

# Reference Guide



**In the United States:**

Datawatch Corporation  
271 Mill Road  
Quorum Office Park  
Chelmsford, MA 01824  
Tech Support Fax: 978-275-8398  
Tech Support Phone: 978-441-2200

# Monarch<sup>TM</sup> 9 Functions

**Copyright Notice**

Monarch 9 Functions Reference Guide copyright © 2008 by Datawatch Corporation. All rights reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without written consent from Datawatch Corporation.

**Acknowledgments**

Throughout this manual, reference is made to several trademarks: Monarch and Datawatch are trademarks of Datawatch Corporation. All other brand and product names are trademarks or registered trademarks of their respective holders.

Printed in the United States of America  
Third Printing  
May 2008

# Table of Contents

## CHAPTER 1

<b>Overview of the Monarch Functions</b> .....	<b>1</b>
Date Functions .....	1
String Functions .....	2
Numeric Functions .....	4
Conversion Functions .....	5
Report Functions .....	6
Special Functions .....	7
Function Syntax Rules .....	8

## CHAPTER 2

<b>The Monarch Functions</b> .....	<b>9</b>
Abs function .....	9
Age function .....	10
Asc function .....	11
Ceiling function .....	12
CharToSeconds function .....	13
Chr function .....	15
Ctod function .....	16
Date function .....	17
DateAdjust function .....	18
DateToJulian function .....	19
Day function .....	20
DayOfYear function .....	20
Dtoc function .....	21
Exp function .....	22
Extract function .....	22
File function .....	23
Floor function .....	24
Hour function .....	25
ID function .....	26
If function .....	27
Instr function .....	27

Int function.....	28
InTrim function .....	29
IsAlpha function.....	30
IsBlank function.....	31
IsEmpty function.....	32
IsLower function .....	32
IsNull function.....	32
IsUpper function .....	34
JulianToDate function .....	35
Left function.....	36
Len function.....	37
Line function.....	38
Lower function.....	39
LSplit function.....	39
LTrim function .....	41
Max function.....	41
Min function.....	42
Minute function.....	43
Mod function.....	44
Month function.....	45
MRound function .....	45
NSplit function .....	46
Page function .....	47
Proper function.....	48
PSplit function .....	49
Qtr function.....	49
Rand function.....	50
RandEx function.....	51
Recno function .....	52
Replace function .....	53
Right function .....	54
Round function .....	55
Rowno function .....	56
RSplit function.....	57
RTrim function.....	58
Second function .....	59
SecondsToChar function .....	59
Space function .....	60
Sqrt function .....	61
Str function .....	62
Strip function .....	63
Stuff function .....	64
Substr function .....	65
TextLine function.....	66

---

Time function .....	67
Today function .....	67
Trim function .....	68
Upper function.....	69
Val function .....	69
Week function .....	70
Weekday function .....	71
Year function.....	72



## CHAPTER 1

# Overview of the Monarch Functions

Monarch provides a host of database functions that can be used in calculated field, filter and find expressions. Most functions require one or more arguments and return either a number, a character string or a date.

This chapter provides a brief description of each supported function and a list of function syntax rules. For detailed information on each function, including calculated field and filter/find expression examples, see Chapter 2.

## Date functions

Date functions operate on or with dates.

<b>Function</b>	<b>Description</b>
Date( <i>[date]</i> )	Returns the date portion of a given date/time or today's date.
DateAdjust( <i>date, years</i> <i>[, months[, days]]</i> )	Returns the given date value adjusted forward or backward by the given integral number of years, months, and optionally, days.
Day( <i>date</i> )	Returns the number of day (1-31) from a date.
DayOfYear( <i>date</i> )	Returns the day number (1-366) from a date/time.
Hour( <i>date</i> )	Returns the hour (0-23) from a date/time.
Minute( <i>date</i> )	Returns the minute (0-59) from a date/time.
Month( <i>date</i> )	Returns the number of month (1-12) from a date.

<b>Function</b>	<b>Description</b>
Qtr( <i>date</i> )	Returns the quarter of the year (1-4) within which a date falls: 1=January-March, 2=April-June, 3=July-September and 4=October-December.
Second( <i>date</i> )	Returns the second (0-59) from a date/time.
Time([ <i>date</i> ])	Returns the time portion of a date/time or the current time.
Today()	Returns today's date.
Week( <i>date</i> [, <i>startday</i> ])	Returns the number of week (1-53) from a date. <i>startday</i> designates the day that begins each week.
Weekday( <i>date</i> [ <i>,startday</i> ])	Returns the number of the weekday (1-7) of a date. <i>startday</i> designates the day that begins each week.
Year( <i>date</i> )	Returns the number of year from a date. Valid range is 1601-2400.

## String functions

String functions operate on or with character strings.

<b>Function</b>	<b>Description</b>
Extract( <i>string</i> , <i>start string</i> [ <i>,end string</i> ])	Extracts a substring between a starting string and an optional ending string.
Instr( <i>search string</i> , <i>target string</i> )	Returns the numeric position of <i>search string</i> in the field specified in <i>target string</i> . If the search string is not found, a value of zero is returned.
InTrim( <i>string</i> )	Trims consecutive spaces within <i>string</i> to a single space, and removes any leading or trailing spaces from the string.
IsAlpha( <i>character</i> )	Returns 1 (true) if <i>character</i> is alphabetic, otherwise returns 0 (false).

---

<b>Function</b>	<b>Description</b>
IsBlank( <i>string</i> )	Returns 1 (true) if <i>string</i> is empty or contains all blanks, otherwise returns 0 (false).
IsLower( <i>character</i> )	Returns 1 (true) if <i>character</i> is a lowercase alphabetic, otherwise returns 0 (false).
IsUpper( <i>character</i> )	Returns 1 (true) if <i>character</i> is an uppercase alphabetic, otherwise returns 0 (false).
Left( <i>string</i> , <i>n</i> )	Returns <i>n</i> number of characters from the beginning of <i>string</i> .
Len( <i>string</i> )	Returns the length of <i>string</i> as a number.
Lower( <i>string</i> )	Converts <i>string</i> to lowercase letters.
LSplit( <i>string</i> , <i>maxparts</i> , <i>sep</i> , <i>n</i> )	Starts on the left and splits <i>string</i> into <i>maxparts</i> number of parts using <i>sep</i> as the separator, then returns substring <i>n</i> .
LTrim( <i>string</i> )	Removes leading spaces from <i>string</i> .
NSplit( <i>string</i> , <i>n</i> )	Reads <i>string</i> as a person's name, splits it into five parts - prefix, first name, middle initial or middle name, last name, and suffix - then returns the specified substring ( <i>n</i> ).
Proper( <i>string</i> )	Forces to uppercase the first letter of each word in <i>string</i> .
PSplit( <i>string</i> , <i>n</i> )	Splits <i>string</i> as a postal code, splits it into 3 substrings - city, state, and postal code - then returns the specified substring ( <i>n</i> ).
Replace( <i>string</i> , <i>old</i> , <i>new</i> )	Replaces within the given <i>string</i> each instance of an old substring with a new substring.
Right( <i>string</i> , <i>n</i> )	Returns <i>n</i> number of characters from the end of <i>string</i> .

<b>Function</b>	<b>Description</b>
<code>RSplit(string,maxparts,sep,n)</code>	Starts on the right and splits <i>string</i> into <i>maxparts</i> number of parts using <i>sep</i> as the separator, then returns substring <i>n</i> .
<code>RTrim(string)</code>	Removes trailing spaces from a string.
<code>Space(number)</code>	Returns a string consisting of a specified number of spaces.
<code>Strip(string,stripchars)</code>	Strips the indicated characters ( <i>stripchars</i> ) from <i>string</i> .
<code>Stuff(s,p,n,c)</code>	Returns a string by replacing <i>n</i> number of characters in string <i>s</i> , starting at position <i>p</i> , using replacement string <i>c</i> .
<code>Substr(string,starting position,length)</code>	Extracts a substring of a specified <i>length</i> and <i>starting position</i> from <i>string</i> .
<code>TextLine(string,n)</code>	Splits a multi-line string at the line breaks and returns the specified line.
<code>Trim(string)</code>	Trims all leading and trailing spaces from <i>string</i> .
<code>Upper(string)</code>	Converts a string to uppercase letters.

## Numeric functions

Numeric functions operate on or with numbers.

<b>Function</b>	<b>Description</b>
<code>Abs(number)</code>	Returns the absolute value of <i>number</i> .
<code>Age(startdate[,enddate [,interval]])</code>	Returns the number of whole intervals between the given start date and end date.
<code>Ceiling(number[,number2])</code>	Rounds a number up (to the next multiple of <i>number2</i> , if specified).
<code>Exp(number)</code>	Returns e raised to a number.

