Оглавление

[**DK\_LCLStrRus** 2](#_Toc125848205)

[**DK\_Const** 3](#_Toc125848206)

[**DK\_Dialogs** 4](#_Toc125848207)

[**DK\_DateUtils** 5](#_Toc125848208)

[**DK\_Vector** 10](#_Toc125848209)

[**DK\_Matrix** 19](#_Toc125848210)

[**DK\_VMShow** 24](#_Toc125848211)

[**DK\_StrUtils** 25](#_Toc125848212)

[**DK\_TextUtils** 29](#_Toc125848213)

[**DK\_PriceUtils** 30](#_Toc125848214)

[**DK\_SQLUtils** 31](#_Toc125848215)

[**DK\_Fonts** 33](#_Toc125848216)

[**DK\_Math** 34](#_Toc125848217)

# **DK\_LCLStrRus**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **НЕТ**

Русификация строковых констант LCL. Перевод происходит в разделе initialization модуля, достаточно включить DK\_LCLStrRus в uses главной формы приложения.

# **DK\_Const**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **Нет**

Константы, использующиеся в модулях пакета.

SYMBOL\_BREAK = LineEnding;

SYMBOL\_EMPTY = '';

SYMBOL\_COMMA = ',';

SYMBOL\_OPENBRACKET = '(';

SYMBOL\_CLOSEBRACKET = ')';

SYMBOL\_SPACE = ' '; //пробел

SYMBOL\_CARRY = '-'; //символ переноса слова

SYMBOLS\_SEPARATOR = SYMBOL\_SPACE; //символы разделения слов

SYMBOLS\_PUNCTUATION = '-,;:.?!/\()"«»'; //пунктуация, разделители

SYMBOLS\_VOWEL = 'АЕЁИОУЭЫЮЯ'; //гласные буквы

SYMBOLS\_CONSONANT = 'БВГДЖЙЗКЛМНПРСТФЧЦХШЩ'; //согласные буквы

SYMBOLS\_SPECIAL = 'ЬЪ';

SYMBOLS\_DIGITS = '1234567890';

SYMBOLS\_OTHER = '№!?,.;:@#$%^&<>()[]{}\/|-\_+\*="''';

LETTERS\_ENGLISH\_LOWER = 'abcdefghijklmnopqrstuvwxyz';

LETTERS\_ENGLISH\_UPPER = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

LETTERS\_RUSSIAN\_LOWER = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя';

LETTERS\_RUSSIAN\_UPPER = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ';

SYMBOLS\_KEYBOARD = SYMBOLS\_DIGITS + SYMBOLS\_OTHER + SYMBOL\_SPACE +

LETTERS\_ENGLISH\_LOWER + LETTERS\_ENGLISH\_UPPER +

LETTERS\_RUSSIAN\_LOWER + LETTERS\_RUSSIAN\_UPPER;

STR\_TRUE = 'да';

STR\_FALSE = 'нет';

NULDATE: TDate = 36526; //01.01.2000

INFDATE: TDate = 401768; //31.12.2999

MONTHSNOM: array [1..12] of String = ('январь', 'февраль', 'март', 'апрель', 'май', 'июнь', 'июль', 'август', 'сентябрь', 'октябрь', 'ноябрь', 'декабрь');

MONTHSGEN: array [1..12] of String = ('января', 'февраля', 'марта', 'апреля', 'мая', 'июня', 'июля', 'августа', 'сентября', 'октября', 'ноября', 'декабря');

MONTHSPREP: array [1..12] of String = ('январе', 'феврале', 'марте', 'апреле', 'мае', 'июне', 'июле', 'августе', 'сентябре', 'октябре', 'ноябре', 'декабре');

WEEKDAYSSHORT: array [1..7] of String = ('Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс');

WEEKDAYSLONG: array [1..7] of String = ('Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота', 'Воскресенье');

VECTOR\_INT\_DEFAULT\_VALUE = 0;

VECTOR\_INT64\_DEFAULT\_VALUE = 0;

VECTOR\_STR\_DEFAULT\_VALUE = '';

VECTOR\_DBL\_DEFAULT\_VALUE = 0;

VECTOR\_BOOL\_DEFAULT\_VALUE = False;

VECTOR\_COLOR\_DEFAULT\_VALUE = 0;

# **DK\_Dialogs**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const**

Диалоговые окна.

|  |  |  |
| --- | --- | --- |
| procedure ShowInfo(const AInfo: String;  const ACaption: String = SpaceStr); | **ОКНО С СООБЩЕНИЕМ**  Кнопки: "Ок"  Входные параметры:  AInfo - текст сообщения  ACaption - заголовок окна сообщения | ShowInfo('Информация', 'Заголовок окна'); |
| function Confirm(const AQuestion: String;  const ACaption: String = SpaceStr): Boolean; | **ОКНО С ЗАПРОСОМ НА УДАЛЕНИЕ**  Кнопки: "Да" [Result=True],  "Нет" (по умолчанию)[Result=False]  Входные параметры:  AQuestion - текст сообщения  ACaption - заголовок окна сообщения | Confirm('Запрос', 'Заголовок окна'); |

# **DK\_DateUtils**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const**

Утилиты работы с датами.

