R:BASE 11

for Windows

Function Index

The Programmer's Guide to Building
R:BASE Databases and Custom Applications

R:BASE 11 for Windows

Function Index

by R:BASE Technologies, Inc.

R:BASE provides a wide range of predefined functions. A function differs from an operator in that a function provides a predefined complex expression to evaluate standard mathematical, trigonometric, financial, or logical functions without requiring the user to enter the formula in a complete R:BASE expression. Function names are reserved words.

R:BASE 11 Function Index

Printed: May 2024 in Murrysville, PA

First Edition

Table of Contents

Part I

1 Function Index

1.1 Function Categories

Arithmetic and Mathematical Functions

ABS	AVG	BRND	COUNT
DIM	EXP	LAVG	LISTOF
LMAX	LMIN	LOG	LOG10
LSTDEV	LSUM	LVARIANCE	MAX
MIN	MOD	PSTDEV	PVARIANCE
RANDOM	RNDDOWN	RNDUP	ROUND
SIGN	SQRT	STDEV	SUM
VARIANCE			

Conversion Functions

AINT	ANINT	CHAR	CTXT
DWRD	FLOAT	HTML	HZC
ICHAR	IHASH	INT	MAKEUTF8
NINT	SOUNDEX	YWRD	ZHC
ZHCX			

Database Utility Functions

CVAL	IINFO	TINFO

Date and Time Functions

ADDDAY	ADDFRC	ADDHR	ADDMIN
ADDMON	ADDSEC	ADDYR	DATETIME
DEXTRACT	DNW	DWE	GETDATE
IDAY	IDIM	IDOY	IDWK
IFRC	IHR	ILY	IMIN
IMON	ISEC	IWOY	IYR
IYR4	JDATE	JDWK	RDATE
RTIME	TDWK	TEXTRACT	TMON

Encryption Functions

DECRYPT	ENCRYPT

Financial Functions

FV1	FV2	PMT1	PMT2
PV1	PV2	RATE1	RATE2
RATE3	TERM1	TERM2	TERM3

Keyboard and Environment Functions

ENVVAL	CVAL	CHKCUR	CHKFILE
CHKFUNC	CHKKEY	CHKTABLE	CHKVAR
CVTYPE	DELFUNC	DLCALL	DLFREE
DLLOAD	FILENAME	FINDFILE	FSTATUS
IFWINDOW	ISTAT	GETKEY	GETPROPERTY
LASTKEY	LASTMOD		

Logical Functions

IFCASEEQ	IFEQ	IFEXISTS	IFF
IFGE	IFGT	IFLE	IFLT
IFNE	IFNULL		

Statistical Functions

1.2 A

1.2.1 ABS

(ABS(*arg*))

Returns the absolute or positive value of *arg* (a value with a DOUBLE, REAL, NUMERIC, or INTEGER data type).

In the following example, the value of *vabs* is *2*.

```
SET VAR vnum = -2
SET VAR vabs = (ABS(.vnum))
```

1.2.2 ACOS

(ACOS(*arg*))

Computes the arccosine of *arg* where *arg* is in the range -1 to 1. The result is an angle in radians between 0 and pi (where pi = 3.14159265358979).

In the following example, the value of *vacos* is *2.094395*, the arccosine of -0.5.

```
SET VAR vacos = (ACOS(-0.5))
```

1.2.3 ADDDAY

(ADDDAY(*date,int*))

Adds the specified number of days to a date or datetime value. Functions that result in an invalid date, for example, February 30, will return NULL.

1.2.4 ADDFRC

(ADDFRC(*time,int*))

Adds the specified number of milliseconds to a time or datetime value.

1.2.5 ADDHR

(ADDHR(*time,int*))

Adds the specified number of hours to a time or datetime value.

1.2.6 ADDMIN

(ADDMIN(*time,int*))

Adds the specified number of minutes to a time or datetime value.

1.2.7 ADDMON

(ADDMON(*date,int*))

Adds the specified number of months to a date or datetime value.

1.2.8 ADDSEC

(ADDSEC(*time,int*))

Adds the specified number of seconds to a time or datetime value.

1.2.9 ADDYR

(ADDYR(*date,int*))

Adds the specified number of years to a date or datetime value.

1.2.10 AINT

(AINT(*arg*))

Truncates the decimal fraction, returning a whole number in the original REAL, NUMERIC, or DOUBLE data type.

In the following example, the value of *vaint* is *1*.

```
SET VAR vaint = (AINT(1.8))
```

1.2.11 ANINT

(ANINT(*arg*))

Rounds the decimal fraction to the nearest integer, returning a whole number in the original REAL, NUMERIC, or DOUBLE data type.

In the following example, the value of *vanint1* is *3.0* and the value of *vanint2* is *4.0*.

```
SET VAR vanint1 = (ANINT(2.6))
```

```
SET VAR vanint2 = (ANINT(4.45))
```

1.2.12 ASIN

(ASIN(*arg*))

Computes the arcsine of *arg* where *arg* is in the range -1 to 1. The result is an angle in radians between -pi/2 and pi/2.

In the following example, the value of *vasin* is *-0.5236* (-pi/6 radians)

```
SET VAR vasin = (ASIN(-0.5))
```

1.2.13 ATAN

(ATAN(*arg*))

Computes the arctangent of *arg* in radians where *arg* is any amount. The result is an angle in radians between -pi/2 and pi/2.

In the following example, the value of *vatan* is *0.7854* (pi/4 radians).

```
SET VAR vatan = (ATAN(1))
```

1.2.14 ATAN2

(ATAN2(*x,y*))

Computes the arctangent of *x/y*. The result is the angle in radians between -pi/2 and pi/2.

In the following example, the value of *vatan2* is *0.7854*.

```
SET VAR vatan2 = (ATAN2(1,1))
```

1.3 B

1.3.1 BRND

(BRND(*arg1,arg2,arg3*))

Rounds REAL, DOUBLE, or CURRENCY data to a specific number of decimal places and allows specification of the number of significant digits to return. *Arg1* is the value to be rounded. *Arg2* is the number of significant digits to return, and *arg3* is the precision The precision is specified as a decimal number, for example, .01 rounds to two decimal places.

In the following example, the value of *vresult* is 1234.57.

```
SET VAR vresult = (BRND(1234.5678342,8,.01))
```

1.4 C

1.4.1 CHAR

(CHAR(*integer*))

Converts an ASCII integer value to its corresponding character. This is not the same as the CHAR data type.

In the following example, the value of *vchar1* is *A* and the value of *vchar2* is *a*.

```
SET VAR vchar1 = (CHAR(65))
SET VAR vchar2 = (CHAR(97))
```

See also:

ASCII Character Chart

1.4.2 CHKCUR

(CHKCUR('*cursorname*'))

Checks to see if a cursor is declared and is not dropped. The function returns an integer value of 1 if the name exists and is not dropped, and 0 if it is not declared or dropped.

1.4.3 CHKFILE

(CHKFILE('*filespec*'))

Checks to see if a file or folder name exists. If no path is specified, the function checks for the file or folder name in the current directory. Otherwise, the function checks for the file or folder name in the specified location.

The function returns a 1 if the file or folder name is found, and 0 if it is not found. Wild cards in the filename will produce unpredictable results.

1.4.4 CHKFUNC

(CHKFUNC('function_name'))

Checks to see if a DLL function exists or not. If the DLL function exists, a 1 is returned. If the DLL function does not exist, a 0 is returned.

Example:

```
SET VAR v1 = (CHKFUNC('FunctionName'))
```

See Also:

DELFUNC
DLLCALL
DLLOAD
DLFREE
LIST FUNCTIONS

1.4.5 CHKKEY

(CHKKEY(0))

Returns an integer value of *1* if there are keystrokes available in the type-ahead buffer. Returns *0* if no keystrokes are available. Use CHKKEY before GETKEY to determine if a key is available.

CHKKEY does nothing with the zero that you enter in parentheses; CHKKEY returns a value without receiving one.

1.4.6 CHKTABLE

(CHKTABLE('*tblview*'))

Checks to see if a table/view exists. The function returns a value based upon the permanent or temporary nature of a table or view, and if a table is attached as a server or dBASE table.

Return Value	Status
0	table/view does not exist
1	permanent table
2	temporary table
3	server table
4	dBASE table
5	permanent view
6	temporary view

Examples:

Example 01:
```
SET VAR vCheckTable INTEGER = (CHKTABLE('Contact'))
SHOW VAR vCheckTable
1
```

Example 02:
```
SET VAR vCheckTable INTEGER = (CHKTABLE('QuarterlySummary'))
SHOW VAR vCheckTable
5
```

1.4.7 CHKVAR

(CHKVAR('var*name*'))

Checks to see if a variable name exists. The function returns an integer value of 1 if the variable name exists or declared, and 0 if it is not found or declared.

1.4.8 COS

(COS(*angle*))

Returns the trigonometric cosine of *angle*. The result is between -1 and 1.

In the following example, the value of *vcos* is *0.5002*.

```
SET VAR vcos = (COS(1.047))
```

1.4.9 COSH

(COSH(*angle*))

Returns the hyperbolic cosine of *angle*.

In the following example, the value of *vcosh* is *1.6000*.

```
SET VAR vcosh = (COSH(1.047))
```

1.4.10 CTR

(CTR(*text,width*))

Centers *text* in *width* characters, returning a text string.

In the following example, the value of *vctr* is *ABCD* .

The text string is centered in a 10-character field.

```
SET VAR vctr = (CTR('ABCD',10))
```

1.4.11 CTXT

(CTXT(*arg*))

Converts an internal value, returning a text string.

In the following example, the value of *vctxt1* is the value *37.6* and is a REAL data type. The value of *vctxt2* is the value *37.6* and is a TEXT data type. If you attempt to use *vctxt2* in a mathematical function, it will fail.

```
SET VAR vctxt2 = (CTXT(.vctxt1))
```

1.4.12 CVAL

(CVAL(*'showkeyword'*))

Returns the current value or setting of *'showkeyword'*. You must either enclose the SHOW keyword in quotation marks or use a dot variable that has a TEXT data type to which you have assigned the SHOW keyword. You can use all SHOW keywords with CVAL, as well as DATABASE, DBPATH, CURRDIR.

The following keywords can be used for (CVAL('keyword')):

- AND
- ANSI
- ANSIOUTPUT
- AUTOCOMMIT
- AUTODROP
- AUTOSKIP
- BELL
- BLANK
- BUILD
- CASE
- CHECKPROP
- CLEAR
- CLIPBOARDTEXT
- CMPAUSE
- COLCHECK
- COLOR

- ZOOMEDIT

Examples:

In the following example, the value of *vcval* is *OFF* if the value of MULTI is set to off.

```
SET VAR vcval = (CVAL('MULTI'))
```

In the following example, the user keyword is loaded into a variable and then the variable is used in the CVAL function. It returns the current user identifier.

```
SET VAR vword text = 'USER'
SET VAR vuser = (CVAL(.vword))
```

For more information about SHOW keywords, see SHOW.

1.4.12.1 AND

(CVAL('AND'))

Returns the status of the AND command parameter (ON/OFF).

1.4.12.2 ANSI

(CVAL('ANSI'))

Returns the status of the ANSI command parameter (ON/OFF).

1.4.12.3 ANSIOUTPUT

(CVAL('ANSIOUTPUT'))

Returns the status of the ANSIOUTPUT command parameter (ON/OFF).

1.4.12.4 AUTOCOMMIT

(CVAL('AUTOCOMMIT'))

Returns the status of the AUTOCOMMIT command parameter (ON/OFF).

1.4.12.5 AUTODROP

(CVAL('AUTODROP'))

Returns the status of the AUTODROP display control (ON/OFF).

1.4.12.6 AUTOSKIP

(CVAL('AUTOSKIP'))

Returns the status of the AUTOSKIP command parameter (ON/OFF).

1.4.12.7 BELL

(CVAL('BELL'))

Returns the status of the BELL command parameter (ON/OFF).

1.4.12.8 BOOLEAN

(CVAL('BOOLEAN'))

Returns the status of the BOOLEAN operating condition (ON/OFF).

1.4.12.9 BLANK

(CVAL('BLANK'))

Returns an empty variable value.

BLANK is similar to a null value, but is not based on your database NULL setting.

1.4.12.10 BUILD

(CVAL('BUILD'))

Returns the build number of the R:BASE executable (front-end).

See also:

(CVAL('VERSION'))
(CVAL('VERSION BUILD'))
SHOW BUILD
SHOW VERSION

1.4.12.11 CASE

(CVAL('CASE'))

Returns the status of the CASE command parameter (ON/OFF).

1.4.12.12 CHECKPROP

(CVAL('CHECKPROP'))

Returns the status of the CHECKPROP command parameter (ON/OFF).

1.4.12.13 CLEAR

(CVAL('CLEAR'))

Returns the status of the CLEAR command parameter (ON/OFF).

1.4.12.14 CLIPBOARDTEXT

(CVAL('CLIPBOARDTEXT'))

Returns the contents of the Windows Clipboard.

The below example can be used to check the clipboard contents.

```
SET VAR vCheckClipBoard = (CVAL('CLIPBOARDTEXT'))
```

1.4.12.15 CMPAUSE

(CVAL('CMPAUSE'))

Returns the status of the CMPAUSE command parameter (ON/OFF).

1.4.12.16 COLCHECK

(CVAL('COLCHECK'))

Returns the status of the COLCHECK command parameter (ON/OFF).

1.4.12.17 COLOR

(CVAL('COLOR'))

Returns the value for the foreground and background COLOR of data displays.

This CVAL parameter is specific to R:BASE for DOS.

1.4.12.18 COMPUTER

(CVAL('COMPUTER'))

Returns the name of your computer.

Example:

```
SET VAR vComp = (CVAL('COMPUTER'))
```

1.4.12.19 CONNECTIONS

(CVAL('CONNECTIONS'))

Returns the number of users currently connected to the current database or 0 if not connected. In the event of a non-graceful disconnect R:BASE will not be able to decrement the connections count. This can occur if the network session terminates unexpectedly, or if the users operating system crashes. If this happens the count will be inaccurate until all users disconnect from the database. This works because the last session of R:BASE will be able to tell that NO users are connected and will reset the count to zero. Unfortunately, due to file system limitations, R:BASE is only able to tell if a database is open by any users or not at all, and is not able to tell, except by this count, how many users are connected.

1.4.12.20 CURRDIR

(CVAL('CURRDIR'))

Returns the current directory. This is the same information returned by using the CD command at the R> Prompt.

1.4.12.21 CURRDRV

(CVAL('CURRDRV'))

Returns the current drive. The drive is in X: format with the drive letter followed by a colon.

1.4.12.22 CURRENCY

(CVAL('CURRENCY'))

The CURRENCY data type holds monetary values of up to 23 digits represented in the currency format, established using SET CURRENCY. Amounts are in the range ±$99,999,999,999,999.99. Commas or the current delimiter can be used. If no decimal point is included, .00 is assumed.

Data is stored as two long integer values, reserving four bytes of internal storage.

1.4.12.23 CURRENTPRINTER

(CVAL('CURRENTPRINTER'))

This parameter returns the current printer for the R:BASE session.

1.4.12.24 DATABASE

(CVAL('DATABASE'))

Returns a text string containing the current connected database or NULL if the user is not connected to a database. This can be used to ensure that the user is connected before trying to execute some code. The example below shows how this might work.

```
SET VAR vDB = (CVAL('DATABASE'))
IF vDB IS NULL THEN
  CONNECT MyDB
ENDIF

SET VAR vDB = (CVAL('DATABASE'))
IF vDB IS NULL THEN
  PAUSE 2 USING 'MyDB is currently unavailable'
ENDIF
```

The check is repeated to ensure that the attempt to connect to the database was successful before continuing with the command file.

1.4.12.25 DATE

(CVAL('DATE'))

Returns the DATE format of the currently connected database.

1.4.12.26 DATE CENTURY

(CVAL('DATE CENTURY'))

Returns the default DATE CENTURY of the currently connected database.

1.4.12.27 DATE FORMAT

(CVAL('DATE FORMAT'))

Returns the default DATE FORMAT of the currently connected database.

1.4.12.28 DATE SEQUENCE

(CVAL('DATE SEQUENCE'))

Returns the default DATE SEQUENCE of the currently connected database.

1.4.12.29 DATE YEAR

(CVAL('DATE YEAR'))

Returns the default DATE YEAR of the currently connected database.

1.4.12.30 DBCOMMENT

(CVAL('DBCOMMENT'))

DBCOMMENT returns the comment for the current connected database.

1.4.12.31 DBPATH

(CVAL('DBPATH'))

DBPATH returns the full path to the current connected database, or NULL if no database is connected.

1.4.12.32 DEBUG

(CVAL('DEBUG'))

Returns the status of the DEBUG command parameter (ON/OFF).

1.4.12.33 DELIMIT

(CVAL('DELIMIT'))

Returns the value of the DELIMIT character setting.

1.4.12.34 DRIVES

(CVAL('DRIVES'))

Returns a list of available drives.

Example:

```
SET VAR vDrives = (CVAL('Drives'))
```

vDrives will include the list of ALL drives!

Result: aCDeF

In that list of drives, drives with CAPITAL letters, such as C,D or F would be the hard disk drive, and drives with lower case letters would be removable drives, such as a or e.

a = floppy disk drive
C = Hard Disk
D = Hard Disk/CD-ROM/DVD
e = zip disk
F = Hard Disk/CD-ROM/DVD or even mapped network drive

All hard drives, including CD-ROM/DVD and network mapped drives would be CAPITAL letters.

All removable drives, including floppy disk drives and zip drives would be lower case letters.

1.4.12.35 ECHO

(CVAL('ECHO'))

Returns the status of the ECHO command parameter (ON/OFF).

1.4.12.36 EDITOR

(CVAL('EDITOR'))

Returns the current text editor used by R:BASE.

The default editor is RBEDIT.

1.4.12.37 EOFCHAR

(CVAL('EOFCHAR'))

Returns the status of the EOFCHAR command parameter (ON/OFF).

1.4.12.38 EQNULL

(CVAL('EQNULL'))

Returns the status of the EQNULL command parameter (ON/OFF).

1.4.12.39 ERROR

(CVAL('ERROR'))

Returns the status of the ERROR MESSAGES display control (ON/OFF).

1.4.12.40 ERROR DETAIL

(CVAL('ERROR DETAIL'))

When an -ERROR- occurs, you can track additional information including the name of the file being run and the byte offset within the file. If the thing being run is a procedure it saves information on the select used to fetch the procedure too. We have implemented a new method of simple "stack" to keep the last three errors. To see the information tracked use this new CVAL option.

```
SET VAR vError = (CVAL('ERROR DETAIL'))
```

Each time you call this particular CVAL function the stack pointer decrements so successive calls allow you to see all three errors. Call it a fourth time and you will see the first one again, etc.

1.4.12.41 ERROR MESSAGE

(CVAL('ERROR MESSAGE'))

When an -ERROR- occurs, users are able to get the error message using this function. The function also return the error message when from an SQL data source.

```
SET VAR vErrorMsg = (CVAL('ERROR MESSAGE'))
```

1.4.12.42 ERROR VARIABLE

(CVAL('ERROR VARIABLE'))

Returns the value of the current ERROR VARIABLE.

1.4.12.43 ESCAPE

(CVAL('ESCAPE'))

Returns the status of the ESCAPE command parameter (ON/OFF).

1.4.12.44 EXPLODE

(CVAL('EXPLODE'))

Returns the status of the EXPLODE command parameter (ON/OFF).

Used with R:BASE for DOS only. When EXPLODE is set on, dialog boxes are displayed in full size instantly. When EXPLODE is set off, dialog boxes are displayed in an expanding fashion from the center.

1.4.12.45 FASTFK

(CVAL('FASTFK'))

Returns the status of the FASTFK command parameter (ON/OFF).

1.4.12.46 FASTLOCK

(CVAL('FASTLOCK'))

Returns the status of the FASTLOCK command parameter (ON/OFF).

1.4.12.47 FEEDBACK

(CVAL('FEEDBACK'))

Returns the status of the FEEDBACK command parameter (ON/OFF).

1.4.12.48 FILES

(CVAL('FILES'))

Returns the value of the FILES operating condition.

1.4.12.49 FIXED

(CVAL('FIXED'))

Returns the status of the FIXED command parameter (ON/OFF).

1.4.12.50 GUID

(CVAL('GUID'))

Returns a globally unique identifier value.

Example:

```
SET VAR vNewGUID = (CVAL('GUID'))
SHOW VAR
vNewGUID = BEDA7594-F6C1-44BD-A732-4BB72F17A1DA     TEXT
```

1.4.12.51 HEADINGS

(CVAL('HEADINGS'))

Returns the status of the HEADINGS display control (ON/OFF).

1.4.12.52 IDQUOTES

(CVAL('IDQUOTES'))

Returns the value of the IDQUOTES character setting.

1.4.12.53 INSERT

(CVAL('INSERT'))

Returns the status of INSERT display control (ON/OFF).

1.4.12.54 INTENSITY

(CVAL('INTENSITY'))

Returns the status of INTENSITY display control (ON/OFF).

Used with R:BASE 6.5++ for Windows only.

1.4.12.55 INTERVAL

(CVAL('INTERVAL'))

Returns value of the INTERVAL command parameter (ON/OFF).

1.4.12.56 ISOWNER

(CVAL('ISOWNER'))

Returns the value of YES when the current user has OWNER permissions, and NO when the user does not.

1.4.12.57 LAST ERROR

(CVAL('LAST ERROR'))

Returns the last error encountered.

This is an alternative to the SET ERROR VARIABLE method of -ERROR- trapping.

When using (CVAL('LAST ERROR')) the variable used to capture the error must be explicitly cleared, as opposed to SET ERROR VARIABLE, which is automatically cleared after each command, requiring error trapping logic immediately after each command. Furthermore, the SET ERROR VARIABLE method does not allow error trapping in nested code segments or large blocks of code.

1.4.12.58 LAST_MOD

(CVAL('LAST_MOD'))

Returns the date and time stamp for the last modification made to the database data. The value is also displayed with the LIST command.

The value is saved in the database. When you connect to database which does not have this value stored yet, the date and time stamp from the RX1-RX4 file synchronization will be used as the initial last modification value.

The CVAL function returns a TEXT value for the date and time stamp. To get the last data modification into a DATETIME variable, you may perform the following:

```
SET VAR vLastMod_Text TEXT = (CVAL('LAST_MOD'))
SET VAR vLastMod_DateTime DATETIME = .vLastMod_Text
```

1.4.12.59 LAST_SCHEMA_MOD

(CVAL('LAST_SCHEMA_MOD'))

Returns the date and time stamp for the last modification made to the database schema. The value is also displayed with the LIST command.

The value is saved in the database. When you connect to database which does not have this value stored yet, the date and time stamp from the RX1-RX4 file synchronization will be used as the initial last schema modification value.

The CVAL function returns a TEXT value for the date and time stamp. To get the last schema modification into a DATETIME variable, you may perform the following:

```
SET VAR vLastSchemaMod_Text TEXT = (CVAL('LAST_SCHEMA_MOD'))
SET VAR vLastSchemaMod_DateTime DATETIME = .vLastSchemaMod_Text
```

1.4.12.60 LASTBLOCKTABLE

(CVAL('LastBlockTable'))

Tells you which table holds the last block in File 2.

If the last block table contains significant dead space, packing it should free up space at the end of File 2.

1.4.12.61 LAYOUT

(CVAL('LAYOUT'))

Returns the status of the LAYOUT display control (ON/OFF).

1.4.12.62 LINEEND

(CVAL('LINEEND'))

Returns the value of the LINEEND character setting.