<b>Function</b>	<b>Description</b>
Floor( <i>number</i> [, <i>number2</i> ])	Rounds a number down (to the previous multiple of <i>number2</i> , if specified).
Int( <i>number</i> )	Returns the integer portion of <i>number</i> .
Max( <i>expr1</i> , <i>expr2</i> [,...])	This function returns the larger of <i>expr1</i> , <i>expr2</i> , and optionally an arbitrary number of additional parameters.
Min( <i>expr1</i> , <i>expr2</i> [,...])	This function returns the smaller of <i>expr1</i> , <i>expr2</i> , and optionally an arbitrary number of additional parameters.
Mod( <i>number1</i> , <i>number2</i> )	Returns the remainder after dividing <i>number1</i> by <i>number2</i> .
MRound( <i>number</i> , <i>number2</i> )	Rounds a number to the nearest multiple of <i>number2</i> .
Round( <i>number</i> [, <i>decimals</i> ])	Rounds <i>number</i> to a specified number of places to the right (or left) of the decimal point. Negative values of <i>decimals</i> round the integer portion of the number rather than the decimal portion.
Sqrt( <i>number</i> )	Returns the square root of <i>number</i> .

## Conversion functions

Conversion functions convert a value of one type to another type, such as a date to a string.

<b>Function</b>	<b>Description</b>
Asc( <i>string</i> )	Returns the numeric value of a character.
CharToSeconds( <i>timestring</i> )	Converts a time string to a number of seconds.
Chr( <i>number</i> )	Returns the character value of a number.
Ctod( <i>string</i> [, <i>date format</i> [, <i>extraction format</i> ]])	Converts a string to date format.

Function	Description
DateToJulian( <i>date</i> [, <i>length</i> ])	Converts a date to a Julian date string of the given length. Default length is 5.
Dtoc( <i>date</i> )	Converts a date to character format.
JulianToDate( <i>string</i> )	Converts a Julian date string to a date.
SecondsToChar( <i>seconds</i> )	Converts a number of seconds to a time string.
Str( <i>number</i> [, <i>length</i> [, <i>decimals</i> [, <i>fill</i> ]])	Converts a number to a string. <i>length</i> , <i>decimals</i> and <i>fill</i> are optional. <i>length</i> is the total length of the string to return, <i>decimals</i> specifies the decimal position to round, and <i>fill</i> is a character used to fill up to the value specified in <i>length</i> .
Val( <i>string</i> )	Converts a string to a number. <b>Note:</b> The string must begin with a numeric character or a negation sign. If the string contains any non-numeric characters (apart from a decimal delimiter character), this function will convert the numeric portion of the string up to the first non-numeric character it encounters.

## Report functions

Report functions return information relating to the source of a record.

Function	Description
File()	Returns the name of the report file used to build a record.
ID()	Returns the number of the report file used to build a record. Each open report is assigned an incrementing ID, beginning with 1, designating the first report opened in the Monarch session.

<b>Function</b>	<b>Description</b>
Line()	Returns the report line number from which the record was generated. The line number is given relative to the top of the report page from which the record was generated.
Page()	Returns the page in the report file of the last detail line for a record.
Recno()	Returns the detail record number. Each record is assigned a record number when the record is generated from a detail line in the report (e.g., Recno=1 always corresponds to the first detail line extracted from the report). Record numbers do not change when a filter is applied to the table or when the table is sorted, but do change if the detail template is modified in such a way as to yield more or fewer records.
Rowno()	Returns the table row number of a record. This function differs from the Recno function in that row numbers are re-assigned each time you sort or filter the table while record numbers are not.

## **Special functions**

Monarch provides the following special functions that can be used to perform comparisons, check the status of fields, or generate random numbers (for use in auditing or other applications).

<b>Function</b>	<b>Description</b>
If( <i>condition,true value, false value</i> )	Returns <i>true value</i> or <i>false value</i> from a condition. If the expression is true, the true value is returned, otherwise the false value is returned. The return value for this function may be a date, a string, or a number.
IsEmpty( <i>field</i> )	Returns 1 if field is empty, otherwise returns 0.
IsNull( <i>expression</i> )	Returns 1 if <i>expression</i> is null, otherwise returns 0.
Rand()	Returns a record's random number in the range 0-32767.
RandEx()	Return's a record's random number in the range 0-4294967295.

## Function Syntax Rules

The following syntax rules apply to the use of functions in filter, find and calculated field expressions:

- ❑ A function may be entered in uppercase or lowercase letters.
- ❑ Function arguments must be enclosed in parentheses.
- ❑ Multiple arguments must be separated by commas (or semicolons, depending upon your Windows Language settings), and they must appear in the order specified by the function definition.
- ❑ A function may be embedded as an argument within another function.

## CHAPTER 2

# The Monarch Functions

This chapter contains detailed information on the Monarch functions. It provides a description of what each function “does”, shows the proper syntax and arguments for each function and, when applicable, includes examples of how each function can be used in calculated field and filter/find expressions.

## Abs function

### Description

Returns the absolute value of a number.

### Syntax

*Abs(number)*

### Arguments

*number* can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

This function returns a positive number.

### Calculated field example

If the Balance field contained the value of **-159.95**, the following expression would return the number **159.95**:

**Abs(Balance)**

## Filter/Find example

Records with a Balance field value of **-159.95** or **227.00** can be selected or found using the expression:

**Abs(Balance)>100.00**

## Age function

### Description

Returns the number of whole intervals between the given start date and end date.

The optional parameter *interval* specifies the time interval (i.e., the units) in which the age is computed, with 1=years, 2=months, 3=weeks, 4=days, 5=hours, 6=minutes, and 7=seconds. The default *interval* is 1 (years).

The *enddate* parameter is also optional, with a default value of Today(). Thus the expression Age(birthdate) will return a person's current age in years.

Note that *startdate* and *enddate* are full date/time values, so if they happen to include a time portion, the time is considered significant when computing the age, even when computing intervals of years, months, weeks or days. If *startdate* is after *enddate*, then the return value is negative.

**Note:** This function returns a null value if there is an error. An error can occur under the following conditions:

- ❑ The specified interval is not one of the interval values listed above (range is 1 to 7).
- ❑ The specified interval is 7 (meaning "seconds"), but the elapsed time would exceed the capacity of a long integer (2147483647). The cutoff at which the age in seconds begins to return an error is an elapsed time of 24855 days, 3 hours, and 14 minutes (i.e., a little over 68 years).

### Syntax

Age(*startdate*[,*enddate*[,*interval*]])

## Arguments

*startdate* designates the date that begins the period for which you want to measure whole time intervals.

*enddate* designates the date that ends the period for which you want to measure whole time intervals.

*interval* is an optional parameter that specifies the time interval (i.e., the units) in which the “age” is computed, with 1=years, 2=months, 3=weeks, 4=days, 5=hours, 6=minutes, and 7=seconds. The default *interval* is 1 (i.e., years).

## Return value

This function returns a number.

## Calculated field example

Using a StartDate of 04/15/1957 and an EndDate of 01/27/2005, the following calculated field expression would return 47 (years):

**Age(StartDate,EndDate, 1).**

## Filter/Find example

Records with an “age” of 47 years can be found using the expression:

**Age(StartDate,EndDate,1)=47**

# Asc function

## Description

Returns the numeric value of a character.

## Syntax

*Asc(string)*

## Arguments

*string* can be a constant, a character field, an expression that results in a single character or a function that returns a single character.

## Return value

This function returns a number in the range 0-255. **Note:** If you pass a string of more than one character to this function, the function will return the value of only the first character of the string.

## Calculated field example

If the Amount field contained the values "876.00" and "¥1022" (indicating an amount in Japanese yen), the following expression would return the string "Dollars" or "Yen":

**If(Asc(Amount)=165,"Yen","Dollars")**

The first part of this expression -- If(Asc(Amount)=165 -- determines whether the Yen symbol (character code 165) appears at the beginning of the Amount field. If it does, the first value "Yen" is returned, otherwise the second value "Dollars" is returned.

## Filter/Find example

Records with a Amount field value beginning with the Japanese yen symbol (¥) can be selected or found using the expression:

**Asc(Amount)=165**

# Ceiling function

## Description

Rounds a number up (to the next multiple of *number2*, if specified).

## Syntax

Ceiling(*number*[,*number2*])

## Arguments

*number* is the value to be rounded. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*number2* is the optional multiple to which *number* is to be rounded. If this is not specified then it defaults to 1.0 (or -1.0) causing rounding/truncating to the nearest integer.

### Return value

This function returns a number. If *number2* is zero, or if the sign of *number2* differs from the sign of *number*, then this function returns a null value.

