

Database  
Management  
System **LINTER**®

Version 5.9

Decimal and Tick Library

Relational Expert Systems

---



# Table of Contents

<b>Introduction</b> .....	<b>4</b>
<b>Decimals Library - Fixed Point Decimals</b> .....	<b>5</b>
ADDDECIMAL.....	5
SUBDECIMAL.....	6
CMPDECIMAL.....	6
MULDECIMAL.....	6
DIVDECIMAL.....	7
Permutations.....	7
STRTODEC.....	7
DECTOSTR.....	7
LongToDec.....	8
DecToLong.....	8
DbIToDec.....	9
DecToDbl.....	9
COPYDEC.....	9
Dec Value Status.....	9
GETDECSTATUS.....	9
OKDECSTATUS.....	10
SETDECSTATUS.....	10
ENTDECIMAL.....	10
RNDDECIMAL.....	10
NEGDECIMAL.....	10
SetMaxDecimal and SetMinDecimal .....	11
<b>Tick Library - Dates</b> .....	<b>12</b>
TICKTODATE.....	12
NAMMON.....	13
MONNAM.....	13
NAMDAY.....	14
DAYNAM.....	14
BIGYEAR.....	15
DAYMON.....	15
DAYYEAR.....	15
DAYNUMBERDATE.....	16
DATEDAYNUMBER.....	16
WEEKDAYNUMBER.....	17
STRTOTICK.....	17

TICKTOSTR .....	18
ADDDATE .....	18
SUBMONTHSFROMDATE .....	19
ADDMONTHSTODATE .....	19
SUBDATE .....	19
DATETOTICK .....	20
PUTDIG .....	20
PUT2 .....	20
PUTYEAR .....	21
CMPDATE .....	21
TICKTOSTRF .....	22
STRTOTICKF .....	23
<b>Int64 Library .....</b>	<b>24</b>
Arithmetic functions .....	24
i64_Add .....	24
i64_Sub .....	24
i64_Mul .....	25
i64_Div .....	25
i64_Rest .....	25
i64_DivRest .....	25
i64_AddLong .....	26
i64_SubLong .....	26
i64_MulLong .....	26
i64_DivLong .....	26
i64_RestLong .....	27
Logical Functions .....	27
i64_And .....	27
i64_Or .....	27
Other Functions .....	28
i64_Cmp .....	28
i64_Neg .....	28
i64_IfNeg .....	28
i64_IfZero .....	28
i64_Inc .....	29
i64_Dec .....	29
i64_MINDLONG .....	29
i64_MAXDLONG .....	29
i64_MAXUDLONG .....	29
i64_InvertBytes .....	30
Macros .....	30

## **Introduction**

This document describes Linter's Decimals and Tick libraries. Work with fixed-point decimals and dates is performed via these two library modules.

The Decimals and Tick modules are written in C and therefore may only be used in programs written in C. The Linter package contains two header files for working with decimals and dates: decimals.h and tick.h.

## Decimals Library - Fixed Point Decimals

Because standard data types do not provide the high degree of precision needed, for example, in accounting, Linter introduces a new data type: fixed point decimals being of type DECIMAL.

The main characteristics of a DECIMAL are:

- 1) Precision - total number of digits – 30;
- 2) Scale - number of digits to the right of the decimal point (decimal point) -10;
- 3) These numbers are stored in a special internal format.

The size of a number in this format is 16 bytes. To be able to use DECIMALs in programs written in C, a special library is provided consisting of the decimals.a file and the header file decimals.h.

The phrase decimal value will hereafter be referenced as dec value.

The Decimals library allows the following actions to be performed with values of type DECIMAL:

- Addition/subtraction,
- Multiplication/division,
- Comparison,
- Changing a string of digits into a dec value and vice versa,
- Changing a dec value into a Long and vice versa,
- Changing a dec value into a double and vice versa.



The first byte of the array holding the dec value contains information about the value sign and/or errors including overflow errors. Using C terminology, the contents of this byte may be shown as follows:

```
typedef enum {
    DECZERO, /* 0 = Zero value */
    DECNEGATIV, /* 1 = Negative value */
    DECPOSITIV, /* 2 = Positive value */
    DECNEGOVER, /* 3 = Negative overflow */
    DECPOSOVER, /* 4 = Positive overflow */
    DECERROR /* 5 = Error */
} DECSTATUS;
```

It is necessary to note that all operations with DECIMALs are performed using 30 digits before the decimal point and with a decimal precision of 10. Specifically defining those values only impacts the function of changing a dec value to a string.

In the module description, only those definitions are used that are implemented in the decimals.h header file.