1.4.12.63 LINES

(CVAL('LINES'))

Returns value of the LINES display control.

1.4.12.64 LOOKUP

(CVAL('LOOKUP'))

Returns the value of the LOOKUP command parameter (ON/OFF).

1.4.12.65 MANOPT

(CVAL('MANOPT'))

Returns value of the MANOPT command parameter (ON/OFF).

1.4.12.66 MANY

(CVAL('MANY'))

Returns the value of the MANY character setting.

1.4.12.67 MAXTRANS

(CVAL('MAXTRANS'))

Returns value of the MAXTRANS operating condition.

1.4.12.68 MDI

(CVAL('MDI'))

Returns the value of the MDI operating condition (ON/OFF).

1.4.12.69 MESSAGES

(CVAL('MESSAGES'))

Returns the status of the MESSAGES display control (ON/OFF).

1.4.12.70 MIRROR

(CVAL('MIRROR'))

Returns the value of the MIRROR operating condition.

1.4.12.71 MULTI

(CVAL('MULTI'))

Returns the value of the MULTI operating condition (ON/OFF).

1.4.12.72 NAME

(CVAL('NAME'))

Returns the currently logged-in R:BASE user NAME.

1.4.12.73 NAMEWIDTH

(CVAL('NAMEWIDTH'))

Returns the value of the NAMEWIDTH operating condition.

1.4.12.74 NETGROUP

(CVAL('NETGROUP'))

Returns a list of global groups to which the current logged-in user belongs.

1.4.12.75 NETLOCALGROUP

(CVAL('NETLOCALGROUP'))

Returns a list of local groups to which the current logged-in user belongs.

1.4.12.76 NETUSER

(CVAL('NETUSER'))

Returns the currently logged-in network user name.

1.4.12.77 NOCALC

(CVAL('NOCALC'))

Returns the value of the NOCALC operating condition (ON/OFF).

1.4.12.78 NOTE_PAD

(CVAL('NOTE_PAD'))

Returns value of the NOTE_PAD operating condition.

1.4.12.79 NULL

(CVAL('NULL'))

Returns the value of the NULL character setting.

1.4.12.80 OFFMESS

(CVAL('OFFMESS'))

Return a text string, separated by a comma, with a list of all turned off -ERROR- message numbers in a current R:BASE session.

See SET ERROR MESSAGE.

1.4.12.81 OLDLINE

(CVAL('OLDLINE'))

Returns the value of the OLDLINE operating condition (ON/OFF).

Used with R:BASE 6.5++ for Windows only. Allows the ability to have presentation objects, such as lines, cross multiple sections in Reports.

1.4.12.82 ONELINE

(CVAL('ONELINE'))

Returns the value of the ONELINE operating condition (ON/OFF).

Used with R:BASE for DOS only. When set to ON NOTE and TEXT fields will never wrap to the next line in Reports and SELECTS. Instead they will be truncated at the end of the column.

1.4.12.83 OUTPUT

(CVAL('OUTPUT'))

Returns the current output value. The default is SCREEN.

If the screen is being used, SCREEN is returned. If the printer is being used, PRINTER will be returned. If a file is being used, the file name is returned. If more than one output is being used at a time, the values are returned in a comma-separated string.

1.4.12.84 PAGELOCK

(CVAL('PAGELOCK'))

Returns the value of the PAGELOCK operating condition (ON/OFF).

1.4.12.85 PAGEMODE

(CVAL('PAGEMODE'))

Returns the value of the PAGEMODE operating condition (ON/OFF).

1.4.12.86 PASSTHROUGH

(CVAL('PASSTHROUGH'))

Returns the value of the PASSTHROUGH operating condition (ON/OFF).

1.4.12.87 PLATFORM

(CVAL('PLATFORM'))

Returns the platform number and version for Windows and DOS based operating systems.

```
SET VAR vPlatform TEXT = (CVAL('PLATFORM'))

SHOW VAR vPlatform
WIN64  OS=Windows 10 Enterprise  Build=19043
```

1.4.12.88 PLUS

(CVAL('PLUS'))

Returns the value of the PLUS character setting.

1.4.12.89 PORTS

(CVAL('PORTS'))

Returns the list of all available printer ports, separated by comma, on that workstation.

Example:

```
SET VAR vAvailablePorts = (CVAL('PORTS'))
SHOW VARIABLE vAvailablePorts
```

Will return the text string with a list of all printer ports that are available on that workstation. Each item in the list will be separated by comma (or the database character settings for comma).

1.4.12.90 POSFIXED

(CVAL('POSFIXED'))

Returns the value of the POSFIXED operating condition.

1.4.12.91 PRINTERS

(CVAL('PRINTERS'))

Returns the list of all installed printers.

1.4.12.92 PRN_COLLATION

(CVAL('PRN_COLLATION'))

Captures the printer collation.

1.4.12.93 PRN_COLORMODE

(CVAL('PRN_COLORMODE'))

Captures the printer color mode.

1.4.12.94 PRN_COPIES

(CVAL('PRN_COPIES'))

Captures the printer copy count.

1.4.12.95 PRN_DUPLEXMODE

(CVAL('PRN_DUPLEXMODE'))

Captures the printer duplex mode.

1.4.12.96 PRN_ORIENTATION

(CVAL('PRN_ORIENTATION'))

Captures the printer orientation.

1.4.12.97 PRN_QUALITY

(CVAL('PRN_QUALITY'))

Captures the printer print quality (DPI).

1.4.12.98 PRN_SIZE

(CVAL('PRN_SIZE'))

Captures the printer paper size.

1.4.12.99 PRN_SOURCE

(CVAL('PRN_SOURCE'))

Captures the printer paper source.

1.4.12.100 PRN_STATUS

(CVAL('PRN_STATUS'))

Captures the printer status.

1.4.12.101 PROGRESS

(CVAL('PROGRESS'))

Returns the value of the PROGRESS operating condition.

1.4.12.102 QUALCOLS

(CVAL('QUALCOLS'))

Returns the value of the QUALCOLS operating condition.

1.4.12.103 QUALKEYS

(CVAL('QUALKEYS'))

Returns the columns assigned as a QualKeys for the current database.

1.4.12.104 QUALKEY TABLES

(CVAL('QUALKEY TABLES'))

Returns the tables assigned with QualKey columns for the current database.

1.4.12.105 QUOTES

(CVAL('QUOTES'))

Returns the value of the QUOTES character setting.

1.4.12.106 RBADMIN

(CVAL('RBADMIN'))

Returns the value of the RBADMIN operating condition.

1.4.12.107 REFRESH

(CVAL('REFRESH'))

Returns the value of the REFRESH operating condition.

1.4.12.108 REVERSE

(CVAL('REVERSE'))

Returns the value of the REVERSE operating condition.

1.4.12.109 ROWCOUNT

(CVAL('ROWCOUNT'))

Assigns line numbers for SELECT command output.

Example:

```
SELECT (INT(CVAL('ROWCOUNT'))) AS LineNo,Company=40 FROM Customer

 LineNo     Company
 ---------- ----------------------------------------
          1 RAM Data Systems, Inc.
          2 Johnson Technologies
          3 MIS by Design
          4 Barton & Associates
          5 The Data Shop
          6 Microcomputer Distribution
          7 Bytes & Words
          8 Lanufacturers Discount Computers
          9 Modular Software, Inc.
         10 Open Systems I/O
         11 Data Solutions
         12 Compumasters Computer Supply
         13 Datacrafters Infosystems
         14 Olympic Sales
         15 State University
         16 Computer Medical Ctr.
         17 Microtech University - II
         18 Softech Database Design
         19 Compdat Computer Consulting
         20 Nordan Distributors, Inc.
         21 Midtown Computer Co.
         22 Computer Warehouse - II
         23 PC Consultation And Design
         24 Industrial Concepts Inc.
         25 Computer Mountain Inc.
         26 Industrial Computers Inc.
         27 Microtech University - I
         28 Computer Warehouse - I
```

1.4.12.110 ROWLOCKS

(CVAL('ROWLOCKS'))

Returns the value of the ROWLOCKS operating condition.

1.4.12.111 RULES

(CVAL('RULES'))

Returns the value of the RULES operating condition.

1.4.12.112 SCRATCH

(CVAL('SCRATCH'))

Returns the value of the SCRATCH operating condition.

1.4.12.113 SCREENSIZE

(CVAL('SCREENSIZE'))

Returns the screen area in pixels.

1.4.12.114 SELMARGIN

(CVAL('SELMARGIN'))

Returns the value of the SELMARGIN operating condition.

1.4.12.115 SEMI

(CVAL('SEMI'))

Returns the value of the SEMI operating condition.

1.4.12.116 SERVER

(CVAL('SERVER'))

Returns the value of the SERVER operating condition.

1.4.12.117 SHORTNAME

(CVAL('SHORTNAME'))

Returns the value of the SHORTNAME operating condition.

1.4.12.118 SINGLE

(CVAL('SINGLE'))

Returns the value of the SINGLE operating condition.

1.4.12.119 SORT

(CVAL('SORT'))

Returns the value of the SORT operating condition.

1.4.12.120 SORTMENU

(CVAL('SORTMENU'))

Returns the value of the SORTMENU operating condition.

1.4.12.121 STATICDB

(CVAL('STATICDB'))

Returns the value of the STATICDB operating condition.

1.4.12.122 TABLELOCKS

(CVAL('TABLELOCKS'))

Returns a delimited text string for all tables with a lock.

Example:

```
SET VAR vTableLocks VARCHAR = (CVAL('TABLELOCKS'))

SHOW VAR vTableLocks
Contact,Customer,InvoiceDetail,InvoiceHeader
```

1.4.12.123 TIME

(CVAL('TIME'))

Returns the TIME format of the currently connected database.

1.4.12.124 TIME FORMAT

(CVAL('TIME FORMAT'))

Returns the default TIME FORMAT of the currently connected database.

1.4.12.125 TIME SEQUENCE

(CVAL('TIME SEQUENCE'))

Returns the default TIME SEQUENCE of the currently connected database.

1.4.12.126 TIMEOUT

(CVAL('TIMEOUT'))

Returns the value of the TIMEOUT operating condition.

1.4.12.127 TOLERANCE

(CVAL('TOLERANCE'))

Returns the value of the TOLERANCE operating condition.

1.4.12.128 TRACE

(CVAL('TRACE'))

Returns the status of the TRACE command parameter (ON/OFF).

1.4.12.129 TRANSACT

(CVAL('TRANSACT'))

Returns the value of the TRANSACT operating condition.

1.4.12.130 UINOTIF

(CVAL('UINOTIF'))

Returns the value of the UINOTIF operating condition.

1.4.12.131 USER

(CVAL('USER'))

Returns the R:BASE USER identifier.

1.4.12.132 USERAPP

(CVAL('USERAPP'))

Returns the file extensions to be displayed in the Object Manager.

Used with R:BASE 6.5++ for Windows only. You can specify which files display in the Object Manager after you click the Apps tab. Your choices are saved in the RBASE.CFG file. You can specify a maximum of three extensions.

1.4.12.133 USERDOMAIN

(CVAL('USERDOMAIN'))

Returns the name of the logged-in network domain. If the computer is not logged into a domain, the return value will be NULL.

1.4.12.134 USERID

(CVAL('USERID'))

Returns the Windows User ID.

1.4.12.135 UTF8

(CVAL('UTF8'))

Returns the value of the UTF8 operating condition.

1.4.12.136 VERIFY

(CVAL('VERIFY'))

Returns the value of the VERIFY operating condition.

1.4.12.137 VERSION

(CVAL('VERSION'))

Returns the product name, version, and build number of the R:BASE program.

See also:

(CVAL('BUILD'))
(CVAL('VERSION BUILD'))
SHOW BUILD
SHOW VERSION

1.4.12.138 VERSION BITS

(CVAL('VERSION BITS')

Returns the database file pointer version (32 or 64), based upon the R:BASE version in use.

1.4.12.139 VERSION BUILD

(CVAL('VERSION BUILD'))

Returns the build number for the R:BASE engine DLL.

The function may be useful in determining which features are available to you.

See also:

(CVAL('BUILD'))
(CVAL('VERSION'))
SHOW BUILD
SHOW VERSION

1.4.12.140 VERSION SYSTEM

(CVAL('VERSION SYSTEM'))

Returns the value WIN or DOS.

1.4.12.141 VIEWLOCKS

(CVAL('VIEWLOCKS'))

Returns a delimited text string for all views with a lock.

Example:

```
SET VAR vViewLocks VARCHAR = (CVAL('VIEWLOCKS'))

SHOW VAR vViewLocks
InvoicesMaster,ProductInvoice,StaffByDept
```

1.4.12.142 WAIT

(CVAL('WAIT'))

Returns the value of the WAIT operating condition.

1.4.12.143 WALKMENU

(CVAL('WALKMENU'))

Returns the value of the WALKMENU operating condition.

1.4.12.144 WAREKI

(CVAL('WAREKI'))

Returns the value of the WAREKI operating condition.

1.4.12.145 WHILEOPT

(CVAL('WHILEOPT'))

Returns the value of the WHILEOPT operating condition (ON/OFF).

1.4.12.146 WIDTH

(CVAL('WIDTH'))

Returns the value of the WIDTH operating condition.

1.4.12.147 WINAUTH

(CVAL('WINAUTH'))

Returns the value of the WINAUTH operating condition.

1.4.12.148 WINBEEP

(CVAL('WINBEEP'))

Returns the value of the WINBEEP operating condition.

1.4.12.149 WINDOWSPRINTER

(CVAL('WINDOWSPRINTER'))

This parameter returns the windows default printer.

1.4.12.150 WRAP

(CVAL('WRAP'))

Returns the value of the WRAP operating condition (ON/OFF).

1.4.12.15 WRITECHK

(CVAL('WRITECHK'))

Returns the value of the WRITECHK operating condition (ON/OFF).

1.4.12.15 ZERO

(CVAL('ZERO'))

Returns the value of the ZERO operating condition (ON/OFF).

1.4.12.15 ZOOMEDIT

(CVAL('ZOOMEDIT'))

Returns the value of the ZOOMEDIT operating condition (ON/OFF).

1.4.13 CVTYPE

(CVTYPE('colvarname',flag))

Returns the data type for a given column or variable name.

To return the data type for a given column, a zero "0" flag must be used. To return the data type for a given variable, the one "1" flag must be used.

Example 01.

```
SET VAR vCustIDType TEXT = (CVTYPE('CustID',0))
SET VAR vEmpCity TEXT = (CVTYPE('EmpCity',0))

SHOW VARIABLES

Variable           = Value            Type
------------------   ---------------  -------
vCustIDType        = INTEGER          TEXT
vEmpCity           = TEXT,20          TEXT
```

Example 02.

```
SET VAR vCustIDType TEXT = (CVTYPE('CustID',0))
SET VAR vVarInquiry TEXT = (CVTYPE('vCustIDType',1))

SHOW VARIABLES

Variable           = Value            Type
------------------   ---------------  ------
vEmpCity           = TEXT,20          TEXT
vVarInquiry        = TEXT,7           TEXT
```

Notes:

- When using the zero flag to return column data types, you must be connected to a database.
- When returning a TEXT data type, the length is included and is separated with a comma. When returning a NUMERIC data type, the precision and scale are separated with commas.

1.5 D

1.5.1 DATETIME

(DATETIME(*date,time*))

Concatenates date and time variables or constants.

The following expression concatenates the #DATE and #TIME values into a variable (*vdatetime*) that has a DATETIME data type.

```
SET VAR vdatetime=(DATETIME(.#DATE,.#TIME))
```

1.5.2 DECRYPT

(DECRYPT(*'string','password'*))

Returns the original string that was encrypted using the ENCRYPT function. A password must be specified to return the original string.

```
SET VAR vDecryptNumber TEXT = (DECRYPT('1CCF2D7B765F4715415D53471641','laser'))
SHOW VAR vDecryptNumber
8005552368
```

If the password specified with DECRYPT is not correct, the result will be a NULL valued string.

1.5.3 DELFUNC

(DELFUNC('function_name'))

Deletes a declared DLL function. If the DLL function is successfully deleted, a 1 is returned. If the DLL function is not deleted, a 0 is returned.

Example:

```
SET VAR v1 = (DELFUNC('FunctionName'))
```

See Also:

CHKFUNC
DLLCALL
DLLOAD
DLFREE
LIST FUNCTIONS

1.5.4 DEXTRACT

(DEXTRACT(*datetime*))

Returns the date portion of a value that has a DATETIME data type.

In the following example, the value of *vdextract* is *06/12/93*.

```
SET VAR vdextract = (DEXTRACT('06/12/93 12:15:30.123'))
```

1.5.5 DIM

(DIM(*arg1,arg2*))

Returns the positive difference between *arg1* and *arg2*. *Arg1* must be a value with a DOUBLE, REAL, NUMERIC, or INTEGER data type. *Arg2* must be any value with a DOUBLE, REAL, NUMERIC, or INTEGER

data type other than 0. The result is always positive or zero. If the result is less than or equal to zero, DIM returns *0*.

In the following example, the value of *vdim1* is *0*; *vdim2* is *2*; *vdim3* is *2*; *vdim4* is *0*.

```
SET VAR vdim1 = (DIM(2,4))
SET VAR vdim2 = (DIM(4,2))
SET VAR vdim3 = (DIM(-2,-4))
SET VAR vdim4 = (DIM(-4,-2))
```

1.5.6 DLCALL

The DLCALL Function calls any Windows dynamic link library (DLL) and loads it into memory for use with R:BASE.

Syntax for External DLL:

(DLCALL('libraryname.ext', 'FunctionOrProcedureName', [arg],[arg],[.....]))

Syntax for Windows API:

(DLCALL('libraryname', 'FunctionOrProcedureName', [arg],[arg],[.....]))

External DLLs must have the file name listed with the extension, such as 'MyLibrary.dll'.

Windows APIs require only the name of the Library, without the extension, such as: 'Kernel32' or 'User32'.

The DLLs must be created as Standard Windows 32-bit DLLs. Any number of functions or procedures can be used in a single DLL. Functions or Procedures to be used with DLCALL must be Exported in the DLL. No special code is necessary in the DLL for it to be used by R:BASE.

DLL Location:

DLLs can be located in the Legal Windows Search Path, and if elsewhere, then specify the FullPathName in DLLoad.

Search Path Used by Windows to Locate a DLL

With both implicit and explicit linking, Windows first searches for "known DLLs", such as Kernel32.dll and User32.dll. Windows then searches for the DLLs in the following sequence:

1. The directory where the executable module for the current process is located.
2. The current directory.
3. The Windows system directory. The GetSystemDirectory function retrieves the path of this directory.
4. The Windows directory. The GetWindowsDirectory function retrieves the path of this directory.
5. The directories listed in the PATH environment variable.

When or If DLLOAD is Used:

DLLOAD may be called "anytime" during the Session to Load the Library into Memory OR as a subsequent call to determine if the Library is Loaded.

In any event, DLLOAD is called internally when the Library is first referenced by DLCALL, and THEN the Library remains in memory until either the R:BASE session is ended OR DLFREE is called.

Data Type Rules:

Data types in the functions must be of the same storage size as the corresponding R:BASE data types.

Example: 32-bit Win32 Integer = 4 bytes of storage vs 32-bit R:BASE Integer = 4 bytes of storage

Declaration Logic:

Calls to a function OR procedure from ANY DLL must be declared at Least ONCE in the session in which they will be referenced.

Two Calling Conventions are supported, using the STDCALL and CDECL Keyword in the Declaration.

STDCALL

Function Declaration using STDCALL:

```
STDCALL FUNCTION 'functionName' (PTR VARCHAR (SIZE), ..... ) : VARCHAR (SIZE)
STDCALL FUNCTION 'functionName' ALIAS 'functionAliasName' (PTR TEXT
(SIZE), ..... ) : TEXT (SIZE)
```

Procedure Declaration using STDCALL:

```
STDCALL VOID FunctionOrProcedureThatHasNoReturnValue (PTR TEXT (SIZE))
```

Windows API Declaration using STDCALL:

```
STDCALL FUNCTION 'GetCurrentDirectoryA' ALIAS 'GetCurrentDir' (PTR TEXT, INTEGER )
: INTEGER
```

CDECL

Function Declaration using CDECL:

```
CDECL FUNCTION 'functionName' (INTEGER) : INTEGER
CDECL FUNCTION 'functionName' ALIAS 'functionAliasName' (PTR DOUBLE) : DOUBLE
```

Procedure Declaration using CDECL

```
CDECL VOID FunctionOrProcedureThatHasNoReturnValue (PTR TEXT (SIZE))
```

Parameters

'SIZE' applies to TEXT and VARCHAR data types.

Important Notes:

A. Parameters in the Declaration are in the REVERSE order from the Actual Function or Procedure in the DLL.
Parameters can be 0 to n Parameters of any legal R:BASE data type.

B. Function Names ARE CASE SENSITIVE in the DECLARATION ONLY.
The Case must match the casing used in the DLL.
Case is INSENSITIVE when used in DLCALL.

Remarks:

- For best results, TEXT and VARCHAR data should be passed with the PTR (Pointer) Attribute and ANY VARCHAR data type Larger than 32K as a Parameter, MUST be passed as PTR.

- VARCHAR data type as a Return Value restricted to 32K, but Any SIZE up to 256MB can be passed as a Pointer to the R:BASE Variable. Modification of the Data Passed as Pointer must be on the data pointed to.

- It is important that the SIZE parameter on TEXT and VARCHAR be Specified to avoid creating excess buffer space that has to be created from the Declaration.

For Example:

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( ptr varchar (nn)) : integer
```
Then *nn* <= 256Mb.

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( ptr varchar ) : integer
```
Then because SIZE is omitted, the buffer for VARCHAR will have default SIZE = 256MB.
*** AVOID THIS UNLESS THAT IS THE ACTUAL SIZE OF THE DATA TO BE PASSED!**

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( integer ) : varchar(nn)
```
Then because SIZE is Specified, nn bytes <= 32K will be returned.

If the Function is Declared like this:
```
STDCALL function 'somefunction' ( integer ) : varchar
```
Then because SIZE is omitted, the buffer for VARCHAR will have default SIZE = 32K.

When SIZE is Specified, the data passed as parameter or as return value, if Greater than the SIZE, will be truncated to SIZE.

Example of a DLL created in Delphi exporting three functions:

```
// Begin Dll Code
library DemoLib;

uses
  SysUtils, Classes;

{$R *.res}

function MultInt (NumIN : Integer) : Integer; stdcall;
begin
  Result := (NumIN * 2);
end;

function MultDbl (NumDbl : Double) : Double; stdcall;
begin
  Result := (NumDbl * 2);
end;

procedure LCaseByREF(DataIN : PChar); stdcall;
begin
  ansiStrLower(DataIN);
end;

function LCaseByVAL (DataIN : PChar) : PChar; Stdcall;
begin
  Result := ansiStrLower(DataIN);
end;

Exports MultInt, LCaseByREF, LCaseByVAL;

begin

end.
// End Dll Code
```

Example Usage From Within R:BASE:

```
-- BEGIN Demo.rmd
-- Declare the functions to be used from the DLL
STDCALL function 'MultInt' ( Integer ) : Integer
STDCALL VOID 'LCaseByREF' (ptr TEXT (30))
STDCALL function 'LCaseByVAL (ptr TEXT (60)) : TEXT (60)

--Set somme variables for use
Set VAR vTEXT TEXT = 'RBASE TECHNOLOGIES'
SET VAR vINT INTEGER = 128
SET VAR v1 INTEGER = 0

-- OPTIONALLY CALL DLLOAD
SET VAR v1 = (DLLOAD('DemoLib.dll'))
IF v1 = 0 THEN
  PAUSE 2 USING 'DemoLib.dll NOT LOADED.. EXITING'
  RETURN
ENDIF

SET VAR V1 = (dlcall('demolib.dll', 'changecase', vtext))

{ The Value for v1 will be null because ChangeCase is a procedure and
doesn't
  RETURN A RESULT, but the value of vTEXT which is passed as a POINTER has
been
  changed to 'rbase technologies'}

SET VAR v1 = (DLCALL('demolib.dll', 'MultInt', vINT))

--The value for v1 will be 256 the value returned from the function.

