Function reference

Ability provides a range of over 200 built-in functions that you can use in formulas.

A built-in function is a fast way to perform often-used calculations (such as totaling a group of numbers) or complicated calculations (like figuring the net present value or the internal rate of return).

The functions in Ability are divided into the following main categories:

- Date functions
- Trigonometric functions
- Statistical functions
- Text functions
- Lookup functions
- Information functions
- Financial functions
- Mathematical functions
- Remote functions
- Document functions
- Logical functions

See also:

- Using built-in functions
- Using the function dialog box to build formulas
- Text arguments
- Arrays
- Arithmetical operators
- Logical operators
- Relational operators
- Priority of evaluation of operators
- Cell error indicators

Using built-in functions

Built-in functions are actually formulas, and must be preceded by an equals sign (=) on the formula bar. You can use the built-in functions to perform date, financial, logical, mathematical, trigonometric, statistical, and other calculations.

You include a built-in function in a formula according to specific rules:

First activate the cell in which you wish the formula result to go, and then type the formula as usual. You can use a built-in function as the entire formula, or as part of one. You must precede the function name with = to show Ability that it is a formula.

You must type the function name exactly; you can't abbreviate the name. The only exceptions are that you can enter AVG for the AVERAGE function, and STD for the STDEV function.

The function name is always followed by an open parenthesis, then the required arguments, and then a close parenthesis. Arguments are the values Ability needs to perform the calculation.

For example, the TOTAL function requires as its argument a list of the values you want to add. The list can be numbers, cell addresses, cell ranges, or cell names. Here's what you might type to find the total of some values in column C of a spreadsheet:

=TOTAL(C1..C15)

If a function doesn't require an argument, you must still enter the parentheses. For example:

=TODAY()

You can use more than one function in a single formula. Use parentheses to group the various functions and control the order of calculation. For example, you can calculate the average of a row of numbers and display the absolute value of the result by using the formula:

=ABS(AVG(A1..A8))

If Ability is unable to perform the calculation you request, you'll see an error indicator displayed in the cell instead of the result of the formula. You can find a list of the error indicators, and an explanation of each, later in this section.

See also:

Using the function dialog box to build formulas

Function reference

Using the formula bar

Using the function dialog box to build formulas

The quick way to add built-in functions to your formulas is by using the "Insert Function" dialog box. This saves you from having to remember all the different function names and the parameters each function requires.

Suppose you want to display the month number of the current date. Follow these steps:

1. Double-click in an empty field to activate the Formula Editor
2. Click on the Insert Function button, or select the Function command from the Insert menu
3. Click on the Date category and then select the Month function. Click on the OK button
4. The cursor will now lie between the brackets inserted with the Month function. Click the Insert Function button again and select the Date category and then select the Today function. The formula should now read MONTH(TODAY()).
5. Click the Confirm button on the formula bar (or press Enter) to finish the formula.

You can call on the "Insert Function" dialog box any time you are editing a formula.

Select the Paste Arguments checkbox before clicking OK, to make Ability insert the formula with text entries for each of the function parameters. You can then replace them with the appropriate cell or range references.

See also:

Using built-in functions

Function reference

Using the formula bar

Text arguments

Some formulas require text as arguments. In such cases, the text can be supplied either directly into the function by surrounding the text by double quotes, or by cell reference, in which case quotes are not required.

Some examples of functions that require text as arguments are:

WPGET(**document**, **fieldname**), LEFT(**text**, num_chars) and UPPER(**text**)

where the text arguments are highlighted in bold.

For example, suppose you wanted to put the sentence "Have a nice day" into uppercase, use one of the following methods:

UPPER("Have a nice day")

UPPER(A1)

where A1 contains the text: **Have a nice day** *without* any quotes.

Text arguments containing quotes

Suppose the text argument itself contains one or more quotes. For example, "Have a "nice" day". The quotes within the text need to be treated as a special case, so that Ability knows where an argument begins and ends. This is done by "escaping" the character, using the caret (^) as follows:

UPPER("Have a ^"nice^" day")

Note that the quotes surrounding the word *nice* are each preceded by the caret. The same thing can be achieved using a cell reference without the need for escaping characters:

UPPER(A1)

where A1 contains the text **Have a "nice" day** and the function will still work as expected.

If your text contains a caret, you need to escape the caret itself, so:

UPPER("The caret is denoted by the ^^ symbol")

returns the text: THE CARET IS DENOTED BY THE ^ SYMBOL.

Arrays

Some formulas require one or more arrays. An array is the same as a list, except that an array may be two-dimensional.

When entering numeric constants directly, rather than by cell reference, into a formula that requires an array as an argument, the array must be bounded by a pair of "braces" {}. For example, the INTERCEPT function (see INTERCEPT) requires two arrays as arguments. These are usually entered as cell ranges:

INTERCEPT(A1..A10, B1..B10)

but can be entered using numbers directly:

INTERCEPT({1, 2, 3, 4}, {1.1, 2.4, 2.9, 4.1})

Note that the arrays themselves are separated by a comma.

Other formulas, for example MDETERM (see MDETERM), where the function performs a calculation on a matrix with an equal number of rows and columns, take as argument a single array. Note that each row in the array is separated by a semi-colon:

MDETERM({2, 5, -3; -1, 10, 1; 2, -3, 1})

You cannot, however, use a range reference as if it were composed of separate ranges or individual cell references. So, for example, the formulas MDETERM({A1..C1; A2..C2; A3..C3}) and MDETERM({A1, B1, C1; A2, B2, C2; A3, B3, C3}) are invalid and return an error message.

Logical operators

There are three logical operators you can use when building logical expressions. These are especially useful when you are using the IF function (see IF).

The following table lists the logical operators in their natural order of evaluation. If you combine either AND or OR with relational operators in an expression, Ability evaluates the relational operators before evaluating the AND or OR.

Operator	**Meaning**
&	AND
|	OR
~	NOT

For example:

Formula	**Returns**
2 * 3 > 7	FALSE
(2 * 3 > 7) | (4 * 1 = 4)	TRUE
(2 * 3 = 6) & ~(4 - (4 * 1))	TRUE
IF (cond1 & cond2 & cond3, truevalue, falsevalue)	If all conditions are met, truevalue

Using the logical operators is equivalent to using the logical functions as follows:

Formula	**Function Equivalent**
(2 * 3 > 7) | (4 * 1 = 4)	OR((2 * 3 > 7), (4 * 1 = 4))
(2 * 3 = 6) & ~(4 - 4 * 1)	AND((2 * 3 = 6), NOT(4 - 4 * 1))
IF (cond1 & cond2 & cond3, truevalue, falsevalue)	IF(AND(cond1, cond2, cond3), truevalue, falsevalue)

See also:

Logical functions

Arithmetical operators

Relational operators

Priority of evaluation of operators

Function reference

Relational operators

You can use the following relational operators when building logical expressions.

Operator	Meaning
<	Less than
>	Greater than
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
<> or ~=	Not equal to.

These are especially useful in combination with the IF(x, true, false) function (see IF(x, true, false)).

If you combine relational operators with arithmetical operators in an expression, Ability performs the arithmetical operations before evaluating the relational operators.

See also:

Logical operators

Arithmetical operators

Priority of evaluation of operators

Function reference

Arithmetical operators

The following standard arithmetical operators are available in Ability. If arithmetical operators are used in conjunction with relational operators in an expression, then the arithmetical operations are calculated before the relational operators are evaluated.

Operator	**Meaning**
+	add
-	subtract
*	multiply
/	divide
^	power e.g. 2 ^ 3 = 2 * 2 * 2
%	percentage e.g. 15% = 0.15
(..)	bracketed expressions e.g. (2 * 8 - 17)

Note that the arithmetical operators are evaluated in the following order in an expression:

(..)
%
^
* and /
+ and -

For example, the expression:

1 + 2 * 3 / 4 ^ 5 % - 6

is calculated as if it were the bracketed expression:

1 + ((2 * 3) / (4 ^ (5%))) - 6

Ability will automatically strip excess brackets from a formula to display it in its simplest form.

See also:

Logical operators

Relational operators

Priority of evaluation of operators

Function reference

Priority of evaluation of operators

The functions and operators in Ability are evaluated in an order of priority. For complex expressions you will not have to worry about this too much, since it is likely that you will have used brackets to control and make clear the way you want the expressions to be calculated. The priorities are as follows, starting with the highest:

Operator / Function	**Meaning**
Built-in functions	Ability's built-in functions
(..)	bracketed expressions
%	percentage
^	power
* and /	multiply and divide
+ and -	add and subtract
<, <=, >, >=, =, <>	relational operators
~	NOT
&	AND
|	OR

See also:

Logical operators

Relational operators

Arithmetical operators

Function reference

Cell error indicators

If Ability cannot perform the calculation you request in a formula, it displays an error indicator in the cell instead of the result of the calculation.

There are five different error messages:

Error	Explanation
#VALUE	Formula cannot be evaluated. For example, the type of an argument does not match, as in SQRT(-16), or an argument is out of range, as in RIGHT("New York", -5).
#FUNC	Wrong number of function arguments or improper function argument. For example, SIN(A1, A2, A3) or COS(A1..B10).
#DIV0	The formula attempts to divide by zero.
#REF	Bad reference argument in formula. For example, SUM(Sales), when there is no range called Sales.
#CIRC	The formula makes a circular link. For example, when the formula =A1 + 5 is entered in A1.

See also:

Function reference

Date functions

Ability allows you to perform arithmetic with dates and times. To do this, it has a built-in numbering sequence, called a date/time code, that starts on 1st Jan 1900 and increases by one for each subsequent day.

The date time code has a fractional component as well, to correspond to the hour, minute and second since midnight. A date/time code of 1.5 represents one-and-a-half days since the start of the year 1900, that is 12:00 noon on the 2nd Jan 1900.

For example:

Date/time code	35484.47917	24th Feb 1997 11:30:00 AM
Date code	35484	24th Feb 1997 i.e. 35484 days since 1/1/1900
Time code	0.47917	11:30 AM i.e. 0.47917 * 24 hours since midnight

A date code, then, is the integer part of a date/time code, and a time code is the fractional remainder.

You can use Ability's built-in date functions to generate a date/time code or to translate a date code into the weekday, day, month or year, and a time code into hours, minutes and seconds.

To change how a date/time code is displayed, use the **Number** command from the **Format** menu and then select a date format.

Here is a complete list of date functions in Ability:

DATE	DATEVALUE
DAY	DAYS360
EDATE	EOMONTH
HOUR	MINUTE
MONTH	NETWORKDAYS
NOW	SECOND
TIME	TIMEVALUE
TODAY	WEEKDAY
WEEKNUM	WORKDAY
YEAR	YEARFRAC

See also:

Year 2000 for details on how Ability deals with dates in the 21st Century.

Other functions

Year 2000

The short answer is that the change of millennium won't cause any problems for Ability.

There are, however, two issues of which you should be aware:

1. Date format - in a spreadsheet showing both 20th and 21st Century dates, you should format all dates to display four-digit years to avoid confusion (see Formattting a date for more details).
2. Date entry - you can enter dates using either two-digit years or four-digit years. Obviously with four-digit years, there's no ambiguity in century. Most people will be used to entering two-digit years, in which case Ability will assume you mean the 20th Century for all years later than and including 1972 and up to 1999, and the 21st Century for all other years entered as two digit. For example:

Enter date as…	**Ability understands you to mean…**
1/1/99	1/1/1999
1/1/00	1/1/2000
1/1/71	1/1/2071
1/1/72	1/1/1972

DATE(year, month, day)

The DATE function calculates a date code for the given **year**, **month** and **day**. For example, the formula:

DATE(97, 8, 8)

produces the date code 35649 or 8th August 1997, depending on your number format.

See also:

Other date functions

DATEVALUE(date_text)

The DATEVALUE function converts a date in the form of text to a date code. For example, the formula:

DATEVALUE("8-Aug-1997")

returns the date code 35649, as does

DATEVALUE("8/8/97")

If the year is left out, the date code will refer to the current year.

See also:

Other date functions

DAY(date-code)

The DAY function calculates the day of the month of the date-code you enter. For example, the formula

    DAY(35649)

produces 8 as the day of the month, since the date code refers to 8th August 1997.

See also:

    Other date functions

DAYS360(start_date, end_date, method)

The DAYS360 function calculates the number of days between a **start_date** and an **end_date**, based on a 12-month year of 30 days each. This is useful when accounts are calculated on the assumption of a 30-day month. The dates can be entered in either text or date code format.

method is either European (TRUE or omitted) or American (FALSE). If TRUE or omitted then start-dates or end-dates that refer to the 31st day of a month become 30; if FALSE then a start-date of 31 becomes 30, but an end-date of 31 becomes the 1st of the next month when the start-date is less than 30, otherwise the end-date becomes 30.

For example, the formula

DAYS360("25/1/97", "31/7/97", TRUE)

returns 185, whereas

DAYS360("25/1/97", "31/7/97", FALSE)

returns 186.

See also:

Other date functions

EDATE(start_date, months)

The EDATE function calculates the date from a **start_date** plus or minus the number of **months** indicated. For example, the formula

EDATE("12 December, 1997", -3)

returns the date "12 September, 1997".

See also:

Other date functions

EOMONTH(start_date, months)

The EOMONTH function calculates the end of the month from a **start_date** plus or minus the number of **months** indicated. For example, the formula

EOMONTH("4/4/96", 5)

returns the date "30 September, 1996".

See also:

Other date functions

HOUR(time_code)

The HOUR function converts a **time_code** to an hour. The hour is calculated using the 24-hour clock as a basis, running from 0 at 12 PM (or 0:00 hours) to 23 at 11 PM (or 23:00 hours). 24-hour cycles correspond to a time code increment of 1. The argument for HOUR can be in either text or time code format.

Examples:

Formula	**Returns**
HOUR(0.3)	7
HOUR(1.5)	12
HOUR(455.5)	21
HOUR("9:30 PM")	21
HOUR("21:30:45")	21.

See also:

Other date functions

MINUTE(time_code)

The MINUTE function converts a **time_code** to a minute. The minute is calculated using 1 hour as a basis, running from 0 to 59. A time code increment of 1 corresponds to 1 day (or 1,440 minutes). The argument for MINUTE can be in either text or time code format.

The following examples all returns 12, that is the 12th minute after the 13th hour of the day.

Formula	**Returns**
MINUTE(0.55)	12
MINUTE(1.55)	12
MINUTE("13:12")	12

See also:

Other date functions

MONTH(date-code)

The MONTH function calculates the month from the **date_code** you enter.

For example,

MONTH(31062)

returns month 1, since the date referred to by the date code is 16th January 1985.

See also:

Other date functions

NETWORKDAYS(start_date, end_date, list_holidays)

The NETWORKDAYS function calculates the number of whole work days between 2 dates, **start_date** and **end_date**, taking into account weekends and holidays.

For example, the formula

> **NETWORKDAYS ("1 Jan 1997", "31 Dec 1997")**

returns 261, that is a year's worth of workdays, taking into account weekends.

> **NETWORKDAYS ("1 Jan 97", "31 Jan 97", "14 Jan 97", "15 Jan 97")**

returns 21, that is the number of workdays in January with two days holiday.

See also:

> Other date functions

NOW( )

The NOW function calculates the current date and time from the date-time code as set by your computer's built-in clock.

For example:

NOW()

would return "July/2/1997 12:23 PM" if that was in fact the current time and date.

If you choose to display the date-time code as a number, then the numbers before the decimal point represent the date, and the numbers after the decimal point represent the time.

The time and date are taken directly from your computer's operating system (e.g. Windows 95) - if they appear incorrect, you need to adjust the clock on your PC.

See also:

Other date functions

SECOND(time_code)

The SECOND function calculates the number of seconds from 0 to 59 from a **time_code**. The function can take as argument either a time code or text.

For example, the formula

SECOND("22:14:10")

returns 10, the number of seconds into the 14th minute of the 22nd hour.

SECOND(0.01)

returns 24, as a 100th of a day is 14 minutes and 24 seconds exactly.

See also:

Other date functions

TIME(hour, minute, second)

The TIME function calculates the time code from a supplied **hour**, **minute** and **second**. This enables you to display the time in a format of your choice. The arguments for TIME are entered according to the 24-hour clock.

For example, the formula

TIME(22,20,40)

returns a time code of 0.93101852 and is equivalent 10:20:40 PM, which is the displayed default format.

Note that the formulas TIME(0, 0, 0) and TIME(24, 0, 0) return time codes of 0 and 1 respectively.

See also:

Other date functions

TIMEVALUE(time_text)

The TIMEVALUE function calculates the time code based on a time represented as text. The time code is a decimal fraction ranging from 0 to 0.99999 (0:00:00 to 23:59:59).

For example, the formula

TIMEVALUE("20:15:30")

returns the time code 0.84409722 when formatted as a plain number.

See also:

Other date functions

TODAY( )

The TODAY function returns today's date. It actually calculates a date code for the current day and, by default, this is then formatted and displayed as a recognizable date.

For example, the formula

    TODAY()

returns "July/2/1997" on that date.

The date is taken directly from the operating system (e.g. Windows 98) - if it appears incorrect, you need to adjust the clock on your PC.

See also:

Other date functions

WEEKDAY(date_code)

The WEEKDAY function calculates the day of the week for the supplied **date_code**. Days of the week are numbered sequentially from Sunday (1) through to Saturday (7).

For example, the formula

    WEEKDAY(35612)

returns 4, as 2nd July 1997 occurs on a Wednesday. The WEEKDAY function can be useful in conjunction with the TODAY function, for example:

    WEEKDAY(TODAY())

returns the current day of the week.

