



## Calculator Reference

*A simple calculator or a powerful programmable computer*

### Contents

Basic calculator features .....	4
The calculator screen .....	4
Calculator buttons .....	5
(Shift Delete) Clear entry .....	5
(= Key) Solution .....	5
(Alt D) Date format .....	5
(Alt \$)Dollar format .....	5
(Alt I) Insert function .....	5
(F1 or F10) Help .....	5
Using the Memory and Result lines.....	6
Adding comments .....	7
Basic features summary .....	7
Advanced calculator features.....	8
Using built in functions .....	8
It all begins with the Ins button .....	9
Built in parameter prompts .....	9
Turning of the prompts .....	11
Create and edit built in functions.....	11
Function attributes .....	11
Adding comments .....	12
Making user prompts.....	12
Tips for making new function.....	14
Inserting functions summary .....	14
A more detailed example.....	15
Other calculator base functions .....	17
Some background information .....	17
The tools in the box .....	17
Working with dates .....	18
Working with strings to get the job done.....	18
Convert numbers to meaningful text .....	19
Using all the tools together .....	19
Summary.....	21

Advanced documentation.....	23
Advanced documentation contents.....	23
Expressions .....	23
Overview.....	23
Expression evaluation.....	23
Operators.....	24
Arithmetic Operators.....	24
The Concatenation Operator.....	24
Logical Operators.....	24
Conditional Operators.....	24
Boolean Operators.....	24
Combined Boolean operators.....	24
Constants.....	25
Numeric Constants.....	25
String Constants.....	25
Types of Expressions.....	26
Numeric Expressions.....	26
String Expressions.....	26
Logical Expressions.....	26
Command reference.....	27
ABS (return absolute value).....	27
ACOS (Returns inverse cosine).....	27
ASIN (return arcsine).....	27
ATAN (return arctangent).....	28
BAND (return bitwise AND).....	28
BOR (return bitwise OR).....	29
BXOR (return bitwise exclusive OR).....	29
CLIP (return string without trailing spaces).....	30
CLOCK (return system time).....	30
COS (return cosine).....	30
DATE (return standard date).....	31
DAY (return day of month).....	32
DAYNAME (return a string of the week day name).....	32
FORMAT (return formatted numbers into a picture).....	33
INT (truncate fraction).....	33
INSTRING (return substring position).....	34
LEFT (return left justified string).....	35
LEN (return length of string).....	35
LOG10 (return base 10 logarithm).....	35
LOGE (return natural logarithm).....	36
LOWER (return lower case).....	36
MONTH (return month of date).....	36
RIGHT (return right justified string).....	37
ROUND (return rounded number).....	37
SIN (return sine).....	38
SQRT (return square root).....	38
SUB (return substring of string).....	39
TAN (return tangent).....	40
TODAY (return system date).....	40
UPPER (return upper case).....	40
YEAR (return year of date).....	41
Picture Tokens.....	42

All Aboard – Calculator reference

Numeric and Currency Pictures .....	42
Scientific Notation Pictures .....	44
String Pictures .....	44
Date Pictures .....	45
Time Pictures.....	46
Pattern Pictures .....	47

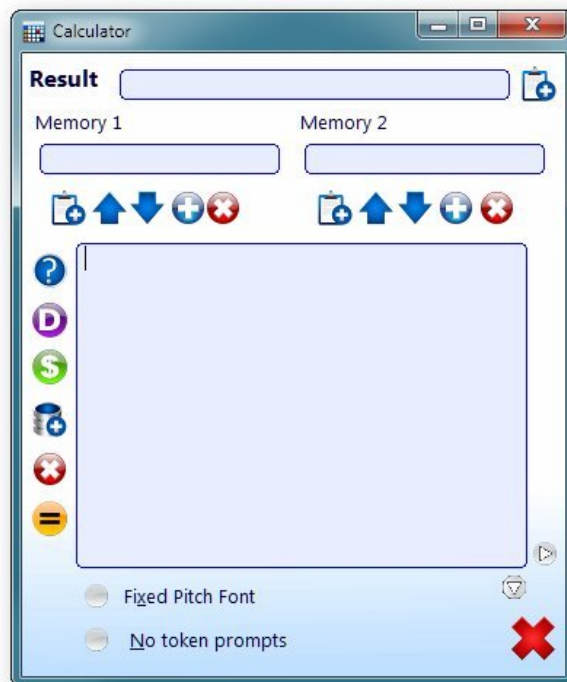
# Basic calculator features

*A quick reference to basic operations*

## The calculator screen

Select the Calculator icon from the main toolbar. The calculator will share the current drawer colors and theme attributes. You may open as many calculators as you desire. Launching the calculator from the Systray is also supported.

A traditional calculator has many buttons but All Aboard provides a calculator



workspace that you can type your mathematical expression and then lets you calculate the answer as well as edit the formula.

Pressing your = key or clicking the = icon with the mouse will display the result above in the Result line.







This uses the most basic features of the calculator. You may use all the common math notation symbols:

- + Plus key to perform addition
- Minus key to perform subtraction
- \* Asterisk key to perform multiplication
- / forward slash to perform division
- ^ Carat to perform exponentiation
- & String concatenation
- % Modulus (remainder from a division)

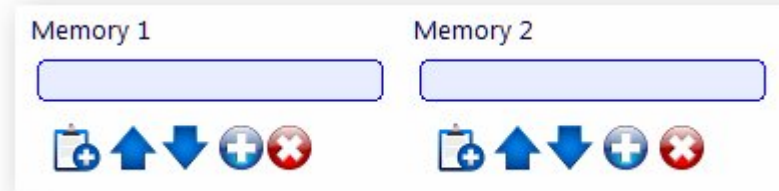
The current calculator workspace is limited to formula length of 6,000 characters. For long expressions you may require left and right parenthesis to group the computation properly. Computing your formula ignores all spaces, tabs, and line returns. This lets you type in a column or type across the entire line in a way that makes it easier for you to read. The workspace can be resized to fit your screen, as you require.

## Calculator buttons






If you type a formula and press the equal key you will see your answer in the Result line. The rest of the buttons on the calculator allow more advanced features. The following are the features supported by these buttons.

	<b>(Shift Delete) Clear entry</b> The CE icon when clicked or selected with Alt E clears the current formula pad. Select a second time to clear the result line too.
	<b>(= Key) Solution</b> The equal icon and equal key request the computer to solve your formula. The solution is displayed on the result line.
	<b>(Alt D) Date format</b> The date Format icon will translate the result numeric date from a raw number to a date format easily understood..
	<b>(Alt \$)Dollar format</b> Dollar format icon or Alt \$ will replace the result line formatted to two decimal places and add a dollar sign.
	<b>(Alt I) Insert function</b> Insert function inserts a built in function to the calculator workspace. See advanced features.
	<b>(F1 or F10) Help</b> This document is displayed.

## Using the Memory and Result lines



Within any formula you may use the contents of either the memory or Results lines in a new formula by typing in the variable name of Results or Memory, Memory1, or Memory2. Version 1.80 added a 2<sup>nd</sup> Memory so the old memory is still maintained as well as the new Memory1 and Memory2. The following chart shows the memory related icons and the hot keys associated with the icons.

Icon	Hot key Memory 1	Hot key Memory 2	Description
	Alt 1	Shift Alt 1	Copy the contents to the clip board
	Alt T	Alt Shift T	Move the current result to memory
	Alt L	Alt Shift L	Move the memory contents to result
	Alt Plus	Alt Shift Plus	Add the result to Memory
	Alt Delete	Alt Shift Delete	Clear memory

These are predefined variables for your use. You may even use them to develop recursive formulas such as the following scenario:

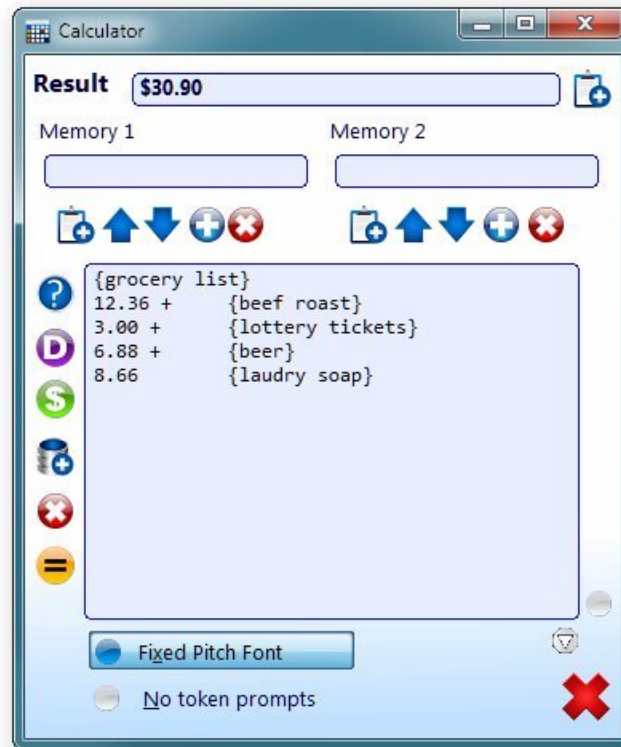
Result	100
Memory	1
Calculator workspace	Memory + Results

## All Aboard – Calculator reference

Each time the equal key is pressed the formula will compute the result causing the Display to increment the result by one each time the equal key is pressed. You may use these variables any place you might use a number.