-- running the following against RRBYW20
SELECT (DLCALL('demolib.dll','lcasebyval', Company))=60 FROM +
Customer WHERE LIMIT = 2

{Yields the following output:
 (DLCALL('demolib.dll','lcasebyval', Company)
 ------------------------------------------------------------
 computer warehouse - ii
 microtech university - i
}

SELECT ((ICAP2((DLCALL('demolib.dll','lcasebyval', Company))))) = 60 +
FROM Customer WHERE LIMIT = 2

{Yields the following otput:
 ((ICAP (DLCALL('demolib.dll','lcasebyval',
 ------------------------------------------------------------
 Computer Warehouse - Ii
 Microtech University - I
}
```

```
-- Optionally CALL DLFREE
SET VAR v1 = (DLFREE('DemoLib.dll'))

-- END Demo.rmd
```

Example of a DLL created in C++ Exporting three functions:

```
// BEGIN C++ DLL
// loaddll.cpp : Defines the entry point for the DLL application.
//
#include <windows.h>
#include <stdio.h>

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                          )
{
    return TRUE;
}
#ifdef __cplusplus    // If used by C++ code,
extern "C" {          // we need to export the C interface
#endif

__declspec(dllexport) int cfunc1(int i){
        return i;
}
__declspec(dllexport) double cfunc2(double *inp){
        double rtn = *inp;
        rtn++;
        return rtn;
}
__declspec(dllexport) char * cfunc3(char *inp){
        strcat(inp," + ");
        return inp;
}

#ifdef __cplusplus
}
#endif

// END C++ DLL
```

See Also:

CHKFUNC
DELFUNC
DLLOAD
DLFREE
LIST FUNCTIONS

Special thanks to:
Mike Byerley (Fort Wayne, Indiana) for his contribution to the introduction, implementation and testing of the DLCALL function in R:BASE.

1.5.7 DLFREE

(DLFREE('libraryname.ext'))

Checks to see if a given library file can be freed.

The function returns an integer value of 1 if the library is freed and 0 if a given library is not freed.

Example:

```
SET VARIABLE vFreePlugin = (DLFREE('RCharts11.RBM'))
```

See Also:

CHKFUNC
DELFUNC
DLLCALL
DLLOAD
LIST FUNCTIONS

1.5.8 DLLOAD

(DLLOAD('libraryname.ext'))

Checks to see if a given library file is loaded.

The function returns an integer value of 1 if the library is loaded and 0 if a given library is not loaded.

Example:

```
SET VARIABLE vLoadPlugin = (DLLOAD('RCharts11.RBM'))
```

See Also:

CHKFUNC
DELFUNC
DLLCALL
DLFREE
LIST FUNCTIONS

1.5.9 DNW

(DNW(*date*))

Returns the date value for the next working (business) day.

This fuction recognizes Monday through Friday as business days and does not recognize Saturday or Sunday. Holidays are not considered.

1.5.10 DWE

(DWE(*date*))

Returns the date value for the next weekend day.

This fuction recognizes Saturday or Sunday as weekend days and does not recognize Monday through Friday. Holidays are not considered.

1.5.11 DWRD

(DWRD(*value*)) or **(DWRD(*value,arg*))**

Converts a currency value to its word representation.

The DRWD function results will differ based upon the CURRENCY symbol value.

With the CURRENCY symbol set to the dollar character ($), the DWRD results include "dollars" and "cents" in the word representation. With the CURRENCY symbol set to any other currency symbol (euro, franc, peso, etc.) including a blank, the DWRD results display fractional whole dollars in the word representation.

The *arg* parameter is optional, which allows for fractional whole dollars to be returned for the cents portion of the amount, even if the currency symbol is set to the dollar character. The *arg* value as 0 will return "dollars" and "cents" in the word representation. The *arg* value as 1 will return fractional whole dollars in the word representation. Using the DWRD arg parameter, users are not required to alter the CURRENCY symbol to display fractional whole dollars. Omitting the optional parameter is the same as 0 for the optional parameter. A non-zero optional parameter provides the fractional whole dollars.

Examples

Example 01:
```
-- Uses the dollar CURRENCY symbol
SET CURRENCY '$'
SET VAR vAmount CURRENCY = 10.50
SET VAR v1 TEXT = (DWRD(.vAmount))
SHOW VAR v1
ten dollars and fifty cents
```

Example 02:
```
-- Uses a blank CURRENCY symbol
SET CURRENCY ' '
SET VAR vAmount CURRENCY = 10.50
SET VAR v2 TEXT = (DWRD(.vAmount))
SHOW VAR v2
ten and 50/100
```

Example 03:
```
-- Uses the dollar CURRENCY symbol and returns fractional whole dollars
SET CURRENCY '$'
SET VAR vAmount CURRENCY = 10.50
SET VAR v3 TEXT = (DWRD(.vAmount,1))
SHOW VAR v3
ten and 50/100 dollars
```

Example 04:
```
-- Uses the dollar CURRENCY symbol and returns the word representation with the ICAP3
function
SET CURRENCY '$'
SET VAR vAmount CURRENCY = 10.50
SET VAR v3 TEXT = (ICAP3(DWRD(.vAmount,0)))
SHOW VAR v3
Ten Dollars and Fifty Cents
```

1.6 E

1.6.1 ENCRYPT

(ENCRYPT(*'string','password'*))

Returns an encrypted string of HEX characters, which is twice as long as the original string, plus 8 more characters. A password must be specified to use with the DECRYPT function in order to return the original string.

```
SET VAR vEncrptNumber TEXT = (ENCRYPT('8005552368','laser'))
SHOW VAR vEncrptNumber
1CCF2D7B765F4715415D53471641
```

Once a value is encrypted, users cannot tell what the value is by selecting the raw encrypted value. If a form retrieves the encrypted value, it can then call the DECRYPT function the value to get the original value.

1.6.2 ENVVAL

(ENVVAL(*'environmentvar'*))

Returns the current value of the specified DOS environment variable. You must either enclose the name of the environment variable in quotation marks or use a text variable to which you have assigned the environment variable.

First, assume you have the command below in your AUTOEXEC.BAT file.

```
SET workstat=10
```

Now, in R:BASE, you can use ENVVAL to find the value of that environment variable, as shown in the example below. The value of *vworkstat* will be the text value *10*.

```
SET VAR vworkstat = (ENVVAL('workstat'))
```

1.6.3 EXP

(EXP(*arg*))

Raises *e* to the *arg* power (where *e* = 2.71828182845905 and *arg* is any value with a DOUBLE, REAL, NUMERIC, or INTEGER data type). By raising *e* to an exponent, EXP performs the inverse operation of LOG.

In the following example, the value of *vexp* is *2.71828182845905*.

```
SET VAR vexp = (EXP(1))
```

1.7 F

1.7.1 FILENAME

(FILENAME(0))

Generates a unique filename with a .$$$ extension, and creates the file in the current SCRATCH directory.

1.7.2 FINDFILE

(FINDFILE('*filename*'))

Returns the location of a file. The function looks first in the current directory and then searches the DOS path for the file. If the file is found, the full path name is returned, if it isn't found, a NULL is returned. Wildcards in the filename will produce unpredictable results.

1.7.3 FISHER

(FISHER(*arg*))

Returns the Fisher transformation at *arg*. This transformation produces a function that is normally distributed rather than skewed. Use this function to perform hypothesis testing on the correlation coefficient.

The *arg* value must be a number value between -1 and 1. The resulting variable value is the DOUBLE data type.

Example:

```
SET VAR vFisherResult DOUBLE = (FISHER(0.063))

SHOW VAR vFisherResult
     0.063083548051763
```

1.7.4 FLOAT

(FLOAT(*arg*))

Converts a number with a TEXT, INTEGER, or CURRENCY data type to a value with a REAL or DOUBLE data type. This is not the same as the FLOAT data type.

In the following example, the value of *vfloat1* is *2.*, a number that has a REAL data type, and the value of *vfloat2* is also *2.*, a number that has a DOUBLE data type. If a variable is not assigned a data type, it becomes DOUBLE.

```
SET VAR vfloat1 REAL = (FLOAT(2))
SET VAR vfloat2 = (FLOAT(2))
```

1.7.5 FORMAT

(FORMAT (*value*,'*picture-format*'))

Prints picture formats to a variable, rather than only to the screen. You can use FORMAT anywhere that you can use a function. The result of the FORMAT function is always text.

In the syntax for this function, *value* is the value you want to be displayed in a particular format; it can be a column, variable, or a constant value. *'Picture-format'* is the picture format you establish.

The FORMAT function can be useful in several ways:

- Aligning decimals
- Capturing date and time using system variables
- Formatting currency
- Formatting text
- Punctuating long numbers

The characters you can use to format your data are listed below.

For All Data

[<]	Data is left justified.

[>]	Data is right justified.
[^]	Data is centered.

For Numbers

[-]	Places a minus sign to the right of a negative number.
[DB]	Places DB to the right of a negative number.
[( )]	Encloses a negative number in parentheses.
[CR]	Places CR to the right of a positive number.
9	Fills unused space with blanks.
0	Fills unused space with zeros.
*	Fills unused space with asterisks.

For Text

_	Letters are uppercase; other characters are blank.
|	Letters are lowercase; other characters are blank.
%	Letters are uppercase; other characters are unchanged.
?	Letters are lowercase; other characters are unchanged.

For Dates

MMDDYYYY	Displays the month, day, and year
MM	Displays the month
DD	Displays the day
YYYY	Displays the year
WWW	Displays the day name, with a 3-letter abbreviation
WWW+	Displays the full name for the day of the week
MMM	Displays the month name, with a 3-letter abbreviation
MMM+	Displays the full name for the month
CC	Displays the century, AD or BC
any combination	Any combination of the month, day and year can be used.

For Times

HHMMSS	Displays the hour, minute, and second
HH	Displays the hour
MM	Displays the minute
SS	Displays the second
.SSS	Displays thousandths of a second
AP	Displays AM or PM when using a 12-hour format
any combination	Any combination of the hours, minutes, and seconds can be used.
NN	Displays minutes (when capturing date and time using #NOW)

Samples:

Aligning Decimals
The following example shows how you can use the FORMAT function to align decimal points in a column:

```
SELECT (FORMAT(bonuspct,'99.000')) FROM salesbonus
```

The following example shows the effect of the FORMAT function on the above SELECT statement:

Using FORMAT	Without FORMAT
0.003	0.003
0.002	0.002
0.000	0
0.001	0.001

Formatting Currency

The following example shows how you can use the FORMAT function to only display whole dollars:

```
SELECT (FORMAT(netamount,'[>]$999,999')) FROM salesbonus
```

This SELECT statement displays data as right justified whole dollars, as shown below:

Using FORMAT	Without FORMAT
$176,000	$176,000.00
$87,500	$87,000.00

Formatting Text
You must include a format character for each text character. The following example shows how you can use the FORMAT function to display text in uppercase:

```
SELECT (FORMAT(empfname,'________')) FROM employee
```

Using FORMAT	Without FORMAT
JUNE	June
ERNEST	Ernest
PETER	Peter

Punctuating Long Numbers
The following example shows how you can use the FORMAT function to include a comma after the thousand's place:

```
SELECT (FORMAT(transid,'999,999')) FROM transmaster
```

The following shows the effect of the FORMAT function on the above SELECT statement:

Using FORMAT	Without FORMAT
104	104
2,002	2002
39,765	39765

Capturing Date and Time Using System Variables
Here's a simple routine to create a unique file name by capturing the date and time using R:BASE system variables:

```
-- Example 01: (Using .#DATE and .#TIME as two separate variables)
  CLS
  CLEAR VARIABLES vDateTxt,vTimeTxt,vFileName
  SET VAR vDateTxt TEXT = NULL
  SET VAR vTimeTxt TEXT = NULL
  SET VAR vFileName TEXT = NULL
  SET VAR vDateTxt = (FORMAT(.#DATE,'MMDDYYYY'))
  SET VAR vTimeTxt = (FORMAT(.#TIME,'HHMM'))
  SET VAR vFileName = +
  ((CVAL('DATABASE'))+'_'+.vDateTxt+'_'+.vTimeTxt+'.BKP')
  CLEAR VARIABLES vDateTxt,vTimeTxt
  RETURN
-- vFileName will return the text variable as:
-- RRBYW20_12302020_1630.BKP
-- Database Name, Date and Time will vary on your end

-- Example 02: (Using .#DATE and .#TIME as one variable in expression)
  CLS
  CLEAR VARIABLES vFileName
  SET VAR vFileName TEXT = NULL
```

```
  SET VAR vFileName = +
  ((CVAL('DATABASE'))+'_'+(FORMAT(.#DATE,'MMDDYYYY'))+ +
  '_'+(FORMAT(.#TIME,'HHMM'))+'.BKP')
  RETURN
-- vFileName will return the text variable as:
-- RRBYW20_12302020_1630.BKP
-- Database Name, Date and Time will vary on your end

-- Example 03: (Using .#NOW) - Short and Swift
  CLS
  CLEAR VARIABLES vFileName
  SET VAR vFileName TEXT = NULL
  SET VAR vFileName = +
  ((CVAL('DATABASE'))+''-'+(FORMAT(.#NOW,'MMDDYYYY_HHNN'))+'.BKP')
  RETURN
-- Notice the "NN" for minutes when using .#NOW (not a typo)
-- vFileName will return the text variable as:
-- RRBYW20_12302020_1630.BKP
-- Database Name, Date and Time will vary on your end
```

Now that you have successfully created a unique file name, you may use the following routine to make a backup of your live database, on demand! Here's how:

```
  OUTPUT .vFileName
  UNLOAD ALL
  OUTPUT SCREEN
```

The resulting two files will be created in the same folder, unless a different folder name is specified:

- RRBYW20_12302020_1630.BKP
- RRBYW20_12302020_1630.LOB

1.7.6 FORMAT2

(FORMAT2(arg,nchar1,nchar2,fill))

Formats a numeric value to a text string allowing a fill character to be loaded to the left of the numeric value. The two width arguments are used specify the total text string width and the characters to preserve right of decimal, including the decimal itself.

(FORMAT2(value, width1, width2, fill)) width1 = total width, width2 = one more than characters to right of decimal, fill = left fill character

Example 01:

```
SET VAR vFormatString TEXT = (FORMAT2(379.2418,14,3,'.'))
SHOW VAR vFormatString
..............379.24
```

Example 02:

```
SET VAR vFormatString TEXT = (FORMAT2(625.79,20,5,'*'))
SHOW VAR vFormatString
************625.7900
```

1.7.7 FSTATUS

(FSTATUS('*filespec*'))

Checks to see if a file or folder name exists, and also returns the status for a file. If no path is specified, the function checks for the file or folder name in the current directory. Otherwise, the function checks for the file or folder name in the specified location.

Return Value	Status
0	file/folder does not exist
1	file/folder exists, but cannot be read or written to. Folders will return this value, as well as file name searches with wild cards.
2	file is marked as a read only
3	file is not marked as read only, but can only read it at this time (the file is opened by another program)
4	file can be read and written to
+10	If the file is marked as hidden, the return value will be incremented by 10; 12 = read only and hidden 14 = can read and write and hidden

Examples:

Example 01 (The brates.xlsx file exists, is marked hidden, and is opened):

```
SET VAR vFileStatus INTEGER = (FSTATUS('brates.xlsx'))
SHOW VAR vFileStatus
13
```

Example 02 (The CheckVersion.rmd file exists and is marked as read only):

```
SET VAR vFileStatus INTEGER = (FSTATUS('CheckVersion.rmd'))
SHOW VAR vFileStatus
2
```

1.7.8 FV1

(FV1(*pmt*,*int*,*per*))

Returns the future value of a series of equal payments of the amount, *pmt*, periodic interest rate, *int*, and compounding periods, *per*.

In the following example, FV1 returns the ending balance in your savings account if you deposit $300 each month for four years with an annual interest rate of 6.25%. The value of *vfv1* (the balance) is *$16,311.90*.

```
SET VAR vfv1 = (FV1(300,(.0625/12),(4 *12)))
```

1.7.9 FV2

(FV2(*pv*,*int*,*per*))

Returns the future value of an amount, *pv*, invested at a periodic interest rate, *int*, for compounding periods, *per*.

In the following example, FV2 returns your ending balance for a savings account where you have deposited $1,800 with an annual interest rate of 5.5% compounded monthly (simple interest divided by compounding periods) for seven years (12 periods times seven years). The value of *vfv2* (the balance) is *$2,642.98*.

```
SET VAR vfv2 = (FV2(1800,(.055/12),(7 *12)))
```

1.8 G

1.8.1 GETDATE

(GETDATE('*parameters*'))

GETDATE will display a graphic calendar interface with today's date circled. After a day is selected, the date is returned to the variable. Based on the calendar type, different methods are used to navigate the months and years.

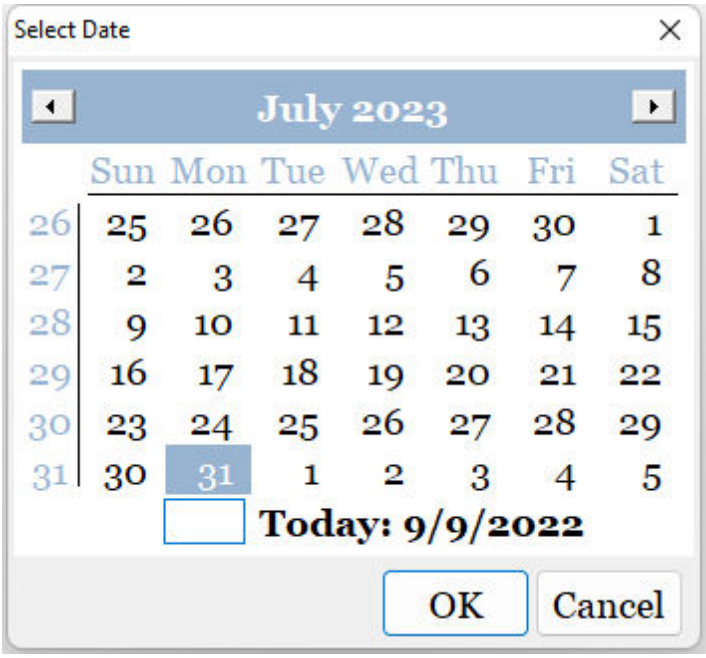
Parameters	**Values**	**Description**
calendar caption	value	specifies a caption for the calendar window
CALENDAR_TYPE	STANDARD ENHANCED	specifies the calendar type. STANDARD will display the default calendar for the operating system. ENHANCED will display a custom calendar.
CALENDAR_FONT_NAME	value	specifies the font name for the calendar
CALENDAR_FONT_SIZE	value	specifies the font size for the calendar
FONT_NAME	value	specifies the font name for the calendar and buttons
FONT_SIZE	value	specifies the font size for the calendar and buttons
INIT_VAR	variable	specifies the initial value variable assigned with the default date
SHOW_TODAY_CIRCLE	ON OFF	specifies whether to display the circle for today's date
THEMENAME	value	specifies a theme that will be applied to the calendar window

Notes:

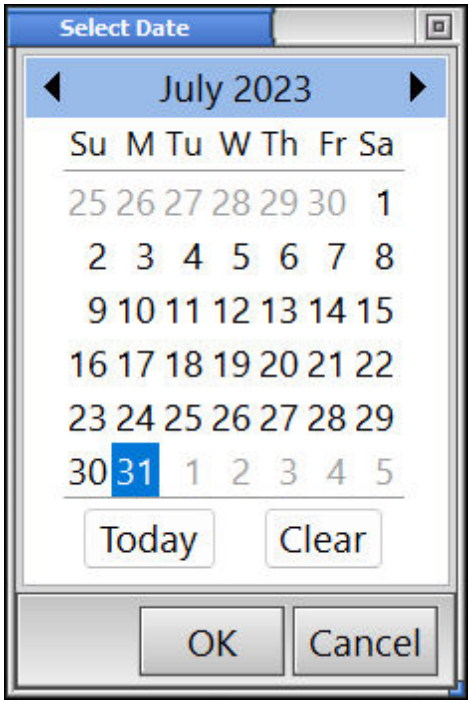
- The parameters must be enclosed in quotes.
- The parameters must be separated with the pipe character "|".
- By default, the CALENDAR_TYPE is STANDARD and SHOW_TODAY_CIRCLE is ON.
- Using the enhanced calendar, an additional "Clear" button is available to clear the selection. Both calendar types allow to select a today option and use today's date as the selected value.

Examples:

```
-- Example 01 (Standard calendar with a default date, font name, and font size):
SET VAR vDate DATE = NULL
SET VAR vDefaultDate DATE = 07/31/2020
SET VAR vDate = (GETDATE('Select Date|INIT_VAR vDefaultDate|FONT_NAME Georgia|
FONT_SIZE 14'))
RETURN
```

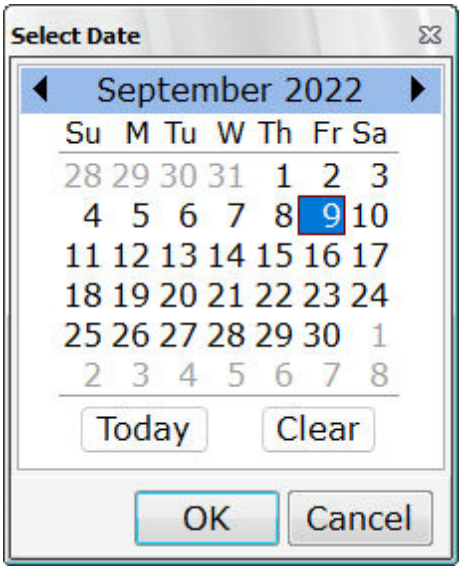



```
-- Example 02 (Enhanced calendar with a default date, theme, and custom font size):
SET VAR vDefaultDate DATE = 07/31/2020
SET VAR vDate = +
(GETDATE('Select Date|INIT_VAR vDefaultDate|CALENDAR_TYPE ENHANCED|THEMENAME QNX|
FONT_SIZE 14'))
RETURN
```




```
-- Example 03 (Dynamic approach using global variables)
SET VAR vDate DATE = NULL
```

```
SET VAR vCaption TEXT = 'Select Date'
SET VAR vCalendarType TEXT = 'ENHANCED'
SET VAR vThemeName TEXT = 'Vista CG'
SET VAR vQuotes = (CVAL('QUOTES'))
SET VAR vFontSize TEXT = '12'
SET VAR vFontName TEXT = 'Verdana'
SET VAR vString TEXT = +
('SET VAR vDate = (GETDATE('+.vQuotes+.vCaption+ +
'|CALENDAR_TYPE '+.vCalendarType+ +
'|THEMENAME '+.vThemeName+ +
'|FONT_SIZE '+.vFontSize+ +
'|FONT_NAME '+.vFontName+.vQuotes+'))')
&vString
CLEAR VARIABLE vCaption,vCalendarType,vThemeName,+
vQuotes,vString,vFontSize,vFontName
RETURN
```



1.8.2 GETKEY

(GETKEY(0))

Gets the text value, in brackets, of the first key available in the type-ahead buffer. If a key is not available, GETKEY waits for the next keystroke. Since R:BASE normally checks the buffer for the [Ctrl] + [Break] keys, you must set ESCAPE to off for GETKEY to work properly. Use CHKKEY before GETKEY to determine if a key is available. GETKEY does nothing with the zero that you enter in parentheses; GETKEY returns a value without receiving one.

If you are executing a process that takes several minutes to complete, you can use the CHKKEY and GETKEY functions to tell R:BASE what to do next, even while the process is executing.

1.8.3 GETPROPERTY

(GETPROPERTY('arg1','arg2'))

The GETPROPERTY function captures the current properties of forms, form controls, applications, reports, report controls, labels, and label controls. The function behaves just as the GETPROPERTY command.

arg1 is the Component ID value or the APPLICATION, RBASE_FORM, REPORT, or RBA_FORM keyword that the property is being captured from.

arg2 is the name of the object's property you wish to get the value for.

APPLICATION
Specifies to capture an application property such as the CAPS lock status, if it is compiled, or the title

Component ID
Specifies the unique identifier assigned to a control (e.g. DB Edit). The Component ID within the GETPROPERTY function must match the unique identifier listed in the control properties. The "Component ID" field is located in the Object Properties for all forms, reports, labels and controls. A "Component ID" option is also located in the speed menu list when you right click on a report/label control.

RBA_FORM
Specifies to capture an application form property such as the current theme, the enabled status for an action, or a caption

RBASE_FORM
Specifies to capture a form property such as height, width, etc.

REPORT
Specifies to capture a report property such as the report name, the total page count, or the number of tables used in the report

Tips:

- The properties of any form control objects can be captured with assigned Component ID values ONLY.
- Always use the current QUOTES character around the variable name.
- The resulting variable value will be TEXT. To manipulate any values to INTEGER, use the INT, FLOAT, or other conversion function.
- To change any property of a form control after the form is displayed, use the PROPERTY command(s) within the "On After Start" EEP section of the Form Properties.

Notes:

- The complete list of GETPROPERTY command parameters are available within the **FormProperties.pdf** and **FormProperties_TOC.pdf** PDF documents, which are provided within the R:BASE program directory and are available for download at the R:BASE Technologies Support page: http://www.rbase.com/support/
- An optional syntax builder add-on product called R:Docs, containing all PROPERTY/GETPROPERTY commands and parameters constructed in an R:BASE application, is also available. Please contact the R:BASE Technologies Sales Staff at sales@rbase.com if you wish to acquire an annual subscription to the stand-alone R:Docs database and application.

Examples:

Example 01.
-- To check an applied "Column Sort" setting value (ON or OFF) for a Variable Lookup List View and alter the control properties for the List View and a button object.