See also:

Other date functions

WEEKNUM(date_code, return_type)

The WEEKNUM function uses **date_code** to calculate the week number in the year. **return_type** is a number that tells the function on what day of the week the week begins: 1 for Sunday; 2 for Monday. If this is omitted WEEKNUM uses 1.

For example, if you want to know the week number for 1st November 1997, assuming the week begins on Monday, use the formula

WEEKNUM("1 Nov 97", 2)

which returns 44 (the 44th week of the year).

See also:

Other date functions

WORKDAY(start_date, days, list_holidays)

The WORKDAY function returns a date that is **start_date** plus or minus **days** - a specified number of workdays, taking into account weekends and holidays given by **list_holidays**.

For example, the formula

WORKDAY("February/1/97", 100, "March/28/97")

returns the date "June/23/1997", a date 100 workdays after 1st Feb 97, given a holiday on 28th March.

See also:

Other date functions

YEAR(date-code)

The YEAR function returns the year of the date code you enter.

For example, the formula

YEAR(31062)

returns the year 1985. At the time this text was written, the following example returns 1997:

YEAR(TODAY())

See also:

TODAY( )

Other date functions

YEARFRAC(start_date, end_date, basis)

The YEARFRAC function calculates the year fraction representing the number of whole days between **start_date** and **end_date**.

basis is the type of day count basis to be used, as defined by the following table:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, if a period has start date, 5 August 1997, and end date, 1 November 1997, with a day count basis of 30/360, the formula

YEARFRAC(35646, 35734, 4)

returns a year fraction for the period of 0.2389 or 23.89%.

See also:

Other date functions

Financial functions

Here is a complete list of financial functions in Ability. You can use the built-in financial functions to calculate compound amounts and net present values of investments, and to perform other forecasting and analytical calculations.

- ACCRINTM
- COMPOUND
- COUPDAYBS
- COUPDAYS
- COUPDAYSNC
- COUPNCD
- COUPNUM
- COUPPCD
- DB
- DISC
- DOLLARDE
- EFFECT
- FV
- INTRATE
- IRR
- MIRR
- NOMINAL
- NPV
- PMT
- PRICE
- PRICEDISC
- PRICEMAT
- PV
- RECEIVED
- TAXBAND
- TBILLPRICE
- TBILLYIELD

See also:

Other functions

ACCRINTM(issue, settlement, rate, par, basis)

The ACCRINTM function calculates the accrued interest for a security that pays interest at maturity. The function arguments are:

issue	is the issue date of the security, expressed as a date code
settlement	is the maturity date of the security, expressed as a date code
rate	is the annual coupon (interest) rate of the security
par	is the par or base value of the security (if par is omitted, ACCRINTM uses $1000)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

The function is calculated using the formula

ACCRINTM = par * rate * (A / D)

where **A** is the number of days accrued, counted according to a monthly basis, and **D** is the annual year basis.

ACCRINTM calculates non-compound (or simple) interest over the security's period.

For example, you have been issued with a coupon worth $10,000 on July 1st 1997 that comes to maturity on September 30th 1997; the coupon's annual interest rate is 12%; and you want to know what the interest will be on September 30th when the coupon achieves maturity. Use the formula:

ACCRINTM(35611, 35702, 0.12, 10000, 3)

to get an interest payment of $299.18.

See also:

Other financial functions

COMPOUND (principal, interest, periods)

The COMPOUND function calculates the compound amount, based on the **principal** and **interest** rate per period over the specified number of **periods**. The formula used is:

$$\text{principle} * (1 + \text{interest})^{\text{periods}}$$

For example, to find out the compound amount on a principal of $500, at an annual interest rate of 13.5%, over 12 years, use the formula:

COMPOUND(500,13.5%,12)

Ability calculates the compound amount and displays the result 2285.1796 or $2,285.18 (depending on the currency formatting of the cell – use the Number command from the Format menu to change it).

See also:

Other financial functions

COUPDAYBS(settlement, maturity, frequency, basis)

The COUPDAYBS function calculates the number of days from the beginning of the coupon period to the settlement date. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPDAYBS(35625, 36037, 4, 1)

returns 45 as the number of days from the beginning of the coupon period to the settlement date.

See also:

Other financial functions

COUPDAYS(settlement, maturity, frequency, basis)

The COUPDAYS function calculates the number of days in the coupon period in which the settlement date occurs. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPDAYS(35625, 36037, 4, 1)

returns 92 as the number of days in the coupon period in which the settlement date occurs.

See also:

Other financial functions

COUPDAYSNC(settlement, maturity, frequency, basis)

The COUPDAYSNC function calculates the number of days from the settlement date to the next coupon date. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPDAYSNC(35625, 36037, 4, 1)

returns 47 as the number of days from the settlement date to the next coupon date.

See also:

Other financial functions

COUPNCD(settlement, maturity, frequency, basis)

The COUPNCD function calculates the next coupon date after the settlement date. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPNCD(35625, 36037, 4, 1)

returns August 31st 1997 or 35672 as the next coupon date after the settlement date.

See also:

Other financial functions

COUPNUM(settlement, maturity, frequency, basis)

The COUPNUM function calculates the number of coupons payable between the settlement date and the maturity date. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPNUM(35625, 36037, 4, 1)

returns 5 as the number of coupons payable between the settlement date and the maturity date.

See also:

Other financial functions

COUPPCD(settlement, maturity, frequency, basis)

The COUPPCD function calculates the previous coupon date before the settlement date. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis to use, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the coupon payments are quarterly, and the day count basis is actual/actual. Using the formula

COUPPCD(35625, 36037, 4, 1)

returns May 31st 1997 or 35580 as the coupon date before the settlement date.

See also:

Other financial functions

DB(cost, salvage, life, period, month)

The DB function calculates the depreciation of an asset for a specified period using the fixed-declining balance method. DB works out a fixed rate which it then uses to calculate depreciation. The function arguments are:

cost	is the initial value of the asset
salvage	is the value at the end of the depreciation
life	is the number of years over which the asset depreciates
period	is the specified period for which the depreciation is sought
month	is the number of months between the purchase date and the end of the first period (12 if omitted)

For all periods except the first and last, DB calculates the depreciation per period using the equation:

(cost - total depreciation from previous periods) * rate

where rate is given by:

$$\text{rate} = 1 - (\text{salvage}/\text{cost})^{1/\text{life}}$$

For depreciation over the first period DB uses the equation:

cost * rate * (month/12)

For depreciation over the last period DB uses the equation:

[(cost - total depreciation from previous periods) * rate * (12 - month)] / 12

For example, a new yacht is purchased for $1,000,000. Its active life is 20 years and its salvage value at the end of this time is $50,000. The purchase was made at the end of May and the buyer, who attends a boat show at the end of every October, wishes to know its depreciation over the periods between boat shows for the first 4 shows after the initial purchase. The first period is of 5 months (from May to October) and the next three periods are whole years. The depreciation for the first 4 periods is calculated using these formulas:

DB(1000000, 50000, 20, 1, 5) returns a depreciation of $57,917

DB(1000000, 50000, 20, 2, 5) returns a depreciation of $130,950

DB(1000000, 50000, 20, 3, 5) returns a depreciation of $112,748

DB(1000000, 50000, 20, 4, 5) returns a depreciation of $97,076

Note: A 20-year span contains at most 21 periods, and therefore setting period to greater than 21 will result in DB returning a #VALUE error message.

See also:

Other financial functions

DISC(settlement, maturity, par, redemption, basis)

The DISC function calculates the discount rate for a security. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
par	is the price of the security per $100 face value
redemption	is the amount received at maturity per $100 face value
basis	is the type of day count basis to use, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

DISC is calculated using the formula:

[(redemption - par)/redemption] * B/DSM

where **B** is the number of days in a year according to the **basis** used and **DSM** is the number of days between settlement and maturity.

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the price of the security is $95 per $100 face value, the redemption value is $100, and the day count basis is actual/actual. Using the formula

DISC(35625, 36037, 95, 100, 1)

returns a bond discount rate of 4.429 %.

See also:

Other financial functions

DOLLARDE(fractional_dollar, fraction)

The DOLLARDE function converts a dollar price expressed as a **fraction**, into a dollar price expressed as a decimal number.

For example, if a dollar value is expressed in 1/16ths of a dollar, the value 7/16 is entered as 0.07 in **fractional_dollar** and the integer 16 is entered in fraction, the formula:

DOLLARDE(0.07, 16)

returns 0.438. If the fraction is greater than 1/10th only one place is needed after the point.

For example, 3/8ths of a dollar is entered using the formula:

DOLLARDE(0.3, 8)

which returns 0.375.

See also:

Other financial functions

EFFECT(nominal_rate, npery)

The EFFECT function calculates the effective annual interest rate. This uses the given nominal annual interest rate, **nominal_rate**, and the number of periods per year where compound interest is applied, **npery**. EFFECT uses the formula:

$$(1 + (nominal\_rate / npery))^{npery} - 1$$

For example, the formula:

EFFECT(7 %, 4)

returns an effective annual interest rate of 7.19% or 0.0719, which is the overall interest rate over a year when the interest has been calculated quarterly at 7/4% and compounded.

EFFECT is the inverse of NOMINAL (see NOMINAL).

See also:

Other financial functions

FV(payment, rate, periods)

The FV function calculates the future value of an annuity, based on the payments per period, interest rate per period, and the number of periods for which you want the value calculated. The formula used is:

$$\text{payment} * \left((1 + \text{rate})^{\text{periods}} - 1 \right) / \text{rate}$$

Example 1: The future value of an annuity with annual $500 payments at an interest rate of 13.5% per annum, compounded over five years, is calculated using the formula:

FV(500, 13.5%, 5)

Ability calculates the future value and displays 3272.44 or $3,272.44 (depending on the currency formatting of the cell – use the Number command from the Format menu to change it).

Example 2: Each month you deposit $50 at a bank. Interest accrues at an annual rate of 13.5% but is applied monthly. To calculate the value of the account after six months, use FV in conjunction with NOMINAL (see NOMINAL) as follows:

FV(50, NOMINAL(13.5%, 12)/12, 6)

This returns a value of $308.07.

See also:

Other financial functions

INTRATE(settlement, maturity, investment, redemption, basis)

The INTRATE function calculates the interest rate for a fully invested security. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
investment	is the amount invested in the security
redemption	is the amount received at maturity
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

INTRATE is calculated using the formula:

[(redemption - investment)/investment] * B/DIM

where **B** is the number of days in a year according to the basis used and **DIM** is the number of days from settlement to maturity.

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement date is July 15th 1997, the amount invested in the security is $10,000, the redemption value is $10,100, and the day count basis is actual/actual. Using the formula

INTRATE(35625, 36037, 10000, 10450, 1)

returns an interest rate of 3.98665% for a fully invested security.

See also:

Other financial functions

IRR(guess, initial, list)

The IRR function calculates the internal rate of return of a series of cash flows, starting with a **guess** at the correct answer. The internal rate of return is the effective interest rate such that the net present value of the cash flow is zero. This function is the inverse of the net present value function.

You should enter the **initial** cash flow, initial, as a negative number to indicate that it is money received, rather than money paid out.

For example, to find the internal rate of return of a series of cash flows that start with a negative flow of $800, followed by payments of $400, $400, $200, and $100 at even intervals, use the following formula:

IRR(0.15, -800, {400, 400, 200, 100})

Ability calculates the interest rate per period and displays 0.18 or 17.9652% (depending on the percent formatting of the cell – use the Number command from the Format menu to change it).

In this example, the result shows that if you borrow $800 and make annual payments of $400, $400, $200 and $100, the underlying interest rate charged to you would be 17.9652%.

See also:

Other financial functions

MIRR(finance_rate, reinvest_rate, list)

The MIRR function calculates a modified internal rate of return where positive cashflows earn interest at a **reinvest_rate** and negative cash flows are financed at a **finance_rate**. The cashflows are contained in **list** and will usually begin with a negative figure, indicating that you start with a loan.

MIRR is calculated as follows:

$$\left[\frac{\sum(+\text{ve cashflows}) + \text{net interest}}{\sum(-\text{ve cashflows})}\right]^{\frac{1}{n-1}} - 1$$

where **net interest** is the return on the reinvested positive cash flows, less the interest accrued on the negative cash flows.

For example, suppose you were loaned $1000.00 to invest in a business venture and you pay interest on this loan at 15%. You predict that at the end of each of the first four years the business will generate the following cash flow: $500, $700, $700, $900. Further suppose that you reinvest the earnings at a fixed rate of 10% and this will compound over the four years.

At the end of the first year, you’ll receive $500, which will be reinvested but you’ll also have to take into account the interest on the loan, which will be $150 at this time. Over the years, net interest will accumulate as follows:

Period (n)	Cash flow	Interest1	Interest2	Interest3	Interest4	Σ Interest
1	-1000	-150.00	-172.50	-198.36	-228.13	-748.99
2	500		50.00	55.00	60.50	165.50
3	700			70.00	77.00	147.00
4	700				70.00	70.00
5	900					0.00
						-366.49

After four years, the original $1000 has generated earnings of 500+700+700+900=$2800 less net interest of $366.49 = $2433.51. This can be expressed as a rate of return, over four years, of 143.35% or [(1+ 143.35%)^(1/4) -1] = 24.9% per annum.

The same result can be obtained using the MIRR functions as follows:

MIRR(15%, 10%, -1000, 500, 700, 700, 900)

which Ability calculates as 24.90%.

See also:

Other financial functions

NOMINAL(effect_rate, npery)

The NOMINAL function calculates the annual nominal interest rate. This uses the given effective annual interest rate, **effect_rate**, and the number of periods per year where compound interest is applied, **npery**. NOMINAL uses the formula:

$$npery * \left((effect\_rate + 1)^{1/npery} - 1\right)$$

For example, the bank quotes an annual interest of 7.19% and calculates interest quarterly, which is then compounded over the year. What is actual interest rate in a given quarter? Use the formula:

NOMINAL(7.19%, 4)/4

which returns a rate of 1.75%.

NOMINAL is the inverse of EFFECT (see EFFECT).

See also:

Other financial functions

NPV(rate, list)

The NPV function calculates the net present value of a set of future cash flows. Ability assumes that the first cash flow occurs at the end of the first period, which means the initial payment is usually not included. (Compare this with the IRR function – see IRR)

Ability calculates the net present value according to the following formula:

$$\frac{\text{list}_1}{1+\text{rate}}+\frac{\text{list}_2}{(1+\text{rate})^2}+\frac{\text{list}_3}{(1+\text{rate})^3}+\ldots+\frac{\text{list}_n}{(1+\text{rate})^n}$$

n is the number of items in the list and list*i* is the *i*th element in the list.

In the function, enter **rate** as the effective interest rate per period. Enter **list** as the value of the payments in the order in which they appear, one payment per period. If you enter list as a range that includes several rows and columns, Ability reads the range from left to right and top to bottom. If blank cells are in the range, Ability skips them when calculating the net present value.

For example, to find the net present value of cash flows of $400, $300, $300, and $200 and assuming inflation will run at 13.25%, type the following formula:

NPV(13.25%,400,300,300,200)

Ability calculates the net present value and displays 915.23 or $915.23 (depending on the currency formatting of the cell – use the Number command from the Format menu to change it). This shows that the real value of these future payments is much less than the sum of the payments after inflation has eroded the cash flow.

You can also use NPV with the COMPOUND function (see COMPOUND) to calculate the end value of a series of payments, reinvested over a period of time. Suppose you know that you'll receive payments of $400, $300, $300, and $200 over the next four years, and that you'll reinvest these payments in an account bearing interest of 12% per annum, compounded over the four years. After the four years, how much will be in the account? This is given using the following formula:

COMPOUND(NPV(13.25%, 400, 300, 300, 200), 13.25%, 4)

Ability calculates the answer as $1505.51.

See also:

Other financial functions

PMT(principal, rate, periods)

The PMT function calculates the amount of payment required per period to pay off a loan or mortgage at the given **principal**, interest **rate** per period, and number of **periods**. Ability calculates the payment based on the following formula:

$$\frac{\text{principle} \; * \text{rate}}{1 - (1 + \text{rate})^{-\text{periods}}}$$

Often the payment amounts end up as fractional numbers. In such cases, you can include the ROUND function (see ROUND) to calculate the payments as even amounts.

To find the payments required to pay off a $17,000 loan, at 14.25% interest per annum, over a period of 3 years, use the following formula:

PMT(17000, 14.25%, 3)

Ability calculates the payments and displays 7353.17 or $7,353.17 per year (depending on the currency formatting of the cell – use the Number command from the Format menu to change it).

To find the payments required per period when the interest is compounded monthly, use PMT in conjunction with EFFECT (see EFFECT) as follows:

PMT(17000, EFFECT(14.25%, 12) / 12, 36)

Ability calculates the monthly payments at $591.13.

See also:

Other financial functions

PRICE(settlement, maturity, rate, yield, redemption, frequency, basis)

The PRICE function calculates the price per $100 face value of a security that pays periodic interest. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
rate	is the annual coupon rate of the security
yield	is the annual yield of the security
redemption	is the redemption value of the security per $100 face value
frequency	is the number of coupon payments per year (1 = annually; 2 = biannually; 4 = quarterly)
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

For example, if a bond has settlement date 15/July/1992, maturity date 15/September/1999, annual coupon rate of 6.50%, annual yield of 7.25%, redemption value of $100, two coupon payments per year, and is calculated according to a year basis of US 30/360, then use the formula

PRICE(33799, 36417, 6.50%, 7.25%, 100, 2, 0).

This returns a bond price of $95.852.

See also:

Other financial functions

PRICEDISC(settlement, maturity, discount, redemption, basis)

The PRICEDISC function calculates the price per $100 face value of a discounted security. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
discount	is the discount rate of the security
redemption	is the redemption value of the security per $100 face value
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

The formula used is

redemption - (discount * redemption * DSM/B)

in which **B** is the number of days in a year according to the year basis used and **DSM** is the number of days from settlement to maturity.

