",
                         "UAF", "adm", 60, 360);
begin
  writeln("Соединение с сервером: ", @CUMR02APL.Host);
  if not @CUMR02APL->Connect() then
    writeln("Ошибка: ", @CUMR02APL->Get_MessErr());
    exit;
  endif;
  if @CUMR02APL->CreateShare("c:\temp", NameShare, "DiskDrive", 2,
                             "Папка общего доступа", @CUMR02APL.Password) then
    writeln("Создание папки общего доступа c:\temp на сервере: ",
            @CUMR02APL.Host);
  else
    writeln("Ошибка создания папки общего доступа c:\temp - " + NameShare +
            " на сервере: ", @CUMR02APL.Host);
    writeln("Ошибка: ", @CUMR02APL->Get_MessErr());
    exit;
  endif;
  writeln("Подключаем сетевой диск");
  ScriptIni("JScript", Script);
  Err := ScriptRun("MapDrive", Type(Err), LocalName,
                   "\\" + @CUMR02APL.Host + "\" + NameShare,
                   @CUMR02APL.User, @CUMR02APL.Password);
  if not IsEmpty(Err) then
    writeln("Ошибка: ", Err);
    exit;
```

```
  endif;
  writeln("Копирование файла на сервер");
  Source := "d:\temp\alfa.atg";
  Destin := LocalName + "\alfa.atg";
  CopyFile(Source, Destin, true);
  writeln("Отключаем сетевой диск");
  ScriptIni("JScript", Script);
  Err := ScriptRun("RemoveDrive", Type(Err), LocalName);
  if not IsEmpty(Err) then
    writeln("Ошибка: ", Err);
    exit;
  endif;
  writeln("Удаляем папку общего доступа");
  if @CUMR02APL->ShareExists(NameShare) then
    if @CUMR02APL->DeleteShare(NameShare) then
      writeln("Папка общего доступа c:\temp - " + NameShare + ", удалена");
    else
      Err := @CUMR02APL->Get_MessErr();
      writeln("Ошибка удаления папки общего доступа c:\temp - " +
              NameShare + ". ", Err);
    endif;
  endif;
end.
```

function DeleteShare (NameShare: string): bool;

Удаление общего ресурса.

Параметры:

NameShare – имя общего ресурса.

Возвращаемый результат:

True – общий ресурс успешно удален или *False*, если произошла ошибка.

Пример:

```
if @UAF->ShareExists("Test_Share") then
  if @UAF->DeleteShare("Test_Share") then
    writeln("Папка общего доступа d:\temp - Test_Share, удалена");
  else
    writeln("Ошибка удаления папки общего доступа d:\temp - Test_Share");
  endif;
endif;
```

function ShareExists (NameShare: string): bool;

Проверка на существование общего ресурса.

Параметры:

NameShare – имя общего ресурса.

Возвращаемый результат:

True – общий ресурс существует или *False*, если отсутствует.

Пример:

```
if @UAF->ShareExists("Test_Share") then
  if @UAF->DeleteShare("Test_Share") then
    writeln("Папка общего доступа d:\temp - Test_Share, удалена");
  else
    writeln("Ошибка удаления папки общего доступа d:\temp - Test_Share");
```

```
  endif;
endif;
```

Разные методы

function NameComputer(): string;

Определяет имя сервера.
Возвращаемый результат:
Строка имени сервера.

Пример:

```
Name := @Serv->NameComputer();
writeln("Имя сервера: ", Name);
```

function Reboot(): bool;

Осуществляет перезагрузку сервера.
Возвращаемый результат:
true – перезагрузка прошла успешно или *false*, если произошла ошибка.

Пример:

```
Flag := @Serv->Reboot();
```

function Get_MessErr(): string;

Сообщение об ошибке последней выполненной команды
Возвращаемый результат:
Возвращает строку сообщения об ошибке или пустую строку, если ошибок нет.

Пример:

```
Err := @Main->Get_MessErr();
if not IsEmpty(Err) then
  writeln("Ошибка: ", Err);
endif;
```

Стандартные процедуры и функции

Функции для работы со строками

function StrToInt (S: string): int;

Перевод строки *S* в целое число.

Пример:

```
N := StrToInt("123");
N := StrToInt(Str);
```

function IntToStr (N: int): string;

Возвращает строку, которая является результатом перевода целого числа *N* в строку.

Пример:

```
Str := IntToStr(123);
Str := IntToStr(N);
```

function Format (StrFormat: string; Value0 [; Value1;] [Value2]): string;

Возвращает отформатированную строку заданного числового значения *Value (*или значений *int* или *real)* в соответствии со строкой формата *StrFormat*.

Строка формата состоит из двух следующих элементов: обычных печатаемых символов, отображаемых в исходном виде, а также команд форматирования. Ниже приведена общая форма команд форматирования.

{argnum[, width][: fmt]}

где

argnum – номер отображаемого аргумента, начиная с 0;
width – минимальная ширина поля;
fmt – спецификатор формата.

Параметры width и fmt не являются обязательными. Поэтому в своей простейшей форме команда форматирования просто указывает конкретные аргументы для отображения. Например, команда {0} указывает аргумент *Value0*, команда {1} – аргумент *Value1* и т.д. Если во время выполнения программы в форматирующей строке встречается команда форматирования, то вместо неё подставляется и затем отображается соответствующий аргумент, определяемый параметром *argnum*. Следовательно, от спецификатора формата в форматирующей строке зависи, где именно будут отображаться соответствующие данные. А номер аргумента определяет конкретный форматируемый аргумент. Если в команде форматирования указывается параметр *fmt*, то данные отображаются в указываемом формате. В противном случае используется формат, выбираемый по умолчанию. Если же в команде форматирования указывается параметр *width*, то выводимые данные дополняются пробелами для достижения минимально необходимой ширины поля. При положительном значении параметра width выводимые данные выравниваются по правому краю, а при отрицательном значении – по левому краю.

Строки стандартных числовых форматов служат для форматирования стандартных числовых типов. Стандартная строка числового формата имеет вид *Axx*, где *A* является символом буквы, называемой спецификатором, а *xx* является опциональным целым числом, называемым спецификатором точности. Спецификатор точности находится в диапазоне от 0 до 99 и влияет на число цифр в результате. Любая строка числового формата, содержащая более одной буквы, включая пробелы, интерпретируется как строка пользовательского числового формата.

В следующей таблице описаны спецификаторы стандартных числовых форматов и отображены примеры выходных данных, производимых каждым спецификатором формата.

Описатель формата	**Имя**	**Описание**

C или c	Валюта	Число преобразуется в строку, представляющую денежные единицы. Требуемое число знаков дробной части задается спецификатором точности.
D или d	Десятичное число	Этот формат доступен только для целых типов. Число преобразуется в строку, состоящую из десятичных цифр (0-9); если число отрицательное, перед ним ставится знак "минус". Минимальное количество знаков в выходной строке задается спецификатором точности. Недостающие знаки в строке заменяются нулями.
E или e	Научный (экспоненциальный)	Число преобразуется в строку вида "-d.ddd…E+ddd" or "-d.ddd…e+ddd », где знак "d" представляет цифру (0-9). Если число отрицательное, в начале строки появляется знак "минус". Перед разделителем целой и дробной части всегда стоит один знак. Требуемое число знаков дробной части задается спецификатором точности. Если спецификатор точности отсутствует, по умолчанию число знаков дробной части равно шести. Регистр спецификатора формата задает регистр буквы, стоящей перед экспонентой ("E" или "e"). Экспонента состоит из знака "плюс" или "минус" и трех цифр. Недостающие до минимума цифры заменяются нулями, если это необходимо.
F или f	Фиксированная запятая	Число преобразуется в строку вида "-ddd.ddd…", где знак "d" представляет цифру (0-9). Если число отрицательное, в начале строки появляется знак "минус". Требуемое число знаков дробной части задается спецификатором точности.
G или g	Общий	Число преобразуется в наиболее короткую запись из записи с фиксированной запятой или экспоненциальной записи, в зависимости от типа числа и наличия спецификатора точности. Если спецификатор точности не задан или равен нулю, точность задается типом переменной, как показано в следующем списке. • Int: 10 • Real: 15 Нотация с фиксированной запятой используется, если экспонента результата в экспоненциальной нотации длиннее пяти знаков, но меньше спецификатора точности, в противном случае используется научная нотация. Результат содержит разделитель целой и дробной частей, последние нули дробной части отбрасываются. Если спецификатор точности задан и число значащих цифр результата превосходит его значение, лишние знаки отбрасываются округлением.
N или N	Число	Число преобразуется в строку вида "-d,ddd,ddd.ddd… », где знак '-' при необходимости представляет знак "минус", знак "d" - цифра (0-9), знак "," - разделитель тысяч, а знак "" - разделитель целой и дробной части. Требуемое число знаков дробной части задается спецификатором точности.
P или p	Процент	Число преобразуется в строку, представляющую проценты. Преобразуемое число умножается на 100, чтобы соответствовать процентам. Требуемое число знаков дробной части задается спецификатором точности.

X или x	Шестнадцатеричный	Этот формат доступен только для целых типов. Число преобразуется в строку шестнадцатеричных знаков. Регистр шестнадцатеричных знаков, превосходящих 9, совпадает с регистром указателя формата. Например, чтобы отображать эти цифры в виде "ABCDEF", задайте указатель "X"; и указатель "x", чтобы получить "abcdef". Минимальное количество знаков в выходной строке задается спецификатором точности. Недостающие знаки в строке заменяются нулями.

Создаваемая строка настраиваемого числового формата состоит из одного или нескольких задаваемых спецификаторов числового формата и определяет, как форматируются числовые данные. Любая строка, не являющаяся строкой стандартного числового формата, определяется, как строка пользовательского числового формата.

В следующей таблице описаны спецификаторы пользовательских числовых форматов.

Описатель формата	**Имя**	**Описание**
0	Знак-заместитель нуля	Если форматируемое значение содержит цифру в позиции, где "0" находится в строке формата, то эта цифра копируется в выходную строку; в противном случае "0" отображается в выходной строке. Позиция крайнего левого "0" до десятичной точки и позиция крайнего правого "0" после десятичной точки определяют диапазон цифр, которые всегда включаются в выходную строку. Спецификатор "00" приводит к округлению значения до ближайшего значения цифры, предшествующей десятичной точке-разделителю, если назначено использование округления от нуля. Например, в результате форматирования числа 34,5 с помощью строки "00" будет получена строка со значением"35".
#	Заместитель цифры	Цифра, расположенная в соответствующей позиции форматируемого значения будет скопирована в выходную строку, если в этой позиции в строке формата присутствует знак "#". В противном случае в выходной строке на этой позиции ничего не записывается. Обратите внимание, что ноль не будет отображен, если он не является значащей цифрой, даже если это единственный знак строки. Ноль отображается, только если он является значащей цифрой форматируемого значения. Строка формата "##" приводит к округлению значения до ближайшего значения цифры, предшествующей десятичному разделителю, если назначено использование округления от нуля. Например, в результате форматирования числа 34,5 с помощью строки "##" будет получена строка со значением "35".
.	Разделитель	Первый знак "." определяет расположение разделителя целой и дробной частей, дополнительные знаки "." игнорируются.
,	Разделитель числовых разрядов и масштабирование чисел	Символ "," служит в качестве спецификатора разделителя числовых разрядов и спецификатора масштабирования чисел. Спецификатор разделителя числовых разрядов: Если один или несколько символов "," указаны между двумя заместителями цифр (0 или #), которые форматируют целые разряды числа, то символ разделителя групп вставляется между каждой группой числа в составной части выходных данных. Спецификатор масштабирования

		чисел: Если один или несколько символов "," указаны непосредственно слева от явной или неявной десятичной точки, то форматируемое число делится на 1000 каждый раз, когда встречается спецификатор масштабирования числа. Например, если строка "0,," используется для форматирования числа 100 миллионов, то результатом является "100".
%	Заместитель процентов	При использовании знака "%" в строке форматирования число будет умножено на 100. В соответствующую позицию выходной строки будет вставлен знак "%".
E0 E+0 E-0 e0 e+0 e-0	Экспоненциальное представление чисел	Если в строке формата присутствует один из знаков "E", "E+", "E-", "e", "e+" или "e-", за которым следует по крайней мере один знак "0", число представляется в экспоненциальной форме; между числом и экспонентой вставляется знак "E" или "e". Минимальная длина экспоненты в строке вывода определяется количеством нулей, расположенных за знаком формата. Знаки "E+" и "e+" устанавливают обязательное отображение знака "плюс" или "минус" перед экспонентой. Знаки "E", "e", "E-" и "e-" устанавливают отображение знака только для отрицательных чисел.
'ABC' "ABC"	Символьная строка	Символы, заключенные в одинарные или двойные кавычки, копируются в выходную строку без форматирования.
;	Разделитель секций	Знак ";" служит для разделения секций положительных, отрицательных и нулевых чисел в строке формата. Если в строке пользовательского формата две секции, то крайняя левая секция определяет форматирование положительных чисел и нулей, а крайняя правая – форматирование отрицательных чисел. Если в строке пользовательского формата три секции, то крайняя левая секция определяет форматирование положительных чисел, средняя секция определяет форматирование нулей, а крайняя правая – форматирование отрицательных чисел.

Пример:

```
var
  v : real;
  v2 : real;
  x : int;


v := 17688.65849;
v2 := 0.15;
x := 15;
writeln(Format("{0:F2}", v));
writeln(Format("{0:N5}", v));
writeln(Format("{0:e}", v));
writeln(Format("{0:r}", v));
writeln(Format("{0:p}", v2));
writeln(Format("{0:X}", x));
writeln(Format("{0:D12}", x));
writeln(Format("{0:C}", 189.99));
```

```
writeln(Format("{0:F2} {0:F3} {0:e}", 10.12345));
writeln(Format("{2:d} {0:d} {1:d}", 1, 2, 3));
writeln(Format("{0:00##.#00}", 21.3));

// 17688.66
// 17 688.65849
// 1.768866e+004
// 17688.65849
// 15.00%
// F
// 000000000015
// 189.99р.
// 10.12 10.123 1.012345e+001
// 3 1 2
// 0021.300
```

В качестве примера приведена программа, отображающая текущую сумму и произведение чисел от 0 до 10.

```
#on_wait
#on_log
// Тестовый пример применения функции Format
program TestFormat;
var
  i : int;
  sum : int;
  prod : int;
  str : string;
begin
  sum := 0;
  prod := 1;
  // отобразить текущую сумму и произведение чисел
  for i:=1 to 10
    sum := sum + i;
    prod := prod * i;
    str := Format("Сумма:{0,3:D} Произведение:{1,8:D}", sum, prod);
    writeln(str);
  endfor;
end.
```

Ниже приведен результат выполнения этой программы.

```
************************************************************
TestFormat
  Дата: 15.12.2010  Время: 12:49:53
  Время выполнения: 00:00.00
************************************************************
Сумма:  1 Произведение:       1
Сумма:  3 Произведение:       2
Сумма:  6 Произведение:       6
Сумма: 10 Произведение:      24
Сумма: 15 Произведение:     120
Сумма: 21 Произведение:     720
Сумма: 28 Произведение:    5040
Сумма: 36 Произведение:   40320
Сумма: 45 Произведение:  362880
Сумма: 55 Произведение: 3628800
```

function Length (S: string): int;

Определение длины строки.

Пример:

```
LenStr := Length(S);
```

function Pos (S: string; Value: string): int;

Возвращает индекс первого вхождения указанной подстроки *Value* в исходной строке *S*. Если подстрока *Value* не найдена, возвращается -1, при значении *Value* = “” – функция возвращает 0. Индексация начинается с нуля.

Пример:

```
nI := Pos(S, "test.txt");
nI := Pos(S, NameFile);
```

function LastPos (S: string; Value: string): int;

Возвращает индекс первого вхождения указанной подстроки *Value* в исходной строке *S*. Поиск начинается с позиции последнего знака строки *S* и выполняется от конца к началу, пока не обнаружится *Value*, либо не будет проверен первый знак. Если подстрока *Value* не найдена, возвращается -1, при значении *Value* = “” – функция возвращает последнею позицию индекса строки *S*. Индексация начинается с нуля.

Пример:

```
nI := LastPos(S, "test.txt");
nI := LastPos(S, NameFile);
```

function SubStr (S: string; Pos: int; Len: int): int;

Возвращает подстроку максимум *Len* символов, начинающуюся в позиции *Pos* строки *S*.

Пример:

```
if (SubStr(sPar, 0, 2) = "/f") then
  if (sParStr = "") then
    sParStr := SubStr(sPar, 2, Length(sPar) - 2);
  else
    writeln("Ошибка в командной строке, дубликат ключа - ", sPar);
    Err := Err + 1;
  endif;
endif;
```

function Upper (S: string): string;

Возвращает строку, которая является результатом перевода строки *S* в верхний регистр.

Пример:

```
S := Upper(S);
```

function Lower (S: string): string;

Возвращает строку, которая является результатом перевода строки *S* в нижний регистр.

Пример:

```
S := Lower(S);
```

function PadLeft (S: string; Width: int): string;

Возвращает строку, которая является результатом добавления пробелов слева в строку *S* до указанной общей длины *Width*.

Пример:

```
S := PadLeft("+++++",10);
//     +++++
```

function PadRight (S: string; Width: int): string;

Возвращает строку, которая является результатом добавления пробелов справа в строку *S* (т.е. в конец строки) до указанной общей длины *Width*.

Пример:

```
S := PadRight("+++++",10);
```

function Trim (S: string): string;

Возвращает строку, которая является результатом удаления из строки *S* начальных и конечных пробелов.

Пример:

```
S := Trim(S);
```

function TrimBeg (S: string; TrimChars: string): string;

Возвращает строку, которая является результатом удаления из строки *S* всех начальных вхождений набора символов строки *TrimChars*. Если строка *TrimChrs* пуста, то из строки *S* удаляются начальные пробелы.

Пример:

```
S := TrimBeg("123ABC456","123456");
// ABC456
```

function TrimEnd (S: string; TrimChars: string): string;

Возвращает строку, которая является результатом удаления из строки *S* всех конечных вхождений набора символов строки *TrimChars*. Если строка *TrimChrs* пуста, то из строки *S* удаляются конечные пробелы.

Пример:

```
S := TrimEnd("123ABC456","123456");
// 123ABC
```

function IsBegStr (S: string; S1: string): bool;

Определяет, совпадает ли начало строки *S* с указанной строкой *S1*, если совпадает, возвращается *true*, иначе *false*. Функция выполняет сравнение по словам с учетом регистра и языка.

Пример:

```
if IsBegStr(sParStr, "-") then
  sParStr := ParamStr(0);
endif;
```

function IsEmpty (S: string): bool;

Указывает, является ли заданная строка *S* пустой, если строка пуста, возвращается *true*, иначе *false*.

Пример:

```
if (IsEmpty(sParStr)) then
  sParStr := ParamStr(0);
endif;
```

function Compare (S1: string; S2: string; IgnoreCase: bool): int;

Сравнивает две указанные строки *S1* и *S2*, игнорируя или учитывая регистр *IgnoreCase*, возвращает целое со знаком, которое является результатом сравнения.

Результат сравнения приведен в следующей таблице.

Значение	Условие
Меньше нуля	Значение параметра *S1* меньше значения параметра *S2*.
Нуль	Значения параметров *S1* и *S2* равны.
Больше нуля	Значение параметра *S1* больше значения параметра *S2*.

Пример:

```
if (Compare(StrA, StrB, true) = 0) then
  writeln("Строки равны");
else
  writeln("Строки не равны");
endif;
```

function ApostStr (S: string): string;

Возвращает строку S, заключенную в двойные кавычки.

Пример:

```
sSQL := sSQL + "  '" + "move /y " +
  ApostStr(aIniServ[PathDBSource] + "\" + sNameArc + "_Log.LDF") + " " +
  ApostStr(aIniServ[PathDBDestin]) + "'" + ", NO_OUTPUT" + CRLF();
```

function Insert (S: string; Index: int; Value: string): string;

Вставляет указанную строку *Value* в строку *S* по заданному индексу *Index*. Возвращает результирующую строку.

Пример:

```
var
  S : string = "abc";

  S := Insert(S, 2, "XYZ");
  writeln(S);

// abXYZc
```

function Remove (S: string; Index: int; Count: int): string;

Удаляет заданное число знаков *Count* из строки *S,* начиная с указанной позиции *Index*. Возвращает результирующую строку.

Пример:

```
var
  S : string = "abcXYZ";

  S := Remove(S, 3, 3);
  writeln(S);

// abc
```

function Replace (S: string; Old: string; New: string): string;

Заменяет все вхождения значения *Old* в данной строке *S*, другим значением *New*. Возвращает результирующую строку.

Пример:

```
var
  S : string = "abcXYZ";

  S := Replace(S, "abc", "XYZ");
  writeln(S);

// XYZXYZ
```

function Split (S: string; Separator: var): array of string;

Разделяет строку *S* на элементы, используя заданные разделители *Separator*. Параметр *Separator* может быть представлен строкой состоящей из единственного символа разделителя или строковым массивом, элементы которого состоят из символов разделителей. Результат помещается в строковый массив.

Пример:

```
var
  nI : int;
  StrArr : array 4 of string;
  Sep : array 1 of string = (",");

Str := "один,два,три,четыре";
StrArr := Split(Str, Sep);
for nI := 0 to Len(StrArr) - 1
  writeln("StrArr[", nI, "]= ", StrArr[nI]);
endfor;

Str := Join(Sep[0], StrArr);
writeln("Str= ", Str);
```

function Join (Separator: string; Value: array of string): string;

Слияние массива строк *Value* в единую строку, между элементами массива вставляются разделители *Separator*. Результат помещается в строку.

Пример:

```
var
  nI : int;
  StrArr : array 4 of string;
  Sep : array 1 of string = (",");

Str := "один,два,три,четыре";
StrArr := Split(Str, Sep);
```

```
for nI := 0 to Len(StrArr) - 1
  writeln("StrArr[", nI, "]= ", StrArr[nI]);
endfor;


Str := Join(Sep[0], StrArr);
writeln("Str= ", Str);
```

Функции для работы с регулярными выражениями

Регулярные выражения представляют собой особым образом отформатированные строки, используемые для поиска шаблонов в тексте, а также для подтверждения корректности информации в целях обеспечения надлежащего формата данных. Ниже подробно описан набор знаков, операторов и конструкций, которые используются для определения регулярных выражений.

Escape-знаки

Большинство основных операторов языка регулярных выражений — это одиночные знаки без escape-знака. Escape-знак "\" (одиночная обратная косая черта) указывает модулю обработки регулярных выражений, что следующий за ним знак не является оператором. Например, звездочка (*) интерпретируется модулем обработки как квантор повторения, а звездочка, перед которой стоит знак обратной косой черты (*), — как знак Юникода 002A.

Escape-знаки, перечисленные в следующей таблице, распознаются как в регулярных выражениях, так и в шаблонах замены.

Escape-знак	Описание
Обычные знаки	Все знаки, кроме ".", "$", "^", "{", "[", "(", "|", ")", "*", "+", "?" и "\" соответствуют сами себе.
\a	Соответствует знаку колокольчика (будильника) \u0007.
\b	Соответствует символу BACKSPACE \u0008, если находится в классе знаков []. В других случаях см. примечание после этой таблицы.
\t	Соответствует знаку табуляции \u0009.
\r	Соответствует знаку возврата каретки \u000D.
\v	Соответствует знаку вертикальной табуляции \u000B.
\f	Соответствует знаку перевода страницы \u000C.
\n	Соответствует знаку новой строки \u000A.
\e	Соответствует escape-знаку \u001B.

\040	Соответствует знаку ASCII как восьмеричному числу (до трех разрядов). Числа, начинающиеся не с нуля, представляют собой обратные ссылки, если они состоят из одной цифры или соответствуют номеру собираемой группы. Дополнительные сведения см. в разделе Обратные ссылки. Например, знак \040 соответствует пробелу.
\x20	Соответствует знаку ASCII в шестнадцатеричном представлении (строго две цифры).
\cC	Соответствует управляющему знаку ASCII. Например, \cC — CTRL+C.
\u0020	Соответствует знаку Юникода в шестнадцатеричном представлении (строго четыре цифры).
****	Со следующим знаком, который не распознается как escape-знак, соответствует этому знаку. Например, ***** совпадает с **\x2A**.

Подстановки

Подстановки допускаются только в шаблонах замены. В регулярных выражениях вместо них используются обратные ссылки (например \1). В шаблонах замены распознаются только две специальные конструкции: escape-знаки и подстановки. Все синтаксические конструкции, описанные ниже, допускаются только в регулярных выражениях. Они не распознаются в шаблонах замены. Например, шаблон замены a*${txt}b вставляет строку "a*", за которой следует подстрока, соответствующая собираемой группе txt (если она имеется), а затем строка "b". Знак * не распознается как метазнак внутри шаблона замены. Шаблоны $ также не распознаются в шаблонах регулярных выражений. В регулярных выражениях знак $ обозначает конец строки.

В следующей таблице приводится описание именованных и нумерованных шаблонов замены.

Символ	Описание
$ *число*	Замещает последнюю подстроку, соответствующую группе с десятичным номером *число*.
${ *имя* **}**	Замещает последнюю подстроку, соответствующую группе (?<*имя*>).
$$	Замещает один литерал "$".
$&	Замещает копию самого соответствия целиком.
$`	Замещает весь текст входной строки до соответствия.
$'	Замещает весь текст входной строки после соответствия.
$+	Замещает последнюю собранную группу.
$_	Замещает всю входную строку.

Классы знаков

Класс знаков представляет собой набор знаков, который можно сопоставить с входной строкой. Буквенные знаки, escape-знаки и классы знаков можно объединять для создания шаблона регулярного выражения. Классы знаков определяют наборы знаков.

Синтаксис класса знаков

В следующей таблице перечислены классы знаков и их синтаксис.

Класс знаков	Описание
[*группа_знаков* **]**	Группа положительных знаков. Соответствует всем знакам в указанной группе знаков. Группа знаков состоит из одного или нескольких буквенных знаков, escape-знаков, диапазонов знаков или сцепленных классов знаков. Например, чтобы указать все гласные, используйте код [aeiou].. Чтобы указать все знаки препинания и десятичные числа используйте [\p{P}\d].
[^ *группа знаков* **]**	Группа отрицательных знаков. Соответствует всем знакам, которые отсутствуют в указанной группе знаков. Группа знаков состоит из одного или нескольких буквенных знаков, escape-знаков, диапазонов знаков или сцепленных классов знаков. Начальный знак (**^**) является обязательным и указывает на то, что группа знаков представляет собой группу отрицательных знаков, а не положительных. Например, чтобы указать все знаки, кроме гласных, используйте код [^aeiou]. . Чтобы задать все знаки, кроме знаков препинания и десятичных чисел, используйте [^\p{P}\d].
[*первыйЗнак* **-** *последнийЗнак* **]**	Диапазон знаков. Определяет все знаки в диапазоне знаков. Диапазон знаков — это непрерывная последовательность знаков, которая задается указанием первого и последнего знака в последовательности и дефиса между ними. Например, чтобы задать диапазон десятичных цифр от "0" до "9", диапазон строчных букв от "a" до "f" и диапазон прописных букв от "A" до "F" используйте [0-9a-fA-F].
.	Точка. Соответствует любому знаку, кроме **\n**. Обратите внимание, что точка в положительной или отрицательной группе знаков (знак в квадратных скобках) рассматривается как изменяемый буквенный знак, а не как класс знаков.
\p{ *имя* **}**	Соответствует всем знакам в именованном блоке, заданным в параметре *имя.*
\P{ *имя* **}**	Соответствует всем знакам, отсутствующим в именованном блоке, указанном в параметре *имя.*
\w	Соответствует любому алфавитно-цифровому знаку.
\W	Соответствует любому знаку, не являющемуся цифрой или буквой.
\s	Соответствует любому знаку пробела.
\S	Соответствует любому знаку, не являющемуся пробелом.
\d	Соответствует любой десятичной цифре.

\D	Соответствует любому знаку, не являющемуся цифрой.

Параметры регулярных выражений

Можно изменить шаблон регулярного выражения с помощью параметров, влияющих на поведение сопоставления. Параметры регулярных выражений можно задавать в группе знаков регулярного выражения, используя встроенную конструкцию группировки (?imnsx-imnsx:) или другую конструкцию (?imnsx-imnsx).

Во встроенных конструкциях параметра знак минус (-) перед параметром или набором параметров отключает эти параметры.

Встроенный знак	Описание
i	Задает сопоставление, не учитывающее регистр.
m	Устанавливает многострочный режим. Изменяет функцию знаков **^** и **$** так, что они определяют соответствие в начале и конце любой строки, а не только в начале и конце полной строки, содержащей эти строки.
n	Указывает, что допустимыми собираемыми группами являются только группы, заданные явно именем или номером в форме (?<*name*>…). Это позволяет использовать круглые скобки для задания групп, не являющихся собираемыми, без применения синтаксически неудачной конструкции (?:…).
s	Устанавливает однострочный режим. Изменяет функцию точки (.) так, что она соответствует любому знаку, за исключением **\n**.
x	Предписывает исключить из группы знаков пробелы, не являющиеся escape-знаками, и включает комментарии, начинающиеся со знака #. (Список escape-знаков см. в разделе Escape-знаки.) Пробелы никогда не исключаются из состава класса знаков.

Неразложимые утверждения нулевой ширины

Служебные символы, описанные в следующей таблице, не предписывают обработчику перемещаться по строке или обрабатывать те или иные знаки. Они лишь указывают, считается ли найденное соответствие действительным или недействительным, в зависимости от текущего положения в строке. Например, знак (^) означает текущее положение в начале строки или текста. Таким образом, регулярное выражение `^FTP` возвращает только те последовательности знаков "FTP", которые встречаются в начале строки.

Утверждение	Описание
^	Указывает, что совпадение должно находиться в начале текста или строки. Для получения дополнительных сведений смотрите параметр **Multiline** в разделе Параметры регулярных выражений.
$	Указывает, что совпадение должно находиться в конце строкового значения, перед **\n** в конце строки или в конце текста. Для получения дополнительных сведений смотрите параметр **Multiline** в

	разделе Параметры регулярных выражений.
\A	Указывает, что соответствие должно находиться в начале строки (параметр **Multiline** игнорируется).
\Z	Указывает, что соответствие должно находиться в конце строки или перед **\n** в конце строки (параметр **Multiline** игнорируется).
\z	Указывает, что соответствие должно находиться в конце строки (параметр **Multiline** игнорируется).
\G	Указывает, что соответствие должно находиться в той точке, где заканчивается предыдущее соответствие. При использовании с Match.NextMatch() это гарантирует, что соответствие будет непрерывным.
\b	Указывает, что соответствие должно находиться на границе между знаками **\w** (алфавитно-цифровыми) и **\W** (не алфавитно-цифровыми). Эти соответствия должны быть на границах слов (то есть на первом и последнем знаках в словах, разделенных знаками, не являющимися буквами или цифрами). Соответствие может также находиться на границе слова в конце строки.
\B	Указывает, что соответствие не должно находиться на границе**\b**.

Кванторы

Кванторы добавляют в регулярное выражение дополнительное количество данных. Выражение квантора применяется к знаку, группе или классу знаков, которые непосредственно предшествуют ему.

В следующей таблице описаны специальные символы, которые влияют на сопоставление. Количества *n* и *m* являются целочисленными константами.

Квантор	Описание
*****	Соответствует любому числу совпадений или их отсутствию, например: \w* или (abc)*. Эквивалентно **{0,}**.
+	Соответствует любому числу совпадений, например: \w+ или (abc)+. Эквивалентно **{1,}**.
?	Соответствует единственному совпадению или его отсутствию, например: \w? или (abc)?. Эквивалентно **{0,1}**.
{ *n* **}**	Задает точное число *n* совпадений, например: (pizza){2}.
{ *n* **,}**	Задает не менее *n* повторений, например: (abc){2,}.
{ *n* **,** *m* **}**	Задает не менее *n*, но не более чем *m* повторений.
***?**	Соответствует первому совпадению, включающему наименьшее возможное количество повторений (равнозначно "отложенному" *).
+?	Задает наименьшее число повторений, но не меньше чем одно (равнозначно "отложенному" +).
??	Задает отсутствие повторений, если это возможно, или одно повторение ("отложенный" квантор ?).

{ *n* **}?**	Эквивалентно **{n}** ("отложенное" **{n}**).
{ *n* **,}?**	Задает как можно меньше повторений, но не меньше чем *n* ("отложенный" **{***n***,}**).
{ *n* **,** *m* **}?**	Задает как можно меньше повторений от *n* до *m* (отложенное **{***n***,***m***}**).

Конструкции группирования

Конструкции группирования отображают часть выражения регулярных выражений и обычно выделяют часть строки входной строки. В следующей таблице описаны конструкции группирования регулярных выражений.

Конструкция группирования	Описание
(*часть выражения* **)**	Выделяет соответствующую часть выражения (или невыделяемую группу). Выделяемые области, использующие **()**, нумеруются автоматически по порядку открывающей скобки, начиная с 1. Первый элемент, элемент номер 0, — это текст, соответствующий всему регулярному выражению.
(?< *имя* **>** *часть выражения***)**	Выделяет соответствующую часть выражения в имени группы или в имени номера. Строка, используемая для компонента *имя*, не должна содержать знаков пунктуации и не может начинаться с цифры. Вместо угловых скобок можно использовать одинарные кавычки, например, (?'name').
(?< *имя1* **-** *имя2* **>** *часть выражения***)**	Сбалансированное определение группы. Удаляет определение ранее определенной группы *имя2* и сохраняет в группе *имя1* интервал между ранее определенной группой *имя2* и текущей группой. Если группа *имя2* не определена, то сопоставление возвращается. Поскольку после удаления последнего определения *имя2* открывается предыдущее определение *имя2*, то эта конструкция позволяет использовать набор выделяемый областей для группы *имя2* как счетчик для отслеживания вложенных конструкций, таких как скобки. В этой конструкции *имя1* — необязательно. Вместо угловых скобок можно использовать одинарные кавычки, например, (?'name1-name2').
(?: *часть выражения***)**	(Невыделяемая группа). Не выделяет часть строки, соответствующую части выражения.
(?imnsx-imnsx: *часть выражения***)**	Использует или отключает заданные параметры внутри части выражения. Например, (?i-s:) включает учет регистра и делает невозможным однострочный режим. Дополнительные сведения см. в разделе Параметры регулярных выражений.
(?= *часть выражения***)**	(Положительное утверждение просмотра вперед нулевой ширины). Продолжает сопоставление, при условии, что часть выражения совпадает справа от этой позиции. Например, \w+(?=\d) соответствует слову, за которым следует цифра, но не соответствует цифре. Эта конструкция не возвращается.
(?! *часть выражения***)**	(Отрицательное утверждение просмотра вперед нулевой ширины). Продолжает сопоставление, только если часть выражения не соответствует справа от этой позиции. Например, \b(?!un)\w+\b соответствует слову, которое не начинается с un.
(?<= *часть выражения***)**	(Положительное утверждение просмотра назад нулевой ширины). Продолжает сопоставление, только если часть выражения соответствует слева от этой позиции. Например, (?<=19)99

	соответствует всем сочетаниям "99", следующим за "19". Эта конструкция не возвращается.
(?<! *часть выражения***)**	(Отрицательное утверждение просмотра назад нулевой ширины). Продолжает сопоставление, только если часть выражения не соответствует слева от этой позиции.
(?> *часть выражения***)**	(Невозвращающая часть выражения (также известное как "жадная" часть выражения)). После того как часть выражения полностью совпала один раз, она не участвует в возвращении по частям. (то есть, часть выражения соответствует только тем строкам, которые сопоставились бы только с одной этой частью выражения). По умолчанию, если соответствие не найдено, то для возвращения будут найдены другие возможные совпадения. Если известно, что возвращение нельзя осуществить, то можно использовать невозвращающую часть выражения для предотвращения ненужного поиска, что повышает производительность.

Именованные выделяемые области нумеруются последовательно по порядку открывающих скобок слева направо (как неименованные выделяемые области), но нумерация именованных областей начинается после нумерации неименованных. Например, шаблон `((?<One>abc)\d+)?(?<Two>xyz)(.*)` создает следующие выделяемые группы по номеру и имени. (первая выделяемая область (номер 0) всегда ссылается на весь шаблон).

Номер	Имя	Шаблон
0	0 (имя по умолчанию)	((?<One>abc)\d+)?(?<Two>xyz)(.*)
1	1 (имя по умолчанию)	((?<One>abc)\d+)
2	2 (имя по умолчанию)	(.*)
3	Один	(?<One>abc)
4	Два	(?<Two>xyz)

Конструкции обратных ссылок

В следующей таблице перечислены дополнительные параметры, добавляющие модификаторы обратных ссылок в регулярное выражение.

Конструкция обратных ссылок	Определение
**** *число*	Обратная ссылка. Например, (\w)\1 находит символы, представленные двойными словами (double word).
\k<*имя*>	Именованная обратная ссылка. Например, (?<char>\w)\k<char> находит символы, представленные двойными словами (double word). Выражение (?<43>\w)\43 делает то же самое. Вместо угловых скобок можно использовать одинарные кавычки, например, \k'char'.

Конструкции изменения

В следующей таблице перечислены специальные знаки, изменяющие регулярное выражение так, что оно может сопоставляться по принципу "либо-либо".

Конструкция изменения	Определение
|	Соответствует любому из элементов, разделенных "|" (вертикальной чертой); например cat|dog|tiger. В первую очередь соответствует крайнее слева слово.
(?(*выражение* **)да|нет)**	Соответствует части "да", если в данной точке выражение совпадает; в противном случае соответствует части "нет". Часть "нет" может быть опущена. Выражение может быть любым допустимым вложенным выражением, но оно превращается в утверждение нулевой ширины, поэтому данный синтаксис равнозначен **(?(?**=*выражение*)**да|нет)**. Обратите внимание, что если выражение является названием именованной группы или номером собираемой группы, конструкция изменения интерпретируется как проверка собираемой группы (как описано в следующей строке таблицы). Чтобы избежать путаницы в таких случаях, можно явно развернуть вложенное **(?=***выражение***)**.
(?(*имя* **)да|нет)**	Соответствует части "да", если имеется соответствие строке именованной собранной группы; в противном случае соответствует части "нет". Часть "нет" может быть опущена. Если заданное имя не соответствует имени или номеру собранной группы, использованной в этом выражении, конструкция изменения интерпретируется как проверка выражения (как описано в предыдущей строке таблицы).

Другие конструкции

В приведенной ниже таблице перечислены подвыражения, изменяющие регулярное выражение.

Конструкция	Определение
(? *imnsx* **-** *imnsx* **)**	Устанавливает или отключает такие параметры, как учет регистра, которые устанавливаются и отключаются внутри шаблона. Сведения о конкретных параметрах см. в разделе Параметры регулярных выражений. Изменение параметров действительно до конца содержащей группы. См. также сведения о конструкции группировки **(?***imnsx***-***imnsx***:)**, которая является формой очистки.
(?#)	Встроенный комментарий внутри регулярного выражения. Комментарий заканчивается первой закрывающей скобкой.
# [до конца строки]	Комментарий режима X. Комментарий начинается от знака **#** без обратной косой черты и продолжается до конца строки (обратите внимание, что для распознавания такого комментария должен быть включен параметр **x**).

function IsMatchReg (S: string; Pattern: string): bool;

Указывает на то, обнаруживается ли соответствие регулярному выражению во входной строке *S*, выражение, указанное в параметре *Pattern*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
var
  S : string = "abcXYZ";

  if IsMatchReg(S, "ab") then ;
    writeln("Найдено!");
```

```
  else
    writeln("Не найдено!");
  endif;
```

function SearchReg (S: string; Pattern: string; Index: int): int;

Осуществляет поиск подстроки в строке *S*, удовлетворяющая регулярному выражению *Pattern* с указанной начальной позиции *Index* в строке. Функция возвращает позицию в исходной строке, в которой обнаружен первый символ найденой подстроки или -1 в противном случае.