## ADDDECIMAL

### Prototype

```
Void ADDDECIMAL (
    DECIMAL D1, /* 1st value */
    DECIMAL D2, /* 2nd value */
    DECIMAL S); /* Sum */
```

**Description**

The ADDDECIMAL procedure adds two dec values, D1 and D2. The result is returned as S ( $S = D_1 + D_2$ ).

**SUBDECIMAL****Prototype**

```
void SUBDECIMAL (  
    DECIMAL D1,      /* 1st value */  
    DECIMAL D2,      /* 2nd value */  
    DECIMAL D);      /* Result */
```

**Description**

The SUBDECIMAL procedure subtracts one dec value (D2) from another (D1), the result is returned as D ( $D = D_1 - D_2$ ).

**CMPDECIMAL****Prototype**

```
INT CMPDECIMAL (  
    DECIMAL D1,      /* 1st value */  
    DECIMAL D2);     /* 2nd value */
```

**Description**

The CMPDECIMAL function compares two dec values.

The function returns:

- +1, if  $D_1 > D_2$ ;
- 0, if  $D_1 = D_2$ ;
- -1, if  $D_1 < D_2$ .

**Example**

```
Printf('d1 ');  
if ((c =CMPDECIMAL(d1,d2)) == 0)  
    printf('=');  
Else  
    printf('%c' ,(c ==1)? '>':'<');  
printf('d2');
```

**MULDECIMAL****Prototype**

```
Void MULDECIMAL (  
    DECIMAL D1,      /* 1st value */
```

---

```

    DECIMAL D2,      /* 2nd value */
    DECIMAL M);     /* Result */

```

**Description**

The MULDECIMAL procedure multiplies two dec values (D1 and D2). The result is returned as M ( $M = D1 * D2$ ).

**DIVDECIMAL****Prototype**

```

void DIVDECIMAL (
    DECIMAL D1,      /* 1st value */
    DECIMAL D2,      /* 2nd value */
    DECIMAL D);     /* Result */

```

**Description**

The DIVDECIMAL procedure divides D1 by D2. The result is returned as D ( $D = D1/D2$ ).

**Permutations**

All conversions between DECIMAL and other data types return an integer: 1, if successful; zero for an error.

**STRTODEC****Prototype**

```

INT STRTODEC (
    CHAR * Ss,      /*Source string*/
    DECIMAL Od);   /* DECIMAL type variable*/

```

**Description**

The STRTODEC function changes the string in Ss into a dec value, and the result is returned as Od.

**Example**

```

...
STRTODEC (" +3.14" ,pdec);
...

```

**DECTOSTR****Prototype**

```

void DECTOSTR (
    DECIMAL d,      /* Decimal type variable */
    CHAR * Os,      /* String */
    INT Ln,         /* # of digits before decimal*/

```

```

INT Lns,      /* precision */
INT IsSgn); /* Sign necessity */

```

### Description

DECTOSTR changes a dec value into a string of numerals according to the specified format, Ln and Lns in the above Prototype. If IsSgn=1, then the sign will always be displayed, and if IsSgn = zero, then the minus (-) sign will be displayed and the plus (+) sign will be omitted.

### Example

```

...
DECTOSTR(pdec,Str , 20, 2, 1);
/* 30 digits before decimal point, precision 2, sign compulsory! */

```

## LongToDec

### Prototype

```

INT LongToDec (
    LONG Arg,      /* Source long-value */
    DECIMAL Od); /* Resultant dec value */

```

### Description

The LongToDec function changes a LONG value into a DECIMAL. The result is returned as Od.

## DecToLong

### Prototype

```

INT DecToLong (
    DECIMAL Id, /* Source dec value */
    LONG * OI, /* Resultant Long-value */
    INT Round); /* Rounding */

```

### Description

DecToLong changes a dec value, Id, into a LONG value. With rounding, Round = 1; without, Round = 0. The result OI is returned.

With either Round value, OI returns only the part before the decimal point. If Round =1 and the decimal part of Id is non-zero, an error will be generated.

### Example

```

...
if ((ok =DecToLong(vdec,&LStr ,1)) == 0)
    /* Rounding error*/
    { Printf('\n Error: runtime overflow');
      goto lend;
    } /* Ifthen */
...

```

## DbIToDec

### Prototype

```
INT DbIToDec (  
    double ldbl, /* Source double-value */  
    DECIMAL Od); /* Resultant dec value */
```

### Description

DbIToDec changes a DOUBLE value into a dec value.