For example, if a bond has settlement date 15th July, 1997, maturity date 31st October, 1997, discount rate of 4.9%, and redemption value $100, with a year basis of actual/365, then the formula

PRICEDISC(35625, 35733, 0.049, 100, 3)

returns a price of $98.55.

See also:

Other financial functions

PRICEMAT(settlement, maturity, issue, rate, yield, basis)

The PRICEMAT function calculates the price per $100 face value of a security that pays interest at maturity. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
issue	is the issue date of the security, expressed as a date code
rate	is the interest rate of the security at date of issue
yield	is the annual yield of the security
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

The formula used is

[100 + (DIM/B * rate * 100)] / [1 + (DSM/B * yield)]   - (A/B * rate * 100)

where

B	is the number of days in a year according to year basis used
DSM	is the number of days from settlement to maturity
DIM	is the number of days from issue to maturity
A	is the number of days from issue to settlement.

For example, if a bond has settlement date 15th July, 1997, maturity date 31st October, 1997, and issue date 1st October, 1996, with an interest rate of 4.9% at date of issue and an annual yield of 5.4%, and is calculated according to a 30/360 year basis, then the formula

PRICEMAT(35625, 35733, 35338, 0.049, 0.054, 0)

returns a price of $99.79.

See also:

Other financial functions

PV(payment, rate, periods)

The PV function calculates the present value of an annuity, based on the amount of the **payment**, the interest **rate** per period, and the number of **periods**. Ability computes present value using the following formula:

$$\frac{\text{payment} * \left(1 - (1 + \text{rate})^{-\text{periods}}\right)}{\text{rate}}$$

For example, to find the present value of an annuity with payments of $400, at 12.5% interest per period, after 4 periods, type:

PV(400,12.5%,4)

Ability calculates the present value and displays 1202.26 or $1,202.26 (depending on the currency formatting of the cell – use the Number command from the Format menu to change it).

See also:

Other financial functions

RECEIVED(settlement, maturity, investment, discount, basis)

The RECEIVED function calculates the amount received at maturity for a fully invested security. The function arguments are:

settlement	is the settlement date of the security, expressed as a date code
maturity	is the maturity date of the security, expressed as a date code
investment	is the amount invested in the security
discount	is the discount rate of the security
basis	is the type of day count basis used, where basis is one of the following:

Basis	**Day count basis**
0	US 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4 or omitted	European 30/360

RECEIVED is calculated using the formula:

investment / [1 - (discount * DIM/B)]

Where **B** is the number of days in a year according to the basis used and **DIM** is the number of days from issue to maturity.

For example, you have been issued with a bond that comes to maturity on August 31st 1998, the settlement (issue) date is July 15th 1997, the amount invested in the security is $10,000, the discount rate is 4.2%, and the day count basis is actual/actual. Using the formula

RECEIVED(35625, 36037, 10000, 0.042, 1)

returns a total amount received at maturity of $10,497.68.

See also:

Other financial functions

TAXBAND(income, taxrates, taxbands)

The TAXBAND function calculates the total tax payable on a given **income** subject to a series of banded tax rates. **taxrates** and **taxbands** are both lists and must be of equal length.

The function is calculated summing the tax rate and band product over the number of items in each list:

$$\sum_{i=1}^{n} \left[\mathrm{MAX}\left(\mathrm{MIN}\left(\mathrm{income} - \mathrm{taxband}_i, \mathrm{taxband}_{i+1} - \mathrm{taxband}_i\right), 0\right) * \mathrm{taxrate}_i\right]$$

where **n** is the number of items in each list and taxband[n+1] is defined as being equal to income.

Example 1: suppose you earned £28,800 in the tax year ending 1996 and had a tax-free personal allowance of £3,525. The tax rates for that year were 20% on the first £3,200 of taxable income, 25% up to £24,300 and 40% on the remainder. You would use the TAXBAND function to calculate tax due as follows:

TAXBAND(28800, {20%, 25%, 40%}, {3525, 6725, 27825}) = 6305

Note that the personal allowance needs to be added back onto the taxbands in this example because the supplied figures are net of this amount.

Example 2: suppose you want to subtract tax-free earnings before calculating tax due. You can use TAXBAND as follows:

TAXBAND(28800-3525, {20%, 25%, 40%}, {0, 3200, 24300}) = 6305

Some governments publish tax bands in this format.

Example 3: suppose a proportion of your income of £28,800 was taxed at source to contribute towards health care. The tax was at a flat rate of 7.3% on all income over £6,640 up to a ceiling of £22,880. The tax can be calculated as follows:

TAXBAND(28800, {7.3%, 0%}, {6640, 22880}) = 1185.52

See also:

Other financial functions

TBILLPRICE(settlement, maturity, discount)

The TBILLPRICE function calculates the price per $100 face value for a Treasury bill. The function arguments are:

settlement	is the settlement date of the Treasury bill, expressed as a date code
maturity	is the maturity date of the Treasury bill, expressed as a date code
discount	is the discount rate of the Treasury bill

TBILLPRICE is calculated using the formula:

100 * [1 - (discount * DSM/360)]

where DSM is the number of days from the settlement to the maturity date.

The maturity date should not be more than 1 calendar year after the settlement date, otherwise an error message is returned.

For example, a Treasury bill has settlement date 10th May, 1997 and maturity date 1st November, 1997. The discount rate is 7%. The formula

TBILLPRICE(35559, 35734, 0.07)

returns a price of $96.60.

See also:

Other financial functions

TBILLYIELD(settlement, maturity, par)

The TBILLYIELD function calculates the yield for a Treasury bill. The yield is a measure of the value per dollar spent when the bill reaches maturity, expressed as an annual interest rate. The function arguments are:

settlement	is the settlement date of the Treasury bill, expressed as a date code
maturity	is the maturity date of the Treasury bill, expressed as a date code
par	is the price per $100 face value of the Treasury bill

TBILLYIELD is calculated using the formula:

[(100 - par)/par] * 360/DSM]

where **DSM** is the number of days from the settlement to the maturity date.

The maturity date should not be more than 1 calendar year after the settlement date, otherwise an error message is returned.

For example, a Treasury bill has settlement date 10th May, 1997 and maturity date 1st November, 1997. The price per $100 of the Treasury bill is $95.75. The formula

TBILLYIELD(35559, 35734, 95.75)

returns a yield of 9.13%.

See also:

Other financial functions

Logical functions

You can use the built-in logical functions to test for specific conditions in a calculation.

Logical functions generally either return an answer of TRUE or FALSE or are based on a condition that evaluates to TRUE or FALSE. The value FALSE equates to zero and TRUE as 1 (or anything non-zero).

Here is a list of logical functions in Ability:

AND

IF

NOT

OR

See also:

Logical operators

Other functions

AND(list of logical conditions)

The AND function returns the logical AND of its arguments. The arguments must be logical conditions that can take the value TRUE or FALSE.

Examples

AND(TRUE, TRUE)	returns TRUE
AND(TRUE, FALSE)	returns FALSE
AND(6 < 10, 40 > 20)	returns TRUE

See also:

Logical operators

Other logical functions

IF(x, true, false)

The IF function calculates x and evaluates it. If **x** is TRUE (or not zero), the **true** value is displayed; if **x** is FALSE (or zero), the **false** value is displayed.

For example:

> **IF(m > n, m, n)**

returns the larger value of **m** and **n**.

You can use logical and relational operators in an IF function. For example:

> **IF (A2 > 5 & A2 < 7, "A2 is OK", "A2 is not OK")**

If the value in the cell A2 is greater than 5 and less than 7, Ability displays the true value "A2 is OK"; otherwise Ability displays the false value "A2 is not OK".

See also:

> Logical operators
>
> Other logical functions

NOT(logical)

The NOT function returns the logical NOT of its argument. The argument should have a logical value of TRUE or FALSE, which NOT then negates to give the opposite value.

For example:

NOT(TRUE)	returns FALSE
NOT(2+2=5)	returns TRUE

See also:

Logical operators

Other logical functions

OR(list of logical conditions)

The OR function returns the logical OR of its arguments. The arguments must be logical conditions that can take the value TRUE or FALSE.

For example:

OR(FALSE, FALSE)	returns FALSE
OR(TRUE, FALSE)	returns TRUE
OR(2*3 < 7, 100 = 10)	returns TRUE

See also:

Logical operators

Other logical functions

Mathematical functions

You use the built-in mathematical functions to perform arithmetic and other calculations. With these functions you can also work with angles, logarithms and factorials.

Here is a complete list of mathematical functions in Ability:

- ABS
- ACOSH
- ACOTANH
- ASINH
- ATANH
- COSH
- COTANH
- DIVIDE
- EVEN
- EXP
- FACT
- FACTDOUBLE
- GCD
- INT
- LCM
- LN
- LOG
- LOG10
- MDETERM
- MINUS
- MOD
- MULTINOMIAL
- ODD
- PLUS
- POWER
- RAND
- RANDBETWEEN
- ROUND
- ROUNDDOWN
- ROUNDUP
- SERIESSUM
- SIGN
- SINH
- SQR
- SQRT
- SUM
- SUMIF
- SUMSQ
- SUMTIMES
- SUMX2MY2
- SUMX2PY2
- SUMXMY2
- SUMXPY2
- TANH
- TIMES
- TOTAL

See also:

Other functions

ABS(x)

The ABS function calculates the absolute value (i.e. it removes the negation sign, if present) of a single number, **x**. You can also enter **x** as a single cell address or cell name.

For example:

ABS(31)	returns 31
ABS(-31)	returns 31
ABS(B6)	returns positive value of the cell B6

See also:

Other mathematical functions

ACOSH(x)

The ACOSH function calculates the inverse hyperbolic cosine of a number. Number must be greater than or equal to 1.

See also:

Other mathematical functions

ACOTANH(x)

The ACOTANH function calculates the inverse hyperbolic cotangent of a number.

See also:

Other mathematical functions

ASINH(x)

The ASINH function calculates the inverse hyperbolic sine of a number.

See also:

Other mathematical functions

ATANH(x)

The ATANH function calculates the inverse hyperbolic tangent of a number. Number must be between -1 and 1 (excluding -1 and 1).

See also:

Other mathematical functions

COSH(x)

The COSH function calculates the hyperbolic cosine of a number, using the formula:

COSH(x) = (EXP(x) + EXP(-x)) / 2

See also:

EXP(x)

Other mathematical functions

COTANH(x)

The COTANH function calculates the hyperbolic cotangent of a number.

See also:

Other mathematical functions

DIVIDE(list)

The DIVIDE function calculates the division of its arguments.

For example:

DIVIDE(10, 2, 4)

returns the value 1.25, having divided 10 by 2, and the result of this by 4. The list can also be a range in a spreadsheet, as folllows:

DIVIDE(A1..A10).

See also:

Other mathematical functions

EVEN(x)

The EVEN function rounds a number up to the nearest even integer. The number will be rounded up away from 0, whether it is + or -.

For example:

EVEN(3)	returns 4
EVEN(-3)	returns -4.

See also:

Other mathematical functions

EXP(x)

The EXP function calculates the value of *e* to the power **x**, according to the following:

e^x

where *e* is the base of the natural logarithms and is approximately equal to 2.718282.

See also:

Other mathematical functions

FACT(n)

The FACT function calculates the factorial of a number **n** where n is traditionally a positive integer. The factorial of n is defined as:

n * (n-1) * (n-2) *…….* 4 * 3 * 2 * 1

with the factorial of n = 0 being defined as 1.

For example:

FACT(4)

returns 24 as calculated by 4 * 3 * 2 * 1.

FACT can be used to determine the probability of ordered events. For example, the probability of drawing 6 balls, numbered 1 through 6, in the precise sequence 1, 2, 3, 4, 5, 6 is given by the formula:

1/FACT(6)

which returns 0.001389.

FACT can be used with numbers having a fractional part. The factorial of *x* where *x* is not an integer, is calculated using the formula:

***x* * (*x* - 1) * (*x* - 2) *…..* (*x* - n)**

where 2 > (*x* - n) > 1.

The factorial of any number 0 < *x* < 1 is defined as 1. For example:

FACT(0.5)	returns 1
FACT(1.5)	returns 1.5
FACT(3.5)	returns 3.5 * 2.5 * 1.5 = 13.125

Factorial functions do not work with negative numbers, so if you enter a negative number Ability displays #VALUE in the cell.

Factorial calculations can create very large numbers. The factorial of any number bigger than 170 will cause overflow errors and #VALUE will be returned.

See also:

FACTDOUBLE(n)

Other mathematical functions

FACTDOUBLE(n)

The FACTDOUBLE function calculates the double factorial of a number **n**. It takes into account whether n is even or odd, and then carries out a factorial calculation using either the even or the odd numbers up to n.

For example:

FACTDOUBLE(5)	returns 15, that is 5 * 3 * 1.
FACTDOUBLE(6)	returns 48, that is 6 * 4 * 2.

FACTDOUBLE also works with fractional numbers (see FACT). For example:

FACTDOUBLE(5.5)	returns 28.875, that is 5.5 * 3.5 * 1.5.
FACTDOUBLE(6.5)	returns 73.125, that is 6.5 x 4.5 x 2.5.

See also:

Other mathematical functions

GCD(list)

The GCD function calculates the greatest common divisor of a **list** of integers. The greatest common divisor is the largest integer that exactly divides all the integers in a list.

For example:

GCD(24, 36)	returns 12
GCD(24, 36, 72)	returns 12
GCD(2, 7, 9)	returns 1

See also:

Other mathematical functions

INT(x)

The INT function returns the integer part of the value **x**.

For example:

INT(3.12543)

returns 3.

See also:

Other mathematical functions

LCM(list)

The LCM function calculates the least common multiple of a **list** of numbers.

For example:

LCM(3,7,21)

returns 21, which is the smallest number that is a common multiple of 3, 7 and 21.

See also:

Other mathematical functions

LN(x)

The LN function calculates the natural logarithm of the number **x**, for any positive number. Natural logarithms are based on the constant *e*, which is approximately 2.718281828.

Note that LN is the inverse of the EXP function (see EXP).

See also:

  Other mathematical functions

LOG(x, b)

The LOG function calculates the logarithm of the positive number **x** to a specified base **b**. If b is omitted, LOG uses 10 instead.

For example:

LOG(8, 2)

returns the value 3, since 2 raised to the power 3 is 8.

LOG(1e6)

returns 6, since 10 raised to the power 6 is 1,000,000 i.e. 1E6.

See also:

Other mathematical functions

LOG10(x)

The LOG10 function calculates the logarithm of the number **x** to base 10, for any positive number.

For example:

LOG(80)

returns 1.9031 since 10 raised to the power 1.9031 is approximately 80.

See also:

Other mathematical functions

MDETERM(array)

The MDETERM function calculates the matrix determinant of an array. Array must be numeric, have an equal number of rows and columns, and not contain any empty cells. If these conditions are not met MDETERM will return a #VALUE error message. Array can be given as numeric constants, for example

MDETERM({3,7,9; 1,7,2; 4,3,2})

or call upon a range of cells containing numeric values:

MDETERM(A1..C3)

MDETERM can be used to solve systems of linear equations. For example, the system of linear equations

2x -y + 2z = 2
x + 10y - 3z = 5
-x + y + z = -3

can be arranged so that the coefficients appear in the cell range A1..D3 as follows:

row/ col	**A**	**B**	**C**	**D**
1	2	-1	2	2
2	1	10	-3	5
3	-1	-1	1	-3

The system of equations can then be solved using the following formulas:

x is given by =MDETERM({2, 5, -3; -1, 10, 1; 2, -3, 1}) / MDETERM(A1..C3) = 2
y is given by =MDETERM({2, 1, -1; 2, 5, -3; 2, -3, 1}) / MDETERM(A1..C3) = 0
z is given by =MDETERM({2, 1, -1; -1, 10, 1; 2, 5, -3}) / MDETERM(A1..C3) = -1

See also:

Arrays

Other mathematical functions

MINUS(list)

The MINUS function calculates a successive subtraction from the first item of **list,** all other items of **list**.

For example:

MINUS(10, 3, 4)	returns 3, that is 10 - 3 - 4
MINUS(10, 3, -4)	returns 11, that is 10 - 3 - (-4)

See also:

Other mathematical functions

MOD(x, y)

The MOD function calculates the remainder of **x** divided by **y** according to the following formula:

MOD(x, y) = x - (y * INT(x / y))

You can calculate the fractional part of a number x using:

MOD(x,1)

See also:

INT(x)

Other mathematical functions

MULTINOMIAL(list)

The MULTINOMIAL function calculates the multinomial of a **list** of numbers. The function is calculated using according to the following:

MULTINOMIAL (a, b, c) = FACT(a + b +c) / [FACT(a)*FACT(b)*FACT(c)]

For example:

MULTINOMIAL(1, 2, 3)

returns 60.

See also:

FACT(n)

Other mathematical functions

ODD(x)

The ODD function rounds a number up to the nearest odd integer.

For example:

ODD(4)	returns 5
ODD(1.5)	returns 3
ODD(-2)	returns –3

See also:

Other mathematical functions

PLUS(list)

The PLUS function calculates the sum of its arguments.

For example:

PLUS(3, 4, 6)	returns 13, that is 3+4+6
PLUS(3, 4, -6)	returns 1, that is 3+4+(-6)

See also:

Other mathematical functions

POWER(list)

The POWER function calculates the result of a number raised to a power. The first argument place in the list contains the number and any places after this contain the power or powers.