```
IF (GETPROPERTY('VarLookupListView','COLUMN_SORT')) = 'TRUE' THEN
   PROPERTY VarLookupListView COLUMN_SORT 'FALSE'
   PROPERTY Btn_ShowSortStatus FONT_COLOR 'GREEN'
   PROPERTY Btn_ShowSortStatus CAPTION 'Enable Column Sort'
ELSE
   PROPERTY VarLookupListView COLUMN_SORT 'TRUE'
   PROPERTY Btn_ShowSortStatus FONT_COLOR 'RED'
   PROPERTY Btn_ShowSortStatus CAPTION 'Disable Column Sort'
ENDIF
```

Example 02.
-- Captures the value for a report label

```
SET VARIABLE vCoverPageTot = (FLOAT((GETPROPERTY('CoverPageInvoiceTotal','VALUE'))))
```

Example 03.
-- Captures the form caption
```
SET VAR vCaption TEXT = (GETPROPERTY('RBASE_FORM','CAPTION'))
```

Example 04.
-- Checks if the program is an R:BASE compiled application
```
SET VAR vIsCompiled TEXT = (GETPROPERTY('APPLICATION','ISCOMPILED'))
```

Example 05.
-- Captures the page count for a report
```
SET VAR vPageCount = (GETPROPERTY('REPORT','TOTALPAGECOUNT'))
```

1.8.4 GETVAL

(GETVAL('arg1', 'arg2'))

Gets a value based on the argument data provided. The followng GETVAL arguments are available:

- CheckMessageStatus
- GetDriveReady
- GetIPAddress
- GetLock
- GetMACAddr
- GetVolumeID
- PlayAndExit
- PlayAndWait

1.8.4.1 CheckMessageStatus

(GETVAL('CheckMessageStatus','####'))

Example:

```
SET VAR vStatus TEXT = NULL
SET VAR vStatus = (GETVAL('CheckMessageStatus','2038'))
```

Variable vStatus will return the text value of the current message status (ON/OFF) for error message 2038.

1.8.4.2 GetDriveReady

(GETVAL('GetDriveReady','driveletter'))

Example:

```
SET VAR vReady = (GETVAL('GetDriveReady','A'))
```

Result: False if not ready, True if ready.

1.8.4.3 GetIPAddress

(GETVAL('GetIPAddress','n'))

Where 'n' is the parameter to either retrieve the number of active network adapters or to retrieve the IP address of a given active network adapter. Use '0' to retrieve the number of active network adapters and 1-9 to retrieve the IP address(es) of active network adapter(s) of a given network workstation/server.

Example:

```
-- Start
```

```
-- GetIPAddress.RMD
CLS
CLEAR VARIABLE vActiveAdapters,vIPAddress1,vIPAddress2, +
vIPAddress3,vIPAddress4,vPauseMessage,vCaption
SET VAR vActiveAdapters TEXT = NULL
SET VAR vIPAddress1 TEXT = NULL
SET VAR vIPAddress2 TEXT = NULL
SET VAR vIPAddress3 TEXT = NULL
SET VAR vIPAddress4 TEXT = NULL
SET VAR vPauseMessage TEXT = NULL
SET VAR vCaption TEXT = 'Understanding New GETVAL Function'

-- To retrieve the number of active network adapters
SET VAR vActiveAdapters = (GETVAL('GetIPAddress','0'))

-- To retrieve the IP address of first active network adapter
SET VAR vIPAddress1 = (GETVAL('GetIPAddress','1'))

-- To retrieve the IP address of second active network adapter
SET VAR vIPAddress2 = (GETVAL('GetIPAddress','2'))

-- To retrieve the IP address of third active network adapter
SET VAR vIPAddress3 = (GETVAL('GetIPAddress','3'))

-- To retrieve the IP address of fourth active network adapter
SET VAR vIPAddress4 = (GETVAL('GetIPAddress','4'))

SET VAR vPauseMessage = +
('Number of Active Adapter(s):'+(CHAR(009))&.vActiveAdapters+ +
  (CHAR(009))+(CHAR(013))+ +
 'IP Address of Active Adapter 1:'+(CHAR(009))&.vIPAddress1+ +
  (CHAR(009))+(CHAR(013))+ +
 'IP Address of Active Adapter 2:'+(CHAR(009))&.vIPAddress2+ +
  (CHAR(009))+(CHAR(013))+ +
 'IP Address of Active Adapter 3:'+(CHAR(009))&.vIPAddress3+ +
  (CHAR(009))+(CHAR(013))+ +
 'IP Address of Active Adapter 4:'+(CHAR(009))&.vIPAddress4+ +
  (CHAR(009))+(CHAR(013)))

CLS
PAUSE 2 USING .vPauseMessage CAPTION .vCaption +
ICON APP +
Button 'Yes, this is the R:BASE you have always wanted!' +
OPTION BACK_COLOR WHITE +
|MESSAGE_COLOR WHITE +
|MESSAGE_FONT_COLOR GREEN +
|BUTTON_COLOR WHITE
CLS
CLEAR VARIABLE vActiveAdapters,vIPAddress1,vIPAddress2, +
vIPAddress3,vIPAddress4,vPauseMessage,vCaption
RETURN
-- end
```

1.8.4.4 GetLock

(GETVAL('GetLock','tableviewname'))

GetLock is the first required parameter and the table/view name is the name of the table or view. Use this function to find the LOCK status of a table or view. The returning value is ON or OFF, depending on whether a lock is in place upon the table or view.

Example:

```
SET VAR vCheckLock = (GETVAL('GetLock','Customer'))
```

vCheckLock will return the value of ON or OFF for the *Customer* table

1.8.4.5 GetLockType

(GETVAL('GetLockType','tableviewname'))

GetLockType is the first required parameter and the table/view name is the name of the table or view. Use this function to find the lock type of a table or view. The returning value may be one of the following: None, Local, Remote, Cursor, Row

Example:

```
SET VAR vCheckLockType TEXT = (GETVAL('GetLockType','Customer'))
```

vCheckLockType may return the value of "Cursor" for the *Customer* table if a DECLARE CURSOR lock is on the table.

1.8.4.6 GetMACAddr

Use (GETVAL('GetMACAddr','n')) to retrieve the number of active network adapter(s) on workstation/server as well as the physical MAC (Media Access Control) address(es) of active network adapter(s).

Media Access Control address is a physical hardware address that uniquely identifies each node of a network. In IEEE 802 networks, the Data Link Control (DLC) layer of the OSI Reference Model is divided into two sub-layers: the Logical Link Control (LLC) layer and the Media Access Control (MAC) layer. The MAC layer interfaces directly with the network media. Consequently, each different type of network media requires a different MAC layer.

```
(GETVAL('GetMACAddr','n'))
```

Where 'n' is the parameter to either retrieve the number of active network adapters or to retrieve the MAC address of given active network adapter. Use '0' to retrieve the number of active network adapters and 1-9 to retrieve the MAC address of active network adapter(s) on given network station/server.

Examples:

Example 01: To get the number of active network adapter(s)

```
SET VAR vActiveAdapters = (GETVAL('GetMACAddr','0'))
```

The variable vActiveAdapters will return the number of active network adapters on that work station/server.

Example 02: To retrieve the MAC address of first active adapter

```
SET VAR vMACAddress = (GETVAL('GetMACAddr','1'))
```

The variable vMACAddress will return the value of first active network adapter on that workstation.

Example 03: To retrieve the MAC address of second active adapter

```
SET VAR vMACAddress = (GETVAL('GetMACAddr','2'))
```

The variable vMACAddress will return the value of second active network adapter on that workstation.

Example 04: To retrieve the MAC address of third active adapter

```
SET VAR vMACAddress = (GETVAL('GetMACAddr','3'))
```

The variable vMACAddress will return the value of third active network adapter on that workstation.

Practically, this new (GETVAL('GetMACAddr','n')) function can be used to customize access to your R:BASE for Windows Applications. Consequently, you can use this function to customize the properties of any control and/or settings of form using the PROPERTY command based on unique MAC address of workstation.

Example 05: Disabling/Hiding Application Menus based on MAC Address

Assuming your application includes 6 main menu options, namely Customers, Contacts, Employees, Products, Sales, Quarterly Sales Reports, General Inquiry (Read Only) with Component IDs MM_Customers, MM_Contacts, MM_Employees, MM_Products, MM_Sales, MMQuarterlyReports, MM_GeneralInquiry accordingly.

In a secure work environment, suppose only one workstation or notebook can have access to all menus with MAC address 00-0D-43-2B-D6-B4 and everyone else can only access to General Inquiry (Read Only) menu.

If you would like to disable Customers, Contacts, Employees, Products, Sales and Quarterly Sales Report menus to all except the workstation or notebook with the MAC address 00-0D-43-2B-D6-B4, then here's an example to use as embedded Custom EEP in "On After Start EEP" option of form properties:

```
IF (GETVAL('GetMACAddr','1')) <> '00-0D-43-2B-D6-B4' THEN
    PROPERTY MM_Customers ENABLED 'FALSE'
    PROPERTY MM_Contacts ENABLED 'FALSE'
    PROPERTY MM_Employees ENABLED 'FALSE'
    PROPERTY MM_Products ENABLED 'FALSE'
    PROPERTY MM_Sales ENABLED 'FALSE'
    PROPERTY MM_QuarterlyReports ENABLED 'FALSE'
ENDIF
RETURN
```

If you would like to hide Customers, Contacts, Employees, Products, Sales and Quarterly Sales Report menus to all except the workstation or notebook with the MAC address 00-0D-43-2B-D6-B4, then here's an example to use as embedded Custom EEP in "On After Start EEP" option of form properties:

```
IF (GETVAL('GetMACAddr','1')) <> '00-0D-43-2B-D6-B4' THEN
    PROPERTY MM_Customers VISIBLE 'FALSE'
    PROPERTY MM_Contacts VISIBLE 'FALSE'
    PROPERTY MM_Employees VISIBLE 'FALSE'
    PROPERTY MM_Products VISIBLE 'FALSE'
    PROPERTY MM_Sales VISIBLE 'FALSE'
    PROPERTY MM_QuarterlyReports VISIBLE 'FALSE'
ENDIF
RETURN
```

1.8.4.7 GetVolumeID

(GETVAL('GetVolumeID','driveletter'))

Example:

```
SET VAR vVolumeID = (GETVAL('GetVolumeID','C'))
```

Will return the label name of drive C.

1.8.4.8 PlayAndExit

(GETVAL('PlayAndExit','filepath'))

Example:

```
SET VAR vPlay = (GETVAL('PlayAndExit','c:\windows\media\tada.wav'))
```

Will play sound/wave file and continue the next command in a command file or EEP.

1.8.4.9 PlayAndWait

(GETVAL('PlayAndWait','filepath'))

Example:

```
SET VAR vPlay = (GETVAL('PlayAndWait','c:\windows\media\tada.wav'))
```

Will play sound/wave file and waits for the sound to finish before continue to a next command. This could be a long time if the sound file is long. You would want this if, for instance, you're writing an answering machine aplication in R:BASE for Windows.

1.9 H

1.9.1 HZC

(HZC(arg))

Converts a single byte value to a double byte value. The function is specific to R:BASE use for a computer whose system locale is configured for Japan.

The function is based on having previous style Japanese characters. Internally R:BASE has a table of character codes that correspond to the Japanese locale character mappings. The mappings are what has been used in Japan for a long time and are called "half width characters". There is another table that has to same characters but stored a "full width characters".

As an example, the lower case "a" followed by a blank equals 61 20 as hex values. HZC maps these to 82 81 as hex values which is a "full width" lower case "a". The ZHC function can take 81 81 and map it back to 61 20.

1.9.2 HTML

(HTML(*string*))

Converts a text value to HTML code.

1.10 I

1.10.1 ICAP

(ICAP(*arg*))

Converts *arg* to a string with an initial capital letter on only the first word.

In the following example, the value of *vicap* is *Mary is going to Murrysville*.

```
SET VAR vicap = (ICAP('mary is going to Murrysville'))
```

1.10.2 ICAP1

(ICAP1(*arg*))

Converts *arg* to a string with an initial upper case letter on the first word, and lower case on an initial capital letter on any following words. This is also known as Sentence Casing.

In the following example, the value of *vicap1* is *Mary went down the street.*

```
SET VAR vicap1 = (ICAP1('mary went down The Street.'))
```

1.10.3 ICAP2

(ICAP2(*arg*))

Converts *arg* to a string with an initial capital letter on each word. This is also known as Word Case.

In the following example, the value of *vicap2* is *John Smith*.

```
SET VAR vicap2 = (ICAP2('john smith'))
```

In the following example, the value of *vHyfnTxt1* is *Daniel Day-Lewis*.

```
SET VAR vHyfnTxt1 TEXT = (ICAP2('daniel day-lewis'))
```

In the following example, the value of *vName2* is *Brian O'Rielly*.

```
SET VAR vName TEXT = ('BRIAN O''RIELLY')
SET VAR vName2 TEXT = (ICAP2(.vName))
```

In the following example, the value of *vText2* is *David Chadwick's Office Supplies*.

```
SET VAR vText TEXT = ('DAVID CHADWICK''S OFFICE SUPPLIES')
SET VAR vText2 TEXT = (ICAP2(.vText))
```

1.10.4 ICAP3

(ICAP3(*arg*))

Converts *arg* to a string with an initial upper case letter on the first word, and principal words. Non-capitalized letters/words include articles (a, an, the), conjunctions (e.g., and, but, or), and prepositions (e.g., on, in, with). This is also known as Title Casing.

In the following example, the value for vTitle is "Snow White and the Seven Dwarfs".

```
SET VAR vTitle TEXT = (ICAP3('SNOW WHITE AND THE SEVEN DWARFS'))
```

1.10.5 IFCASEEQ

(IFCASEEQ(*arg1,arg2,arg3,arg4*))

If the case and values for *arg1* and *arg2* are equal, IFCASEEQ returns the value of *arg3*. If the case and values for *arg1* and *arg2* are not equal, IFCASEEQ returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

The function enables easier conformance for password verification, as a mixture of upper and lower case is a standard.

In the following example, the value for vCheckPW1 is NO.

```
SET VAR vCheckPW1 TEXT = (IFCASEEQ('SuperStrong1147','SUPERSTRONG1147','Yes','No'))
```

In the following example, the value for vCheckPW2 is TRUE.

```
SET VAR vCheckPW2 TEXT = (IFCASEEQ('LastOne','LastOne','True','False'))
```

1.10.6 ICHAR

(ICHAR(*arg*))

Converts a single character, returning its corresponding ASCII integer value.

In the following example, the integer value of *vichar* is *65*.

```
SET VAR vichar = (ICHAR('A'))
```

1.10.7 IDAY

(IDAY(*arg*))

Where *arg* is a value that has either a DATE or DATETIME data type, IDAY returns the integer day of the month for a particular date.

In the following example, the value of *viday* is *12*.

```
SET VAR viday = (IDAY('06/12/93'))
```

1.10.8 IDIM

(IDIM(*arg*))

Where *arg* is a value that has a DATE data type, IDIM returns the number of days within that month.

In the following example, the value of *vidim* is *31*.

```
SET VAR vidim = (IDIM('03/15/2006'))
```

1.10.9 IDOY

(IDOY(*arg*))

Where *arg* is a value that has a DATE data type, IDOY returns the number day of the year.

In the following example, the value of *vidoy* is *74*.

```
SET VAR vidoy = (IDOY('03/15/2006'))
```

1.10.10 IDWK

(IDWK(*arg*))

Where *arg* is a value that has either a DATE or DATETIME data type, IDWK returns the day of the week where Monday is 1.

In the following example, the value of *vidwk* is *3*.

```
SET VAR vidwk = (IDWK('06/16/93'))
```

1.10.11 IFEQ

(IFEQ(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* and *arg2* are equal, IFEQ returns the value of *arg3*. If *arg1* and *arg2* are not equal, IFEQ returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

The following command changes the value of the *thiscol* column to *100* if the values of the variables *vaval* and *vbval* are equal. If *vaval* and *vbval* are not equal, the value of *thiscol* becomes *200* in rows where *thiscol* is greater than or equal to *100*.

```
UPDATE thistab SET thiscol=(IFEQ(.vaval, .vbval, 100, 200))+
WHERE thiscol >= 100
```

See IFCASEEQ for case-sensitive text comparisons.

1.10.12 IFEXISTS

(IFEXISTS(*arg1*,*arg2*,*arg3*))

If *arg1* contains a value, IFEXISTS returns the value of *arg2*. If *arg1* is null, then the value of *arg3* is returned.

The IFEXISTS function does not recognize the value of zero (0) as NULL, even if ZERO is set ON.

1.10.13 IFF

(IFF((*condition*),*arg1*,*arg2*))

If the condition is met, then the value of arg1 is returned. If condition is not met, then the value of arg2 is returned.

The condition must follow the syntax specifications as set with IF...ENDIF command structure.

The condition must list a set of conditions that combine to form a statement that is either true or false. Conditions can be combined with the connecting operators AND, OR, AND NOT, and OR NOT. It is important to note that the condition needs to be a single item, which is why quotes are used in the examples below.

The data types of arg1 and arg2 must match.

Examples:

```
-- Example 01:
- -Comparison of Variables
SET VAR v1 INTEGER = 1
SET VAR v2 INTEGER = 2
SET VAR v3 = (IFF('.v1<.v2','First','Second'))
SHOW VAR v3
```

```
First

-- Example 02:
-- SELECT and Display Feedback on Data

CONNECT RRBYW19
R>SELECT NetAmount,(IFF('NetAmount>$100000','Good','Needs Improvement'))=30 AS Status
FROM SalesBonus

 NetAmount       Status
 --------------- ------------------------------
     $176,000.00 Good
      $87,500.00 Needs Improvement
      $22,500.00 Needs Improvement
      $40,500.00 Needs Improvement
      $76,800.00 Needs Improvement
      $36,625.00 Needs Improvement
      $56,250.00 Needs Improvement
     $152,250.00 Good
     $108,750.00 Good
      $78,750.00 Needs Improvement
      $27,000.00 Needs Improvement
       $9,500.00 Needs Improvement
     $210,625.00 Good
```

1.10.14 IFGE

(IFGE(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* is greater than or equal to *arg2*, IFGE returns the value of *arg3*. If *arg1* is less than *arg2*, IFGE returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

In the following example, the value of *vifge* is *4*, because the date 01/10/2013 is greater than the date 10/15/2012.

```
SET VAR vifge = (IFGE('01/10/2013','10/15/2012',4,5))
```

1.10.15 IFGT

(IFGT(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* is greater than *arg2*, IFGT returns the value of *arg3*. If *arg1* is less than *arg2*, IFGT returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

In the following example, the value of *vifgt* is *4*, because the date 1/1/96 is greater than the date 1/1/95.

```
SET VAR vifgt = (IFGT('1/1/96','1/1/95',4,5))
```

1.10.16 IFLE

(IFLE(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* is less than or equal to *arg2*, IFLE returns the value of *arg3*. If *arg1* is greater than *arg2*, IFLE returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

In the following example, the value of *vifle* is *100*, since *A* is less than *B*.

```
SET VAR A = 37
SET VAR B = 48
SET VAR vifle = (IFLE(.A,.B,100,100))
```

1.10.17 IFLT

(IFLT(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* is less than *arg2*, IFLT returns the value of *arg3*. If *arg1* is greater than *arg2*, IFLT returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

In the following example, the value of *viflt* is *4*, since *A* is less than *B*.

```
SET VAR A = 37
SET VAR B = 48
SET VAR viflt = (IFLT(.A,.B,4,5))
```

1.10.18 IFNE

(IFNE(*arg1*,*arg2*,*arg3*,*arg4*))

If *arg1* and *arg2* are not equal, IFNE returns the value of *arg3*. If *arg1* and *arg2* are equal, IFNE returns the value of *arg4*. The data types of *arg1* and *arg2* must match and the data types of *arg3* and *arg4* must match.

In the following example, the value of *vifne* is *7*, since v1 is not equal to v2.

```
SET VAR v1 = 25
SET VAR v2 = 30
SET VAR vifne = (IFNE(.v1,.v2,7,8))
```

1.10.19 IFNULL

(IFNULL(*arg1,arg2,arg3*))

If *arg1* is null, then the value of *arg2* is returned. If *arg1* is not null, then the value of *arg3* is returned.

The IFNULL function does not recognize the value of zero (0) as NULL, even if ZERO is set ON.

1.10.20 IFRC

(IFRC(*arg*))

Where *arg* is a value that has either a TIME or DATETIME data type, IFRC returns the current thousandth of a second. For this function to work correctly, the TIME format must be set to include thousandths of a second.

In the following example, the value of *vifrc* is *123*.

