AeroCalc Tutorial and Language Reference

Table Of Contents

AeroCalc Tutorial

Language Reference

Appendix - Functions

AeroCalc Tutorial

Overview

The AeroCalc tutorial lessons touch upon the basics of designing and writing scripts. There are four lessons:

1. Script Basics – Learn the basics of how a script works, and to demonstrate a few of the features of the AeroCalc Script Editor.
2. Conditional Statement Execution -- Learn how to write scripts which respond to the test-conditions which you specify. Introduces the IF...ENDIF control structure.
3. Automatic Script Triggering -- Learn about the Preset and Presave scripts and how they are triggered by AeroLog Pro.
4. Communicating with the User -- Learn how to program script which display message boxes and react based on the user's response

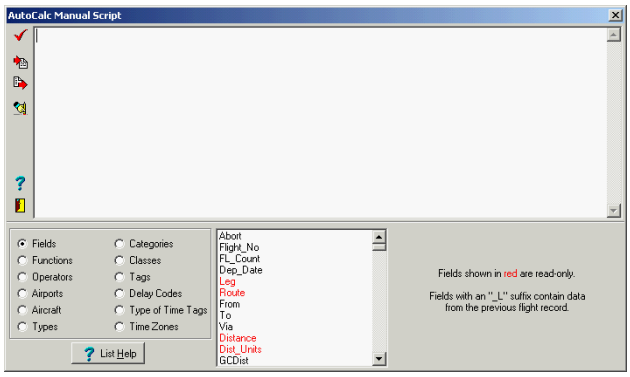
The tutorials are intended to get you started with script writing. However, they only scratch the surface of what you can accomplish using AeroCalc. If you are having difficulty writing a script which does what you need, contact Polaris Microsystems for help. We are happy to offer technical assistance on any aspect of AeroLog Pro, especially the trickier ones like scripts.

Lesson 1 – Script Basics

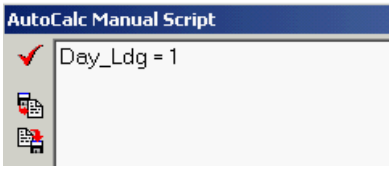
In this lesson you will write and test a simple two-line script. The objective is to learn the basics of how a script works, and to demonstrate a few of the features of the AeroCalc Script Editor.

WARNING! In the following tutorial, you will be running a script which will make changes to records in the Flight Log. To avoid making changes to your actual flight records, you can either open and work with the sample pilot logbook or you can make a backup copy of you records.

- Open the Pilot Logbook Window and select the pilot logbook you want to work with.
- Open the AutoCalc Script Window by selecting [Calc|Edit AutoCalc Scripts...] from the pull-down menus at the top of the Pilot Logbook Window.
- Select the Manual script by clicking the tab at the bottom.
- Double-click anywhere within the script display area to open the AeroCalc Script Editor.



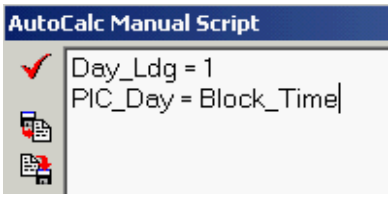
- Click on the Fields radio button. A list of field names will be displayed.
- Find the "Day_Ldg" field in the list and select it by double-clicking on it with the mouse. AeroLog will "type" the field name into the edit area for you.
- Complete the line by typing " = 1 ". The entire line should look like this...



The line you have just entered is a complete AeroCalc statement - in particular it is an "Assignment" statement. When this statement executes, the numeric value "1" will be assigned to the Day_Ldg field in the flight record. This value being assigned (the portion of the statement to the right of the "=") is an expression in its most simple form, a constant. Assignment statements can also assign calculated values which depend on the contents of other fields.

- At this point, the cursor should be to the right of the "1". Press the <Enter> key to move it down to the beginning of the second line.
- Find and select "PIC_Day" from the list below. Type

an "=" sign then select the "Block_Time" field from the list. The complete script should now look like this...



The second assignment statement assigns a value equal to the contents of the Block_Time (Duration) field to the PIC_Day field. In other words, it copies the value of the Block_Time field into the PIC_Day field. The value stored in the Block_Time field is not altered by the process. Assignment statements only modify the field on the left of the "=".

Script statements are executed one at a time beginning from the top. The script above will set the Day_Ldg field to 1 first, and then set the PIC_Day field to the Block_Time value second.