For example:

POWER(2, 3)	returns 8, that is 2^3
POWER(2, 3, 2)	returns 64, that is (2^3)^2

See also:

Other mathematical functions

RAND(x)

The RAND function generates a random number from 0 to **x** (including 0 but excluding **x**). Each time you enter a RAND function, Ability displays a new random number.

For example:

> **RAND(100)**

generates a number from 0 to 100 (excluding 100).

Note that in Spreadsheet, you can recalculate RAND using the Calculate Now command from the Tools menu.

See also:

> Other mathematical functions

RANDBETWEEN(bottom, top)

The RANDBETWEEN function calculates a random number between the numbers you specify as function arguments. Each time you enter a RANDBETWEEN function, Ability displays a new random number.

For example,

RANDBETWEEN(100, 200)

generates a number from 100 to 200 (excluding 200).

Note that in Spreadsheet, you can recalculate RANDBETWEEN using the Calculate Now command from the Tools menu.

See also:

Other mathematical functions

ROUND(x, y)

The ROUND function calculates the value of **x** rounded to the nearest **y**.

For example:

ROUND(2.8234, 1)	returns 3 (round to nearest whole number)
ROUND(2.8234, 0.01)	returns 2.82 (round to nearest hundredth)
ROUND(2678, 100)	returns 2700 (round to nearest hundred)

See also:

Other mathematical functions

ROUNDDOWN(number, num_digits)

The ROUNDDOWN function rounds a **number** down, that is towards 0. **num_digits** is a positive or negative integer which specifies the number of digits to which you want to round down. **num_digit** set to a positive integer will round down to the right of the decimal point (the fractional part); set to a negative integer will round down to the left of the decimal point (the whole number part).

For example:

ROUNDDOWN(2.8234, 1)	returns 2.8
ROUNDDOWN(2.8234, 0)	returns 2
ROUNDDOWN(2678, -2)	returns 2600

See also:

Other mathematical functions

ROUNDUP(number, num_digits)

The ROUNDUP function rounds a **number** up, away from 0. **num_digits** is a positive or negative integer which specifies the number of digits to which you want to round up. **num_digit** set to a positive integer will round up to the right of the decimal point (the fractional part); set to a negative integer will round up to the left of the decimal point (the whole number part).

For example:

ROUNDUP(2.8234, 1)	returns 2.9
ROUNDUP(2.8234, 0)	returns 3
ROUNDUP(2678, -2)	returns 2700

See also:

Other mathematical functions

SERIESSUM(x, n, m, coefficients)

The SERIESSUM function calculates the sum of a power series:

$$a_1x^n + a_2x^{n+m} + a_3x^{n+2m} + ... + a_jx^{n+(j-1)m}$$

where **coefficients** is the list denoted by a1, a2, a3 …. aj.

Power series expansions are useful in approximating many functions. The function arguments are:

x	is a variable that contains the input value to the power series
n	is the initial power to which x is to be raised
m	is the step by which n is to be increased for each term of the series
coefficients	is a set of coefficients by which the successive powers of x are to be multiplied. The number of coefficients determines the number of terms in the power series.

For example, a power series can be used to approximate the function SIN(x). The power series expansion for SIN(x) is:

x - (x^3)/3! + (x^5)/5! - (x^7)/7! + ….

The sequence of coefficients 1, -1/3!, 1/5!, -1/7! ….. can be written as

(-1)^0, ((-1)^1)/(m + n)!, ((-1)^2)/(2m + n)!, ((-1)^3)/(3m + n)! ….. where n = 1 and m =2.

Thus to calculate sin(45˚) or, in radians, sin(π/4), set **n** = 1, **m** = 2, **x** = π/4, and put the coefficients that are to be used in a range. In a spreadsheet, if B1..B5 contain the first 5 coefficients (i.e. 1, -1/FACT(3), 1/FACT(5), -1/FACT(7), 1/FACT(9)), then the following formula can be used to approximate the sine function:

SERIESSUM(PI()/4, 1, 2, B1..B5)

which returns 0.707107.

The accuracy of the approximation depends, of course, on how many coefficients are used. The greatest accuracy possible is attained when the number of coefficients used is infinite! Note that computational modeling of mathematical functions such as sine and cosine is ultimately achieved by using power series expansions.

See also:

Other mathematical functions

SIGN(x)

The SIGN function returns the sign of a number, according to the following convention: 1 stands for **x** is positive; 0 stands for **x** is zero; -1 stands for **x** is negative.

For example:

SIGN(20 - 20)	returns 0
SIGN(-87)	returns -1
SIGN(20)	returns 1

See also:

Other mathematical functions

SINH(x)

The SINH function calculates the hyperbolic sine of a number **x**, using the formula:

SINH(x) = (EXP(x) - EXP(-x)) / 2

See also:

EXP(x)

Other mathematical functions

SQR(x)

The SQR function calculates the square of a number **x**.

For example,

SQR(5)

returns 25, that is 5 raised to the power 2.

See also:

Other mathematical functions

SQRT(x)

The SQRT function calculates the square root of **x**. The value **x** must be equal to or greater than zero. If it is not, #VALUE is displayed.

See also:

Other mathematical functions

SUM(list)

The SUM function calculates the total of the values in the **list**. The list can be numbers, cell addresses, ranges, and cell names. The SUM function is the same as the TOTAL function (see TOTAL).

Here are some examples:

SUM(12, 255, 10)

SUM(A5..C9)

SUM(Sales, C2, Expenses, Equip)

See also:

Other mathematical functions

SUMIF(list, condition)

Conditionally sums values in the **list**. For example:

SUMIF(A1..A10, “<100”)

adds up all the values less than 100 in the range A1..A10. Note that the condition must be surrounded by quotes if it contains non-numeric characters.

More examples:

Formula	**Returns**
SUMIF({5, 4, 5, 3, 5}, “>=4”)	19
SUMIF({5, 4, 5, 3, 5}, 5)	15

See also:

COUNTIF(list, condition)

Other mathematical functions

SUMSQ(list)

The SUMSQ function calculates the sum of the squares of a list of numbers.

For example

SUMSQ(3, -2, 2)

returns 17, that is 3^2 + (-2)^2 + 2^2.

See also:

Other mathematical functions

SUMTIMES(range1, range2, range3…)

The SUMTIMES function calculates the sum of the products of corresponding values from two or more arrays.

The array arguments must have the same dimensions or the error message #VALUE will be returned. Non-numeric values will be counted as 0. The array components can be entered as a range.

For example:

SUMTIMES({1,2,3}, {2,3,4})	returns 20, that is 1*2 + 2*3 + 3 * 4
SUMTIMES({1,2},{3,4}, {5,6})	returns 63, that is 1*3*5 + 2*4*6
SUMTIMES({2,3; 4,6}, {7,5; 4,9})	returns 99, that is 2*7 + 3*5 + 4*4 + 6*9
SUMTIMES(A1..B2, D1..E2)	returns A1*D1 + B1*E1 + A2*D2 + B2*E2

See also:

Other mathematical functions

SUMX2MY2(range_x, range_y)

The SUMX2MY2 function calculates the sum of the differences of squares of corresponding values in two arrays, according to the formula:

$\sum (x^2 - y^2)$

The number of values in both arrays must be the same

For example:

SUMX2MY2({4,5,6}, {7,4,3})

returns 3, that is (4^2 - 7^2) + (5^2 - 4^2) +(6^2 - 3^2)

See also:

Other mathematical functions

SUMX2PY2(range_x, range_y)

The SUMX2PY2 function calculates the sum of the sum of squares of corresponding values in two arrays, according to the formula:

$$\sum (x^2 + y^2)$$

The number of values in both arrays must be the same

For example:

SUMX2PY2({4,5,6}, {7,4,3})

returns 151, that is (4^2 + 7^2) + (5^2 + 4^2) + (6^2 + 3^2).

See also:

Other mathematical functions

SUMXMY2(range_x, range_y)

The SUMXMY2 function calculates the sum of squares of differences of corresponding values in two arrays, according to the formula:

$\sum (x - y)^2$

The number of values in both arrays must be the same

For example:

SUMXMY2({4,5,6}, {7,4,3})

returns 19, that is (4 - 7)^2 + (5 - 4)^2 + (6 - 3)^2.

See also:

Other mathematical functions

SUMXPY2(range_x, range_y)

The SUMXPY2 function calculates the sum of squares of sum of corresponding values in two arrays, according to the formula:

$$\sum (x + y)^2$$

The number of values in both arrays must be the same.

For example:

SUMXPY2({4,5,6}, {7,4,3})

returns 283, that is (4 + 7)^2 + (5 + 4)^2 + (6 + 3)^2.

See also:

Other mathematical functions

TANH(x)

The TANH function calculates the hyperbolic tangent of a number **x**, calculated as follows:

TANH(x) = SINH(x) / COSH(x)

See also:

Other mathematical functions

TIMES(list)

The TIMES function calculates the product of a list of numbers.

For example:

TIMES(4, 5, 6)

returns 120, that is 4*5*6.

See also:

Other mathematical functions

TOTAL(list)

The TOTAL function calculates the total of the list. The list can be numbers, cell addresses, ranges, and cell names. This function is the same as the SUM function (see SUM).

See also:

Other mathematical functions

Trigonometric functions

You use the trigonometric functions to solve standard problems in trigonometry, such as computing the cosine of a given angle.

Note that all angles must be supplied in radians, where pi/2 radians = 90 degrees, and all answers will be given in radians. To convert radians to degrees and degrees to radians use the DEGREES and RADIANS functions respectively.

Here is a complete list of trigonometric functions in Ability:

- ACOS
- ACOTAN
- ASIN
- ATAN
- ATAN2
- COS
- COTAN
- DEGREES
- PI
- RADIANS
- SIN
- SQRTPI
- TAN

See also:

Other functions

ACOS(x)

The ACOS function calculates the inverse cosine of the number **x**. The result is given in radians.

See also:

Other trigonometric functions

ACOTAN(x)

The COTAN function calculates the two-quadrant inverse cotangent of the number **x**. The result is given in radians.

See also:

Other trigonometric functions

ASIN(x)

The ASIN function calculates the inverse sine of the number **x**. The result is given in radians.

See also:

Other trigonometric functions

ATAN(x)

The ATAN function calculates the two-quadrant inverse tangent of the number **x**. The result is given in radians.

See also:

Other trigonometric functions

ATAN2(x, y)

The ATAN2 function calculates the four-quadrant determination of the angle formed by the point (**x**, **y**) and the x-axis. The result is given in radians.

See also:

Other trigonometric functions

COS(x)

The COS function calculates the cosine of the number **x**, where **x** is an angle given in radians.

See also:

Other trigonometric functions

COTAN(x)

The COTAN function calculates the cotangent of the number **x**, where **x** is an angle given in radians.

See also:

Other trigonometric functions

DEGREES(angle)

The DEGREES function converts an angle expressed in radians to degrees.

See also:

Other trigonometric functions

PI()

The PI function always returns the universal constant π or 3.14159265359 approximately.

See also:

Other trigonometric functions

RADIANS(angle)

The RADIANS function converts an angle expressed in degrees to radians.

See also:

Other trigonometric functions

SIN(x)

The SIN function calculates the sine of **x**, where **x** is an angle given in radians.

See also:

Other trigonometric functions

SQRTPI(x)

The SQRTPI function calculates the square root of a number **x** multiplied by π, according to the formula:

$$\sqrt{x * PI()}$$

See also:

PI()

Other trigonometric functions

TAN(x)

The TAN function calculates the tangent of **x**, where **x** is an angle given in radians.

See also:

Other trigonometric functions

Lookup functions

The built-in functions listed in this section are known as the Lookup Functions. You use these functions to search for the location of specific information. These functions are most useful if you are working with tax tables or other kinds of spreadsheet tables that you want to use for looking up and referring to information.

Here is a complete list of lookup functions in Ability:

- ADDRESS
- AREAS
- CHOOSE
- COLUMN
- COLUMNS
- FIND
- FINDEX
- INDEX
- INDIRECT
- LOOKUP
- LOOKUPEX
- ROW
- ROWS

See also:

Other functions

ADDRESS(row_number, column_number, abs_mode)

The ADDRESS function calculates the location of a reference to a single cell and returns the location as text.

The row and column numbers determine the address and the mode number determines the mode. In Spreadsheet, for example, the row number will specify the row number of the spreadsheet, the column number will specify the column letter, and the mode number will specify whether the reference address is absolute or relative, according to the following key:

abs_mode	**Reference**
1	Absolute
2	Absolute row, relative column
3	Relative row, absolute column
4 or omitted	Relative

For example:

ADDRESS(1,1,1)	returns A1
ADDRESS(10,5,3)	returns $E10
ADDRESS(100,200)	returns GR100

See also:

Other lookup functions

AREAS(list)

The AREAS function calculates the number of areas in a reference or **list** of references. An area is a range of cells that are contiguous, including a single cell.

For example:

> **AREAS(A1..D3, E10)**

returns 2.

See also:

> Other lookup functions

CHOOSE(index_num, list)

The CHOOSE function chooses a value from a **list** of values. **index_num** is the position of the value in list. The arguments of list can be numbers, formulas, text, cell references or ranges.

For example:

> **CHOOSE(3, "Sales", "Profit", "Cost")**

returns "Cost".

CHOOSE can also return a range which can be use in conjunction with other functions. For example:

> **SUM(CHOOSE(2, A1..A3, B1..B3, C1..C3))**

returns the result of SUM(B1..B3).

See also:

> Other lookup functions

COLUMN(ref)

The COLUMN function returns the column number of **ref** according to the following:

ref	**returns**
cell	column number
range	number of leftmost column in a range
omitted	current column number

For example:

COLUMN(Z100)	returns 26
COLUMN(D5..G10)	returns 4
COLUMN(mycell)	returns the column number of a named cell, mycell, or the leftmost column within mycell if it is a range.
COLUMN( )	returns 5 if the function is entered in any cell in column E

See also:

Other lookup functions

COLUMNS(array)

The COLUMNS function returns the number of columns in a reference or array.

For example:

COLUMNS(F1..H10)	returns 3
COLUMNS({1, 2; 9, 8})	returns 2

If you have created a named range called *myarea* that refers to the range A1..D4, then

COLUMNS(myarea)	returns 4.

See also:

Other lookup functions

FIND(value, list)

The FIND function searches the **list** to find where the largest item that is less than or equal to **value** occurs in the list. The list can contain numbers, text, cell addresses, ranges, and cell names.

The list must be sorted in ascending order (as contrasted with the FINDEX function – see FINDEX).

For example,

FIND(19, 1, 15, 20, 30, 35, 40)

returns 2, the second position in the list, as 15 is the last value in the list that is less than or equal to the test value.

If the list you want to search is a range that contains several rows and several columns, Ability searches the range from left to right and top to bottom.

The FIND function is similar to the LOOKUP function (see LOOKUP), except that LOOKUP returns an actual value, rather than the position of the value in a list.

The INDEX function (see INDEX) is also similar to the FIND function, except that with INDEX you enter the position rather than a test value in a list.

See also:

Other lookup functions

FINDEX(value, list)

The FINDEX function searches the **list** and returns the position of **value** in the list. If **list** does not contain **value**, #VALUE is returned.

The list can contain numbers, text, cell addresses, ranges, and cell names. The list can be in any order (which contrasts with FIND).

For example,

FINDEX(15, 18, 15, 10, 3, 35, 12)

returns 2, the second position in the list.

If the list you want to search is a range that contains several rows and several columns, Ability searches the range from left to right and top to bottom.

The FINDEX function is similar to the LOOKUP function (see LOOKUPEX), except that LOOKUPEX returns an actual value, rather than the position of the value in a list.

The INDEX function (see INDEX) is also similar to the FINDEX function, except that with INDEX you enter the position rather than a test value in a list.

See also:

Other lookup functions

INDEX(number, list)

The INDEX function returns the value in a **list** that is in the position given by **number**. The list can be numbers, text, cell addresses, ranges, and cell names. Ability searches the list and returns the value that occurs in the specified position.

For example,

INDEX(2, 1, 15, 20, 30, 35, 40)

returns 15, which is in the second position in the list.

See also:

Other lookup functions

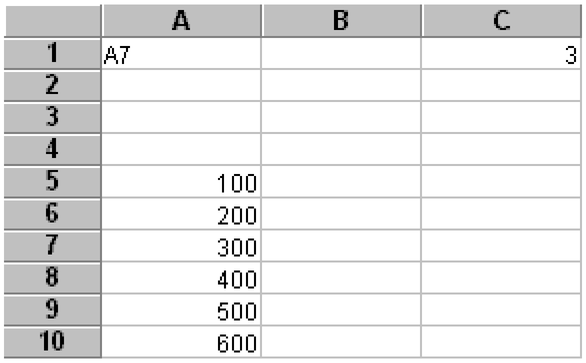
INDIRECT(ref_as_text, mode)

INDIRECT allows a text reference to be turned into a cell reference. For example:

INDIRECT("B10")

Returns the contents of cell B10 in the same way as a direct reference would (i.e. typing B10 into a formula).

The INDIRECT funtion is more useful when used with cell references. For example, consider the formula = INDIRECT(A1) / 100 in a cell, say C1. A1 contains the text A5 and the range A5..A10 contains numbers. The sum displayed in cell D1 can now be controlled by entering a different any of "A5", "A6", "A7"...."A10" in cell A1, as shown below:



	A	B	C
1	A7		3
2			
3			
4			
5	100		
6	200		
7	300		
8	400		
9	500		
10	600		

Mode is optional and can be set to true or false to switch between A1 style notation and R1C1 style notation. For example, the following will all return the contents of cell B10:

INDIRECT("B10")
INDIRECT("B10", TRUE)
INDIRECT("R10C2", FALSE)

See also:

Other lookup functions

LOOKUP(value, in_range, out_range)

The LOOKUP function searches the list **in_range**, to find the last item that is less than or equal to **value** and returns the corresponding item from the list **out_range**.

