

Using dBase Expressions

Copyright © 2004 FrontRange Solutions, Inc.

Department: Technical Support

Created: April 21, 2004

Last Updated: April 24, 2004

Copyright

Copyright © 2004 FrontRange Solutions Inc. All Rights Reserved. GoldMine, HEAT, and other FrontRange products and brands are registered trademarks or trademarks of FrontRange Solutions Inc. in the U.S. and/or other countries. Other products and brands are registered trademarks or trademarks of their respective owners/companies.

USE OF THE SOFTWARE DESCRIBED IN THIS PAPER AND ITS RELATED USER DOCUMENTATION ARE SUBJECT TO THE TERMS AND CONDITIONS OF THE APPLICABLE END-USER LICENSE AGREEMENT (EULA).

The information contained in this document is provided “as is” without warranty of any kind. To the maximum extent permitted by applicable law, FrontRange Solutions disclaims all warranties, either express or implied, including the warranties for merchantability and fitness for a particular purpose; and in no event shall FrontRange Solutions or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if advised of the possibility of such damages.

Table of Contents

COPYRIGHT 2

USING DBASE EXPRESSIONS..... 4

 OVERVIEW 4

 FUNCTION/PARAMETER TYPES 4

 CONDITIONALS, OPERATORS AND LOGICAL EVALUATORS 5

Conditionals 6

Operators 8

Logical Evaluators 9

 DBASE FUNCTIONS 9

String Functions: 10

Date Functions: 15

Numeric Functions: 18

Miscellaneous Functions: 20

Using dBase Expressions

Overview

This document is provided for individuals who have at least an intermediate knowledge of programming. ***Improper use of these functions may result in data that is not recoverable.*** Please be sure to backup your data before attempting to use any of these functions.

GoldMine offers a variety of dBase expression functions to:

- Manipulate data for comparison, such as for creating filters and groups
- Store data, such as for global replacements and updates to field data (LOOKUP.INI)
- Evaluate and return data when using DDE and GMxS32.DLL function calls.

To ensure that your dBASE functions work correctly, GoldMine also features a realtime expression tester. To activate the tester on an active record window, press *Ctrl+Shift+d*.

Filter expressions work equally well on dBase or SQL tables. With SQL, the dBase filter is evaluated on the client side, not the server side.

The following pages list dBASE functions in three sections:

- Function/Parameter types
- Conditionals, Operators, and Logical Evaluators
- dBASE Functions

Function/Parameter Types

dBase functions recognize and return several types of data. These data types represent the format of the data, such as a number. To properly evaluate and return a value, a function must include the correct parameter types. For example, a function may require that a date is passed as a parameter. Trying to pass a name to the function would not be accepted. In many cases, you can use a special function to convert one data type to another.

Data types may be referenced literally, either as a field name of a specific type, or as the result of a dBase function.

The following list describes valid data types for dBase functions, and shows examples of use when referenced as a literal, field value, or function result.

String This is a sequence of any printable character.
Literal use: ("my string")
Field use: Upper(Contact1->Company)
Function Use: Upper(Substr("test123",5,3))

Date Special numeric value representing a date.
Literal use: {03/10/1999}
Field use: DTOS(Contact2->UBirthday)
Function use: DTOS(DATE())

Numeric Value representing a number.
Literal use: (100)
Field use: STR(Contact2->UBalance)
Function use: STR(100 + VAL("100"))

Boolean Value that results whenever a comparison is made. Boolean values are *either* TRUE or FALSE.

Conditionals, Operators and Logical Evaluators

A function can manipulate values by using one of the following:

- **Conditional:** compares one value to another using the specified standard, or condition, such as "equal to," "greater than," etc .
- **Operator:** performs an arithmetic operation on the values, such as addition or multiplication.
- **Logical evaluator:** compares values as a true/false condition, so that a value either meets or fails the standard for selection. This type of comparison is also known as a Boolean operator.

You can use the following conditionals, operators, and logical evaluators in conjunction with the dBase functions.