- Click the button to verify the syntax of your script.
- Click the button to close the script editor. Your script will be displayed in the AutoCalc Script Window.
- Check the **Enabled** checkbox at the top to enable the Manual script, then close AutoCalc Script Window and return to the Pilot Logbook Window. Notice the button at the top of the Edit tab. This button "triggers" the execution of the Manual AutoCalc Script (the one we just entered).
- Click New Flight to add a new flight record, then trigger the Manual script by clicking . Notice that a 1 appears in the Day Landings field, as expected, but nothing changes in the Day PIC field. It might appear that the first statement of the script worked but not the second. Actually both worked. At the time the script was triggered, the value of the Duration (Block_Time) field was empty, so zero was assigned to the Day PIC field.
- Press <Tab> until the cursor is in the Duration field. Enter "2:00" then press <Tab> (to record the change). Click again and notice that 2:00 (the value in Duration) is copied to the Day PIC field.
- Click to cancel the new record.

Lesson 2 – Condition Statement Execution

Scripts like the one created in lesson 1 are of limited usefulness because all the statements are executed when the script is triggered. The real power of a script, or any computer program for that matter, is the ability to execute statements only under certain conditions.

Consider the case where you want to set the Cross Country PIC field to the flight Duration. A simple assignment statement will do the trick.

PIC_XC = Block_Time

Adding this statement to a script is easy, and it will work as is, as long as every flight is a cross country flight. On a local flight, the Cross Country PIC field will be incorrectly set. We need a way to test if the flight is cross country and conditionally execute the above statement. The AeroCalc language provides this ability with the IF...ENDIF control structure.

- Open the AutoCalc Script Window, select the Manual script then open the AeroCalc Script Editor.
- Add the last three lines shown below to your script from lesson 1.

The IF...ENDIF structure evaluates the logical (True/False) expression enclosed in the brackets ([ ]), and if the result is *True*, it executes all the statements between the IF and the ENDIF. If the expression is *False*, all statements enclosed in the structure are skipped.

In the script above, the expression "From<>To" is within the brackets. The "<>" (entered by typing a "<" followed immediately by a ">") is the "does not

equal" operator. It compares two values, in this case the From field and the To field, and returns *True* if they are not the same, otherwise it returns *False*. Combined, the IF...ENDIF structure and the "From<>To" expression cause the "PIC_XC = Block_Time" statement to be executed only when the From field is not the same as the To field.

☞ The "{set cross country if From is not the same as To }" is called a comment. Note that the comment text is enclosed in curly braces {}. When a script is executed, anything enclosed in curly braces is ignored. Comments are added to a script to make it more readable to humans. It is a good practice to add comments to "explain" what a statement or group of statements is doing.

- Click the  button to verify the syntax of your script.
- Click the  button to close the script editor, verify that Enabled is still checked, then close AutoCalc Script Window and return to the Pilot Logbook Window.
- Click  to add a new flight record and enter the following field values. You can skip any fields not listed.

  From: KRIC To: KRIC Duration: 1:00

- <Tab> out of the Duration field, then click  to trigger the script. Notice that Duration is not copied to PIC Cross Country. The condition "From<>To" is *False* (From and To are the same), so the IF construct skipped the "PIC_XC = Block_Time" statement.
- Move the cursor back to the To field and change it to "KSBY". <Tab> out of the field then trigger the script again. This time, since the IF condition is satisfied, PIC Cross Country is set to flight duration value.
- Click  to cancel the new record.

Lesson 3 – Automatic Script Triggering

In the previous two lessons, we created scripts which were triggered manually by clicking the  button. These scripts were created in the Manual tab of the AutoCalc Script Window. This lesson will examine the Preset and Presave scripts, and introduce the concept of automatic script triggering.

As we have seen, the Manual script is only executed when triggered by the user during flight entry and editing. The Preset and Presave scripts, on the other hand, are triggered automatically by AeroLog Pro at very specific times:

- The Preset script is triggered immediately after a new flight record created, but before it is displayed in the Pilot Logbook Window. As the name implies, the Preset script is typically used to pre-set values into selected fields of a new flight record.
- The Presave script is triggered immediately before a new or modified flight record is posted to the Flight Log. The timing of the Presave script triggering makes it idea for adding custom error checking to the flight entry process.

To illustrate how the Preset, Manual and Presave scripts can be used, consider an example where a pilot flies privately and for an airline. The private flying is in single engine aircraft, and the airline flying is in multi-engine aircraft. We will also assume for this example that the pilot rarely flies at night.

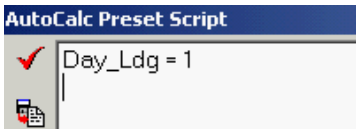
With this in mind, we'll set up AutoCalc to do the following:

- Pre-set 1 into the Day Landings field.
- If the flight is a Private, set From to the local airfield (K17N), else set From to the airline's home base (KPHL).
- Set PIC Day to the flight Duration.
- Set the PIC Cross Country field to the flight Duration for non-local flights.
- Set the Type of Time field to "91 Private" for private flights and "Part 121" for airline flights.