```
SET TIME FOR HH:MM:SS.SSS
SET VAR vifrc = (IFRC('12:15:30.123'))
```

1.10.21 IFWINDOW

(IFWINDOW('*windowname*'))

Returns 1 if a form with the *windowname* is open, 0 if not. *Windowname* is the name given to the instance of an MDI form started with the "AS *alias*" option when using the ENTER, EDIT USING, or BROWSE USING commands.

1.10.22 IHASH

(IHASH(*arg,n*))

R:BASE includes a function that can be used to create an integer value from a text value. The function was designed to create effective integer keys from long text columns. The function, IHASH, converts the entire text value, or just a specified number of characters.

Using this method is more complex that just indexing the LASTNAME column. First you need to add a computed column to your table using the IHASH function on the LASTNAME column to convert its text to integer values. You can modify your table through the Data Designer or the ALTER TABLE command:

```
ALTER TABLE employee ADD Hash_Lname=(IHASH(lastname,0)) INTEGER
```

The IHASH function converts the entire name to integer when used with the parameter 0. A different parameter converts the specified number of characters from the name, starting at the first character. For example, the parameter 7 will convert the first 7 characters of the lastname to an integer value. Deciding on the number of characters to convert can be one of the hardest things about using this method. Consider the relationships expressed in the following chart:

	Convert FEW characters	**Convert MORE characters**
PROS	Less input required	Less duplicate values
CONS	Greater duplicate values	More input required

After adding the computed column to your table, you need to use some programming commands as shown below to query that column. Using the IHASH function directly in a WHERE clause won't use indexes. First set a variable equal to IHASH of the value you're searching for, then use the variable in the WHERE clause. When using an IHASH column for searching, you won't be able to do ad hoc queries from the R:BASE main menu .

```
SET VAR vname = (IHASH('Smith',0))
SELECT * FROM employee WHERE Hash_Lname = .vname
```

This method does provide greater flexibility in that you can have users enter anywhere from 1 character to the entire name based on the number of characters you specify in the IHASH function. For example, add a computed column to the table that will IHASH the first four characters of the name. Then, in your program, check the length that the user enters and if it's greater than four characters use an extra condition on your WHERE clause.

```
DIALOG 'Enter lastname (at least 4 characters):' vname vendkey 1
SET VAR vlen1=(SLEN(.vname)),vname1=(SGET(.VNAME,4,1)),+
  vhash=(IHASH(.vname1,4))
  IF vlen1 > 4 THEN
    CHOOSE vchoice FROM #VALUES FOR (firstname & lastname) +
      FROM employee WHERE Hash_Lname=.vhash AND +
      (SGET(lastname,.vlen1,1))=.vname
  ELSE
    CHOOSE vchoice FROM #VALUES FOR (firstname & lastname) +
      FROM employee WHERE Hash_Lname=.vhash
  ENDIF
```

A user can enter any number of letters for use with an IHASH computation, but must enter at least as many characters as specified in the IHASH column definition or enter the full name. If the entry less than the specified number of characters and less than the full length of the name, the correct data is not found. For example, with a column defined as (IHASH(lastname,7)), entering "WILL" will not find "WILLIAMS", it will only find "WILL".

The advantages of using a computed column with the IHASH function are that you can turn an inefficient TEXT index into an efficient INTEGER index and you can provide flexibility in searching. Users will have a larger selection of names to choose from and can select the appropriate person from the list. For example, entering WILLIAM will find WILLIAMSON, WILLIAM, and WILLIAMS if you use IHASH(lastname,7).

A disadvantage of this method is that you need to add columns to your database. An extra computed column can slow down data entry. If you are tight on disk space this may not be an option. To determine how much additional disk space you'll need for an IHASH column, take the number of rows in the table and multiply by 4. The answer is the number of bytes of disk space you'll need for the additional column.

1.10.23 IHR

(IHR(*arg*))

Returns the integer hour of time, where *arg* is a value that has a TIME or DATETIME data type.

In the following example, the value of *vihr* is *12*.

```
SET VAR vihr = (IHR(12:15:30))
```

1.10.24 IINFO

(IINFO(arg1,*arg2,arg3*))

The IINFO function is used to return information about tables, columns, or indexes by reading internal bitmask flags. The function requires the ID number for the table, column, or index. This ID number can be obtained from the system tables SYS_TABLES, SYS_COLUMNS, or SYS_INDEXES, respectively. IINFO returns 0 if FALSE, or the number in argument 3 if TRUE.

Syntax:

```
(IINFO(flagtype,id,bitmask))
```

Where:

flagtype specifies the type of information returned; table flags, column flags, column flags for server tables, index flags, row ID

id specifies the ID number from the system tables for the table ID, column ID, or index ID

bitmask species the flag in the system table

Remarks:

- The values for arg1, arg2, and arg3 must be non-null integers, even if a particular argument is not needed for that case.
- IINFO returns 0 if FALSE, or the bitmask number in parameter 3 if TRUE (flags 4-7).

Flags:

Info	Flag Type	ID	Bitmask	Description
Row ID	0	0	0	Returns the rowid of the current row. The rowid is the offset from the start of file 2 where that row is stored. The values for arg2 and arg3 are not used. Using a 0 for arg2 and arg3 is recommended.
Minimum Data Type Scale	1	integer	0	Returns the minimum scale for the data type. The value for arg2 must be an integer value. The value for arg3 is not used. Using a 0 for arg3 is recommended.
Maximum Data Type Scale	2	integer	0	Returns the maximum scale for the data type. The value for arg2 must be an integer value. The value for arg3 is not used. Using a 0 for arg3 is recommended.

Table Cascade Flag	3	table ID	0	Returns cascade flag for a table. The value for arg3 is not used. Using a 0 for arg3 is recommended.
Column Flags	4	column ID	1	Returns bitmask value if is an autonumber column
	4	column ID	2	Returns bitmask value if a comment exists for column
	4	column ID	4	Returns bitmask value if column has a default value
	4	column ID	8	Returns bitmask value if column is temporary
	4	column ID	16	Returns bitmask value if column has an index
	4	column ID	32	Returns bitmask value if contains a USER default
	4	column ID	64	Returns bitmask value if contains a Not NULL flag
	4	column ID	128	Returns bitmask value if is a primary key or unique key
Column Flags (Server Tables)	5	column ID	1	Returns bitmask value if column is an optimal row qualifier
	5	column ID	2	Returns bitmask value if server column is read only
	5	column ID	4	Returns bitmask value if server column is autonumbered
	5	column ID	8	Returns bitmask value if server column row version qualifier
Table Flags	6	table ID	1	Returns bitmask value if comment exists for table
	6	table ID	2	Returns bitmask value if table has a primary key
	6	table ID	4	Returns bitmask value if table has a foreign key
	6	table ID	8	Returns bitmask value if table has an autonumbered column
	6	table ID	16	Returns bitmask value if table has a default column
	6	table ID	32	Returns bitmask value if table is readonly (dBASE)
	6	table ID	64	Returns bitmask value if table is temporary
	6	table ID	128	Returns bitmask value if table has a referenced key
	6	table ID	256	Returns bitmask value if table has a Not NULL column
	6	table ID	512	Returns bitmask value if table has a unique key
	6	table ID	1024	Returns bitmask value if table has a column with a data type greater than 10
	6	table ID	2048	Returns bitmask value if table has a VARBIT/VARCHAR column
	6	table ID	4096	Returns bitmask value if a cascade flag updates and deletes through all primary keys and unique keys
	6	table ID	8192	Returns bitmask value if server table has column aliases
	6	table ID	16384	Returns bitmask value if there is at least one trigger defined for the table
	6	table ID	32768	Returns bitmask value if relation as system view which created during multiple inner joins
Index Flags	7	index ID	3	Returns bitmask value for the constraint type: 0 = index, 1 = foreign key, 2 = primary key, 3 = unique key
	7	index ID	4	Returns bitmask value if this is a dBase index
	7	index ID	8	Returns bitmask value if this is a unique index
	7	index ID	16	Returns bitmask value if index is temporary
	7	index ID	32	Returns bitmask value if this is a referenced key
	7	index ID	64	Returns bitmask value if this is a case sensitive index
	7	index ID	128	Returns bitmask value if this is a Foreign Index

Examples:

```
-- Example 01:
-- Using flag type 0 for the Titles table in the RRBYW20 database
SELECT EmpTID,EmpTitle,(IINFO(0,0,0)) FROM Titles

 EmpTID     EmpTitle                      (IINFO(0,0
 ---------- ----------------------------- ----------
          1 Office Manager                    524289
          2 Receptionist                      524341
          3 Sales Clerk                       524393
          4 Director Marketing                524445
          5 Director Corporate Sales          524497
          6 Director Government Sales         524549
          7 Manager Support & Services        524601
          8 Outside Sales                     524653


-- Example 02:
-- Returns minimum and maximum data type scales using IINFO and (CVAL('ROWCOUNT'))
with the RRBYW20 database.
-- The values for currency will vary based upon the current CURRENCY DIGITS setting.
SELECT (CVAL('ROWCOUNT')) AS SYS_TYPE, SYS_TYPE_NAME=18, +
(IINFO(1, (INT(CVAL('ROWCOUNT'))), 0)) AS SYS_MIN, +
(IINFO(2, (INT(CVAL('ROWCOUNT'))), 0)) AS SYS_MAX +
FROM SYS_TYPES
```

Here is what it generates:

```
 SYS_TYPE        SYS_TYPE_NAME      SYS_MIN    SYS_MAX
 --------------- ------------------ ---------- ----------
 1               CURRENCY                    2          2
 2               VARBIT             -0-        -0-
 3               BITNOTE            -0-        -0-
 4               BIT                -0-        -0-
 5               VARCHAR            -0-        -0-
 6               BIGNUM             -0-        -0-
 7               BSTR               -0-        -0-
 8               GUID               -0-        -0-
 9               TEXT               -0-        -0-
 10              NUMERIC                     0         15
 11              INTEGER                     0          0
 12              REAL               -0-        -0-
 13              DOUBLE             -0-        -0-
 14              DATE               -0-        -0-
 15              TIME                        0          3
 16              DATETIME                    0          3
 17              NOTE               -0-        -0-
```

```
-- Example 03:
-- Displays tables with Cascade within the RRBYW20 database.
-- Note that the tables with 1 for the cascade value are tables
-- with primary keys that cascade to tables with foreign keys.
SELECT SYS_TABLE_NAME=20, +
SYS_TABLE_ID, +
(IINFO(3,SYS_TABLE_ID,0)) AS SYS_CASCADE +
FROM SYS_TABLES WHERE SYS_TABLE_TYPE = 'TABLE'
```

```
 SYS_TABLE_NAME       SYS_TABLE_ SYS_CASCAD
 -------------------- ---------- ----------
 Customer                     29          1
 CompUsed                     30          0
 SalesBonus                   31          0
 PaymentTerms                 32          0
 Contact                      33          0
 ProdLocation                 34          0
 Levels                       35          0
 Component                    36          0
 Product                      37          0
 SecurityTable                38          0
 InvoiceHeader                39          0
 PrintOptions                 40          0
 InvoiceDetail                41          0
 LicenseInformation           43          0
 Titles                       44          1
 Employee                     45          1
 StateAbr                     46          0
 FormTable                    47          0
 BonusRate                    48          0
 TestNote                     49          0
 ContactCallNotes             51          0
 tempemployee                 74          0


-- Example 04:
-- Checks that the CustState colun in the Customer table is indexed
SET VAR vCustStateHasIndex = (IINFO(4,191,16))
SHOW VAR vCustStateHasIndex
         16

-- Example 05:
-- Checks that the Employee table has a primary key
SET VAR vEmployeeHasPK = (IINFO(6,45,2))
SHOW VAR vEmployeeHasPK
          2

-- Example 06:
-- Checks that the Component table has a referenced key.
SET VAR vComponentRef = (IINFO(6,39,128))
SHOW VAR vComponentRef
        128

-- Example 07:
-- Returns the constraint type for the EmpID in the SalesBonus table
-- (0 = index, 1 = foreign key, 2 = primary key, 3 = unique key)
SET VAR vIsForeignKey = (IINFO(7,42,3))
SHOW VAR vIsForeignKey
          1
```

1.10.25 ILY

(ILY(*arg*))

Checks to see if the current year is a leap year.

The function returns a 1 if the year is a leap year, and 0 if it is not a leap year.

In the following example, the value of *vily* is *0*.

```
SET VAR vily = (ILY('03/15/2006'))
```

1.10.26 IMIN

(IMIN(*arg*))

Returns the integer minutes of time, where *arg* is a value that has a TIME or DATETIME data type.

In the following example, the value of *vmin* is *15*.

```
SET VAR vmin TO (IMIN(12:15:30))
```

1.10.27 IMON

(IMON(*arg*))

Returns the integer month for a particular date, where *arg* is a value that has a DATE or DATETIME data type.

In the following example, the value of *vimon* is *5*.

```
SET VAR vimon = (IMON('05/20/95'))
```

1.10.28 INT

(INT(*arg*))

Truncates a number that has a REAL, DOUBLE, NUMERIC or CURRENCY data type, returning a value that has an INTEGER data type.

The function can also be used to convert numeric TEXT and NOTE variables to the INTEGER data type. In using TEXT/NOTE, the INT function will automatically round and truncate the value.

In the following example, the value of *vint* is *1*.

```
SET VAR vint = (INT(1.6))
```

In the following example, the value of *vint2* is 7.

```
SET VAR vtext TEXT = '6.8'
SET VAR vint2 = (INT(.vtext))
```

To truncate a text, the following example can be used, where the value of *vTruncInt* is 1.

```
SET VAR MyTextStringVar TEXT = '1.5'
SET VAR vTruncInt integer = (INT(FLOAT(.MyTextStringVar)))
```

1.10.29 ISALPHA

(ISALPHA(*value*))

Checks the first character of a TEXT string. The function returns a 1 if true and a 0 if false. For example, (ISALPHA('abc')) returns 1 because the first character is a letter.

1.10.30 ISDIGIT

(ISDIGIT(*value*))

Checks the first character of a TEXT string. The function returns a 1 if true and a 0 if false. For example, (ISDIGIT('abc')) returns 0 because the first character is not a number.

1.10.31 ISEC

(ISEC(*arg*))

Returns the integer seconds of time, where *arg* is a value that has a TIME or DATETIME data type.

In the following example, the value of *visec* is *30*.

```
SET VAR visec = (ISEC(12:15:30))
```

1.10.32 ISLOWER

(ISLOWER(*value*))

Checks the first character of a TEXT string. The function returns a 1 if true and a 0 if false. For example, (ISLOWER('abc')) would return 1 because the first character is lower case.

1.10.33 ISSPACE

(ISSPACE(*value*))

Checks the first character of a TEXT string. The function returns a 1 if true and a 0 if false. The ISSPACE function evaluates as true when the first character is a space or one of the following: LF (char (10)), CR (char (13)), VT (char (11)), TAB (char (9)), FF (char (12)), EOL (char (254)).

1.10.34 ISTAT

(ISTAT('keyword'))

Using ISTAT parameters, you can determine database activity, and the current mouse and cursor column or row coordinates.

Use ISTAT to check the efficiency of settings to adjust locking scheme. You should be able to see differences in the results returned by the ISTAT keywords TOTALREADS, TOTALWRITES, and TOTALLOCKS depending on the locking scheme you have set.

The ISTAT function has a number of options that report on available memory in the dynamic data area R:BASE uses for processing.

You must either enclose the keyword in quotation marks or use a dot variable that has a TEXT data type.

Following keywords can be used for (ISTAT('keyword')):

- CURRNUMALLOC
- CURSORCOL
- CURSORROW
- DBSIZE
- DISKSPACE
- FORM_CONTROL_TYPE
- FORM_DIRTY_FLAG
- ISRUNTIME
- LIMITNUMALLOC
- MAXFREE
- MAXNUMALLOC
- MEMORY
- MOUSECOL

- MOUSEROW
- PAGECOL
- PAGEROW
- RX1SIZE
- RX2SIZE
- RX3SIZE
- RX4SIZE
- TOTALALLOC
- TOTALFREE
- TOTALLOCKS
- TOTALREADS
- TOTALWRITES

1.10.34.1 CURRNUMALLOC

(ISTAT('CURRNUMALLOC'))

The CURRNUMALLOC parameter allows you to check the number of memory handles R:BASE for DOS has open. This returns 0 when used with R:BASE for Windows.

1.10.34.2 CURSORCOL

(ISTAT('CURSORCOL'))

(ISTAT('CursorCol')) will return an integer value indicating the current column that contains the cursor.

1.10.34.3 CURSORROW

(ISTAT('CURSORROW'))

(ISTAT('CursorRow')) will return an integer value indicating the current row that contains the cursor.

1.10.34.4 DBSIZE

(ISTAT('DBSIZE'))

DBSIZE returns the size of the currently open database (total of the four database files). If no database is connected, ISTAT returns 0.

If you wish to check for a database size that is over 2 gigabytes you must use a data type which supports a larger number.

```
SET VAR vDatabaseSize DOUBLE = (ISTAT('DBSIZE'))
```

This is necessary because the default data type returned by the function, INTEGER, cannot hold the required number of digits to report over 2 gigabytes.

1.10.34.5 DISKSPACE

(ISTAT('DISKSPACE'))

DISKSPACE returns the amount of free disk space on the current drive. If you wish to check for the free space on drives with over 2 gigabytes of free you must use syntax similar to this:

```
SET VAR vSpace DOUBLE = (ISTAT('DISKSPACE'))
```

This is necessary because the default data type returned by the function, INTEGER, cannot hold the required number of digits to report over 2 gigabytes of free space.

When depending on this routine it is best to test it on a platform as close to your target platform as possible.

1.10.34.6 FORM_CONTROL_TYPE

(ISTAT('FORM_CONTROL_TYPE'))

FORM_CONTROL_TYPE returns a value based on which object has focus when the entry/exit procedure containing the ISTAT function is run. Returns the following.

0	No current control
1	Edit Field
2	Combo Box
3	Check Box
5	Radio Button
7	Push Button

1.10.34.7 FORM_DIRTY_FLAG

(ISTAT('FORM_DIRTY_FLAG'))

FORM_DIRTY_FLAG Returns 1 if a change has been made to the data, 0 if no change has been made.

1.10.34.8 ISRUNTIME

(ISTAT('ISRUNTIME'))

Use the ISRUNTIME parameter to determine if the end user is accessing R:BASE via a full version or a Runtime version.

If the end user is using Runtime the value returned will be 1. If the end user is using a Full Version then the value returned will be 0.

1.10.34.9 LIMITNUMALLOC

(ISTAT('LIMITNUMALLOC'))

The LIMITNUMALLOC parameter allows you to check the maximum number of memory handles R:BASE for DOS can open. This returns 0 when used with R:BASE for Windows.

Unless you use the -h startup option this will be 300.

1.10.34.10 MAXFREE

(ISTAT('MAXFREE'))

The MAXFREE parameter allows you to determine the largest free memory block within the area allocated to R:BASE. The TOTALALLOC parameter can be used to determine the total amount of dynamic memory allocated to R:BASE. MAXFREE will always be smaller than TOTALALLOC.

This will not return valid information under R:BASE for Windows.

1.10.34.11 MAXNUMALLOC

(ISTAT('MAXNUMALLOC'))

The MAXNUMALLOC parameter allows you to check the highest number of memory handles R:BASE for DOS has held open at any one point during this session. This returns 0 when used with R:BASE for Windows.

1.10.34.12 MEMORY

(ISTAT('MEMORY'))

The MEMORY parameter allows you to retrieve the amount of memory, in bytes, remaining for R:BASE for DOS to use.

This will not return a valid amount when running R:BASE for Windows.

1.10.34.13 MOUSECOL

(ISTAT('MOUSECOL'))

MOUSECOL returns the column where the mouse pointer is.

This is only supported in DOS, and will not return a valid amount when running R:BASE for Windows.

1.10.34.14 MOUSEROW

(ISTAT('MOUSEROW'))

MOUSEROW returns the row where the mouse pointer is.

This is only supported in DOS, and will not return a valid amount when running R:BASE for Windows.

1.10.34.15 PAGECOL

(ISTAT('PAGECOL'))

PAGECOL returns the column location of the cursor on a virtual page when you are using the SET PAGEMODE command. PAGECOL can only be used with the SHOW VARIABLE command; it does not work with the WRITE command.

1.10.34.16 PAGEROW

(ISTAT('PAGEROW'))

PAGEROW returns the row location of the cursor on a virtual page when you are using the SET PAGEMODE command. PAGEROW can only be used with the SHOW VARIABLE command; it does not work with the WRITE command.

1.10.34.17 RX1SIZE

(ISTAT('RX1SIZE'))

The RX1SIZE parameter returns the size, in bytes, of the RX1 file of the currently connected database. You may need to use the following syntax for large databases:

```
SET VAR vSize DOUBLE = (ISTAT('RX1SIZE'))
```

This is simply because the default data type, INTEGER, may not be able to hold values as large as you need.

For a combined size use the DBSIZE parameter.

1.10.34.18 RX2SIZE

(ISTAT('RX2SIZE'))

The RX2SIZE parameter returns the size, in bytes, of the RX2 file of the currently connected database. You may need to use the following syntax for large databases:

```
SET VAR vSize DOUBLE = (ISTAT('RX2SIZE'))
```

This is simply because the default data type, INTEGER, may not be able to hold values as large as you need.

For a combined size use the DBSIZE parameter.

1.10.34.19 RX3SIZE

(ISTAT('RX3SIZE'))

The RX3SIZE parameter returns the size, in bytes, of the RX3 file of the currently connected database. You may need to use the following syntax for large databases:

```
SET VAR vSize DOUBLE = (ISTAT('RX3SIZE'))
```

This is simply because the default data type, INTEGER, may not be able to hold values as large as you need.

For a combined size use the DBSIZE parameter.

1.10.34.20 RX4SIZE

(ISTAT('RX4SIZE'))

The RX4SIZE parameter returns the size, in bytes, of the RX4 file of the currently connected database. You may need to use the following syntax for large databases:

```
SET VAR vSize DOUBLE = (ISTAT('RX4SIZE'))
```

This is simply because the default data type, INTEGER, may not be able to hold values as large as you need.

For a combined size use the DBSIZE parameter.

1.10.34.21 TOTALALLOC

(ISTAT('TOTALALLOC'))

The TOTALALLOC parameter allows you to determine the amount of dynamic memory allocated to R:BASE. Unless you ZIP out to another program this number should not change during an R:BASE session.

This will not return valid information under R:BASE for Windows.

1.10.34.22 TOTALFREE

(ISTAT('TOTALFREE'))

The TOTALFREE parameter allows you to determine the total amount of free memory within the area allocated to R:BASE. The TOTALALLOC parameter can be used to determine the total amount of dynamic memory allocated to R:BASE. TOTALFREE will always be smaller than TOTALALLOC.

This will not return valid information under R:BASE for Windows.

1.10.34.23 TOTALLOCKS

(ISTAT('TOTALLOCKS'))

TOTALLOCKS is used to determine the total number of locks placed on any assortment of tables, rows, and indexes.

1.10.34.24 TOTALREADS

(ISTAT('TOTALREADS'))

TOTALREADS returns the total number of disk reads since R:BASE began or the number of reads since the statement was last executed.

1.10.34.25 TOTALWRITES

(ISTAT('TOTALWRITES'))

TOTALWRITES returns the total disk writes since opening or since the last time the command was executed.

1.10.35 ISTR

(ISTR('*string*',*position*))

Returns the corresponding integer value.

This function converts a single character, which you specify within a string by position, returning its corresponding ASCII Character Chart Decimal value.

In the following example, the INTEGER value of *vDecimalValue* is 65 for the capital letter A.

Example 01:

Start R:BASE for Windows

At the R> Prompt:

```
SET VARIABLE vDecimalValue = (ISTR('R:BASE Rocks!',4))

SHOW VARIABLE

vDecimalValue = 65 INTEGER
```

1.10.36 ISUPPER

(ISUPPER(*value*))

Checks the first character of a TEXT string. The function returns a 1 if true and a 0 if false. For example, (ISUPPER('abc')) returns 0 because the first character is not upper case.

1.10.37 ITEMCNT

(ITEMCNT('*Text String*'))

Use to count the number of items in a text string separated by a comma (or the current delimiter).

In the following example, the value of *vItems* is *3*.