The **in_range** must be sorted into ascending order (use LOOKUPEX for unsorted lists). If it isn’t, Ability may display an unexpected answer. LOOKUP works with both sorted numbers and text.

For example:

LOOKUP(3, {1, 2, 3, 4, 5}, {10, 20, 30, 40, 50})	returns 30
LOOKUP("red", {"blue", "red", "yellow"}, {3, 2, 1})	returns 2

The LOOKUP function produces the same result as combining INDEX and FIND, as follows:

INDEX(FIND(x, in_range), out_range)

See also:

Other lookup functions

LOOKUPEX(value, in_range, out_range)

The LOOKUPEX function searches the list **in_range**, to find the item that is equal to **value** and returns the corresponding item from the list **out_range. I**f the value is not found, LOOKUPEX returns #VALUE.

The **in_range** can be in any order (as contrasted with LOOKUP). LOOKUPEX works with both numbers and text.

For example:

LOOKUPEX(3, {1, 5, 4, 3, 2}, {10, 20, 30, 40, 50})	returns 40
LOOKUPEX("blue", {"red", "blue", "yellow"}, {3, 2, 1})	returns 2

The LOOKUPEX function produces the same result as combining INDEX and FINDEX, as follows:

INDEX(FINDEX(x, in_range), out_range)

See also:

Other lookup functions

ROW(ref)

The ROW function returns the row number of **ref**, according to the following:

ref	**returns**
cell	row number
range	number of topmost row in a range
omitted	current row number

For example:

ROW(Z100)	returns 100
ROW(D5..G10)	returns 5
ROW(mycell)	returns the row number of a named cell, mycell, or the topmost row within mycell if it is a range
ROW( )	returns 5 if the function is entered in any cell in row 5

See also:

Other lookup functions

ROWS(array)

The ROW function returns the number of rows in a reference or array.

For example:

ROWS(F5..H10)	returns 6
ROWS({1, 2; 9, 8})	returns 2

If you have created a named range called myarea, that refers to the range A1..D4, then:

ROWS(myarea)	returns 5

See also:

Other lookup functions

Information functions

You can use Ability's built-in information functions to find out if a value is a number, text or error indicator, and how many blank cells or error indicators there are in a range.

Here is a complete list of information functions in Ability:

COUNTBLANK	ISBLANK
ISERR	ISERROR
ISEVEN	ISNUMBER
ISODD	ISREF
ISTEXT	

See also:

Other functions

COUNTBLANK(list)

The COUNTBLANK function counts the number of blank cells within a specified range.

For example, if A1 is blank, A2 contains a number, and A3 contains an error message, then the formula

COUNTBLANK(A1..A3)

returns 1.

Note that if a cell contains a formula which returns a blank value (for example, CONCATENATE(" ")), then Ability does not count the cell itself as blank.

See also:

Other information functions

ISBLANK(value)

The ISBLANK function returns TRUE if the value is blank or if the cell referred to is empty.

Note that if a cell contains a formula that returns a blank value (for example, CONCATENATE(" ")), then Ability does not count the cell itself as blank.

See also:

Other information functions

ISERR(list)

The ISERR function checks the list and returns a count of the number of cells that contain #CIRC, #FUNC, #DIV0, #VALUE or #REF error indicators (see Cell error indicators). The list can be cell addresses, ranges, and cell names.

You can combine the IF function (see IF(x, true, false)) and the ISERR function to check for errors and return an error message if something is incorrect. For example, in a spreadsheet that runs from A1 to E20, you might use the following:

IF(ISERR(A1..E20), "A problem", "No problem")

See also:

Other information functions

ISERROR(ref)

The ISERROR function returns TRUE if the field or cell referenced by **ref** contains any error message.

See also:

Other information functions

ISEVEN(value)

The ISEVEN function returns TRUE if the value is an even number, FALSE for all other values.

See also:

Other information functions

ISNUMBER(value)

The ISNUMBER function returns TRUE if the value is a number, FALSE for all other values.

See also:

Other information functions

ISODD(value)

The ISODD function returns TRUE if the value is an odd number, FALSE for all other values.

See also:

Other information functions

ISREF(value)

The ISREF function returns TRUE if the value is a reference.

For example,

    ISREF(sales)

returns TRUE when **sales** is the name of a range A1..A10.

See also:

Other information functions

ISTEXT(value)

The ISTEXT function returns TRUE if the value is text.

See also:

Other information functions

Statistical functions

You can use the built-in statistical functions to perform statistical and other calculations. With these functions you can automatically find the average, minimum and maximum of a set of values, calculate the number of permutations, and perform other complex computations.

Here is a complete list of statistical functions in Ability:

- AVEDEV
- AVERAGE
- BINOMDIST
- COMB
- CORREL
- COUNT
- COUNTIF
- COUNTN
- COVAR
- DEVSQ
- EXPONDIST
- FISHER
- FISHERINV
- FORECAST
- GAMMADIST
- GAMMALN
- GEOMEAN
- HARMEAN
- HYPGEOMDIST
- INTERCEPT
- KURT
- LARGE
- MAX
- MEDIAN
- MIN
- MODE
- NEGBINOMDIST
- NORMDIST
- PEARSON
- PERCENTILE
- PERCENTRANK
- PERM
- POISSON
- PROB
- QUARTILE
- RANK
- RSQ
- SKEW
- SLOPE
- SMALL
- STANDARDIZE
- STD
- STDEVP
- STEYX
- VAR
- VARP
- WAVG
- WEIBULL

See also:

Other functions

AVEDEV(list)

The AVEDEV function calculates the average deviations of numbers in a list from the mean value of the list. The formula for average deviation is:

$$\frac{1}{n}\,\text{å}\;(\text{ABS}\,(x - \text{AVG}\,(x)))$$

For example:

> **AVEDEV(2, 6, 9, 4, 5)**

returns 1.84, which is the average deviation from the mean of 5.2.

See also:

- ABS(x)
- AVERAGE(list) or AVG(list)
- Other statistical functions

AVERAGE(list) or AVG(list)

The AVERAGE function calculates the unweighted average value of one or more values in a **list**. The formula used is:

SUM(list)/ n

where **n** is the number of values in the list. The list can include cell addresses, ranges, and cell names. This function can be abbreviated to AVG.

For example:

AVERAGE(10, 20, 30)

returns 20.

See also:

Other statistical functions

BINOMDIST(number, trials, probability, cumulative)

The BINOMDIST function is used to calculate the binomial distribution. This is useful in situations where there is a fixed number of trials, the trials are independent of each other, the outcome of each trial is either success or failure, and the probability of success or failure is constant throughout the test.

The arguments of the function represent the following:

number	the number of occurrences of success in the trials
trials	the number of trials to be performed
probability	the probability of success for a single trial
cumulative	a logical value, which determines the form of the function: TRUE calculates the cumulative distribution function, which gives the probability that there are at most number successes FALSE calculates the probability mass function, which gives the probability that there are exactly number successes

For example, we want to know the probability of getting 12 heads from 20 flips of a coin. Each flip of the coin can only give either heads or tails ("success" or "failure"), and the probability of heads on each flip is constant at 0.5. Therefore, setting cumulative to FALSE, the formula:

BINOMDIST(12, 20, 0.5, FALSE)

returns the probability 0.12 that we will get *exactly* 12 successes from 20 trials.

If cumulative is set to TRUE, then the formula:

BINOMDIST(12, 20, 0.5, TRUE)

returns 0.87, that is the probability that we will get *at most* 12 heads (12 or less).

See also:

Other statistical functions

COMB(n, m)

The COMB function calculates the number of combinations of **n** distinct things taken **m** at a time using the following formula:

> **COMB(n, m) = FACT(n) / (FACT(m) * FACT(n - m))**

A combination is a selection of one or more things or events without regard to order. This can be contrasted with a permutation (see PERM), where order does matter.

For example, the chances of correctly selecting six numbers drawn from 49 in a lottery is given by:

> **COMB(49, 6)**

which returns 13,983,816, that is, 1 chance in 13,983,816.

See also:

> Other statistical functions

CORREL(array1, array2)

The CORREL function calculates the correlation coefficient between two sets of data.

The function arguments should contain numbers or refer to ranges containing numbers, and should have matching numbers of data points. The correlation coefficient is a number between -1 and 1. The formula used is:

$$(n\Sigma xy - \Sigma x\Sigma y)\Big/\sqrt{(n\Sigma x^2 - (\Sigma x)^2)(n\Sigma y^2 - (\Sigma y)^2)}$$

For example, the correlation coefficient between the height (array1) and weight (array2) of six people, can be calculated using the formula:

CORREL({1.52, 1.75, 1.96, 1.85, 1.55, 1.30}, {170, 167, 210, 182, 154, 100})

This returns a correlation coefficient of 0.918156, which indicates high correlation.

Low correlation tends towards 0, as can be seen when array1 and array2 contain many data points filled with random numbers.

See also:

Other statistical functions

COUNT(list)

The COUNT function counts the number of items of non-blank entries in **list**. The list can contain numbers, cell addresses, ranges, and cell names.

For example:

> **COUNT(A1..A10)**

returns a count of the number of cells A1 through to A10 that are non-blank.

Note that if a cell contains a formula that returns a blank value (for example CONCATENATE(" ")), then Ability does not regard the cell itself as blank.

See also:

> Other statistical functions

COUNTIF(list, condition)

Counts the the number of cells in a list matching a condition. For example:

> **COUNTIF(A1..A10, “>5”)**

returns a count of the number of cells having a value greater than 5. Note that the condition should be surrounded by quotes if it contains any non-numeric characters.

More examples:

Formula	**Returns**
COUNTIF({“Mon”, “Tue”, “Mon”, “Thu”}, “Mon”)	2
COUNTIF({5, 4, 5, 3, 5}, “>=4”)	4
COUNTIF({5, 4, 5, 3, 5}, 5)	3

See also:

> SUMIF(list, condition)
>
> Other statistical functions

COUNTN(list)

The COUNTN function counts how many numbers there are in a list of arguments. The arguments can also be ranges.

For example:

COUNTN(A1..A10)

returns a count of the number of cells A1 through to A10 containing a number, a formula with a numeric result or a date.

See also:

Other statistical functions

COVAR(array1, array2)

The COVAR function calculates the covariance between two sets of data. This is a measure of the average of the products of deviation for matching pairs from two arrays.

The function arguments should contain numbers or refer to ranges containing numbers, and should have a matching number of data points.

See also:

Other statistical functions

DEVSQ(list)

The DEVSQ function calculates the sum of squares of deviations from the mean of a list of numbers. The formula used is:

$$\sum (x - \mathrm{AVG}(x))^2$$

For example:

DEVSQ(9, 4, 8, 1, 14)

returns 98.8.

See also:

AVERAGE(list) or AVG(list)

Other statistical functions

EXPONDIST(x, lambda, cumulative)

The EXPONDIST function calculates the exponential distribution. **x** is the input value to the function; **lambda** is a distribution parameter; and **cumulative** is a logical value with the following effect:

FALSE	the probability density function is calculated, using the equation: $f(x,l\ ) = l\ e^{-l\,x}$
TRUE	the cumulative distribution function is calculated, using the equation: $F(x,l\ ) = 1 - e^{-l\,x}$

For example

EXPONDIST(0.3, 9, TRUE)	returns 0.932794
EXPONDIST(0.3, 9, FALSE)	returns 0.604850

See also:

Other statistical functions

FISHER(x)

The FISHER function calculates the Fisher transformation at **x**.

The formula used is:

f(x) = ½ LN((1 + x)/(1 - x))

for $-1 < x < 1$.

The transformation can be used to convert a skewed distribution into a more normal one, provided there is some theoretical support.

For example:

FISHER(0.85)

returns 1.256153.

See also:

LN(x)

Other statistical functions

FISHERINV(y)

The FISHERINV function calculates the inverse of the Fisher transformation (see FISHER), using the formula:

$$x = \frac{e^{2y} - 1}{e^{2y} + 1}$$

where y = FISHER(x).

For example,

FISHER(1.256153)

returns 0.85.

See also:

Other statistical functions

FORECAST(x, array_y, array_x)

The FORECAST function returns a prediction of y for a given **x** after first performing a linear regression on the data points described by **array_y** and **array_x**.

A linear regression produces a "best fit" line through the data points by minimizing the sum of the (squared horizontal) distances from each data point to the line. FORECAST then uses this line to predict a **y** value for the supplied **x** as follows:

$$y = \frac{\sum y}{n} + \left(\frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}\right)\left(x - \frac{\sum x}{n}\right)$$

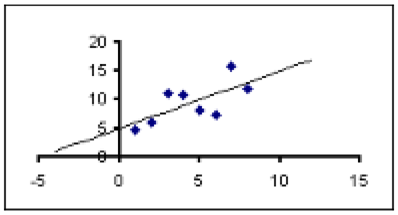
which is the same as:

y = INTERCEPT(array_y, array_x) + SLOPE(array_y, array_x)*x

For example, suppose the following y-values were observed for the supplied x-values:

Y	4.7	6.0	11.2	10.6	8.2	7.3	15.8	11.7
X	1	2	3	4	5	6	7	8

A linear regression through these points would look like this:



You could use FORECAST to predict the y value at x = 10 using the following formula:

FORECAST(10, {4.7, 6, 11.2, 10.6, 8.2, 7.3, 15.8, 11.7}, {1, 2, 3, 4, 5, 6, 7, 8})

which returns the value 14.9.

See also:

- INTERCEPT(array_y, array_x)
- SLOPE(array_x, array_y)
- STEYX(array_y, array_x)
- Other statistical functions

GAMMADIST(x, alpha, beta)

The GAMMADIST function calculates the gamma distribution. It is used when working with variables that may have a skewed distribution. **x** is the value at which the function is to be evaluated; **alpha** and **beta** are parameters to the function. The formula for the gamma distribution is:

$$f(x; a, b) = \frac{x^{a-1} e^{-N/b}}{b^{a} G(a)}$$

and for the standard gamma distribution, that is when $\beta = 1$, is:

$$f(x; a) = \frac{x^{a-1} e^{-N}}{G(a)}$$

For example:

GAMMADIST(15, 7, 4)

returns 0.022709.

See also:

Other statistical functions

GAMMALN(x)

The GAMMALN function calculates the natural logarithm of the gamma function, Γ(x) (see GAMMADIST).

The formula used is:

GAMMALN = LN(Γ(x))

See also:

Other statistical functions

GEOMEAN(list)

The GEOMEAN function calculates the geometric mean of a **list** of numbers.

Each number y of the list should be > 0.

The formula for the geometric mean is:

$\sqrt[n]{y_1 * y_2 * y_3 * .... * y_n}$

For example:

> **GEOMEAN(3, 8, 7)**

returns 5.5178, that is the 3rd or cubed root of 3*8*7.

See also:

> Other statistical functions

HARMEAN(list)

The HARMEAN function calculates the harmonic mean of a **list** of numbers. This is the reciprocal of the arithmetic mean of reciprocals.

Each number x of the list should be > 0. The harmonic mean is always less than the geometric mean, which is always less than the arithmetic mean, for the list of numbers.

The formula for the harmonic mean is:

$$\frac{n}{{}^{1}\!/_{x_1} + {}^{1}\!/_{x_2} + {}^{1}\!/_{x_3} + \dots + {}^{1}\!/_{x_n}}$$

where n is the size of the list and x1 ,x2 …. xn   are the numbers in the list.

For example,

HARMEAN(10, 5, 4, 7, 8)

returns 6.113537, given by 5/(1/10 + 1/5 + 1/4 + 1/7 + 1/8).

See also:

Other statistical functions

HYPGEOMDIST(sample, n_sample, population, n_population)

The HYPGEOMDIST function calculates the hypergeometric distribution.

This is the measure of the probability of a given number of sample "successes" from a finite population, without replacement of samples and given the sample size and the number of population "successes". Each sample is defined as either a "success" or a "failure"; for example, when flipping a coin, heads might be a success and a tails a failure.

The arguments of the function represent the following:

sample	the number of successes in the sample, for which we seek the probability
n_sample	the size of the sample
population	the number of successes in the population
n_population	the size of the population

For example, a lucky dip has been organized, with 40 items in total. 35 of these are worthless trinkets; 5 are diamonds. We wish to know the probability of getting exactly 2 diamonds from a selection of 4 items.

Each diamond counts as a success. There are 40 items in total: this is the **n_population**. The size of the sample is 4: this is the **n_sample**. The number of successes, that is to say, diamonds, in the total population is 5: this is **population**. The number of successes in the sample, for which we seek the probability, is 2: this is the **sample**.

Applying the formula:

HYPGEOMDIST(2, 4, 5, 40)

returns 0.065106 as the probability.

See also:

Other statistical functions

INTERCEPT(array_y, array_x)

The INTERCEPT function returns the point where a "best fit" line meets the y-axis. This is calculated by applying a linear regression on the data points in **array_x** (the independent or controlled variable) and **array_y** (the dependent variable).

The formula used for intercept is:

$$\frac{\sum y}{n} - \left(\frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}\right)\frac{\sum x}{n}$$

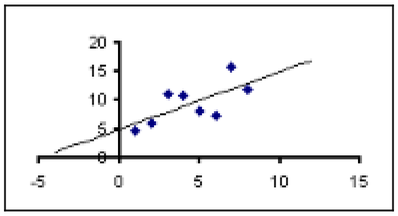
which is the same as:

AVG(array_y) - SLOPE(array_x, array_y) * AVG(array_x)

For example, suppose the following y-values were observed for the supplied x-values:

Y	4.7	6.0	11.2	10.6	8.2	7.3	15.8	11.7
X	1	2	3	4	5	6	7	8

A linear regression through these points would look like this:



You calculate where the line crosses the y-axis using:

INTERCEPT({4.7, 6, 11.2, 10.6, 8.2, 7.3, 15.8, 11.7}, {1, 2, 3, 4, 5, 6, 7, 8})

which returns 4.942857.