### Adding comments

You may comments any place inside the formula as long as you always start them with a curly brace and end them with a curly brace. The comment may span multiple lines and neither the curly braces are everything in between will be used in the computation. The above is a grocery list with the items identified. Comments are not required but are often useful.



### Basic features summary

Use the formula pad to type any numeric expression then press the = key or button. To sum a list of number just type them separated by a plus sign then press equal. You may choose to add enter keys or tabs to make it easier to read and proof. If any one number is incorrect just edit the number and reselect the equal key or button. Copy the result to the clipboard to paste it in another application. With the basic features you can perform all the functions on any basic calculator.

Once you have gained comfort with basic operations you can explore the advanced features of built in functions.

# Advanced calculator features

*A reference to advanced operations*

## Using built in functions

The insert function button permits you to use more advanced function within your own formulas you build in the calculator workspace. You can still write your own formula by typing it too. Using advanced features you can nest functions and perform more complex solutions such as trigonometry, calendar mathematics, and string manipulations with less typing.

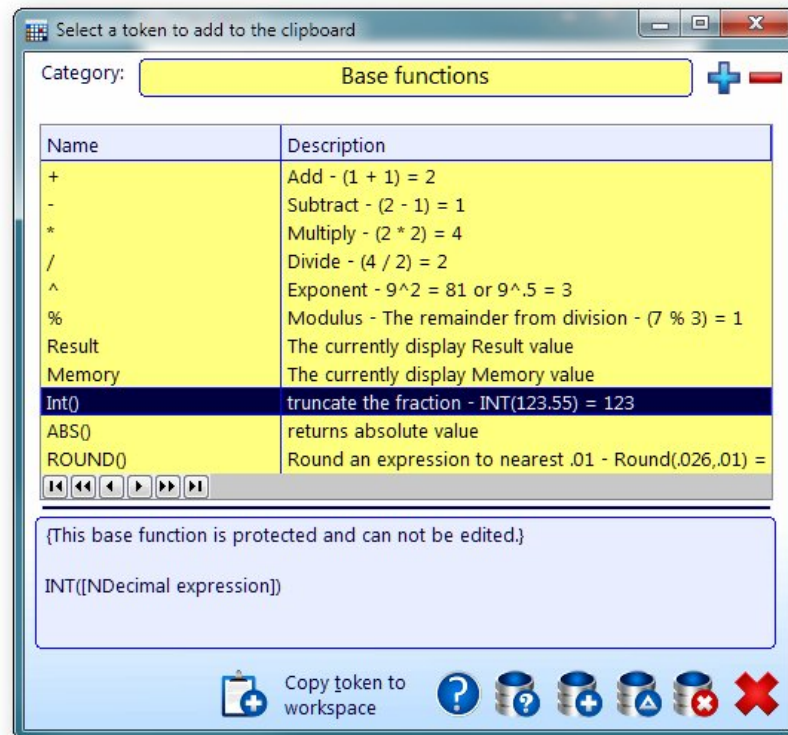
Before we can begin the advanced discussion we need to cover some of the basic formula requirements that set the limit of what you can do. Each time you press the solution button (equal key) the formula in the calculator workspace is solved as one expression that yields only one result. There are often many intermediate results but in the end there is only one. You do not have the ability to write lines of code as you do with a programming language but you will see you can still accomplish a lot of power and write easy to use equations. In this way you are only writing one line of computer code – it can be a big one however!

The All Aboard Calculator incorporates a run time evaluation library created by Soft Velocity makers of the Clarion programming family of development tools. All parameters and functions start as text values implicitly converted from string to real numbers, new strings, integers, binary, hexadecimal, and octal formats as demanded by the functions that contain them. All results are string data that can then be passed as input to another function. In this way we can mix and nest function to perform many calculations. It also means the whole complexity of converting data from one type to another is all automatic.

## It all begins with the Ins button

By selecting the Insert function button (Alt I) a Screen of functions appears.

Functions are categorized just like note records are classified into categories. They display in color based on the defined category colors that you can change. Selection of a row in the table highlights the selected function and displays its formula below the list box. Double clicking will paste the formula into the calculator workspace. The area below the list shows just what will be added to your formula.



### Built in parameter prompts

In the above screen you will see the INT() function has been selected on the screen and the formula below shows as:

INT([NDecimal expression])

The INT() function accepts any value and truncates the fractional part of the number. The number 123.45 is truncated to 123. To use this in a formula you could type:

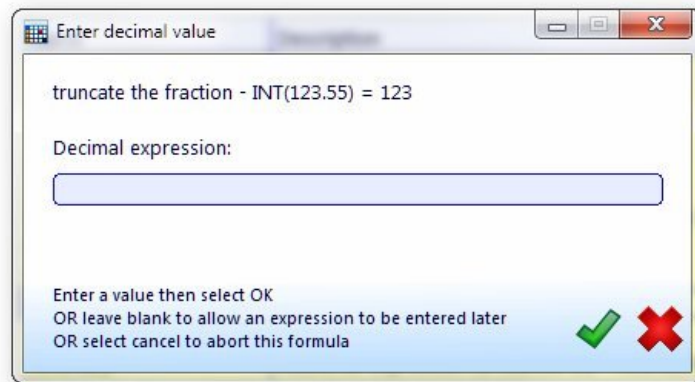
INT(123.45)

The formula you see above includes some extra characters:

[NDecimal expression]

## All Aboard – Calculator reference

This is the prompt component to the built in function. When you double click the INT() function in the list you will be displayed a second screen.



The Window title is instructing you to enter a decimal value. The very first letter of the text in the parameter determines what type of data the formula requires for this particular parameter. In this case the "N" signifies that the data type is decimal.

The first line of text in the window is the Description attribute of the function. It tells the user what function they are using.

The prompt text above the entry field is directly from the formula prompt text. Text enclosed in square brackets makes the "Decimal expression:" prompt you see in the above screen. The text has a colon added automatically.

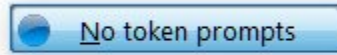
The entry field is a free form entry field. You could enter a number or you may need to enter a complex expression that eventually will be used in the INT() function.

You are then permitted to Click the OK button (enter key) and leave it blank. This will paste the whole formula into the calculator and it will be up to you to replace the prompt text with appropriate values. You would use this when building complex formulas using base functions. The second option is to enter a number or an expression and then it will substitute the prompt text for your data entry. The third option is to cancel (escape key) and nothing will be inserted into the calculator workspace.

Other functions may have as many prompts as required for the formula. When the formula gets complicated, prompting for them makes it easy for anyone to perform complex computations.

### Turning of the prompts

For the most simple base functions like INT() you may quickly grow tired of the prompts. You have two choices. The first option is to just type the function in the



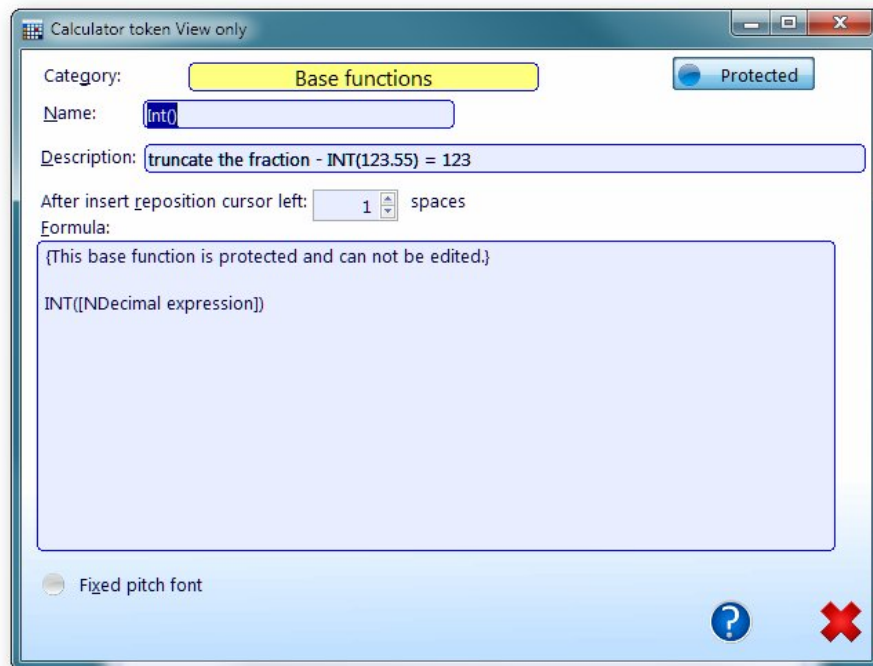
calculator workspace manually and not use the Insert function button OR click on the “No calculator token prompts” button on the main calculator screen.

you may then manually type the contents into the calculator workspace with out a prompt. All non protected base functions will still continue to prompt you.

### Create and edit built in functions

The add, change, and delete buttons at the bottom of the screen provide access to the attributes of the built in functions and / or let you make new functions. Most of the functions installed are protected and cannot be modified. Formulas you create can be added to the list.