Пример:

```
var
  S : string = "abcXYZ";

  if SearchReg(S, "ab", 0) >= 0 then ;
    writeln("Найдено!");
  else
    writeln("Не найдено!");
  endif;
```

function MatchReg (S: string; Pattern: string; FullForm: bool): array of string;

Возвращает в виде строкового массива результат поиска в строке *S* подстроки, задаваемой регулярным выражением *Pattern*. Параметр *FullForm* определяет форму выдачи данных. При *FullForm* = *true* полная форма выдачи данных, иначе краткая.

Краткая форма:

<Элемент_массива> = <Подстрока>

Полнная форма:

<Элемент_массива> = <Подстрока> “|” <Индекс_в_строке>

Пример:

```
program Test;
var
  S : string = "желтый шар красный шар голубой шар";
  P : string = "(\w+)\s+(шар)";
  Res : array 1 of string;
  W : array 1 of string;
  I : int;
begin
  writeln("Найденные подстроки:");
  Res := MatchReg(S, P, true);
  for I := 0 to Len(Res) - 1
    W := Split(Res[I], "|");
    writeln("Подстрока (", I, "): ", W[0], " индекс: ", W[1]);
  endfor;
end.

// Найденные подстроки:
// Подстрока (0): желтый шар индекс: 0
// Подстрока (1): красный шар индекс: 11
// Подстрока (2): голубой шар индекс: 23
```

function ReplaceReg (S: string; Pattern: string; Value: string): string;

В указанной входной строке *S* заменяет все строки, соответствующие указанному регулярному выражению *Pattern*, указанной строкой замены или строкой шаблона замены *Value*. Возвращает результирующую строку.

Пример:

В следующем примере кода функция *ReplaceReg* заменяет дату формата мм/дд/гг на дату формата дд-мм-гг.

```
var
  S : string = "12/22/2011";
  R : string = "\b(?<month>\d{1,2})/(?<day>\d{1,2})/(?<year>\d{2,4})\b";
  V : string = "${day}-${month}-${year}";

  S := ReplaceReg(S, R, V);
  writeln(S);

// 22-12-2011
```

Примечание:

Параметр *Pattern* состоит из различных элементов языка регулярного выражения, с помощью символов описывающих строку для сопоставления. Поиск соответствий начинается с начала строки параметра S.

Подстановки разрешены только в шаблоне замены. Для сходных функциональных возможностей в регулярном выражении используйте обратную ссылку, такую как "\1".

Escape-последовательности и подстановки являются единственными особыми конструкциями, распознаваемыми в шаблоне замены. Все остальные синтаксические конструкции разрешены только в регулярных выражениях и не распознаются в шаблонах замены. Например, шаблон замены "a*${test}b" вставляет строку "a*", за которой следует подстрока, сопоставленная с группой записи "test" (если имеется), а затем строка "b". Символ "*" не распознается в шаблоне замены как метасимвол. Так же, $-шаблоны не распознаются в шаблоне сопоставления регулярного выражения. В регулярном выражении "$" означает конец строки. Другими примерами являются: "$123" замещает последнюю подстроку, сопоставленную с группой номер 123 (десятичное), "${name}" замещает последнюю подстроку, сопоставленную с группой (?<name>).

function SplitReg (S: string; Pattern: string): array of string;

Разделяет входную строку в позициях, определенных шаблоном регулярного выражения. Возвращает массив строк.

Пример:

```
var
  I : int;
  S : string = "07/14/2007";
  A : array 1 of string;
```

```
  A := SplitReg(S, "(-)|(/)");
  for I := 0 to Len(A) - 1
    writeln(A[I]);
  endfor;

// 07
// /
// 14
// /
// 2007
```

Функции для работы со строками даты и времени

Все функции для работы со строками даты и времени, работают со строкой следующего формата: "*yyyy.MM.dd HH:mm:ss*".

Чтобы получить строку текущей даты и времени нужно сделать следующее:

```
StrDateTime := Date() + " " + Time();
```

function Date(): string;

Возвращает строку текущей даты в формате "*yyyy.MM.dd*".

Пример:

```
sDate := Date();

// sDate = 2010.12.04
```

function Time(): string;

Возвращает строку текущего времени в формате "*HH:mm:ss*".

Пример:

```
sTime := Time();

// sTime = 09:34:00
```

function CreateStrDateTime (Year: int; Month: int; Day: int;
Hour: int; Minute: int; Second: int): string;

Возвращает строку даты и времени заданной параметрами: *Year* - годом, *Month* -месяцем, *Day* - днем, *Hour* - часом, *Minute* - минутой и *Second* - секундой.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", Date);

// Date = 2010.12.04 09:34:00
```

function GetDate (Date: string): string;

Возвращает строку *Date* даты и времени, время при этом, равно значению полуночи.

Пример:

```
var
```

```
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", GetDate(Date));

// Date = 2010.12.04 00:00:00
```

function GetDay (Date: string): int;

Возвращает номер дня месяца из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Day = ", GetDay(Date));

// Day = 4
```

function GetDayOfWeek (Date: string): int;

Возвращает номер дня недели из строки *Date* даты и времени. Возвращаемое значение лежит в диапазоне от 0 (воскресенье) до 6 (суббота).

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
switch GetDayOfWeek(Date)
  case 0:
    writeln("Воскресенье");
  endcase;
  case 1:
    writeln("Понедельник");
  endcase;
  case 2:
    writeln("Вторник");
  endcase;
  case 3:
    writeln("Среда");
  endcase;
  case 4:
    writeln("Четверг");
  endcase;
  case 5:
    writeln("Пятница");
  endcase;
  case 6:
    writeln("Суббота");
  endcase;
endswitch;

// Суббота
```

function GetMonth (Date: string): int;

Возвращает номер месяца из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Month = ", GetMonth(Date));

// Month = 12
```

function GetYear (Date: string): int;

Возвращает четырехзначный номер года из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Year = ", GetYear(Date));

// Year = 2010
```

function GetHour (Date: string): int;

Возвращает число часов из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Hour = ", GetHour(Date));

// Hour = 9
```

function GetMinute (Date: string): int;

Возвращает число минут из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Minute = ", GetMinute(Date));

// Minute = 34
```

function GetSecond (Date: string): int;

Возвращает значение числа секунд из строки *Date* даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Second = ", GetSecond(Date));
```

```
// Second = 0
```

function AddDays (Date: string; Value: int): string;

Прибавляет указанное число дней *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddDays(Date, 1));

// Date = 2010.12.05 09:34:00
```

function AddHours (Date: string; Value: int): string;

Прибавляет указанное число часов *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddHours(Date, 20));

// Date = 2010.12.05 05:34:00
```

function AddMinutes (Date: string; Value: int): string;

Прибавляет указанное число минут *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddMinutes(Date, 30));

// Date = 2010.12.04 10:04:00
```

function AddMonths (Date: string; Value: int): string;

Прибавляет указанное число месяцев *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddMonths(Date, 1));
```

```
// Date = 2011.01.04 09:34:00
```

function AddSeconds (Date: string; Value: int): string;

Прибавляет указанное число секунд *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddSeconds(Date, 61));

// Date = 2010.12.04 09:35:01
```

function AddYears (Date: string; Value: int): string;

Прибавляет указанное число лет *Value* к дате *Date* и возвращает новое строковое значение даты и времени.

Пример:

```
var
  Date : string;

Date := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
writeln("Date = ", AddYears(Date, 1));

// Date = 2011.12.04 09:34:00
```

function CompareStrDateTime (T1: string; T2:string): int;

Сравнивает две строки даты и времени *T1* и *T2* и возвращает значение, показывающее, как соотносятся их значения.

Возвращаемое значение, число со знаком, обозначающее относительные значения параметров *T1* и *T2*.

Значение	Условие
Меньше нуля	T1 меньше T2.
Нуль	T1 равно T2.
Больше нуля	T1 больше T2.

Пример:

```
var
  Date1 : string;
  Date2 : string;
  Compare : int;

Date1 := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
Date2 := CreateStrDateTime(2010, 12, 5, 9, 34, 0);
Compare := CompareStrDateTime(Date1, Date2);
if (Compare < 0) then
```

```
  writeln("Date1 меньше Date2");
endif;
if (Compare = 0) then
  writeln("Date1 равно Date2");
endif;
if (Compare > 0) then
  writeln("Date1 больше Date2");
endif;

// Date1 меньше Date2
```

function SubtractStrDateTime (T1: string; T2:string): string;

Вычитает из T1 указанную дату и время T2. Возвращает строковое значение, которое может быть представлено как [-]*д.чч:мм:сс*, где необязательный знак минус указывает, что интервал времени является отрицательным, компонент *д* представляет число дней, компонент *чч* — число часов по 24-часовой временной шкале, *мм* — число минут, *сс* — число секунд. Иными словами, временной интервал состоит из положительного или отрицательного количества дней без указания времени суток или количества дней с указанием времени суток или только из времени суток.

Пример:

```
var
  Date1 : string;
  Date2 : string;
  Compare : int;

Date1 := CreateStrDateTime(2010, 12, 4, 9, 34, 0);
Date2 := CreateStrDateTime(2010, 12, 5, 9, 34, 0);
writeln("Time = ", SubtractStrDateTime(Date2, Date1);

// Time = 1.00:00:00
```

function DateTimeToOLE (Date: string): real;

Преобразует строковое значение даты времени в эквивалентное ему значение даты OLE-автоматизации.

Пример:

```
var
  Date : string;

Date := Date() + " " + Time();
writeln(DateTimeToOA(Date));
```

function OLEToDateTime (Date: real): string;

Преобразует значение даты времени OLE-автоматизации в эквивалентное ему строковое значение даты времени.

Пример:

```
var
  DateOA : real;
  Date : string

Date := OAToDateTime(DateOA);
writeln(Date);
```

function WMIToDateTime (Date: string): string;

Преобразует значение даты времени в формате WMI в эквивалентное ему строковое значение даты времени.

Пример:

```
var
  DateWMI : string;
  Date : string

Date := WMIToDateTime(DateWMI);
writeln(Date);
```

function DateReverse (Date: string): string;

Преобразует значение даты времени в формате “yyyy.MM.dd HH:mm:ss” в эквивалентное ему строковое значение даты времени в формате “dd.MM.yyyy HH:mm:ss”.

Пример:

```
var
  Date : string

Date := DateToddMMyyyy(GetData());
writeln(Date);
```

function DaysInMonth (Year: int; Month: int): int;

Возвращает число дней в указанном месяце указанного года.

Пример:

```
var
  Days : int;

Days := DaysInMonth(2011, 1);
```

function IsLeapYear (Year: int): bool;

Возвращает сведения о том, является ли указанный год високосным.

Пример:

```
var
  Flag : bool;

Flag := IsLeapYear(2013);
```

Функции для работы со строками пути к файлу или каталогу

function CreateFileName (): string;

Возвращает произвольное имя каталога или файла.

Пример:

```
TempNameFile := CreateFileName();
```

function IsPathRoot (Path: string): bool;

Возвращает значение, показывающее, содержит ли указанная строка пути *Path* сведения абсолютного или относительного пути. Возвращаемое значение *true*, если параметр *Path* содержит абсолютный путь; в противном случае — *false*.

Пример:

```
var
  FPath : bool;

FPath := IsPathRoot("c:\Distr");
writeln(FPath);

// true
```

function GetDirectoryRoot (Path: string): string;

Возвращает для заданного пути *Path* строку корневого каталога.

Пример:

```
sPath := GetDirectoryRoot("c:\Distr");

// c:\
```

function ChangeExtension (Path: string; Ext: string): string;

Изменяет расширение пути *Path* на новое расширение (с предшествующей точкой) *Ext*.

Пример:

```
sNameFileLog := ChangeExtension(sNameFileIni, ".log");
```

function GetDirectoryName (Path: string): string;

Возвращает для указанной строки пути *Path* строку каталога.

Пример:

```
sPath := GetDirectoryName(sNameFile);
// определить текущий каталог
SetCurrentDirectory(sPath);
```

function GetExtension (Path: string): string;

Возвращает расширение указанной строки пути *Path* (включая символ “.”).

Пример:

```
sExt := GetExtension(sNameFile);
```

function GetFileName (S: string): string;

Возвращает имя файла и расширение указанной строки пути *Path*.

Пример:

```
sName := GetFileName(sNameFile);
```

function GetFileNameWithoutExtension (Path: string): string;

Возвращает имя файла указанной строки пути *Path* без расширения.

Пример:

```
sName := GetFileNameWithoutExtension(sNameFile);
```

function GetFullPath (Path: string): string;

Возвращает для указанной строки пути *Path* абсолютный путь. Эта функция использует сведения текущего каталога и тома для полного значения параметра *Path*. Если имя файла задано только в *Path*, *GetFullPath* возвращает полный путь текущего каталога. Если передано короткое имя файла, оно расширяется до длинного имени.

Пример:

```
sNameFile := GetFullPath("test.txt");
```

function GetTempFileName(): string;

Создает на диске временный пустой файл с уникальным именем и возвращает полный путь этого файла.

Пример:

```
sNameTempFile := GetTempFileName();
```

function GetTempPath(): string;

Возвращает путь временного каталога текущей системы.

Пример:

```
sNameTempPath := GetTempPath();
```

Функции для работы с массивами

function Len (A: var): int;

Определение размерности массива *A* или длины строки *A*.

Пример:

```
var
  I : int;
  Path : string;
  Files : array 1 of string;

Path := "d:\user\alfa\prog";
Files := GetFiles(Path, "*.*");
for I := 0 to Len(Files) - 1
  writeln("  ", Files[I]);
endfor;
```

procedure ResizeArray (var A: array; NewSize : int);

Изменяет количество элементов в массиве до указанной величины.

Пример:

```
var
  IntArray : array 5 of int = (1, 2, 3, 4, 5);

ResizeArray(IntArray, 2);
```

procedure Clear (var A: array);

Присваивание элементам массива *A* значений по умолчанию.

Пример:

```
var
  IntArray : array 5 of int = (1, 2, 3, 4, 5);

Clear(IntArray);
```

procedure Reverse (var A: array);

Изменение порядка следования элементов массива *A* на обратный.

Пример:

```
var
  IntArray : array 5 of int = (1, 2, 3, 4, 5);

Reverse(IntArray);
```

procedure Sort (var A: array);

Упорядочивание элементов массива *A*.

Пример:

```
var
  IntArray : array 5 of int = (5, 2, 12, 7, 5);

Sort(IntArray);
```

function BinarySearch (A: array, Value): int;

Двоичный поиск значения *Value* в отсортированном массиве *A*. Возвращает индекс найденного элемента. Если элемент не найден, возвращается отрицательное число.

Пример:

```
var
  Index : int;
  IntArray : array 5 of int = (5, 2, 12, 7, 5);

Sort(IntArray);
Index := BinarySearch(IntArray, 12);
```

function IndexOf (A: array, Value): int;

Поиск значения *Value* в массиве *A*. Возвращает индекс первого вхождения элемента. Если элемент не найден, возвращается значение -1.

Пример:

```
var
  Index : int;
  IntArray : array 5 of int = (5, 2, 12, 7, 5);

Index := IndexOf(IntArray, 12);
```

procedure CopyToArray (SourceArray: array; var DestinArray: array);

Копирование всех элементов массива *SourceArray* в массив *DestinArray.*

Пример:

```
var
  IntArray : array 5 of int = (5, 2, 12, 7, 5);
  IntArray1 : array 5 of int;

CopyToArray(IntArray, IntArray1);
```

procedure CopyOfRange (SourceArray: array; IndexSource: int;
var DestinArray: array; IndexDestin: int;
Length: int);

Копирование диапазона элементов массива *SourceArray* в массив *DestinArray. IndexSource* - индекс в массиве *SourceArray*, с которого начинается копирование. *IndexDestin* - индекс в массиве *DestinArray*, с которого начинается сохранение. *Length* – представляет собой число, подлежащих копированию.

Пример:

```
var
  IntArray : array 5 of int = (5, 2, 12, 7, 5);
  IntArray1 : array 10 of int;

Clear(IntArray1);
CopyOfRange(IntArray, 0, IntArray1, 5, 5);
```

function UpperArray (var A: array of string): array of string;

Перевод элементов строкового массива *A* в верхний регистр.

Пример:

```
var
  IntArray : array 5 of string = ("Один", "Два", "Три", "Четыре", "Пять");

IntArray := UpperArray(IntArray);
```

function LowerArray (var A: array of string): array of string;

Перевод элементов строкового массива *A* в нижний регистр.

Пример:

```
var
  IntArray : array 5 of string = ("Один", "Два", "Три", "Четыре", "Пять");

IntArray := LowerArray(IntArray);
```

Функции для работы с файлами

function Eof (F: file): bool;

Функция возвращает *true*, если маркер позиции достиг конца файла *F*, и *false* – в противном случае

Пример:

```
while (not Eof(F))
  // считывем данные из файла в строку
  readln(F, sS);
  sStr :=  sStr + sS;
endwhile;
```

function FileExists (NameFile: string): bool;

Позволяет проверить, существует ли указанный файл *NameFile*. Функция возвращает *true*, если файл существует, и *false* – если нет. *NameFile* – полное имя файла.

Пример:

```
if FileExists("d:\temp\alfa.atg") then
  writeln("Файл d:\temp\alfa.atg - существует");
else
  writeln("Файл d:\temp\alfa.atg - не существует");
endif;
```

function CopyFile (SourceNameFile: string; DestNameFile: string; Overwrite: bool): bool;

Копирует существующий файл *SourceNameFile* в новый файл *DestNameFile*. Перезапись файла с тем же именем разрешена, если *Overwrite* равно *true*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
CopyFile("d:\temp\alfa.atg", "c:\alfa\alfa.atg", true);
```

function DeleteFile (NameFile: string): bool;

Удаляет указанный файл *NameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
DeleteFile("d:\temp\alfa.atg");
```

function MoveFile (SourceNameFile: string; DestNameFile: string): bool;

Перемещает существующий файл *SourceNameFile* в новый путь для файла *DestNameFile*. Разрешает переименование файла. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
MoveFile("d:\temp\alfa.atg", "c:\alfa\alfa.atg");
```

function ReplaceFile (SourceNameFile: string; DestNameFile: string; BackupNameFile: string): bool;

Заменяет содержимое файла *DestNameFile* на содержимое файла *SourceNameFile*, удаляя исходный файл *DestNameFile* и создавая резервную копию замененного файла *BackupNameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Если SourceNameFile и DestNameFile находятся в разных томах, этот метод вызовет ошибку. Если *BackupNameFile* находится в другом томе из исходного файла, файл резервной копии будет удален.