|  |  |  |
| --- | --- | --- |
| function DateToStrNominative(const ADate: TDate): String; | ДАТА ПРОПИСЬЮ В ИМЕНИТЕЛЬНОМ ПАДЕЖЕ  Входные параметры:  ADate - дата | DateToStrNominative(StrToDate(‘01.04.2020’))  ’01 апрель 2020г.’ |
| function DateToStrGenitive(const ADate: TDate): String; | ДАТА ПРОПИСЬЮ В РОДИТЕЛЬНОМ ПАДЕЖЕ  Входные параметры:  ADate - дата | DateToStrGenitive(StrToDate(‘01.04.2020’))  ’01 апреля 2020г.’ |
| function DayOfDate(const ADate: TDate): Word; | ИЗВЛЕЧЕНИЕ НОМЕРА ДНЯ ИЗ ДАТЫ  Входные параметры:  ADate - дата | DayOfDate(StrToDate(‘01.04.2020’))  1 |
| function MonthOfDate(const ADate: TDate): Word; | ИЗВЛЕЧЕНИЕ НОМЕРА МЕСЯЦА ИЗ ДАТЫ  Входные параметры:  ADate - дата | MonthOfDate(StrToDate(‘01.04.2020’))  4 |
| function YearOfDate(const ADate: TDate): Word; | ИЗВЛЕЧЕНИЕ НОМЕРА ГОДА ИЗ ДАТЫ  Входные параметры:  ADate - дата | YearOfDate(StrToDate(‘01.04.2020’))  2020 |
| function DayNumberInYear(const ADate: TDate): Word; | НОМЕР ДНЯ В ГОДУ  Входные параметры:  ADate - дата |  |
| function DayNumberInYear(  const ADay, AMonth, AYear: Word): Word; | НОМЕР ДНЯ В ГОДУ  Входные параметры:  ADay – число  AMonth – месяц  AYear – год |  |
| function DayNumberInWeek(const ADate: TDate): Word; | НОМЕР ДНЯ В НЕДЕЛЕ  Входные параметры:  ADate - дата |  |
| function DayNumberInWeek(  const ADay, AMonth, AYear: Word): Word; | НОМЕР ДНЯ В НЕДЕЛЕ  Входные параметры:  ADay – число  AMonth – месяц  AYear – год |  |
| function WeekNumberInMonth(const ADate: TDate): Word; | НОМЕР НЕДЕЛИ В МЕСЯЦЕ  Входные параметры:  ADate - дата |  |
| function WeekNumberInMonth(  const ADay, AMonth, AYear: Word): Word; | НОМЕР НЕДЕЛИ В МЕСЯЦЕ  Входные параметры:  ADay – число  AMonth – месяц  AYear – год |  |
| function QuarterNumber(const AMonth: Byte): Byte; | НОМЕР КВАРТАЛА  Входные параметры:  AMonth – номер месяца в году | QuarterNumber(3)  1 |
| function QuarterNumber(const ADate: TDate): Byte; | НОМЕР КВАРТАЛА  Входные параметры:  ADate - дата | QuarterNumber(StrToDate(‘01.04.2020’))  2 |
| function MonthNumberInQuarter(const AMonth: Byte): Byte; | НОМЕР МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  AMonth – номер месяца в году | MonthNumberInQuarter(4)  1 |
| function MonthNumberInQuarter(const ADate: TDate): Byte; | НОМЕР МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  ADate - дата | MonthNumberInQuarter(StrToDate(‘01.04.2020’))  1 |
| function FirstMonthInQuarter(const AQuart: Byte): Byte; | НОМЕР ПЕРВОГО МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  AQuart – номер квартала | FirstMonthInQuarter(2)  4 |
| function FirstMonthInQuarter(const ADate: TDate): Byte; | НОМЕР ПЕРВОГО МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  ADate - дата | FirstMonthInQuarter(StrToDate(‘01.04.2020)’)  4 |
| function LastMonthInQuarter(const AQuart: Byte): Byte; | НОМЕР ПОСЛЕДНЕГО МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  AQuart – номер квартала | LastMonthInQuarter(2)  6 |
| function LastMonthInQuarter(const ADate: TDate): Byte; | НОМЕР ПОСЛЕДНЕГО МЕСЯЦА В КВАРТАЛЕ  Входные параметры:  ADate - дата | LastMonthInQuarter(StrToDate(‘01.04.2020’))  6 |
| function QuarterNumberToRome(const AQuart: Byte): String; | ПЕРЕВОД НОМЕРА КВАРТАЛА ИЗ АРАБСКОЙ ЦИФРЫ В РИМСКУЮ  Входные параметры:  AQuart – номер квартала | QuarterNumberToRome(2)  ‘II’ |
| function HalfYearNumber(const AMonth: Byte): Byte; | НОМЕР ПОЛУГОДИЯ  Входные параметры:  AMonth – месяц | HalfYearNumber(7)  2 |
| function HalfYearNumber(const ADate: TDate): Byte; | НОМЕР ПОЛУГОДИЯ  Входные параметры:  ADate - дата | HalfYearNumber(StrToDate(‘01.04.2020’))  1 |
| function FirstDayInMonth(const ADate: TDate): TDate; | ДАТА НАЧАЛА МЕСЯЦА  Входные параметры:  ADate - дата | FirstDayInMonth(StrToDate(‘20.04.2020’))  //01.04.2020 |
| function FirstDayInMonth(const AMonth, AYear: Word): TDate; | ДАТА НАЧАЛА МЕСЯЦА  Входные параметры:  AMonth – месяц  AYear – год | FirstDayInMonth(4, 2020)  //01.04.2020 |
| function LastDayInMonth(const ADate: TDate): TDate; | ДАТА КОНЦА МЕСЯЦА  Входные параметры:  ADate - дата | LastDayInMonth(StrToDate(‘20.04.2020’))  //30.04.2020 |
| function LastDayInMonth(const AMonth, AYear: Word): TDate; | ДАТА КОНЦА МЕСЯЦА  Входные параметры:  AMonth – месяц  AYear – год | LastDayInMonth(4, 2020)  //30.04.2020 |
| procedure FirstLastDayInMonth(const ADate: TDate;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА МЕСЯЦА  Входные параметры:  ADate – дата  Выходные параметры:  AFirstDay – дата начала месяца  ALastDay – дата конца месяца | FirstLastDayInMonth(StrToDate(‘20.04.2020’),  AFirstDay, ALastDay)  //AFirstDay=01.04.2020, ALastDay=30.04.2020 |
| procedure FirstLastDayInMonth(const AMonth, AYear: Word;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА МЕСЯЦА  Входные параметры:  AMonth – месяц  AYear – год  Выходные параметры:  AFirstDay – дата начала месяца  ALastDay – дата конца месяца | FirstLastDayInMonth(4, 2020,  AFirstDay, ALastDay)  //AFirstDay=01.04.2020, ALastDay=30.04.2020 |
| function FirstDayInYear(const AYear: Word): TDate; | ДАТА НАЧАЛА ГОДА  Входные параметры:  AYear – год | FirstDayInYear(2020)  //01.01.2020 |
| function FirstDayInYear(const ADate: TDate): TDate; | ДАТА НАЧАЛА ГОДА  Входные параметры:  ADate – дата | FirstDayInYear(StrToDate(‘20.04.2020’))  //01.01.2020 |
| function LastDayInYear(const AYear: Word): TDate; | ДАТА КОНЦА ГОДА  Входные параметры:  AYear – год | LastDayInYear(2020)  //31.12.2020 |
| function LastDayInYear(const ADate: TDate): TDate; | ДАТА КОНЦА ГОДА  Входные параметры:  ADate - дата | LastDayInYear(StrToDate(‘20.04.2020’))  //31.12.2020 |
| procedure FirstLastDayInYear(const AYear: Word;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА ГОДА  Входные параметры:  AYear – год  Выходные параметры:  AFirstDay – дата начала года  ALastDay – дата конца года | FirstLastDayInYear(2020,  AFirstDay, ALastDay)  //AFirstDay=01.01.2020, ALastDay=31.12.2020 |
| procedure FirstLastDayInYear(const ADate: TDate;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА ГОДА  Входные параметры:  ADate – дата  Выходные параметры:  AFirstDay – дата начала года  ALastDay – дата конца года | FirstLastDayInYear(StrToDate(‘20.04.2020’),  AFirstDay, ALastDay)  //AFirstDay=01.01.2020, ALastDay=31.12.2020 |
| function FirstDayInQuarter(const AQuart, AYear: Word): TDate; | ДАТА НАЧАЛА КВАРТАЛА  Входные параметры:  AQuart – квартал  AYear – год | FirstDayInQuarter(2, 2020)  //01.04.2020 |
| function FirstDayInQuarter(const ADate: TDate): TDate; | ДАТА НАЧАЛА КВАРТАЛА  Входные параметры:  ADate – дата | FirstDayInQuarter(StrToDate(‘20.04.2020’))  //01.04.2020 |
| function LastDayInQuarter (const AQuart, AYear: Word): TDate; | ДАТА КОНЦА КВАРТАЛА  Входные параметры:  AQuart – квартал  AYear – год | LastDayInQuarter (2, 2020)  //30.06.2020 |
| function LastDayInQuarter (const ADate: TDate): TDate; | ДАТА КОНЦА КВАРТАЛА  Входные параметры:  ADate – дата | LastDayInQuarter(StrToDate(‘20.04.2020’))  //30.06.2020 |
| procedure FirstLastDayInQuarter(const AQuart, AYear: Word;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА КВАРТАЛА  Входные параметры:  AQuart – квартал  AYear – год  Выходные параметры:  AFirstDay – дата начала квартала  ALastDay – дата конца квартала | FirstLastDayInQuarter (2, 2020,  AFirstDay, ALastDay)  //AFirstDay=01.04.2020, ALastDay=30.06.2020 |
| procedure FirstLastDayInQuarter(const ADate: TDate;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА КВАРТАЛА  Входные параметры:  ADate – дата  Выходные параметры:  AFirstDay – дата начала квартала  ALastDay – дата конца квартала | FirstLastDayInQuarter(StrToDate(‘20.04.2020’),  AFirstDay, ALastDay)  //AFirstDay=01.04.2020, ALastDay=30.06.2020 |
| procedure FirstLastDayInHalfYear(const AHalf, AYear: Word;  out AFirstDay, ALastDay: TDate); |  | FirstLastDayInHalfYear(2, 2020,  AFirstDay, ALastDay)  //AFirstDay=01.07.2020, ALastDay=31.12.2020 |
| procedure FirstLastDayInHalfYear(const ADate: TDate;  out AFirstDay, ALastDay: TDate); | ДАТЫ НАЧАЛА И КОНЦА ПОЛУГОДИЯ  Входные параметры:  ADate – дата  Выходные параметры:  AFirstDay – дата начала полугодия  ALastDay – дата конца полугодия | FirstLastDayInHalfYear(StrToDate(‘20.04.2020’),  AFirstDay, ALastDay)  //AFirstDay=01.01.2020, ALastDay=30.06.2020 |
| function MinDate(const ADate1, ADate2: TDate): TDate; | МЕНЬШАЯ ИЗ ДВУХ ДАТ  Входные параметры:  ADate1– 1-ая дата  ADate2– 2-ая дата | MinDate(StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  //01.04.2020 |
| function MaxDate(const ADate1, ADate2: TDate): TDate; | БОЛЬШАЯ ИЗ ДВУХ ДАТ  Входные параметры:  ADate1– 1-ая дата  ADate2– 2-ая дата | MaxDate(StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  //02.04.2020 |
| function DaysBetweenDates(const ADate1, ADate2: TDate): Integer; | КОЛИЧЕСТВО ДНЕЙ МЕЖДУ ДАТАМИ  Входные параметры:  ADate1– 1-ая дата  ADate2– 2-ая дата | DaysBetweenDates(StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  1 |
| function DaysInPeriod(const ADate1, ADate2: TDate): Integer; | КОЛИЧЕСТВО ДНЕЙ ЗА ПЕРИОД  Входные параметры:  ADate1– начальная дата периода  ADate2– конечная дата периода | DaysInPeriod(StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  2 |
| function IsDateInPeriod(  const ADate, ADate1, ADate2: TDate): Boolean; | ПРОВЕРКА НАХОЖДЕНИЯ ДАТЫ В ПЕРИОДЕ  Входные параметры:  ADate – проверяемая дата  ADate1– начальная дата периода  ADate2– конечная дата периода | IsDateInPeriod(StrToDate(‘01.04.2020’),  StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  //true |
| function IsEqualPeriods(  const ABegin1, AEnd1, ABegin2, AEnd2: TDate): Boolean; | ПРОВЕРКА РАВЕНСТВА ПЕРИОДОВ  Входные параметры:  ABegin1– начальная дата 1-ого периода  AEnd1– конечная дата 1-ого периода  ABegin2– начальная дата 2-ого периода  AEnd2– конечная дата 2-ого периода | IsEqualPeriods(StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’), StrToDate(‘01.04.2020’),  StrToDate(‘02.04.2020’))  //true |
| function IsPeriodIntersect(  const ABegin1, AEnd1, ABegin2, AEnd2: TDate): Boolean; | ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПЕРИОДОВ  Входные параметры:  ABegin1– начальная дата 1-ого периода  AEnd1– конечная дата 1-ого периода  ABegin2– начальная дата 2-ого периода  AEnd2– конечная дата 2-ого периода | IsEqualPeriods(StrToDate(‘01.04.2020’),  StrToDate(‘10.04.2020’), StrToDate(‘03.04.2020’),  StrToDate(‘20.04.2020’))  //true |
| function IsPeriodIntersect(  const ABegin1, AEnd1, ABegin2, AEnd2: TDate;  out ABegin, AEnd: TDate): Boolean; | ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПЕРИОДОВ  Входные параметры:  ABegin1– начальная дата 1-ого периода  AEnd1– конечная дата 1-ого периода  ABegin2– начальная дата 2-ого периода  AEnd2– конечная дата 2-ого периода  Выходные параметры:  ABegin– начальная дата пересечения  AEnd– конечная дата пересечения  Если false, то ABegin=ABegin1, AEnd=AEnd1 | IsEqualPeriods(StrToDate(‘01.04.2020’),  StrToDate(‘10.04.2020’), StrToDate(‘03.04.2020’),  StrToDate(‘20.04.2020’),  ABegin, AEnd)  //true  //ABegin=03.04.2020, AEnd=10.04.2020 |
| function IncMonthExt(  const ADate: TDate; const ADeltaMonth: Extended): TDate; | ПРИБАВЛЯЕТ К ДАТЕ ДРОБНОЕ ЧИСЛО МЕСЯЦЕВ  Входные параметры:  ADate – дата  ADeltaMonth – количество месяцев | IncMonthExt(StrToDate(‘01.04.2020’), 0.5)  //16.04.2020 |
| function IsBoundaryDate(const ADate: TDate): Boolean; | ОПРЕДЕЛЯЕТ, ЯВЛЯЕТСЯ ЛИ ДАТА ГРАНИЧНОЙ  {0, NULDATE, INFDATE, MinDateTime, MaxDateTime} |  |

# **DK\_Vector**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const, DK\_DateUtils, DK\_StrUtils**

Работа с векторами (динамическими массивами)

type

TIntVector = array of Integer;

TInt64Vector = array of Int64;

TStrVector = array of String;

TDblVector = array of Double;

TDateVector = type TDblVector;

TTimeVector = type TDblVector;

TBoolVector = array of Boolean;

TColorVector = array of TColor;