### Function attributes



A function is comprised of the following parts:

- |             |   |
|-------------|---|
| Name        | Short description of the function. This appears in the left most column of the display list. Use this when you make a new function so it is easier to find. |
| Description | This is a longer description with an example if possible to act as documentation for the user. It displays at the top of any user prompt fields you create. |

## All Aboard – Calculator reference

Formula	The actual formula inserted into the calculator pad. This is the actual code that is used and may or may not contain user prompts. Extra spaces and enter keys can be added to make the formula easier to read. Extra characters are never inserted into the calculator workspace. Spaces used inside the square brackets will remain and should be added to make the prompt look correctly. Your formula can be up to 6000 characters long including any documentation or prompts.
Cursor reposition	The number of spaces to move the cursor to the left after insertion allows the user to enter the function parameters by positioning the cursor where the first parameter belongs. For example, the INT() built in command has a value of 1 so the cursor will locate between the left and right parenthesis. This is where you would add a number or an expression. Leaving this value blank or zero will locate the cursor at the end of the command after insertion.
Category	The category makes it easier to find functions. You may create new categories for your own functions too. The category name and description are only for organization and documentation. Creating your own functions and categories you perform regularly saves you time and improves your accuracy. You may choose to change the existing category colors but you may not delete the base categories or edit a built in functions when it shows the protected icon in the upper right corner. Making your own versions of these same functions is allowed.

## Adding comments

You may add comments as you like to the base function. Comments will be displayed in the browse and in the edit screen. Comments will not be inserted into the calculator work space. Prior to inserting the function in the calculator workspaces all comments, line feeds and carriage return characters will be removed. This means when you write complex functions you can store notes and documentation in the function but not clutter the calculator workspace when you actually use it. Base functions that ship with All Aboard are noted as being protected and not editable. You also see the protected icon at the top of the edit screen.

## Making user prompts

The format for a user prompt is shown below:

[<type of data><The prompt that the user sees>]

The first and last characters must be square brackets. You may not use square brackets as part of the prompt or for any other purpose.

<type of data> The first letter in the user prompt will change the window title that pops up to ask for user prompted data. Using an unrecognized type will ignore the type and let you enter anything. Using the wrong type won't restrict what you can enter but will confuse a user of your function. The following are valid types and may be upper or lower case:

N Numeric data that can have a decimal point

## All Aboard – Calculator reference

- I Integers (whole numbers) with no decimal point
- A An angle that can be a decimal value in degrees
- S String data that can include letters or number

<The prompt that the user sees> Starting with the second character and up to but not including the closing square bracket is prompt displayed to the user of your base function. These prompts are substituted after the user supplies the data for that prompt in all instances.

A user prompt that is repeated in more than one place in the formula can be used so the user only has to enter the data one time. You must make sure that these prompts are completely identical because they are case sensitive.

When prompting for a pure text string such as the following function:

```
UPPER([SEnter your name])
```

If the user enters:

```
John Smith
```

The function will fail because the UPPER() base command requires a text string for input or an expression. The error is because the text "John" is not seen as text but as a variable or function. The error displayed in the result will be "! BIND has not been called for John". If you expect only pure text and never want a text expression it is better to define your function as follows:

```
UPPER('SEnter your name')
```

In this revision we added single quotes on either side of the user prompt so whatever the user enters will be treated as pure text. If you expect to enter other functions as an answer to the prompt such as:

```
SUB('John Smith',1,4)
```

This would yield the following pasted in the calculator

```
UPPER('SUB('John Smith',1,4)')
```

The correct way it should look is like this:

```
UPPER(SUB('John Smith',1,4))
```

You only need to be concerned with this issue when you are working with text. The last example following this chapter and the command reference documentation at the end will provide more details about using text and string data in the calculator. Working with quotes is the important issue when you use the calculator to compute strings. For numbers only it's never a problem.

## Tips for making new function

Create a manual example in the calculator itself to test your ideas. When the example works, copy and paste it into a formula and add the short name and description. Then test it again to make sure it still works. As a last step replace all the fixed values with user prompts and test it again.

All text and string data must be enclosed with single straight quotes (the apostrophe key). If a quote itself is part of the data you must precede the quote with another quote. See the detailed documentation in the last chapter for more details about making string expressions. Working with quotes can be a bit confusing so you need to test each step as you go along.

All trigonometric functions SIN(), COS, and TAN() operate in radians and ASIN(), ACOS, and ATAN() return radians not degrees. Use the conversion variable DEG2RAD and RAD2DEG to switch between degrees and radians. Several of the included area formulas already installed in All Aboard actually require radians. Keeping the two straight will make sure your results are accurate.

The last chapters in this manual provide subtle details about the functions reviewing that material may show you ideas and allow you to get the most out of your efforts.

## Inserting functions summary

The ability to insert built in functions opens the door to more complex solutions with less typing. By using the prompted parameters anyone can insert the proper data where it belongs. Once you have inserted functions you can still manually add and edit more

nested functions and your own numbers before you attempt to solve the solution. All the built in functions are documented in the last chapter of the calculator documentation. The next two sections will provide some real examples to help you get the ideas about writing complex functions for the calculator.

# A more detailed example

*A formula for computing the area of a triangle from the length of 3 sides*

One of the more complex functions included with the calculator is the formula for computing the area of any triangle. The formula is derived from Heron's formula combined with the Law of Cosines. You enter only the lengths of the three sides in any order and the area formula is written to your calculator and only requires you to press the equal key or button. Let's take a closer look at this function included in the Trigonometry category.

Step 1 From a blank calculator select the Ins button (or Alt I keys).

Step 2 From the list of functions scroll down and select:

Double click it, or click then select the green checkmark or click and press the Enter key. Any of these three methods will select the function.

ACOS()	Arccosine of an angle using radians
ATAN()	Arctangent of an angle using radians
Area 3 sides	Area of a triangle from 3 sides
Area of a circle	From the length of the radius compute the area

Step 3 A little background on this function will help you understand the rest of the example. To compute the area of a triangle you need to know the length three sides of triangle. You could type in three random numbers but you may find out that sometimes it won't compute properly. In order for the triangle to be a real triangle the sides must connect at the three vertices. If any side is longer than the sum of the lengths of the other two it cannot be a triangle connected at 3 vertices. Our function has to assume that you are not going to enter invalid data. As long as the data entered is real then our formula will work. Invalid data will always yield zero.

The formula will now prompt us to enter the length of the three sides.

Enter decimal value

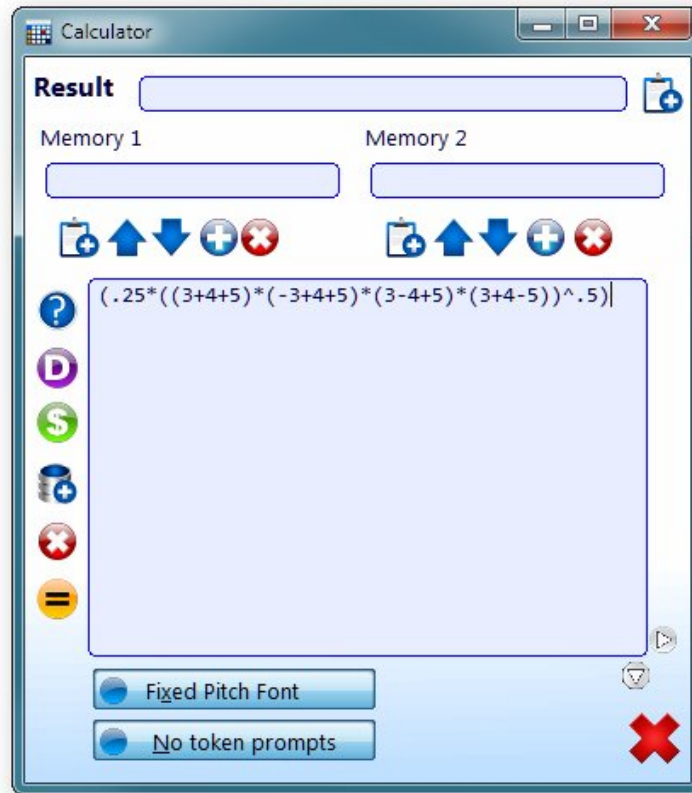
Area of a triangle from 3 sides

Side a:

## All Aboard – Calculator reference

For each side a, b, and c we are prompted to enter a length. For our example I'll use the values 3, 4, and 5. These values will define a proper triangle and will let us complete our example. I will also show an alternate way to solve this triangle and prove our formula really works.

Step 4 After we enter the last side we return to the calculator workspace our formula appears.



The above shows how the calculator took our three parameters and substituted them into a formula defined in the calculator. We could have memorized this formula and typed it in the calculator workspace manually. This is not a formula you will easily memorize. The calculator uses three side lengths you enter but inserts each side length four times each. Your typing efficiency is now 4 times greater using the calculator.

Step 6 The last step is to press the equal button or the equal hot key to display the result. The result is 6 of course. We know that a triangle with sides 3, 4, 5 units long form a perfect right triangle and the area of a right triangle is equal to the two short side multiplied together and divided by 2. So the check for this example is  $(3*4)/2$ . If you press the CE button (Alt E) you can enter the check formula as well and get the same result of 6. Using both methods we can now be sure our formula works properly.

Step 7 We have now used a formula one time and that demonstrates the usefulness of our triangle function but we can go farther. Suppose we need to compute the total area of 25 triangular areas so we can estimate the amount of paint we need to buy. In our above example enter a+ key in the calculator workspace after the previous formula and insert another triangle. You can then compute the area of 2 triangles added together. Continue adding more triangles and in this way sum many triangles. You also have the added benefit of copying your formula and pasting it in a note record in an appropriate All Aboard drawer to save it for later. Add comments to the formula about which triangle is for what and now you have the documented numbers with the formula that computes a proper total.