### Example

```
...  
double d = 1234567890.123456;  
...  
DbIToDec (d ,decvar);  
...
```

## DecToDbI

### Prototype

```
INT DecToDbI (  
    DECIMAL Id, /* Source double-value */  
    double * Odbl); /* Resultant dec value */
```

### Description

The DecToDbI function changes a dec value into a DOUBLE.

## COPYDEC

### Prototype

```
void COPYDEC (  
    DECIMAL Id, /* Source dec value */  
    DECIMAL Od); /* Resultant dec value */
```

### Description

The COPYDEC procedure copies the contents of Id into Od.

## Dec Value Status

## GETDECSTATUS

### Prototype

```
DECSTATUS GETDECSTATUS (  
    DECIMAL D); /* Source decvalue */
```

### Description

GETDECSTATUS returns the status information, contained in the first byte, about the number. See the table preceding the ADDDECIMAL function.

## OKDECSTATUS

### Prototype

```
INT OKDECSTATUS (  
    DECIMAL D); /* Source decvalue */
```

### Description

This function checks a dec value for errors and overflows, 1 is returned if it does not contain errors or overflows; otherwise, 0.

## SETDECSTATUS

### Prototype

```
void SETDECSTATUS (  
    DECSTATUS St, /* New status */  
    DECIMAL D); /* Target dec value*/
```

### Description

SETDECSTATUS sets a new dec value status.

## ENTDECIMAL

### Prototype

```
void ENTDECIMAL (  
    DECIMAL Id, /* Source dec value */  
    DECIMAL Od); /* Resulting dec value */
```

### Description

This procedure truncates the decimal part of Id and returns it in the dec-variable Od.

## RNDDECIMAL

### Prototype

```
void RNDDECIMAL (  
    DECIMAL Id, /* Source dec value */  
    BYTE Prec, /* Precision */  
    BYTE Scale); /* Scale */
```

### Description

This procedure rounds a decimal value, Id, according to the specified format, Prec and Scale.

## NEGDECIMAL

### Prototype

```
void NEGDECIMAL (  
    DECIMAL Id, /* Source dec value */
```

```
DECIMAL Od); /* Resultant dec value */
```

**Description**

The procedure changes the sign of the value to the opposite and places it in the Dec-variable Od.

**SetMaxDecimal and SetMinDecimal****Prototype**

```
void SetMaxDecimal (DECIMAL D);
```

```
void SetMinDecimal (DECIMAL D);
```

**Description**

The SetMaxDecimal and SetMinDecimal procedures set the max and min value of the dec value D.

## Tick Library - Dates

The DATE data type holds information about the current date and time. Linter's smallest unit of time is the Tick, 1/100 of a second. The library automatically adjusts the day-month-year values when a time function moves backwards or forwards across midnight.

Linter has implemented the Tick library for C programmers to enable working with tick level precision in dates. The Tick library includes the tick.a file and the header file tick.h. The main characteristics of the DATE data type are:

- 1) The data is stored in 16 byte variables of type DECIMAL;
- 2) The variable holds the number of ticks from 00 A.D. to the present time;
- 3) The maximum year that may be entered in the date variable is 2099.

The Tick library enables performance of:

- Changing dates from the internal format to various widely used formats and vice versa;
- Defining various date characteristics such as: number of days in a specific month, leap years, etc.;

In the module's descriptions, only those definitions are used that are predefined in the tick.h and decimals.h header files.

Because of Linter's extensive Russian and American connections, some date functions require identifying the language or code page to be used. The variable setlang appears in these functions and has the following options:

<u>Language or Code Set</u>	<u>Value for Variable setlang</u>
Russian ASCII	0
English ASCII	1
KOI8	2
KOI8	3
MS DOS 866	4

There is no default, setlang, must be included each time a date function providing for it is used.

## TICKTODATE

### Prototype

```
void TICKTODATE (
    DECIMAL Date,      /* Source date */
    LONG * days,      /* Returned number of days from 00 A.D. */
    LONG * ticks);    /* Returned number of ticks in the last day */
```

**Description**

The TICKTODATE procedure computes, from 0000 AD, the full number of days and the number of ticks in the last day of the source date, Date, and stores them in the days and ticks variables respectively. This is a timestamp function.

**Example**

```
DECIMAL Date;
LONG    days;
LONG    ticks;
...
TICKTODATE (Date ,&days ,&ticks);
...
```

**NAMMON****Prototype**

```
void NAMMON (
    INT Nmonth,          /* Source month number */
    CHAR * month,       /* Returned month name */
    INT koder);         /* Type of month name coding */
```

**Description**

NAMMON converts the number of the month, Nmonth, an integer between 1 and 12, into the name of the month that is stored in the month variable.

The name of the month will appear in the language represented by the value in setlang. See the table immediately preceding the TICKTODATE function.