|  |  |  |
| --- | --- | --- |
| procedure VDim(var V: TIntVector; const Size: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure VDim(var V: TInt64Vector; const Size: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure VDim(var V: TStrVector; const Size: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure VDim(var V: TDblVector; const Size: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure VDim(var V: TBoolVector; const Size: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE);  procedure VDim(var V: TColorVector; const Size: Integer;  const DefaultValue: TColor = VECTOR\_COLOR\_DEFAULT\_VALUE); | VDim(V, Size, DefaultValue) - cоздает вектор V длиной Size, заполненный значениями DefaultValue. Если в V что-то было - информация теряется |  |
| procedure VReDim(var V: TIntVector; const Size: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure VReDim(var V: TInt64Vector; const Size: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure VReDim(var V: TStrVector; const Size: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure VReDim(var V: TDblVector; const Size: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure VReDim(var V: TBoolVector; const Size: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE);  procedure VReDim(var V: TColorVector; const Size: Integer;  const DefaultValue: TColor = VECTOR\_COLOR\_DEFAULT\_VALUE); | VReDim(V, Size, DefaultValue) - изменяет длину вектора V до размера Size. Если новый размер меньше исходного, вектор усекается, усеченные элементы теряются. Если новый размер больше исходного, вектор дополняется элементами  со значениями DefaultValue; оставшиеся элементы не изменяются. |  |
| function VIsNil(const V: TIntVector): Boolean;  function VIsNil(const V: TInt64Vector): Boolean;  function VIsNil(const V: TStrVector): Boolean;  function VIsNil(const V: TDblVector): Boolean;  function VIsNil(const V: TBoolVector): Boolean;  function VIsNil(const V: TColorVector): Boolean; | VIsNil(V) – проверка V=nil |  |
| function VCreateInt(const V: array of Integer): TIntVector;  function VCreateInt64(const V: array of Int64): TInt64Vector;  function VCreateStr(const V: array of String): TStrVector;  function VCreateDbl(const V: array of Double): TDblVector;  function VCreateDate(const V: array of TDate): TDateVector;  function VCreateTime(const V: array of TTime): TTimeVector;  function VCreateBool(const V: array of Boolean): TBoolVector;  function VCreateColor(const V: array of TColor): TColorVector; | VCreateXXX() – создание вектора из массива | V:= VCreateInt([1,2,3]) |
| function VFromStrings(const S: TStrings): TStrVector; | VFromStrings – создание строкового вектора из списка TStrings |  |
| procedure VToStrings(const V: TStrVector; const S: TStrings); | VToStrings – заполнение списка TStrings из строкового вектора |  |
| function VStrToVector(const Str, Delimiter: String): TStrVector; | VStrToVector – разбиавет строку Str на части, разделенные Delimiter и записывает полученные части в строковый вектор |  |
| function VVectorToStr(const V: TStrVector; const Delimiter: String): String; | VVectorToStr – склеивает элементы вектора V в строку, вставляя между ними Delimiter |  |
| function VFirst(const V: TIntVector): Integer;  function VFirst(const V: TInt64Vector): Int64;  function VFirst(const V: TStrVector): String;  function VFirst(const V: TDblVector): Double;  function VFirst(const V: TBoolVector): Boolean;  function VFirst(const V: TColorVector): TColor; | VFirst(V) – возвращает значение первого элемента вектора V |  |
| function VLast(const V: TIntVector): Integer;  function VLast(const V: TInt64Vector): Int64;  function VLast(const V: TStrVector): String;  function VLast(const V: TDblVector): Double;  function VLast(const V: TBoolVector): Boolean;  function VLast(const V: TColorVector): TColor; | VLast (V) – возвращает значение последнего элемента вектора V |  |
| procedure VSwap(var V: TIntVector;  const Index1, Index2: Integer);  procedure VSwap(var V: TInt64Vector;  const Index1, Index2: Integer);  procedure VSwap(var V: TStrVector;  const Index1, Index2: Integer);  procedure VSwap(var V: TDblVector;  const Index1, Index2: Integer);  procedure VSwap(var V: TBoolVector;  const Index1, Index2: Integer);  procedure VSwap(var V: TColorVector;  const Index1, Index2: Integer); | VSwap(V,Index1,Index2) – меняет местами значения элементов вектора V с индексами Index1 и Index2 |  |
| procedure VDel(var V: TIntVector; const Index1: Integer; Index2: Integer = -1);  procedure VDel(var V: TInt64Vector; const Index1: Integer; Index2: Integer = -1);  procedure VDel(var V: TStrVector; const Index1: Integer; Index2: Integer = -1);  procedure VDel(var V: TDblVector; const Index1: Integer; Index2: Integer = -1);  procedure VDel(var V: TBoolVector; const Index1: Integer; Index2: Integer = -1);  procedure VDel(var V: TColorVector; const Index1: Integer; Index2: Integer = -1); | VDel(V,Index1,Index2) – удаляет элементы вектора V с индекса Index1 до индекса Index2 (включительно).  Если Index2 не указан – удаляет только один элемент с индексом Index1.  Если Index2<Index1, будет удален только 1 элемент с индексом Index1 |  |
| procedure VCopy(const Source: TIntVector; var Dest: TIntVector; const FromIndex: Integer = 0);  procedure VCopy(const Source: TInt64Vector;var Dest: TInt64Vector; const FromIndex: Integer = 0);  procedure VCopy(const Source: TStrVector; var Dest: TStrVector; const FromIndex: Integer = 0);  procedure VCopy(const Source: TDblVector; var Dest: TDblVector; const FromIndex: Integer = 0);  procedure VCopy(const Source: TBoolVector; var Dest: TBoolVector; const FromIndex: Integer = 0);  procedure VCopy(const Source: TColorVector; var Dest: TColorVector;const FromIndex: Integer = 0); | VCopy(Source, Dest, FromIndex) - копирует все элементы вектора Source в вектор Dest, начиная с позиции FromIndex (вектора Dest).  Если FromIndex не указывать - копирование в начало вектора.  Если длина Dest недостаточна для записи вектора Source - длина увеличивается до необходимого размера |  |
| procedure VIns(var V: TIntVector; const Ind: Integer; const Value: Integer);  procedure VIns(var V: TInt64Vector;const Ind: Integer; const Value: Int64);  procedure VIns(var V: TStrVector; const Ind: Integer; const Value: String);  procedure VIns(var V: TDblVector; const Ind: Integer; const Value: Double);  procedure VIns(var V: TBoolVector; const Ind: Integer; const Value: Boolean);  procedure VIns(var V: TColorVector; const Ind: Integer; const Value: TColor); | VIns (V,Ind,Value)- вставляет скаляр Value в позицию Ind вектора V |  |
| procedure VIns(var V: TIntVector; const Ind: Integer; const Source: TIntVector);  procedure VIns(var V: TInt64Vector; const Ind: Integer; const Source: TInt64Vector);  procedure VIns(var V: TStrVector; const Ind: Integer; const Source: TStrVector);  procedure VIns(var V: TDblVector; const Ind: Integer; const Source: TDblVector);  procedure VIns(var V: TBoolVector; const Ind: Integer; const Source: TBoolVector);  procedure VIns(var V: TColorVector; const Ind: Integer; const Source: TColorVector); | VIns (V,Ind,Source)- вставляет вектор Source в позицию Ind вектора V |  |
| procedure VChangeIn(var V: TIntVector; const NewValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIn(var V: TInt64Vector; const NewValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIn(var V: TStrVector; const NewValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIn(var V: TDblVector; const NewValue: Double;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIn(var V: TBoolVector; const NewValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIn(var V: TColorVector; const NewValue: TColor;  FromIndex: Integer=-1; ToIndex: Integer=-1); | ChangeIn(V,NewValue,FromIndex,ToIndex) - изменяет значения вектора V на NewValue c индекса FromIndex до ToIndex (включительно). FromIndex, ToIndex можно не указывать, тогда будет изменение по всему вектору V |  |
| procedure VChangeNotIn(var V: TIntVector; const NewValue: Integer; FromIndex, ToIndex: Integer);  procedure VChangeNotIn(var V: TInt64Vector; const NewValue: Int64; FromIndex, ToIndex: Integer);  procedure VChangeNotIn(var V: TStrVector; const NewValue: String; FromIndex, ToIndex: Integer);  procedure VChangeNotIn(var V: TDblVector; const NewValue: Double; FromIndex, ToIndex: Integer);  procedure VChangeNotIn(var V: TBoolVector; const NewValue: Boolean; FromIndex, ToIndex: Integer);  procedure VChangeNotIn(var V: TColorVector; const NewValue: TColor; FromIndex, ToIndex: Integer); | ChangeNotIn(V,NewValue,FromIndex, ToIndex) - изменяет значения вектора V, лежащие за пределами диапазона c индекса FromIndex до ToIndex на NewValue.  FromIndex, ToIndex можно не указывать, тогда будет изменение по всему вектору V |  |
| procedure VChangeIf(var V: TIntVector; const IfValue, NewValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIf(var V: TInt64Vector; const IfValue, NewValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIf(var V: TStrVector; const IfValue, NewValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True);  procedure VChangeIf**Date**(var V: TDateVector; const IfValue, NewValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIf**Time**(var V: TTimeVector; const IfValue, NewValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIf(var V: TBoolVector; const IfValue, NewValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1);  procedure VChangeIf(var V: TColorVector; const IfValue, NewValue: TColor;  FromIndex: Integer=-1; ToIndex: Integer=-1); | VChangeIf(V,IfValue,NewValue, FromIndex, ToIndex) – изменяет значения вектора V, равные IfValue, на NewValue c индекса FromIndex до ToIndex (включительно).  FromIndex, ToIndex можно не указывать, тогда будет изменение по всему вектору V  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VCut(const V: TIntVector; FromIndex: Integer=-1; ToIndex: Integer=-1): TIntVector;  function VCut(const V: TInt64Vector; FromIndex: Integer=-1; ToIndex: Integer=-1): TInt64Vector;  function VCut(const V: TStrVector; FromIndex: Integer=-1; ToIndex: Integer=-1): TStrVector;  function VCut(const V: TDblVector; FromIndex: Integer=-1; ToIndex: Integer=-1): TDblVector;  function VCut(const V: TBoolVector; FromIndex: Integer=-1; ToIndex: Integer=-1): TBoolVector;  function VCut(const V: TColorVector; FromIndex: Integer=-1; ToIndex: Integer=-1): TColorVector; | VCut(V,FromIndex,ToIndex) – возвращает срез вектора V от FromIndex до ToIndex (включительно).  Если ToIndex не указан – срез от FromIndex до конца вектора.  Если не указаны оба индекса – копия вектора целиком |  |
| function VCut(const V: TDateVector; const ABeginDate, AEndDate: TDate): TDateVector; | VCut(V,ABeginDate,AEndDate) – возвращает срез упорядоченного вектора дат, в которых включаются элементы, лежащие в интервале от даты ABeginDate до даты AEndDate (включительно) |  |
| function VCut(const V: TIntVector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TIntVector;  function VCut(const V: TInt64Vector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TInt64Vector;  function VCut(const V: TStrVector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TStrVector;  function VCut(const V: TDblVector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TDblVector;  function VCut(const V: TBoolVector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TBoolVector;  function VCut(const V: TColorVector; const Used: TBoolVector;  FromIndex: Integer=-1; ToIndex: Integer=-1): TColorVector; | VCut(V,Used,FromIndex,ToIndex) – возвращает срез вектора V от FromIndex до ToIndex (включительно) с условием входждения - в срез включаются только элементы V[i], для которых Used[i]=True.  Если ToIndex не указан – срез от FromIndex до конца вектора.  Если не указаны оба индекса – копия вектора целиком |  |
| function VAdd(const V1,V2: TIntVector): TIntVector;  function VAdd(const V1,V2: TInt64Vector): TInt64Vector;  function VAdd(const V1,V2: TStrVector): TStrVector;  function VAdd(const V1,V2: TDblVector): TDblVector;  function VAdd(const V1,V2: TBoolVector): TBoolVector;  function VAdd(const V1,V2: TColorVector): TColorVector; | VAdd(V1,V2) – возвращает сцепление векторов (присоединяет вектор V2 к концу V1 и возвращает получившийся вектор) |  |
| function VIndexOf(const V: TIntVector; const FindValue: Integer): Integer;  function VIndexOf(const V: TInt64Vector; const FindValue: Int64): Integer;  function VIndexOf(const V: TStrVector; const FindValue: String;  const ACaseSensitivity: Boolean = True): Integer;  function VIndexOf**Date**(const V: TDateVector; const FindValue: TDate): Integer;  function VIndexOf**Time**(const V: TTimeVector; const FindValue: TTime): Integer;  function VIndexOf(const V: TBoolVector; const FindValue: Boolean): Integer;  function VIndexOf(const V: TColorVector; const FindValue: TColor): Integer; | VIndexOf(V, FindValue) – поиск значения FindValue в векторе V (начиная с индекса 0) - возвращает индекс найденного элемента. Если такого элемента нет – возвращает -1  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VInsOfAsc(const V: TIntVector; const InsValue: Integer): Integer;  function VInsOfAsc(const V: TInt64Vector; const InsValue: Int64): Integer;  function VInsOfAsc(const V: TStrVector; const InsValue: String;  const ACaseSensitivity: Boolean = True): Integer;  function VInsOfAsc**Date**(const V: TDateVector; const InsValue: TDate): Integer;  function VInsOfAsc**Time**(const V: TTimeVector; const InsValue: TTime): Integer; | VInsOfAsc(V, InsValue) – поиск позиции для вставки нового значения InsValue в упорядоченный по возрастанию вектор V (для сохранения порядка значений) - возвращает индекс найденной позиции.  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VInsOfDesc(const V: TIntVector; const InsValue: Integer): Integer;  function VInsOfDesc(const V: TInt64Vector; const InsValue: Int64): Integer;  function VInsOfDesc(const V: TStrVector; const InsValue: String;  const ACaseSensitivity: Boolean = True): Integer;  function VInsOfDesc**Date**(const V: TDateVector; const InsValue: TDate): Integer;  function VInsOfDesc**Time**(const V: TTimeVector; const InsValue: TTime): Integer; | VInsOfDesc(V, InsValue) – поиск позиции для вставки нового значения InsValue в упорядоченный по убыванию вектор V (для сохранения порядка значений) - возвращает индекс найденной позиции.  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| procedure VAppend(var V: TIntVector; const Value: Integer);  procedure VAppend(var V: TInt64Vector; const Value: Int64);  procedure VAppend(var V: TStrVector; const Value: String);  procedure VAppend(var V: TDblVector; const Value: Double);  procedure VAppend(var V: TBoolVector; const Value: Boolean);  procedure VAppend(var V: TColorVector; const Value: TColor); | VAppend(V, Value) – присоединяет скаляр Value к концу вектора V. |  |