## Other calculator base functions

*Dates and text strings (open a calculator and follow along)*

Math functions are a handy feature of the calculator but we can do other operations as well. The calculator can operate on other types of data and we will now examine a second type of calculation that you might actually use in All Aboard.

### Some background information

All projects start with some idea of a required task to be done and the rules to follow. Our next example is a real problem I had while creating All Aboard. I wanted to generate a default title to be used when creating log entries in my purple drawer. If you explore the All Aboard configuration system you will find that you can designate a default title for a log entry different for each drawer. There is a mysterious check box called "Default description is an expression". This feature allows you to create an expression just like a calculator formula that is executed each time a log entry is added to a specific drawer. Each drawer can have it's own different expression or no expression. It is limited to only 254 characters so it can not be as long and cannot contain user prompts.

The mission for this default title started with a need to create a weekly time log I always create on Monday for the week ending on Sunday the previous day. It seemed to me there should be a way to do it automatically they would all look uniform and always have a Sunday date.

### The tools in the box

When you explore the calculator tool box you find text handling tools as well as some calendar handling tools. With this in mind I was ready to make the formula expression that could make exactly what I needed. The design was to make the following string:

Week ending MM/DD/YYYY

I could have just typed the above into the default description for the logs in my purple drawer and that is exactly what they would look like. This is close but I wanted the expression to substitute the MM, DD, and YYYY for the real month, day.

## Working with dates

All Aboard has a built in function you can use in your calculator to compute calendar dates. It's called:

TODAY()

The TODAY() function returns the current date stored in your computer. In addition the function CLOCK() will return time. If you try it in your calculator this is what you would find as the result on the date this documentation was first written - September 26,2007. Your date won't be the same unless your computer is set wrong.

75512

Interesting, but you may be thinking "How does that help me and what does 75512 mean?" Computers don't always make sense but they do have a standard way of dealing with dates that is quite accurate and useful. To understand what 75512 means I'm going to tell you that a computer date of 4 means January 1, 1801 and it was a Thursday. From here we can compute all the rest of our days longer than any of us will survive.

To work our solution we need to know not only the date but the day of the week as well. One of the interesting functions is the Modulus function. It is written like the following:

$75512 \% 7$

The answer you can compute in your calculator workspace by typing it in right now. You will see the correct result is 3. When you divide this number by 7 the remainder is 3. Modulus throws away the answer to the division but returns the remainder instead. In our daily lives we don't often need the remainder without the result but in computing we commonly do. Any number divided by 7 will always have a modulus of between zero and six. 7 divided by 7 has a remainder of zero and 3 divided by 7 has a remainder (a Modulus) of 3. When combined with the TODAY() function it tells us the one thing we really need – the day of the week. If the modulus of the date is zero we know the day of the week is Sunday and if the modulus is 6 we know it is Saturday and the days in between follow in order.

DayName(Today())

Enter the above into your calculator and press the equal key. You can type this everyday and it will always tell you the day of the week. It does the modulus function for you and returns back the day of the week as text based on your computers calendar. Your calculator now computes text as well as dates! This is not what you normally consider a function of a calculator.

All this exercise is to get to know how dates in the computer work relative to the calculator. It means we now have a way to compute the current date and in turn the day of the week, We are in great shape to take on this project.

## Working with strings to get the job done

Our mission also requires us to assemble a text string that we will use for our default log description. The way we add text strings is not too unlike adding numbers together. Type the following into your calculator workspace:

## All Aboard – Calculator reference

`'Week ending ' & TODAY()`

The above is a text “expression”. Actually everything you enter into the calculator workspace is an expression. Note that I added a space after the letter g and before the final quote. Copying the text from this document requires you to replace the open and closing quotes with plain straight quotes. It will display an answer in the result line:

Week ending 75512

Your date won't be the same as above though. Now this is getting more like what we want to see but not quite. Let's recap the process of making strings then move on. The text 'Week ending ' is enclosed in single quotation marks. Notice there is a space after the letter g. The next part is the ampersand symbol. This is called a concatenation

operator. It's like a plus sign for joining text expressions together. It joins the results from the TODAY() function we showed earlier with the text string we just demonstrated.

## Convert numbers to meaningful text

Just having the correct number is not always the total mission. It is often a requirement that the number look properly in addition to being accurate. The date is the best example since no one knows what 75512 means as far as a date on the calendar is concerned. Fortunately, we have a powerful function called FORMAT(). A fully detailed section in the back of this document will explain a lot of the options for FORMAT(). There are many formats for all sorts of numbers, dates, text and time. We will use just one for this project.

FORMAT() requires just two parameters. The first is the data to format and the second is called a “picture”. A picture is the set of instructions for reformatting the data. The

FORMAT() function will return formatted data as text the way we want it to look. Let's examine the following format.

`FORMAT(75512,@D2)`

If you paste this into the calculator you will see it displays the proper date - 9/26/2007. This is what we need to make a date number look like date text.

## Using all the tools together

Now we are ready to write the expression that will work the way we want. The following is the first version.

`'Week ending' & FORMAT(TODAY()-1,@D2)`

If you cut and paste be sure you are using single quote characters. It should now display a result that looks like this (except with today's date not the following old date):

Week ending 9/25/2007

The above expression takes the current date and subtracts 1 day and formats it as a date and appends it to the string 'Week ending '. Because I said I always make the entry on Monday, this will display the date for Sunday. Sorry, I but forgot to tell you something. I have asked you to write an expression but I failed to provide all the information. Sometimes I don't make the log on Monday. I might not do it until

## All Aboard – Calculator reference

Wednesday. We now have to make another adjustment and for that we need to understand the days of the week from a computing perspective. If you already know your days of the week in order by day you are ready for this part.

If you read back to the early discussion about days of the week, you will recall that when the Date % 7 is equal to zero we know the date is a Sunday. We are now going to add a little trick to make sure I never get a date that isn't a Sunday.

```
'Week ending' & FORMAT(TODAY() - (TODAY()%7) ,@D2)
```

I'll now explain how this works to return the prior Sunday no matter what the date is currently. If the Modulus of any date can only be 0 through 6 and Sunday is always a zero; we can subtract the modulus of the current date from the date to obtain the date of the prior Sunday unless the current date is a Sunday and we will get the same date. Subtracting the modulus 7 of any date from the date is always going to compute "last" Sunday unless it already is Sunday. That will work for our solution just fine.

Just when we thought we had the mission completed it turns out I forgot something else. I don't always do the log on Monday or even on Tuesday. Sometimes I even do it early! We need some logic to help get it right. The real complete rule is if I make an entry on Monday, Tuesday or Wednesday then I'm late! If I make it Thursday, Friday or Saturday then I'm early for the week ahead. If I make it on Sunday then that was the correct date. Now no matter what the date is we can now know what to do. We can be sure I can't change my mind again.

We really need some tool we have not discussed yet and it is a tool called CHOOSE(). CHOOSE() solves a single logical expression that determines which of two other expressions then apply. In computer programming it's called a branch. This one tool alone is all we have to work within our calculator toolbox to perform a "logical branch".

To speed up the example I'm going to tell you that the logical expression we want to use is:

```
TODAY() % 7 < 4
```

The above rule says anytime the date modulus 7 is less than 4 (0,1,2,or 3) then the condition is TRUE otherwise it is FALSE. To us, a modulus value of zero to three means the current date is Sunday through Wednesday otherwise it must be Thursday through Saturday. Let's see what the new formula looks like:

```
'Week ending ' & FORMAT(  
TODAY()  
- TODAY() % 7  
+ CHOOSE( TODAY() % 7 < 4 , 0, 7)    {<-the new part is here}  
,@D2)
```

I have added line breaks and a comment in the formula so it is easier to read. As written this will actually work even with the comment added. Let us walk through the parts of the formula:

'Week ending ' is a single text expression we always want no matter what. Notice we include a space after the letter g

FORMAT( begins the second part of the text we want to make because we are going to compute the date for our text then convert it into a readable

date formatted string. What goes inside is what we will compute, but the result of that numeric expression is what the FORMAT() function requires.

,@D2) concludes the FORMAT() function by specifying the type of formatting we want. In our final expression we want the date displayed in the format mm/dd/yyyy so that format picture we require is @D2. The reference chapter at the end covers all the various format pictures. There are many date formats and maybe you would like to try others.

TODAY() - TODAY() % 7 + CHOOSE( TODAY() % 7 < 4, 0, 7) is the computed "numeric expression" that is our date we want to display. Let's take that apart into the sub expressions that make this large expression:

CHOOSE(TODAY() % 7 < 4, 0, 7) The CHOOSE() command is the new numeric expression we added. It will compute the modulus of the current date then compare the result to 4. If it less than 4 the CHOOSE result is zero. If the CHOOSE() evaluated expression is false the result is the numeric expression 7. This form of CHOOSE() returns the first expression value if true and the second expression value if false. See the command reference section for details about CHOOSE(). Choose only operates on a "logical expression" but can return either a text or a numeric result. The result can also be another expression and could include an inner CHOOSE() expression thus letting us perform many branches and sub branches. With practice it opens up a lot of different computations.

Here is a recap of how CHOOSE works for our problem. If we add 0 to the date of "last Sunday" it still is the same date, but if we add 7 it will be Sunday 7 days later. We have the problem of being early or late solved using CHOOSE().