See also:

FORECAST(x, array_y, array_x)

SLOPE(array_x, array_y)

STEYX(array_y, array_x)

Other statistical functions

KURT(list)

The KURT function calculates the kurtosis of a **list** of numbers.

Kurtosis is used to measure how much data is massed in the center of a distribution compared to the normal distribution - a relatively flat distribution will have negative kurtosis and a pointed one a positive kurtosis. KURT needs at least 4 numbers in the list.

See also:

Other statistical functions

LARGE(k, list)

The LARGE function calculates the **k**-th largest number in a **list** of numbers. **k** should be greater than 0 and not greater than the number of numbers in the list.

For example,

LARGE(3, 9, 234, 4, 56, 888)

returns 56, the third largest number.

See also:

Other statistical functions

MAX(list)

The MAX function returns the maximum (largest) value in the **list**. The list can include numbers, cell addresses, ranges, and cell names.

For example:

MAX(1, 2, -16, 8)

returns largest number in the list, which is 8.

See also:

Other statistical functions

MEDIAN(list)

The MEDIAN function calculates the median of a **list** of numbers. The median is the middle value in a list of numbers, after the list has been arranged in ascending order. MEDIAN both orders the list and finds the middle value, so you can enter the list values in any order you like. If the list contains an even number of numbers, the median is calculated as the average of the two middle numbers.

For example:

> **MEDIAN(3, 5 ,17, 34, 98)**

returns 17.

> **MEDIAN(3, 5, 17, 26, 34, 98)**

returns 21.5, which is the average of the two middle numbers 17 and 26.

MEDIAN will return an error message if any of the list values are non-numeric, though it will ignore non-numeric values if calling on a range.

See also:

> Other statistical functions

MIN(list)

The MIN function returns the minimum (smallest) value in the **list**. The list can include numbers, cell addresses, ranges, and cell names.

For example,

> **MIN(1, 2, -16, 8)**.

Ability displays the smallest number, which is -16.

See also:

> Other statistical functions

MODE(list)

The MODE function calculates the mode of a **list** of numbers. The mode is the most frequently occurring number in the list.

If all the numbers occur with equal frequency then an error message is returned. However, if a subset of the numbers occur with greater frequency than the others, then the first number in the subset is returned.

For example:

MODE(4, 5, 9, 2, 9)	returns 9
MODE(4, 5, 2, 9, 2, 9)	returns 2
MODE(1, 2, 3, 4, 5)	returns an error message

MODE will return an error message if any of the list values or values held in a range are non-numeric.

See also:

Other statistical functions

NEGBINOMDIST(number_f, number_s, probability_s)

The NEGBINOMDIST function calculates the negative binomial distribution.

This is a measure of the probability that a certain number of failures will occur before the nth success, given that the probability of success is constant. It is used in situations where the number of trials is variable and each trial is independent of the others, but the number of successes is fixed.

The function arguments are:

number_f	the number of failures
number_s	the number of successes
probability_s	the probability of a success

For example, we want to know the probability of getting exactly 10 tails ("failures") before we get 12 heads ("successes") in a coin-tossing experiment. The probability of a success is constant at 0.5. The formula:

NEGBINOMDIST(10, 12, 0.5)

returns 0.084094 as the probability.

See also:

Other statistical functions

NORMDIST(x, mean, stdev)

The NORMDIST function calculates the normal distribution for a given **mean** and standard deviation (**stdev**), where **x** is the value for which you want to know the distribution. The formula used is:

$$\frac{1}{s\sqrt{2p}}e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2}$$

NORMDIST calculates the probability mass function.

For example:

NORMDIST(3, 2, 1)

returns 0.242, which is the probability of the event x = 3, given a normal distribution with a mean of 2 and standard deviation 1.

See also:

Other statistical functions

PEARSON(array_x, array_y)

The PEARSON function calculates the Pearson product moment correlation coefficient. This is a measure, between -1 and 1 inclusive, of the extent of the relationship between two sets of data.

array_x is the set of independent and **array_y** is the set of dependent numbers. The arrays must have a matching number of values. PEARSON is calculated using the formula:

$$r = (n\text{å}\ xy - \text{å}\ x\text{å}\ y)\Big/\sqrt{(n\text{å}\ x^2 - (\text{å}\ x)^2)(n\text{å}\ y^2 - (\text{å}\ y)^2)}$$

where r is the correlation coefficient of the linear regression line running through the data points described by array_x and array_y.

For example:

PEARSON({10, 7, 5, 3, 2}, {12, 8, 13, 4, 1})

returns 0.756437.

See also:

Other statistical functions

PERCENTILE(k, array)

The PERCENTILE function calculates the k-th percentile of the numbers in an **array** or range. **k** is a number from 0 to 1 inclusive (or 0% to 100%).

The k-th percentile P of a range of numbers is a number such that at least a proportion k of the range of numbers is smaller than or equal to P, and also at least a proportion (1 - k) of those numbers is larger than or equal to P.

For example

PERCENTILE(25%, {0, 1, 2, 3, 4})

returns 1.

PERCENTILE(25%, {1, 2, 3, 4})

returns 1.75.

See also:

Other statistical functions

PERCENTRANK(x, array)

The PERCENTRANK function calculates the percentage rank of a number in a set of numbers.

The percent rank of a value is a measure of its standing in relation to a set of values. It is always a number from 0 to 1 inclusive. **x** is the value for which the percentage rank is sought in comparison to the set of numbers in an **array** or range.

For example,

> **PERCENTRANK(2, {0, 1, 2, 3, 4})**

returns 0.5

> **PERCENTRANK(2, {1, 2, 3, 4})**

returns 0.333333.

See also:

> Other statistical functions

PERM(n, m)

The PERM function calculates the number of permutations of **n** distinct things taken **m** at a time, using the following formula:

> **PERM = FACT(n) / FACT(n - m)**

For example, to get the number of permutations of the three letters a, b and c, taking 2 at a time, use the formula:

> **PERM(3, 2)**

which returns 6. The six permutations are ab, ac, bc, ba, ca, cb.

A more complex example is a coded telegram where each word is 5 letters in length. The number of possible different words, where each word contains a letter no more than once, is given by the formula:

> **PERM(26, 5)**

which returns 7893600.

See also:

> COMB(n, m)
>
> FACT(n)
>
> Other statistical functions

POISSON(x, mean, cumulative)

The POISSON function calculates the Poisson distribution. This is useful for predicting the probability of a number of events over a specific measure, for example time, length or area.

The arguments of the function represent the following:

x	the number of events, for which we seek the probability
mean	the known average rate of events
cumulative	a logical value, which determines the form of the function: TRUE calculates the cumulative Poisson probability, which gives the probability that the number of events will be between 0 and x inclusive. FALSE calculates the Poisson probability mass function, which gives the probability that the number of events will be exactly x.

For example, if 7 cars per minute is the average traffic rate on a particular road, and their arrival is entirely random, we can calculate the probability of there being exactly 4 cars in a minute by using the formula:

POISSON(4, 7, FALSE)

which returns a probability of 0.091226.

The probability that there will be between 0 and 4 cars per minute inclusive is given by:

POISSON(4, 7, TRUE)

that is 0.172992.

See also:

Other statistical functions

PROB(x_range, prob_range, lower_limit, upper_limit)

The PROB function calculates the probability that numbers in a range are between two limits.

x_range is a range of numbers with which there is associated a range of corresponding probabilities in **prob_range**.

lower_limit and **upper_limit** are the limits of the range of numbers for which we want to know the probability. If upper_limit is omitted then it becomes equal to lower_limit, and PROB calculates the probability of just the one number.

Note that the values in prob_range must total 1, and that this range and x_range must contain the same number of values.

For example, if {2, 4, 6, 8, 10} is a list of all possible outcomes of an experiment, with corresponding probabilities {0.3, 0.1, 0.1, 0.2, 0.3}, then the probability of an event being between 4 and 8 inclusive is given by the formula:

PROB({2,4,6,8,10}, {0.3, 0.1, 0.1, 0.2, 0.3}, 4, 8)

which returns 0.4.

See also:

Other statistical functions

QUARTILE(quart, array)

The QUARTILE function calculates the quartile of an array or range of numbers.

quart is either 0, 1, 2, 3 or 4, and represents either the zero (minimum), 25th, 50th (median), 75th or 100th (maximum) percentile respectively.

For example, the 3rd quartile of a data set {12, 9, 11, 7, 4, 15} is given using the formula:

QUARTILE(3, {12, 9, 11, 7, 4, 15})

which returns 11.75. This is equivalent to using the Percentile function (see PERCENTILE):

PERCENTILE(75%, {12, 9, 11, 7, 4, 15})

Note that using quart = 0, 2, or 4 has the same effect as using the functions MIN, MEDIAN or MAX respectively.

See:

MIN

MEDIAN

MAX

Other statistical functions

RANK(number, ref, order)

The RANK function calculates the rank of a number in a list of numbers. The rank of a number is its size relative to the other numbers in the list.

The arguments for the function are:

number	the number in the list that we want to rank.
ref	the reference for the list of numbers, usually an array, range or range name.
order	determines whether the rank is to be calculated against a list sorted in descending or ascending order: setting order at 0 (or omitting it) sorts in descending order, at 1 or >1 sorts in ascending order.

For example:

RANK(12, {6, 8, 2, 4, 14, 12, 10}, 0)

returns 2; whereas:

RANK(12, {6, 8, 2, 4, 14, 12, 10}, 1)

returns 6.

See also:

Other statistical functions

RSQ(array_x, array_y)

The RSQ function calculates the square of the Pearson product moment correlation coefficient (see PEARSON). The formula is equivalent to:

> **SQR(PEARSON(array_x, array_y)**

See also:

> Other statistical functions

SKEW(list)

The SKEW function calculates the skewness of a distribution given by **list**.

Skewness is a measure of the asymmetery of a distribution round its mean. A distribution with positive skewness has an asymmetric tail that stretches towards more positive values. A distribution with negative skewness has an asymmetric tail that stretches towards more negative values.

SKEW works with a list of numbers, a range or an array. An error message is returned if there are less than 3 values.

For example:

> **SKEW(16, 17, 19, 12, 11, 10, 9)**

returns 0.367654.

See also:

> Other statistical functions

SLOPE(array_y, array_x)

The SLOPE function calculates the slope of the linear regression line through the data points described by **array_x** (the independent or controlled variable) and **array_y** (the dependent variable).

The slope is the vertical distance between any two data points divided by the horizontal distance between the same two data points. It is a measure of the rate of change of the regression line, and is given by the formula:

$$b = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

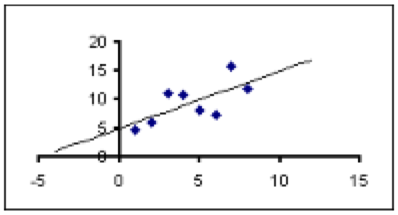
where b is the slope of the regression line.

The x- and y-arrays or ranges must have the same number of data points.

For example, suppose the following y-values were observed for the supplied x-values:

Y	4.7	6.0	11.2	10.6	8.2	7.3	15.8	11.7
X	1	2	3	4	5	6	7	8

A linear regression through these points would look like this:



You calculate the slope of the line using:

SLOPE({4.7, 6, 11.2, 10.6, 8.2, 7.3, 15.8, 11.7}, {1, 2, 3, 4, 5, 6, 7, 8})

which returns 0.9988.

See also:

- FORECAST(x, array_y, array_x)
- INTERCEPT(array_y, array_x)
- STEYX(array_y, array_x)
- Other statistical functions

SMALL(k, list)

The SMALL function calculates the **k**-th smallest value in a set of numbers in **list**.

For example:

SMALL(2, {19, 7, 1, 18, 10})

returns 7.

See also:

Other statistical functions

STANDARDIZE(x, mean, stdev)

The STANDARDIZE function calculates a nomalized value for **x**, by subtracting the **mean** and dividing the result by the standard deviation (**stdev**) of a distribution. This allows direct comparison to the standard normal distribution.

For example:

> **STANDARDIZE(30, 20, 1.8)**

returns 5.56, that is (30-20)/1.8.

See also:

> Other statistical functions

STD(list) or STDEV(list)

The STD or STDEV function estimates the standard deviation of a population based on a random sample of that population as supplied by **list**.

The standard deviation is a measure of the dispersal of data from the mean value and is defined as the square root of the variance (see VAR).

The formula used by STD is:

> **SQRT(VAR(list))**

STD assumes that **list** is a sample of the population, so use STDEVP (see STDEVP) if you wish to work with a whole population.

For example, if 5 people are chosen at random and measured for height, from a company with 40 employees, the formula:

> **STD(1.85, 1.9, 1.8, 1.75, 1.7)**

returns 0.079 as the estimated standard deviation for the population. To get the standard deviation based on all 40 employees use STDEVP.

See also:

> Other statistical functions

STDEVP(list)

The STDEVP function calculates the standard deviation of a population based on the whole of the population as supplied by **list**.

The standard deviation is a measure of the dispersal of data from the mean value. The formula used by STDEVP is:

SQRT(VARP(list))

STDEVP assumes that its arguments are the whole of a population, so use STD if you wish to estimate the standard deviation based on a sample.

For example, if you wish to calculate the standard deviation of the times, in seconds, for the members of a 4 x 400m running team, use the formula

STDEVP(55, 57, 56, 59)

which returns 1.479 seconds.

See also:

Other statistical functions

STEYX(array_y, array_x)

The STEYX function calculates the standard error associated with a linear regression on the data points described by **array_y** (the dependent variable) and **array_x** (the independent or controlled variable) according to the formula:

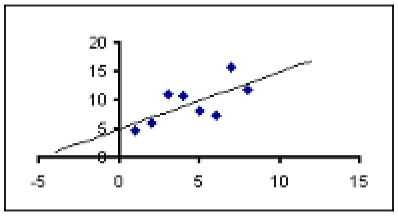
$$\sqrt{\frac{1}{n(n-2)}\left[n\sum y^2-\left(\sum y\right)^2-\frac{\left(n\sum xy-\sum x\sum y\right)^2}{n\sum x^2-\left(\sum x\right)^2}\right]}$$

Standard error is a measure of the amount of error associated with a prediction of y given x.

For example, suppose the following y-values were observed for the supplied x-values:

Y	4.7	6.0	11.2	10.6	8.2	7.3	15.8	11.7
X	1	2	3	4	5	6	7	8

A linear regression through these points would look like this:



Suppose you want to forecast y for x = 10 and supply a standard error with this forecast. The standard error would be calculated using:

STEYX({4.7, 6, 11.2, 10.6, 8.2, 7.3, 15.8, 11.7}, {1, 2, 3, 4, 5, 6, 7, 8})

which returns 2.84.

See also:

FORECAST(x, array_y, array_x)

INTERCEPT(array_y, array_x)

SLOPE(array_x, array_y)

Other statistical functions

VAR(list)

The VAR function calculates the sample variance of the **list**. The list can contain numbers, cell addresses, ranges, and cell names.

VAR is calculated according to the formula:

$$\frac{1}{n-1}\hat{a}\ (x - \mathrm{AVG}\,(x))^2$$

See also:

- AVERAGE(list) or AVG(list)
- Other statistical functions

VARP(list)

The VARP function calculates the variance of a **list** on the assumption that list contains the whole population.

The variance is a measure of the spread of data from the mean, and is calculated using the formula:

$$\frac{1}{n}\sum (x - \mathrm{AVG}(x))^2$$

VARP assumes that list is the whole of a population, so use VAR (see VAR) if you wish to estimate the variance based on a sample.

For example:

VARP(9, 6, 5, 1, 10, 9, 3)

returns 9.836735.

See also:

AVERAGE(list) or AVG(list)

Other statistical functions

WAVG(array_x, array_y)

The WAVG function calculates the weighted average of the values in **array_x**, with corresponding weights taken from **array_y**. The arguments are either ranges or arrays.

WAVG is calculated using the formula:

$$\frac{x_1y_1 + x_2y_2 + x_3y_3 + \ldots + x_ny_n}{y_1 + y_2 + y_3 + \ldots + y_n}$$

For example:

WAVG({1, 2, 3, 4, 5}, {0.3, 0.3, 0.1, 0.7, 0.8})

returns 3.64.

See also:

Other statistical functions

WEIBULL(x, alpha, beta, cumulative)

The WEIBULL function calculates the Weibull distribution. This is often used in reliability analysis.

The arguments of the function represent the following:

x	the value at which to evaluate the function
alpha	a parameter to the distribution
beta	a parameter to the distribution
cumulative	a logical value, which determines the form of the function: TRUE calculates the Weibull cumulative distribution function, which uses the formula: **F(x; α, β) = 1 - EXP(-(x/β)^α)** FALSE calculates the Weibull probability density function, which uses the formula: **f(x; α, β) = (α / β^α) x^(α - 1) EXP(-(x/β)^α)**

See also:

Other statistical functions

Text functions

You can use Ability's built-in text functions to manipulate, transform and compare text.

Here is a complete list of text functions in Ability:

CHAR	CLEAN
CODE	CONCATENATE
DOLLAR	EXACT
FINDTEXT	FIXED
LEFT	LEN
LOWER	MID
PROPER	REPLACE
REPT	RIGHT
SEARCHTEXT	SUBSTITUTE
T	TRIM
UPPER	VALUE

See also:

Other functions

CHAR(number)

The CHAR function returns the character specified by the code number. This is used to convert codes from other types of computers to characters. **number** is from 1 to 255, each number representing an ASCII character.