| function VUnique(const V: TIntVector): TIntVector;  function VUnique(const V: TInt64Vector): TInt64Vector;  function VUnique(const V: TStrVector; const ACaseSensitivity: Boolean = True): TStrVector;  function VUnique**Date**(const V: TDateVector): TDateVector;  function VUnique**Time**(const V: TTimeVector): TTimeVector;  function VUnique(const V: TColorVector): TColorVector; | VUnique(V)– возвращает вектор уникальных значений элементов вектора V (удаление дублей)  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VUnion(const V1,V2: TIntVector): TIntVector;  function VUnion(const V1,V2: TInt64Vector): TInt64Vector;  function VUnion(const V1,V2: TStrVector): TStrVector;  function VUnion(const V1,V2: TDateVector): TDateVector;  function VUnion(const V1,V2: TColorVector): TColorVector; | VUnion(V1,V2) – операция объединения векторов – возвращает результирующий вектор |  |
| function VMin(const V: TIntVector): Integer;  function VMin(const V: TInt64Vector): Int64;  function VMin(const V: TStrVector; const ACaseSensitivity: Boolean = True): String;  function VMin**Date**(const V: TDateVector): TDate;  function VMin**Time**(const V: TTimeVector): TTime; | VMin(V)– возвращает наименьшее значение из вектора V  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VMax(const V: TIntVector): Integer;  function VMax(const V: TInt64Vector): Int64;  function VMax(const V: TStrVector; const ACaseSensitivity: Boolean = True): String;  function VMax**Date**(const V: TDateVector): TDate;  function VMax**Time**(const V: TTimeVector): TTime; | VMax(V)– возвращает наибольшее значение из вектора V  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VCountIf(const V: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIf(const V: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIf(const V: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Integer;  function VCountIf**Date**(const V: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIf**Time**(const V: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIf(const V: TBoolVector; const IfValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIf(const V: TColorVector; const IfValue: TColor;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer; | VCountIf(V,IfValue,FromIndex,ToIndex) – возвращает количество элементов вектора V, расположенных от индекса FromIndex до индекса ToIndex, значение которых равно IfValue.  Если ToIndex не указан – подсчет идет от FromIndex до конца вектора.  Если не указаны оба индекса – подсчет по всему вектору целиком.  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VCountIfNot(const V: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIfNot(const V: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIfNot(const V: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Integer;  function VCountIfNot**Date**(const V: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIfNot**Time**(const V: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIfNot(const V: TBoolVector;const IfValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VCountIfNot(const V: TColorVector;const IfValue: TColor;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer; | VCountIfNot(V,IfValue,FromIndex,ToIndex) – возвращает количество элементов вектора V, расположенных от индекса FromIndex до индекса ToIndex, значение которых НЕ равно IfValue.  Если ToIndex не указан – подсчет идет от FromIndex до конца вектора.  Если не указаны оба индекса – подсчет по всему вектору целиком.  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VIntToStr(const V: TIntVector; const ZeroIsEmpty: Boolean = False): TStrVector;  function VIntToStr(const V: TInt64Vector; const ZeroIsEmpty: Boolean = False): TStrVector;  function VFloatToStr(const V: TDblVector): TStrVector;  function VDateToStr(const V: TDateVector; const BoundaryIsEmpty: Boolean = False): TStrVector;  function VTimeToStr(const V: TTimeVector): TStrVector;  function VBoolToStr(const V: TBoolVector): TStrVector; | VXXXToStr(V) – преобразование к строковому вектору.  Применяет к каждому элементу вектора V соответствующую функцию XXXToStr и возвращает вектор получившихся значений.  VIntToStr вставляет в результирующий вектор пустые строки для нулевых значений вектора V, если ZeroIsEmpty=True  VDateToStr вставляет в результирующий вектор пустые строки для граничных (DK\_DateUtils.IsBoundaryDate) значений вектора V, если BoundaryIsEmpty=True  VBoolToStr – замена True->STR\_TRUE,  False->STR\_FALSE из DK\_Const |  |
| function VFormatDateTime(const FormatStr: String; const V: TDblVector;  const BoundaryIsEmpty: Boolean = False;  Options: TFormatDateTimeOptions = []): TStrVector; | VFormatDateTime – формат даты – применяет функцию DateUtils.FormatDateTime к каждому элементу вектора V и возвращает вектор получившихся значений.  Вставляет в результирующий вектор пустые строки для граничных (DK\_DateUtils.IsBoundaryDate) значений вектора V, если BoundaryIsEmpty=True |  |
| function VTrim(const V: TStrVector): TStrVector; | VTrim – применяет функцию DK\_StrUtils.STrim к каждому элементу вектора V и возвращает вектор получившихся значений. |  |
| function VSum(const V1,V2: TIntVector) : TIntVector;  function VSum(const V1,V2: TInt64Vector): TInt64Vector;  function VSum(const V1,V2: TStrVector): TStrVector;  function VSum(const V1,V2: TDblVector): TDblVector; | VSum(V1,V2)– сложение (а для строк – конкатенация) соответствующих элементов векторов V1 и V2 равной длины – возвращает вектор полученных значений. Если векторы не одинаковые по длине возвращает Nil |  |
| function VSum(const V: TIntVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSum(const V: TInt64Vector; FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSum(const V: TStrVector; FromIndex: Integer=-1; ToIndex: Integer=-1): String;  function VSum(const V: TDblVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Double; | VSum(V,FromIndex,ToIndex) – возвращает результат сложения (а для строк – конкатенации) всех элементов вектора V с индекса FromIndex до индекса ToIndex (включительно).  Если ToIndex не указан – суммирование (конкатенация) идет от FromIndex до конца вектора.  Если не указаны оба индекса – суммирование (конкатенация) по всему вектору целиком.  Если вектор пустой возвращает 0 (а для строк EmptyStr) |  |
| function VSumIf(const V: TIntVector; const IfVector: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIf(const V: TIntVector; const IfVector: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIf(const V: TIntVector; const IfVector: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Integer;  function VSumIf**Date**(const V: TIntVector; const IfVector: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIf**Time**(const V: TIntVector; const IfVector: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIf(const V: TIntVector; const IfVector: TBoolVector; const IfValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIf(const V: TInt64Vector; const IfVector: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIf(const V: TInt64Vector; const IfVector: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIf(const V: TInt64Vector; const IfVector: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Int64;  function VSumIf**Date**(const V: TInt64Vector; const IfVector: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIf**Time**(const V: TInt64Vector; const IfVector: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIf(const V: TInt64Vector; const IfVector: TBoolVector; const IfValue: Boolean;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64; | VSumIf(V,IfVector,IfValue,FromIndex,ToIndex)  суммирование с условием – возвращает результат сложения всех элементов вектора V с индекса FromIndex до индекса ToIndex (включительно) при условии, если соответствующий элемент IfVector вектора равен IfValue.  Если ToIndex не указан – суммирование (конкатенация) идет от FromIndex до конца вектора.  Если не указаны оба индекса – суммирование по всему вектору целиком.  Если вектор пустой или условии не выполняется возвращает 0  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VSumIfNot(const V: TIntVector; const IfVector: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIfNot(const V: TIntVector; const IfVector: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIfNot(const V: TIntVector; const IfVector: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Integer;  function VSumIfNot**Date**(const V: TIntVector; const IfVector: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIfNot**Time**(const V: TIntVector; const IfVector: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Integer;  function VSumIfNot(const V: TInt64Vector; const IfVector: TIntVector; const IfValue: Integer;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIfNot(const V: TInt64Vector; const IfVector: TInt64Vector; const IfValue: Int64;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIfNot(const V: TInt64Vector; const IfVector: TStrVector; const IfValue: String;  FromIndex: Integer=-1; ToIndex: Integer=-1;  const ACaseSensitivity: Boolean = True): Int64;  function VSumIfNot**Date**(const V: TInt64Vector; const IfVector: TDateVector; const IfValue: TDate;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64;  function VSumIfNot**Time**(const V: TInt64Vector; const IfVector: TTimeVector; const IfValue: TTime;  FromIndex: Integer=-1; ToIndex: Integer=-1): Int64; | VSumIfNot(V,IfVector,IfValue,FromIndex,ToIndex)  суммирование с условием – возвращает результат сложения всех элементов вектора V с индекса FromIndex до индекса ToIndex (включительно) при условии, если соответствующий элемент IfVector вектора НЕ равен IfValue.  Если ToIndex не указан – суммирование (конкатенация) идет от FromIndex до конца вектора.  Если не указаны оба индекса – суммирование по всему вектору целиком.  Если вектор пустой или условии не выполняется возвращает 0  Для строкового вектора параметр ACaseSensitivity определяет чувствительность к регистру |  |
| function VMult(const V1,V2: TIntVector): TIntVector;  function VMult(const V1,V2: TInt64Vector): TInt64Vector; | VMult(V1,V2)– произведение соответствующих элементов векторов V1 и V2 равной длины – возвращает вектор полученных значений. Если векторы не одинаковые по длине возвращает Nil |  |
| function VMult(const V1: TStrVector; const V2: TIntVector): TStrVector; | VMult(V1,V2) – «произведение» соответствующих элементов строкового вектора V1 и целочисленного вектора V2 равной длины. Если V2[i]=0 , то результат =EmptyStr, если V2[i]<>0, то результат =V1[i] .  Возвращает вектор полученных значений. Если векторы не одинаковые по длине возвращает Nil |  |
| function VCrossInd(const V: TDateVector; const ABeginDate, AEndDate: TDate;  out I1,I2: Integer): Boolean; | VCrossInd(V,ABeginDate,AEndDate,I1,I2)- определяет, пересекается ли упорядоченный вектор дат V и период с ABeginDate по AEndDate (включительно). Если пересекается, то записывает в I1,I2 соответствующие индексы начального и конечного значений V, принадлежащих периоду, и возвращает True, в противном случае I1=I2=-1 и возвращает False. |  |
| function VCountIn(const V: TDateVector; const ABeginDate, AEndDate: TDate): Integer; | VCountIn(V,ABeginDate,AEndDate) - возвращает кол-во значений упорядоченного вектора дат V, принадлежащих периоду с ABeginDate по AEndDate (включительно) |  |
| function VCountBefore(const V: TDateVector; const ADate: TDate): Integer; | VCountBefore(V, ADate) - количество элементов упорядоченного вектора дат V, значения которых меньше (строго) даты ADate |  |
| function VCountAfter(const V: TDateVector; const ADate: TDate): Integer; | VCountAfter(V, ADate) - количество элементов упорядоченного вектора дат V, значения которых больше (строго) даты ADate |  |
| function VOrder(const MaxValue: Integer; const AZeroFirst: Boolean = False): TIntVector; | VOrder(MaxValue) - возвращает вектор {1,2,3,…, MaxValue}, если AZeroFirst=False  {0,2,3,…, MaxValue}, если AZeroFirst=True |  |
| function VRange(const AFirstValue, ALastValue: Integer): TIntVector; | VRange(AFirstValue, ALastValue) - возвращает вектор  {AFirstValue, AFirstValue+1,…, ALastValue} |  |
| procedure VSort(const V: TStrVector; out Indexes: TIntVector; const Desc: Boolean = False);  procedure VSort(const V: TIntVector; out Indexes: TIntVector; const Desc: Boolean = False);  procedure VSort(const V: TInt64Vector; out Indexes: TIntVector; const Desc: Boolean = False);  procedure VSort**Date**(const V: TDateVector; out Indexes: TIntVector; const Desc: Boolean = False);  procedure VSort**Time**(const V: TTimeVector; out Indexes: TIntVector; const Desc: Boolean = False); | VSort(V,Indexes,Desc) – записывает в вектор Indexes индексы элементов вектора V, упорядоченных по возрастанию при Desc=False (или по убыванию при Desc=True) |  |
| procedure VSort(var V: TStrVector; const Desc: Boolean = False);  procedure VSort(var V: TIntVector; const Desc: Boolean = False);  procedure VSort(var V: TInt64Vector; const Desc: Boolean = False);  procedure VSort**Date**(var V: TDateVector; const Desc: Boolean = False);  procedure VSort**Time**(var V: TTimeVector; const Desc: Boolean = False); | VSort(V,Desc) – сортирует элементы вектора V по возрастанию при Desc=False (или по убыванию при Desc=True) |  |
| procedure VReplace(var V: TStrVector; const Indexes: TIntVector);  procedure VReplace(var V: TIntVector; const Indexes: TIntVector);  procedure VReplace(var V: TInt64Vector; const Indexes: TIntVector);  procedure VReplace(var V: TDblVector; const Indexes: TIntVector); | VReplace(V,Indexes) – переставляет элементы вектора V в соответствии с индексами, записанными в векторе Indexes |  |
| function VIsTrue(const V: TBoolVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Boolean; | VIsTrue(V) – если вектор V c индекса FromIndex до ToIndex (включительно) содержит хотя бы одно значение True, то возвращает True иначе False.  Если ToIndex не указан – проверка идет от FromIndex до конца вектора.  Если не указаны оба индекса – проверка по всему вектору целиком. |  |
| function VIsFalse(const V: TBoolVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Boolean; | VIsFalse(V) – если вектор V c индекса FromIndex до ToIndex (включительно) содержит хотя бы одно значение False, то возвращает True иначе False.  Если ToIndex не указан – проверка идет от FromIndex до конца вектора.  Если не указаны оба индекса – проверка по всему вектору целиком. |  |
| function VIsAllTrue(const V: TBoolVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Boolean; | VIsAllTrue(V) – если все элементы вектора V c индекса FromIndex до ToIndex (включительно) имеют значение True, то возвращает True иначе False.  Если ToIndex не указан – проверка идет от FromIndex до конца вектора.  Если не указаны оба индекса – проверка по всему вектору целиком. |  |
| function VIsAllFalse(const V: TBoolVector; FromIndex: Integer=-1; ToIndex: Integer=-1): Boolean; | VIsAllFalse(V) – если все элементы вектора V c индекса FromIndex до ToIndex (включительно) имеют значение False, то возвращает True иначе False.  Если ToIndex не указан – проверка идет от FromIndex до конца вектора.  Если не указаны оба индекса – проверка по всему вектору целиком. |  |
| procedure VReverse(const V: TIntVector);  procedure VReverse(const V: TInt64Vector);  procedure VReverse(const V: TDblVector);  procedure VReverse(const V: TStrVector); | VReverse(V) – меняет порядок элементов вектора на противовположный |  |
| function VNameLong(const AFs, ANs, APs: TStrVector): TStrVector; | VNameLong – аналог SNameLong для векторов |  |
| function VNameShort(const AFs, ANs, APs: TStrVector): TStrVector; | VNameShort – аналог SNameShort для векторов |  |