What makes this all work are the rules of precedence (see command reference chapter). The CHOOSE() command needs to be resolved before the rest of the expression can be resolved so the evaluation begins by examining the expression inside the choose command then works its way out through the rest of the function. Using parentheses can help you keep clear in your mind when you write these functions as well as define ambiguous expressions.

This expression can now be cut and pasted into the Log record configuration setting followed by selection of the "default description is an expression". Notice we have no user prompts because we will only need the current date. We can generate our log descriptions week after week on any date and always get the correct date. This expression will resolve leap year too!

## Summary

The critical issue in any project is to reduce the problem to its most basic rules after you gather all the information you need. When you understand the mission you can better solve the problem. Writing functions to make computations work faster and accurate are both good reasons to spend time learning the All Aboard Calculator features.

## All Aboard – Calculator reference

We often need to pull several functions together to get not only the correct answer but the correct answer formatted so it looks correct too. You can make your own default Log entry descriptions using the calculator to design and test the expression and then paste it in the configuration Log setting so it works for you all the time.

The Calculator is a fun tool to play with math and other expressions since the tool itself can test your solutions. If you find a useful expression you can save it to the calculator for use in the future. Write a good function one time and spend the rest of your days being lazy. It's what we call being a lazy programmer. Sometimes being lazy is a good thing.

If you write fancy functions you would like to share with others please send them to [AllAboard@ODStrategies.Org](mailto:AllAboard@ODStrategies.Org). We will provide periodic updates and include the best user generated formulas and make sure your name is included in the description.

# Advanced documentation

*Not everything in life is easy but if you know where to find it it's not so bad*

## Advanced documentation contents

The following includes some discussions about the individual built in functions and what they do as well as a discussion about expressions in general and how to think about them. You don't have to memorize all this information but if you have specific job to do you may find this will help you get it done. Should you have more questions please do not hesitate to send them to technical support at All Aboard where we can attempt to answer them.

## Expressions

### Overview

An expression is a mathematical, string, or logical formula that produces a value.

Expressions may contain constant values, variables, and functions which return values, all connected by logical, arithmetic or string operators. The All Aboard Calculator allows you to create one large expression made up of smaller expressions. You now know the advanced part about the Calculator. Everything you do is an expression! Mixing them together can perform complex tasks.

### Expression evaluation

Expressions are evaluated in the standard algebraic order of operations. The precedence of operations is controlled by operator type and placement of parentheses. Each operation produces an (internal) intermediate value used in subsequent operations. Parentheses may be used to group operations within expressions. Using extra parentheses when they are not really needed is useful if it helps you make clear the order of how you want the expression evaluated. Expressions are evaluated beginning with the inner-most set of parentheses and working through to the outer-most set.

Precedence levels for expression evaluation, from highest to lowest, and left-to-right within each level, are:

Level	1	( )	Parenthetical Grouping
Level	2	-	Unary minus (Negative sign)
Level	3	Function call	Gets the RETURN value
Level	4	^	Exponentiation
Level	5	* / %	Multiplication, Division, Modulus
Level	6	+ -	Addition, Subtraction
Level	7	&	Concatenation
Level	8	= <>	Logical Comparisons
Level	9	NOT, AND, OR/XOR	Boolean expressions

Expressions may produce numeric values, string values, or logical values (true/false evaluation). An expression may contain no operators at all; it may be a single variable, constant value, or function call which returns a value.

## Operators

### Arithmetic Operators

An arithmetic operator combines two operands arithmetically to produce an intermediate value. The operators are:

+	Addition	A + B sum of A and B
-	Subtraction	A – B difference of A and B
*	Multiplication	A * B multiples A by B
/	Division	A / B divides A by B
^	Exponentiation	A ^ B raises A to power of B
%	Modulus division	A % B remainder of A divided by B

### The Concatenation Operator

The ampersand ( & ) concatenation operator is used to append one string or string variable to another. The length of the resulting string is the sum of the lengths of the two values being concatenated. Numeric data types may be concatenated with strings or other numeric variables or constants. In many cases, the CLIP function should be used to remove any trailing spaces from a string being concatenated to another string. When concatenating a number to a string, the number is converted to a string first then concatenated as a string. It's automatically done on your behalf.

### Logical Operators

A logical operator compares two operands or expressions and produces a true or false condition. There are two types of logical operators: conditional and Boolean. Conditional operators compare two values or expressions. Boolean operators connect string, numeric, or logical expressions together to determine true-false logic. Operators may be combined to create complex operators.

### Conditional Operators

- = Equal sign
- < Less than
- > Greater than

### Boolean Operators

- NOT Boolean (logical) NOT
- ~ Tilde (logical NOT)
- AND Boolean AND
- OR Boolean OR
- XOR Boolean eXclusive OR

### Combined Boolean operators

- <> Not equal
- ~= Not equal
- NOT= Not equal
- <= Less than or equal to
- =< Less than or equal to
- ~> Not greater than
- NOT> Not greater than
- >= Greater than or equal to
- => Greater than or equal to
- ~< Not less than

## All Aboard – Calculator reference

During logical evaluation, any non-zero numeric value or non-blank string value indicates a true condition, and a null (blank) string or zero numeric value indicates a false condition.

Example:

Logical expression	Result true when
A = B	A is equal to B
A < B	A is less than B
A > B	A is greater than B
A <> B, A ~= B, A NOT= B	A is not equal to B
A ~< B, A >= B, A NOT< B	A is not less than B
A ~> B, A <= B, A NOT> B	A is not greater than B
~ A, NOT A	A is null or zero
A AND B	A is true and B is true
A OR B	A is true, or B is true, or both are true
A XOR B	A is true or B is true, but not both.

## Constants

### Numeric Constants

Numeric constants are fixed numeric values. They may occur in expressions, and as parameters of functions. A numeric constant may be represented in decimal (base 10-- the default), binary (base 2), octal (base 8), hexadecimal (base 16), or scientific notation formats. Formatting characters, such as dollar signs and commas, are not permitted in numeric constants; only leading plus or minus signs and the decimal point are allowed. PI, RAD2DEG and DEG2RAD are built in All Aboard "numeric constants" you can use in math functions.

Decimal (base ten) numeric constants may contain an optional leading minus sign(hyphen character), an integer, and an optional decimal with a fractional component.

Binary (base two) numeric constants may contain an optional leading minus sign, the digits 0 and 1, and a terminating B or b character. Octal (base eight) numeric constants contain an optional leading minus sign, the digits 0 through 7, and a terminating O or o character. Hexadecimal (base sixteen) numeric constants contain an optional leading minus sign, the digits 0 through 9, alphabet characters A through F (representing the numbers 10 through 15) and a terminating H or h character. If the left-most character is a letter A through F, a leading zero must be used.

### String Constants

A string constant is a set of characters enclosed in single quotes (apostrophes). The maximum length of a string constant is 255 characters. Characters that cannot be entered from the keyboard may be inserted into a string constant by enclosing their ASCII character codes in angle brackets (<>). ASCII character codes may be represented in decimal, hexadecimal, binary, or octal numeric constant format.

In a string constant, a left angle bracket ( < ) initiates a scan for a right angle bracket. Therefore, to include a left angle bracket in a string constant requires two left angle brackets in succession. To include an apostrophe as part of the value inside a string constant requires two apostrophes in succession. Two apostrophes ( " ), with 0 characters (or just spaces) between them, represents a null, or blank, string.

## All Aboard – Calculator reference

Consecutive occurrences of the same character within a string constant may be represented by repeat count notation. The number of times the character is to be repeated is placed within curly braces ( { } ) immediately following the character to repeat. To include a left curly brace ( { ) as part of the value inside a string constant requires two left curly braces ( { { ) in succession.

The ampersand (&) is always valid in a string constant

Example:

'string constant'	A string constant
'It"s a girl!'	With embedded apostrophe
'<27,15>'	Using decimal ASCII codes
'A<<B'	With embedded left angle
'*{20}'	Twenty asterisks, repeat-count notation
''	null string
	A (blank)

## Types of Expressions

### Numeric Expressions

Numeric expressions may be used as parameters of functions. A numeric expression may contain arithmetic operators and the concatenation operator, but they may not contain logical operators. When used in a numeric expression, string constants and variables are converted to numeric intermediate values. If the concatenation operator is used, the intermediate value is converted to numeric after the concatenation occurs.

Example:

Count + 1	Add 1 to Count
(1 - N * N) / R	N times N subtracted from 1 then divided by R
305 & 7854555	Concatenate area code with phone number

### String Expressions

String expressions may be used as parameters of functions. String expressions may contain a single string or numeric variable, or a complex combination of sub-expressions, functions, and operations.

### Logical Expressions

Logical expressions evaluate true-false conditions. Control is determined by the final result (true or false) of the expression. Logical expressions are evaluated from left to right. The right operand of an AND, OR, or XOR logical expression will only be evaluated if it could affect the result. Parentheses should be used to eliminate ambiguous evaluation and to control evaluation precedence. The level or precedence for the logical operators is as follows:

Level 1	Conditional operators
Level 2	~, NOT
Level 3	AND
Level 4	OR, XOR

# Command reference

*The details about all the built in functions*

## ABS (return absolute value)

ABS(expression)

expression      A constant, variable, or expression.