Conditionals

Conditional: =
Description: Compares values. With string values, = performs a “begins with” comparison.
Applies to: All types
Examples: "TEST"="TEST 123" returns: TRUE
"TEST"="123 TEST" returns: FALSE
Contact2->BirthDate=Date()
Contact2->UCOST=1024.08

Conditional: ==
Description: Exact comparison.
Applies to: Strings
Example: "TEST"=="TEST 123" returns: FALSE

Conditional: \$
Description: Is contained within. Tests if one string is contained within another.
Applies to: Strings
Example: "TEST" \$ "123 TEST 123" returns: TRUE
"TEST 123" \$ "TEST" returns: FALSE

Conditional: >
Description: Greater than
Applies to: All types
Examples: 1>2 returns: FALSE
"BBC">"ABC" returns: TRUE
Date(>Date()-10 returns: TRUE

Conditional: <
Description: Less than
Applies to: All types
Examples: 300<400 returns: TRUE
"MARCELA"<"NELSON" returns: TRUE
Date() < Date()-7 returns: FALSE

Conditional: <>
Description: Greater/Less than (not equal)
Applies to: All types
Examples: 250<>2500 returns: TRUE
"ABC"<>UPPER("abc") returns: FALSE
Date()<>Date()+3 returns: TRUE

Conditional: >=
Description: Greater than or Equal to
Applies to: All types
Examples: 100>=99 returns: TRUE
"ABC">="BBC" returns: FALSE
Date()+10>=-Date() returns: TRUE

Conditional: <=
Description: Less than or equal to
Applies to: All types
Example: 100<=99 returns: FALSE
"ABC"<="BBC" returns: TRUE
Date()+10<=Date() returns: FALSE

Operators

Operator: +

Description: Adds one value to another value

Applies to: All types

Examples: "ABC"+"DEF" (returns: "ABCDEF")

100+23 (returns: 123)

Date()+7 (returns: date one week from today)

Operator: -

Description: Subtracts one value from another value

Applies to: Numeric and Date types

Examples: 123-100 (returns: 23)

Date()-140 (returns: date of two weeks ago)

Operator: /

Description: Divides one number by another

Applies to: Numeric type

Example: 100/4 (returns: 25)

Operator: *

Description: Multiplies one value by another

Applies to: Numeric type

Example: 100*5 (returns: 500)

Operator: %

Description: Modulus

Applies to: Numeric type

Example: 100%33 (returns: 1)

Logical Evaluators

Logical: .OR.

Description: Returns TRUE if either condition is TRUE

Example: State="CA" .OR. Zip="99999"

Logical: .AND.

Description: Returns TRUE only if all conditions are TRUE

Example: Company="GoldMine Software" .AND. Phone1="(310)454-6800"

Logical: .NOT.

Description: Returns the opposite of the condition being tested

Example: .NOT. City="San Francisco"

dBASE Functions

GoldMine recognizes four types of dBASE functions as valid:

- ***String:*** Use primarily for manipulating string data types. A string function can return other data types.
- ***Date:*** Use for any date-related operations. A data function can return other data types.
- ***Numeric:*** Use for numeric operations. A numeric function can return other data types.
- ***Miscellaneous:*** Additional functions fall outside of the previous three categories of data types. These may return any type of data.

For convenience, functions are listed under these four categories, according to how they are most typically used. For example, under "Date Functions," you will find those functions that return numeric or string types from dates.

String Functions:

ALLTRIM(<string>) Returns a string value with both leading and trailing spaces from <string>.

Return type: String

Example

“["+ALLTRIM(" This is a test ")+""]”

returns [This is a test].

ASC(<char>) Returns the ASCII decimal value for <char>.

Return type: Numeric

Example

ASC("A")

returns 65.

AT(<string1>,<string2>) Returns the first position of <string1> in <string2>.

Return type: String

Example

AT("a", "once upon a time")

returns 11.

CHR(<byte>) Returns the ASCII character value for <byte>.

Return type: String

Example

CHR(65)

returns A.

FMTTIME(<time>) Returns a character string (hh:mm:pm format) derived from <time>.

Return type: String

Example

FMTTIME(TIME())

returns 2:28p.

HTTPSTR(<string>) Returns <string> with all nonletter/number characters replaced with %values.

Return type: String

Example

HTTPSTR("www.website.com/some dir")

returns www.website.com%2Fsome%20dir%2F.

IIF(<condition>,<>true result>,<>false result>) Returns *either* <true result> *or* <>false result>, depending on the Boolean evaluation of <condition>.

Return type: Logical

Example

IIF (99 < 100, "Value is Less than 100", "Value is more than 100")

returns "Value is Less than 100".

LEFT(<string>, <length>) Returns the leftmost <length> characters from <string>.

Return type: String

Example

LEFT("Four score and seven",10)

returns Four score.

LEN See LENGTH.

LENGTH(<string>) Returns the number of characters in <string>.

Return type: Numeric

Example

LENGTH("This is a test")

returns 14.

LOWER(<string>) Returns <string> in lower-case letters.

Return type: String

Example

LOWER("TEST THIS FUNCTION")

returns test this function.