# **DK\_Matrix**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const, DK\_Vector**

Работа с матрицами

type

TIntMatrix = array of TIntVector;

TInt64Matrix = array of TInt64Vector;

TStrMatrix = array of TStrVector;

TDblMatrix = array of TDblVector;

TDateMatrix = type TDblMatrix;

TTimeMatrix = type TDblMatrix;

TBoolMatrix = array of TBoolVector;

TIntMatrix3D = array of TIntMatrix;

TInt64Matrix3D = array of TInt64Matrix;

TStrMatrix3D = array of TStrMatrix;

TDblMatrix3D = array of TDblMatrix;

TDateMatrix3D = type TDblMatrix3D;

TTimeMatrix3D = type TDblMatrix3D;

TBoolMatrix3D = array of TBoolMatrix;

|  |  |  |
| --- | --- | --- |
| procedure MDim(var M: TIntMatrix; const Size1, Size2: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure MDim(var M: TInt64Matrix; const Size1, Size2: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure MDim(var M: TStrMatrix; const Size1, Size2: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure MDim(var M: TDblMatrix; const Size1, Size2: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure MDim(var M: TBoolMatrix; const Size1, Size2: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE); | MDim(M, Size1, Size2, DefaultValue) - cоздает матрицу M размерностью Size1 х Size2, заполненную значениями DefaultValue. Если в M что-то было - информация теряется |  |
| procedure MDim(var M: TIntMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure MDim(var M: TInt64Matrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure MDim(var M: TStrMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure MDim(var M: TDblMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure MDim(var M: TBoolMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE); | MDim(M, Size1, Size2, Size3, DefaultValue) - cоздает 3D матрицу M размерностью Size1 х Size2 х Size3, заполненную значениями DefaultValue. Если в M что-то было - информация теряется |  |
| procedure MDim(var M: TIntMatrix; const Size: Integer; const DefaultVector: TIntVector = nil);  procedure MDim(var M: TInt64Matrix; const Size: Integer; const DefaultVector: TInt64Vector = nil);  procedure MDim(var M: TStrMatrix; const Size: Integer; const DefaultVector: TStrVector = nil);  procedure MDim(var M: TDblMatrix; const Size: Integer; const DefaultVector: TDateVector = nil);  procedure MDim(var M: TBoolMatrix; const Size: Integer; const DefaultVector: TBoolVector = nil); | MDim(M, Size, DefaultVector) - cоздает матрицу M в виде вектора размером Size, каждым элементом которого является вектор DefaultVector. То есть матрица представляется в виде вектора векторов. Если в M что-то было - информация теряется |  |
| procedure MDim(var M: TIntMatrix3D; const Size: Integer; const DefaultMatrix: TIntMatrix = nil);  procedure MDim(var M: TInt64Matrix3D; const Size: Integer; const DefaultMatrix: TInt64Matrix = nil);  procedure MDim(var M: TStrMatrix3D; const Size: Integer; const DefaultMatrix: TStrMatrix = nil);  procedure MDim(var M: TDblMatrix3D; const Size: Integer; const DefaultMatrix: TDblMatrix = nil);  procedure MDim(var M: TBoolMatrix3D; const Size: Integer; const DefaultMatrix: TBoolMatrix = nil); | MDim(M, Size, DefaultMatrix) - cоздает 3D матрицу M в виде вектора размером Size, каждым элементом которого является матрица DefaultMatrix. То есть 3D матрица представляется в виде вектора матриц. Если в M что-то было - информация теряется |  |
| procedure MReDim(var M: TIntMatrix; const Size1, Size2: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure MReDim(var M: TInt64Matrix; const Size1, Size2: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure MReDim(var M: TStrMatrix; const Size1, Size2: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure MReDim(var M: TDblMatrix; const Size1, Size2: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure MReDim(var M: TBoolMatrix; const Size1, Size2: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE); | MReDim(M, Size1, Size2, DefaultValue) - изменяет размерность матрицы M до Size1 х Size2. Если новая размерность меньше исходной, матрица усекается, усеченные элементы теряются. Если новая размерность больше исходной, матрица дополняется элементами  со значениями DefaultValue; оставшиеся элементы не изменяются. |  |
| procedure MReDim(var M: TIntMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Integer = VECTOR\_INT\_DEFAULT\_VALUE);  procedure MReDim(var M: TInt64Matrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Int64 = VECTOR\_INT64\_DEFAULT\_VALUE);  procedure MReDim(var M: TStrMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: String = VECTOR\_STR\_DEFAULT\_VALUE);  procedure MReDim(var M: TDblMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Double = VECTOR\_DBL\_DEFAULT\_VALUE);  procedure MReDim(var M: TBoolMatrix3D; const Size1, Size2, Size3: Integer;  const DefaultValue: Boolean = VECTOR\_BOOL\_DEFAULT\_VALUE); | MReDim(M, Size1, Size2, Size3, DefaultValue) - изменяет размерность 3D матрицы M до Size1 х Size2 х Size3. Если новая размерность меньше исходной, матрица усекается, усеченные элементы теряются. Если новая размерность больше исходной, матрица дополняется элементами  со значениями DefaultValue; оставшиеся элементы не изменяются. |  |
| procedure MReDim(var M: TIntMatrix; const Size: Integer; const DefaultVector: TIntVector = nil);  procedure MReDim(var M: TInt64Matrix; const Size: Integer; const DefaultVector: TInt64Vector = nil);  procedure MReDim(var M: TStrMatrix; const Size: Integer; const DefaultVector: TStrVector = nil);  procedure MReDim(var M: TDblMatrix; const Size: Integer; const DefaultVector: TDateVector = nil);  procedure MReDim(var M: TBoolMatrix; const Size: Integer; const DefaultVector: TBoolVector = nil); | MReDim(M, Size, DefaultVector) - изменяет количество векторов, входящих в матрицу M на Size. Если новое количество меньше исходного, матрица усекается, усеченные элементы теряются. Если новое количество больше исходного – матрица дополняется векторами равными DefaultVector |  |
| procedure MReDim(var M: TIntMatrix3D; const Size: Integer; const DefaultMatrix: TIntMatrix = nil);  procedure MReDim(var M: TInt64Matrix3D; const Size: Integer; const DefaultMatrix: TInt64Matrix = nil);  procedure MReDim(var M: TStrMatrix3D; const Size: Integer; const DefaultMatrix: TStrMatrix = nil);  procedure MReDim(var M: TDblMatrix3D; const Size: Integer; const DefaultMatrix: TDblMatrix = nil);  procedure MReDim(var M: TBoolMatrix3D; const Size: Integer; const DefaultMatrix: TBoolMatrix = nil); | MReDim(M, Size, DefaultMatrix) - изменяет количество матриц, входящих в 3D матрицу M на Size. Если новое количество меньше исходного, матрица усекается, усеченные элементы теряются. Если новое количество больше исходного – матрица дополняется матрицами равными DefaultMatrix |  |
| procedure MSwap(var M: TIntMatrix; const Index1, Index2: Integer);  procedure MSwap(var M: TInt64Matrix; const Index1, Index2: Integer);  procedure MSwap(var M: TStrMatrix; const Index1, Index2: Integer);  procedure MSwap(var M: TDblMatrix; const Index1, Index2: Integer);  procedure MSwap(var M: TBoolMatrix; const Index1, Index2: Integer); | MSwap(M, Index1, Index2) - меняет в матрице M местами вектора с индексами Index1, Index2 |  |
| procedure MSwap(var M: TIntMatrix3D; const Index1, Index2: Integer);  procedure MSwap(var M: TInt64Matrix3D; const Index1, Index2: Integer);  procedure MSwap(var M: TStrMatrix3D; const Index1, Index2: Integer);  procedure MSwap(var M: TDblMatrix3D; const Index1, Index2: Integer);  procedure MSwap(var M: TBoolMatrix3D; const Index1, Index2: Integer); | MSwap(M, Index1, Index2) - меняет в 3D матрице M местами матицы с индексами Index1, Index2 |  |
| procedure MDel(var M: TIntMatrix; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TInt64Matrix; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TStrMatrix; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TDblMatrix; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TBoolMatrix; const Index1: Integer; Index2: Integer = -1); | MDel(M, Index1, Index2) - удаляет из матрицы M вектора с индекса Index1 до индекса Index2 включительно. Если Index2 не указан, то удаляется один вектор с индексом Index1 |  |
| procedure MDel(var M: TIntMatrix3D; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TInt64Matrix3D; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TStrMatrix3D; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TDblMatrix3D; const Index1: Integer; Index2: Integer = -1);  procedure MDel(var M: TBoolMatrix3D; const Index1: Integer; Index2: Integer = -1); | MDel(M, Index1, Index2) - удаляет из 3D матрицы M матрицы с индекса Index1 до индекса Index2 включительно. Если Index2 не указан, то удаляется одна матрица с индексом Index1 |  |
| function MCut(const M: TIntMatrix; FromIndex: Integer=-1; ToIndex: Integer=-1): TIntMatrix;  function MCut(const M: TInt64Matrix; FromIndex: Integer=-1; ToIndex: Integer=-1): TInt64Matrix;  function MCut(const M: TStrMatrix; FromIndex: Integer=-1; ToIndex: Integer=-1): TStrMatrix;  function MCut(const M: TDblMatrix; FromIndex: Integer=-1; ToIndex: Integer=-1): TDblMatrix;  function MCut(const M: TBoolMatrix; FromIndex: Integer=-1; ToIndex: Integer=-1): TBoolMatrix; | Возвращает срез матрицы M, в который входят вектора с индекса FromIndex до индекса ToIndex (включительно). Если FromIndex и ToIndex не указаны, возвращается копия матрицы M целиком. |  |