### Calculated field examples

If the Unit\_price field contained the value 2.5, the following expression would return the number 3:

**Ceiling(Unit\_price,1)**

If the Unit\_price field contained the value 12.46, the following expression would return the number 12.50:

**Ceiling(Unit\_price,0.25)**

### Filter/Find example

Records with a Unit\_price field value of 35.01 or 35.17 can be selected or found using the expression:

**Ceiling(Unit\_price,0.50)=35**

## CharToSeconds function

### Description

Converts a time string to a number of seconds.

### Syntax

CharToSeconds(*timestring*)

## Arguments

*timestring* is a string that represents the time of day, in one of the following formats:

Format	12 Hour Clock	24 Hour Clock	Military
h:mm tt	2:04 AM	2:04	204
h:mm:ss tt	2:04:58 AM	2:04:58	20458
h:mm:ss.000 tt	2:04:58.125 AM	2:04:58.125	20458.125
hh:mm tt	02:04 AM	02:04	0204
hh:mm:ss tt	02:04:58 AM	02:04:58	020458
hh:mm:ss.000 tt	02:04:58.125 AM	02:04:58.125	020458.125

If the format contains an AM or PM, the hour is based on the 12-hour clock, where "AM," "am," "A," or "a" indicates time from midnight until noon, and "PM," "pm," "P," or "p" indicates time from noon until midnight. Otherwise, the hour is based on the 24-hour clock.

## Return value

This function returns a number representing the number of seconds since 12:00 AM. If the time string cannot be successfully parsed, a null value is returned.

## Calculated field example

If the *Start\_time* field contained the string 12:10:45AM, and the *End\_time* field contained the string 12:30:00AM, the following expression would return the number 1155, representing a duration of 19 minutes and 15 seconds:

**CharToSeconds(End\_time)-CharToSeconds(Start\_time)**

## Filter/Find example

Records with a duration of less than 1 minute can be selected or found using the expression:

**CharToSeconds(End\_time)-CharToSeconds(Start\_time)<60**

## Chr function

### Description

Returns the character value of a number.

### Syntax

*Chr(number)*

### Arguments

*number* can be a constant, a numeric field, an expression that results in a number or a function that returns a number. For predictable results, the number should be in the range 0-255 (the extent of the character set).

### Return value

This function returns a string.

### Calculated field example

If the Amount field contained the values "876.00" and "¥1022" (indicating an amount in Japanese yen), the following expression would return the string "Dollars" or "Yen":

**If(Left(Amount,1)=Chr(165), "Yen", "Dollars")**

The first part of this expression -- If(Left(Amount,1)=Chr(165)) -- determines whether the Yen symbol (character code 165) appears at the beginning of the Amount field. If it does, the first value "Yen" is returned, otherwise the second value "Dollars" is returned.

### Filter/Find example

Records with an Amount field value beginning with the yen symbol (¥) can be selected or found using the expression:

**Left(Amount,1)=Chr(165)**

## Ctod function

### Description

Converts a string to date format.

### Syntax

```
Ctod(string[,date format[,extraction format]])
```

### Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

*date format* is an optional argument that tells Monarch how to interpret the date. Valid options for this parameter are "m/d/y", "d/m/y" and "y/m/d" (the quotation marks are required).

The optional extraction format parameter specifies the "date/time extraction pattern". Possible extraction patterns are as follows:

Pattern	Meaning
"D"	Extract a date.
"DT"	Extract a date followed by a time.
"T"	Extract a time.
"TD"	Extract a time followed by a date.

**Note:** More than one extraction pattern may be combined to form a list of extraction patterns. For example: "D,DT" means extract a date OR a date followed by a time.

Extraction patterns are *not* case-sensitive.

If an explicit extraction pattern is not provided to Ctod, it uses the default extraction pattern from the model or registry.

### Return value

This function returns a date in the form YYYYMMDD.

## Calculated field example

Monarch provides a date format setting that determines how date fields are interpreted when extracted from a report file. But some reports contain dates in more than one format. To allow for multiple date formats, the Ctod function accepts an optional *type* parameter that lets you specify how to interpret the date.

For example, let's say a report has two dates. The Rpt\_date field values are expressed as 03/08/2001 (MDY format) but the Ship\_date field values is expressed as 2001/03/08 (YMD format).

To extract both dates, set the Date Format setting to MDY. Monarch will interpret the Rpt\_date field, but will extract the Ship\_date (YMD) as a character field. To convert the character field to a date field, use the following expression:

**Ctod(Ship\_date,"y/m/d")**

The optional "y/m/d" parameter tells Monarch how to interpret the date, in this case as YMD format. Valid options for this parameter are "m/d/y", "d/m/y" and "y/m/d" (the quotation marks are required).

## Filter/Find example

Records with a Hire\_date field containing dates prior to 19800601 (June 01, 1980) can be selected or found using the expression:

**Hire\_date<=Ctod("June 01, 1980")**

# Date function

## Description

Returns the date portion of a given date/time or today's date.

## Syntax

Date([date])

## Arguments

Date/time (optional).

## Return value

This function returns a date in the form YYYYMMDD.

## Calculated field example

If you were reading this on September 17, 2001, the following function would return the value 20010917 (today's date in YYYYMMDD format):

**Date()**

## Filter/Find example

Records representing invoices more than thirty days past due can be selected or found using the expression:

**Date()-Due\_date>30**

# DateAdjust

## Description

Returns the given date value adjusted forward or backward by the given integral number of years, months, and optionally, days.

**Note:** DateAdjust does not adjust the time portion, if any, of the given date/time value. Thus the adjusted date/time value will represent the same time on a different date.

## Syntax

DateAdjust(*date*,*years*[,*months*[,*days*]])

## Arguments

*date* specifies the date field to adjust forward or backward.

*years* specifies the number of years to adjust *date* forward or backward.

*months* specifies the number of months to adjust *date* forward or backward.

**Note:** If this optional argument is omitted, it defaults to zero.

*days* specifies the number of days to adjust *date* forward or backward. **Note:** If this optional argument is omitted, it defaults to zero.

**Note:** The values passed in for the years, months, and days parameters are taken to be integers. Any fractional portions of these values are simply ignored. Negative values adjust the date backward, positive values adjust forward.

## Return value

This function returns a date.

## Calculated field example

If the `Ship_date` field contained the value **4/6/2006**, the following calculated field expression would return a value of **5/7/2007** (i.e., 1 year, 1 month and 1 day after the ship date):

**DateAdjust(Ship\_date,1,1,1)**

Editing the expression to read **DateAdjust(Ship\_date,-1,-1,-1)** would return a value of **3/5/2005**, i.e., 1 year, 1 month and 1 day *before* the ship date.

# DateToJulian

## Description

Converts a date to a Julian date string of the given length. Default length is 5.

## Syntax

DateToJulian(*date*[,*length*])

## Arguments

*date* is any date value. *length* is an optional parameter to specify how many digits of Julian date are desired. The default length is 5. Allowable values for *length* are 3, 4, 5, or 7.

## Return value

Returns a Julian date string of the specified length.

## Calculated field example

The following expression will return **2003305** from a Date field containing **11/1/2003 12:05:33 AM**:

**DateToJulian(Date,7)**

## Day function

### Description

Returns the number of day (1-31) from a date.

### Syntax

Day(*date*)

### Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

### Return value

This function returns a number in the range 1-31.

### Calculated field example

If the Ship\_date field contained the value **20010526**, the following expression would return the value **26**:

**Day(Ship\_date)**

### Filter/Find example

Records with the dates **19890722**, **19920526** or **19940317** in the Ship\_date field can be selected or found using the expression:

**Day(Ship\_date)>15**

## DayOfYear function

### Description

Returns the number of day (1-366) from a date/time.

### Syntax

DayOfYear(*date*)

## Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

## Return value

This function returns a number. The return value is the day of the year where January 1st=1, January 2nd=2, etc. The range of return values is 1-366, where 366 is possible only in leap years.

## Calculated field example

The following expression would return **305** from a Date field containing **11/1/2003 12:05:33 AM**:

**DayOfYear(Date)**

# Dtoc function

## Description

Converts a date to character format.

## Syntax

Dtoc(*date*)

## Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

## Return value

This function returns a string in the form MM/DD/YYYY.

## Calculated Field example

The following expression would convert "1999-04-21" in the Ship Date field to "04/21/1999".

**Dtoc([Ship Date])**

## Exp function

### Description

Returns  $e$  – the base of natural logarithms – raised to a number. **Note:** The constant  $e$  approximately equals 2.718282.

### Syntax

`Exp(number)`

### Arguments

*number* can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

This function returns a number.

### Calculated field example

The following expression returns the value of  $e$  calculated to 7 decimal places (i.e., 2.7182818):

`Exp(1)`

## Extract function

### Description

Extracts a substring from between a starting string and an optional ending string. The string to delimit the start of the desired substring *must* be specified. The string to delimit the end of the substring is optional. If the second argument is not found the function extracts everything from the starting string to the end of the field. The case of the delimiting string is ignored.