LTRIM(<string>) Returns <string> with all leftmost spaces removed.

Return type: String

Example

"[" + LTRIM(" This is a test ") + "]"

returns [This is a test].

LTRIMPAD(<string>, <length>, <fill>) Returns <string> with leftmost spaces removed and padded to <length> with <fill> character.

Return type: String

Example

"[" + LTRIMPAD(" 1341", 10, "0") + "]"

returns 0000001341.

- MID(<string>, <start>, <length>)** Returns the string of <length> characters starting at position <start> within <string>.
- Return type: String
- Example**
MID("Four score and seven",6,5)
returns score.
- PAD(<string>, <length>, <fill>, <mode>)** Returns <string> padded to <length> with the <fill> character.
<fill>
- This optional parameter defaults to a space.
<mode>
- can be 0 for right pad (the default), 1 for centered, and 2 for left pad.
- Return type: String
- Example**
PAD("TEST", 8, "x", 1)
returns xxTESTxx.
- PADL(<string>, <length>, <fill>)** Returns <string> padded to <length> with the <fill> character.
<fill>
- This optional parameter defaults to a space. PADL pads from the left.
- Return type: String
- Example**
PADL("TEST", 8, "x")
returns xxxxTEST.
- PADR(<string>, <length>, <fill>)** Same as PADL, except that PADR pads the string to the right.
- Return type: String
- Example**
PADR("TEST", 8, "x")
returns TESTxxxx.
- PROPER(<string>)** Returns a string in which the first letter of *each word* in <string> is capitalized, and the all following letters are lower-case.
- Return type: String
- Example**
PROPER("fighting IRISH")
returns Fighting Irish.

- RAT(<string1>,string2>)** Returns the last position of <string1> in <string2>.
Return type: Numeric
Example
RAT("t", "this is a test.")
returns 14.
- RIGHT(<string>, <length>)** Returns the rightmost <length> characters from <string>.
Return type: String
Example
RIGHT("Four score and seven",5)
returns seven.
- RTRIM(<string>)** Returns <string> with all rightmost spaces removed.
Return type: String
Example
"[" + RTRIM(" This is a test " + "]"
returns [This is a test].
- STR(<value>,<length>,<decimal s>,<fill char>)** Returns the numeric <value> formatted as a string. The <value> parameters is required. All other parameters are optional. The <length> parameter pads the number to the left with spaces, or with the <fill char> if specified.
Return type: String
Example
STR(456, 7, 2, "0")
returns 0456.00.
- STRTRAN(<string1>, <string2>, <string3>)** Returns a string based on <string1> with all occurrences of <string2> translated to <string3>.
Return type: String
Example
STRTRAN("A1B1C1D1", "1", "x")
returns AxBxCxDx.
- SUBSTR(<string>, <start>, <length>)** Returns the string of <length> characters starting at position <start> within <string>.
Return type: String
Example
SUBSTR("Four score and seven",6,5)
returns score.

TRIM(<string>) See RTRIM.

UPPER(<string>) Returns the <string> in uppercase.

Return type: String

Example

UPPER("this is a test")

returns THIS IS A TEST.

WORD(<string>,<pos>) Returns the <pos> word within <string>.

Return type: String

Example

WORD("this is a test for the WORD function",4)

returns test.

Date Functions:

- ACCDATE(<string>)** Returns a date value for <string>, where <string> is a valid GoldMine AccountNo.
Return type: Date
Example
ACCDATE(Contact1->ACCOUNTNO)
returns 4/20/99.
- AGE(<date>)** Returns the age in years since <date>.
Return type: Numeric
Example
AGE(Contact2->UBDATE)
returns 32.
- CTOD(<string>)** Returns a date value based on <string>. The <string> parameter should be in the format: mm/dd/yy.
Return type: Date
Example
CTOD("4/20/99")+5
returns 4/25/99.
- DATE()** Returns today's date in date format. To add/subtract from this value, simply use the number of days in your expression. For example: DATE()+7 will add 7 days to today's date.
Return type: Date
Example
Assuming today's date is 4/20/99, DATE()+7
returns 4/27/99.
- DAY(<date>)** Returns that day of the month for the specified <date>.
Return type: Numeric
Example
DAY(DATE())
returns 18.
- DOBINDAYS(<date>)** Returns the number of days until the month/day in <date>.
Return type: Numeric
Example
DOBINDAYS(STOD("19681024"))
returns 232.