| function MCut(const M: TIntMatrix3D; FromIndex: Integer=-1; ToIndex: Integer=-1): TIntMatrix3D;  function MCut(const M: TInt64Matrix3D; FromIndex: Integer=-1; ToIndex: Integer=-1): TInt64Matrix3D;  function MCut(const M: TStrMatrix3D; FromIndex: Integer=-1; ToIndex: Integer=-1): TStrMatrix3D;  function MCut(const M: TDblMatrix3D; FromIndex: Integer=-1; ToIndex: Integer=-1): TDblMatrix3D;  function MCut(const M: TBoolMatrix3D; FromIndex: Integer=-1; ToIndex: Integer=-1): TBoolMatrix3D; | Возвращает срез 3D матрицы M, в который входят матрицы с индекса FromIndex до индекса ToIndex (включительно). Если FromIndex и ToIndex не указаны, возвращается копия 3D матрицы M целиком. |  |
| function MAdd(var M1,M2: TIntMatrix) : TIntMatrix;  function MAdd(var M1,M2: TInt64Matrix): TInt64Matrix;  function MAdd(var M1,M2: TStrMatrix) : TStrMatrix;  function MAdd(var M1,M2: TDblMatrix) : TDblMatrix;  function MAdd(var M1,M2: TBoolMatrix) : TBoolMatrix; | Сцепляет (добавляет векторы матрицы M2 после векторов матрицы M1) матрицы M1и M2 и возвращает полученную матрицу |  |
| function MAdd(var M1,M2: TIntMatrix3D) : TIntMatrix3D;  function MAdd(var M1,M2: TInt64Matrix3D): TInt64Matrix3D;  function MAdd(var M1,M2: TStrMatrix3D) : TStrMatrix3D;  function MAdd(var M1,M2: TDblMatrix3D) : TDblMatrix3D;  function MAdd(var M1,M2: TBoolMatrix3D) : TBoolMatrix3D; | Сцепляет 3D матрицы M1и M2 и возвращает полученную 3D матрицу |  |
| procedure MAppend(var M: TIntMatrix; const V: TIntVector);  procedure MAppend(var M: TInt64Matrix; const V: TInt64Vector);  procedure MAppend(var M: TStrMatrix; const V: TStrVector);  procedure MAppend(var M: TDblMatrix; const V: TDblVector);  procedure MAppend(var M: TBoolMatrix; const V: TBoolVector); | Добавляет вектор V после векторов матрицы M |  |
| procedure MAppend(var M: TIntMatrix3D; const MValue: TIntMatrix);  procedure MAppend(var M: TInt64Matrix3D; const MValue: TInt64Matrix);  procedure MAppend(var M: TStrMatrix3D; const MValue: TStrMatrix);  procedure MAppend(var M: TDblMatrix3D; const MValue: TDblMatrix);  procedure MAppend(var M: TBoolMatrix3D; const MValue: TBoolMatrix); | Добавляет матрицу MValue последним элементом 3D матрицы M |  |
| function MIndexOf(const M: TIntMatrix; const FindValue: Integer;  out Index1, Index2: Integer): Boolean;  function MIndexOf(const M: TInt64Matrix; const FindValue: Int64;  out Index1, Index2: Integer): Boolean;  function MIndexOf(const M: TStrMatrix; const FindValue: String;  out Index1, Index2: Integer): Boolean;  function MIndexOf**Date**(const M: TDateMatrix; const FindValue: TDate;  out Index1, Index2: Integer): Boolean;  function MIndexOf**Time**(const M: TTimeMatrix; const FindValue: TTime;  out Index1, Index2: Integer): Boolean;  function MIndexOf(const M: TBoolMatrix; const FindValue: Boolean;  out Index1, Index2: Integer): Boolean; | Поиск в матрице M элемента со значением FindValue. Если такой элемент найден, то его индексы записываются в Index1 и Index2, а функция возвращает True. Если такого элемента нет, то Index1= Index2 =-1 , а функция возвращает False. |  |
| function MIndexOf(const M: TIntMatrix3D; const FindValue: Integer;  out Index1, Index2, Index3: Integer): Boolean;  function MIndexOf(const M: TInt64Matrix3D; const FindValue: Int64;  out Index1, Index2, Index3: Integer): Boolean;  function MIndexOf(const M: TStrMatrix3D; const FindValue: String;  out Index1, Index2, Index3: Integer): Boolean;  function MIndexOf**Date**(const M: TDateMatrix3D; const FindValue: TDate;  out Index1, Index2, Index3: Integer): Boolean;  function MIndexOf**Time**(const M: TTimeMatrix3D; const FindValue: TTime;  out Index1, Index2, Index3: Integer): Boolean;  function MIndexOf(const M: TBoolMatrix3D; const FindValue: Boolean;  out Index1, Index2, Index3: Integer): Boolean; | Поиск в 3D матрице M элемента со значением FindValue. Если такой элемент найден, то его индексы записываются в Index1, Index2, Index3, а функция возвращает True. Если такого элемента нет, то Index1=Index2=Index3 =-1 , а функция возвращает False. |  |
| function MToVector(const M: TIntMatrix): TIntVector;  function MToVector(const M: TInt64Matrix): TInt64Vector;  function MToVector(const M: TStrMatrix): TStrVector;  function MToVector(const M: TDblMatrix): TDblVector;  function MToVector(const M: TBoolMatrix): TBoolVector; | Возвращает вектор, составленный из элементов векторов матрицы M, записанных друг за другом |  |
| function MToVector(const M: TIntMatrix; const Used: TBoolMatrix): TIntVector;  function MToVector(const M: TInt64Matrix; const Used: TBoolMatrix): TInt64Vector;  function MToVector(const M: TStrMatrix; const Used: TBoolMatrix): TStrVector;  function MToVector(const M: TDblMatrix; const Used: TBoolMatrix): TDblVector;  function MToVector(const M: TBoolMatrix; const Used: TBoolMatrix): TBoolVector; | Возвращает вектор, составленный из элементов векторов матрицы M, записанных друг за другом. Условие вхождения элемента определяется соответствующим значением матрицы Used |  |
| function MMin(const M: TIntMatrix) : Integer;  function MMin(const M: TInt64Matrix): Int64;  function MMin(const M: TStrMatrix) : String;  function MMin**Date**(const M: TDateMatrix) : TDate;  function MMin**Time**(const M: TTimeMatrix) : TTime; | MMin(M) - возвращает наименьшее значение элементов матрицы M  Если M пустая, возвращает (0,0, EmptyStr, 0) |  |
| function MMax(const M: TIntMatrix) : Integer;  function MMax(const M: TInt64Matrix): Int64;  function MMax(const M: TStrMatrix) : String;  function MMax**Date**(const M: TDateMatrix) : TDate;  function MMax**Time**(const M: TTimeMatrix) : TTime; | MMax(M) - возвращает наибольшее значение элементов матрицы M  Если M пустая, возвращает (0,0, EmptyStr, 0) |  |
| function MIsNil(const M: TIntMatrix): Boolean;  function MIsNil(const M: TInt64Matrix): Boolean;  function MIsNil(const M: TStrMatrix): Boolean;  function MIsNil(const M: TDblMatrix): Boolean;  function MIsNil(const M: TBoolMatrix): Boolean; | MIsNil(M) - возвращает True, если матрица M пустая |  |
| function MIsNil(const M: TIntMatrix3D): Boolean;  function MIsNil(const M: TInt64Matrix3D): Boolean;  function MIsNil(const M: TStrMatrix3D): Boolean;  function MIsNil(const M: TDblMatrix3D): Boolean;  function MIsNil(const M: TBoolMatrix3D): Boolean; | MIsNil(M) - возвращает True, если 3D матрица M пустая |  |
| function MMaxLength(const M: TIntMatrix): Integer;  function MMaxLength(const M: TInt64Matrix): Integer;  function MMaxLength(const M: TStrMatrix): Integer;  function MMaxLength(const M: TDblMatrix): Integer;  function MMaxLength(const M: TBoolMatrix): Integer; | MMaxLength(M) - возвращает размер самого длинного вектора из матрицы M (если представить матрицу в виде вектора векторов) |  |
| function MIntToStr(const M: TIntMatrix; const ZeroIsEmpty: Boolean = False): TStrMatrix;  function MIntToStr(const M: TInt64Matrix; const ZeroIsEmpty: Boolean = False): TStrMatrix;  function MBoolToStr(const M: TBoolMatrix): TStrMatrix;  function MFloatToStr(const M: TDblMatrix): TStrMatrix;  function MDateToStr(const M: TDateMatrix; const BoundaryIsEmpty: Boolean = False): TStrMatrix;  function MTimeToStr(const M: TTimeMatrix): TStrMatrix;  function MIntToStr(const M: TIntMatrix3D; const ZeroIsEmpty: Boolean = False): TStrMatrix3D;  function MIntToStr(const M: TInt64Matrix3D; const ZeroIsEmpty: Boolean = False): TStrMatrix3D;  function MBoolToStr(const M: TBoolMatrix3D): TStrMatrix3D;  function MFloatToStr(const M: TDblMatrix3D): TStrMatrix3D;  function MDateToStr(const M: TDateMatrix3D; const BoundaryIsEmpty: Boolean = False): TStrMatrix3D;  function MTimeToStr(const M: TTimeMatrix3D): TStrMatrix3D; | MXXXToStr(M) – преобразование к строковой матрице.  Применяет к каждому элементу матрицы M соответствующую функцию XXXToStr и возвращает матрицу получившихся значений.  MIntToStr вставляет в результирующую марицу пустые строки для нулевых значений матрицы M, если ZeroIsEmpty=True  MDateToStr вставляет в результирующую марицу пустые строки для граничных (DK\_DateUtils.IsBoundaryDate) значений матрицы M, если BoundaryIsEmpty=True  MBoolToStr – замена True->STR\_TRUE,  False->STR\_FALSE из DK\_Const |  |
| function MFormatDateTime(const FormatStr: String; const M: TDblMatrix;  const BoundaryIsEmpty: Boolean = False; Options: TFormatDateTimeOptions = []): TStrMatrix;    function MFormatDateTime(const FormatStr: String; const M: TDblMatrix3D;  const BoundaryIsEmpty: Boolean = False; Options: TFormatDateTimeOptions = []): TStrMatrix3D; | MFormatDateTime – формат даты – применяет функцию DateUtils.FormatDateTime к каждому элементу матрицы M и возвращает матрицу получившихся значений.  Вставляет в результирующую марицу пустые строки для граничных (DK\_DateUtils.IsBoundaryDate) значений матрицы M, если BoundaryIsEmpty=True |  |
| function MLengths(const M: TIntMatrix): TIntVector;  function MLengths(const M: TInt64Matrix): TIntVector;  function MLengths(const M: TStrMatrix): TIntVector;  function MLengths(const M: TDblMatrix): TIntVector;  function MLengths(const M: TBoolMatrix): TIntVector; | MLengths(M) – возвращает вектор значений длины векторов, входящих в матрицу M |  |