The ABS function returns the absolute value of an expression. The absolute value of a number is always positive (or zero).

Example:

ABS(A - B)    absolute value of the difference  
ABS(-2)      returns positive 2  
ABS(2)        returns positive 2

## ACOS (Returns inverse cosine)

ACOS(expression)

expression      A numeric constant, variable, or expression for the value of the cosine.

The ACOS function returns the inverse cosine. The inverse of a cosine is the angle that produces the cosine. The return value is the angle in radians.  $\pi$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2\pi$  radians (or 360 degrees) in a circle. The variable Rad2Deg is predefined so you can multiply radians to return degrees.

Example:

ACOS(0) \* Rad2Deg      90 degrees

## ASIN (return arcsine)

ASIN(expression)

expression      A numeric constant, variable, or expression for the value of the sine.

The ASIN function returns the inverse sine. The inverse of a sine is the angle that produces the sine. The return value is the angle in radians.  $\pi$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2\pi$  radians (or 360 degrees) in a circle. The variable Rad2Deg is predefined so you can multiply radians to return degrees.

Example:

ASIN(1.0) \* Rad2Deg      =    90 degrees

## ATAN (return arctangent)

ATAN(expression)

expression      A numeric constant, variable, or expression for the value of the tangent.

The ATAN function returns the inverse tangent. The inverse of a tangent is the angle that produces the tangent. The return value is the angle in radians.  $\pi$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2\pi$  radians (or 360 degrees) in a circle. The variable Rad2Deg is predefined so you can multiply radians to return degrees.

Example:

ATAN(0) \* Rad2Deg    is 0 degrees

## BAND (return bitwise AND)

BAND(value, mask)

Value              A numeric constant, variable, or expression for the bit value to be compared to the bit mask. The value is converted to a LONG data type prior to the operation, if necessary.

Mask                A numeric constant, variable, or expression for the bit mask. The mask is converted to a LONG data type prior to the operation, if necessary.

The BAND function compares the value to the mask, performing a Boolean AND operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where the value and the mask both contain one (1), and zeroes in all other bit positions. BAND is usually used to determine whether an individual bit, or multiple bits, are on (1) or off (0) within a variable.

Example:

1011B = 11 decimal, 10B = 2 decimal

BAND(1011B, 10B) = 2 decimal ("B" signifies data is binary not decimal)

BAND(11, 2) = 2 decimal (the same as the above)

## BOR (return bitwise OR)

BOR(value, mask)

BOR Performs bitwise OR operation.

**Value** A numeric constant, variable, or expression for the bit value to be compared to the bit mask. The value is converted to a LONG data type prior to the operation, if necessary.

**Mask** A numeric constant, variable, or expression for the bit mask. The mask is converted to a LONG data type prior to the operation, if necessary.

The BOR function compares the value to the mask, performing a Boolean OR operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where the value, or the mask, or both, contain a one (1), and zeroes in all other bit positions.

BOR is usually used to unconditionally turn on (set to one), an individual bit, or multiple bits, within a variable.

Example:

1011B = 11 decimal, 10B = 2 decimal  
BOR(1011B, 10B) = 11 decimal ("B" signifies data is binary)  
BOR(11, 2) = 11 decimal (the same as the above)

## BXOR (return bitwise exclusive OR)

BXOR(value, mask)

BXOR Performs bitwise exclusive OR operation.

**Value** A numeric constant, variable, or expression for the bit value to be compared to the bit mask. The value is converted to a LONG data type prior to the operation, if necessary.

**Mask** A numeric constant, variable, or expression for the bit mask. The mask is converted to a LONG data type prior to the operation, if necessary.

The BXOR function compares the value to the mask, performing a Boolean XOR operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where either the value or the mask contain a one (1), but not both. Zeroes are returned in all bit positions where the bits in the value and mask are alike.

Example:

1011B = 11 decimal, 10B = 2 decimal  
BXOR(1011B, 10B) = 9 decimal ("B" signifies data is binary)  
BXOR(11, 2) = 9 decimal (the same as the above)

## CLIP (return string without trailing spaces)

CLIP(string)

CLIP Removes trailing spaces.

String            A string expression.

The CLIP function removes trailing spaces from a string. The return string is a substring with no trailing spaces. CLIP is frequently used with the concatenation operator in string expressions using STRING data types.

CLIP is not normally needed with CSTRING data types, since these have a terminating character. CLIP is also not normally needed with PSTRING data types, since these have a length byte.

When used in conjunction with the LEFT function, you can remove both leading and trailing spaces (frequently called ALLTRIM in other languages).

## CLOCK (return system time)

CLOCK()

The CLOCK function returns the time of day from the operating system time in standard time (expressed as hundredths of a second since midnight, plus one). Although the time is expressed to the nearest hundredth of a second, the system clock is only updated 18.2 times a second (approximately every 5.5 hundredths of a second).

Example:

```
ROUND(CLOCK()/100,1)    whole seconds since midnight
ROUND(CLOCK()/6000,1)  whole minutes since midnight
ROUND(CLOCK()/360000,1) whole hours since midnight
```

## COS (return cosine)

COS(radians)

COS Returns cosine.

Radians            A numeric constant, variable or expression for the angle in radians.  $p$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2p$  radians (or 360 degrees) in a circle.

The COS function returns the trigonometric cosine of an angle measured in radians. The cosine is the ratio of the length of the angle's adjacent side divided by the length of the hypotenuse. The variable Deg2Rad is predefined so you can multiply degrees to return radians.

Example:

```
COS(0 * Deg2Rad) = 1
```

## DATE (return standard date)

DATE(month, day, year)

DATE Return standard date.

Month	A positive numeric constant, variable, or expression for the month.
Day	A positive numeric constant, variable, or expression for the day of the month.
Year	A numeric constant, variable or expression for the year. The valid range for a year value is 00 through 99 (using "Intellidate" logic), or 1801 through 2099.

The DATE function returns a standard date for a given month, day, and year. The month and day parameters do allow positive out-of-range values (zero or negative values are invalid). A month value of 13 is interpreted as January of the next year. A day value of 32 in January is interpreted as the first of February. Consequently, DATE(12,32,97), DATE(13,1,97), and DATE(1,1,98) all produce the same result.

The century for a two-digit year parameter is resolved using the default "Intellidate" logic, which assumes the date falls in the range of the next 20 or previous 80 years from the current operating system date. For example, assuming the current year is 1998, if the year parameter is "15," the date returned is in the year 2015, and if the year parameter is "60," the date returned is in 1960.

Example:

Compute the first of the month based on the current date:

```
DATE(MONTH(TODAY()), 1, YEAR(TODAY()))
```

Compute the last date of the previous month

```
DATE(MONTH(TODAY()), 1, YEAR(TODAY())) - 1
```

## DAY (return day of month)

DAY(date)

DAY returns day of month as a number.

Date A numeric constant, variable, expression, or the label of a STRING, CSTRING, or PSTRING variable declared with a date picture token. The date must be a standard date. A variable declared with a date picture token is automatically converted to a standard date intermediate value.

The DAY function computes the day of the month (1 to 31) for a given standard date.

Example:

DAY(TODAY()) Get the day from today's date  
DAY(TODAY() + 2) Calculate the return day

## DAYNAME (return a string of the week day name)

DAYNAME(value)

DAYNAME Returns a string of the name of the day of the week based on the date passed.

Value A numeric expression for the the date to be converted.

The DAYNAME procedure returns a string based on the integer value of the date passed.

Return Data Type: STRING

Example:

DAYNAME(TODAY()) = the day of the week that is the current date

## FORMAT (return formatted numbers into a picture)

FORMAT(value, picture)

FORMAT Returns a formatted numeric string.

Value            A numeric expression for the value to be formatted.

Picture           A picture token or the label of a CSTRING variable containing a picture token.

The FORMAT procedure returns a numeric string formatted according to the picture parameter.

Return Data Type: STRING

Example:

See Picture Tokens in the final chapter

## INT (truncate fraction)

INT(expression)

INT returns an integer.

Expression       A numeric constant, variable, or expression.

The INT function returns the integer portion of a numeric expression. No rounding is performed, and the sign remains unchanged.

Example:

```
INT(25 /6)  4.0
INT(25 * .333)  8.0
```

## INSTRING (return substring position)

INSTRING(substring, string [,step] [,start])

INSTRING Searches for a substring in a string.

Substring	A string constant, variable, or expression that contains the string for which to search. You should CLIP a variable substring so INSTRING will not look for a match that contains the trailing spaces in the variable.
String	A string constant, or expression to be searched.
Step	A numeric constant, variable, or expression which specifies the step length of the search. A step of 1 will search for the substring beginning at every character in the string, a step of 2 starts at every other character, and so on. A negative step value will search from right to left within the string. If step is omitted, the step length defaults to the length of the substring.
Start	A numeric constant, variable, or expression which specifies where to begin the search of the string. If omitted, the search starts at the first character position.

The INSTRING procedure steps through a string, searching for the occurrence of a substring. If the substring is found, the procedure returns the step number on which the substring was found. If the substring is not found in the string, INSTRING returns zero.