Пример:

```
ReplaceFile("d:\temp\alfa.atg", "d:\alfa\alfa.atg", "d:\alfa\alfa.atgold");
```

function GetFiles (Path: string; SearchTemplate: string): array of string;

Возвращает массив имен файлов из указанного каталога по заданному шаблону поиска. *Path* – путь для поиска. *SearchTemplate* – шаблон поиска. В *SearchPattern* разрешены следующие подстановочные знаки.

Подстановочный знак	Описание
*	Ноль или более символов.
?	Ровно один символ.

Знаки, отличные от подстановочных, представляют сами себя. Так, строка *SearchTempate* *t выполняет поиск всех имен, оканчивающихся буквой t, в *path*. А строка *SearchTempate* s* выполняет поиск всех имен, начинающихся с буквы s, в *Path*. Если в папке нет файлов, или же нет файлов, соответствующих параметру *SearchTempate*, данный метод возвращает пустой массив. Параметр *Path* может задавать сведения об абсолютном или относительном пути. Данные относительного пути интерпретируются относительно текущего рабочего каталога. При работе с параметром *Path* регистр не учитывается.

Пример:

```
const
  SearchTemplate = "*.*";
var
  I : int;
  Path : string;
  Files : array 1 of string;
begin
  Path := "d:\user\alfa\prog";
  Files := GetFiles(Path, SearchTemplate);
  writeln("----------------------------");
  writeln("Список файлов каталога ", Path, " :");
  for I := 0 to Len(Files) - 1
    writeln("  ", Files[I]);
  endfor;
  writeln("----------------------------");
end;
```

function GetLengthFile (NameFile: string): real;

Возвращает длину файла *NameFile* в байтах. Если, файл *NameFile* не существует, функция возвращает значение -1.

Пример:

```
NameFile := "d:\update.doc";
writeln("Длина файла = ", GetLengthFile(NameFile));

// Длина файла = 73728
```

function GetCreationTimeFile (NameFile: string): string;

Возвращает строку даты и времени создания файла *NameFile*.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
writeln("Date = ", GetCreationTimeFile(NameFile));

// Date = 2010.12.05 09:35:51
```

function SetCreationTimeFile (NameFile: string; Date: string): bool;

Устанавливает дату и время *Date* создания файла *NameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
Date := CreateStrDateTime(2010, 12, 4, 9, 0, 0);
SetCreationTimeFile(NameFile, Date);
writeln("Date = ", GetCreationTimeFile(NameFile));

// Date = 2010.12.04 09:00:00
```

function GetLastAccessTimeFile (NameFile: string): string;

Возвращает строку даты и времени последнего обращения к файлу *NameFile*.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
writeln("Date = ", GetLastAccessTimeFile(NameFile));

// Date = 2010.12.05 09:35:51
```

function SetLastAccessTimeFile (NameFile: string; Date: string): bool;

Устанавливает дату и время *Date* последнего доступа к файлу *NameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
Date := CreateStrDateTime(2010, 12, 4, 9, 0, 0);
SetLastAccessTimeFile(NameFile, Date);
writeln("Date = ", GetLastAccessTimeFile(NameFile));

// Date = 2010.12.04 09:00:00
```

function GetLastWriteTimeFile (NameFile: string): string;

Возвращает строку даты и времени последней операции записи в файл *NameFile*.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
writeln("Date = ", GetLastWriteTimeFile(NameFile));

// Date = 2010.12.05 09:35:51
```

function SetLastWriteTimeFile (NameFile: string; Date: string): bool;

Устанавливает дату и время *Date* последней операции записи в файл *NameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
Date := CreateStrDateTime(2010, 12, 4, 9, 0, 0);
SetLastWriteTimeFile(NameFile, Date);
writeln("Date = ", GetLastWriteTimeFile(NameFile));

// Date = 2010.12.04 09:00:00
```

function GetAttrFile (NameFile: string): int;

Возвращает атрибуты заданного файла *NameFile*. Файлы могут иметь следующие атрибуты:

Значение	Имя	Описание
0x01	**ReadOnly**	Файл доступен только для чтения.
0x02	**Hidden**	Файл скрытый и, таким образом, не включается в обычный список каталога.
0x04	**System**	Файл является системным. Этот файл является частью операционной системы или используется исключительно операционной системой.
0x10	**Directory**	Файл представляет собой каталог.
0x20	**Archive**	Архивный статус файла. Приложения используют этот атрибут, чтобы пометить файлы для резервного копирования или удаления.
0x40	**Device**	Зарезервировано для использования в будущем.
0x80	**Normal**	Файл обычный, и другие атрибуты не установлены. Этот атрибут действителен, только если используется отдельно.
0x100	**Temporary**	Файл является временным. Файловые системы для ускорения доступа пытаются держать все данные в памяти, а не сбрасывать их назад в массовую память. Приложение должно стереть временный файл сразу после того, как он перестанет быть нужным.
0x200	**SparseFile**	Файл представляет собой разреженный файл. Разреженными файлами обычно являются большие файлы, в которых в основном нулевые данные.
0x400	**ReparsePoint**	Файл содержит точку повторной обработки, блокирующую определяемые пользователем данные, связанные с файлом или каталогом.
0x800	**Compressed**	Файл сжат.

0x1000	**Offline**	Файл находится в автономном режиме. Данные этого файла недоступны непосредственно.
0x2000	**NotContentIndexed**	Файл не будет индексироваться службой индексирования содержимого операционной системы.
0x4000	**Encrypted**	Зашифрованный файл или каталог. Для файла это означает, что все данные в файле зашифрованы. Для каталога это означает, что шифрование производится по умолчанию для вновь создаваемых файлов и каталогов.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
writeln("Атрибут = ", GetAttrFile(NameFile));

// Атрибут = 128
```

function SetAttrFile (NameFile: string; Attr: int): bool;

Устанавливает атрибуты файла *NameFile*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
NameFile := "d:\user\alfa\prog\test12.alf";
SetAttrFile(NameFile, 0x01);
```

function OpenFile (var F: file; NameFile: string; Write: bool; Append: bool): bool;

Позволяет открыть внешний текстовый файл *NameFile*, с которым связана файловая переменная *F*, в режиме записи *Write=true* или в режиме чтения *Write=false*. В режиме записи *Append* определяет, требуется ли добавить в файл данные. Если файл существует и значение параметра *Append=false*, файл перезаписывается. Если файл существует и значение параметра *Append=true*, в файл добавляются данные. В противном случае создается новый файл. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
function PutFile(sStr: string; sNameFile: string): bool;
var
  F : file;
begin
  // открываем выходной файл для записи
  result := OpenFile(F, sNameFile, true, false);
  if not result then
    exit;
  endif;
  // записываем текстовую строку в файл
  writeln(F, sStr);
  // закрываем выходной файл
  result := CloseFile(F);
end;  // PutFile
```

function CloseFile (var F: file): bool;

Позволяет закрыть внешний текстовый файл, с которым связана файловая переменная *F*. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
// закрываем файл
Flag := CloseFile(F);
```

function AppendStrFile (NameFile: string; Str: string; CodePage: int): bool;

Добавляет указанную строку *Str* в заданной кодировке *CodePage* в файл *NameFile*, создавая файл, если он не существует. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
const
  NameFile = "d:\user\alfa\prog\test2.txt";
var
  Str : string;

Str := "Тестовая строка 1" + CRLF() + "Тестовая строка 2" + CRLF();
AppendStrFile(NameFile, Str, 1251);
```

function ReadArrayFile (NameFile: string; CodePage: int): array of string;

Открывает файл *NameFile*, считывает все строки файла с заданной кодировкой *CodePage* в строковый массив и затем закрывает файл. Функция возвращает строковый массив.

Пример:

```
const
  NameFile = "d:\user\alfa\prog\test2.txt";
var
  Arr : array 1 of string;

Arr := ReadArrayFile(NameFile, 1251);
```

function ReadStrFile (NameFile: string; CodePage: int): array of string;

Открывает файл *NameFile*, считывает все строки файла с заданной кодировкой *CodePage* в строку и затем закрывает файл. Функция возвращает строку.

Пример:

```
const
  NameFile = "d:\user\alfa\prog\test2.txt";
var
  Str : string;

Str := ReadStrFile(NameFile, 1251);
```

function WriteArrayFile (NameFile: string; Arr: array of string; CodePage: int): bool;

Создает новый файл *NameFile*, записывает указанный массив строк *Arr* в этот файл, используя заданную кодировку *CodePage*, и затем закрывает файл. Если целевой файл уже существует, он будет переопределен. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
const
  NameFile = "d:\user\alfa\prog\test2.txt";
var
  Arr : array 2 of string;

Arr[0] := "Строка 1";
Arr[1] := "Строка 2";
WriteArrayFile(NameFile, Arr, 1251);
```

function WriteStrFile (NameFile: string; Str: string; CodePage: int): bool;

Создает новый файл *NameFile*, записывает указанную строку *Str* в этот файл, используя заданную кодировку *CodePage*, и затем закрывает файл. Если целевой файл уже существует, он будет переопределен. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
const
  NameFile = "d:\user\alfa\prog\test2.txt";
var
  Str : string;

Str := "Строка символов";
WriteStrFile(NameFile, Str, 1251);
```

Функции для работы с каталогами

function DirectoryExists (NameDirectory: string): bool;

Позволяет проверить, существует ли указанный каталог *NameDirectory*. Функция возвращает *true*, если каталог существует, и *false* – если нет. *NameDirectory* – имя каталога.

Пример:

```
if DirectoryExists("d:\test") then
  writeln("Директория d:\test - существует");
else
  writeln("Директория d:\test - не существует");
endif;
```

function CreateDirectory (NameDirectory: string): bool;

Создает все каталоги и подкаталоги, указанные в параметре *NameDirectory*. Если каталог уже существует, процедура ничего не делает. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
CreateDirectory("d:\test");
```

function DeleteDirectory (NameDirectory: string; DelSubDir: bool): bool;

Удаляет заданный каталог по заданному пути *NameDirectory* и при установленном параметре *DelSubDir = true*, все подкаталоги и файлы в нем. Если параметр *DelSubDir = false,* удаляется только пустой каталог. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
DeleteDirectory("d:\test");
```

function MoveDirectory (SourceDirectory: string; DestDirectory: string): bool;

Перемещает файл или каталог со всем его содержимым в новое местоположение. *SourceDirectory* – путь к файлу или каталогу, который необходимо переместить. *DestDirectory* – путь к новому местоположению. Если *SourceDirectory* является файлом, то параметр *DestDirectory* также должен быть именем файла. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
MoveDirectory("d:\test", "d:\test1");
```

function GetDirectories (Path: string; SearchTemplate: string): array of string;

Возвращает массив имен каталогов по заданному шаблону поиска в отсортированном виде. *Path* – путь для поиска. *SearchTemplate* – шаблон поиска. В *SearchTemplate* разрешены следующие подстановочные знаки.

Подстановочный знак	Описание
*	Ноль или более символов.
?	Ровно один символ.

Знаки, отличные от подстановочных, представляют сами себя. Так, строка *SearchTempate* *t выполняет поиск всех имен, оканчивающихся буквой t, в *path*. А строка *SearchTempate* s* выполняет поиск всех имен, начинающихся с буквы s, в *Path*. Если в папке нет подкаталогов, или же нет подкаталогов, соответствующих параметру *SearchTempate*, данный метод возвращает пустой массив. Параметр *Path* может задавать сведения об абсолютном или относительном пути. Данные относительного пути интерпретируются относительно текущего рабочего каталога. При работе с параметром *Path* регистр не учитывается.

Пример:

```
const
  SearchTemplate = "*.*";
var
  I : int;
  Path : string;
  Files : array 1 of string;
begin
  Path := "d:\user";
  Files := GetDirectories(Path, SearchTemplate);
  writeln("Список каталогов ", Path, " :");
  for I := 0 to Len(Files) - 1
    writeln("  ", Files[I]);
```

```
  endfor;
end;
```

function GetLogicalDrives(): array of string;

Возвращает массив имен логических устройств (в верхнем регистре) данного компьютера в формате "<имя устройства>:\".

GetLogicalDrives возвращает все устройства, доступные на конкретном компьютере, включая привод для чтения дискет или оптических дисков.

Пример:

```
var
  I : int;
  Drive : array 1 of string;
begin
  Files := GetLogicalDrives();
  writeln("Список логических устройств: ");
  for I := 0 to Len(Drive) - 1
    writeln("  ", Drive[I]);
  endfor;
end;
```

function GetFreelDrive(Drives: array of string): string;

Возвращает строку свободного имени логического устройства (в верхнем регистре) данного компьютера в формате "<имя устройства>:\", на основе массива устройств *Drives,* доступных на конкретном компьютере, в том же формате.

Пример:

```
var
  I : int;
  Drive : array 0 of string;
  DriveFree : string;
begin
  Drive := GetLogicalDrives();
  writeln("Список логических устройств: ");
  for I := 0 to Len(Drive) - 1
    writeln("  ", Drive[I]);
  endfor;
  DriveFree := GetFreeDrive(Drive);
  writeln("Свободное логическое устройство: ", DriveFree);
  readln();
end;
```

function GetCurrentDirectory () : string;

Функция возвращает текущий рабочий каталог приложения.

Пример:

```
// определить текущий каталог
sPath := GetCurrentDirectory();
```

function SetCurrentDirectory (Path: string): bool;

Устанавливает заданный каталог в качестве текущего рабочего каталога приложения. *Path* – путь, который должен быть назначен рабочему каталогу. Функция возвращает *true*, если операция прошла успешно, и *false* – в противном случае.

Пример:

```
// установить текущий каталог
SetCurrentDirectory(sPath);
```

function GetFileSystemEntries (Path: string; SearchTemplate: string): array of string;

Возвращает массив элементов файловой системы (каталоги и файлы) по заданному шаблону поиска. *Path* – путь для поиска. *SearchTemplate* – шаблон поиска. В *SearchTemplate* разрешены следующие подстановочные знаки.

Подстановочный знак	Описание
*	Ноль или более символов.
?	Ровно один символ.

Знаки, отличные от подстановочных, представляют сами себя. Так, строка *SearchTempate* *t выполняет поиск всех имен, оканчивающихся буквой t, в *path*. А строка *SearchTempate* s* выполняет поиск всех имен, начинающихся с буквы s, в *Path*. Если в папке нет подкаталогов, или же нет подкаталогов, соответствующих параметру *SearchTempate*, данный метод возвращает пустой массив. Параметр *Path* может задавать сведения об абсолютном или относительном пути. Данные относительного пути интерпретируются относительно текущего рабочего каталога. При работе с параметром *Path* регистр не учитывается.

Пример:

```
const
  SearchTemplate = "*.*";
var
  I : int;
  Path : string;
  Files : array 1 of string;
begin
  Path := "d:\user\alfa";
  Files := GetFileSystemEntries(Path, SearchTemplate);
  writeln("Список элементов файловой системы ", Path, " :");
  for I := 0 to Len(Files) - 1
    writeln("  ", Files[I]);
  endfor;
end;
```

Функции для работы с файлом инициализации

procedure IniInit (FileName: string);

Осуществляет соединение с указанным файлом инициализации *FileName*.

Примечание:

Метод *IniInit* обязан быть вызван (один раз), перед вызовом других процедур или функций для работы с файлом инициализации.

Пример:

```
IniInit(sNameFile);
```

procedure IniDeleteKey (Section: string; Ident: string);

Удаляет заданный идентификатор *Ident* из заданного раздела *Section.*

Пример:

```
IniDeleteKey("Server", "DBName");
```

procedure IniEraseSection (Section: string);

Удаляет заданный раздел *Section.*

Пример:

```
IniEraseSection("Server");
```

function IniReadSection (Section: string): array of string;

Извлекает имена идентификаторов в указанном разделе *Section* и возвращает их в строковом массиве.

Пример:

```
var
  S : array 1 of string;

S := IniReadSection("Table00");
```

function IniReadSections(): array of string;

Извлекает имена идентификаторов из всех разделов и возвращает их в строковом массиве.

Пример:

```
var
  S : array 1 of string;

S := IniReadSections("Table00");
```

function IniReadSectionValues (Section: string): array of string;

Извлекает все ключи из раздела *Section* и возвращает их в строковом массиве.

Пример:

```
var
  S : array 1 of string;

S := IniReadSectionsValues("Table00");
```

function IniSectionExists (Section: string): bool;

Проверка на существование заданного раздела *Section*.

Пример:

```
if not IniSectionExists("Table00") then
  writeln("Секция [Table00] – не существует!");
endif;
```

function IniReadBool (Section: string; Ident: string; Default: bool): bool;

Возвращает булевское значение заданного идентификатора *Ident* заданного раздела *Section*. Значение по умолчанию *Default*.

Пример:

```
Flag := IniReadBool("Table01", "Insert", true);
```

function IniReadInteger (Section: string; Ident: string; Default: int): int;

Возвращает целое значение заданного идентификатора *Ident* заданного раздела *Section*. Значение по умолчанию *Default*.

Пример:

```
Num := IniReadInteger("Table01", "FieldLen", 2);
```

function IniReadReal (Section: string; Ident: string; Default: real): real;

Возвращает значение с плавающей точкой заданного идентификатора *Ident* заданного раздела *Section*. Значение по умолчанию *Default*.

Пример:

```
Num := IniReadReal("Table01", "FieldPrec", 3.4);
```

function IniReadString (Section: string; Ident: string; Default: string): string;

Возвращает строковое значение заданного идентификатора *Ident* заданного раздела *Section*. Значение по умолчанию *Default*.

Пример:

```
aIniServ[PasswordDB] := IniReadString("Server", "PasswordDB", "");
```

procedure IniWriteString (Section: string; Ident: string; Value: string);

Записывает строковое значение заданное параметром *Value*. Раздел и идентификатор, куда будет записываться значение, указывается параметрами *Section* и *Ident*.

Пример:

```
IniWriteString("Server", "PasswordDB", aIniServ[PasswordDB]);
```

procedure IniWriteBool (Section: string; Ident: string; Value: bool);

Записывает булевское значение заданное параметром *Value*. Раздел и идентификатор, куда будет записываться значение, указывается параметрами *Section* и *Ident*.

Пример:

```
IniWriteBool("Table01", "Insert", Flag);
```

procedure IniWriteInteger (Section: string; Ident: string; Value: int);

Записывает целое значение заданное параметром *Value*. Раздел и идентификатор, куда будет записываться значение, указывается параметрами *Section* и *Ident*.

Пример:

```
IniWriteInteger("Table01", "FieldLen", 2);
```

procedure IniWriteReal (Section: string; Ident: string; Value: int);

Записывает значение с плавающей точкой заданное параметром *Value*. Раздел и идентификатор, куда будет записываться значение, указывается параметрами *Section* и *Ident*.

Пример:

```
IniWriteReal("Table01", "FieldPrec", 12.35);
```

function IniValueExists (Section: string; Ident: string): bool;

Проверяет на существование значение ключа *Ident* заданного раздела *Section*.

Пример:

```
if not IniValueExists("Table01", "Insert") then
  writeln("Ключ [Table00]->Insert – не существует!");
endif;
```

function IniGet_FileName(): string;

Возвращает текущее имя файла инициализации

Пример:

```
Str := IniGet_FileName();
```

procedure IniSet_FileName (Name: string);

Устанавливает имя файла инициализации *Name*. После установки имени файла происходит подсоединение к указанному файлу инициализации.