Returned value: 1 = success; 0 = error.

**Example**

```
INT    i;
INT    Nmonth;
CHAR   month[8];
...
i = NAMMON (Nmonth ,month,0);
...
```

**MONNAM****Prototype**

```
INT MONNAM (
    CHAR * month,       /*Source month name*/
    INT koder);         /*Type of source name coding*/
```

**Description**

MONNAM converts the name of the month, Month, into the number of the month, an integer between 1 and 12, which is the returned value of the function. To perform a correct conversion,

it is necessary to set setlang for the source month name. See the table immediately preceding the TICKTODATE function.

Returned value: Number of month = successful termination; 0 = error.

### Example

```
INT    i;
CHAR   Month[4];
...
strcpy(Month, "MAR");
i = MONNAM (Month,0);
...
```

## NAMDAY

### Prototype

```
void NAMDAY (
    INT Ndayweek,      /* Source weekday number*/
    CHAR * dayweek,    /* Resultant weekday name */
    INT koder);        /* Type of coding */
```

### Description

The NAMDAY function converts the number of the weekday Ndayweek, an integer between 1 and 7 (Monday = 1), into the name of the weekday that is stored in the dayweek variable. Monday = 1.

The name of the day will appear in the language represented by the value in setlang. See the table immediately preceding the TICKTODATE function.

Returned value: 1, Successful; 0, error.

### Example

```
INT    i;
INT    Ndayweek;
CHAR   dayweek[4];
...
i = NAMDAY (Ndayweek ,dayweek,0);
...
```

## DAYNAM

### Prototype

### Прототип

```
INT DAYNAM (
    CHAR * Dayweek); /* Weekday name */
```

### Description

DAYNAM converts the name of the weekday, Dayweek, into the number of the weekday, an integer between 1 and 7 (Monday = 1), which is the returned value of the function. To perform a correct conversion it is necessary to define the correct language code. The name of the day will appear in the language represented by the value in setlang. See the table immediately preceding the TICKTODATE function.

Returned value: Weekday number = successful termination; 0 = error.

### Example

```
INT    i;
CHAR  Dayweek[3];
...
strcpy(Dayweek , "Cy6");
i = DAYNAM (Dayweek);
...
```

## BIGYEAR

### Prototype

```
INT BIGYEAR (
    INT Year); /* Year from 00 A.D. */
```

### Description

The BIGYEAR function checks whether the year, Year, specified, an integer from 1 to 2099, is a leap year.

Return value: 1, yes; zero, no.

### Example

```
INT    i;
INT    Year;
...
Year= 1994;
i = BIGYEAR (Year);
...
```

## DAYMON

### Prototype

```
INT DAYMON (
    INT Month, /* Number of month */
    INT Year); /* Year */
```

### Description

The DAYMON function determines the number of days, an integer between 28 and 31, in the month, Month, of the specified year, Year, an integer between 1 and 2099.

Return value: Number of days = success; 0 = incorrect parameters.

### Example

```
INT    i;
INT    Month;
INT    Year;
...
Year = 1994;
Month= 2;
i = DAYMON (Month ,Year);
...
```

## DAYYEAR

### Prototype

```
INT DAYYEAR (
```

```

INT Day,      /* Day of month */
INT Month,    /* Month */
INT Year);    /* Year */

```

### Description

DAYYEAR determines the number of days from the beginning of the year, Year, 1 to 2099, to the date specified by the day of the month, Day, 1 to 31, and the number of month, Month, 1 to 12.

Return value: Number of days = successful; 0 = incorrect parameters.

### Example

```

INT i;
INT Day;
INT Month;
INT Year;
...
Year = 1994;
Month = 5;
Day = 15;
i = DAYYEAR (Day, Month, Year);
...

```

## DAYNUMBERDATE

### Prototype

```

LONG DAYNUMBERDATE (
    INT Day,      /* Day of month */
    INT Month,    /* Month */
    INT Year);    /* Year */

```

### Description

The DAYNUMBERDATE function determines the number of days from 00 A.D. to the date specified by the day of the month, Day, 1 to 31, number of month, Month, 1 to 12, and year, Year, 1 to 2099.

Return value: Number of days - successful; 0 - incorrect parameters.

### Example

```

LONG i;
INT Day;
INT Month;
INT Year;
...
Year = 1994;
Month = 5;
Day = 15;
i = DAYNUMBERDATE (Day, Month, Year);
...