```
SET VAR vItems = (ITEMCNT('a,b,c'))
```

Here is an example of using this function in a command block to format a CHOOSE box:

```
SET VAR vModels TEXT = NULL
SET VAR vLines INTEGER = NULL
SET VAR vModel TEXT = NULL
SELECT (LISTOF(Model)) INTO vModels INDIC IModel FROM Product
SET VAR vLines = (ITEMCNT(.vModels))
IF vLines > 18 THEN
SET LINES = 18
ENDIF
CLS
CHOOSE vModel FROM #LIST .vModels AT 4 30 TITLE 'Choose Model' +
CAPTION 'Available Models' Lines .vLines FORMATTED
IF vModel IS NULL OR vModel = '[Esc]' THEN
GOTO Done
ENDIF

-- Do what you have to do here ...

LABEL Done
CLEAR ALL VAR
QUIT TO MainMenu.RMD
RETURN
```

1.10.38 IWOY

(IWOY(*arg1*)) or (IWOY(*arg1,arg2*))

Returns the week number of the year for the specified date. The use of *arg2* is optional, to indicate different week counting methods.

Syntax:

```
(IWOY(date,parameter))
```

Where:

date is a valid DATE data type value

parameter specifies week counting method

Parameters:

Parameter	Description
1	Specifies that weeks start on Sunday, and January 1 is in the first week of the year
2	Specifies that weeks start on Monday, and January 1 is in the first week of the year
21	Specifies that weeks follow the ISO 8601 standard, which means that week 1 for the year is the first week with 4 or more days of January in it. In addition, the week starts on Monday.

Remarks:

- The first parameter must be a valid DATE value.
- The IWOY function with no second parameter is identical to using the 1 parameter.

Examples:

01.
In the following example, the value of *viwoy_1* is *12*.

```
SET VAR viwoy_1 = (IWOY('03/15/2016'))
```

02.
In the following example, the value of *viwoy_2* is *11*.

```
SET VAR viwoy_2 = (IWOY('03/15/2016',21))
```

03.
In the following example, the value of *viwoy_3* is 39.

```
SET VAR vDate DATE = 09/19/2016
SET VAR viwoy_3 = (IWOY(.vDate,2))
```

04.
In the following example, the value of *viwoy_4* is 38.

```
SET VAR vDate DATE = 09/19/2016
SET VAR viwoy_4 = (IWOY(.vDate,21))
```

1.10.39 IYR

(IYR(*arg*))

Where *arg* is a value that has a DATE or DATETIME data type, IYR returns a two or four-digit integer year of date, depending on how the DATE format is set.

In the following example, the value of *viyr* is *1995* if the year in the DATE format is set to YYYY. The value of *viyr* is *95* if the year in the DATE format is set to YY.

```
SET VAR viyr = (IYR('09/13/95'))
```

1.10.40 IYR4

(IYR4(date)) or (IYR4(datetime))

Where *arg* is a value that has a DATE or DATETIME data type, IYR4 always returns a four-digit integer value for the year. This is different from the IYR function which depends on how the DATE format is set.

This function will allow users to capture four digit year, regardless of the DATE FORMAT or DATE SEQUENCE settings. Results will be based on the DATE YEAR and CENTURY settings. See DATE format.

Example 01 (Database with settings):

```
DATE FORMAT: MM/DD/YYYY
DATE SEQ: MMDDYY
DATE YEAR: 30
DATE CENT: 19
SET VAR v1 = (IYR4(.#DATE))
```

Variable v1 will return the four digit INTEGER value of 2000

Example 02:

```
SET VAR v2 DATETIME = ('06/26/2000 08:00')
SET VAR v3 = (IYR4(.v2))
```

Variable v3 will return the four digit INTEGER value of 2000

Example 03 (Database with settings):

```
DATE FORMAT: MM/DD/YY
```

```
DATE SEQ: MMDDYY
DATE YEAR: 30
DATE CENT: 19
SET VAR v1 = (IYR4(.#DATE))
```

Variable v1 will ALSO return the four digit INTEGER value of 2000

Example 04:

```
SET VAR v2 DATETIME = ('06/26/00 08:00')
SET VAR v3 = (IYR4(.v2))
```

Variable v3 will ALSO return the four digit INTEGER value of 2000

1.11 J

1.11.1 JDATE

(JDATE(*arg*))

Where *arg* is a value that has a DATE or DATETIME data type, JDATE returns the Julian date of the date in the form YYYYDDD. This is a change from versions prior to R:BASE 6.1a which returned a value in the format YYDDD. This two digit year format was incompatible with the year 2000 and may require altering database structure or program code.

In the following example, the value of *vjdate* is *1995163*. The year is *1995*, and *163* means that the date is the 163rd day of 1995.

```
SET VAR vjdate = (JDATE('06/12/95'))
```

1.11.2 JDWK

(JDWK(arg))

Returns the Japanese day of the week, where arg is either a DATE or DATETIME data type value. The function is specific to R:BASE use for a computer whose system locale is configured for Japan.

1.11.3 JIDX

(JIDX('string',position))

Returns the corresponding ASCII character chart integer value for a character, specified within a string by position. When Japanese characters are the default, 0 or -1 is returned for Kanji characters. The function is specific to R:BASE use for a computer whose system locale is configured for Japan.

1.12 L

1.12.1 LASTKEY

(LASTKEY(*arg*))

Where *arg* is *0* or *1*. When *arg* is *1*, LASTKEY returns the original mapping for a key that has been remapped (the key that was pressed). When *arg* is *0*, LASTKEY returns the current mapping (the key that was executed).

In the following example, DIALOG displays the message and the current date, which the user can edit. LASTKEY captures the last key pressed by the user, enabling the user to press [Esc] to avoid entering a date.

```
SET VARIABLE vdate TEXT
SET VARIABLE vdate = (CTXT(.#DATE))
DIALOG 'Enter the invoice date to print:' vdate vendkey 1
SET VARIABLE vlast = (LASTKEY(0))
IF vlast = '[Esc]' THEN
   GOTO skipprnt
ENDIF
```

1.12.2 LASTMOD

(LASTMOD('tablename'))

Returns the last structural modification date time value for a table or view.

In the following example, the date and time is provided for when the structure of the *Tasks* table was last modified.

```
SET VAR vTaskTableModDate DATETIME = (LASTMOD('Tasks'))

R>SHOW VAR vTaskTableModDate
01/06/2019 10:50 AM
```

1.12.3 LAVG

(LAVG(*list*))

Returns the average of the values in *list*. Values in a list can be a CURRENCY, DOUBLE, REAL, INTEGER, NUMERIC, DATE, or TIME data type. The function LAVG is not the same as the function AVG, which is used only with the SELECT command.

In the following example, the value of *vlavg* is *5*. The total of the list is 20, which is divided by 4, the number of values in the list.

```
SET VAR vlavg = (LAVG(2,4,6,8))
```

1.12.4 LJS

(LJS(*text*,*width*))

Left justifies *text* in *width* characters, returning a text string.

In the following example, the value of *vljs* is *ABCD*. The text string is left justified in the field. In this case, trailing blanks are removed.

```
SET VAR vljs = (LJS('ABCD',10))
```

The value of *vljs2* in the following example is *COLUMN ONE COLUMN TWO*. You can use LJS to embed spaces and concatenate strings, for example, to create a row of column headings for a screen or variable form. When you concatenate strings before assigning the result to a variable, as in this example, trailing blanks are retained. These spaces are preserved even if the first value is null.

```
SET VAR vljs2 = (LJS('COLUMN ONE',20) + 'COLUMN TWO')
```

The following example returns *COLUMN TWO* if *vcolname* is null.

```
SET VAR vcol2 TEXT = 'COLUMN TWO'
SET VAR vljs3 TEXT = (LJS(.vcolname,20) + .vcol2))
```

1.12.5 LMAX

(LMAX(*list*))

Returns the maximum value in *list*. Values in a list can be a CURRENCY, DOUBLE, REAL, INTEGER, NUMERIC, DATE, or TIME data type. The function LMAX is not the same as the function MAX, which is used only with the SELECT command.

In the following example, the value of *vlmax* is *80*, the highest number in the list.

```
SET VAR vlmax = (LMAX(2,80,14,22))
```

1.12.6 LMIN

(LMIN(*list*))

Returns the minimum value in *list*. Values in a list can be a CURRENCY, DOUBLE, REAL, INTEGER, NUMERIC, DATE, or TIME data type. The function LMIN is not the same as the function MIN, which is used only with the SELECT command.

In the following example, the value of *vlmin* is *2*, the lowest number in the list.

```
SET VAR vlmin = (LMIN(2,80,14,22))
```

1.12.7 LOG

(LOG(*arg*))

Returns log base *e* of *arg* (where *e* = 2.71828182845905). *Arg* must be a positive value and have a DOUBLE, REAL, NUMERIC, or INTEGER data type. LOG performs the inverse operation of EXP.

In the following example, the value of *vlog* is *0.6931*.

```
SET VAR vlog = (LOG(2))
```

1.12.8 LOG10

(LOG10(*arg*))

Returns log base 10 of *arg*. *Arg* must be a positive value and have a DOUBLE, REAL, NUMERIC, or INTEGER data type.

In the following example, the value of *vlog2* is *2*.

```
SET VAR vlog2 = (LOG10(100))
```

1.12.9 LSTDEV

(LSTDEV(*list*))

Returns the standard deviation for a list of values.

Remarks:

- The standard deviation is a measure of how widely values are dispersed from the average value.
- LSTDEV supports CURRENCY, DECIMAL, DOUBLE, FLOAT, REAL, NUMERIC, or INTEGER data type.
- The LSTDEV performs calculations in the same fashion as the COMPUTE command.

Example:

```
SET VAR vListStDev DOUBLE =
(LSTDEV(173.20,189.45,3929.14,434.75,333.25,257.50,88.70,641.86))
SHOW VAR vListStDev
```

```
1293.84205284038
```

1.12.10 LSUM

(LSUM(*list*))

Return the sum for a list of values.

Remarks:

- LSUM supports CURRENCY, DOUBLE, REAL, NUMERIC, INTEGER, BIGINT, SMALLINT, or BIGNUM data types.

Example:

```
SET VAR vListSum DOUBLE = (LSUM(189.45,79.14,43.75,33.25,27.58,789.40,6.12))
SHOW VAR vListSum
1168.69
```

1.12.11 LTRIM

(LTRIM(*text*))

Trims leading blanks from *text*, returning a text string.

In the following example, the value of *vltrim* is the text string *ABCDE* without the leading blanks.

```
SET VAR vltrim = (LTRIM('   ABCDE'))
```

1.12.12 LUC

(LUC(*arg*))

Converts *arg* from lowercase to uppercase, returning a text string.

In the following example, the value of *vluc* is an uppercase *A*.

```
SET VAR vluc = (LUC('a'))
```

1.12.13 LVARIANCE

(LVARIANCE(*list*))

Return the variance for a list of values.

Remarks:

- LVARIANCE supports CURRENCY, DECIMAL, DOUBLE, FLOAT, REAL, NUMERIC, or INTEGER data type.
- The LVARIANCE performs calculations in the same fashion as the COMPUTE command.

Example:

```
SET VAR vListVariance DOUBLE =
(LVARIANCE(528.00,175.00,36.63,112.50,456.75,326.25,157.50,27.00,631.88))
SHOW VAR vListVariance
49438.425925
```

1.13 M

1.13.1 MAKEUTF8

(MAKEUTF8(*text*))

Converts upper ASCII characters text data into UTF-8 encoded characters.

Example:

```
R>SHOW VAR vText
Franz Wei? has met Anna G?rtner

R>SET VAR vUTF8Text = (MAKEUTF8(.vText))

R>SHOW VAR v%
Variable           = Value                                  Type
------------------   ------------------------------         -------
vText              = Franz Wei? has met Anna                TEXT
                     G?rtner
vUTF8Text          = Franz Weiß has met Anna                TEXT
                     Gärtner
```

1.13.2 MOD

(MOD(*arg1*,*arg2*))

Computes a modulus or remainder of *arg1* divided by *arg2*. *Arg1* and *arg2* must be values that have DOUBLE, REAL, NUMERIC, or INTEGER data types and *arg2* cannot be 0.

In the following example, the value of *vmod* is *1*, the remainder when 3 is divided by 2.

```
SET VAR vmod = (MOD(3,2))
```

In the following example, the value of vCheck4EvenOdd is checked if it is even or odd.

```
SET VAR vNumber2Check INTEGER = 3
SET VAR vCheck4EvenOdd INTEGER = (IFEQ((MOD(.vNumber2Check,2)),0,2,1))
IF vCheck4EvenOdd = 2 THEN
   PAUSE 2 USING 'The number is even.'
ELSE
   PAUSE 2 USING 'The number is odd.'
ENDIF
```

1.14 N

1.14.1 NEXT

(NEXT(tblname,autonumcol))

Returns the next value of an autonumbered column.

Where colname is an autonumbered column in tblname NEXT returns, and increments, the value of the next available autonumber. You cannot use this function with INSERT, but you can use it with LOAD;NONUM. For example: Assume that you have autonumbered the column EmployeeID in the table Employees. The highest number currently used in the database is 134. In this case, in the following example, the value of vNextOne will be 135 and the value of vNextTwo will be 136.

Notice that the value has incremented even though no other commands or functions were issued.

```
SET VAR vNextOne = (NEXT(Employees,EmployeeID))
SET VAR vNextTwo = (NEXT(Employees,EmployeeID))
```

1.14.2 NINT

(NINT(*arg*))

Rounds a number that has a TEXT, REAL, DOUBLE, NUMERIC, or CURRENCY data type to the nearest integer, returning a value that has an INTEGER data type.

In the following example, the value of *vnint1* is *3* and the value of *vnint2* is *4*.

```
SET VAR vnint1 = (NINT(2.6))
SET VAR vnint2 = (NINT(4.4))
```

1.15 P

1.15.1 PMT1

(PMT1(*int*,*per*,*pv*))

Returns the amount of the periodic payment needed to pay off the present value, *pv*, based on the periodic interest rate, *int*, for the number of compounding periods, *per*.

In the following example, PMT1 returns the monthly payment amount for a loan of $12,000 with a 12% annual interest rate and five years to pay it off. The value of *vpmt1* (the monthly payment) is *$266.93*.

```
SET VAR vpmt1 = (PMT1(.01,60,12000))
```

1.15.2 PMT2

(PMT2(*int*,*per*,*fv*))

Returns the amount of the periodic payment to accrue the future value, *fv*, based on the periodic interest rate, *int*, for the number of compounding periods, *per*.

In the following example, PMT2 returns the monthly payment you must make if you want to have $25,000 in 10 years and your annual interest rate is 7%, compounded monthly. The value of *vpmt2* (the payment) is *$144.44*.

```
SET VAR vpmt2 = (PMT2((.07/12),120,25000))
```

1.15.3 PV1

(PV1(*pmt*,*int*,*per*))

Returns the present value of a series of equal payments of the amount, *pmt*, periodic interest rate, *int*, and compounding periods, *per*.

In the following example, PV1 returns the present value of an annuity with an annual interest rate of 9% and for which you want to pay $500 each month for 20 years. The value of *vpv1* (the present value) is *$55,572.48*.

```
SET VAR vpv1 = (PV1(500,.0075,240))
```

1.15.4 PV2

(PV2(*fv*,*int*,*per*))

Returns the present value based on the future value, *fv*, interest rate, *int*, and the number of compounding periods, *per*.

In the following example, PV2 returns the amount you must invest for a period of one year to return a future value of $3,500 where the annual rate is 7.6% (compounded daily). The value of *vpv2* (the amount to invest) is *$3,247.63*.

```
SET VAR vpv2 = (PV2(3500,(.075/365),365))
```

1.16 R

1.16.1 RANDOM

(RANDOM(*value*))

Generates a random number between 0 and one value below the value entered.

1.16.2 RANKAVG

(RANKAVG(*arg,column,table/view,order*))

Returns the statistical rank of a given value, within a supplied array of values. If there are duplicate values in the list, the average rank is returned.

finds the rank for the first occurrence of the value

arg - specifies the item to rank. The value must be TEXT. The CTXT may be used, if needed.
column - specifies the column to compare *arg* to. The value must be TEXT.
table/view - specifies the table/view which contains the column The value must be TEXT.
order - 0 for ranking from highest to lowest (like for scores)
- 1 for ranking from lowest to highest (like for race times)

The resulting variable value is the DOUBLE data type.

After the RANKAVG function is used, the ProcRANKAVG stored procedure will be loaded into the database, if the procedure does not already exist.

The below calculates the average rank of 1534 as 5 in a list of test scores.

```
SET VAR vRankAvg DOUBLE = (RANKAVG('1534','Score','TestScores',1))
SHOW VAR vRankAvg
         5.
```

1.16.3 RANKEQ

(RANKEQ(*arg,column,table/view,order*))

Returns the statistical rank of a given value, within a supplied array of values. If there are duplicate values in the list, these are given the same rank.

arg - specifies the item to rank. The value must be TEXT. The CTXT may be used, if needed.
column - specifies the column to compare *arg* to. The value must be TEXT.
table/view - specifies the table/view which contains the column The value must be TEXT.
order - 0 for ranking from highest to lowest (like for scores)
- 1 for ranking from lowest to highest (like for race times)

The resulting variable value is the INTEGER data type.

After the RANKEQ function is used, the ProcRANKEQ stored procedure will be loaded into the database, if the procedure does not already exist.

The below calculates the rank of 80,000 as 23 in a list of car sales.

```
SET VAR vRankEQ DOUBLE = (RANKEQ('80,000','InvoicePrice','CarSales',1))
SHOW VAR vRankEQ
         23.
```

1.16.4 RATE1

(RATE1(*fv*,*pv*,*per*))

Returns the periodic interest rate required to return the future value, *fv*, based on the present value, *pv*, over the number of compounding periods, *per*.

In the following example, RATE1 returns the interest rate you must have if your initial investment is $8,500 and you want a future yield of $10,000 after 24 months. The value of *vrate1* (the interest rate) is *.0068*, a monthly rate of .68 percent, or 8.16% annually.

```
SET VAR vrate1 = (RATE1(10000,8500,24))
```

1.16.5 RATE2

(RATE2(*fv*,*pmt*,*per*))

Returns the periodic interest rate on a series of regular payments, *pmt*, whose future value is *fv* over the number of compounding periods, *per*.

In the following example, RATE2 returns the interest rate you must have if your goal is $37,000 and you deposit $500 each month for five years. The value of *vrate2* is *.0069*, a monthly rate of .69 percent, or 8.28% annually.

```
SET VAR vrate2 = (RATE2(37000,500,60))
```

1.16.6 RATE3

(RATE3(*pv*,*pmt*,*per*))

Returns the periodic interest rate required for an annuity of value *pv*, to return a series of equal payments, *pmt*, over a number of compounding periods, *per*.

In the following example, RATE3 returns the interest rate of an annuity, purchased at $50,000, which will pay $570 monthly for 10 years. The value of *vrate3* is *.0055*, a monthly rate of .55 percent, or 6.6% annually.

```
SET VAR vrate3 = (RATE3(50000,570,120))
```

1.16.7 RDATE

(RDATE(*mon*,*day*,*yr*))

Converts integers *mon*, *day*, and *yr* to a DATE data type. *Yr* must be a four-digit year. The result returned by RDATE will vary, depending on the DATE format.

The following command assigns to the *transdate* column in the *transmaster* table the date derived from the values of *vmon* and *vday* (integers) as the month and day, and *1995* as the year in rows where the *transdate* column has a value.

```
UPDATE transmaster SET transdate = (RDATE(.vmon, .vday, 1995)) +
WHERE transdate IS NOT NULL
```

1.16.8 REVERSE

(REVERSE(*text*))

Returns the reverse order of text in a string.

In the following example, the value for vReverse is "erawtfoS evitcaretnI".

```
SET VAR vReverse TEXT = (REVERSE('Interactive Software'))
```

1.16.9 RJS

(RJS(*text,width*))

Right justifies *text* in *width* characters returning a text string.

In the following example, the value of *vrjs* is *ABCD*. The text string is right justified in a 10-character field.

```
SET VAR vrjs = (RJS('ABCD',10))
```

1.16.10 RNDDOWN

(RNDDOWN(arg1, arg2))

Returns a number rounded down to a specified number of digits. The resulting variable value will be a DOUBLE data type. *arg2* must be an INTEGER value.

Where: arg1 is the value to be rounded
arg2 is the position to be rounded down after the decimal point

Remarks:

- RNDDOWN behaves like ROUND, except that it always rounds a number down.
- If digits is greater than 0 (zero), then number is rounded down to the specified number of decimal places.
- If digits is 0, then number is rounded down to the nearest integer.
- If digits is less than 0, then number is rounded down to the left of the decimal point.

Examples

Example 01:

```
SET VAR v1 DOUBLE = (RNDDOWN(662.79,0))
SHOW VAR v1
662
```

Example 02:

```
SET VAR v2 DOUBLE = (RNDDOWN(662.79, 1))
SHOW VAR v2
662.7
```

Example 03:

```
SET VAR v3 DOUBLE = (RNDDOWN(54.1, -1))
SHOW VAR v3
50
```

Example 04:

```
SET VAR v4 DOUBLE = (RNDDOWN(55.1, -1))
SHOW VAR v4
50
```

Example 05:

```
SET VAR v5 DOUBLE = (RNDDOWN(-23.67, 1))
SHOW VAR v5
-23.6
```

1.16.11 RNDUP

(RNDUP(arg1, arg2))

Returns a number value rounded up to a specified number of digits. The resulting variable value will be a DOUBLE data type. *arg2* must be an INTEGER value.

Where: arg1 is the value to be rounded
arg2 is the position to be rounded up after the decimal point

Remarks:

- RNDUP behaves like ROUNDDWN, except that it always rounds a number up
- If arg2 is greater than 0 (zero), then number is rounded up to the specified number of decimal places.
- If arg2 is 0, then number is rounded up to the nearest integer.
- If arg2 is less than 0, then number is rounded up to the left of the decimal point.

Examples:

Example 01:

```
SET VAR vRoundUp1 DOUBLE = (RNDUP(762.273735,2))
SHOW VAR vRoundUp1
    762.28
```

Example 02:

```
SET VAR vRoundUp2 DOUBLE = (RNDUP(7796162.1455,-2))
SHOW VAR vRoundUp2
    7796200.
```

1.16.12 ROUND

(ROUND(arg1, arg2))

Returns a number rounded to a specified number of digits. The resulting variable value will be a DOUBLE data type. *arg2* must be an INTEGER value.

Where: arg1 is the value to be rounded
arg2 is the position to be rounded after the decimal point

Example 01:

```
SET VAR vNumber DOUBLE = 1.4567
SET VAR vRound = (ROUND(.vNumber,1))
```

Resulting vRound will be equal to 1.5

Example 02:

```
SET VAR vNumber DOUBLE = 1.4567
SET VAR vRound = (ROUND(.vNumber,2))
```

Resulting vRound will be equal to 1.46

Example 03:

```
SET VAR vNumber DOUBLE = 1.4567
SET VAR vRound = (ROUND(.vNumber,3))
```

Resulting vRound will be equal to 1.457

1.16.13 RTIME

(RTIME(*hrs,min,sec,frc*))

Converts integers *hrs*, *min*, *sec*, and *frc* to a TIME data type. *Hrs* is on a 24-hour scale. RTIME can be specified for up to thousandths of a second. The *frc* argument is optional.

In the following example, the value of *vrtime* is *12:15:30*, stored in internal R:BASE time format.

The time value will be displayed according to how you have set the TIME format. When you use time data in expressions, the result is given in seconds. You can use RTIME to convert this result to hours, minutes, and seconds. The value of *velapsed1* is *1170* seconds; the value of *velapsed2* is *0:19:30*.