Пример:

```
IniGet_FileName(Str);
```

procedure IniUpdateFile();

Вызывает обновление файла инициализации после записи значений.

function IniIntToHex (Mean: int; Digits: int): string;

Возвращает число *Mean* в шестнадцатиричном строковом представлении с числом цифр *Digits.*

Пример:

```
Str := IniIntToHex(128, 2);
```

function IniConvOEMtoANSI (Str: string): string;

Возвращает конвертированную строку *Str* из *OEM* в *ANSI* кодировку.

Пример:

```
Str := IniConvOEMtoANSI(Str);
```

function IniConvANSItoOEM (Str: string): string;

Возвращает конвертированную строку *Str* из *ANSI* в *OEM* кодировку.

Пример:

```
Str := IniConvANSItoOEM(Str);
```

function IniDelDelim (Str: string; Delim: string): string;

Возвращает строку *Str* с удаленными ограничителями строки *Delim*.

Пример:

```
Str := IniDelDelim(Str, "'");
```

function IniWordNum (S: string; WordDelim: string): int;

Возвращает количество слов в строке *S*, слова в строке разделены *WordDelim*.

Пример:

```
Num := IniWordNum(Str, ",");
```

function IniWordPos (N: int; S: string; WordDelim: string): int;

Возвращает позицию заданного слова *N* в строке *S, с*лова в строке *S* разделены *WordDelim*.

Пример:

```
Num := IniWordPos(2, Str, ",");
```

function IniWordExtr (N: int; S: string; WordDelim: string): string;

Возвращает значение заданного слова *N* в строке *S, с*лова в строке *S* разделены *WordDelim*.

Пример:

```
S := IniWordExtr(3, Str, ",");
```

function IniWordExtrPos (N: int; S: string; WordDelim: string; var Pos: int): string;

Возвращает значение заданного слова *N* в строке *S,* слова в строке *S* разделены *WordDelim*. *Pos* - определяет позицию следующего слова в строке.

Пример:

```
Pos := 3;
S := IniWordExtrPos(Pos, Str, ",", Pos);
```

function IniStrZero (Num: int; Len: int): string;

Возвращает строковое представление числа *Num* заданной длины *Len* с лидирующими нулями.

Пример:

```
S := IniStrZero(20, 3);
```

procedure InilClose();

Закрывает текущее соединение с файлом инициализации. Освобождает все занятые ресурсы.

Пример:

```
IniClose();
```

Математические функции

function Real (Value: int): real;

Выполняет преобразование значения *Value* типа *int* в *real*.

Пример:

```
var
  R: real;

R := Real(122);
```

function Abs (Value) : value;

Возвращает абсолютное значение заданного числа *Value*.

Пример:

```
MeanReal := Abs(-5.63);
MeanInt := Abs(-5);
```

function Acos (Value: real): real;

Возвращает угол, косинус которого равен указанному числу. *Value* - число, представляющее косинус, где -1 ≤ *Value* ≤ 1. Возвращаемое значение - угол, ϑ, измеренный в радианах, такой, что 0 ≤ ϑ ≤π.

Пример:

```
Mean := Acos(0.33);
```

function Asin (Value: real): real;

Возвращает угол, синус которого равен указанному числу *Value*. *Value* - число, представляющее синус, где -1 ≤ *Value* ≤ 1. Положительное возвращаемое значение представляет угол от оси X против часовой стрелки, отрицательное — угол по часовой стрелке.

Пример:

```
Mean := Asin(0.5);
```

function Atan (Value: real): real;

Возвращает угол, тангенс которого равен указанному числу *Value*. Возвращаемое значение - угол, θ, измеренный в радианах, такой, что -π/2 ≤ θ ≤π/2.

Пример:

```
Mean := Atan(30);
```

function Atan2 (Y: real; X: real): real;

Возвращает угол, тангенс которого равен отношению указанной координаты точки Y и X. возвращаемое значение - угол, θ, измеренный в радианах, такой, что $-\pi \leq \theta \leq \pi$ и $\tan(\theta) = y / x$, где (x, y) — точка в Декартовой системе координат.

Нужно обратить внимание на следующее.

Для (x, y) в квадранте 1, 0 < θ < π/2.
Для (x, y) в квадранте 2, 0 π/2 < θ≤π.
Для (x, y) в квадранте 3, -π < θ < -π/2.
Для (x, y) в квадранте 4, -π/2 < θ < 0.

Для точек за пределами указанных квадрантов возвращаемое значение указано ниже.

Если x равно 0 и y не является отрицательным, θ = 0.
Если x равно 0 и y является отрицательным, θ = π.
Если x является положительным и y равно 0, θ = π/2.
Если x является отрицательным и y равно 0, θ = -π/2.

Пример:

```
Mean := Atan2(2.0, 1.0);
```

function Ceiling (Value: real): real;

Возвращает наименьшее целое число, которое больше или равно заданному числу *Value*.

Пример:

```
Mean := Ceiling(1.0);
Mean := Ceiling(1.1);

// Mean = 1
// Mean = 2
```

function Cos (Value: real): real;

Возвращает косинус указанного угла *Value*.

Угол, *Value*, должен быть выражен в радианах. Для перевода из градусов в радианы нужно умножить значение на $\pi/180$.

Пример:

```
Mean := Cos(15);
```

function Cosh (Value: real): real;

Возвращает гиперболический косинус указанного угла *Value*. Угол, V*alue*, должен быть выражен в радианах.

Пример:

```
Mean := Cosh(0.1);
```

function DivRem (A:int; B:int; var Res:int): int;

Вычисляет частное двух чисел *A* – делимое, *B* – делитель и возвращает остаток в выходном параметре *Res*. Возвращаемое значение содержит частное указанных чисел.

Пример:

```
Mean := DivRem(2147483647, 2, Res);

// Mean = 1073741823
// Res = 1
```

function Exp (Value: real): real;

Возвращает **e**, возведенное в указанную степень Value.

Пример:

```
Mean := Exp(2);
```

function Floor (Value: real): real;

Возвращает наибольшее целое число, которое меньше или равно указанному числу *Value*.

Пример:

```
Mean := Floor(1.9);
Mean := Floor(2.1);

// Mean = 1;
// Mean = 2;
```

function Remainder (X: real; Y:real): real;

Возвращает остаток от деления одного указанного числа *X* на другое указанное число *Y*.

Число, равное *X - (Y Q)*, где *Q* является частным *X / Y*, округленным до ближайшего целого числа (если *X / Y* находится на равном расстоянии от двух целых чисел, выбирается четное число).

Если значение *X - (Y Q)* равно нулю, возвращается значение +0 при положительном *X*, или значение -0 при отрицательном *X*.

Пример:

```
Mean := Remainder(2.0, 3.0);
```

function Log (A: real; NewBase: real): real;

Возвращает логарифм указанного числа *A* в системе счисления с указанным основанием *NewBase*.

Пример:

```
Mean := Log(9.9, 4.9);

// Mean = 1.4425396251981288E+000
```

function Log10 (Value: real): real;

Возвращает логарифм с основанием 10 указанного числа *Value*.

Пример:

```
Mean := Log10(9.9);
```

function Max (Val1; Val2): value;

Возвращает большее из двух указанных чисел *Val1* и *Val2* .

Пример:

```
MeanInt := Max(9, 12);
MeanReal := Max(4.33, 1.78);
```

function Min (Val1; Val2): value;

Возвращает меньшее из двух указанных чисел *Val1* и *Val2* .

Пример:

```
MeanInt := Min(9, 12);
MeanReal := Min(4.33, 1.78);
```

function Pow (X: real; Y:real): real;

Возвращает указанное число *X*, возведенное в указанную степень *Y*.

Пример:

```
Mean := Pow(2.0, 3.0);
```

function Round (Value: real): real;

Округляет заданное значение *Value* до ближайшего целого.

Возвращаемое значение целое число, ближайшее к значению параметра *Value*. Если дробная часть *Value* находится на равном расстоянии от двух целых чисел (четного и нечетного), возвращается четное число.

Пример:

```
Mean := Round(2.55);
```

function Sign (Value): int;

Возвращает значение, определяющее знак числа *Value*.

Возвращаемое значение - число, определяющее знак *value*.

Число	Описание
-1	Значение параметра V*alue* меньше нуля.
0	Значение параметра V*alue* равно нулю.
1	Значение параметра V*alue* больше нуля.

Пример:

```
Mean := Sign(-2.55);
Mean := Sign(-5);
```

function Sin (Value: real): real;

Возвращает синус указанного угла *Value*. Угол, *Value*, должен быть выражен в радианах. Для перевода из градусов в радианы нужно умножить значение на $\pi/180$.

Пример:

```
Mean := Sin(30);
```

function Sinh (Value: real): real;

Возвращает гиперболический синус указанного угла *Value*. Угол, V*alue*, должен быть выражен в радианах.

Пример:

```
Mean := Sinh(30);
```

function Sqrt (Value: real): real;

Возвращает квадратный корень из указанного числа *Value*.

Пример:

```
Mean := Sqrt(4.0);
```

function Tan (Value: real): real;

Возвращает тангенс указанного угла *Value*. Угол, V*alue*, должен быть выражен в радианах.

Пример:

```
Mean := Tan(19.0);
```

function Tanh (Value: real): real;

Возвращает гиперболический тангенс указанного угла *Value*. Угол, V*alue*, должен быть выражен в радианах.

Пример:

```
Mean := Tanh(19.0);
```

function Truncate (Value: real): real;

Вычисляет целую часть числа. Возвращаемое значение целая часть *Value*; то есть, число, остающееся после отбрасывания дробной части.

Пример:

```
Mean := Truncate(12.7);
```

Функции для работы с COM-объектами

function CreateComObj (ProgID: string): comobj;

Создает объект внешнего сервера автоматизации по указанному программному идентификатору объекта *ProgID*. Возвращает созданный объект типа *comobj* или *nil* в случае ошибки.

Пример:

```
FS := CreateComObj("Scripting.FileSystemObject");
```

function Enumerator (Obj: comobj): comobj;

Возвращает объект перечислителя коллекции *Obj* или *nil* в случае ошибки.

Пример:

```
Enum := Enumerator(FS.Drives);
```

function MoveNext (EnumObj: comobj; var Elem: comobj): bool;

Получает элемент коллекции *Elem* из перечислителя коллекции *EnumObj*. Возвращает *true*, если возможно получение следующего элемента коллекции или *false* в противном случае.

Пример:

```
Flag := MoveNext(Enum, Elem);
```

function CurrentEnum (EnumObj: comobj): comobj;

Возвращает текущий элемент коллекции из перечислителя коллекции *EnumObj* или *nil* в случае ошибки.

Пример:

```
Elem := CurrentEnum(Enum);
```

procedure ResetEnum (EnumObj: comobj);

Устанавливает перечислитель коллекции *EnumObj* в начальное положение, перед первым элементом коллекции.

Пример:

```
ResetEnum(Enum);
```

Функции для работы с E-Mail

procedure MailIni (Host: string; Port: int; User: string; Password: string
From: string; To: string; CC: string);

Инициализирует параметры соединения с почтовым сервером, который позволяет отправлять сообщения по протоколу *SMTP* (*Simple Mail Transfer Protocol*). *Host* – имя или *IP* адрес почтового сервера. *Port* – номер порта почтового сервера. *User* – имя учетной записи. *Password* – пароль учетной записи. *From* – адрес отправителя электронной почты. *To* – адрес получателя

электронной почты. *CC* – список адресов получателей копии электронной почты, если список пуст, то передается пустая строка.

Формат строки представления адреса получателя и отправителя:

"<E-Mail>[,<DisplayName>]"

Формат строки представления, списка получателей копии электронной почты:

"[<E-Mail[,<E-Mail>, … , <E-Mail>]"

где

E-Mail – адрес электронной почты;

DisplayName - отображаемое имя.

Примечание:

Процедура *MailIni* должна быть вызвана (один раз), перед вызовом других функций для работы с E-Mail.

Пример:

```
MailIni("67.13.256.48", 25, "ssk/cnt_notif", "adm",
        "Sidorov@fsk.sdd,Сидоров Иван Иванович",
        "Petrov@fsk.sdd,Петров Алексей Николаевич", "");
```

function MailSend (Subject: string; Body: string; AttachFile: string): bool;

Отправляет указанное сообщение *Body* на сервер *SMTP* для доставки получателем. *Subject* – тема сообщения. *AttachFile* – полное имя вложенного файла в сообщение или пустая строка, если его нет. Функция возвращает *true*, если сообщение отправлено успешно или *false* в любом другом случае.

Пример:

```
var
  Flag : bool;

begin
  MailIni("67.13.256.48", 25, "ssk/cnt_notif", "adm",
          "Sidorov@fsk.sdd,Сидоров Иван Иванович",
          "Petrov@fsk.sdd,Петров Алексей Николаевич", "");
  Flag := MailSend("Тест", "Тестовое сообщение", "d:\test.txt");
  if Flag then
    writeln("Сообщение отправлено!");
  else
    writeln("Сообщение не отправлено!");
  endif;
  MailClose();
end.
```

procedure MailClose();

Закрывает текущее соединение с почтовым сервером по протоколу SMTP. Освобождает все занятые ресурсы.

Пример:

```
MailClose();
```

function MailGet_MessErr() : string;

Возвращает строку с сообщением об ошибке, возникшей при отправке почтового сообщения, или пустую строку, если ошибок нет.

Пример:

```
ErrStr := MailGet_MessErr();
```

Функции для работы с FTP

procedure FTPIni (Host: string; Port : int; User: string; Password: string);

Инициализирует параметры соединения с сервером FTP. *Host* – имя или IP адрес FTP сервера. *Port* – номер порта сервера FTP. *User* – имя учетной записи. *Password* – пароль учетной записи.

Примечание:

Процедура *FTPIni* должна быть вызвана (один раз), перед вызовом других функций для работы с FTP.

Пример:

```
FTPIni("10.240.3.110", 21, "anonymous", "alex@mail.ru");
```

function FTPSaveFile (PathFTP: string; NameFile: string): bool;

Записывает файл *NameFile* на сервер FTP по указанному пути *PathFTP*. Функция возвращает *true*, если запись файла прошла успешно, иначе возвращает *false*. Путь *PathFTP* указывается без конечного слэша.

Пример:

```
FTPSaveFile("Program\User", "d:\temp\alfa.doc");
```

function FTPLoadFile (NameFileFTP: string; NameFileLocal: string): bool;

Загружает файл *NameFileFTP* с сервера FTP и записывает загруженный файл, на локальный компьтер с именем *NameFileLocal*. Функция возвращает *true*, если загрузка файла прошла успешно, иначе возвращает *false*.

Пример:

```
FTPLoadFile("Program\User\alfa.doc", "d:\temp\alfa.doc");
```

function FTPListDirectory (DirFTP: string; FullForm: bool): array of string;

Возвращает строковый массив списка файлов и каталогов, находящихся в указанном каталоге *DirFTP* на сервере FTP. Если в указанном каталоге на сервере FTP файлов и каталогов нет, возвращается строковый массив нулевой длины. Параметр *FullForm* определяет форму выдачи данных. При *FullForm* = *true* полная форма выдачи данных, иначе краткая. Полная форма выдачи данных зависит от платформы, на которой установлен сервер FTP.

Платформа Windows

```
06-04-10  09:25AM     <DIR>      Book
06-15-06  10:27AM     <DIR>      docs
02-20-07  03:13PM     <DIR>      docum
```

```
12-30-10  10:40AM     <DIR>          Instr
06-30-09  02:04PM     <DIR>          Monitoring
01-31-11  11:20AM              28027 POOL.RET
03-05-10  01:58PM     <DIR>          Program
01-31-11  09:44AM     <DIR>          Tov
01-31-11  11:20AM              97960 user.net
11-24-10  02:00PM     <DIR>          Бланки
12-18-08  01:46PM             339248 Схема серверов ЦУМР.emf
```

Платформа Unix

```
dr-xr-xr-x  1 user    group        0 Feb  3 13:27 ! ВНИМАНИЕ! НОВОСТИ!
dr-xr-xr-x  1 user    group        0 Feb  4 09:40 !Обновление_компонентов
dr-xr-xr-x  1 user    group        0 Jan 24 18:51 .
dr-xr-xr-x  1 user    group        0 Jan 24 18:51 ..
dr-xr-xr-x  1 user    group        0 Jan 21 09:38 1
dr-xr-xr-x  1 user    group        0 Nov  8 11:20 2
dr-xr-xr-x  1 user    group        0 Sep 30 15:39 A-TRANS
dr-xr-xr-x  1 user    group        0 Dec  9 09:36 ARMENIA
dr-xr-xr-x  1 user    group        0 Nov  8  2009 ARMOPD(WIN)
dr-xr-xr-x  1 user    group        0 Jan 13 17:18 ARM_CD
dr-xr-xr-x  1 user    group        0 Nov 25 10:55 DB2_Грузовой _экспресс
```

Если установлен параметр выдачи краткой формы, функция возвращает строковый массив, состоящий только из списка файлов и каталогов входящих в заданный каталог на сервере FTP.

Пример:

```
var
  ListFile : array 1 of string;
  I : int;
begin
  FTPIni("10.240.3.110", 21, "anonymous", "alex@mail.ru");
  writeln("Получить список файлов");
  ListFile := FTPListDirectory("", true);
  if FTPGet_MessErr() <> "" then
    writeln("Ошибка. ", FTPGet_MessErr());
  endif;
  writeln("Список файлов:");
  if Len(ListFile) > 0 then
    for I := 0 to Len(ListFile) - 1
      writeln("ListFile[", I, "] = ", ListFile[I]);
    endfor;
  else
    writeln("  Нет файлов!");
  endif;
  FTPClose();
end;
```

function FTPListFile (DirFTP: string; Files: bool): array of string;

Возвращает строковый массив списка файлов или каталогов находящихся в указанном каталоге *DirFTP* на сервере FTP. Если в указанном каталоге на сервере FTP файлов или каталогов нет, возвращается строковый массив нулевой длины. Параметр *Files* определяет содержимое списка. При *Files* = *true* выдается список файлов, иначе список каталогов.

Примечание:

Если внутри имени файла или каталога присутствуют пробелы, то такая строка обрамляется в двойные кавычки. Каталог *DirFTP* должен иметь конечный символ “\”.

Пример:

```
var
  ListFile : array 1 of string;
  I : int;
begin
  FTPIni("10.240.3.110", 21, "anonymous", "alex@mail.ru");
  writeln("Получить список файлов");
  ListFile := FTPListFile("", true);
  if Len(ListFile) > 0 then
    writeln("Список файлов:");
    for I := 0 to Len(ListFile) - 1
      writeln("ListFile[", I, "] = ", ListFile[I]);
    endfor;
  else
    writeln("Нет файлов!");
  endif;
  FTPClose();
end;
```

procedure FTPClose();

Закрывает текущее соединение с сервером FTP. Освобождает все занятые ресурсы.

Пример:

```
FTPClose();
```

function FTPGet_MessErr() : string;

Возвращает строку с сообщением об ошибке, возникшей при работе с FTP сервером, или пустую строку, если ошибок нет.

Пример:

```
ErrStr := FTPGet_MessErr();
```

Функции для выполнения кода на языках программирования JScript и VBScript

procedure ScriptIni (Language: string; ScriptText: string);

Инициализирует параметры для последующего выполнения исходного кода на языках программирования JScript и VBScript. *Language* – строка определяющая язык программирования, может принимать значения “JScript” или “VBScript”. *ScriptText* – строка исходного текста скрипта.

Примечание:

Процедура *ScriptIni* должна быть вызвана (один раз), перед вызовом других функций.