```

## DATEDAYNUMBER

### Prototype

```

void DATEDAYNUMBER (
    LONG Ndate, /* Number of days */

```

```

INT * day,      /* Returned day */
INT * month,    /* Returned month */
INT * year);    /* Returned year */

```

**Description**

The DATEDAYNUMBER function computes the date by the number of days, Ndate, a positive long integer, elapsed since 00 A.D. and stores it as day of month, number of month, and year.

**Example**

```

LONG  Nday;
INT   day;
INT   month;
INT   year;
...
Nday = 790000;
DATEDAYNUMBER (Nday ,&day ,&month ,&year);
...

```

**WEEKDAYNUMBER****Prototype**

```

INT WEEKDAYNUMBER (
    LONG Date);    /* Number of days */
                    */

```

**Description**

The WEEKDAYNUMBER function determines the weekday, 1 to 7, by the number of days from 00 A.D.

Return value: Weekday number – success; 0 – incorrect parameter.

**Example**

```

INT   weekday;
LONG  Date;
...
Date  = 790000;
weekday= WEEKDAYNUMBER (Date);
...

```

**STRTOTICK****Prototype**

```

INT STRTOTICK (
    CHAR * Str,      /* Date as string */
    DECIMAL date); /*Date in internal format*/

```

**Description**

STRTOTICK converts the date represented as a string Str in the

```
dd.mm.yyyy[: [h] h [: [m] m [: [s] s [. [t] t ] ] ] ]
```

or

```
dd/mm/yyyy[:[h]h[:[m]m[:[s]s[.[t]t]]]
```

format into the internal date format.

Return value: 1, success; 0, error.

### Example

```
INT    i;
DECIMAL date;
CHAR * Str = "15.05.2006:12:3:05.75";
...
i = STRTOTICK (Str ,date);
...
```

## TICKTOSTR

### Prototype

```
Void TICKTOSTR (
    DECIMAL Date,      /* Date in internal format*/
    INT DateFormat,    /* Format of date */
    INT YearFormat,     /* Format of year */
    CHAR * str,        /* Date symbol representation */
    INT LengthStr);    /* Length of string */
```

### Description

The TICKTOSTR procedure converts the date represented in internal format to a string, Str, according to the defined conversion formats.

Allowed formats:

DateFormat=0	date separated by dots: dd.mm.yyyy;
DateFormat=1	date separated by backslashes: dd/mm/yyyy;
YearFormat=0	year without century: dd.mm.yy;
YearFormat=1	year with century: dd.mm.yyyy.

### Example

```
typedef char Tstr[40];
...
INT    i;
DATE  Date;
Tstr  str;
...
TICKTOSTR (Date ,0 ,0 ,sizeof(Tstr) ,str);
...
```

## ADDDATE

### Prototype

```
void ADDDATE (
    DECIMAL D1,      /* Date 1 */
    DECIMAL D2,      /* Date 2 */
    DECIMAL dsumm); /* Sum */
```

### Description

The ADDDATE function performs the addition of two dates D1 and D2 as numbers of ticks from 00 A.D. and stores the result in dsumm.

**Example**

```

INT    i;
DECIMAL D, Ds;
...
STRTOTICK ("02.01.0001:00:00:00.00" ,Ds);
ADDDATE (D ,Ds ,D);
...

```

**SUBMONTHSFROMDATE****Prototype**

```

void SUBMONTHSFROMDATE (
    DECIMAL D1,          /* Date*/
    DECIMAL D2,          /* Number of months to be subtracted */
    DECIMAL D3);        /* Resultant date */

```

**Description**

SUBMONTHSFROMDATE subtracts the number of months D2 from date D1 represented as the number of Ticks from 00 A.D. and stores the result in D3. The format of D2 must be MM, MM.YYYY, or MM.YY, where MM = the number of months and YY(YY) = the number of years to be subtracted.

**ADDMONTHSTODATE****Prototype**

```

Void ADDMONTHSTODATE (
    DECIMAL D1,          /* Date */
    DECIMAL D2,          /* Number of months to be added */
    DECIMAL D3);        /* Resultant date */

```

**Description**

The ADDMONTHSTODATE function adds the number of months D2 to date D1, both represented as the number of ticks from 00 A.D. and stores the result in D3. The format of D2 must be MM, MM.YYYY or MM.YY, where MM = the number of months and YY(YY) = the number of years to be added.

**SUBDATE****Prototype**

```

Void SUBDATE (
    DECIMAL D1,          /* Date 1 */
    DECIMAL D2,          /* Date 2 */
    DECIMAL D3);        /* Result */

```

**Description**

SUBDATE computes the difference between D1 and D2 represented as ticks from 00 A.D. and stores the result in D3.