### Syntax

`Extract(string,start string[,end string])`

## Arguments

*string* is the string containing the substring. The string can be a constant, a character field, an expression that results in a string or a function that returns a string.

*start string* is the string (required) delimiting the start of the desired substring.

*end string* is the string (optional) delimiting the end of the desired substring.

**Note:** If *end string* is not specified, this function extracts everything from *start string* to the end of the field.

## Return value

This function returns a character string.

## Calculated field example

If the Letters field contained the value of "abc[def]ghi", the following expression would return "def":

```
Extract(Letters,"[","]")
```

## Filter/Find example

Records with a Letters field value of "def" can be selected or found using the expression:

```
Extract(Letters,"[","]") = "def"
```

# File function

## Description

Returns the name of the report file used to build a record.

## Syntax

```
File()
```

## Arguments

None.

## Return value

This function returns a string in the form of a path and filename (e.g., C:\Program Files\Monarch\Reports\Classmay.prn).

## Calculated field example

If both the Classapr.prn report and the Classmay.prn report were loaded from the C:\Reports directory, the following expression would return the value "C:\Reports\Classapr.prn" or C:\Reports\Classmay.prn", depending upon which report file each record in the table was extracted from:

**File()**

## Filter/Find example

The following filter or find expression would return or find those records which were extracted from the Classapr.prn file:

**RSplit(File(),2,"\",1)="Classapr.prn"**

In this example, the File() function would return the path and filename of the source report file for each record in the table. The RSplit function searches the string from right to left for the backslash character (\), then splits the string at that position into two substrings, returning the rightmost substring (the filename without the path). This is then compared to the string "Classapr.prn". Only the records extracted from the Classapr.prn file would match this expression.

# Floor function

## Description

Rounds a number down (to the previous multiple of *number2*, if specified).

## Syntax

Floor(*number*[,*number2*])

## Arguments

*number* is the value to be rounded. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*number2* is the optional multiple to which *number* is to be rounded. If this is not specified then it defaults to 1.0 (or -1.0), causing rounding/truncating to the nearest integer.

## Return value

This function returns a number. If *number2* is zero, or if the sign of *number2* differs from the sign of *number*, then this function returns a null value.

## Calculated field examples

If the Unit\_price field contained the value “2.5”, the following expression would return the number “2”:

**Floor(Unit\_price,1)**

If the Unit\_price field contained the value “12.46”, the following expression would return the number “12.25”:

**Floor(Unit\_price,0.25)**

## Filter/Find example

Records with a Unit\_price field value of “35.49” or “35.17” can be selected or found using the expression:

**Floor(Unit\_price,0.50)=35**

# Hour function

## Description

Returns the hour (0-23) from a date/time.

## Syntax

Hour(*date*)

## Arguments

*date* is a date/time containing a time portion.

## Return value

This function returns a number in the range 0-23.

## Calculated field example

The following expression will return "12" from a Date field containing "11/1/2003 12:05:33 AM":

**Hour(Date)**

## ID function

### Description

Returns the number of the report file used to build a record. Monarch assigns an ID number to each report when the report is opened. The first report opened is assigned 1, the second 2, etc. When a report is closed, Monarch re-assigns ID numbers to collapse the list (e.g., when the second of three open reports is closed, the ID of the third report is re-assigned from 3 to 2).

### Syntax

ID()

### Arguments

None.

### Return value

This function returns a number.

## Calculated field example

If both the Classapr.prn report and the Classmay.prn report were opened, in that order, the following expression would return the value "1" for all records extracted from the Classapr.prn report and "2" for all records extracted from the Classmay.prn report:

**ID()**

## Filter/Find example

The following filter or find expression would return or find those records which were extracted from the second open report (i.e., Classmay.prn):

**ID()=2**

## If function

### Description

Returns *true value* or *false value* from a condition.

### Syntax

If(*condition,true value,false value*)

### Arguments

*condition* can be any valid expression which performs a comparison, such as "Amount>1000 or Quantity\*Price=Total".

*true value* is the value returned if the condition is true. This value can be a constant, a field, an expression or a function.

*false value* is the value returned if the condition is false. This value can be a constant, a field, an expression or a function.

### Return value

This function may return a number, a date or a character string.

### Calculated field example

To create a Sale\_price field that calculates a 20% discount off quantities of 5 or more, use the expression:

**If(Quantity>=5,Amount\*0.80,Amount)**

Quantity>=5 comparison expression. If the quantity is greater than or equal to 5, the function returns Amount\*0.80, representing a 20% discount. For quantities less than 5, the function returns the full Amount field value, representing no discount.

## Instr function

### Description

Returns the numeric position of the search string in the field specified in the target string.

## Syntax

`Instr(search string,target string)`

## Arguments

*search string* is the substring you want to locate. The search string can be a constant, a character field, an expression that results in a string or a function that returns a string.

*target string* is the string containing the substring you want to locate. The target string can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a number representing the offset of the search string within the target string. If the search string is not found, the function returns 0.

## Calculated field example

If the Lastname field contained the value "Bradley", the following expression would return the number "5" (indicating that "ley" was first found starting at character position 5 in the Lastname field):

`Instr("ley",Lastname)`

## Filter/Find example

Records with a Lastname field value of "Bradley", "Ashley", or "Pleyton" could be selected or found using the expression:

`Instr("ley",Lastname)>0`

# Int function

## Description

Returns the integer portion of a number.

## Syntax

`Int(number)`

## Arguments

*number* can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

## Return value

This function returns a number.

## Calculated field example

If the Price field contained the value “10.99”, the following expression would return the number “10” (i.e., the integer portion of the field's value):

**Int(Price)**

## Filter/Find example

Records with a Price field value of “10.99”, “10.02” or “10” can be selected or found using the expression:

**Int(Price)=10**

# InTrim

## Description

Trims any sequence of consecutive spaces (i.e., two or more adjacent spaces) within a string to a single space. It also removes any leading or trailing spaces from the string.

## Syntax

InTrim(*string*)

## Arguments

*string* is the string in which you want to trim consecutive spaces greater than one space. *string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a string.

### Calculated field example

If a Contact field contained " John Smith ", the following calculated field expression would return "John Smith":

**InTrim(Contact)**

### Filter/Find example

Records with the value "John Smith" could be selected or found with the expression:

**Intrim(Contact)="John Smith"**

## IsAlpha function

### Description

Determines whether *character* is alphabetic.

### Syntax

IsAlpha(*character*)

### Arguments

*character* can be a constant, a character field, an expression that results in a string or a function that returns a string. If *character* is longer than a single character, IsAlpha operates on only the first character.

### Return value

This function returns the number 1 if *character* is alphabetic, otherwise it returns 0.

### Calculated field example

If the Part\_no field begins with an alphabetic character, the following expression would return the field value, otherwise the expression will return a blank value.

**If(IsAlpha(Part\_no)=1,Part\_no,"")**

This expression might be used to return only part numbers that begin with an alphabetic character code.

### Filter/Find example

Records with a Part\_no field beginning with an alphabetic character can be selected or found using the expression:

**IsAlpha(Part\_no)=1**

## IsBlank function

### Description

Determines whether a string is empty or contains all blanks.

### Syntax

IsBlank(*string*)

### Arguments

*string* can be a character field, an expression that results in a string or a function that returns a string.

### Return value

This function returns the number 1 if *string* is empty or contains only blanks, otherwise it returns 0 (if the field contains data, for example).

### Calculated field example

If the Amount field contained no value or just spaces, the following expression would return a 1, indicating that the field is empty:

**IsBlank(Amount)**

### Filter/Find example

Records with a blank Amount field can be selected or found using the expression:

**IsBlank(Amount)=1**

## IsEmpty function

### Description

Determines whether a field is empty.

### Syntax

IsEmpty(*field*)

### Arguments

*field* can be the name of any field in the table.

### Return value

Returns 1 if the field is empty, otherwise it returns 0 (i.e., if the field contains data).

### Calculated field example

If the Name field contained no value, the following calculated field expression would return a 1, indicating that the field is empty:

IsEmpty(Name)

### Filter/Find example

Records with a blank Name field can be selected or found using the expression:

IsEmpty(Name)=1

## IsLower function

### Description

Determines whether *character* is lowercase alphabetic.

### Syntax

IsLower(*character*)

## Arguments

*character* can be a constant, a character field, an expression that results in a string or a function that returns a string. If *character* is longer than a single character, IsLower operates on only the first character.

## Return value

Returns 1 if *character* is lowercase alphabetic, otherwise returns 0.

## Calculated field example

The IsLower() function has no practical application in a calculated field expression.