Пример:

```
const
  JScriptStr =
<!--
function Main()
{
var
  fso,
  fldr;
  fso = new ActiveXObject("Scripting.FileSystemObject");
```

```
  fldr = fso.CreateFolder("C:\\MyTest");
  // возвращаем результат
  return "Выполнено";
}  // Main
-->;
…
ScriptIni("JScript", JScriptStr);
```

procedure ScriptIni1 (NameFile: string);

Инициализирует параметры для последующего выполнения исходного кода на языках программирования JScript и VBScript. *NameFile* – полное имя файла исходного текста скрипта. Расширение файла ".js" – определяет, что скрипт написан на яыке программирования "JScript", а расширение файла ".vbs", что скрипт написан на языке "VBScript".

Примечание:

Процедура *ScriptIni1* должна быть вызвана (один раз), перед вызовом других функций.

Пример:

```
ScriptIni1("d:\user\alfa\test.js");
```

function ScriptRun (NameFunc: string; TypeResult: string; [Par1; Par2; … ParN]): Value;

Выполняет именованную процедуру *NameFunc* из числа ранее добавленной функцией *ScriptIni*. *TypeResult* – тип возвращаемого результата, может принимать следующие значения: "*string*", "*int*", "*real*", "*bool*", "*array of string*", "*array of int*", "*array of real*", "*array of bool*", "*none*". *Par1, Par2, … ParN* – необязательные параметры передаваемые в вызываемую процедуру. Функция возвращает значение *Value* – результат, возвращаемый процедурой *NameFunc*.

Пример:

```
const
  JScriptStr =
<!--
function Main()
{
var
  fso,
  fldr;
  fso = new ActiveXObject("Scripting.FileSystemObject");
  fldr = fso.CreateFolder("C:\\MyTest");
  // возвращаем результат
  return "Выполнено";
}  // Main
-->;

var
  Str: string;
…
ScriptIni("JScript", JScriptStr);
Str := ScriptRun("Main", Type(Str));
writeln("Результат: ", Str);
ScriptClose();

// Результат: Выполнено
```

procedure ScriptReset();

Сбрасывает все установленные ранее параметры процедурой *ScriptIni()* или *ScriptIni1()*.

Пример:

```
ScriptReset();
```

procedure ScriptClose();

Сбрасывает все установленные ранее параметры процедурой *ScriptIni()* или *ScriptIni1()* и освобождает все занятые ресурсы.

Пример:

```
ScriptClose();
```

Разные процедуры и функции

function Get_NameLog(): string;

Возвращает строку полного имени файла протокола программы.

Пример:

```
NameLog := Get_NameLog();
```

function Get_MessErr(): string;

Сообщение об ошибке последней выполненной функции. Возвращает строку сообщения об ошибке или пустую строку, если ошибок нет.

Пример:

```
Err := Get_MessErr();
if not IsEmpty(Err) then
  writeln("Ошибка: ", Err);
endif;
```

Примечание:

Выдает сообщения об ошибке после исполнения следующих функций: *OpenFile*, *CloseFile*, *CopyFile*, *DeleteFile*, *MoveFile*, *ReplaceFile*, *AppendStrFile*, *WriteArrayFile*, *WriteStrFile*, *CreateDirectory*, *DeleteDirectory*, *MoveDirectory*, *SetAttrFile, SetCurrentDirectory, SetCreationTimeFile, SetLastAccessTimeFile, SetLastWriteTimeFile, CreateComObj, CreateServObj, Enumerator, MoveNext, CurrentEnum*.

procedure WaitBegin ();
procedure WaitEnd ();

WaitBegin() – вывод на консоль анимации ожидания. *WaitEnd()* – останов вывода на консоль анимации ожидания. Процедуры *WaitBegin()* и *WaitEnd()* используются только парами и предназначены для уведомления пользователя о длительных операциях. Процедуры выполняются, если директива интерпретатора *#on_wait* установлена, иначе они игнорируются.

Пример:

```
writeln("Копирование файла: ", NameFile);
WaitBegin();
```

```
CopyFile("c:\" + NameFile, "d:\" + NameFile, true);
WaitEnd();
```

Примечание:

Методы объекта *server*, функции для работы с *E-Mail*, функции дя работы с *FTP*, функции для выполнения кода на языках программирования *Jscript* и *VBScript* имеют встроенную поддержку вызова процедур *WaitBegin()* и *WaitEnd().*

procedure Beep (Freq: int; Dur: int);

Воспроизводит звуковой сигнал заданной частоты *Freq* и длительности *Dur* через динамик консоли. *Freq* - частота сигнала в диапазоне от 37 до 32767 Гц. *Dur* - длительность сигнала в миллисекундах.

Пример:

```
Beep(262, 500);
```

procedure Sleep (Time: int);

Блокирует текущее приложение на заданное количество миллисекунд *Time*.

Пример:

```
Sleep(1000);
```

function Run (FileName: string; Arg: string; WorkDir: string; WaitExit: bool; TimeWaitExit: int): bool;

Позволяет запустить внешнее приложение на локальном компьютере. Параметр *FileName* определяет полный путь к приложению и если нужно, с командной строкой запуска. Параметр Arg определяет аргументы командной строки запуска приложения, если он не используется, нужно передать пустую строку. Параметр *WorkDir* определяет рабочий каталог приложения, если он не используется, нужно передать пустую строку. Параметр *WaitExit* = *true* указывает, что надо ждать завершение приложения, *WaitExit* = *false* - нет. Параметр *TimeWaitExit* – определяет максимальное время ожидания завершения приложения в секундах. Функция возвращает *true,* когда время ожидания завершения приложения не истекло и код возврата приложения равен нулю, во всех других случаях возвращает *false*.

Пример:

```
if (not Run("osql.exe",
            " -U " + aIniServ[UserDB] +
            " -P " + aIniServ[PasswordDB] +
            " -S " + aIniServ[ServerName] +
            " -d " + aIniServ[DBName] +
            " -i " + aIniServ[InputSQLName] +
            " -o " + aIniServ[OutputSQLName], "", false, 0)) then
  writeln("Ошибка запуска процесса: osql.exe");
  exit;
endif;
```

procedure RunError (ErrorCode: int; ErrorMess: string);

Останавливает выполнение программы, генерируя в текущей инструкции ошибку времени выполнения с заданным кодом *ErrorCode* и сообщением *ErrorMess*. Значение *ErrorCode* должно быть в диапазоне от 1 до 255.

Пример:

```
RunError(2, "Аварийное завершение программы");
```

function ParamStr (NumParam: int): string;

Возвращает заданный параметр командной строки приложения. Аргумент *NumParam* определяет номер параметра командной строки.

Пример:

```
i := 1;
// анализ параметров командной строки
while (i <= LengthParamStr())
  sPar := ParamStr(i);
  if IsEmpty(sPar) then
    exit;
  endif;

  …
endwhile;
```

function LengthParamStr(): int;

Возвращает число параметров командной строки приложения.

Пример:

```
NumParam := LengthParamStr();
```

function Type (Value): string;

Возвращает строку типа передаваемого аргумента *Value*. В качестве аргумента *Value* может выступать любая переменная, выражение или конкретное значение. Возвращаемое значение представляет собой строку типа и может принимать следующие значения: “*int*”, “*real*”, “*string*”, ”*bool*”, “*array of int*”, “*array of real*”, “*array of string*”, “*array of bool*”, “ *array of file*”, “*array of server*”, “*array of comobj*”, “*array of proc*“, “*array of func*“, “ *file*”, “*server*”,“*comobj*”, “*proc*“, “*func*“.

Пример:

```
program TestType;
var
  ArrI : array 1 of int;
  ArrR : array 1 of real;
  ArrS : array 1 of string;
  ArrB : array 1 of bool;
  Str : string;
  I : int;
  R : real;
  F : file;
  @Serv : server;
  COM : comobj;
  F1 : func;
  P : proc;
begin
  writeln("Type 5.2 = ", Type(5.2));
  writeln("Type 3 = ", Type(3));
  writeln("Type true = ", Type(true));
  writeln("Type Строка = ", Type("Строка"));
  writeln("Type ArrI = ", Type(ArrI));
  writeln("Type ArrR = ", Type(ArrR));
  writeln("Type ArrS = ", Type(ArrS));
```

```
  writeln("Type ArrB = ", Type(ArrB));
  writeln("Type Str = ", Type(Str));
  writeln("Type I = ", Type(I));
  writeln("Type R = ", Type(R));
  writeln("Type F = ", Type(F));
  writeln("Type @Serv = ", Type(@Serv));
  writeln("Type COM = ", Type(@Serv));
  writeln("Type F1 = ", Type(F1));
  writeln("Type P = ", Type(P));
end.

// Type 5.2 = real
// Type 3 = int
// Type true = bool
// Type Строка = string
// Type ArrI = array of int
// Type ArrR = array of real
// Type ArrS = array of string
// Type ArrB = array of bool
// Type Str = string
// Type I = int
// Type R = real
// Type F = file
// Type @Serv = server
// Type COM = comobj
// Type F1 = func
// Type P = proc
```

function IsNull(Var): bool;

Указывает, имеет ли передаваемый аргумент *Var* значение *nil*. В качестве аргумента *Var* может выступать любая переменная, выражение или конкретное значение. Если значение *Var* равно *nil* функция возвращает *true*, иначе *false*.

Пример:

```
var
  COM : comobj;
  Flag : bool;

Flag := IsNull(COM);
```

Реализация языка программирования – интерпретатор *Alfa*

Для реализации языка программирования *Alfa* был разработан интерпретатор. Он написан на языке *C#* в инструментальной среде *Microsoft Visual Studio 2010* с применением компилятора компиляторов *Coco/R*. Интерпретатор реализован, как консольное приложение. Интерпретатор позволяет выполнять файлы исходного текста программы в различных кодировках - *Win-1251*, *UTF8*, а также файлы программ в зашифрованном виде. Шифрование файлов исходного текста программ осуществляется с помощью специальной утилиты *CryptFile*, которая входит в состав дистрибутива интерпретатора.

Системные требования

Для функционирования интерпретатора требуется наличие операционной системы *Windows* (*Windows 2000, Windows XP, Windows 2003* или выше) с установленной платформой *.NET Framework v4.0* или выше.

Установка интерпретатора

Для установки интерпретатора, создайте временный каталог и скопируйте установочные программы в него (*Setup.exe*, *Setup.msi*). Запустите программу *Setup.exe*. Следуйте инструкциям выдаваемых программой установки. По умолчанию интерпретатор *Alfa* устанавливается по следующему пути: *c:\Program Files\UAF\Alfa*. После установки интерпретатора, в программе проводника *Windows*, файлы исходного текста программы будут отображаться в следующем виде:

- иконка файла исходного текста программы в кодировке *Win-1251*;

- иконка файла исходного текста программы в кодировке *UTF-8*;

- иконка *зашифрованного* файла исходного текста программы в кодировке *UTF-8*.

Двойной щелчек мыши в проводнике *Windows* на иконке файла исходного текста, будет приводить к немедленному вызову интерпретатора и выполнению исходного текста программы.

Командная строка запуска интерпретатора

Командная строка запуска:

<Path>Alfa.exe <PathProg>NameProg.alf |
<PathProg>NameProg.alfa |
<PathProg>NameProg.alfc [-ck:<CryptKey>]

где

Path – путь доступа к интерпретатору;
Alfa.exe – выполняемый файл интерпретатора;
PathProg – путь доступа к файлу исходного текста программы;
NameProg.alf – имя файла исходного текста программы с расширением "*.alf*" в кодировке *Win-1251*;
NameProg.alfa – имя файла исходного текста программы с расширением "*.alfa*" в кодировке *UTF8*;
NameProg.alfc – имя файла исходного текста программы с расширением "*.alfc*" в кодировке *UTF8*, в *зашифрованом* виде;
<CryptKey> - ключ шифрования (8 символов), если он не задан, интерпретатор использует ключ шифрования по умолчанию, такой же как в утилите *CryptFile*.

Директивы интерпретатора

Каждая директива располагается на отдельной строке до ключевого слова *program* и не заканчивается точкой с запятой, в отличие от операторов языка. В одной строке с директивой может располагаться только комментарий вида //. Перечень и краткое описание директив приведено ниже в табл. 1.

Наименование	Описание
#on_wait, *#off_wait*	Установка или отмена установки вывода анимации ожидания на консоль. По умолчанию установлено *#on_wait*. Осуществляется вывод анимации ожидания.
#on_log, *#off_log*	Задание или отмена задания, записи консольного вывода в файл протокола. Файл протокола имеет такое же имя как имя файла исходного текста программы, с расширением "*.log*". По умолчанию установлено *#on_log*. Вывод на консоль записывается в файл протокола.
#on_UTF8log, #off_UTF8log	Установка или отмена кодировки файла протокола в UTF8. По умолчанию установлено *#off_UTF8log*. Файл протокола записывается в кодировке WIN-1251.
#on_keywait, #off_keywait	Установка или отмена ожидания нажатия клавиши, для закрытия консоли приложения, при выводе сообщения о произошедшем исключении. По умолчанию установлено *#on_keywait*. Во время обработки исключения, осуществляется вывод сообщения об ошибке на консоль. Консоль приложения закрывается только после нажатия любой клавиши пользователем.

Таблица 1. Директивы интерпретатора

Использование директив интерпретатора и результаты приведены в "*Примеры программ на языке Alfa*"

Обработка ошибок интерпретатором

При возникновении любой ошибки синтаксической, семантической, логической и т.д. интерпретатор завершает выполнение программы и выдает сообщение об ошибке на консоль или в файл протокола, если установлена соответствующая директива в исходном тексте программы. Формат выдачи сообщения об ошибке следующий:

Error, line <*номер_строки*> **col** <*номер_позиции*>**:** <*сообщение_об_ошибке*>
-> <*ошибочная_строка_исходного_текста_программы*>
^
| указатель на ошибочную позицию в строке

Пример выдачи сообщения об ошибке при выполнении приложения приведен на Рис. 1.

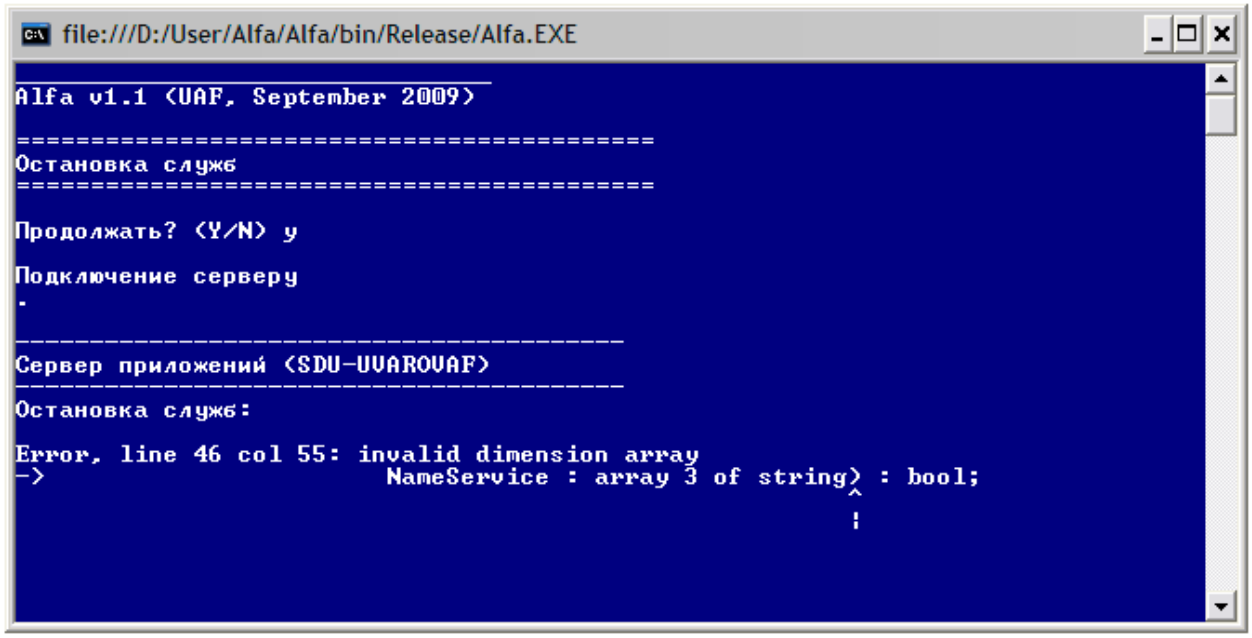

Рис. 1. Вывод ошибочной строки исходного текста программы

Прекращение работы интерпретатора по команде пользователя

Работу интерпретатора можно прекратить в любой момент нажатием комбинации клавиш CTRL+C. При этом завершается выполнение программы, с выдачей сообщения на консоль и в файл протокола - “Внимание! Прекращение выполнения программы по команде CTRL+C ”. См. Рис.2.

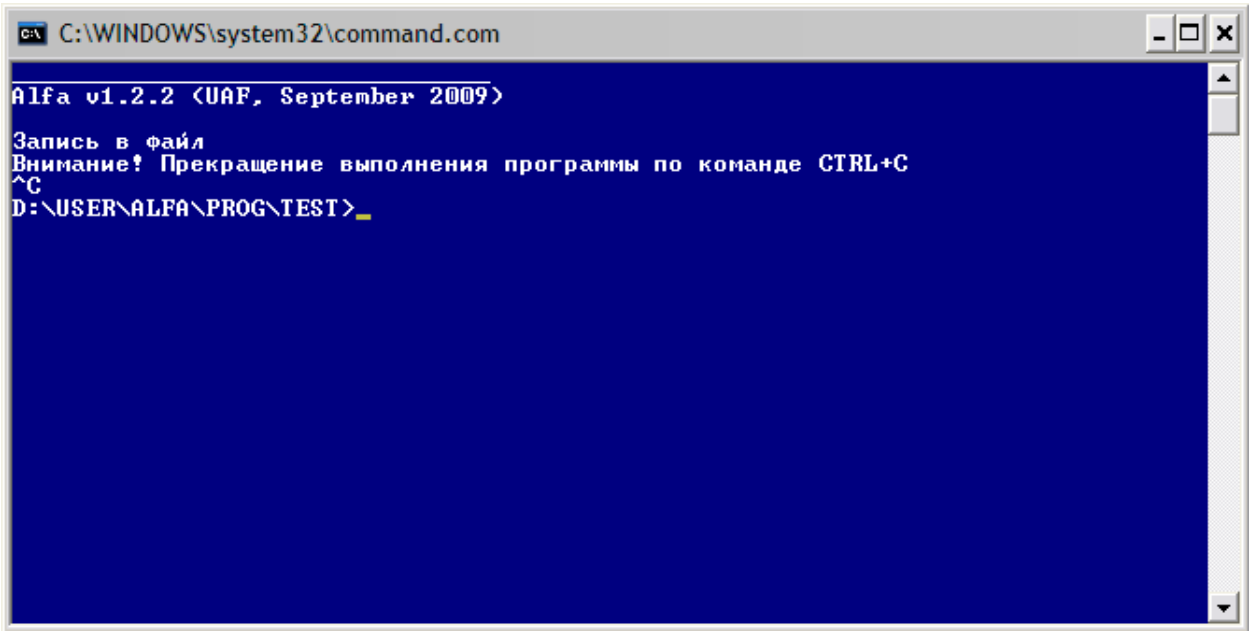

Рис. 2. Прекращение выполнения программы

Утилита шифрования исходных текстов программ *CryptFile*

Утилита *CryptFile* позволяет шифровать или дешифрировать файлы исходного текста программ представленных в кодировке *UTF8*. Шифрование файлов производится с помощью симметричного

алгоритма *DES (Data Encryption Standart),* 64-битным ключем. Режим шифрования или дешифрования определяется расширением файла исходного текста программы. Если ключ шифрования не задан, используется ключ шифрования по умолчанию.