**Example**

```

DECIMAL D1, D2, D3;
...
STRTOTICK ("01.01.0001:00:00:00.00" ,D1);
STRTOTICK ("02.01.0001:00:00:00.00" ,D2);
SUBDATE (D2 ,D1, D3);

```

...

## DATETOTICK

### Prototype

```
void DATETOTICK (  
    LONG L1,      /* number of days from 00 A.D. */  
    LONG L2,      /* number of ticks elapsed in last day */  
    DECIMAL D);  /* Result */
```

### Description

DATETOTICK formats the returned date D in standard format using the defined number of days L1 from 00 A.D. and number of ticks elapsed in the last day L2.

### Example

```
LONG l1;  
LONG l2;  
DECIMAL D;  
...  
DATETOTICK (l1,l2,D);  
...
```

## PUTDIG

### Prototype

```
Void PUTDIG (  
    INT C,      /* digit */  
    CHAR * S,   /* string */  
    INT * N,    /* beginning of string */  
    INT Ok);   /* flag */
```

### Description

The PUTDIG function places a digit from variable C (if flag Ok = 1) or a "?" (if flag Ok = zero) into position N of string S and increments N by one.

### Example

```
CHAR str[3];  
INT i = 0;  
...  
PUTDIG (i,str,&i,1);  
PUTDIG (i,str,&i,0);  
PUTDIG (i,str,&i,1);  
...
```

## PUT2

### Prototype

```
void PUT2 (  
    INT X,      /* digit */  
    CHAR * S,   /* string */  
    INT * N,    /* beginning of string */
```

```
INT Ok);    /* flag */
```

### Description

The PUT2 function places a two digit number from variable S (if flag Ok = 1) or "??" (if flag Ok = zero) into position N of string X and increments N by one. If X is not a two digit number, the result is unpredictable.

### Example

```
CHAR str[2];
INT i = 0;
INT x = 73;
...
PUT2 (x,str,&i,1);
...
```

## PUTYEAR

### Prototype

```
void PUTYEAR (
    INT Y,           /* Number */
    INT ISCENTURY, /* Flag */
    CHAR * S,       /* String */
    INT * N,        /* Beginning of string */
    INT Ok);       /* Flag */
```

### Description

PUTYEAR places the number of the year from variable Y into position N of string S. If the flag ISCENTURY is zero, only the last two digits of the year will be placed in the string and N will be incremented by two. If ISCENTURY is 1, the four digit year will be inserted and N will be increased by four. If the flag Ok is zero, the string will have"??" or"????" inserted, depending on the ISCENTURY flag.

### Example

```
CHAR str[4];
INT i = 0;
INT x = 1996;
...
PUTYEAR (x,1,str,&i,1);
...
```

## CMPDATE

### Prototype

```
INT CMPDATE (
    DECIMAL D1, /* Date 1 */
    DECIMAL D2); /* Date 2 */
```

### Description

### Description

CMPDATE compares two dates, D1 and D2, represented in internal format, DECIMAL.

Returned values:

- +1, if D1 > D2;
- 0, if D1 = D2;
- -1, if D1 < D2.

## TICKTOSTRF

### Prototype

```
INT TICKTOSTRF (
    DECIMAL D1, /* Date */
    CHAR * F, /* Conversion format */
    CHAR * S); /* Returned string */
```

### Description

The TICKTOSTRF function converts date D from internal format to string S according to format F. To specify the string format, the following may be used:

<u>Unit of Time</u>	<u>Format Options</u>
Day	DD, dd
Month	MM, mm, Mon, MON
Year	YY, YYYY
Hour	HH24, hh24
Minute	Mi, mi
Second	SS, ss
Ticks	FF, ff

Separators are minus (-), slash (/), colon (:), and period (.).

Examples of date formats:

```
"DD-Mon-YY", MM/DD/YYYY",
"DD-Mon-YYYY", "DD.MM.YY",
"MM/DD/YY", "DD.MM.YYYY".
```

Examples of time formats:

```
"HH24", "HH24:MI:SS",
"HH24:MI", "HH24:MI:SS.FF".
```

Return value: zero, success; 1, error.

### Example

```
DECIMAL d;
CHAR s[8];
INT Error;
...
Error = TICKTOSTRF (d, "DD.MM.YY", s);
...
```

## STRTOTICKF

### Prototype

```
INT STRTOTICKF (  
    CHAR * S,      /* string */  
    CHAR * F,      /* string format */  
    DECIMAL D);   /* Resultant date */
```

### Description

The STRTOTICKF function converts string S into date D according to the format defined in F. The specifics of the formats are described in TICKTOSTRF, immediately above.

Return values: 0, success; 1, error.