## Filter/Find example

Records with a Name field beginning with a lowercase character can be selected or found using the expression:

```
IsLower(Name)=1
```

# IsNull function

## Description

Determines whether *expression* is null.

## Syntax

```
IsNull(expression)
```

## Arguments

*expression* is a value, field, or expression (character, number or date) to be evaluated.

## Return value

Returns 1 if *expression* is null, otherwise returns 0.

### Calculated field example

The following expression returns the value of the Average Price field if the value is valid. If the Average Price Field value is invalid, the expression returns a number that indicates this (i.e., 999999).

**If(IsNull([Average Price])=1,999999,[Average Price])**

### Filter/Find example

This expression removes records with Null values in the Average Price field:

**IsNull([Average Price])=0**

## IsUpper function

### Description

Determines whether *character* is uppercase alphabetic.

### Syntax

IsUpper(*character*)

### Arguments

*character* can be a constant, a character field, an expression that results in a string or a function that returns a string. If *character* is longer than a single character, IsUpper operates on only the first character.

### Return value

Returns 1 if *character* is uppercase alphabetic, otherwise returns 0.

### Calculated field example

The IsUpper() function has no practical application in a calculated field expression.

## Filter/Find example

Records with a Part\_no field that have an uppercase alphabetic character in the third character position (75T versus 75t) can be selected or found using the expression:

```
IsUpper(Substr(Part_no,3,1))=1
```

## JulianToDate function

### Description

Converts a Julian date string to a date.

### Syntax

```
JulianToDate(string)
```

### Arguments

*string* is a Julian date string, for example "02105". The Julian date string must consist of only digits, and may have a length of 3, 4, 5 or 7 digits. In each case, the last 3 digits specify the day of the year (e.g., 001=January 1st, 002=January 2nd, 031=January 31st, 032=February 1st, etc).

### Return value

This function returns a date.

If the Julian date string has 7 digits, then the first 4 digits unambiguously specify the year. If the Julian date string has 5 digits, then the first 2 digits are taken as a 2-digit year that is then resolved to a complete year value using the normal pivot year mechanism. If the Julian date string has only 4 digits, then the first digit is taken as the least significant digit of a year. In order to resolve this to a complete year value, JulianToDate assumes that it refers to some year within the last 10 years. For example: "2105" would be interpreted as 4/15/2002 (the 105th day of 2002), whereas "3105" would be interpreted as 4/15/1993 (the 105th day of 1993). If users need to override this behavior they are encouraged to prefix their data with the desired digits. For example, if they want "3105" to be interpreted as 4/15/2003, then they could prefix their 4-digit Julian dates with the digits "200". If the Julian date string has only 3 digits, then JulianToDate assumes that it refers to a date within the current year. Thus "105" is interpreted as 4/15/2002. Again, users can always prefix their data with a suitable 4-digit year if necessary.

**Note:** The function returns NULL if the given string cannot be interpreted as a Julian date.

### Calculated field example

The following expression will return “4/6/2001” from a Julian Date field containing “01096”:

**JulianToDate([Julian Date])**

## Left function

### Description

Extracts a string of specified length from the beginning of another string.

### Syntax

`Left(string,n)`

### Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

*n* is the length of the substring you want to extract. The length can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

Returns *n* number of characters from the beginning of a string.

### Calculated field example

If the Company field contained the value “Widgets Inc”, the following expression would return “Widgets”:

**Left(Company,7)**

## Filter/Find example

Records with a Company field value of "Wright Corp." or "Wright Brothers" can be selected or found using the expression:

```
Left(Company,6)="Wright"
```

## Len function

### Description

Returns the length of a string as a number.

### Syntax

```
Len(string)
```

### Arguments

*string* can be a constant, a character field, a memo field, an expression that results in a string or a function that returns a string.

### Return value

Returns the length of a string as a number. **Note:** If *string* is a memo field, the return value accounts for a Carriage Return and Line Feed character at the end of each line, except for the last line. For example, a 5 line memo field with a template width of 20 will return a length of 108 (5 lines \* 20 characters to account for the data in the field plus 4 lines \* 2 characters to account for the Carriage Return and Line Feed characters that Monarch adds to all but the last line).

### Calculated field example

If the City field contained the value "Pittsfield" or "Chelmsford", the following expression would return the number 10:

```
Len(City)
```

**Note:** Len(City) returns the actual length of the City field, not the length of the data in the field. If the field contains trailing spaces, use the RTrim function inside the Len function to first remove the trailing spaces, as in Len(RTrim(City))

### Filter/Find example

Records with a blank Job\_title field can be selected or found using the expression:

**Len(Job\_title)=0**

or

**Len(RTrim(Job\_title))=0**

## Line function

### Description

Returns the report line number from which the record was generated. The line number is given relative to the top of the report page from which the record was generated.

### Syntax

Line()

### Arguments

None.

### Return value

This function returns a number.

### Calculated field example

The following expression returns the report page and line for each record, as a string:

**"Page: "+LTrim(Str(Page()),7,0)+" Line: "+LTrim(Str(Line()),3,0)**

The output of this expression is in the form:

**Page: 125 Line: 17**

### Filter/Find example

This function might be used along with the Rand() function to perform auditing of report data, but otherwise has no practical value in a filter expression.

## Lower function

### Description

Converts all uppercase letters in a string to lowercase letters.

### Syntax

Lower(*string*)

### Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

### Return value

This function returns a character string in lowercase letters.

### Calculated field example

If the Billing field contained the value "MONTHLY", the following expression would return "monthly":

**Lower(Billing)**

### Filter/Find example

All filter expressions are case insensitive, so Lower has no practical application. For find expressions, use the Match Case setting to determine case sensitivity.

## LSplit function

### Description

Starts on the left and splits *string* into *maxparts* number of parts using *sep* as the separator, then returns substring *n*.

### Syntax

LSplit(*string,maxparts,sep,n*)

## Arguments

*string* is the string you want to split apart. It can be a constant, a character field, an expression that results in a string or a function that returns a string.

*maxparts* represents the maximum number of substrings you want to parse the string into. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*sep* is the character that tells Monarch where to split the string. It can be a single character or a substring.

*n* tells Monarch which substring (from the left) you want the function to return. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

## Return value

This function returns a character string.

## Calculated field example

If the Title field contained the value "Bach, Cantatas BWV 56,57,57,152", the following expression would return "Cantatas BWV 56,57,57,152":

**LSplit(Title,2,",",2)**

The first argument (Title) specifies the field to split. The second argument (2) tells Monarch to split the field into a maximum of 2 substrings. The third argument (",") tells Monarch where to split the field, in this case on the comma between "Bach" and "Cantatas". The fourth argument (2) tells Monarch which substring to return, in this case the second substring "Cantatas BWV 56,57,57,152".

**Note 1:** If we were to specify "3" as the second argument, Monarch would split the field into three substrings of "Bach", "Cantatas BWV 56" and "57,57,152". This time Monarch would return "Cantatas BWV 56", the second of three substrings.

**Note 2:** In this example it makes sense to split the field based upon the comma delimiter. You can use any character as the delimiter, but be careful, Monarch does not include the delimiter when it breaks out each substring.

## Filter/Find example

Records with a Title field value of "Bach, Art of Fugue" or "Bach, Cantatas BWV 56,57,57,152" can be selected or found using the expression:

**LSplit(Title,2,",",1)="Bach"**

## LTrim function

### Description

Removes all leading spaces from a string.

### Syntax

LTrim(*string*)

### Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

### Return value

This function returns a character string.

### Calculated field example

If the Product field contained the value of " chair", the following expression would return "chair":

**LTrim(Product)**

### Filter/Find example

Records with a Product field value of "chair" or " chair" can be selected or found using the expression:

**LTrim(Product)="chair"**

## Max function

### Description

Returns the larger of *expr1*, *expr2*, and optionally an arbitrary number of additional parameters. **Note:** All parameters must be of the same type: numeric, date or character.

### Syntax

Max(*expr1*,*expr2*[,...])

## Arguments

*expr* can be a constant, a numeric field, a character field, a date field or an expression. **Note:** In the case of a character field, z would be larger than x.

## Return value

This function returns the larger of *expr1* and *expr2*, and optionally an arbitrary number of additional parameters. Accepts numeric, date or character input.

**Note:** In the case of a character field, z would be larger than x.

## Calculated field example

If the Test1, Test2 and Test3 fields contained a series of numeric test values, the following calculated field expression would return the difference between the maximum test value and the minimum test value:

**Max(Test1,Test2,Test3)-Min(Test1,Test2,Test3)**

## Filter/Find example

Records with a third test value above both the first and second test values can be selected or found using the expression:

**Test3>Max(Test1,Test2)**

This expression is equivalent to the compound expression:

**Test3>Test1.And.Test3>Test2**

# Min function

## Description

Returns the smaller of *expr1*, *expr2* and optionally an arbitrary number of additional parameters. **Note:** All parameters must be of the same type: numeric, date or character.