Командная строка запуска утилиты:

<Path>CryptFile.exe <PathProg>NameProg.alf [<CryptKey>] |
<PathProg>NameProg.alfa [<CryptKey>] |
<PathProg>NameProg.alfc [<CryptKey>]

где

Path – путь доступа к интерпретатору;
CryptFile.exe – выполняемый файл утилиты;
PathProg – путь доступа к файлу исходного текста программы;
NameProg.alf - имя файла исходного текста программы с расширением "*.alf"* в кодировке WIN-1251;
NameProg.alfa – имя файла исходного текста программы с расширением "*.alfa"* в кодировке UTF-8;
NameProg.alfc – имя файла исходного текста программы с расширением "*.alfc"* в кодировке UTF-8, в зашифрованом виде;
<CryptKey> - ключ шифрования (8 символов).

Перечень входных и выходных файлов программы CryptFile приведен в табл. 2.

№	Входной файл	Выходной файл	Действие
1	*NameProg.alf* – файл в кодировке WIN-1251	*NameProg.alfa* – файл в кодировке UTF-8 *NameProg.alfc* – зашифрованный файл в кодировке UTF-8	Создание файла в кодировке UTF-8. Шифрование созданного файла.
2	*NameProg.alfa* - файл в кодировке UTF-8	*NameProg.alfc* – зашифрованный файл в кодировке UTF-8	Шифрование входного файла.
3	*NameProg.alfc* - зашифрованный файл в кодировке UTF-8	*NameProg.alfa* – дешифрированный файл в кодировке UTF-8	Дешифрирование входного файла.

Таблица 2. Входные и выходные файлы программы CryptFile

Программа *CryptFile* запускается из командной строки, в качестве её обязательного параметра указывается имя файла исходного файла программы. Если в качестве параметра указан файл с расширением ".alf" то создается файл в кодировке UTF-8 с расширением ".alfa", созданный файл шифруется указанным ключем или ключем по умолчанию и записывается в файл с расширением

“.alfc”. Далее, если в качестве параметра указан файл с расширением “*.alfa*” то, файл шифруется указанным ключем или ключем по умолчанию и записывается файл с тем же самым именем, но с расширением “*.alfc*”. И наконец, если указан файл с расширением “*.alfc*” то, файл дешифрируется и записывается с тем же именем, но с расширением “*.alfa*” и в кодировке UTF-8.

Примеры программ на языке *Alfa*

Задача 1

Пусть у нас имеется комплекс серверов. Нужно удаленно, с любого рабочего места, как можно быстро, проверить состояние серверов, а так же запущены ли необходимые службы на них. Если службы, по какой либо причине не запущены (“упали”), то запустить их (“поднять”).

Ниже представлен исходный код на языке программирования *Alfa*, для решения поставленной задачи.

```
#on_wait
#on_log

program ControlServers;

var
  I : int;
  // службы на сервере телеобработки
  Main_TO : array 4 of string =
    (
       "Телеобработка",
       "Монитор взаимодействия с Этран",
       "Модуль взаимодействия с АС ЭТД",
       "StatClient"
    );
  // службы на основном сервере приложений
  Main_SP : array 4 of string =
    (
       "Планировщик СП основной",
       "Отчетность АСУСТ (основная)",
       "Сервер подсистемы КСАРМ ВЧД",
       "StatClient(основной)"
    );
  // службы на резервном сервере приложений
  Res_SP : array 4 of string =
    (
       "Планировщик СП резервный";
       "Отчетность АСУСТ (резервная)";
       "Сервер подсистемы КСАРМ ВЧД";
       "StatClient(резервный)";
    );

declare @CUMR04_TO : server
  Title = "Сервер телеобработки";
  Host = "1.2.2.15";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

declare @CUMR04A : server
  Title = "Основной сервер приложений";
  Host = "1.2.2.11";
  User = "adm";
  Password = "pass";
```

```
  IntervalWait = 60;
  WaitReboot = 360;
end;

declare @CUMR05A : server
  Title = "Резервный сервер приложений";
  Host = "1.2.2.21";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

procedure OutMess (Mess : string; NameService : string);
begin
  if not(Mess = "") then
    writeln("Ошибка: ", Mess);
  else
    writeln("Служба: " + "'" + NameService + "'" + " - запущена!");
  endif;
end;

procedure ServiceControl (var @Serv : server; NameService : string);
begin
  @Serv->ControlService(NameService);
  OutMess(@Serv->Get_MessErr(), NameService);
end;

begin

  writeln("========================================");
  writeln("Контроль состояния служб на серверах ЦУМР");
  writeln("========================================");

  // подключение к серверу телеобработки
  writeln("Подключение к серверу телеобработки");
  if not @CUMR04_TO->Connect() then
    writeln("Ошибка: ", @CUMR04_TO->Get_MessErr());
    exit;
  endif;

  // подключение к основному серверу приложений
  writeln("Подключение к основному серверу приложений");
  if not @CUMR04A->Connect() then
    writeln("Ошибка: ", @CUMR04A->Get_MessErr());
    exit;
  endif;

  // подключение к резервному серверу приложений
  writeln("Подключение к резервному серверу приложений");
  if not @CUMR05A->Connect() then
    writeln("Ошибка: ", @CUMR05A->Get_MessErr());
    exit;
  endif;

  writeln("----------------------------------------");
  writeln(@CUMR04_TO.Title + " (" + @CUMR04_TO.Host + ")");

  // проверка запуска служб на сервере телеобработки
  for I := 0 to Len(Main_TO) - 1
    ServiceControl(@CUMR04_TO, Main_TO[I]);
  endfor;

  writeln("----------------------------------------");
  writeln(@CUMR04A.Title + " (" + @CUMR04A.Host + ")");

  // проверка запуска служб на основном сервере приложений
```

```
  for I := 0 to Len(Main_SP) - 1
    ServiceControl(@CUMR04A, Main_SP[I]);
  endfor;

  writeln("----------------------------------------");
  writeln(@CUMR05A.Title + " (" + @CUMR05A.Host + ")");

  // проверка запуска служб резервном сервере приложений
  for I := 0 to Len(Res_SP) - 1
    ServiceControl(@CUMR05A, Res_SP[I]);
  endfor;

  writeln("----------------------------------------");

  // запрос на завершение работы
  readln();

end.
```

Ниже представлен файл протокола программы ControlServers.

```
************************************************************
ControlServers
  Дата: 22.10.2009  Время: 16:03:11
  Время выполнения: 00:11.79
************************************************************

========================================
Контроль состояния служб на серверах ЦУМР
========================================
Подключение к серверу телеобработки
Подключение к основному серверу приложений
Подключение к резервному серверу приложений
----------------------------------------
Сервер телеобработки (1.2.2.15)
Служба: 'Телеобработка' - запущена!
Служба: 'Монитор взаимодействия с Этран' - запущена!
Служба: 'Модуль взаимодействия с АС ЭТД' - запущена!
Служба: 'StatClient' - запущена!
----------------------------------------
Основной сервер приложений (1.2.2.11)
Служба: 'Планировщик СП основной' - запущена!
Служба: 'Отчетность АСУСТ (основная)' - запущена!
Служба: 'Сервер подсистемы КСАРМ ВЧД' - запущена!
Служба: 'StatClient(основной)' - запущена!
----------------------------------------
Резервный сервер приложений (1.2.2.21)
Служба: 'Планировщик СП резервный' - запущена!
Служба: 'Отчетность АСУСТ (резервная)' - запущена!
Служба: 'Сервер подсистемы КСАРМ ВЧД' - запущена!
Служба: 'StatClient(резервный)' - запущена!
----------------------------------------
```

Задача 2

Пусть имеются два сервера, основной и резервный (Main и Reserve соответственно). См. Рис. 3.

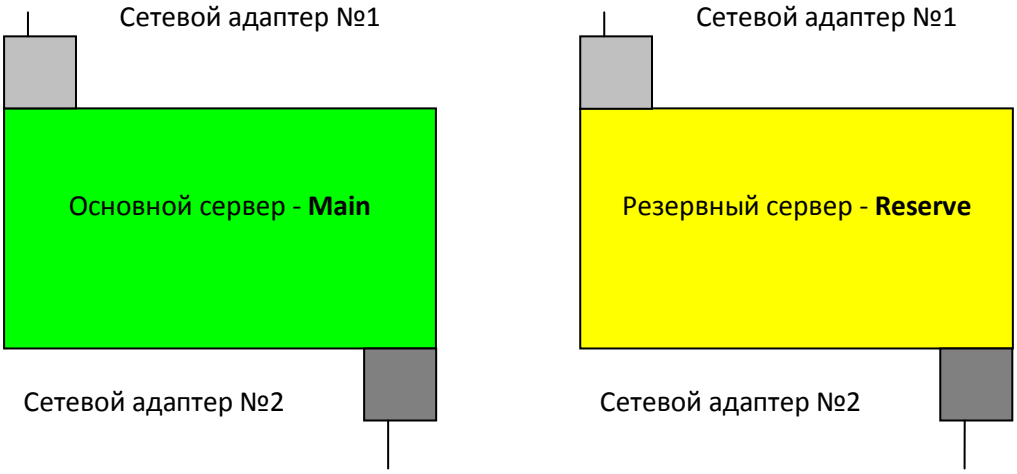

Рис. 3.

На каждом из серверов установлена одинаковая операционная система и программное обеспечение. Программно поддерживается репликация данных на серверах. Серверы работают в режиме реального времени. Серверы имеют два сетевых адаптера, на которых установлены статические IP – адреса, причем сетевые адаптеры №1 предназначены для клиентского трафика, а сетевые адаптеры №2 только для целей внешнего управления серверами. Все внешние данные поступают на сетевой адаптер №1 основного сервера Main, а его IP адрес является основным для внешнего потока данных.

Нужно программно, удаленно, с любого рабочего места, по возможности корректно завершить работу прикладного программного обеспечения на основном сервере Main и резервном сервере Reserve. Поменять IP основной адрес для внешнего потока данных с основного сервера на резервный. Корректно запустить необходимое прикладное программное обеспечение. Все операции должны осуществляться по возможности без воздействия человека. Время простоя серверов (время без приема и обработки внешних данных) должно быть минимальным.

Ниже представлен исходный код на языке программирования *Alfa*, для решения поставленной задачи.

```
//*********************************************************
// Переход с основного на резервный сервер приложений  ЦУМР
//*********************************************************

program Change_SP;

const
  // временный IP сервера приложений
  TempIP_SP = "1.2.2.9";
  // IP основного сервера приложений
  IP_SP = "1.2.2.11";
  // IP резервного сервера приложений
  IP_SP_Res = "1.2.2.21";
  // маска подсети
  SubNetMask = "255.255.252.0";
  // отображаемое имя службы КСАРМ ВЧД
  Tov_Service = "Сервер подсистемы КСАРМ ВЧД";
```

```
  // основной процесс КСАРМ ВЧД
  Tov_Process = "nettov.exe";
  Title_SP = "Сервер приложений";

var
  Name_Main : string;
  Main_Service : array 4 of string =
    (
       "Планировщик СП основной",
       "Отчетность АСУСТ (основная)",
       "Сервер подсистемы КСАРМ ВЧД",
       "StatClient(основной)"
    );
  Res_Service : array 4 of string =
    (
       "Планировщик СП резервный",
       "Отчетность АСУСТ (резервная)",
       "Сервер подсистемы КСАРМ ВЧД",
       "StatClient(резервный)"
    );
  @Res : server;

declare @Main : server;
  Title = "Сервер приложений основной";
  Host = "1.2.2.11";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

declare @CUMR04A : server
  Title = "Сервер приложений";
  Host = "1.2.2.10";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

declare @CUMR05A : server
  Title = "Сервер приложений";
  Host = "1.2.2.20";
  User = "adm";
  Password = "pass";
  IntervalWait = 60;
  WaitReboot = 360;
end;

procedure OutMess (Mess : string; NameService : string; Start : bool);
begin
  if not(Mess = "") then
    writeln("  Ошибка: ", Mess);
  else
    if Start then
      writeln("  Служба: " + "'" + NameService + "'" + " - запущена!");
    else
      writeln("  Служба: " + "'" + NameService + "'" + " - остановлена!");
    endif
  endif;
end;

// запустить-остановить заданые службы на сервере
function StartStopService (var @Serv: server;
                           NameService : array of string;
                           Start : bool) : bool;
var
```

```
  I : int;
begin
  result := true;
  for I := 0 to Len(NameService) - 1
    if Start then
      result := @Serv->StartService(NameService[I]);
    else
      result := @Serv->StopService(NameService[I]);
      if NameService[I] = Tov_Service then
        // ожидаем завершение основного процесса КСАРМ ВЧД 60 сек
        // из за неправильной работы службы
        result := (@Serv->WaitForProcess(Tov_Process, 60)) then
      endif;
    endif;
    OutMess(@Serv->Get_MessErr(), NameService[I], Start);
    if not result then
      exit;
    endif;
  endfor;
end;

// замена IP адреса на сервере
function RepIP (var @Serv : server; OldIP : string;
                NewIP : string; SubNetMask : string) : bool;
var
  Status : string;
begin
  result := true;
  Status := @Serv->ReplaceIP(OldIP, NewIP, SubNetMask);
  switch Status
    case "Reboot":
      writeln("Перезагрузка сервера: " + @Serv.Title);
      if not @Serv->Reboot() then
        writeln("Ошибка при перезагрузе сервера: " + @Serv.Title);
        result := false;
      endif;
    endcase;
    case "Error":
      writeln("Ошибка при замене IP на сервере: " + @Serv.Title + "!");
      writeln("Требуется срочное вмешательство и выполнение перевода вручную!");
      result := false;
    endcase;
  endswitch;
end;

// вывод текущей конфигурации серверов
procedure Config (var @Main : server; var @Res : server);
begin
  writeln("");
  writeln("-------------------------------------------------");
  writeln("Текущая конфигурация серверов:");
  writeln("  " + @Main.Title);
  writeln("  " + @Res.Title);
  writeln("-------------------------------------------------");
  writeln("");
end;

// подтверждение от пользователя (Y/N)
function InputYesOrNo (Mess : string) : bool;
var
  Key : string;
begin
  result := false;
  loop
    readln(Mess, Key);
    Key := Upper(Key);
    if (Key = "Y") or (Key = "N") then
```

```
      if (Key = "Y") then
        result := true;
      else
        result := false;
      endif;
      exit;
    endif;
  endloop;
end;

begin
  writeln("=============================================");
  writeln("Переход с основного на резервный сервер СП ЦУМР");
  writeln("=============================================");

  // определение текущей конфигурации серверов
  writeln("");
  writeln("Определение текущей конфигурации серверов");
  writeln("");
  if not @Main->Connect() then
    writeln("Ошибка: ", @CUMR04A->Get_MessErr());
    exit;
  endif;
  Name_Main := Upper(@Main->NameComputer());
  if (Name_Main = "CUMR04A") then
    @Main := @CUMR04A;
    @Main.Title := Title_SP + " основной (CUMR04A)";
    @Res := @CUMR05A;
    @Res.Title := Title_SP + " резервный (CUMR05A)";
  else
    @Main := @CUMR05A;
    @Main.Title := Title_SP + " основной (CUMR05A)";
    @Res := @CUMR04A;
    @Res.Title := Title_SP + " резервный (CUMR04A)";
  endif;
  // вывод текущей конфигурации
  Config(@Main, @Res);

  // запрос на продолжение выполнения программы
  writeln("");
  if not InputYesOrNo("Продолжать? (Y/N)") then
    exit;
  endif;
  writeln("");

  // подключение к основному серверу приложений
  writeln("Подключение к основному серверу приложений");
  if not @Main->Connect() then
    writeln("Ошибка: ", @Main->Get_MessErr());
    exit;
  endif;

  // подключение к резервному серверу приложений
  writeln("Подключение к резервному серверу приложений");
  if not @Res->Connect() then
    writeln("Ошибка: ", @Res->Get_MessErr());
    exit;
  endif;

  // останов служб на основном сервере приложений
  if not StartStop(@Main, Main_Service, false) then
    exit;
  endif;

  // останов служб на резервном сервере приложений
  if not StartStop(@Res, Res_Service, false) then
    exit;
```

```
  endif;

  // замена IP адреса на сервере приложений на временный IP
  writeln("Замена IP на основном сервере приложений на временный: " + TempIP_SP);
  if not RepIP(@Main, IP_SP, TempIP_SP, SubNetMask) then
    exit;
  endif;

  // замена IP адреса на сервере приложений
  writeln("Замена IP на резервном сервере приложений на: " + IP_SP);
  if not RepIP(@Res, IP_SP_Res, IP_SP, SubNetMask) then
    exit;
  endif;

  // запуск служб на резервном сервере приложений (он теперь стал основным)
  // копирование файлов настроек сервера подсистемы КСАРМ ВЧД
  @Res->RunProcess("c:\cittrans\tov\osn\scl_05a_Reserve_to_Main.bat", "");
  // останов служб на резервном сервере приложений
  if not StartStop(@Res, Res_Service, true) then
    exit;
  endif;

  // замена IP адреса на сервере приложений
  writeln("Замена IP на основном сервере приложений на: " + IP_SP_Res);
  if not RepIP(@Main, TempIP_SP, IP_SP_Res, SubNetMask) then
    exit;
  endif;

  // запуск служб на основном сервере приложений (он теперь стал резервным)
  // копирование файлов настроек сервера подсистемы КСАРМ ВЧД
  @Main->RunProcess("c:\cittrans\tov\rez\scl_04a_Main_to_Reserve.bat", "");
  if not StartStop(@Main, Main_Service, true) then
    exit;
  endif;

  // определение текущей конфигурации
  Name_Main := Upper(@Res->NameComputer());
  if (Name_Main = "CUMR04A") then
    @Res.Title := Title_SP + " основной (CUMR04A)";
    @Main.Title := Title_SP + " резервный (CUMR05A)";
  else
    @Res.Title := Title_SP + " основной (CUMR05A)";
    @Main.Title := Title_SP + " резервный (CUMR04A)";
  endif;
  // вывод текущей конфигурации
  Config(@Res, @Main);
  writeln("Выполнение команд завершено успешно!");
  // запрос на завершение работы
  readln();
end.
```

Задача 3

Пусть у нас имеется сервер, на котором установлена СУБД MSSQL. Прикладное ПО, которое установлено на сервере, порождает так называемые архивные БД. Нужно периодически удаленно, с любого рабочего места, перемещать файлы БД с текущего местоположения, в заданный каталог на заданный локальный диск сервера.

Выполнение перемещения БД для полного контроля осуществляется в три этапа: отключение БД, физическое перемещение, подключение БД. Для этого, динамически формируются SQL запросы, которые записываются во входной файл сценария. Вызывается в фоновом режиме процесс OSQL на локальном компьютере с передачей всех необходимых

параметров (адрес хоста, имя учетной записи и пароль, имя входного файла с выполняемым SQL запросом, имя выходного файла). Фоновый процесс OSQL осуществляет удаленное соединение с БД сервера и выполнение SQL запроса. После завершения своей работы, процесс OSQL записывает результат в указанный выходной файл. Программа MoveDB контролирует работу процесса OSQL, анализируя выходной файл. Если не было ошибок, корректирует файл списка БД, удаляя из списка обработанную БД, иначе выдает сообщение об ошибке и завершает свою работу. Все действия программы выводятся на экран и записываются в файл протокола. Файл протокола создается в том же каталоге, где находится файлы инициализации.