For example:

CHAR(37)	returns "%"
CHAR(57)	returns "9"

See also:

Other text functions

CLEAN(text)

The CLEAN function removes all non-printable characters from **text**. These are characters that are not normally displayed by your operating system.

For example:

> **CLEAN (CONCATENATE (CHAR (7), "Hello"))**

returns "Hello".

See also:

> Other text functions

CODE(text)

The CODE function returns the numeric code for the first character in a **text** string. The code corresponds to a character in ASCII.

For example:

> **CODE("W")**

and

> **CODE("Wittgenstein")**

both return 87.

See also:

> Other text functions

CONCAT(text_list)

The CONCAT function joins several text items into one text item. **text_list** can be composed of text, numbers or single-cell references.

CONCAT is an abbreviation for the CONCATENATE function.

See:

CONCATENATE

Other text functions

CONCATENATE(text_list)

The CONCATENATE function joins several text items into one text item. **text_list** can be composed of text, numbers or single-cell references.

For example, if the cell A1 contains the number 5, the formula:

> **CONCATENATE("Petrol is $", A1, " today")**

returns "Petrol is $5 today".

An abbreviated version of this function is CONCAT.

See:

CONCAT

Other text functions

DOLLAR(number, precision)

The DOLLAR function converts a **number** to text, using a currency format. number is the number to be converted and formatted.

precision specifies how the number is to be rounded. For example, 2 will round the number to 2 decimal places, while -2 will round the number to 2 places to the left of the point. If precision is omitted the default currency precision is taken from Windows settings.

For example:

DOLLAR(234.19999, 2)	returns "$234.20"
DOLLAR(234.19999, -2)	returns "$200"

See also:

Other text functions

EXACT(text1, text2)

The EXACT function checks to see if two text values are identical. TRUE is returned if they are, FALSE if not.

EXACT is case sensitive.

For example, suppose a field called *myfield* contains the text "Happy today", the following results are obtained:

EXACT("Happy today", myfield)	TRUE
EXACT("happy today", myfield)	FALSE
EXACT("Happy", myfield)	FALSE

See also:

Other text functions

FINDTEXT(find_text, within_text, start_num)

The FINDTEXT function finds one text string within another and returns the position of the character at which the two text strings first match.

FINDTEXT is case sensitive and does not allow wildcard characters (in this it differs from SEARCHTEXT – see SEARCHTEXT).

find_text is the text you wish to find.

within_text is the object text.

start_num is the position of the character in **within_text** from which you want to begin the search.

If **find_text** is "" (empty) the character at **start_num** is returned. If **start_num** is omitted it is assumed to be 1. The #VALUE error message is returned if FINDTEXT fails to find a match.

For example:

FINDTEXT("br", "abracadabra",5)	returns 9
FINDTEXT("r", "Robertson", 1)	returns 5
FINDTEXT("R", "Robertson", 1)	returns 1
FINDTEXT(" ", "Mr Smith")	returns 3

See also:

Other text functions

FIXED(number, precision, commas)

The FIXED function formats a number as text with a fixed number of decimals.

number is the number to be rounded and formatted.

precision specifies how the number is to be rounded. For instance, 2 will round the number to 2 decimal places, while -2 will round the number to 2 places to the left of the point.

commas is a logical value which determines whether the number is to have commas (TRUE) or not (FALSE or omitted).

For example:

FIXED(1234.19999, 2, TRUE)	returns "1,234.20"
FIXED(1234.19999, -2, FALSE)	returns "1200"

See also:

Other text functions

LEFT(text, num_chars)

The LEFT function returns the left-most characters from a **text** string.

num_chars is the number of characters from the left of the text string. If this is omitted, it is assumed to be 1; if it is greater than the number of characters in text, then LEFT will return the whole text string.

For example:

LEFT("WC1 5NH", 3)

returns "WC1".

See also:

Other text functions

LEN(text)

The LEN function returns the number of characters in a **text** string. Spaces are counted as characters.

For example,

LEN("The Queen of England")

returns 20.

See also:

Other text functions

LOWER(text)

The LOWER function converts **text** to lower case.

For example,

LOWER("SALES REPORT")

returns "sales report".

See also:

Other text functions

MID(text, start_num, num_chars)

The MID function returns a specific number of characters from a **text** string, starting at a specified position.

If **start_num** is greater than the length of text, MID returns "" (empty string).

If **num_chars** is greater than the number of characters remaining after **start_num**, or if it is omitted, MID returns the remaining characters.

For example,

MID("The discount is 10% in March", 24, 10)

returns "March".

See also:

Other text functions

PROPER(text)

The PROPER function capitalizes the first letter in each word of a **text** value. A word is any continuous string of letters that follows a character other than a letter. Letters that are not capitalized by PROPER are reduced to lower case.

For example:

PROPER("no smoking")	returns "No Smoking"
PROPER("NO SMOKING")	returns "No Smoking"
PROPER("Leonardo’s masterPiece")	returns "Leonardo’S Masterpiece"

See also:

Other text functions

REPLACE(old_text, start_num, num_chars, new_text)

The REPLACE function replaces characters within text. This enables you to replace a specified text string with another text string.

old_text contains the string which is to be replaced.

start_num gives the number of the character at which the replacement is to begin.

num_chars is the number of characters to be replaced.

new_text is the text that will replace the relevant string in **old_text**.

For example:

REPLACE("November", 1, 3, "Dec")	returns "December"
REPLACE("Mr Bill Clinton", 1, 7, "President")	returns "President Clinton"

See also:

Other text functions

REPT(text, number_times)

The REPT function repeats **text** a given **number** of times.

For example,

REPT("x", 5)

returns "xxxxx".

See also:

Other text functions

RIGHT(text, num_chars)

The RIGHT function returns the right-most characters from a **text** string.

num_chars is the number of characters from the right of the text string. If this is omitted it is assumed to be 1. If it is greater than the number of characters in **text** then RIGHT will return the whole text string.

For example,

RIGHT("New York", 4)	returns "York"
RIGHT("Washington", 20)	returns "Washington"

See also:

Other text functions

SEARCHTEXT(find_text, within_text, start_num)

The SEARCHTEXT function finds one text string within another and returns the position of the character at which the two text strings first match.

SEARCHTEXT is case-insensitive and allows wildcard characters (* or ?) (This differs from FINDTEXT – see FINDTEXT). If you wish to search for the characters * or ? themselves, use ~* or ~?.

find_text is the text you wish to find. If **find_text** is "" (empty) the position of the character at **start_num** is returned.

within_text is the object text.

start_num is the position of the character in **within_text** from which you want to begin the search. If **start_num** is omitted it is assumed to be 1.

The #VALUE error message is returned if SEARCHTEXT fails to find a match.

For example,

=SEARCHTEXT("br", "AbraCADaBRa", 5)	returns 9
=SEARCHTEXT("r", "Robertson", 1)	returns 1
=SEARCHTEXT("R*t", "Robertson", 1)	returns 1
=SEARCHTEXT("", "Mr Smith", 4)	returns 4
=SEARCHTEXT("~*", "XXXX*XX")	returns 5

See also:

Other text functions

SUBSTITUTE(text, old_text, new_text, instance_num)

The SUBSTITUTE function substitutes new text for old text in a text string. This enables you to replace a specified text string with another text string (see REPLACE if you wish to replace text at a specific location in a text string).

text contains the text string that is to be replaced.

old_text is the text that is to be replaced.

new_text is the text that is to be substituted for **old_text**.

instance_num specifies which instance of **old_text** is to be replaced (if this is omitted all instances of **old_text** are replaced).

For example:

SUBSTITUTE("Happy New Year", "New Year", "Birthday")

returns "Happy Birthday"

SUBSTITUTE("Apri1/1/1997", "9", "8", 2)

returns "April/1/1987"

SUBSTITUTE(SUBSTITUTE("(x + y)/(x - y)", "(", "["), ")", "]")

returns "[x + y]/[x - y]"

See also:

Other text functions

T(value)

The T function returns its argument if it is text, otherwise it converts a value to text. This is not a function that will often be needed, since Ability automatically converts most values as necessary in formulas.

For example:

T("FALSE")	returns "FALSE"
T(FALSE)	returns "FALSE"
T(1000)	returns "1000"

This function is the reverse of VALUE (see VALUE).

See also:

Other text functions

TRIM(text)

The TRIM function removes spaces from **text**, except single spaces between words.

For example

TRIM("Dear Mr   Clinton")

returns "Dear Mr Clinton".

See also:

Other text functions

UPPER(text)

The UPPER function converts **text** to uppercase.

For example:

> **UPPER("hello")**

returns "HELLO".

See also:

> Other text functions

VALUE(text)

The VALUE function converts a **text** argument to a number.

For example,

VALUE("222")

returns 222.

This function is the reverse of T(value) (see T(value)).

See also:

Other text functions

Document functions

You can use Ability's built-in document functions to display information about the document in which you are currently working. This is useful if you want a print-out to contain document information within the document itself.

The document functions directly access the information in **Summary** and **Statistics** contained in **Properties** under the **File** menu.

Here is a complete list of document functions in Ability:

Function	**Operation**
AUTHOR	returns author from Summary Information
COMMENTS	returns comments from Summary Information
CREATEDATE	returns the date document was created from Statistics
EDITTIME	returns the total document editing time from Statistics
FILENAME	returns the document name and location
FILESIZE	returns the size on the disk of the document
KEYWORDS	returns keywords from Summary Information
MERGECOUNT	returns the total number of merge records in the document
MERGEREC	returns the number of the current merge record
NUMCHARS	returns the number of characters in the document from Statistics (only applies to Write)
NUMPAGES	returns the number of pages in the document from Statistics
NUMWORDS	returns the number of words in the document from Statistics (only applies to Write)
PAGE	returns the number of the current page in the document
PRINTDATE	returns the date the document was last printed from Statistics
REVNUM	returns the number of times the document has been saved from Statistics
SAVEDATE	returns the date the document was last saved from Statistics
SUBJECT	returns the subject of the document from Summary Information
TITLE	returns the title of the document from Summary Information

See also:

Other functions

AUTHOR()

Returns the author's name. This is taken directly from the Summary page under the File/Properties menu.

See also:

Other document function

COMMENTS()

Returns document comments. This is taken directly from the Summary page under the File/Properties menu.

See also:

Other document function

CREATEDATE()

Returns the date the document was created. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

EDITTIME()

Returns the total document editing time. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

FILENAME()

Returns the document name and location.

See also:

Other document function

FILESIZE()

Returns the number of bytes the document occupies on disk.

See also:

Other document function

KEYWORDS()

Returns the document keywords. This is taken directly from the Summary page under the File/Properties menu.

See also:

Other document function

MERGECOUNT()

Returns the total number of merge records in the document.

If mail merge is enabled, this is the total number of records in the current mail merge. (A tick will show next to Mail merge on the Tools menu). If mail merge is not enabled, returns #VALUE.

See also:

Other document function

MERGEREC()

Returns the number of the current merge record, providing mail merge is enabled. (A tick will show next to Mail merge on the Tools menu).

If mail merge is not enabled, returns #VALUE.

See also:

Other document function

NUMCHARS()

Returns the total number of characters in current document. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

NUMPAGES()

Returns the number of pages in the current document. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

NUMWORDS()

Returns the number of words in the current document. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

PAGE()

Returns the number of the current page in the document.

See also:

Other document function

PRINTDATE()

Returns the date the document was last printed. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

REVNUM()

Returns the number of times the document has been saved. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

SAVEDATE()

Returns the date the document was last saved. This is taken directly from the Statistics page under the File/Properties menu.

See also:

Other document function

SUBJECT()

Returns the subject of the document. This is taken directly from the Summary page under the File/Properties menu.

See also:

Other document function

TITLE()

Returns the title of the current document. This is taken directly from the Summary page under the File/Properties menu.

See also:

Other document function

Remote functions

You can use Ability's built-in remote functions to work with information from other documents. These allow you to display, manipulate and edit external information in the current document. The remote functions are useful when you need to access external database, spreadsheet or write information from a current spreadsheet or write document. Most of the remote functions call on a database object. This is a table, query, relation or SQL statement.

Here is a complete list of remote functions in Ability:

DBFIELDCOUNT	DBFIELDNAME
DBFILTER	DBFILTERSORT
DBGET	DBQUERYCOUNT
DBQUERYNAME	DBRELATIONCOUNT
DBRELATIONNAME	DBSORT
DBSQL	DBSQLFILTER
DBSQLSORT	DBSQLFILTERSORT
DBTABLECOUNT	DBTABLENAME
HYPERLINK	REMOTE
SSGET	WPGET

See also:

Other functions

DBFIELDCOUNT(database, source)

The DBFIELDCOUNT function returns the number of fields in a database object.

database is the name of a database and must include the full path if not in the same directory as the open document.

source is the name of the database object, which can be a table, query, relation or SQL statement.

For example, if there is a database called wineshop.adb in a folder c:\ability\samples, which contains a table called Contacts, then:

DBFIELDCOUNT("c:\ability\samples\wineshop.adb", "Contacts")

returns the number of fields.

See also:

Other remote functions

DBFIELDNAME(database, source, index)

The DBFIELDNAME function returns the name of a field in a database object.

database is the name of a database and must include the full path if not in the same directory as the open document.

source is the name of the database object, which can be a table, query, relation or SQL statement.

index is a field number.

The function arguments are case insensitive.

For example, if there is a database called wineshop.adb in a folder c:\ability\samples, which contains a table called Contacts, then:

DBFIELDNAME("c:\ability\samples\wineshop.adb", "Contacts", 1)

returns the name of the first field in the table.

See also:

Other remote functions

DBFILTER(database, table, filter, field_name, rec_num)

The DBFILTER function returns the contents of a field from a database table, after a specified named filter has been applied.

This requires that the filter has already been defined in Database. This contrasts with DBSQLFILTER (see DBSQLFILTER), which is similar in functionality but allows you to define simple filters "on-the-fly".

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

filter is the named filter that has been applied to the table or relation.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. A filter called Red has been defined that filters out all wines other than red. The table contains the name of each wine in a Title field and the price in a field called Price as follows:

Title	Price	Classification
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBFILTER can be used to obtain the following results:

DBFILTER("c:\ability\samples\wineshop.adb", "products", "red")

returns "Château Haut du Puy" as this is the first column and row off the filtered table.

DBFILTER("c:\ability\samples\wineshop.adb", "products", "red", 1, 3)

returns "Le Chambertin", as this is the value from 3rd row and 1st column.

DBFILTER("c:\ability\samples\wineshop.adb", "products", "red", "price", 2)

returns 27, the 2nd record of the Price column.

AVG(DBFILTER("c:\ability\samples\wineshop.adb", "products", "red", "price"))

returns 31.375 - the average price of red wine.

ROWS(DBFILTER("c:\ability\samples\wineshop.adb", "products", "red"))

returns 12, the total number of records in the filtered table.

See also:

Other remote functions

DBFILTERSORT(database, table, filter, sort, field_name, rec_num)

The DBFILTERSORT function returns the contents of a field from a database table after a specified named filter and sort order have been applied.

This requires that the filter and sort order have already been defined in Database. This contrasts with DBSQLFILTERSORT (see DBSQLFILTERSORT), which is similar in functionality but allows you to define simple filters and sort orders "on-the-fly".

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

filter is the named filter that has been applied to the table or relation.

sort is the named sort order that has been applied to the table or relation.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. A filter called Red has been defined that filters out all wines other than red. The table contains the name of each wine in a Title field and the price in a field called Price as follows:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

A sort order called ByPrice has been created that sorts in ascending order the price field.

DBFILTERSORT can be used to obtain the following results:

DBFILTERSORT("c:\ability\samples\wineshop.adb", "products", "red", "byprice")

returns "Château Haut du Puy" as this is the first column and row off the filtered table when sorted by price.

DBFILTERSORT("c:\ability\samples\wineshop.adb", "products", "red", "byprice", 1, 3)

returns " Chambolle-Musigny", as this is the 3rd cheapest wine.

DBFILTERSORT("c:\ability\samples\wineshop.adb", "products", "red", "byprice", "price", 2)

returns the price of the second cheapest wine, 27.

See also:

Other remote functions

DBGET(database, source, field_name, rec_num)

The DBGET function returns the contents of a field in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

source is the name of the database object, which can be a table, query, relation or SQL statement.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a table called Products, which details a list of wines. The table contains three fields as follows:

Title	Price	Classification
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBGET can be used in the following ways:

DBGET("c:\ability\samples\wineshop.adb", "products", 1, 1)

returns "Château Haut du Puy", the first field from the first record.

DBGET("c:\ability\samples\wineshop.adb", "products", "price", 3)

returns 53.00, the Price field from record three.

SUM(DBGET("c:\ability\samples\wineshop.adb", "products", "price"))

returns 125.50, the sum of the Price field.

See also:

Linking to Write

Linking to Spreadsheet

Other remote functions

DBQUERYCOUNT(database)

The DBQUERYCOUNT function returns the number of queries in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a three queries:

> **DBQUERYCOUNT("c:\ability\samples\wineshop.adb")**

will return 3.

See also:

> Other remote functions

DBQUERYNAME(database, index)

The DBQUERYNAME function returns the name of a specified query in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

index is the query number, starting at 1.

For example, you know a database called wineshop, located in a folder c:\ability\samples, contains at least one query:

DBQUERYNAME("c:\ability\samples\wineshop.adb", 1)

returns the name of the first query in the database.

Suppose you enter "c:\ability\samples\wineshop.adb" into a cell in a spreadsheet, say A1. Then the following:

DBQUERYNAME(A1, DBQUERYCOUNT(A1))

returns the name of the last query.