## Syntax

Min(*expr1,expr2[,...]*)

## Arguments

*expr* can be a constant, a numeric field, a character field, a date field or an expression. **Note:** In the case of a character field, x would be smaller than z.

## Return value

This function returns the smaller of *expr1*, *expr2* and optionally an arbitrary number of additional parameters. Accepts numeric, date or character input.

**Note:** In the case of a character field, x would be smaller than z.

## Calculated field example

If the Projected field contained the value "1000000" and the Actual field contained the value "40000", the following expression would return the value "40000":

**Min(Projected,Actual)**

## Filter/Find example

Records with Projected and Actual values both greater than or equal to "50000" can be selected or found using the expression:

**Min(Projected,Actual)>=50000**

This expression is equivalent to the compound expression:

**Projected>=50000.And.Actual>=50000**

# Minute function

## Description

Returns the minute (0-59) from a date/time.

## Syntax

Minute(*date*)

## Arguments

*date* is a date/time containing a time portion.

## Return value

This function returns a number in the range 0-59.

## Calculated field example

The following expression will return “5” from a Date field containing “11/1/2003 12:05:33 AM”:

**Minute(Date)**

# Mod function

## Description

Returns the remainder after dividing *number1* by *number2*.

## Syntax

`Mod(number1,number2)`

## Arguments

*number1* is the dividend (i.e., the number or numeric expression) that is divided by *number2*.

*number2* is the divisor (i.e., the number or numeric expression) by which *number1* is divided.

## Return value

This function returns a numeric value.

## Calculated field example

If the Minutes field contained the number “32446”, and the outstanding minutes past the particular hour is required, the following expression would return the value “46”:

**Mod(Minutes,60)**

## Filter/Find example

Records in which the Minutes field is equal to 46 past the hour can be found with:

**Mod(Minutes,60)=46**

## Month function

### Description

Returns the number of month (1-12) from a date.

### Syntax

Month(*date*)

### Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

### Return value

This function returns a number in the range 1-12.

### Calculated field example

If the Ship\_date field contained the value "20010704", the following expression would return the value "7":

**Month(Ship\_date)**

### Filter/Find example

Records with the date "19910114", "19890131" or "19940107" in the Ship\_date field can be selected or found using the expression:

**Month(Ship\_date)=1**

## MRound function

### Description

Rounds a number to the nearest multiple of *number2*.

### Syntax

MRound(*number,number2*)

## Arguments

*number* is the value to be rounded. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*number2* is the multiple to which *number* is to be rounded. If this is not specified then it defaults to 1.0 (or -1.0), causing rounding/truncating to the nearest integer.

## Return value

This function returns a number. If *number2* is zero, or if the sign of *number2* differs from the sign of *number*, then MRound returns a null value.

## Calculated field examples

If the Unit\_price field contained the value “2.5”, the following expression would return the number 3:

**MRound(Unit\_price,1)**

If the Unit\_price field contained the value “12.46”, the following expression would return the number “12.50”:

**MRound(Unit\_price,0.25)**

## Filter/Find example

Records with a Unit\_price field value of “34.70” or “35.17” can be selected or found using the expression:

**MRound(Unit\_price,0.50)=35**

# NSplit function

## Description

Splits a string representing a person's name into five substrings (Prefix, Firstname, Middle, Lastname, and Suffix), then returns the specified substring.

## Syntax

NSplit(*string,n*)

## Arguments

*string* is the person's name. The string can be a constant, a character field, an expression that results in a string or a function that returns a string.

*n* tells Monarch which substring you want the function to return. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

## Return value

This function returns a character string.

## Calculated field example

If the Name field contained the value "Dr. Juan H. Fernandez, DDS", the following expression would return "Fernandez":

**NSplit(Name,4)**

## Filter/Find example

Records with a Name field value of "Dr. Elizabeth R. Jones, MD" or "Dr. Peterson, MD" can be selected or found using the expression:

**NSplit(Name,5)="MD"**

**Note:** The substrings are always numbered as follows: 1=Prefix, 2=Firstname, 3=Middle name (or initial), 4=Lastname, 5=Suffix. This rule applies in all cases, even when one or more of the components is missing from a name.

# Page function

## Description

Returns the page in the report file of the last detail line for a record.

## Syntax

Page()

## Arguments

None.

## Return value

This function returns a number.

## Calculated field example

The following expression returns the report page and line for each record as a string:

**"Page: "+LTrim(Str(Page()),7,0)+" Line: "+LTrim(Str(Line()),3,0)**

The output of this expression is in the form:

**Page: 125 Line: 17**

## Filter/Find example

This function might be used along with the Rand() function to perform auditing of report data, but otherwise has no practical value in a filter expression.

# Proper function

## Description

Converts to uppercase the first letter of each word in a string.

## Syntax

Proper(*string*)

## Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a character string.

## Calculated field example

If the Book Title field contained the value "the great gatsby", the following expression would return "The Great Gatsby":

**Proper([Book Title])**

## PSplit function

### Description

Splits a postal address line into three substrings (City, State and Zipcode), then returns the specified substring.

### Syntax

`PSplit(string,n)`

### Arguments

*string* is the address line. The string can be a constant, a character field, an expression that results in a string or a function that returns a string.

*n* tells Monarch which substring you want the function to return. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

This function returns a character string.

### Calculated field example

If the Postal field contained the value "Boston, MA 02145", the following expression would return "02145":

`PSplit(Postal,3)`

### Filter/Find example

Records with a Postal field value of "Boston, MA 02145" or "Pittsfield, MA, 01201" can be selected or found using the expression:

`PSplit(Postal,2)="MA"`

## Qtr function

### Description

Indicates which quarter of the year a date falls within: 1 = January - March, 2 = April - June, 3 = July - September and 4 = October - December.

## Syntax

`Qtr(date)`

## Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

## Return value

This function returns a number in the range 1-4.

## Calculated field example

If the `Ship_date` field contained the value 20010915, the following expression would return the value 3 (indicating that the 9th month of the year falls in the 3rd quarter):

`Qtr(Ship_date)`

## Filter/Find example

Records with `Ship_date` values which fall in the first half of the year, such as 20010401, 19990215 or 19940629, can be selected or found using the expression:

`Qtr(Ship_date)=1.Or.Qtr(Ship_date)=2`

# Rand function

## Description

Returns a record's random number in the range 0-32767.

## Syntax

`Rand()`

## Arguments

None.

## Return value

This function returns a number in the range 0-32767.

## Calculated field example

To generate a field filled with random numbers, use the following expression:

Rand()

This application of the Rand() function can be useful in auditing, for example (see the Filter/Find example).

## Filter/Find example

Certain tasks, such as auditing, require that you generate a random subset of the table database. This can be done in Monarch using the Rand() function. A random group of records can be selected using the filter expression:

**Rand()>=300.And.Rand()<=350**

You may use any numeric range in the set of numbers from 0-32767 to select a random group of records. You should be aware, however, that with this use of the Rand() function, the number of records returned is variable. If none of the randomly generated numbers happen to fall within the range you specify, Monarch will return a blank table database.

# RandEx function

## Description

Returns a record's random number in the range 0-4294967295. **Note:** This is an expanded version of the Rand() function, which only returns numbers in the range 0-32767.

## Syntax

RandEx()

## Arguments

None.

## Return value

This function returns a number in the range of 0-4294967295.

## Calculated field example

To generate a field filled with random numbers, use the following expression:

### **RandEx()**

This application of the RandEx() function can be useful in auditing, for example (see the Filter/Find example).

## Filter/Find example

Certain tasks, such as auditing, require that you generate a random subset of the table database. This can be done in Monarch using the RandEx() function. A random group of records can be selected using the filter expression:

### **RandEx()>=300.And.RandEx()<=350**

You may use any numeric range in the set of numbers from 0-4294967295 to select a random group of records. You should be aware, however, that with this use of the RandEx() function, the number of records returned is variable. If none of the randomly generated numbers happen to fall within the range you specify, Monarch will return a blank table database.

# Recno function

## Description

Returns the detail record number of a record in the table. You can think of the detail record number as the detail line number, since any given record number will always correspond to the same detail line from the report (e.g., Recno = 1 always corresponds to the first detail line extracted from the report, and Recno = 12 always corresponds to the 12th detail line, etc.). This applies even if the table has been sorted. This is not the case for the row numbers that you see at the left edge of the table.

## Syntax

Recno()

## Arguments

None.

## Return value

This function returns a number.