# **DK\_VMShow**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Vector, DK\_Matrix**

Отладочная форма с таблицей для просмотра значений элементов матриц и векторов.

|  |  |  |
| --- | --- | --- |
| procedure VShowAdd(const V: TIntVector; const VName: String = '');  procedure VShowAdd(const V: TInt64Vector; const VName: String = '');  procedure VShowAdd(const V: TStrVector; const VName: String = '');  procedure VShowAdd(const V: TDblVector; const VName: String = '');  procedure VShowAdd**Date**(const V: TDateVector; const VName: String = '');  procedure VShowAdd**Time**(const V: TTimeVector; const VName: String = '');  procedure VShowAdd(const V: TBoolVector; const VName: String = ''); | Добавляет в таблицу столбец с именем VName, в который записываются значения элементов вектора V. С помощью нескольких вызовов VShowAdd формируется таблица с нужными векторами. |  |
| procedure VShow; | Отображает форму с таблицей, сформированной с помощью VShowAdd |  |
| procedure VShow(const V: TIntVector; const VName: String = '');  procedure VShow(const V: TInt64Vector; const VName: String = '');  procedure VShow(const V: TStrVector; const VName: String = '');  procedure VShow(const V: TDblVector; const VName: String = '');  procedure VShow**Date**(const V: TDateVector; const VName: String = '');  procedure VShow**Time**(const V: TTimeVector; const VName: String = '');  procedure VShow(const V: TBoolVector; const VName: String = ''); | Отображает таблицу с одним столбцом с именем VName с элементами одного вектора V. |  |
| procedure MShow(const M: TIntMatrix);  procedure MShow(const M: TInt64Matrix);  procedure MShow(const M: TStrMatrix);  procedure MShow**Date**(const M: TDateMatrix);  procedure MShow**Time**(const M: TTimeMatrix);  procedure MShow(const M: TBoolMatrix); | Отображает таблицу с элементами матрицы M |  |

# **DK\_StrUtils**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const**

Утилиты работы со строками.

TSymbolType = (stUnknown, stPunctuation, stSeparator, stVowel, stConsonant, stSpecial);

|  |  |  |
| --- | --- | --- |
| function SSetUTF8(const AStr: String): String; | УСТАНОВКА КОДИРОВКИ СТРОКИ В UTF8  Входные параметры:  AStr – строка |  |
| function SLength(const AStr: String): Integer; | ДЛИНА СТРОКИ (КОЛИЧЕСТВО СИМВОЛОВ)  Входные параметры:  AStr – строка | SLength(‘строка’)  //6 |
| function SUpper(const AStr: String): String; | ПРЕОБРАЗОВАНИЕ В ВЕРХНИЙ РЕГИСТР  Входные параметры:  AStr – строка | SUpper(‘строка’)  // ‘СТРОКА’ |
| function SLower(const AStr: String): String; | ПРЕОБРАЗОВАНИЕ В НИЖНИЙ РЕГИСТР  Входные параметры:  AStr – строка | SLower(‘СТРОКА’)  //‘строка’ |
| function SFirstUpper(const AStr: String): String; | ПРЕОБРАЗОВАНИЕ В ВЕРХНИЙ РЕГИСТР ПЕРВОГО СИМВОЛА  Входные параметры:  AStr – строка | SFirstUpper(‘строка’)  // ‘Строка’ |
| function SFirstLower(const AStr: String): String; | ПРЕОБРАЗОВАНИЕ В НИЖНИЙ РЕГИСТР ПЕРВОГО СИМВОЛА  Входные параметры:  AStr – строка | SFirstLower(‘СТРОКА’)  //‘сТРОКА’ |
| function STrim(const AStr: String): String; | УДАЛЕНИЕ ПРОБЕЛОВ И УПРАВЛЯЮЩИХ СИМВОЛОВ ИЗ НАЧАЛА И КОНЦА СТРОКИ  Входные параметры:  AStr – строка | STrim(‘ строка ’)  //‘строка’ |
| function SCutRight(const AStr: String;  const ACount: Integer): String; | ВЫРЕЗАЕТ СИМВОЛЫ СПРАВА  Входные параметры:  AStr – строка  ACount – количество отрезаемых символов | SCutRight(‘строка’, 2)  //‘стро’ |
| function SCutLeft(const AStr: String;  const ACount: Integer): String; | ВЫРЕЗАЕТ СИМВОЛЫ СЛЕВА  Входные параметры:  AStr – строка  ACount – количество отрезаемых символов | SCutLeft(‘строка’, 2)  //‘рока’ |
| function SCut(const AStr: String;  const ALeftCount, ARightCount: Integer): String; | ВЫРЕЗАЕТ СИМВОЛЫ СЛЕВА И СПРАВА  Входные параметры:  AStr – строка  ALeftCount – количество отрезаемых символов слева  ARightCount – количество отрезаемых символов справа | SCut(‘строка’, 1, 2)  //‘тро’ |
| function SCompare(const AStr1, AStr2: String;  const ACaseSensitivity: Boolean = True): Integer; | СРАВНЕНИЕ ДВУХ СТРОК  Входные параметры:  AStr1– 1-ая строка  AStr2– 2-ая строка  ACaseSensitivity – чувствительность к регистру  Возвращает:  -1 при AStr1<AStr2  0 при AStr1=AStr2  1 при AStr1>AStr2 | SCompare (‘строка’, ‘строка’)  //0 |
| function SSame(const AStr1, AStr2: String;  const ACaseSensitivity: Boolean = True): Boolean; | ПРОВЕРКА НА ИДЕНТИЧНОСТЬ ДВУХ СТРОК  Входные параметры:  AStr1– 1-ая строка  AStr2– 2-ая строка  ACaseSensitivity – чувствительность к регистру | SSame(‘строка’, ‘строка’)  //true |
| function SEmpty(const AStr: String): Boolean; | ПРОВЕРКА НА ПУСТУЮ СТРОКУ  Входные параметры:  AStr – строка | SEmpty(‘’)  //true |
| function SPos(const AStr, AValue: String;  const AStartPos: Integer = 1): Integer; | ПОИСК ПОДСТРОКИ В СТРОКЕ  Возвращает порядковый номер символа, с которого начинается искомая подстрока в строке или 0, если подстроки нет  Входные параметры:  AStr – строка (где ищем)  AValue – подстрока (что ищем)  AStartPos – порядковый номер символа, с которого начинаем поиск | SPos(‘строка’, ‘ро’)  //3 |
| function SCopy(const AStr: String;  const AStart, AEnd: Integer): String; | КОПИЯ СТРОКИ  Входные параметры:  AStr – строка  AStart – порядковый номер символа, с которого начинается копирование  AEnd – порядковый номер символа, на котором заканчивается копирование | SCopy(‘строка’, 2, 4)  //‘тро’ |
| function SCopyCount(const AStr: String;  const AStart, ACount: Integer): String; | КОПИЯ СТРОКИ  Входные параметры:  AStr – строка  AStart – порядковый номер символа, с которого начинается копирование  ACount – количество копируемых символов | SCopyCount(‘строка’, 2, 3)  //‘тро’ |
| function SDivide(const AStr: String;  const ADivideSymbol: String;  out APart1, APart2: String): Boolean; | РАЗДЕЛЯЕТ СТРОКУ НА ДВЕ ЧАСТИ  Входные параметры:  AStr – строка  ADivideSymbol – символ разделения | SDivide(‘1234/5678’, ‘/’, APart1, APart2)  // APart1= ‘1234’, APart2 = ‘5678’ |
| function SDel(const AStr: String;  const AStart, AEnd: Integer): String; | УДАЛЕНИЕ СТРОКИ  Входные параметры:  AStr – строка  AStart – порядковый номер символа, с которого начинается удаление  AEnd – порядковый номер символа, на котором заканчивается удаление | SDel(‘строка’, 3, 4)  //‘стка’ |
| function SDelCount(const AStr: String;  const AStart, ACount: Integer): String; | УДАЛЕНИЕ СТРОКИ  Входные параметры:  AStr – строка  AStart – порядковый номер символа, с которого начинается удаление  ACount – количество удаляемых символов | SDelCount(‘строка’, 3, 2)  //‘стка’ |
| function SSymbol(const AStr: String;  const ASymbolPos: Integer): String; | ВЫБОР СИМВОЛА ИЗ СТРОКИ  Входные параметры:  AStr – строка  ASymbolPos – порядковый номер символа | SSymbol(‘строка’, 3)  //‘р’ |
| function SSymbolFirst(const AStr: String): String; | ПЕРВЫЙ СИМВОЛ СТРОКИ  Входные параметры:  AStr – строка | SSymbolFirst(‘строка’)  //‘с’ |
| function SSymbolLast(const AStr: String): String; | ПОСЛЕДНИЙ СИМВОЛ СТРОКИ  Входные параметры:  AStr – строка | SSymbolLast(‘строка’)  //‘а’ |
| function SDigit(const AStr: String;  const ASymbolPos: Integer): Byte; | ВЫБОР ЦИФРЫ ИЗ СТРОКИ  Входные параметры:  AStr – строка  ASymbolPos – порядковый номер символа цифры | SDigit(‘54321’, 1)  //5 |
| function SRedLine(const ASpacesCount: Integer): String; | КРАСНАЯ СТРОКА (СТРОКА ПРОБЕЛОВ)  Входные параметры:  ASpacesCount – количество пробелов | SRedLine(1)  //‘ ’ |
| function SFillRight(const AStr: String;  const ANeedLength: Integer): String; | ЗАПОЛНЕНИЕ СТРОКИ ПРОБЕЛАМИ СПРАВА ДО НУЖНОЙ ДЛИНЫ  Входные параметры:  AStr – строка  ANeedLength – нужная длина строки  Если длина строки больше или равна требуемой длине, возвращает исходную строку без изменений | SFillRight(‘стр’, 4)  // ‘стр ’ |
| function SFillLeft(const AStr: String;  const ANeedLength: Integer): String; | ЗАПОЛНЕНИЕ СТРОКИ ПРОБЕЛАМИ СЛЕВА ДО НУЖНОЙ ДЛИНЫ  Входные параметры:  AStr – строка  ANeedLength – нужная длина строки  Если длина строки больше или равна требуемой длине, возвращает исходную строку без изменений | SFillLeft (‘стр’, 4)  // ‘ стр’ |
| function SSymbolType(const ASymbol: String): TSymbolType; | ТИП СИМВОЛА  Входные параметры:  ASymbol– символ | SSymbolType(‘?’)  //stPunctuation |
| function SSymbolType(const AStr: String;  const ASymbolPos: Integer): TSymbolType; | ТИП СИМВОЛА В СТРОКЕ  Входные параметры:  AStr – строка  ASymbolPos – порядковый номер символа | SSymbolType(‘строка’, 1)  //stConsonant |
| function SFont(const AFontName: String; const AFontSize: Single;  const AFontStyle: TFontStyles=[]): TFont; | СОЗДАЕТ ЭКЗЕМПЛЯР КЛАССА TFont  Входные параметры:  AFontName – название шрифта  AFontSize – размер шрифта  AFontStyle – стиль шрифта |  |
| function SWidth(const AStr: String; const AFont: TFont): Integer; | ШИРИНА СТРОКИ (размерность px)  Входные параметры:  AStr – строка  AFont – шрифт |  |
| function SWidth(const AStr, AFontName: String;  const AFontSize: Single;  const AFontStyle: TFontStyles=[]): Integer; | ШИРИНА СТРОКИ (размерность px)  Входные параметры:  AStr – строка  AFontName – название шрифта  AFontSize – размер шрифта  AFontStyle – стиль шрифта |  |
| function SHeight(const AStr: String;  const AFont: TFont): Integer; | ВЫСОТА СТРОКИ (размерность px)  Входные параметры:  AStr – строка  AFont – шрифт |  |
| function SHeight(const AStr, AFontName: String;  const AFontSize: Single;  const AFontStyle: TFontStyles=[]): Integer; | ВЫСОТА СТРОКИ (размерность px)  Входные параметры:  AStr – строка  AFontName – название шрифта  AFontSize – размер шрифта  AFontStyle – стиль шрифта |  |
| function SNameLong(  const AFamily, AName, APatronymic: String): String; |  | SNameLong(‘Фамилия’, ‘Имя’, ‘Отчество’)  // ‘Фамилия Имя Отчество’ |
| function SNameShort(  const AFamily, AName, APatronymic: String): String; |  | SNameShort (‘Фамилия’, ‘Имя’, ‘Отчество’)  // ‘Фамилия И.О.’ |