- DOW(<date>)** Returns the day of the week in numeric format; for example, Sunday = 0, Monday = 1, etc.
Return type: Numeric
Example
DOW(STOD("19990909"))
returns 4.
- DOY(<date>)** Returns the number of days elapsed from the beginning of the year in <date> to the month/day in <date>.
Return type: Numeric
Example
DOY(Contact2->UPDATE)
returns 220.
- DTOC(<date>)** Returns a character string (MM/DD/YY format) derived from <date>.
Return type: String
Example
DTOC(Contact2->UPDATE)
returns 10/24/98.
- DTOS(<date>)** Returns a character string (YYYYMMDD format) derived from <date>.
Return type: String
Example
DTOS(Contact2->UPDATE)
returns 19981024.
- MONTH(<date>)** Returns that numeric month for the specified <date>.
Return type: Numeric
example:
Example
MONTH(Contact2->UPDATE)
returns 2.
- STOD(<string>)** Converts a <string> value into a date value. <string> should be in the format YYYYMMDD.
Return type: Date
Example
STOD("19990122")
returns 1/22/1999.

WDATE(<date>, <format>) Returns the <date> formatted in variety of ways, based on the optional parameter <format>.

<format>

| | | |
|---|-----------------|------------------------|
| 0 | mmm dd, yy | Jan 22, 97 |
| 1 | ddd, mmm dd, yy | Thu, Jan 22, 97 |
| 2 | mmm dd | Jan 22 |
| 3 | Long date style | Thursday, Jan 22, 1997 |

The Long date style format 3 is taken from the Windows Regional Settings.

Return type: String

Example

WDATE(Contact2->UDate, 1)

returns Thu, Jan 22, 97.

YEAR(<date>) Returns the numeric year value of <date>.

Return type: Numeric

Example

YEAR(Contact2->UDate)

returns 1999.

Numeric Functions:

CEILING(<number>) Returns the nearest integer that is greater than or equal to the numeric expression.

Return type: Numeric

Example

CEILING(3.1)

returns 4.

COUNTER(<string>, <inc>, <start>, <action>) Returns a sequence of consecutive numbers each time the expression is evaluated. Each of the parameters is described below.

<name>

This counter must be unique, and can be a maximum of 10 characters.

<inc>

Each evaluation of the function increments the counter by the <inc> value.

<start> and **<action>**

Optional parameters

When <action> is 1, the <start> value is used to reset the counter. The counter is deleted when <action> is 2.

COUNTER works similarly to the SEQUENCE function. The key difference is that COUNTER stores the count value between GoldMine sessions, and is shared by all GoldMine users. The COUNTER function updates a database counter, so COUNTER is much slower than SEQUENCE, which updates a memory counter. The SEQUENCE counter is local to the operation, and its count is lost at the end of the operation.

GoldMine can track an unlimited number of uniquely named counters. The counter values are stored in the LOOKUP table.

Return type: Numeric

Example

COUNTER("InvoiceNo", 1, 1000)

returns 1000.

FLOOR(<number>) Returns the nearest integer that is less than or equal to the numeric expression

Return type: Numeric

Example

FLOOR(2.8)

returns 2.

INT(<number>) Returns the integer part of a number without rounding.

Return type: Numeric

Example

INT(123.95)

returns 123.

RANDOM(<range>) Returns a random number.

<range> can be any number between 1–32,761. The returned random number will range between zero and <range>, not including the range limit. If not specified, the <range> parameter defaults to 32,761. You can generate random numbers up to two billion with the expression random(32761) * random(32761).

Return type: Numeric

Example

RANDOM(10)

returns a number between 0–9.

SEQUENCE(<start>, <inc>) Returns a sequence of consecutive numbers each time the expression is evaluated. When the expression is first evaluated, the <start> parameter starts the counter. Each subsequent evaluation of the function increments the counter by the <inc> value. The SEQUENCE counter is local to the operation, and its count is lost at the end of the operation.

Return type: Numeric

VAL(<string>) Converts <string> to a numeric value.

Return type: Numeric

Example

VAL("123.45")

returns 123.45.

Miscellaneous Functions:

RECCOUNT() Returns the number of records in Contact1. (May be time-consuming on large SQL tables.)

Return type: Numeric

Example

RECCOUNT()

returns 35671.

RECNO() Returns the current record number (dBase) or RecID (SQL) for the active Contact1 record.

Return type: Numeric

Example

RECNO()

returns 351.

RECNOCOUNT() Returns the current record number and total records. This function is not available for SQL tables.

Return type: String

Example

RECNOCOUNT()

returns 236 of 2204.

TIME() Returns the current time.

Return type: Time

Example

TIME()

returns 14:56:22.