## Calculated field example

The following expression creates a calculated field which displays the record number for each record in the table:

**Recno()**

## Filter/Find example

Records 20-25 can be selected or found using the expression:

**Recno()>=20.And.Recno()<=25**

Every third record can be returned by using the expression:

**Recno()/3=Int(Recno()/3)**

# Replace function

## Description

Replaces a substring in a string with a new substring. **Note:** Replacements are case-sensitive.

## Syntax

Replace(string,old,new)

**Note:** Replacements are case-sensitive. If old is empty then no replacements are made. If new is empty, then instances of old are effectively removed from the string.

## Arguments

*string* is the string that contains the substring you want to replace.

*old* is the substring to be replaced.

*new* is the substring to replace old.

## Return value

This function returns a character string.

## Calculated field example

If a Contact field contained "Mary Smith", the following calculated field expression would return "Mary Smith-Jones":

**Replace(Contact,"Smith","Smith-Jones")**

## Filter/Find example

Records with a value of "Mary Smith-Jones" could be found by the expression:

**Replace(Contact,"Smith","Smith-Jones")="Mary Smith Jones"**

# Right function

## Description

Returns *n* number of characters from the end of a string.

## Syntax

Right(*string*,*n*)

## Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

*n* is the length of the substring you want to extract.

## Return value

This function returns a character string.

## Calculated field example

If the Company field contained the value "Widgets Inc.", the following expression would return "Inc.":

**Right(Company,4)**

**Note:** Right(Company,4) returns the last four characters in the Company field. If the Company field contains trailing spaces, use the RTrim function inside the Right function to first remove the trailing spaces, as in Right(RTrim(Company,4))

### Filter/Find example

Records with a Company field value of "Xerox Corp." or "FMC Corp." can be selected or found using the expression:

**Right(RTrim(Company),5)="Corp."**

## Round function

### Description

Rounds a number to a specified number of places to the right (or left) of the decimal point.

### Syntax

Round(*number*[,*decimals*])

### Arguments

*number* is the value to be rounded. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*decimals* is the position of the digit to round relative to the decimal point. **Note:** Positive numbers round to the right of the decimal point, negative numbers to the left. Zero rounds a number to an integer. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

This function returns a number.

### Calculated field examples

If the Unit\_price field contained the value "29.85", the following expression would return "29.9":

**Round(Unit\_price,1)**

If the Unit\_price field contained the value “1229.93”, the following expression would return “1200”:

**Round(Unit\_price,-2)**

### **Filter/Find example**

Records with a Unit\_price field value of “34.70” or “35.02” can be selected or found using the expression:

**Round(Unit\_price,0)=35**

## **Rowno function**

### **Description**

Returns a record's table row number from the currently displayed table. **Note:** This function differs from the Recno function in that row numbers (Rowno) are re-assigned each time you sort or filter the table while record numbers (Recno) are assigned only when the table is created.

### **Syntax**

Rowno()

### **Arguments**

None.

### **Return value**

This function returns a number.

### **Calculated field example**

The following expression creates a calculated field which displays the row number of each record in the currently displayed table. This is useful if you need to export the row numbers as a field.

**Rowno()**

## Find example

You can jump directly to row 1000 in the table using the following find expression:

**Rowno()=1000**

**Note:** The Rowno function cannot be used in a filter expression.

## RSplit function

### Description

Splits a string into a specified number of parts, then returns a specified substring. The string is searched from right to left for a specified delimiter character. When the delimiter character is found, the portion of the string up to that point is split off into a substring. The search then continues for the next occurrence of the delimiter. After a specified number of substrings have been created, the function returns the specified substring from the right.

### Syntax

`RSplit(string,maxparts,sep,n)`

### Arguments

*string* is the string you want to split apart. The string can be a constant, a character field, an expression that results in a string or a function that returns a string.

*maxparts* represents the maximum number of substrings you want to parse the string into. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*sep* is the character that tells Monarch where to split the string. The delimiter character can be a constant, an expression that results in a single character or a function that returns a single character.

*n* tells Monarch which substring (from the right) you want the function to return. This value can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

### Return value

This function returns a character string.

## Calculated field example

If the Home field contained the value "6rm, ranch, split, attached garage", the following expression would return " attached garage":

**RSplit(Home,2,",",1)**

The first argument (Home) specifies the field. The second argument (2) tells Monarch to split the field into a maximum of 2 substrings. The third argument (",") tells Monarch where to split the field, in this case on the rightmost comma (between "6rm, ranch, split" and " attached garage"). The fourth argument (1) tells Monarch which substring to return, in this case the first substring from the right, i.e. " attached garage".

**Note:** In this example it makes sense to split the field based upon the comma delimiter. You can use any character as the delimiter, but be careful: Monarch does *not* include the delimiter when it breaks out each substring.

## Filter/Find example

Records with a Home field value of "6rm, ranch, split, attached garage" or "7rm, colonial, attached garage" can be selected or found using the expression:

**RSplit(Home,2,",",1)=" attached garage"**

# RTrim function

## Description

Removes all spaces from the end of a string.

## Syntax

RTrim(*string*)

## Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a character string.

### Calculated field example

If the Firstname field contained the value "Santa ", the following expression would return "Santa":

```
RTrim(Firstname)
```

### Filter/Find example

Records with a Firstname field value of "Tom" or "Tom " can be selected or found using the expression:

```
RTrim(Firstname)="Tom"
```

## Second function

### Description

Returns the second (0-59) from a date/time.

### Syntax

```
Second(date)
```

### Arguments

*date* is a date/time field containing a time portion.

### Return value

This function returns a number in the range 0-59.

### Calculated field example

The following calculated field will return "33" from a Date field containing "11/1/2003 12:05:33":

```
Second(Date)
```

## SecondsToChar function

### Description

Converts a number of seconds to a time string.

## Syntax

SecondsToChar(*seconds*)

## Arguments

*seconds* can be a constant, a numeric field, an expression that results in a number or a function that returns a number representing the number of seconds since 12:00 AM.

## Return value

This function returns a string representing the time of day, in the format specified by the Windows Regional Time settings. **Note:** If the number is greater than or equal to the number of seconds in a day (86400), the number is adjusted by dividing by 86400, with only the remainder passed as the argument to the function.

## Calculated field example

To add 45 minutes to each value in the Start\_time field, assuming that this field's values are represented as time strings, such as "12:10pm", you could use the following expression:

**SecondsToChar(CharToSeconds(Start\_time)+(45\*60))**

This expression first converts each Start\_time value to a number of seconds since midnight using the CharToSeconds function. Then 45 minutes are added by adding the appropriate number of seconds to this value. Finally, the value is converted back to a time string.

# Space function

## Description

Returns a string consisting of a specified number of spaces.

## Syntax

Space(*number*)

## Arguments

*number* is the number to convert. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

## Return value

This function returns a character string.

## Calculated field example

If the Product field contains the following classical music recordings, you could use the Space function to create a new field with the composer and the symphony separated into distinct columns.

### Initial field values

"Dvorak, Concerto for Cello"  
"Gershwin, An American in Paris"  
"Mozart, Symphony in D, K.202"

The following expression would create a new field with the composer in one column and the symphony in a second column:

**LSplit(Product,2,",",1)+Space(18-Instr(",",Product))+LSplit(Product,2,",",2)**

The first part of this expression -- LSplit(Product,2,",",1) -- splits off the composer from the Product field. The second part -- +Space(18-Instr(",",Product)) -- concatenates a variable number of spaces, based upon the length of the composer name, to create space between the first and second columns of data. The third part of the expression -- +LSplit(Product,2,",",2) -- concatenates the symphony as the second column of data. The result would be:

### Resulting field values

"Dvorak	Concerto for Cello"
"Gershwin	An American in Paris"
"Mozart	Symphony in D, K.202"

## Sqrt function

### Description

Returns the square root of a number.

### Syntax

Sqrt(*number*)

## Arguments

*number* is the number to convert. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

**Note:** If *number* is a negative value, this function returns 0.

## Return value

This function returns a number.

## Calculated Field Example

If a Squared field contained the number “14400”, the following expression would return “120”:

**Sqrt(Squared)**

# Str function

## Description

Converts a number to a string.

## Syntax

*Str(number[,length[,decimals[,fillchar]])*

## Arguments

*number* is the number to convert. The number can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*length* specifies the length of the resulting string. *length* must be in the range 1-256.

*decimals* specifies the decimal position to round.

*fillchar* specifies the fill character to use to fill up to the length specified.

**Note:** *fillchar* is optional.

## Return value

This function returns a character string.

## Calculated field examples

If the `Unit_price` field contained the value "159.95", the following expression would return " 160":

**Str(Unit\_price,6,0)**

**Note:** If there is no fill character specified, the `Str` function pads the result with leading spaces to match the string length specified. You must ensure that a proper data length is selected for the calculated field.