# **DK\_TextUtils**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Const, DK\_StrUtils, DK\_Vector**

Утилиты работы с текстом.

|  |  |  |
| --- | --- | --- |
| function TextToParts(  const AText, ADelimiter: String): TStrVector; | Разбивает текст AText на части, разделенные ADelimiter . Возвращает вектор этих частей |  |
| function TextToWords(const AText: String): TStrVector; | Разбивает текст AText на слова. Возвращает вектор этих слов. |  |
| function WordToParts(const AWord: String): TStrVector; | Разбивает слово на слоги для возможного переноса на другую строку. Возвращает вектор этих слогов.  Не учитываются слова, не подлежащие переносу (например "узнаю")  Могут быть ошибки в  1. сложносокращенных словах (например "спецодежда"),  2. сложных словах (например "пятиграммовый") |  |
| function TextToCell(var AText: String; const AFont: TFont;  const ACellWidth: Integer;  const ARedLineWidth: Integer = 0;  const AWrapToWordParts: Boolean = False;  const ABreakSymbol: String = SYMBOL\_BREAK): Integer; | Вставляет в текст AText с шрифтом AFont символы разрыва (переноса) строки ABreakSymbol так, чтобы полученные строки умещались в ячейке шириной ACellWidth (размерность в px), а также символы пробела шириной ARedLineWidth (размерность в px) в начало текста (красная строка).  Возвращает требуемую высоту ячейки в px. |  |
| procedure TextToWidth(const AText: String; const AFont: TFont;  const AWidth: Integer;  out ATotalHeight: Integer;  out ARowValues: TStrVector;  const AWrapToWordParts: Boolean = False;  const ARedLineWidth: Integer = 0); | Разбивает в текст AText с шрифтом AFont на части так, чтобы полученные строки умещались в ячейке шириной AWidth (размерность в px), а также символы пробела шириной ARedLineWidth (размерность в px) в начало текста (красная строка).  Записывает требуемую высоту ячейки в px в AHeight, вектор получившихся частей в ARows |  |
| function SurnameNP(  const ASurname, AName, APatronymic: String): String; | Фамилия Имя Отчество -> Фамилия И.О. |  |

# **DK\_PriceUtils**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_StrUtils**

Утилиты работы с ценами.

|  |  |  |
| --- | --- | --- |
| function PriceStrToInt(const APrice: String): Int64; | Преобразование строки с ценой APrice в целочисленный формат | PriceStrToInt('1234,56') // 123456  PriceStrToInt('1 234,56') // 123456 |
| function PriceIntToStr(const APrice: Int64;  const ANeedThousandSeparator: Boolean = False): String; | Преобразование цены APrice из целочисленного формата в строку. | PriceIntToStr(123456, False) // '1234,56'  PriceIntToStr(123456, True) // '1 234,56' |
| function PriceToString(const APrice: Int64;  const ANeedThousandSeparator: Boolean = False): String; | Преобразование цены APrice из целочисленного формата в строку с наименованием денежных единиц | PriceToString(123456, False) // '1234 руб. 56 коп.'  PriceToString(123456, True) // '1 234 руб. 56 коп.' |
| function PriceToText(const APrice: Int64;  const AFirstCapital: Boolean = False): String; | Преобразование цены APrice из целочисленного формата в строку суммы прописью. | PriceToText(123456, False)  // 'одна тысяча двести тридцать четыре рубля 56 копеек'  PriceToText(123456, True)  // 'Одна тысяча двести тридцать четыре рубля 56 копеек' |

# **DK\_SQLUtils**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **DK\_Dialogs, DK\_Const**

Утилиты работы с запросами.

{Query Utils}

procedure QSetQuery(AQuery: TSQLQuery);

procedure QSetSQL(const ASQLText: String);

procedure QShowSQL; //debug util

procedure QOpen;

procedure QExec;

procedure QClose;

procedure QCommit;

procedure QCommitEnd;

procedure QRollback;

procedure QRollbackEnd;

procedure QFirst;

procedure QNext;

procedure QGotoBookmark(ABookmark: TBookmark);

procedure QDisableControls;

procedure QEnableControls;

procedure QParamInt(const AParamName: String; const AParamValue: Integer);

procedure QParamInt64(const AParamName: String; const AParamValue: Int64);

procedure QParamStr(const AParamName: String; const AParamValue: String);

procedure QParamDT(const AParamName: String; const AParamValue: TDateTime);

procedure QParamsInt(const AParamValues: array of Integer);

procedure QParamsInt64(const AParamValues: array of Int64);

procedure QParamsStr(const AParamValues: array of String);

procedure QParamsDT(const AParamValues: array of TDateTime);

procedure QParamsFloat(const AParamValues: array of Double);

function QIsNull(const AFieldName: String): Boolean;

function QIsEmpty: Boolean;

function QEOF: Boolean;

function QGetBookmark: TBookmark;

function QFieldInt(const AFieldName: String): Integer;

function QFieldInt64(const AFieldName: String): Int64;

function QFieldStr(const AFieldName: String): String;

function QFieldDT(const AFieldName: String): TDateTime;

|  |  |  |
| --- | --- | --- |
| function SqlSpaces(const AStr: String): String; | Добавляет в начало и конец строки AStr пробел | SqlSpaces('ABCD') //' ABCD ' |
| function SqlBrackets(const AStr: String): String; | Добавляет в начало и конец строки AStr скобки и пробел | SqlBrackets('ABCD') //' (ABCD) ' |
| function SqlExprLogic(  const ALeftArg, AAssertion, ARightArg: String): String; | Собирает логическое выражение | SqlExprLogic('A', '<', 'B') //'A<B' |
| function SqlOR(  const AExpressions: array of String): String; | Собирает логическое выражение | SqlOR(['A<B', 'A>C', 'A=D'])  //' (A<B) OR (A>C) OR (A=D) ' |
| function SqlAND(  const AExpressions: array of String): String; | Собирает логическое выражение | SqlAND(['A<B', 'A>C', 'A=D'])  //' (A<B) AND (A>C) AND (A=D) ' |
| function SqlCROSS(  const AMin1, AMax1, AMin2, AMax2: String): String; | Собирает логическое выражение – пересечение интервалов  [AMin1, AMax1] и [Amin2, Amax2] | SqlCROSS('BeginDate', 'EndDate', '"10.01.2021"', '"20.12.2021"')  //' ( (BeginDate <= "10.01.2021") AND (EndDate >= "20.12.2021") ) OR ( (BeginDate <= "10.01.2021") AND (EndDate >= "10.01.2021") ) OR ( (BeginDate <= "20.12.2021") AND (EndDate >= "20.12.2021") ) OR ( (BeginDate >= "10.01.2021") AND (EndDate <= "20.12.2021") ) ' |
| function SqlINSERT(const ATableName: String;  const AFields: array of String;  const AORExpression: String): String; |  | SqlINSERT('TableName', ['Field1', 'Field2', 'Field3'])  //'INSERT INTO TableName ( Field1, Field2, Field3 ) VALUES (:Field1, :Field2, :Field3) '  SqlINSERT('TableName', ['Field1', 'Field2', 'Field3'], 'REPLACE')  //'INSERT OR REPLACE INTO TableName ( Field1, Field2, Field3 ) VALUES (:Field1, :Field2, :Field3) ' |
| function SqlUPDATE(const ATableName: String;  const AFields: array of String): String; |  | SqlUPDATE('TableName', ['Field1', 'Field2', 'Field3'])  //'UPDATE TableName SET Field1=:Field1, Field2=:Field2, Field3=:Field3 ' |
| function SqlEsc(const AStr: String ;  const ANeedSpaces: Boolean = True): String; |  | Экранирование идентификаторов  Если ANeedSpaces=True, добавляет в начало и конец пробелы |
| function SqlIN(const AFieldName: String;  const AValuesCount: Integer): String; |  | SqlIN(‘FieldName’, 3)  // ‘ (FieldName IN (:Value1, :Value2, :Value3)) ’ |

# **DK\_Fonts**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **Нет**

Утилиты работы с шрифтами.

const

ARIAL\_FONTS: array [0..2] of String =

('Arial', 'Liberation Sans', 'Nimbus Sans L');

TIMES\_FONTS: array [0..2] of String =

('Times New Roman', 'Liberation Serif', 'Nimbus Roman No9 L');

COURIER\_FONTS: array [0..2] of String =

('Courier New', 'Liberation Mono', 'Nimbus Mono L');

type

TFontLike = (flArial, flTimes, flCourier);

|  |  |  |
| --- | --- | --- |
| function FontLikeToName(const AFontLike: TFontLike): String; | Подбор имени шрифта, установленного в системе, из списка  AFontLike=flArial 🡪 ARIAL\_FONTS  AFontLike=flTimes 🡪 TIMES\_FONTS  AFontLike=flCourier 🡪 COURIER\_FONTS  При отсутствии этих шрифтов в системе возвращается имя системного шрифта |  |
| procedure LoadFontFromControl(const AControl: TControl;  out AFontName: String;  out AFontSize: Single); | Получение конкретного имени AFontName (не ‘Default’) и размера шрифта AFontSize из контрола AControl |  |

# **DK\_Math**

ЗАВИСИТ ОТ МОДУЛЕЙ ПАКЕТА: **Нет**

Расчеты.

|  |  |  |
| --- | --- | --- |
| function Min(const AValue1, AValue2: Double): Double;  function Min(const AValue1, AValue2: Integer): Integer; | Меньшее из значений AValue1 и AValue2 |  |
| function Max(const AValue1, AValue2: Double): Double;  function Max(const AValue1, AValue2: Integer): Integer; | Большее из значений AValue1 и AValue2 |  |
| function IsInRange(const AValue, AMinValue, AMaxValue: Double): Boolean; | Проверка AValue ∈ [AMinValue, AMaxValue] |  |
| function IsValidPercent(const APercent: Double): Boolean; | Проверка APercent ∈ [0, 100] |  |
| function Percent(const AValue, APercent: Double): Double; | Процент APercent от числа AValue |  |
| function RandomInRange(const AMinValue, AMaxValue: Integer): Integer; | Случайное число из диапазона [AMinValue, AMaxValue] |  |
| function RandomInRange(const AValue: Integer;  const AMinPercent, AMaxPercent: Double): Integer; | Случайное число из диапазона, границы которого рассчитываются, как процент AMinPercent и AMaxPercent от числа AValue |  |