INSTRING starts to search for substring from the start position in the string and moves forward with step until the substring is found, or the unchecked tail of the string is less than length of the substring. In the latter case, INSTRING returns zero. If the substring is found, the result is equal to the number of steps from the origin of the string to the found position. If the value of step is not equal to 1, the result is rounded up to the whole number of steps as follows:

$$\text{INT} ((\text{found position} - 1) / \text{step}) + 1$$

Example:

```
INSTRING('DEF', 'ABCDEFGHIJ', 1, 1)) returns 4
INSTRING('DEF', 'ABCDEFGHIJ', 1, 2)) returns 4
INSTRING('DEF', 'ABCDEFGHIJ', 1, 3)) returns 4
INSTRING('DEF', 'ABCDEFGHIJ', 1, 4)) returns 4
INSTRING('DEF', 'ABCDEFGHIJ', 1, 5)) returns 0
INSTRING('DEF', 'ABCDEFGHIJ', 2, 1)) returns 0
INSTRING('DEF', 'ABCDEFGHIJ', 2, 2)) returns 2
INSTRING('DEF', 'ABCDEFGHIJ', 3, 1)) returns 2
```

## LEFT (return left justified string)

LEFT(string [,length])

LEFT left justifies a string.

String A string constant, variable, or expression.

Length A numeric constant, variable, or expression for the length of the return string. If omitted, length defaults to the length of the string.

The LEFT function returns a left justified string. Leading spaces are removed from the string.

Example:

```
LEFT(' 123')           '123'  
LEFT(DAYNAME(TODAY()), 3) First three letters of the day name
```

## LEN (return length of string)

LEN(string)

LEN returns length of a string.

String A string constant, variable, or expression.

The LEN function returns the length of a string. If the string parameter is the label of a STRING variable, the function will return the declared length of the variable. If the string parameter is the label of a CSTRING or PSTRING variable, the function will return the length of the contents of the variable. Numeric variables are automatically converted to STRING intermediate values.

Example:

```
LEN('1234567') 7
```

## LOG10 (return base 10 logarithm)

LOG10(expression)

LOG10 returns base 10 logarithm.

Expression A numeric constant, variable, or expression. If the value of the expression is zero or less, the return value will be zero. The base 10 logarithm is undefined for values less than or equal to zero.

The LOG10 (pronounced "log ten") function returns the base 10 logarithm of a numeric expression. The base 10 logarithm of a value is the power to which 10 must be raised to equal that value.

Example:

```
LOG10(100) 2
```

## LOGE (return natural logarithm)

LOGE(expression)

LOGE returns the natural logarithm.

**Expression** A numeric constant, variable, or expression. If the value of the expression is less than zero, the return value is zero. The natural logarithm is undefined for values less than zero.

The LOGE (pronounced "log-e") function returns the natural logarithm of a numeric expression. The natural logarithm of a value is the power to which e must be raised to equal that value. The value of e used internally for these calculations is

2.71828182846.

Example:

LOGE (100) 4.60517018598809

## LOWER (return lower case)

LOWER(string)

LOWER converts a string to all lower case.

**String** A string constant, variable, or expression for the string to be converted. The LOWER function returns a string with all letters converted to lower case.

Example:

LOWER(DAYNAME(0)) Sunday

## MONTH (return month of date)

MONTH(date)

MONTH Returns month in year.

**Date** A numeric constant, variable, OR expression. The date must be a standard date. A variable declared with a date picture token is automatically converted to a standard date intermediate value.

The MONTH function returns the month of the year (1 to 12) for a given standard date.

Example:

MONTH(1,1,1999) 1 (it's January)

## RIGHT (return right justified string)

RIGHT(string [,length])

RIGHT right justifies a string.

String A string constant, variable, or expression.

Length A numeric constant, variable, or expression for the length of the return string. If omitted, the length is set to the length of the string.

The RIGHT function returns a right justified string. Trailing spaces are removed, then the string is right justified and returned with leading spaces.

Example:

```
RIGHT(DAYNAME(0),3) 'day'
```

## ROUND (return rounded number)

ROUND(expression, order)

ROUND returns rounded value.

Expression A numeric constant, variable, or expression.

Order A numeric expression with a value equal to a power of ten, such as 1, 10,100,0.1, 0.001, etc. If the value is not an even power of ten, the next lowest power is used; 0.55 will use 0.1 and 155 will use 100.

The ROUND function returns the value of an expression rounded to a power of ten. If the order is a LONG or DECIMAL Base Type, then rounding is performed as a Binary coded decimal operation. Note that if you want to round a real number larger than  $1^3$ , you should use ROUND(num,1.0e°), and not ROUND(num,1). The ROUND function is very efficient ("cheap") as a Binary Coded Decimal operation and should be used to compare REALs to DECIMALs at decimal width.

Example:

```
ROUND(5163,100) returns 5200
```

```
ROUND(657.50,1) returns 658
```

```
ROUND(51.63594,.01) returns 51.64
```

## SIN (return sine)

SIN(radians)

SIN returns sine.

**Radians** A numeric constant, variable or expression for the angle expressed in radians.  $\pi$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2\pi$  radians (or 360 degrees) in a circle.

The SIN function returns the trigonometric sine of an angle measured in radians. The sine is the ratio of the length of the angle's opposite side divided by the length of the hypotenuse. The variable Deg2Rad is predefined so you can multiply degrees to return radians.

Example:

$$\text{SIN}(90 * \text{Deg2Rad}) = 1$$

## SQRT (return square root)

SQRT(expression)

SQRT returns square root.

**Expression** A numeric constant, variable, or expression. If the value of the expression is less than zero, the return value is zero.

The SQRT function returns the square root of the expression. If X represents any positive real number, the square root of X is a number that, when multiplied by itself, produces a product equal to X.

Example:

$$\text{SQRT}(9) = 3.0$$

## SUB (return substring of string)

SUB(string, position, length)

SUB returns a portion of a string.

**String** A string constant, variable or expression.

**Position** A integer constant, variable, or expression. If positive, it points to a character position relative to the beginning of the string. If negative, it points to the character position relative to the end of the string (i.e., a position value of -3 points to a position 3 characters from the end of the string).

**Length** A numeric constant, variable, or expression of the number of characters to return.

The SUB function parses out a sub-string from a string by returning length characters from the string, starting at position.

The SUB function is similar to the "string slicing" operation on STRING and expressions. SUB is less flexible and efficient than string slicing, but SUB is "safer" because it ensures that the operation does not overflow the bounds of the string.

"String slicing" is more flexible than SUB because it may be used on both the destination and source sides of an assignment statement, while the SUB function can only be used as the source. It is more efficient because it takes less memory than individual character assignments or the SUB function (however, no bounds checking occurs).

To take a "slice" of a string, the beginning and ending character numbers are separated by a colon (:) and placed in the implicit array dimension position within the square brackets ([]) of the string. The position numbers may be integer constants, variables, or expressions. If variables are used, there must be at least one blank space between the variable name and the colon separating the beginning and ending number (to prevent prefix confusion).

Example:

```
SUB('ABCDEFGHI',1,1) returns 'A'  
SUB('ABCDEFGHI',-1,1) returns 'I'  
SUB('ABCDEFGHI',4,3) returns 'DEF'
```

## TAN (return tangent)

TAN(radians)

TAN returns tangent.

**Radians** A numeric constant, variable or expression for the angle in radians.  $\pi$  is a constant which represents the ratio of the circumference and radius of a circle. There are  $2\pi$  radians (or 360 degrees) in a circle. The variable Deg2Rad is predefined so you can multiply degrees to return radians.

The TAN function returns the trigonometric tangent of an angle measured in radians. The tangent is the ratio of the angle's opposite side divided by its adjacent side.

Example:

```
TAN( 0.000000 * Deg2Rad) = 0
TAN( 45.000000 * Deg2Rad) = 0.999999999996104
TAN( 90.000000 * Deg2Rad) = 256640577671.349
```

Note: computer trigonometric calculations are not perfect, the correct answer for 45 degrees and 90 degrees is 1.0 and infinity, but this should be close enough except for interplanetary distance calculations.

## TODAY (return system date)

TODAY()

The TODAY function returns the operating system date as a standard date. The range of possible dates is from January 1, 1801 (standard date 4) to December 31, 2099 (standard date 109,211).

Example:

```
DAYNAME(TODAY()) The current day of the week
```

## UPPER (return upper case)

UPPER(string)

UPPER returns all upper case string.

**String** A string constant, variable, or expression for the string to be converted. The UPPER function returns a string with all letters converted to upper case.

Example:

```
UPPER(DAYNAME(0)) 'SUNDAY'
```

## YEAR (return year of date)

YEAR(date)

YEAR returns the year.

Date A numeric constant, variable, or expression.

The YEAR function returns a four digit number for the year of a standard date (1801 to 9999).

Example:

YEAR(TODAY()) The 4 digit current year

# Picture Tokens

*Picture tokens are used with the Format function*

Picture tokens provide a masking format for displaying and editing variables. There are even types of picture tokens: numeric and currency, scientific notation, string, date, time, pattern, and key-in template.