If the `Unit_price` field contained the value "159.95", the following expression would return "\*\*\*\*160":

**Str(Unit\_price,6,0,"\*")**

## Filter/Find example

Records with a `Unit_price` field value of "99.95" or "100.32" can be selected or found using the expression:

`Str(Unit_price,3,0)="100"`

# Strip function

## Description

Strips the indicated characters from the string.

## Syntax

`Strip(string,stripchars)`

## Arguments

*string* is the string from which you wish to strip characters.

*stripchars* is the list of characters to strip out. **Note:** *stripchars* is case sensitive.

## Return value

This function returns a string without the characters specified in *stripchars*.

### Calculated field example

To strip dashes from a Social Security number, use the following expression:

```
Strip("123-45-6789","-")
```

### Filter/Find example

Records with a normalized telephone number of 9784412200 could be found using:

```
Strip(TelNo,"- ()")="9784412200"
```

## Stuff function

### Description

Returns a string by replacing  $n$  number of characters in string  $s$ , starting at position  $p$ , using replacement string  $c$ .

### Syntax

```
Stuff( $s,p,n,c$ )
```

### Arguments

$s$  is the string you want to insert into,  $p$  is the starting position of the substring,  $n$  is the number of characters you wish to replace in  $s$ , and  $c$  is the substring you want to return.

### Return value

This function returns a string.

### Calculated field example

If a Customer field contained the value "Betty's Music Store", the following expression would change it to "Betty's New Music Store":

```
Stuff(Customer,9,0,"New ")
```

## Substr function

### Description

Extracts a substring of specified length and starting position from a string.

### Syntax

`Substr(string,starting position,length)`

### Arguments

*string* is the string containing the substring. *string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

*starting position* specifies the starting position of the substring. *starting position* can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

*length* specifies the length of the substring. *length* can be a constant, a numeric field, an expression that results in a number or a function that returns a number.

**Note:** If the number of characters specified is larger than the number of characters available, the substring is extracted to the end of the string.

### Return value

This function returns a character string.

### Calculated field example

If the `Prod_code` field contained the value of "PZ-123B", the following expression would return "123":

```
Substr(Prod_code,4,3)
```

### Filter/Find example

Records with a `Prod_code` field value of "TC-789A" or "RB-789C" can be selected or found using the expression:

```
Substr(Prod_code,4,3)="789"
```

## TextLine function

### Description

Splits a multi-line string at the line breaks and returns the specified line.

Where *string* is any string, presumably containing multiple lines, and *n* is a 1-based index indicating which line is to be returned. Such a string will be the result of trapping a multiple line field using one of the Monarch Advanced field properties, End Field On options.

A line break is defined as a LF (i.e., Chr(10)), or a CR/LF combination (i.e., Chr(13)+Chr(10)). These are the most common forms in which line breaks are found in text values.

### Syntax

TextLine(*string*,*n*)

### Arguments

*string* is the multi-line string expression to be split.

*n* is the 1-based index specifying which line to return.

### Return value

This function returns a character string.

**Note:** TextLine returns an empty string if *n* is less than or equal to zero or if *n* is greater than the number of lines in the given string.

### Calculated field example

The following will extract line 2 of a field named Comments, that contains a multi-line string.

**Textline(Comments,2)**

### Filter/Find example

Records that contain no text in the second line of the Comments field can be found using the following expression.

**Textline(Comments,2)=""**

## Time function

### Description

Returns the time portion of a date/time or the current time. Must be formatted as a Time format.

### Syntax

Time(*[date]*)

### Arguments

*date* is a date field with an optional time portion.

### Return value

This function returns the time portion of a date/time or the current time.

### Calculated field example

The following expression will return "12:05:33 AM" from a Date field containing "11/1/2003 12:05:33 AM":

**Time(Date)**

## Today function

### Description

Returns today's date.

### Syntax

Today()

### Arguments

None.

### Return value

This function returns a date in the form YYYYMMDD.

## Calculated field example

If you were reading this on September 17, 2001, the following expression would return the value 20010917 (i.e., today's date in YYYYMMDD format):

**Today()**

## Filter/Find example

Records representing invoices more than thirty days past due can be selected or found using the expression:

**Today()-Due\_date>30**

# Trim function

## Description

Trims all leading and trailing spaces from a string.

## Syntax

Trim(*string*)

## Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a character string.

## Calculated field example

If the Firstname field contained the value " Tristan ", the following calculated field expression would return "Tristan":

**Trim(Firstname)**

## Filter/Find example

Records with a Firstname field value of " Isabella" or " Isabella " can be selected or found using the expression:

**Trim(Firstname)="Isabella"**

## Upper function

### Description

Converts a string to uppercase letters.

### Syntax

Upper(*string*)

### Arguments

*string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

### Return value

This function returns a character string in uppercase letters.

### Calculated field example

If the State field contained the value "Mi", the following expression would return "MI":

**Upper(State)**

### Filter/Find example

All filter expressions are case insensitive, so Upper has no practical application here. For find expressions, use the Match Case setting to determine case sensitivity.

## Val function

### Description

Converts a string to a number. **Note:** The string must begin with a numeric character or a negation sign. If the string contains any non-numeric characters (apart from a decimal point) the function will convert the numeric portion of the string up to the first non-numeric character it encounters.

## Syntax

`Val(string)`

## Arguments

*string* is the string to convert. *string* can be a constant, a character field, an expression that results in a string or a function that returns a string.

## Return value

This function returns a number.

## Calculated field example

If the `Prod_code` field contained the value of "PZ-123B", the following expression would return 123:

`Val(Substr(Prod_code,4,3))`

## Filter/Find example

Records with a `Prod_code` field value of "TC-789A" or "RB-800C" can be selected or found using the expression:

`Val(Substr(Prod_code,4,3))>=789`

**Note:** These examples incorporate the `Substr` function. Use `Substr` to extract a substring from a string.

# Week function

## Description

Returns the number of the week (1-53) from a date.

## Syntax

`Week(date[,startday])`

## Arguments

*date* may be any valid date in the range 01/01/1900 – 12/31/2100.

*startday* is an optional argument that designates the day that begins each week and may be in the range 1 (representing Sunday) to 7 (representing Saturday).

Week #1 of each year is assumed to begin on the first occurrence of the designated day. Any date that falls earlier in the year is assigned to week #0.

**Note:** If *startday* is omitted, the first week of the year (and every week thereafter), is assumed to begin on whichever weekday falls on the first day of that year (whether it be a Sunday, a Monday, a Tuesday, etc). In this case, the Week function returns a week number based on the equivalent formula  $\text{Int}(\text{JulianDate}/7)$ .

## Return value

This function returns a number in the range 0-53.

**Note:** If *startday* is omitted, or if the designated *startday* happens to occur on the first day of the year, that year will return week numbers in the range 1-53. Otherwise, the year will typically return week numbers in the range 0-52. There is one scenario where a year could return the entire range of values (0-53 inclusive); that case being a leap year where *startday* happens to occur on the second day of the year.

## Calculated field example

If you were reading this on September 17, 2001, the following function would return the value 38, indicating that September 17 falls within the 38<sup>th</sup> week of the year 2001:

**Week(Today())**

## Filter/Find example

Records representing shipments during the first four weeks of the year 1999 can be selected or found using the expression:

**Year(Ship\_date)=1999.And.Week(Ship\_date)<=4**

# Weekday function

## Description

Returns the number of the weekday (1-7) from a date.

## Syntax

**Weekday**(*date*[,*startday*])

## Arguments

*date* may be any valid date in the range 01/01/1601-12/31/2400.

*startday* is an optional argument that designates the day that begins each week. This value may be in the range 1 (representing Sunday) to 7 (representing Saturday).

**Note:** If the *startday* argument is omitted, Sunday is assumed to be the first day of the week.

## Return value

This function returns a number in the range 1-7, representing the relative offset starting with the day that has been designated to begin the week (e.g., if weeks begin on Sunday, 1 represents Sunday, 2 represents Monday, etc).

## Calculated field example

If you were reading this on September 07, 2000, the following expression would return the value 5, indicating that today is a Thursday:

**Weekday(Today())**

## Filter/Find example

Records representing long distance telephone calls placed on weekend days can be selected or found using the expression:

**Weekday(Call\_date)=1.Or.Weekday(Call\_date)=7**

# Year function

## Description

Returns the number of year from a date. Valid range is 1601-2400.

## Syntax

Year(*date*)

## Arguments

*date* can be a constant, a date field, an expression that results in a date or a function that returns a date.

## **Return value**

This function returns a number in the range 1601-2400.

## **Calculated field example**

If the Ship\_date field contained the value "19920801", the following expression would return "1992":

**Year(Ship\_date)**

## **Filter/Find example**

Records with a Ship\_date field value of "19940303", "19940601" or "19940217" can be selected or found using the expression:

**Year(Ship\_date)=1994**