```
SET VAR vrtime = (RTIME(12,15,30))
SET VAR vstart TIME = '1:10:40'
SET VAR vend TIME = '1:30:10'
SET VAR velapsed1 = (.vend - .vstart)
SET VAR velapsed2 = (RTIME(0,0,.velapsed1))
```

1.16.14 RTRIM

(RTRIM(*text*))

Trims trailing blanks from *text*, returning a text string.

In the following example, the value of *vrtrim* is the text string *ABCDE* without the trailing blanks.

```
SET VAR vrtrim = (RTRIM('ABCDE   '))
```

1.16.15 RWP

(RWP('*string*','*actualword*',*occurrence*))

Returns the position of a specified word and occurrence in a TEXT, NOTE or VARCHAR string. RWP = Relative Word Position

The resulting variable returns an integer value. If the word is not found a NULL is returned.

Example 01:

```
SET VAR vRWP INTEGER = (RWP('Imagine The
Possibilities','Possibilities',1))
```

Variable vRWP will return the value of 3.

Example 02:

```
SET VAR vRWP2 INTEGER = (RWP('We have shocked the world and we will do it
again!','we',2))
```
Variable vRWP2 will return the value of 7.

Example 03:
```
SET VAR vRWP3 INTEGER = (RWP('Keep in mind that nothing is
impossible','nothing',1))
```
Variable vRWP3 will return the value of 5.

1.17 S

1.17.1 SFIL

(SFIL(*chr*,*nchar*))

Fills a text string with a specified character *chr*, for *nchar* characters up to 500. You cannot use a number as a character. Instead, assign the number to a variable that has a TEXT data type using the variable name in SFIL.

In the following example, the value of *vsfil* is the text string `==========`. R:BASE programs often include SFIL to draw lines.

```
SET VAR vsfil = (SFIL('=',10))
```

1.17.2 SGET

(SGET(*text*,*nchar*,*pos*))

Gets *nchar* characters from *text* starting at *pos*, returning a text string.

In the following example, the value of *vsget* is *BCD*, the three characters starting in the second position of the text string.

```
SET VAR vsget = (SGET('ABCDE',3,2))
```

1.17.3 SIGN

(SIGN(*arg1*,*arg2*))

Transfers the sign of *arg2* to *arg1*. *Arg1* and *arg2* must be values that have DOUBLE, REAL, NUMERIC, or INTEGER data types.

In the following example, the value of *vsign* is *-15*, changing the sign of the first argument to the sign of the second argument.

```
SET VAR vsign = (SIGN(15,-20))
```

1.17.4 SIN

(SIN(*angle*))

Returns the trigonometric sine of *angle*.

In the following example, the value of *vsin* is *0.8659*.

```
SET VAR vsin = (SIN(1.047))
```

1.17.5 SINH

(SINH(*angle*))

Returns the hyperbolic sine of *angle*.

In the following example, the value of *vsinh* is *1.2491*.

```
SET VAR vsinh = (SINH(1.047))
```

1.17.6 SKEEP

(SKEEP(*source*, *chars*))

Keeps characters within the *source* string, using *chars* as a comparison.

This function is CASE SENSITIVE.

In the following example, the value of *vskeep* is *ldilliamennighway.*

```
SET VAR vskeep = (SKEEP('3935 Old William Penn
Highway','abcdefghijklmnopqrstuvwxyz'))
```

Spaces are also recognized.

In the following example, the value of *vskeep2* is *ld illiam enn ighway.*

```
SET VAR vskeep2 = (SKEEP('3935 Old William Penn Highway','
abcdefghijklmnopqrstuvwxyz'))
```

1.17.7 SKEEPI

(SKEEPI(*source*, *chars*))

Keeps characters within the *source* string, using *chars* as a comparison.

This function is NOT CASE SENSITIVE.

In the following example, the value of *vskeepi* is *OldWilliamPennHighway.*

```
SET VAR vskeepi = (SKEEPI('3935 Old William Penn
Highway','abcdefghijklmnopqrstuvwxyz'))
```

Spaces are also recognized.

In the following example, the value of *vskeepi2* is *Old William Penn Highway.*

```
SET VAR vskeepi2 = (SKEEPI('3935 Old William Penn Highway','
abcdefghijklmnopqrstuvwxyz'))
```

1.17.8 SLEN

(SLEN(*text*))

Returns the length of a *text string*.

SLEN can be used to ensure that a string does not exceed the space allowed for it on a form, variable form, or report. When strings are concatenated or passed as parameters to a procedure file, the length of a string might be unknown.

In the following example, the value of *vslen* is *5*, the number of characters in the text string.

```
SET VAR vslen = (SLEN('ABCDE'))
```

1.17.9 SLOC

(SLOC(*text,string*))

Locates *string* in *text*, returning the position if the string is found, *0* if it is not found.
In the following example, the value of *vsloc1* is *3*.

```
SET VAR vsloc1 = (SLOC('ABCDE','C'))
```

The value of *vsloc2* in the following example is *0*, since the text string *X* does not exist in the text string *ABCDE*.

```
SET VAR vsloc2 = (SLOC('ABCDE','X'))
```

In the following example, the *custzip* column contains a customer's zip code, in which the first five characters might be followed by a dash and another four characters. If the first row contained the string *02178-5243*, *Sloc3* would capture the position of the dash (*6*), which is in the sixth position.

```
SET VAR v5zip = custzip IN customer WHERE COUNT = 1
SET VAR sloc3 = (SLOC(.v5zip,'-'))
```

1.17.10 SLOCI

(SLOCI(*TextNoteVarcharValue,string,arg*))

Returns the INTEGER value for the number of instances a string appears in a TEXT, NOTE, or VARCHAR value.

The argument parameter determines whether the string search is case sensitive, where "0" is not case sensitive while "1" is case sensitive.

In the following example, the number of instances of a colon is 2.

```
SET VAR vProduct1 INTEGER = (SLOCI('R:BASE Single Seat: Upgrade',':',0))
```

In the following example, the number of instances of an upper case S is 2.

```
SET VAR vProduct2 INTEGER = (SLOCI('RBZip Single Seat License: Upgrade','S',1))
```

In the following example, the number of instances of an upper case or lower case S is 3.

```
SET VAR vProduct3 INTEGER = (SLOCI('RBZip Single Seat License: Upgrade','S',0))
```

1.17.11 SLOCP

(SLOCP(*TextNoteVarcharValue,string,occurrence*))

Locates the exact position of a given string and occurrence in a TEXT, NOTE or VARCHAR value, returning the position if the string is found, *0* if it is not found. Using -1 as the third parameter will return the LAST occurrence.

The resulting variable returns an integer value.

In the following example, the position for the first occurrence of *AB* is 1.

```
SET VARIABLE v1 VARCHAR = 'ABCDEABC_AB'
SET VARIABLE vslocp1 = (SLOCP(.v1,'AB',1))
```

In the following example, the position for the second occurrence of *AB* is 6.

```
SET VARIABLE v1 VARCHAR = 'ABCDEABC_AB'
SET VARIABLE vslocp2 = (SLOCP(.v1,'AB',2))
```

To find the LAST occurrence in the value of a string, use the "-1" parameter. In the following example, the position for the last occurrence of *AB* is 10:

```
SET VARIABLE v1 VARCHAR = 'ABCDEABC_AB'
SET VARIABLE vSLocLast = (SLOCP(.v1,'AB',-1))
```

1.17.12 SMOVE

(SMOVE(*text*,*pos1*,*nchar*,*string*,*pos2*))

From *text*, starting at position *pos1*, moves *nchar* characters to *string* starting at position *pos2*.

In the example below, the value of *vsmove1* is *XBCDX*. The characters *BCD* in the first string are moved into the second through fourth positions in the string *XYZXX*.

```
SET VAR vsmove1 = (SMOVE('ABCDE',2,3,'XYZXX',2))
```

In the following example, the *custcity* column is 12 characters wide and contains the string *ANCHORAGE* in the first row. You can use SMOVE to fill in a blank (13 characters in the example above) in order to customize a report title. The value of *vfulltitle* is *ANCHORAGE WAREHOUSE*.

```
SET VAR vcity = custcity IN customer WHERE COUNT = 1
SET VAR vfulltitle = (SMOVE(.vcity,1,12,'             WAREHOUSE',1))
```

1.17.13 SOUNDEX

(SOUNDEX(*value*))

Converts a text value to the corresponding SOUNDEX code.

1.17.14 SPUT

(SPUT(*text*,*string*,*pos*))

Puts *string* into *text,* starting at *pos*, returning a text string.

The value of *vsput1* in the following example is *AXCDE*. The character *X* is put into the second position in the string *ABCDE*.

```
SET VAR vsput1 = (SPUT('ABCDE','X',2))
```

1.17.15 SQRT

(SQRT(*arg*))

Returns square root of *arg*. *Arg* must be a positive value with a DOUBLE, REAL, NUMERIC, or INTEGER data type.

In the following example, the value of *vsqrt* is *10*.

```
SET VAR vsqrt = (SQRT(100))
```

1.17.16 SRPL

(SRPL(*sourcestring,searchstring,replacestring,*[0|1]))

Enables searching for and replacing text within a specified string of text.

sourcestring - specifies a string of text to search
searchstring - specifies text to search for
replacestring - specifies the replacement text
flag - 0 = replacement occurs for every matching string
- 1 = replacement occurs for whole word matches only

The following "0 flag" example replaces *04/20/64* with *04-20-64*:

```
SET VAR vsrpl = (SRPL('04/20/64','/','-',0))
```

The following "1 flag" example replaces *Dear Joe* with *Dear Anne*:

```
SET VAR vsrpl = (SRPL('Dear Joe','Joe','Anne',1))
```

However, with this next "1 flag" example *DearJoe* remains the same as *DearJoe* is a whole word without a space.

```
SET VAR vsrpl = (SRPL('DearJoe','Joe','Anne',1))
```

1.17.17 SSTRIP

(SSTRIP(*source*, *chars*))

Strips characters from the *source* string, using *chars* as a comparison.

This function is CASE SENSITIVE.

In the following example, the value of *vsstrip* is *3935 O W P H.*

```
SET VAR vsstrip = (SSTRIP('3935 Old William Penn
Highway','abcdefghijklmnopqrstuvwxyz'))
```

Spaces are also recognized.

In the following example, the value of *vsstrip2* is *3935OWPH.*

```
SET VAR vsstrip2 = (SSTRIP('3935 Old William Penn Highway','
abcdefghijklmnopqrstuvwxyz'))
```

1.17.18 SSTRIPI

(SSTRIPI(*source*, *chars*))

Strips characters from the *source* string, using *chars* as a comparison.

This function is NOT CASE SENSITIVE.

In the following example, the value of *vsstripi* is *3935 , 15668.*

```
SET VAR vsstripi = (SSTRIPI('3935 Old William Penn Highway,
15668','abcdefghijklmnopqrstuvwxyz'))
```

Spaces are also recognized.

In the following example, the value of *vsstripi* is *3935,15668.*

```
SET VAR vsstripi = (SSTRIPI('3935 Old William Penn Highway, 15668','
abcdefghijklmnopqrstuvwxyz'))
```

1.17.19 SSUB

(SSUB(*text,n*))

Captures substring number *n* from *text*, returning a text string. Substrings in *text* are separated by a comma (or the current delimiter).

The SSUB function is often used with the CHOOSE command when capturing menu options from a pull-down menu.

In the following example, the value of *vssub* is *yearend*.

```
SET VAR vtext = 'reports,yearend'
SET VAR vssub = (SSUB(.vtext,2))
```

The following example shows that SSUB separates items based on a blank rather than the current delimiter when *n* is less than zero.

```
SET VAR vtext = 'reports yearend'
SET VAR vssub2 = (SSUB(.vtext,-2))
```

For this example, assume that *twodim* is a bar with a pull-down menu with the options *Edit* and *Enter* stored as text numbers according to their positions on the menu. The pop-up menu for both options contains a list of form names, so the second item in the CHOOSE variable will be a form name.

```
CHOOSE vtwodim FROM twodim
SET VARIABLE vbar = (SSUB(.vtwodim,1))
SET VARIABLE vpull = (SSUB(.vtwodim,2))
IF vbar = '1' THEN
   EDIT USING .vpull
ELSE
   ENTER .vpull
ENDIF
```

1.17.20 SSUBCD

(SSUBCD(*text,n,delimiter*))

Captures substring number n from text, returning a text string. Substrings in text are separated with a specified custom delimiter.

In the following example, the value for vSubString is "Reports".

```
SET VAR vSubString = (SSUBCD('Forms|Reports|Labels',2,'|'))
```

1.17.21 STRIM

(STRIM(*text*))

Trims trailing blanks from *text*, returning a text string.

In the following example, the value of *vstrim* is the text string *ABCDE* without the trailing blanks.

```
SET VAR vstrim = (STRIM('ABCDE   '))
```

1.18 T

1.18.1 TAN

(TAN(*angle*))

Returns the trigonometric tangent of *angle*.

In the following example, TAN defines a new column *newtan* for the *mytable* table. *Newtan* is a computed column providing the tangent of the angle in radians stored in the *oldangle* column.

```
ALTER TABLE mytable ADD newtan = (TAN(oldangle)) DOUBLE
```

1.18.2 TANH

(TANH(*angle*))

Returns the hyperbolic tangent of *angle*.

In the following example, the value of *vtanh* is *.7616*.

```
SET VAR vtanh = (TANH(1))
```

1.18.3 TDWK

(TDWK(*arg*))

Returns the day of the week as text, where *arg* is a value that has either a DATE or DATETIME data type.

In the following example, the value of *vtdwk* is *Saturday*.

```
SET VAR vtdwk = (TDWK('12/02/95'))
```

1.18.4 TERM1

(TERM1(*pv*,*int*,*fv*))

Returns the number of compounding periods (the term) for a return of future value *fv*, based on the present value, *pv*, and the interest rate, *int*.

In the following example, TERM1 returns the number of months your money must stay invested if you want to accumulate $10,000 on an initial investment of $5,000 at a compounded monthly rate of 1% (12% annually). The value of *vterm1* (the term) is *70*.

```
SET VAR vterm1 = (TERM1(5000,.01,10000))
```

1.18.5 TERM2

(TERM2(*pmt*,*int*,*fv*))

Returns the number of compounding periods (the term) for a return of future value *fv*, based on the payment, *pmt*, and interest rate, *int*.

In the following example, TERM2 returns the number of years you must make payments if you want to accumulate $75,000 by making annual installments of $2,000 at 8% annual interest. The value of *vterm2* (the term) is *18*.

```
SET VAR vterm2 = (TERM2(2000,.08,75000))
```

1.18.6 TERM3

(TERM3(*pmt,int,pv*))

Returns the number of periods (the term) for the present value, *pv*, to reach 0 based on the payment, *pmt*, and the interest rate, *int*.

In the following example, TERM3 returns the number of payments from a $15,000 annuity if the annual interest rate is 12.5% compounded monthly and you would like to receive $300 every month. The value of *vterm3* is *71*.

```
SET VAR vterm3 = (TERM3(300,(.125/12),15000))
```

1.18.7 TEXTRACT

(TEXTRACT(*datetime*))

Returns the time portion of DATETIME.

In the following example, the value of *vtextract* is *12:15:30.123*.

```
SET VAR vtextract = (TEXTRACT('08/09/95 12:15:30.123'))
```

1.18.8 TINFO

(TINFO(arg1*,arg2,arg3*))

Returns access right information for the current user.

Where: arg1 is zero (0), which specifies to show table permissions
arg2 is the system table ID (SYS_TABLE_ID) value (can be located in the SYS_TABLES system table)
arg3 can be:
a) A specific system column ID (SYS_COLUMN_ID) in that table
b) A value of 0 which means show permissions that apply to all the columns in that table
c) A value of -1 which means show permissions that apply to any of the columns in that table

Remarks:

- TINFO returns a comma delimited string of the access rights.
- In most instances table and column privileges are identical. Cases where they are not are autonumber column (no INSERT), computed columns, (no INSERT or UPDATE), and where the UPDATE privilege was granted to specific columns only.
- Permissions are returned based upon the current USER.

Examples:
The following example is based upon a database configured with an OWNER and users, where limited permissions are supplied to a user Jim for the Component table.

While connected to the database as the OWNER, the system table ID and system column ID can be found for the Component table and CompDesc column.

```
SELECT SYS_TABLE_ID FROM SYS_TABLES WHERE SYS_TABLE_NAME = 'Component'
 SYS_TABLE_ID
 ------------
           29

R>SELECT SYS_COLUMN_ID FROM SYS_COLUMNS WHERE SYS_COLUMN_NAME = 'CompDesc'
 SYS_COLUMN_ID
 -------------
           188
```

```
R>SET USER JIM JIM999
```

Example 01:
-- Permissions for the column CompDesc

```
R>SET VAR vUserInfoColumn = (TINFO(0,29,188))
R>SHOW VAR vUserInfoColumn
SELECT, UPDATE
```

Example 02:
-- Permissions for all column. Only SELECT is returned since that is the only permission applied to all columns.

```
R>SET VAR vUserInfoAllColumns = (TINFO(0,29,0))
R>SHOW VAR vUserInfoAllColumns
SELECT
```

Example 03:
-- Permissions for any column. Both SELECT and UPDATE are returned since there are some columns with both of those permissions.

```
R>SET VAR vUserInfoAnyColumn = (TINFO(0,29,-1))
R>SHOW VAR vUserInfoAnyColumn
SELECT, UPDATE
```

1.18.9 TMON

(TMON(*arg*))

Returns the month name as text where *arg* is a value that has either a DATE or DATETIME data type.

In the following example, the value of *vtmon* is *November*.

```
SET VAR vtmon = (TMON('11/12/95'))
```

1.18.10 TRANSLATE

(TRANSLATE(*'inputstring','characters','translations','pad'*))

Returns the string provided as a first argument after some characters specified in the second argument are translated into a destination set of characters. In the process, TRANSLATE replaces a single character at a time. For example, it will replace the 1st character in the "input string" with the 1st character in the "replacement string". Then, it will replace the 2nd character in the "input string" with the 2nd character in the "replacement string", and so forth. The pad parameter is optional which can be used instead of a blank in the return string.

In the following example the square and curly braces in the input string are replaced with parentheses, resulting with "2*(3+4)/(7-2)".

```
SET VAR vBracketSwitch = (TRANSLATE('2*[3+4]/{7-2}','[]{}','()()'))
```

In the following example the "w" and "t" replace the "8" and "7" corresponding number values in the string. The "9" and "0" are replaced with the pad character, resulting in "123456tw..".

```
SET VAR vNumberText = (TRANSLATE('1234567890','8790','wt','.'))
```

1.18.11 TRIM

(TRIM(*text*))

Trims leading and trailing blanks from *text*, returning a text string.

In the following example, the value of *vtrim* is the text string *ABCDE* without the leading and trailing blanks.

```
SET VAR vtrim = (TRIM('   ABCDE   '))
```

1.19 U

1.19.1 UDF (User-Defined Functions)

(UDF ('*exe-name*','*parameter-list*'))
(UDF ('-*exe-name*','*parameter-list*'))
(UDF ('+*exe-name*','*parameter-list*'))
(UDF ('@dll-name','parameter-list'))

The logic behind using User Defined Functions or adding custom functionality to R:BASE is currently supported with plugins. Please refer to the PLUGIN command

You can use a user-defined function (UDF) anywhere you can use a function in R:BASE:

- Forms
- Reports
- Entry/exit procedures
- Applications
- Commands

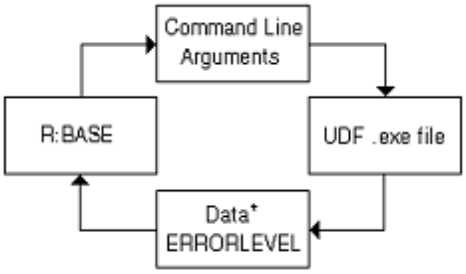
The arguments for a UDF are as follows:

@, +, -, or nothing
If you do not specify a plus or minus, the function is hidden when it runs; this setting is the default. A plus (+) runs the function in a normalized window. A minus (-) runs the function as a minimized icon. The at sign (@) is used to indicate that the UDF being called is a DLL. This only applies to Windows and is not available to the DOS version of R:BASE.

exe-name,dll-name
Specifies the name of the UDF (the name of the executable file into which you have compiled and linked your program). Under Windows this can be a Windows program. When using DLL's you MUST include the .DLL extension or the DLL may not be executed.

parameter-list
Specifies a text string. Multiple parameters must be separated by spaces, as shown:

```
SET VAR v1=(UDF('random','25 345'))
```



UDFs function as follows:

- R:BASE passes command line arguments to the UDF executable file.
- The UDF executable file completes its calculations and passes the data and the ERRORLEVEL to R:BASE.

This file can be found in the same directory as the main R:BASE executable.

When used with executables the parameter-list, from the UDF syntax, acts exactly as if you typed it at the a command prompt. Remember to account for the length of the executable name as well as the memory address (eight characters) plus the spaces between the .EXE name, the parameter list, and the address name. The parameter list can be space delimited with EXE UDF's.

You can write a UDF executable file in any language that allows writing to far pointers, such as Microsoft C. UDF executable files require protected mode libraries. You can return only 1,500 bytes of text data to R:BASE; that is, when you copy your results to either the address passed in or the shared-memory segment, you must not write more than 1,499 characters plus the NULL terminator (/0). If you write more than 1,500 total characters to the address you will corrupt the R:BASE variables and R:BASE might "hang."

If you forget the NULL terminator (/0) you will probably get high-order ASCII characters in your text variable. To pass back multiple values, delimit them with commas and then use other R:BASE functions to pull out the values. R:BASE stops looking at the returned text when it first encounters a NULL (/0). If your UDF executable file hangs, It is very likely that R:BASE will as well.

The size of a UDF is limited by the amount of available memory. Keep your executable files as small as possible. You will have to experiment to determine the maximum possible memory in a given situation. You can have different amounts of available memory at different times in an R:BASE session. Because you had an amount of memory available at one point in an R:BASE session does not mean that you will have the same amount of memory available in your next R:BASE session.

If your UDF fails to write to the memory given to it, then the UDF function in R:BASE sets the variable to NULL (-0-). You could use this in your UDF executable file to show whether the UDF failed. UDFs treat error variables the same as R:BASE does; therefore, you can have your application tell you if the UDF was successful or not by using the R:BASE error variable and the compiler's function that allows the return of error variables, such as Microsoft C exit( ) function to return the ERRORLEVEL. R:BASE places the ERRORLEVEL value in the R:BASE error variable.

Error Level
If your UDF fails to write to the memory given to it, then the UDF function in R:BASE sets the variable to NULL (-0-). You could use this in your own UDF executable file to show whether the UDF failed.

UDFs treat error variables the same as R:BASE does; therefore, you can have your application tell you if the UDF was successful or not by using the R:BASE error variable and the compiler's function that allows the return of error variables, such as Microsoft C exit( ) function to return the ERRORLEVEL. R:BASE places the ERRORLEVEL value in the R:BASE error variable.

Memory Issues
The size of a UDF is limited by the amount of available memory. Keep your executable files as small as possible. Depending on where the UDF is run and how much memory R:BASE has available to it at that moment, you could possibly run an executable file as large as 50K. You will have to experiment to determine the maximum possible memory in a given situation.

You can have different amounts of available memory at different times in an R:BASE session. Because you had an amount of memory available at one point in an R:BASE session does not mean that you will have the same amount of memory available in your next R:BASE session.

Returning Multiple Values
A UDF can return up to 1,500 characters. You can write a UDF that returns multiple values; then, you can write a command file that interprets coded strings and tells your application what was returned. For example, you could set up the following coded strings:

Coded String	What was done
100	Cube root of the input number
200	Standard deviation of input number

The UDF returns the information below to R:BASE with values for the coded strings 100 and 200 in comma-delimited format:

```
100,1.23456,200,2.34567
```

The command file would interpret that 1.23456 is the cube root of the input number, and 2.34567 is the standard deviation of the input number.

1.19.1.1 Sample UDF

The following example uses a UDF named dwrd to convert a currency value to words:

```
SET VAR v1=(UDF('dwrd','$1,456.99'))
```

If you execute the above command and then enter SHOW VARIABLE at the R> Prompt, you see the following output:

one thousand four hundred fifty-six dollars and ninety-nine cents

Examples

The following is a R:BASE command that calls the UDF named random.