## Numeric and Currency Pictures

@N [currency] [sign] [fill]size [grouping] [places] [sign] [currency] [B]

@N	All numeric and currency pictures begin with @N.
Currency	Either a dollar sign (\$) or any string constant enclosed in tildes (~). When it precedes the sign indicator and there is no fill indicator, the currency symbol "floats" to the left of the high order digit. If there is a fill indicator, the currency symbol remains fixed in the left-most position. If the currency indicator follows the size and grouping, it appears at the end of the number displayed.
Sign	Specifies the display format for negative numbers. If a hyphen ( - ) precedes the fill and size indicators, negative numbers will display with a leading minus sign. If a hyphen follows the size, places, and currency indicators, negative numbers will display with a trailing minus sign. If parentheses are placed in both positions, negative numbers will be displayed enclosed in parentheses. To prevent ambiguity, a trailing minus sign should always have grouping specified.
Fill	Specifies leading zeros, spaces, or asterisks (*) in any leading zero positions, and suppresses default grouping. If the fill is omitted, leading zeros are suppressed.  0 (zero) Produces leading zeroes _ (underscore) Produces leading spaces * (asterisk) Produces leading asterisks
Size	The size is required to specify the total number of significant digits to display, including the number of digits in the places indicator and any formatting characters.
Grouping	A grouping symbol, other than a comma (the default), can appear right of the size indicator to specify a three digit group separator. To prevent ambiguity, a hyphen grouping indicator should also specify the sign.  (period) Produces periods (hyphen) Produces hyphens (underscore) Produces spaces
Places	Specifies the decimal separator symbol and the number of decimal digits. The number of decimal digits must be less than the size. The decimal separator may be a period (.), grave accent ( ` ) (produces

## All Aboard – Calculator reference

periods grouping unless overridden), or the letter "v" (used only for STRING field storage declarations--not for display).

- . (period) Produces a period
- ' (grave accent) Produces a comma
- V Produces no decimal separator

B Specifies blank display whenever its value is zero.

The numeric and currency pictures format numeric values for screen display or in reports. If the value is greater than the maximum value the picture can display, a string of pound signs (#) is displayed.

Example:

### Numeric Result Format

@N9	4,550,000	Nine	digits,	group with commas (default)
@N_9B	4550000	Nine	digits,	no grouping, leading blanks if zero
@N09	004550000	Nine	digits,	leading zero
@N*9	***45,000	Nine	digits,	asterisk fill, group with commas
@N9_	4550000	Nine	digits,	group with spaces
@N9.	4.550.000	Nine	digits,	group with periods

### Decimal Result Format

@N9.2	4,550.75	Two	decimal places,	period decimal separator
@N_9'2	4550,75	Two	decimal places,	comma decimal separator
@N9.'2	4.550,75	Comma	decimal separator,	group with periods
@N9_'2	4550,75	Comma	decimal separator,	group with spaces

### Signed Result Format

@N-9.2B	-2,347.25	Leading	minus sign,	blank if zero
@N9.2-	2,347.25-	Trailing	minus sign	
@N(10.2)	(2,347.25)	Enclosed	in parens	if negative

### Currency Result Format

@N\$9.2B	\$2,347.25	Leading	dollar sign,	blank if zero
@N\$10.2-	\$2,347.25-	Leading	dollar sign,	trailing minus if negative
@N\$(11.2)	\$(2,347.25)	Leading	dollar sign,	in parens if negative

### International Currency Result Format

@N12_'2~ F~ 1	5430,50 F	France
@N~L. ~12'	L. 1.430.050	Italy
@N~£~12.2	£1,240.50	United Kingdom
@N~kr~12'2	kr1.430,50	Norway
@N~DM~12'2	DM1.430,50	Germany

## Scientific Notation Pictures

@Emsn[B]

- @E All scientific notation pictures begin with @E.
- M Determines the total number of characters in the format provided by the picture.
- S Specifies the decimal separation character, and the grouping character when the n value is greater than 3.
- . (period) period and comma
  - .. (period period) period and period
  - ' (grave accent) comma and period
  - .\_ (underscore period) period and space
- N Indicates the number of digits that appear to the left of the decimal point.
- B Specifies that the format displays as blank when the value is zero.

The scientific notation picture formats very large or very small numbers. The format is a decimal number raised by a power of ten.

Example:

Picture	Value	Result
@E9.0	1,967,865	.20e+007
@E12.1	1,967,865	1.9679e+006
@E12.1B	0	displayed as blank
@E12.1	-1,967,865	-1.9679e+006
@E12.1	.000000032	3.2000e-008
@E12_.4	1,967,865	1 967.865e+003

## String Pictures

@Slength

- @S All string pictures begin with @S.
- Length Determines the number of characters in the picture format.
- A string picture describes an unformatted string of a specific length.

Example:

@S20 A 20 character string field

## Date Pictures

@Dn [s] [direction [range] ] [B]

- @D All date pictures begin with @D.
- N Determines the date picture format. Date picture formats range from 1 through 18. A leading zero (0) indicates a zero-filled day or month.
- S A separation character between the month, day, and year components. If omitted, the slash ( / ) appears.
- . (period) Produces periods
  - ' (grave accent) Produces commas
  - (hyphen) Produces hyphens
  - \_ (underscore) Produces spaces
- Direction A right or left angle bracket (> or <) that specifies the "Intellidate" direction (> indicates future, < indicates past) for the range parameter. Valid only on
- B Specifies that the format displays as blank when the value is zero.

The century for dates in any picture with a two-digit year is resolved using "Intellidate" logic. Date pictures that do not specify direction and range parameters assume the date falls in the range of the next 19 or previous 80 years. The direction and range parameters allow you to change this default. The direction parameter specifies whether the range specifies the future or past value. The opposite direction then receives the opposite value (100-range) so that any two-digit year results in the correct century.

For example, the picture @D1>60 specifies using the appropriate century for each year60 years

in the future and 39 years in the past. If the current year is 1996, when the user enters "5/01/40," the date is in the year 2040, and when the user enters "5/01/60," the date is in the year 1960.

Example:

Picture	Format	Result
@D1	mm/dd/yy	10/31/59
@D1>40	mm/dd/yy	10/31/59
@D01	mm/dd/yy	01/01/95
@D2	mm/dd/yyyy	10/31/1959

## All Aboard – Calculator reference

Picture	Format	Result
@D3	mmm dd,yyyy	OCT 31,1959
@D4	mmmmmmmm dd, yyyy	October 31, 1959
@D5	dd/mm/yy	31/10/59
@D6	dd/mm/yyyy	31/10/1959
@D7	dd mmm yy	31 OCT 59
@D8	dd mmm yyyy	31 OCT 1959
@D9	yy/mm/dd	59/10/31
@D10	yyyy/mm/dd	1959/10/31
@D11	yymmdd	591031
@D12	yyyymmdd	19591031
@D13	mm/yy	10/59
@D14	mm/yyyy	10/1959
@D15	yy/mm	59/10
@D16	yyyy/mm	1959/10
@D17	Windows Control Panel setting for Short Date	
@D18	Windows Control Panel setting for Long Date	

### Alternate separators

@D1.	mm.dd.yy	Period separator
@D2-	mm-dd-yyyy	Dash separator
@D5_ dd	mm yy	Underscore produces space separator
@D6'	dd,mm,yyyy	Grave accent produces comma separator

## Time Pictures

### @Tn[s][B]

@T	All time pictures begin with @T.
N	Determines the time picture format. Time picture formats range from 1 through 8. A leading zero (0) indicates zero-filled hours.
S	A separation character. By default, colon ( : ) characters appear between the hour, minute, and second components of certain time picture formats. The following s indicators provide an alternate separation character for these formats.  . ' - _
B	Specifies that the format displays as blank when the value is zero.

The value of time is the number of hundredths of a second since midnight. The picture token converts the value to one of the eight time formats.

## All Aboard – Calculator reference

Example:

Picture	Format	Result
@T1	hh:mm	17:30
@T2	hhmm	1730
@T3	hh:mmXM	5:30PM
@T03	hh:mmXM	05:30PM
@T4	hh:mm:ss	17:30:00
@T5	hhmmss	173000
@T6	hh:mm:ssXM	5:30:00PM
@T7	Windows Control Panel setting for Short Time	
@T8	Windows Control Panel setting for Long Time	

### Alternate separators

@T1.	hh.mm	Period separator
@T1-	hh-mm	Dash separator
@T3_	hh mmXM	Underscore produces space separator
@T4'	hh,mm,ss	Grave accent produces comma separator

## Pattern Pictures

@P[<][#][x]P[B]

@P	All pattern pictures begin with the @P delimiter and end with the P delimiter. The case of the delimiters must be the same.
<	Specifies an integer position that is blank for leading zeroes.
#	Specifies an integer position.
X	Represents optional display characters. These characters appear in the final result string.
P	All pattern pictures must end with P. If a lower case @p delimiter is used, the ending P delimiter must also be lower case.
B	Specifies that the format displays as blank when the value is zero.

Pattern pictures contain optional integer positions and optional edit characters. Any character other than < or # is considered an edit character which will appear in the formatted picture string. The @P and P delimiters are case sensitive. Therefore, an upper case "P" can be included as an edit character if the delimiters are both lowercase "p" and vice versa.

Pattern pictures do not recognize decimal points, in order to permit the period to be used as an edit character. Therefore, the value formatted by a pattern picture should be an integer. If a floating point value is formatted by a pattern picture, only the integer portion of the number will appear in the result.

## All Aboard – Calculator reference

Example:

Picture	Value	Result
@P###-##-#### P	215846377	215-84-6377
@P<#/##/##P	103159	10/31/59
@P(###)###-####P	3057854555	(305)785-4555
@P###/###-####P	7854555	000/785-4555
@p<#:##PMp	530	5:30PM
@P<#' <#"P	506	5' 6"
@P<#lb. <#oz.P	902	9lb. 2oz.