### Example

```
DECIMAL D;  
INT Error;  
...  
Error = STRTOTICKF ("29|08|96", "DD|MM|YY", D);  
...
```

## Int64 Library

The Int64 Library is intended to perform operation with 64-bit integers (BIGINT data type in DBMS Linter). This data type was introduced due to the necessity to address BLOB data that can reach 64 gigabytes in size.

The functionality of the library supports the following functions:

- 1) Arithmetic functions;
- 2) Logical functions;
- 3) Other functions.

The library can be utilized only in C/C++ programs.


The C/C++ module in which the library functions are to be used should include the int64.h header file.

Also, the int64.c file should be added to the application project.

 int64.h and int64.c files are included in the DBMS Linter distributive.

The main characteristics are:

- Long integers are stored in 8-byte fields;
- When performing arithmetic functions, overflow errors are not detected, and the corresponding termination codes are not returned;
- On hardware platforms supporting functions with long integers, library calls will be automatically replaced with their hardware realization.

 Library functions and types of arguments are described using the designation defined in the int64.h header file.

## Arithmetic functions

### i64\_Add

#### Prototype

```
L_DLONG i64_Add (  
    L_DLONG num1, /* first operand */  
    L_DLONG num2); /* second operand*/
```

#### Description

The i64\_Add function returns the sum of two 64-bit integers (num1 and num2).

### i64\_Sub

#### Prototype

```
L_DLONG i64_Sub (  
    L_DLONG num1, /* first operand */  
    L_DLONG num2); /* second operand*/
```

**Description**

The `i64_Sub` function returns the difference of two 64-bit integers (`num1` and `num2`).

**i64\_Mul****Prototype**

```
L_DLONG i64_Mul (  
    L_DLONG num1,      /* first operand */  
    L_DLONG num2);    /* second operand*/
```

**Description**

The `i64_Mul` function returns the product of two 64-bit integers (`num1` and `num2`).

**i64\_Div****Prototype**

```
L_DLONG i64_Div (  
    L_DLONG num1,      /* first operand */  
    L_DLONG num2);    /* second operand*/
```

**Description**

The `i64_Div` function performs integer division of a 64-bit integer (`num1`), by another 64-bit integer (`num2`). If the `num2` equals zero, the function returns the maximum `L_DLONG` value.

**i64\_Rest****Prototype**

```
L_DLONG i64_Rest (  
    L_DLONG num1,      /* first operand */  
    L_DLONG num2);    /* second operand*/
```

**Description**

The `i64_Rest` function performs integer division of a 64-bit integer (`num1`), by another 64-bit integer (`num2`), and returns the remainder of this division. If the `num2` equals zero, the function returns the random `L_DLONG` value.

**i64\_DivRest****Prototype**

```
L_DLONG i64_DivRest (  
    L_DLONG num1,      /* first operand */  
    L_DLONG num2,      /* second operand*/  
    L_DLONG *rest);    /* result */
```

**Description**

The `i64_DivRest` function returns the result of integer division of a 64-bit integer (`num1`), by another 64-bit integer (`num2`), and the remainder of this division is placed at the address indicated by the `*rest` argument, if it is not NULL.

**i64\_AddLong****Prototype**

```
L_DLONG i64_AddLong (  
    L_DLONG num1,    /* first operand */  
    L_LONG num2);    /* second operand*/
```

**Description**

The `i64_AddLong` function returns the sum of an 64-bit integer (`num 1`) and a 32-bit integer (`num2`).

**i64\_SubLong****Prototype**

```
L_DLONG i64_SubLong (  
    L_DLONG num1,    /* first operand */  
    L_LONG num2);    /* second operand*/
```

**Description**

The `i64_SubLong` function returns the difference of a 64-bit integer (`num 1`) and a 32-bit integer (`num2`).

**i64\_MulLong****Prototype**

```
L_DLONG i64_MulLong (  
    L_DLONG num1,    /* first operand */  
    L_LONG num2);    /* second operand*/
```

**Description**

The `i64_MulLong` function returns the product of a 64-bit integer (`num 1`) and a 32-bit integer (`num2`).

**i64\_DivLong****Prototype**

```
L_DLONG i64_DivLong (  
    L_DLONG num1, /* first operand */  
    L_LONG num2); /* second operand*/
```

**Description**

The `i64_DivLong` function returns the quotient of a 64-bit integer (`num1`) and a 32-bit integer (`num2`). If the `num2` equals zero, the function returns the maximum value of the `num1`. If the `num2` equals zero, the function returns the random `L_DLONG` value.