See also:

DBQUERYCOUNT(database)

Other remote functions

DBRELATIONCOUNT(database)

The DBRELATIONCOUNT function returns the number of relations in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a four relations:

> **DBRELATIONCOUNT("c:\ability\samples\wineshop.adb")**

will return 4.

See also:

> Other remote functions

DBRELATIONNAME(database, index)

The DBRELATIONNAME function returns the name of a specified relation in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

index is the relation number, starting at 1.

For example, you know a database called wineshop, located in a folder c:\ability\samples, contains at least one relation:

DBRELATIONNAME("c:\ability\samples\wineshop.adb", 1)

returns the name of the first relation in the database.

Suppose you enter "c:\ability\samples\wineshop.adb" into a cell in a spreadsheet, say A1. Then the following:

DBRELATIONNAME(A1, DBRELATIONCOUNT(A1))

returns the name of the last relation.

See also:

DBRELATIONCOUNT(database)

Other remote functions

DBSORT(database, table, sort, field_name, rec_num)

The DBSORT function returns the contents of a field from a database table, after a specified named sort order has been applied.

This requires that the sort order has already been defined in Database. This contrasts with DBSQLSORT (see DBSQLSORT), which is similar in functionality but allows you to define simple sort orders "on-the-fly".

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

sort is the named sort order that has been applied to the table or relation.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. A sort order called ByPrice has been defined acting in ascending order on the Price field. The unsorted table looks like:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Meursault-Charmes	54.00	white
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBSORT can be used to obtain the following results:

DBSORT("c:\ability\samples\wineshop.adb", "products", "byprice")

returns "Château Haut du Puy" as this is the first column and row of the table when sorted by price.

DBSORT("c:\ability\samples\wineshop.adb", "products", "byprice",   1, 3)

returns " Chambolle-Musigny", as this is the 3rd cheapest wine.

DBSORT("c:\ability\samples\wineshop.adb", "products", "byprice", "price", 2)

returns the price of the second cheapest wine, 27.

See also:

Other remote functions

DBSQL(database, SQL_statement, field_name, rec_num)

The DBSQL function returns the value of an SQL query.

database is the name of a database and must include the full path if not in the same directory as the open document.

SQL_statement is a text string containing the full syntax of a SQL statement.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive with the exception that SQL commands within **SQL_statement** should all be upper case.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. Products contains three fields as follows:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Meursault-Charmes	54.00	white
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBSQL can be used to obtain the following results:

DBSQL("c:\ability\samples\wineshop.adb", "SELECT * FROM products")

returns the entire table. "Château Haut du Puy", is displayed as it is contents of the first column from the first record.

DBSQL("c:\ability\samples\wineshop.adb", "SELECT * FROM products", 3, 3)

returns "white", the third column from the third record.

ROWS(DBSQL("c:\ability\samples\wineshop.adb", "SELECT Price FROM products WHERE Price > 50"))

returns 2, since there are two wines over 50.

The examples given here only touch on the possibilities with SQL. For a general introduction to SQL, with the full range of supported SQL commands in Ability, see the SQL Reference Guide.

See also:

Other remote functions

DBSQLFILTER(database, table, filter, field_name, rec_num)

The DBSQLFILTER function returns the value of a database SQL query that applies a filter.

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

filter is a text string containing an SQL filter statement and takes the general form of *fieldname* <*operato*r> *comparison*. The following could all be used in the examples below:

filter	**meaning**
Price = 50	Price exactly equal to 50
Price >= 50	Price greater than or equal to 50
Price > 25 AND Price < 50	Price less than 50 but greater than 25
Classification = 'red'	Classification exactly equal to 'red'
Title LIKE 'Château*'	Title begins with ' Château' where * is a wildcard

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive with the exception of SQL operators within the filter statement, which must be upper case.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. Products contains three fields as follows:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Meursault-Charmes	54.00	white
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBSQLFILTER can be used to obtain the following results:

DBSQLFILTER("c:\ability\samples\wineshop.adb", "products", "Classification = 'white' ")

returns "Meursault-Charmes", as this is the contents of the first column from the first (and only in this case) record that matches the filter condition. Note that the filter parameter is surrounded by double quotes and the text that Classification is compared to - 'white' - is in single quotes. This is necessary to distinguish the end of each text string respectively.

DBSQLFILTER("c:\ability\samples\wineshop.adb", "products", "Classification = 'red' ", 2, 3)

returns 53.00, the second column from the third record that matches the filter.

ROWS(DBSQLFILTER("c:\ability\samples\wineshop.adb", "products", "Price > 50", "Price"))

returns 2, since there are two wines over 50.

Note that

DBSQLFILTER(database, table, filter, column, record)

is equivalent to DBSQL (see DBSQL) as follows:

DBSQL(database, "SELECT * FROM <table> WHERE <filter>", column, record).

For more details on how to build valid filter conditions, see the SQL primer.

See also:

Other remote functions

DBSQLSORT(database, table, sort, field_name, rec_num)

The DBSQLSORT function returns the value of a database SQL query that applies a sort order.

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

sort is a text string containing an SQL sort statement. This is a field name or names with an optional "DESC" to reverse the default ascending order.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. Products contains three fields as follows:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Meursault-Charmes	54.00	white
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBSQLSORT can be used to obtain the following results:

DBSQLSORT("c:\ability\samples\wineshop.adb", "price DESC")

returns "Meursault-Charmes", the first column of the first record of the table when sorted by Price in descending order.

DBSQLSORT("c:\ability\samples\wineshop.adb", "price", 2, 3)

returns 27.00, the second cheapest wine price.

Note that:

DBSQLSORT(database, table, sort, column, record)

is equivalent to using DBSQL (see DBSQL) as follows:

DBSQL(database, "SELECT * FROM <table> ORDER BY <sort>", column, record)

See also:

Other remote functions

DBSQLFILTERSORT(database, table, filter, sort, field_name, rec_num)

The DBSQLFILTERSORT function returns the value of a database SQL query that applies a filter and a sort order.

database is the name of a database and must include the full path if not in the same directory as the open document.

table is the name of the table within the database. **table** could also be the name of a relation.

filter is a text string containing an SQL filter statement and takes the general form of *fieldname* <*operato*r> *comparison*. The following could all be used in the examples below:

filter	**meaning**
Price = 50	Price exactly equal to 50
Price >= 50	Price greater than or equal to 50
Price > 25 AND Price < 50	Price less than 50 but greater than 25
Classification = 'red'	Classification exactly equal to 'red'
Title LIKE 'Château*'	Title begins with 'Château' where * is a wildcard

sort is a text string containing an SQL sort statement. This is a field name or names with an optional "DESC" to reverse the default ascending order.

field_name is optional and can be either the name of the field within a table or relation or the column number. If omitted, every row and column of the filtered table is returned.

rec_num is optional (and cannot be specified if **field_name** is omitted) and determines the record or row number of the returned value. If omitted, the entire column is returned (or table if **field_name** is also omitted).

The function arguments are case insensitive with the exception of SQL operators within the **filter** statement, which must be upper case.

For example, a database called wineshop, located in a folder c:\ability\samples, contains a Product table of wines. Products contains three fields as follows:

Title	**Price**	**Classification**
Château Haut du Puy	17.00	red
Volnay-Santenots	27.00	red
Meursault-Charmes	54.00	white
Le Chambertin	53.00	red
Chambolle-Musigny	28.50	red

DBSQLFILTER can be used to obtain the following results:

DBSQLFILTERSORT("c:\ability\samples\wineshop.adb", "products", "classification = 'red' ","price DESC")

Returns "Le Chambertin", the most expensive red wine.

DBSQLFILTER("c:\ability\samples\wineshop.adb", "products", "classification = 'red' ", "price", 2, 3)

returns 28.50, the second column from the third cheapest, red wine.

Note that:

DBSQLFILTERSORT(database, table, filter, sort, column, record)

is equivalent to using DBSQL (see DBSQL) as follows:

DBSQL(database, "SELECT * FROM <table> WHERE <filter> ORDER BY <sort>", column, record)

See also:

Other remote functions

DBTABLECOUNT(database)

The DBTABLECOUNT function returns the number of tables in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

For example, if a database called wineshop. located in a folder c:\ability\samples, contains seven tables, then

> **DBTABLECOUNT("c:\ability\samples\wineshop.adb")**

will return 7.

See also:

> Other remote functions

DBTABLENAME(database, index)

The DBTABLENAME function returns the name of a specified table in a database.

database is the name of a database and must include the full path if not in the same directory as the open document.

index is the table number, starting at 1.

For example, you know a database called wineshop, located in a folder c:\ability\samples, contains at least one table:

DBTABLENAME("c:\ability\samples\wineshop.adb", 1)

returns the name of the first table in the database.

Suppose you enter "c:\ability\samples\wineshop.adb" into a cell in a spreadsheet, say A1. Then the following:

DBTABLENAME(A1, DBTABLECOUNT(A1))

returns the name of the last table.

See also:

DBTABLECOUNT(database)

Other remote functions

HYPERLINK(link location, friendly name)

Creates a hyperlink jump to a remote document specified by **link location**. The remote document could be any local file (a Write document or spreadsheet) or a URL to an Internet location.

Friendly name is an optional parameter that masks the actual link details with more meaningful text. If friendly name is omitted, the function displays the link location.

HYPERLINK is different to all other functions in that it creates an interface componant for the user – as the mouse pointer is moved over the text it will change into a pointer and by default, the text is displayed in a blue underlined font. HYPERLINK is very similar to links used in web pages.

Examples:

=HYPERLINK(“http://www.ability.com/”)	Displays http://www.ability.com/ and opens the web site using the default browser.
=HYPERLINK("http://www.ability.com/", "Ability's web site")	Displays Ability's web site but otherwise behaves exactly as above.
=HYPERLINK(A1, A2)	Displays whatever is in A2 and opens the file/URL defined in A1.
=HYPERLINK("c:\my documents\myfile.aww", "My text file")	Runs Write and opens the file myfile.aww

Note: if the link location points to a local document, the associated program will be launched to open the file. For example if you have Microsoft's Paint as the default editor for all bmp (bitmap) files, then Paint would be launched if HYPERLINK specified a file with a .bmp extension.

See also:

Other remote functions

REMOTE(document_name, reference_name)

The REMOTE function returns the value of a remote object. Usually this means the value of a cell in a spreadsheet or a field in a document. REMOTE is the "grandfather" of all linking functions - every other remote function can be expressed using just REMOTE.

document_name is the name of a remote document and must include the full path if not in the same directory as the open document. The document can be a spreadsheet, database or word processing document.

reference_name is the name of the object whose value is sought. An object can be a named field in a write or spreadsheet document, a cell or range reference in a spreadsheet, or a table, query, relation or SQL statement in a database.

Note that there are easier-to-use functions that will achieve the same result, as follows:

WPGET	for linking to Write
SSGET	for linking to Spreadsheet
DBGET	for linking to Database

Here are some simple examples, using REMOTE to link to spreadsheets and write documents.

REMOTE("c:\ability\samples\letter.aww", "myfield")

returns the value of the field *myfield* from the Write document *letter.aww*.

REMOTE("c:\ability\samples\income.aws", "A1")

returns the value of the cell A1 from the spreadsheet *income.aws*.

REMOTE("spread1", "C1..E17"))

returns the range C1..E17 from an untitled spreadsheet, *Spread1*. Note that since Spread1 has not yet been saved it does not have an extension.

Using REMOTE for database linking is usually unnecessary, as there is a full range of specific database linking functions.

The full syntax for REMOTE, including database links, is as follows:

Application	**reference_name**
Write	**reference_name** is a **field**, usually inserted into a write document in order to perform calculations, or to establish a link to a spreadsheet cell or database field. For example, to get the value of a field called "daysales" in a write document called "report.aww", in the current directory, use the formula: **REMOTE("report.aww", "daysales")**
Spreadsheet	**reference_name** is either: **cell reference** ("A1") **range reference** ("A1..C3") **range name** ("myrange") Spreadsheet documents are made up of cells. Groups of these are called ranges and can be named. It is possible to refer to a cell, a range of cells or a named range. Spreadsheet also allows fields, which enable links to be established with other Ability applications, for example, when using mail merge, or with other fields in the same

spreadsheet. All these are spreadsheet objects that can be referred to by REMOTE.

For example, to get the value of the cell reference B21 in a spreadsheet called "manager", in the current directory, use the formula:

REMOTE("manager.aws", "B21")

Database

reference_name is either:

table

query

relation

SQL statement

In addition, a filter and/or sort order can be applied to the object and it is possible to extract a specific field or record.

The syntax for tables is:

"table !sort_order !filter !field_name ! record_number"

The syntax for queries is:

"query !!!field_name !record_number"

The syntax for relations is:

"relation !sort_order !filter !field_name ! record_number"

The syntax for SQL statements is:

"SQL_statement !!!field_name !record_number"

Note that reference_name must consist of exactly one database object, namely a table, query, relation or SQL statement. This is optionally followed by a combination of database query components (sort order, filter) and specific locations (field name, record number). Each query component and location following the database object must be prefixed by an exclamation mark - !.

Tables and relations can have a sort_order and filter applied to them, and refer to a field_name and record_number; queries and SQL statements can refer to a field_name and record_number, but are always empty for sort_order and filter.

The arguments of reference_name are always entered in the same order, namely sort_order, filter, field_name and record_number. If any of these are empty, the ! must still be entered in the reference, but only up to the last argument used in the reference.

For instance, if the reference is to a field_name in a table, but no sort_order, filter or record_number is specified, then reference_name contains table and field_name, which is prefixed by !, and !! to stand for the two empty query components between these. No ! is needed, however, for the missing record_number, since this comes after field_name. The full reference_name is "table!!!field_name".

If no query components or locations are specified, REMOTE will return the database object, for example, a table, but display only the first field of the first record.

Likewise, if a reference is made to a field_name or record_number, then the field (column) or record (row) is returned, but only the first cell in the field or record is displayed. Of course, if a reference is made to a field_name and a record_number, there is only one cell that can be returned and displayed.

You can use REMOTE in conjunction with other functions to perform operations on rows and columns of database objects. For example, the formula =SUM(REMOTE("Company.adb", "Salary!!!GrossPay") might sum the gross pay of all the employees in a company.

Examples:

REMOTE("c:\ability\samples\mydata.adb", "Employee")

returns the table Employee from the database mydata.adb.

REMOTE("c:\ability\samples\mydata.adb", "Customer!ByCity")

returns the table Customer from the database mydata.adb with saved sort order ByCity applied.

REMOTE("c:\ability\samples\mydata.adb", "Customer!ByCity!UpperHalf")

returns the table Customer from the database mydata.adb with saved filter UpperHalf and sort order ByCity applied.

REMOTE("c:\ability\samples\mydata.adb", "Customer!ByCity!UpperHalf!Company")

returns the field Company from table Customer from the database mydata.adb with saved filter UpperHalf and sort order ByCity applied.

REMOTE("c:\ability\samples\mydata.adb", "Customer!ByCity!UpperHalf!Company!12")

returns the 12th record from field Company from table Customer from the database mydata.adb, with saved filter UpperHalf and sort order ByCity applied.

REMOTE("c:\ability\samples\mydata.adb", "Employee!!!LAST_NAME!3")

returns the third last name from table Employee from the database mydata.adb.

REMOTE("c:\ability\samples\mydata.adb", "QCalc!!!!2")

returns the second record from the query QCalc from the database mydata.adb.

REMOTE("c:\ability\samples\mydata.adb", "JoinOnCustID!!HANOP")

returns the relation JoinOnCustID from the database mydata.adb with applied filter HANOP.

REMOTE("c:\ability\samples\mydata.adb", "SELECT Quantity FROM Orders WHERE CustID = BERGS")

returns the range of quantities from Orders table from the database mydata.adb where CustID = BERGS.

See also:

Other remote functions

SSGET(spreadsheet, reference)

The SSGET function returns the contents of a cell (or cells) from a spreadsheet. SSGET can be used from within Write documents, Spreadsheet files and Database forms.

spreadsheet is the name of a spreadsheet and must include the full path if not in the same directory as the open document. The spreadsheet extension (.aws) is not required.

reference can be a cell, range or named cell or range.

Both parameters need to be surrounded by quotes if supplied as direct references into the function.

For example:

=SSGET("sales", "A1")

returns the contents of cell A1 in a spreadsheet called sales, which must reside in the current directory.

=SSGET(B1, B2)

would return the same if cells B1 and B2 simply contain the text ***sales*** and ***A1*** respectively.

=SUM(SSGET("sales", "C1..C10"))

returns the sum of cells C1 through C10. If C1 through C10 were named as a range called totalsales, then the following function would be identical to the above:

=SUM(SSGET("sales", "totalsales"))

To return a cell from a spreadsheet in a specific directory, use the full path as follows:

=SSGET("C:\My Documents\sales", "A1")

See also:

Linking to Database

Linking to Write

Other remote functions

WPGET(document, fieldname)

The WPGET function returns the contents of a field from a Write document. WPGET can be used from within Write documents, Spreadsheet files and Database forms.

document is the name of a document and must include the full path if not in the same directory as the open document. The document extension (.aww) is not required.

fieldname is the name of the required field.

Both parameters need to be surrounded by quotes if supplied as direct references into the function.

For example:

=WPGET("myreport", "myfield")

returns the contents of the field named myfield in the Write document called myreport, which must reside in the current directory.

=WPGET(targetdoc, targetfld)

would return the same if the local fields targetdoc and targetfld simply contain the text ***myreport*** and ***myfield*** respectively.

To return a field from a document in a specific directory, use the full path as follows:

=WPGET("C:\My Documents\myreport", "myfield")

See also:

Linking to Database

Linking to Spreadsheet

Other remote functions