Командная строка запуска:

alfa <путь>MoveDB.alf /f<имя_файла_инициализации>.ini

Файл инициализации должен иметь расширение .ini. Имя файла должно указываться с полным путем доступа.

Для работы необходимы следующие файлы:

- **MoveDB.alf** – файл программы;
- **MoveDB.ini –** файл инициализации программы перемещения архивов БД;
- **OSQL.exe** – утилита выполнения запросов SQL на удаленном сервере;

Файл инициализации имеет следующую структуру:

```
[Server]
ServerName=<Имя_Сервера>
DBName=<Имя_БД>
UserDB=<Пользователь_БД>
PasswordDB=<Пароль_БД>
PathDBSource=<Путь_к_источнику_БД>
PathDBDestin=<Путь_к_приемнику_БД>
[Files]
InputSQLName=<Имя_входного_файла>
OutputSQLName=<Имя_выходного_файла>
ListArcName=<Имя_файла_списка_БД>
```

где:

Имя_Сервера – имя сервера БД или его IP адрес;
Имя_БД – имя БД;
Пользователь_БД – имя пользователя БД;
Пароль_БД – пароль пользователя БД;
Путь_к_источнику_БД – путь к текущему месторасположению файлов БД;
Путь_к_приемнику_БД – путь к новому месторасположению файлов БД.

Примечание:

Пути к файлам указывать без конечного обратного слеша.

Имя_входного_файла – имя файла SQL сценария;
Имя_выходного_файла – имя выходного файла результата SQL сценария;
Имя_файла_списка_БД – имя текстового файла списка БД.

Примечание:

Список БД в файле должен быть представлен одной текстовой строкой состоящей, из имен БД, разделенных между собой символов запятая. Имена файлов БД должны быть без файлового расширения.

Пример файла инициализации

```
;*************************************************
;-------------------- MoveDB -------------------------
; Файл инициализации программы перемещения архивных БД
;*************************************************
; Секция определения сервера БД
[Server]
; Имя сервера
ServerName=ASUS
; Имя БД
DBName=AlertsLog
; Имя пользователя БД
UserDB=UAF
; Пароль пользователя БД
PasswordDB=admin
; Путь к БД источника
PathDBSource=d:\Program Files\Microsoft SQL Server\MSSQL\data
; Путь к БД приемника
PathDBDestin=d:\TestDB
; Секция определения рабочих файлов
[Files]
; Имя файла сценария
InputSQLName=DBScript.sql
; Имя выходного файла результа выполнения сценария
OutputSQLName=ErrDBScript.txt
; Имя файла списка БД
ListArcName=ListArc.txt
```

Пример файла списка БД

```
Test
```

Ниже представлен исходный код на языке программирования *Alfa*, для решения поставленной задачи.

```
#on_wait
#on_log

program MoveDB;
const
  // заголовок программы
  sTitle = "MoveDB v1.0 UAF - Перемещение БД";
  // определение индексов массива настройки параметров сервера
  // имя сервера
  ServerName = 0;
  // имя БД
  DBName = 1;
  // имя пользователя БД
```

```
  UserDB = 2;
  // пароль пользователя БД
  PasswordDB = 3;
  // путь к БД источника
  PathDBSource = 4;
  // путь к БД приемника
  PathDBDestin = 5;
  // имя файла сценария SQL
  InputSQLName = 6;
  // имя выходного файла результата выполнения сценария SQL
  OutputSQLName = 7;
  // имя файла списка БД
  ListArcName = 8;
  // размерность массива настроек параметров сервера
  LenIniArr = 9;
var
  nI : int;
  nLen : int;
  // имя LOG файла
  sNameFileLog : string;
  // имя INI файла
  sNameFileIni : string;
  // строка SQL запроса
  sSQL : string;
  // текущий список имен БД
  sArc : string;
  // текущее имя архива
  sNameArc : string;
  // массив настроек
  aIniServ : array LenIniArr of string;
  // об'явление локального компьютера
  @Local : server

  //************************************************************
  //* Function: ControlComStr
  //* Notes: Проверка командной строки сценария *
  //*

  function ControlComStr (var sParStr: string) : bool;
  var
    i : int;
    sPar : string;
    Err : int;
  begin
    Err := 0;
    i := 1;
    // анализ входной строки
    while (i <= LengthParamStr())
      sPar := ParamStr(i);
      if IsEmpty(sPar) then
        exit;
      endif;
      i := i + 1;
      if (SubStr(sPar, 0, 2) = "/f") then
        if (sParStr = "") then
          sParStr := SubStr(sPar, 2, Length(sPar) - 2);
        else
          writeln("Ошибка в командной строке, дубликат ключа - ", sPar);
          Err := Err + 1;
        endif;
      else
        if (SubStr(sPar, 0, 2) = "/h") then
          writeln("Командная строка запуска :");
          writeln("  alfa MoveDB.alf [/f<Имя_файла_инициализации>] [/h]");
        endif;
        Err := Err + 1;
        result := false;
```

```
      exit;
    endif;
  endwhile;
  if ( Err > 0 ) then
    result := false;
    exit;
  else
    if (IsEmpty(sParStr)) then
      sParStr := ParamStr(0);
    endif;
    sParStr := ChangeExtension(sParStr, ".ini");
  endif;
  result := true;
end;

//*** End of ControlComStr ***********

//******************************
//* Function: ReadIniFile
//* Notes: Чтение файла инициализации
//*

function ReadIniFile(sNameFile: string; var aIniServ: array of string) : bool;
var
  sPath : string;
  I : int;
begin
  result := true;
  // инициализировать об'ект файла инициализации
  IniInit(sNameFile);
  // определить текущий каталог
  sPath := GetDirectoryName(sNameFile);
  SetCurrentDirectory(sPath);
  // считать значение ServerName сервера
  aIniServ[ServerName] := IniReadString("Server", "ServerName", "");
  if (IsEmpty(aIniServ[ServerName])) then
    writeln("  [Server]-->ServerName - неопределенно значение!");
    result := false;
  endif;
  // считать значение DBName сервера
  aIniServ[DBName] := IniReadString("Server", "DBName", "");
  if (aIniServ[DBName] = "") then
    writeln("  [Server]-->DBName - неопределенно значение!");
    result := false;
  endif;
  // считать значение User
  aIniServ[UserDB] := IniReadString("Server", "UserDB", "");
  if (IsEmpty(aIniServ[UserDB])) then
    writeln("  [Server]-->UserDB - неопределенно значение!");
    result := false;
  endif;
  // считать значение Password
  aIniServ[PasswordDB] := IniReadString("Server", "PasswordDB", "");
  if (IsEmpty(aIniServ[PasswordDB])) then
    writeln("  [Server]-->PasswordDB - неопределенно значение!");
    result := false;
  endif;
  // считать значение PathDBSource
  aIniServ[PathDBSource] := IniReadString("Server", "PathDBSource", "");
  if (IsEmpty(aIniServ[PathDBSource])) then
    writeln("  [Server]-->PathDBSource - неопределенно значение!");
    result := false;
  endif;
  // считать значение PathDBDestin
  aIniServ[PathDBDestin] := IniReadString("Server", "PathDBDestin", "");
  if (IsEmpty(aIniServ[PathDBDestin])) then
    writeln("  [Server]-->PathDBDestin - неопределенно значение!");
```

```
    result := false;
  endif;
  // считать значение InputSQLName
  aIniServ[InputSQLName] := IniReadString("Files", "InputSQLName", "");
  if (IsEmpty(aIniServ[InputSQLName])) then
    writeln("  [Files]-->InputSQLName - неопределенно значение!");
    result := false;
  else
    if (Pos(aIniServ[InputSQLName], "\") < 0) then
      aIniServ[InputSQLName] := sPath + "\" + aIniServ[InputSQLName];
    endif;
  endif;
  // считать значение OutputSQLName
  aIniServ[OutputSQLName] := IniReadString("Files", "OutputSQLName", "");
  if (IsEmpty(aIniServ[OutputSQLName])) then
    writeln("  [Files]-->OutputSQLName - неопределенно значение!");
    result := false;
  else
    if (Pos(aIniServ[OutputSQLName], "\") < 0) then
      aIniServ[OutputSQLName] := sPath + "\" + aIniServ[OutputSQLName];
    endif;
  endif;
  // считать значение ListArcName
  aIniServ[ListArcName] := IniReadString("Files", "ListArcName", "");
  if (IsEmpty(aIniServ[ListArcName])) then
    writeln("  [Files]-->ListArcName - неопределенно значение!");
    result := false;
  else
    if (Pos(aIniServ[ListArcName], "\") < 0) then
      aIniServ[ListArcName] := sPath + "\" + aIniServ[ListArcName];
    endif;
    // проверить на существование файла списка архивов БД станции
    if (not FileExists(aIniServ[ListArcName])) then
      writeln("  [Files]-->ListArcName - отсутствует файл: " +
              aIniServ[ListArcName]);
      result := false;
    endif;
  endif;
end; //ReadIniFile

//*** End of ReadIniFile ***********

//******************************
//* Procedure: PutFile
//* Notes: Запись информации в текстовый файл
//*

procedure PutFile(sStr: string; sNameFile: string);
var
  F : file;
begin
  // открываем выходной файл для записи
  OpenFile(F, sNameFile, true, false);
  // записываем текстовую строку в файл
  writeln(F, sStr);
  // закрываем выходной файл
  CloseFile(F);
end;  // PutFile

//*** End of PutFile ***********

//******************************
//* Function: GetFile
//* Notes: Чтение информации из текстового файла
//*

function GetFile(sFile: string) : string;
```

```
  var
    sStr : string;
    sS : string;
    F : file;
  begin
    sStr := "";
    // открываем файл для чтения
    OpenFile(F, sFile, false, false);
    while (not Eof(F))
      // считывем данные из файла в строку
      readln(F, sS);
      sStr :=  sStr + sS;
    endwhile;
    // закрываем файл
    CloseFile(F);
    result := sStr;
  end;  // GetFile

  //*** End of GetFile ***********

begin
  // вывести заголовок программы
  writeln("");
  writeln(sTitle);
  writeln("");
  sNameFileIni := "";
  // анализ и получение параметров командной строки
  if not ControlComStr(sNameFileIni) then
    exit;
  endif;
  // создать имя файла протокола
  sNameFileLog := ChangeExtension(sNameFileIni, ".log");
  // проверить на существование файла инициализации
  if ((sNameFileIni = "") or (not FileExists(sNameFileIni))) then
    writeln("Ошибка! Отсутствует файл инициализации: ", sNameFileIni);
    exit;
  endif;
  writeln("Файл инициализации: ", sNameFileIni);
  writeln("Файл протокола: " + sNameFileLog);
  writeln(" ");
  // чтение файла инициализации
  writeln("Чтение файла инициализации: ", sNameFileIni);
  if (not ReadIniFile(sNameFileIni, aIniServ)) then
    exit;
  endif;
  // осуществляем соединение с локальным компьютером
  writeln("Соединение с WMI локального компьютера");
  if not @Local->Connect() then
    writeln("Ошибка соединения с WMI локального компьютера");
    exit;
  endif;
  // считываем список архивов
  sArc := GetFile(aIniServ[ListArcName]);
  if (sArc = "") then
    writeln("Внимание! Список БД в файле: ",
            aIniServ[ListArcName], " - пуст!");
    exit;
  endif;
  // считываем текущее имя архива
  sNameArc := IniWordExtr(1, sArc, ",");
  // формируем сценарий
  writeln("Формирование сценария для отключения БД");
  sSQL := "";
  sSQL := sSQL + "DECLARE @Result int" + CRLF();
  sSQL := sSQL + "EXEC @Result = sp_detach_db " + sNameArc + "," +
          " TRUE"+ CRLF();
  sSQL := sSQL + "IF (@Result <> 0)" + CRLF();
```

```
sSQL := sSQL + "  PRINT " + "'" + "##ERROR 1##" + "'" + CRLF();
sSQL := sSQL + "GO" + CRLF();
// записываем в файл
PutFile(sSQL, aIniServ[InputSQLName]);
// запуск сценария
writeln("Отключение БД: " + sNameArc);
if (not Run("osql.exe",
            " -U " + aIniServ[UserDB] +
            " -P " + aIniServ[PasswordDB] +
            " -S " + aIniServ[ServerName] +
            " -d " + aIniServ[DBName] +
            " -i " + aIniServ[InputSQLName] +
            " -o " + aIniServ[OutputSQLName], "", false, 0)) then
  writeln("Ошибка запуска процесса: osql.exe");
  exit;
endif;
// ожидание завершения сценария
if (not @Local->WaitForProcess("osql.exe", 300)) {
  writeln("Ошибка, тайм-аут выполнения сценария." + CRLF() +
          "Выполнение сценария принудительно завершено!");
  exit;
endif;
// проверяем результат выполнения сценария
sSQL := GetFile(aIniServ[OutputSQLName]);
// находим позицию начала строки ошибки
nI := Pos(sSQL, "##ERROR 1##");
// создаем сообщение об ошибке, выводим ее и выходим
if (nI >= 0) then
  writeln("Ошибка, невозможно отключить БД: ", sNameArc);
  exit;
endif;
    // формируем сценарий
writeln("Формирование сценария для перемещения БД");
sSQL := "";
sSQL := sSQL + "DECLARE @Result int" + CRLF();
sSQL := sSQL + "USE master" + CRLF();
sSQL := sSQL + "EXEC @Result = xp_cmdshell" + CRLF();
sSQL := sSQL + "  '" + "move /y " +
  ApostStr(aIniServ[PathDBSource] + "\" + sNameArc + "_Data.MDF") + " " +
  ApostStr(aIniServ[PathDBDestin]) + "'" + ", NO_OUTPUT" + CRLF();
sSQL := sSQL + "IF (@Result <> 0)";
sSQL := sSQL + "  BEGIN" + CRLF();
sSQL := sSQL + "    PRINT " + "'" + "##ERROR 2##" + "'" + CRLF();
sSQL := sSQL + "    RETURN" + CRLF();
sSQL := sSQL + "  END" + CRLF();
sSQL := sSQL + "EXEC @Result = xp_cmdshell" + CRLF();
sSQL := sSQL + "  '" + "move /y " +
  ApostStr(aIniServ[PathDBSource] + "\" + sNameArc + "_Log.LDF") + " " +
  ApostStr(aIniServ[PathDBDestin]) + "'" + ", NO_OUTPUT" + CRLF();
sSQL := sSQL + "IF (@Result <> 0)" + CRLF();
sSQL := sSQL + "  BEGIN" + CRLF();
sSQL := sSQL + "    PRINT " + "'" + "##ERROR 2##" + "'" + CRLF();
sSQL := sSQL + "    RETURN" + CRLF();
sSQL := sSQL + "  END" + CRLF();
sSQL := sSQL + "GO" + CRLF();
// записываем в файл
PutFile(sSQL, aIniServ[InputSQLName]);
writeln("Перемещение БД: ", sNameArc, " ",
  aIniServ[PathDBSource], " --> ",  aIniServ[PathDBDestin]);
if (not Run("osql.exe",
            " -U " + aIniServ[UserDB] +
            " -P " + aIniServ[PasswordDB] +
            " -S " + aIniServ[ServerName] +
            " -d " + aIniServ[DBName] +
            " -i " + aIniServ[InputSQLName] +
            " -o " + aIniServ[OutputSQLName], "", false, 0)) then
  writeln("Ошибка запуска процесса: osql.exe");
```

```
  exit;
endif;
// ожидание завершения сценария
if (not @Local->WaitForProcess("osql.exe", 300)) {
  writeln("Ошибка, тайм-аут выполнения сценария." + CRLF() +
          "Выполнение сценария принудительно завершено!");
  exit;
endif;
// проверяем результат выполнения сценария
sSQL := GetFile(aIniServ[OutputSQLName]);
// находим позицию начала строки ошибки
nI := Pos(sSQL, "##ERROR 2##");
// создаем сообщение об ошибке, выводим ее и выходим
if (nI >= 0) then
  writeln("  Ошибка при перемещении БД: ", sNameArc);
  exit;
endif;
// формируем сценарий
writeln("Формирование сценария для подключения БД");
sSQL := "";
sSQL := sSQL + "DECLARE @Result int" + CRLF();
sSQL := sSQL + "EXEC @Result = sp_attach_db @dbname = " +
  "'" + sNameArc + "'" + "," + CRLF();
sSQL := sSQL + "  @filename1 = " + "'" + aIniServ[PathDBDestin] + "\" +
  sNameArc + "_Data.MDF" + "'" + "," + CRLF();
sSQL := sSQL + "  @filename2 = " + "'" + aIniServ[PathDBDestin] + "\" +
  sNameArc + "_Log.LDF" + "'" + CRLF();
sSQL := sSQL + "IF (@Result <> 0)" + CRLF();
sSQL := sSQL + "  PRINT " + "'" + "##ERROR 3##" + "'" + CRLF();
sSQL := sSQL + "GO" + CRLF();
// записываем в файл
PutFile(sSQL, aIniServ[InputSQLName]);
// запуск сценария
writeln("Подключение БД: ", sNameArc);
if (not Run("osql.exe",
            " -U " + aIniServ[UserDB] +
            " -P " + aIniServ[PasswordDB] +
            " -S " + aIniServ[ServerName] +
            " -d " + aIniServ[DBName] +
            " -i " + aIniServ[InputSQLName] +
            " -o " + aIniServ[OutputSQLName], "", false, 0)) then
  writeln("Ошибка запуска процесса: osql.exe");
  exit;
endif;
// ожидание завершения сценария
if (not @Local->WaitForProcess("osql.exe", 300)) {
  writeln("Ошибка, тайм-аут выполнения сценария." + CRLF() +
          "Выполнение сценария принудительно завершено!");
  exit;
endif;
// проверяем результат выполнения сценария
sSQL := GetFile(aIniServ[OutputSQLName]);
// находим позицию начала строки ошибки
nI := Pos(sSQL, "##ERROR 3##");
// создаем сообщение об ошибке, выводим ее и выходим
if (nI >= 0) then
  writeln("Ошибка, невозможно подключить БД: " + sNameArc);
  exit;
endif;
sNameArc := "";
// определяем длину списка БД
nLen := IniWordNum(sArc, ",");
// уменьшаем список БД на один и записываем его в файл
if (nLen > 1) then
  for nI := 2 to nLen
    sNameArc := sNameArc + IniWordExtr(nI, sArc, ",");;
    if nI < nLen then
```

```
        sNameArc := sNameArc + ",";
      endif;
    endfor;
    sArc := sNameArc;
  else
    sArc := "";
  endif;
  PutFile(sArc, aIniServ[ListArcName]);
  // вывод сообщения о завершении изменения БД сервера
  writeln("Перемещение БД завершено успешно!");
end.

//*** End of MoveDB.alf ***********
```

Ниже представлен файл протокола программы MoveDB.

```
************************************************************
MoveDB
  Дата: 30.11.2010  Время: 15:01:56
  Время выполнения: 00:48.49
************************************************************

MoveDB v1.0 UAF - Перемещение БД

Файл инициализации: d:\user\alfa\prog\MoveDB.ini
Файл протокола: d:\user\alfa\prog\MoveDB.log

Чтение файла инициализации: d:\user\alfa\prog\MoveDB.ini
Соединение с WMI локального компьютера
Формирование сценария для отключения БД
Отключение БД: Test
Формирование сценария для перемещения БД
Перемещение БД: Test c:\Program Files\Microsoft SQL Server\MSSQL\data --> c:\test
Формирование сценария для подключения БД
Подключение БД: Test
Перемещение БД завершено успешно!
```