**i64\_RestLong****Prototype**

```
L_DLONG i64_RestLong (  
    L_DLONG num1, /* first operand */  
    L_LONG num2); /* second operand*/
```

**Description**

The `i64_DivLong` function returns the remainder of integer division of a 64-bit integer (`num1`) by a 32-bit integer (`num2`). If the `num2` equals zero, the function returns the maximum value of the `num2`.

**Logical Functions****i64\_And****Prototype**

```
L_DLONG i64_And (  
    L_DLONG num1, /* first operand */  
    L_DLONG num2); /* second operand*/
```

**Description**

The `i64_And` function performs per-bit multiplication (the logical AND) of two 64-bit integers (`num1` and `num2`).

**i64\_Or****Prototype**

```
L_DLONG i64_Or (  
    L_DLONG num1, /* first operand */  
    L_DLONG num2); /* second operand*/
```

**Description**

The `i64_Or` function performs per-bit addition (the logical OR) of two 64-bit integers (`num1` and `num2`).

## Other Functions

### i64\_Cmp

#### Prototype

```
L_LONG i64_Cmp (  
    L_DLONG num1, /* first operand */  
    L_DLONG num2); /* second operand*/
```

#### Description

The i64\_Cmp function compares two 64-bit integers (num1 and num2).

The function returns:

- +1, if num1 > num2;
- 0, if num1=num2;
- -1, if num1<num2.

### i64\_Neg

#### Prototype

```
L_DLONG i64_Neg (L_DLONG num);
```

#### Description

The i64\_Neg function changes the sign of 64-bit integer (num) into the opposite.

### i64\_IfNeg

#### Prototype

```
L_LONG i64_IfNeg (L_DLONG num);
```

#### Description

The i64\_IfNeg function verifies the sign of a 64-bit integer (num) into the opposite.

The function returns:

- 1, if the num is negative;
- 0, if the num is positive.

### i64\_IfZero

#### Prototype

```
L_LONG i64_IfZero (L_DLONG num);
```

#### Description

The i64\_IfZero function verifies whether the 64-bit integer (num) equals zero.

The function returns:

- 1, if the num equals zero;
- 0, if the num doesn't equal zero.

## **i64\_Inc**

### **Prototype**

```
L_DLONG i64_Inc (L_DLONG * num);
```

### **Description**

The `i64_Inc` function returns the value of a 64-bit integer (placed at the `*num` address) increased by one.

## **i64\_Dec**

### **Prototype**

```
L_DLONG i64_Dec (L_DLONG * num);
```

### **Description**

The `i64_Dec` function returns the value of a 64-bit integer (placed at the `*num` address) decreased by one.

## **i64\_MINDLONG**

### **Prototype**

```
L_DLONG i64_MINDLONG (void);
```

### **Description**

The `i64_MINLONG` function returns the minimum sign value of a 64-bit integer.

## **i64\_MAXDLONG**

### **Prototype**

```
L_DLONG i64_MAXDLONG (void);
```

### **Description**

The `i64_MAXLONG` function returns the maximum sign value of a 64-bit integer.

## **i64\_MAXUDLONG**

### **Prototype**

```
L_DLONG i64_MAXUDLONG (void);
```

**Description**

The `i64_MAXLONG` function returns the maximum unsigned value of a 64-bit integer.

**i64\_InvertBytes****Prototype**


```
void i64_InvertBytes (L_DLONG * Val);
```

**Description**

The `i64_MAXLONG` function changes the order of the bytes of a 64-bit integer (placed at the `*num` address).

**Macros**

The `int64` library includes a number of macros. Unlike the library functions, in the process of source code translation, macros are replaced with C/C++ commands or by some of the functions of the `INT64` library.

 The macro texts are in the `int64.h` header file.

Below is the list of macros.

<b><u>Macro name</u></b>	<b><u>Purpose</u></b>
<code>i64_ToLong(i,l)</code>	Conversion of a 32-bit integer to a 64-bit integer
<code>i64_FromLong(l,i)</code>	Conversion of a 64-bit integer to a 32-bit integer
<code>i64_ToDouble(i,d)</code>	Conversion of a 64-bit integer to a double accuracy material number
<code>i64_FromDouble(d,i)</code>	Conversion of a double accuracy material number to a 64-bit integer
<code>i64_GetLo(i)</code>	Getting the lowest four bytes of a 64-bit integer
<code>i64_GetHi(i)</code>	Getting the highest four bytes of a 64-bit integer
<code>i64_Init(l,lo,hi)</code>	Assigning a value to a 64-bit integer
<code>i64_IfPos(num)</code>	Verifying the positive value of a 64-bit integer
<code>i64_Equ(n1 ,n2)</code>	Verification of equality of two 64-bit integers