```
SET VAR v1 = (UDF('random', '25 345'))
```

The following are the command line arguments for the UDF named random for DOS as seen by the UDF code:

```
DOS
argv[0] = random
argv[1] = 25
argv[2] = 345
argv[3] = A345BD3456
argc    = 4
```

Argument 0 is the name of the executable. Argument 1 is the first parameter. Argument 2 is the second parameter. Argument 3 is the memory address. This brings the argument count to 4.

The following example uses a UDF named DWRD.EXE to convert a currency value to words:

```
SET VAR v1=(UDF('dwrd','$1,456.99'))
SHOW VAR v1
```

Result: v1= one thousand, four hundred fifty six dollars and ninety nine cents

The following example uses the DWRD.DLL to accomplish the same function.

```
SET VAR v1=(UDF('@dwrd.dll','$1,456.99'))
SHOW VAR v1
```

Result: v1= one thousand, four hundred fifty six dollars and ninety nine cents

Source Code

What follows is the source code for the dwrd.exe sample UDF, shipped with DOS products, and the source code for the dwrd.dll sample UDF, shipped with Windows products.

```
/*
 *    Copyright 2001 by R:BASE Technologies, Inc.
 *    Author:  Wayne J. Erickson  12 Apr 1991
 *
 ***********************************************************************
```

```
 * *    Routine:  dwrd2 *** DOS VERSION ***
 * *    Purpose:  convert a currency value to words
 *
 *     Input Parameters:
 *            name                Brief description
 *            --------------      ----------------------------------------------
 *            value               Currency string to be parsed
 *            Shared Segment         Shared memory segment to place the parsed string
 *
 ************************************************************************
 */

#include
#include
#include

void conout(char *, int);
void crlf(void);
int lenstr(char *, int);
void dwrd(char *, char *);

cdecl main(argc,argv)
int argc;
char * argv[];
{
       long int i4;
       char *pt;
       char wordnum[160];
       int lw;

       /* Get the argument count.  */
       if(argc <= 1) {
              conout("DWRD number pointer", 19);
              crlf();
              goto done;
              }

       /* Do the conversion */
       dwrd(argv[1], wordnum);

       /* Display the result */
       lw = strlen(wordnum);
       if (argc == 2) {
              conout(wordnum, lw);
              crlf();
       }

       /* See if there was an address to store the result */
       if(argc > 2) {
              i4 = atol(argv[2]);
              pt = (char *)i4;
              memcpy(pt, wordnum, lw+1);
       }
done:
       return(0);
}


/**** Start of DWRD function ****/
void dwrd(strin, strout)
char *strin;
char *strout;
{
```

```
        int lstr;
        static char numbers[20][10] = {
                "         ","one       ","two       ",
                "three     ","four      ","five      ",
                "six       ","seven     ","eight     ",
                "nine      ","ten       ","eleven    ",
                "twelve    ","thirteen  ","fourteen  ",
                "fifteen   ","sixteen   ","seventeen ",
                "eighteen  ","nineteen  "};

        static char tenvals[10][10] = {
                "no        ","ten       ","twenty    ",
                "thirty    ","forty     ","fifty     ",
                "sixty     ","seventy   ","eighty    ",
                "ninety    "};
        char blank = ' ';
        char comma = ',';
        char dsign = '$';
        char dot = '.';
        char minus = '-';
        char ch;
        char tmoney[16];
        int offset;
        int tens;
        int lw, ls, n, intch, l;

        /*  CONVERT THE NUMBER. */

        tens = 0;
        memset(tmoney, ' ', 16);
        lw = 0;
        lstr = strlen(strin);
        ls = 16 - lstr + 1;
        strcpy(&tmoney[ls-1],strin);

        /*  PICK OFF THE CHARACTERS. */

        n = ls - 1;
next_digit:
        n++;
        if(n > 16) goto cleanup;
        ch = tmoney[n-1];

        /*  SKIP THE COSMETIC CHARACTERS. */

        if((ch == blank) || (ch == comma) || (ch == dsign)) goto next_digit;
        if(ch == minus) {
                memcpy(&strout[(lw+1)-1], "minus ", 6);
                lw = lw + 6;
                goto next_digit;
                }
        if((n == 4) || (n == 8) || (n == 12)) tens = 1;

        /*  SPECIAL STUFF WHEN WE HIT THE DECIMAL POINT. */

        if(ch == dot) {
                tens = 1;
                if(lw == 0) {
                        memcpy(&strout[(lw+1)-1], "zero ", 5);
                        lw = lw + 5;
                        }
                memcpy(&strout[(lw+1)-1], "dollars and ", 12);
                lw = lw + 12;
```

```
            if(tmoney[(n+1)-1] == '0') {
                if(tmoney[(n+2)-1] == '0') {
                    memcpy(&strout[(lw+1)-1], "zero ", 5);
                    lw = lw + 5;
                    n = n + 2;
                    }
                }
            goto next_digit;
            }

        /*  CONVERT A CHARACTER INTO AN OFFSET. */

        intch = ch - 48;
        if(tens) {
            if(intch <= 1) {
                offset = 0;
                if(intch == 1) offset = 10;
                n = n + 1;
                ch = tmoney[(n)-1];
                intch = ch - 48 + offset;
                }
            else {
                l = lenstr(tenvals[(intch+1) - 1], 10);
                if(l > 0) {
                    memcpy(&strout[(lw+1)-1], tenvals[(intch+1)-1], l+1);
                    lw = lw + l + 1;
                    }
                tens = 0;
                goto next_digit;
                }
            }
        l = lenstr(numbers[(intch+1)-1], 10);
        if(l > 0) {
            memcpy(&strout[(lw+1)-1], numbers[(intch+1)-1], l+1);
            lw = lw + l + 1;
            }
        tens = 0;
        if(((n == 3) || (n == 7) || (n == 11)) && (intch != 0)) {
            memcpy(&strout[(lw+1)-1], "hundred ", 8);
            lw = lw + 8;
            }
        if(n == 5) {
            memcpy(&strout[(lw+1)-1], "million ", 8);
            lw = lw + 8;
            }
        if(n == 9) {
            memcpy(&strout[(lw+1)-1], "thousand ", 9);
            lw = lw + 9;
            }
        goto next_digit;
    cleanup:
        memcpy(&strout[(lw+1)-1], "cents", 5);
        lw = lw + 5;

        /* Add the null terminator */
        strout[(lw+1)-1] = '\0';

        /*    ALL DONE.  */

        return;
    }
    /**** End of DWRD function ****/
```

```
/**** Start of CONOUT function ****/
void conout(str, count)
char *str;
int count;
{
	int i;
	for (i = 0; i < count; i++) {
		putchar(str[i]);
		}
	return;
}
/**** End of CONOUT function ****/

/**** Start of CRLF function ****/
void crlf()
{
	putchar('\r');
	putchar('\n');
	return;
}
/**** End of CRLF function ****/

/**** Start of LENSTR function ****/
int lenstr(str, count)
char *str;
int count;
{
	int i;

	i = count - 1;
	while (i >= 0) {
		if (str[i] != ' ') break;
		i--;
		}
	return (i+1);
}
/**** End of LENSTR function ****/



#include
/*
 *	Copyright 2001 R:BASE Technologies, Inc.
 *	Author:  Wayne J. Erickson  18 October 1999
 *
 ************************************************************************
 *
 *	Routine:  dwrd  *** DLL VERSION ***
 *
 *	Purpose:  convert a currency value to words
 *
 *	Input Parameters:
 *		name		Brief description
 *		--------------	----------------------------------------------
 *		value		Currency string to be parsed
 *		Shared Segment		Shared memory segment to place the parsed string
 *
 *
 ************************************************************************
 */

#include
void dwrd(char *, char *);
int lenstr(char *, int);
```

```
#pragma hdrstop

//---------------------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
	return 1;
}

//---------------------------------------------------------------------------
extern "C" __declspec(dllexport) int WINAPI RbDllExec(const HWND dc,const char *in, int inlen, char
*out, int outlen)
{
	int returnValue = 0;
	char instr[80];
	char wordnum[160];

	// Put the input string in a null terminated string
	memcpy(instr, in, inlen);
	instr[inlen] = '\0';

	// Convert to a text string
	dwrd(instr, wordnum);

	// Return the result
	outlen = strlen(wordnum);
	if (outlen)
		memcpy(out, wordnum, outlen);
	// returnValue = MessageBox(dc, wordnum,"Result value",MB_OK);
	// returnValue = 0;
	return returnValue;
}

//---------------------------------------------------------------------------
extern "C" __declspec(dllexport) int WINAPI RbDllVersion(void)
{
  return 100;
}

/**** Start of DWRD function ****/
void dwrd(char *strin,char *strout)
{
	int lstr;
	static char numbers[20][11] = {
		"         ","one       ","two       ",
		"three     ","four      ","five      ",
		"six       ","seven     ","eight     ",
		"nine      ","ten       ","eleven    ",
		"twelve    ","thirteen  ","fourteen  ",
		"fifteen   ","sixteen   ","seventeen ",
		"eighteen  ","nineteen  "};

	static char tenvals[10][11] = {
		"no        ","ten       ","twenty    ",
		"thirty    ","forty     ","fifty     ",
		"sixty     ","seventy   ","eighty    ",
		"ninety    "};
	char blank = ' ';
	char comma = ',';
	char dsign = '$';
	char dot = '.';
	char minus = '-';
	char ch;
```

```
        char tmoney[16];
        int offset;
        int tens;
        int lw, ls, n, intch, l;

        /*  CONVERT THE NUMBER. */

        tens = 0;
        memset(tmoney, ' ', 16);
        lw = 0;
        lstr = strlen(strin);
        ls = 16 - lstr + 1;
        strcpy(&tmoney[ls-1],strin);

        /*  PICK OFF THE CHARACTERS. */

        n = ls - 1;
next_digit:
        n++;
        if(n > 16) goto cleanup;
        ch = tmoney[n-1];

        /*  SKIP THE COSMETIC CHARACTERS. */

        if((ch == blank) || (ch == comma) || (ch == dsign)) goto next_digit;
        if(ch == minus) {
                memcpy(&strout[(lw+1)-1], "minus ", 6);
                lw = lw + 6;
                goto next_digit;
                }
        if((n == 4) || (n == 8) || (n == 12)) tens = 1;

        /*  SPECIAL STUFF WHEN WE HIT THE DECIMAL POINT. */

        if(ch == dot) {
                tens = 1;
                if(lw == 0) {
                        memcpy(&strout[(lw+1)-1], "zero ", 5);
                        lw = lw + 5;
                        }
                memcpy(&strout[(lw+1)-1], "dollars and ", 12);
                lw = lw + 12;
                if(tmoney[(n+1)-1] == '0') {
                        if(tmoney[(n+2)-1] == '0') {
                                memcpy(&strout[(lw+1)-1], "zero ", 5);
                                lw = lw + 5;
                                n = n + 2;
                                }
                        }
                goto next_digit;
                }

        /*  CONVERT A CHARACTER INTO AN OFFSET. */

        intch = ch - 48;
        if(tens) {
                if(intch <= 1) {
                        offset = 0;
                        if(intch == 1) offset = 10;
                        n = n + 1;
                        ch = tmoney[(n)-1];
                        intch = ch - 48 + offset;
                        }
```

```
            else {
                l = lenstr(tenvals[(intch+1) - 1], 10);
                if(l > 0) {
                    memcpy(&strout[(lw+1)-1], tenvals[(intch+1)-1], l+1);
                    lw = lw + l + 1;
                    }
                tens = 0;
                goto next_digit;
                }
            }
        l = lenstr(numbers[(intch+1)-1], 10);
        if(l > 0) {
            memcpy(&strout[(lw+1)-1], numbers[(intch+1)-1], l+1);
            lw = lw + l + 1;
            }
        tens = 0;
        if(((n == 3) || (n == 7) || (n == 11)) && (intch != 0)) {
            memcpy(&strout[(lw+1)-1], "hundred ", 8);
            lw = lw + 8;
            }
        if(n == 5) {
            memcpy(&strout[(lw+1)-1], "million ", 8);
            lw = lw + 8;
            }
        if(n == 9) {
            memcpy(&strout[(lw+1)-1], "thousand ", 9);
            lw = lw + 9;
            }
        goto next_digit;
cleanup:
        memcpy(&strout[(lw+1)-1], "cents", 5);
        lw = lw + 5;

        /* Add the null terminator */
        strout[(lw+1)-1] = '\0';

        /*    ALL DONE.  */

        return;
}
/**** End of DWRD function ****/

/**** Start of LENSTR function ****/
int lenstr(char *str, int count)
{
        int i;

        i = count - 1;
        while (i >= 0) {
            if (str[i] != ' ') break;
            i--;
            }
        return (i+1);
}
/**** End of LENSTR function ****/
```

Making and Setting DLL

You can use a DLL (Dynamic Link Library) which you make for use in R:BASE. DLLs which you make can be used column and variable objects in Forms, Reports and Labels.

If you make a DLL, you must follow some rules of making DLLs.

Rules of making DLLs

Value of DLL's version

- int WINAPI RbDllVersion(void);
- You must specify to return 100.

DLL name which is displayed in the Combo Box

- const char* WINAPI RbDllSpec(void);
- You designate the DLL function's name which is displayed in the Combo Box.

DLL's ID

- int WINAPI RbDllId(void);
- It is the identity number of the DLL. You designate to return a number form 2 to 127.

The function called by Form when drawing field

- int WINAPI RbDllDraw(const char*, const HDC, long, long, long, long)
- Return value : 0 success / Not 0 fails
- Argument : const char* Data strings
- const HDC Device Context handle
- long Left upper corner X position of object
- long Left upper corner Y position of object
- long Right lower corner X position of object
- long Right lower corner Y position of object

The function called by Report or Label when drawing field

- int WINAPI RbDllDrawP(const char*, const HDC, long, long, long, long)
- Return value : 0 success / Not 0 fails
- Argument : const char* Data strings
- const HDC Device Context handle
- long Left upper corner X position of object
- long Left upper corner Y position of object
- long Right lower corner X position of object
- long Right lower corner Y position of object

1.19.2 ULC

(ULC(*text*))

Converts *text* from uppercase to lowercase, returning a text string.

In the following example, the value of *vulc* is *abcde*. To ensure that your data is consistent, whether it is imported from outside R:BASE or entered through an R:BASE form, use ULC to convert text fields to lowercase.

```
SET VAR vulc = (ULC('ABCDE'))
```

1.20 W

1.20.1 WINUDF

(WINUDF('exe-name','parameter-list'))
(WINUDF('-exe-name','parameter-list'))
(WINUDF('+exe-name','parameter-list'))
(WINUDF('@dll-name','parameter-list'))

A new style of udf is available which should allow people to write UDF's in most any language they want. This results in a new function called WINUDF which has the same function parameters as the UDF function. The major difference is how R:BASE communicates with the WINUDF function. The exe program that WINUDF calls is passed one parameter which is the name of a file. This file has the parameters that should be passed to the WINUDF program. When the WINUDF program is completed, it is expected to write its results to the file that passed the original parameters.

Here are simple batch files that allow you to create WINUDF programs with at least 3 different C/C++ compilers. You could also use Visual Basic to create these new WINUDF's.

Using Visual Studio:

```
set path=c:\Program Files\Microsoft Visual Studio\VC98\Bin;c:\Program
Files\Microsoft Visual Studio\Common\MSDev98\Bin;%path%
set include=c:\Program Files\Microsoft Visual Studio\VC98\Include
set lib=c:\Program Files\Microsoft Visual Studio\VC98\Lib
cl /nologo /ML /W3 /Gm /GX /ZI /Od /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D
"_MBCS" /GZ /c DWRD.C
LINK /MAP:DWRD.MAP /SUBSYSTEM:CONSOLE DWRD.OBJ
```

Using Borland:

```
set path=c:\f\bcc55\bin;%path%
set include=c:\f\bcc55\include
set lib=c:\f\bcc55\lib
bcc32 -Ic:\f\bcc55\include -Lc:\f\bcc55\lib -M DWRD.C
```

Using Microsoft C:

```
set path=c:\f\msc6;C:\f\msc6\bin;c:\f\msc6\binb;%path%
set include=c:\f\msc6\include
set lib=c:\f\msc6\lib
ERASE SMALL.EXE
cl -c /Gc /Oas /AL /DLINT_ARGS DWRD.C
LINK @DWRD.LNK
EXEPACK DWRD.EXE SMALL.EXE
ERASE DWRD.EXE
REN SMALL.EXE DWRD.EXE
```

Examples:

```
SET VAR vTestA = (WINUDF('wscript.exe scrudf.vbs','The Power!'))
SET VAR vTestB = (WINUDF('wscript.exe scrudf.vbs','Oh Yes!'))
```

1.21 Y

1.21.1 YWRD

(YWRD(value))

Converts a currency value to its yen word representation. The function is similar to DWRD.

1.22 Z

1.22.1 ZHC

(ZHC(arg))

Convert double byte value to single byte value. The function is specific to R:BASE use for a computer whose system locale is configured for Japan.

The function is based on having previous style Japanese characters. Internally R:BASE has a table of character codes that correspond to the Japanese locale character mappings. The mappings are what has been used in Japan for a long time and are called "half width characters". There is another table that has to same characters but stored a "full width characters".

As an example, the lower case "a" followed by a blank equals 61 20 as hex values. HZC maps these to 82 81 as hex values which is a "full width" lower case "a". The ZHC function can take 81 81 and map it back to 61 20.

1.22.2 ZHCX

(ZHCX(arg))

Convert double byte value to single byte value. ZHCX is similar to ZHC except it treats a leading character of 0x82 specially. The function is specific to R:BASE use for a computer whose system locale is configured for Japan.

Part II

2 Aggregate Functions

An aggregate function can be used to provide summary data about a rows in a table or for a provided list of values.

Function	Supported Areas	Description
AVG	SELECT, COMPUTE, LAVG, Data Browser, Query Wizard, Query Builder, Form/Report/Label Expressions	Computes the numeric average. R:BASE rounds averages of integer values to the nearest integer value and currency values to their nearest unit.
COUNT	SELECT, COMPUTE, Data Browser, Query Wizard, Query Builder, Form/Report/Label Expressions	Determines how many non-null entries there are for a particular column item.
LISTOF	SELECT, Query Builder, Form/Report/Label Expressions	Creates a text string of the values separated by the current comma delimiter character. The LISTOF function can be used with to populate a variable with a list of values from multiple rows.
MAX	SELECT, COMPUTE, LMAX, Data Browser, Query Wizard, Query Builder, Form/Report/Label Expressions	Selects the maximum numeric, time, date, or alphabetic value in a column.
MIN	SELECT, COMPUTE, LMIN, Data Browser, Query Wizard, Query Builder, Form/Report/Label Expressions	Selects the minimum numeric, time, date, or alphabetic value in a column.
PSTDEV	SELECT	Calculates population standard deviation.
PVARIANCE	SELECT	Determines population variance.
STDEV	COMPUTE, Data Browser, Query Builder	Computes standard deviation. The standard deviation is a measure of how widely values are dispersed from the average value.
SUM	SELECT, COMPUTE, LAVG, Data Browser, Query Wizard, Query Builder, Form/Report/Label Expressions	Computes the numeric sum.
VARIANCE	COMPUTE, Data Browser, Query Builder	Determines variance.

***** Selecting aggregate functions, such as MIN and MAX, requires that R:BASE keeps an accumulator and choose to only use the first 80 characters for NOTE values. This matches the fact that if you sort on NOTE fields, the sort will be based on the first 80 characters only.

Part III

3 Useful Resources

. R:BASE Home Page:	https://www.rbase.com
. Up-to-Date R:BASE Updates:	https://www.rbaseupdates.com
. Current Product Details and Documentation:	https://www.rbase.com/rbg11
. Support Home Page:	https://www.rbase.com/support
. Product Registration:	https://www.rbase.com/register
. Official R:BASE Facebook Page:	https://www.facebook.com/rbase
. Sample Applications:	https://www.razzak.com/sampleapplications
. Technical Documents (From the Edge):	https://www.razzak.com/fte
. Education and Training:	https://www.rbase.com/training
. Product News:	https://www.rbase.com/news
. Upcoming Events:	https://www.rbase.com/events
. R:BASE Online Help Manual:	https://www.rbase.com/support/rsyntax
. Form Properties Documentation:	https://www.rbase.com/support/FormProperties.pdf
. R:BASE Beginners Tutorial:	https://www.rbase.com/support/rtutorial
. R:BASE Solutions (Vertical Market Applications):	https://www.rbase.com/products/rbasesolutions

Part IV

4 Feedback

Suggestions and Enhancement Requests:

From time to time, everyone comes up with an idea for something they'd like a software product to do differently.

If you come across an idea that you think might make a nice enhancement, your input is always welcome.

Please submit your suggestion and/or enhancement request to the R:BASE Developers' Corner Crew (R:DCC) and describe what you think might make an ideal enhancement. In R:BASE, the R:DCC Client is fully integrated to communicate with the R:BASE development team. From the main menu bar, choose "Help" > "R:DCC Client". If you do not have a login profile, select "New User" to create one.

If you have a sample you wish to provide, have the files prepared within a zip archive prior to initiating the request. You will be prompted to upload any attachments during the submission process.

Unless additional information is needed, you will not receive a direct response. You can periodically check the status of your submitted enhancement request.

If you are experiencing any difficulties with the R:DCC Client, please send an e-mail to rdcc@rbase.com.

Reporting Bugs:

If you experience something you think might be a bug, please report it to the R:BASE Developers' Corner Crew. In R:BASE, the R:DCC Client is fully integrated to communicate with the R:BASE development team. From the main menu bar, choose "Help" > "R:DCC Client". If you do not have a login profile, select "New User" to create one.

You will need to describe:

- What you did, what happened, and what you expected to happen
- The product version and build
- Any error message displayed
- The operating system in use
- Anything else you think might be relevant

If you have a sample you wish to provide, have the files prepared within a zip archive prior to initiating the bug report. You will be prompted to upload any attachments during the submission process.

Unless additional information is needed, you will not receive a direct response. You can periodically check the status of your submitted bug.

If you are experiencing any difficulties with the R:DCC Client, please send an e-mail to rdcc@rbase.com.

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- Z -

Notes