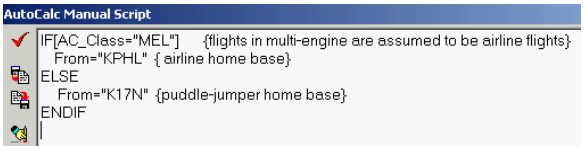
Action # 1 is obviously a pre-set action which can be done before the flight record is displayed. While # 2 could be done automatically, it is better to make it a manually-triggered action since not all flights will be

originating from home base. The remaining 3 actions can be either manual or automatic. For this example, we'll make them automatic pre-save actions.

- Open the AutoCalc Script Window, select the Preset script then open the AeroCalc Script Editor.
- Enter the script as shown below. This one-line script will perform # 1 in the action list above.



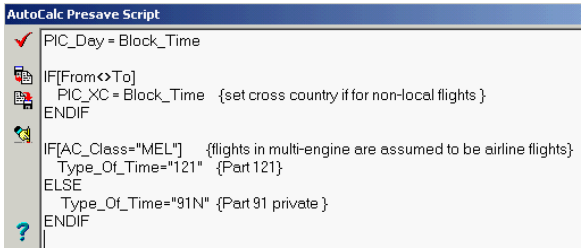
- Click the button to verify the syntax of your script, then click to the script editor.
- Enable the Presave script via the checkbox at the top of the window.
- Select the Manual script then open the script editor.
- Enter the script shown below. (If a script remains from the previous lessons, click first to clear it.) This script will perform # 2 in the action list above.



The IF...ELSE...ENDIF structure used in the above script is similar to the IF...ENDIF structure used in the previous lesson, except that when logical expression enclosed in the brackets ([ ]) is *False*, the statements between ELSE and the ENDIF are executed.

Since the above script checks the aircraft Class (AC_Class) field, it will only function correctly if it is manually triggered after the aircraft used for the flight has been identified. Triggering the script before entering the Aircraft ID will result in From being set to "K17N" (the ELSE portion of the IF...ELSE...ENDIF).

- Click the button to verify the syntax of your script, then click to close the script editor.
- Enable the Manual script via the checkbox at the top of the window.
- Select the Presave script then open the script editor.
- Enter the script shown below. This script will perform #s 3 thru 5 in the action list above.



When entering the above script, instead of typing in "MEL", you can select it from the Class list in the Script Writing Assistant area below. The Assistant will automatically add the quotation marks (") for you. The same applies to the "121" and "91N" which can be selected from the Type of Time list.

- Click the button to verify the syntax of your script, then click to close the script editor.
- Enable the Manual script via the checkbox at the top of the window.

To see the scripts in action...

- Close AutoCalc Script Window and return to the Pilot Logbook Window.
- Click New Flight to add a new flight record. Note that a 1 has been pre-set into the Day Landings field. This is the result of the Presave script which was automatically executed when the record was created.
- Skip to the Aircraft field and enter the identifier for a single-engine (Class=SEL) aircraft (if there are none in your aircraft list, you can create one on the spot by entering a new identifier)).
- Click to trigger the Manual script. Notice how From is set to "K17N".
- Go back to the Aircraft field and enter or select a multi-engine aircraft (Class=MEL). Once again, click and notice that From changes to "KPHL".

- Complete the flight entry by entering the following. You can skip any fields not listed.

  To: KBWI Duration: 0:30

- Click  to post the record. Notice that the Type of Time changes to "Part 121", and the Duration (0:30) has been set into the PIC Day and PIC Cross Country fields. These three changes are the result of Presave script which was triggered by the post.

Try entering other flights, varying the aircraft and the route (some local and some cross country).

You can "turn off" a script without erasing it. Just open the AutoCalc Script Window and un-check the appropriate Enabled checkbox.

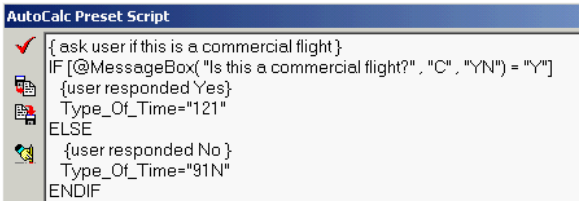
Lesson 4 – Communicating with the User

AeroCalc includes a special function (@MessageBox) which displays a message (dialog box) and waits for the user's response. The response is returned to the script and can optionally be tested and used to control the subsequent actions of the script. In this lesson we'll look at some examples of how to use @MessageBox to enhance your scripts.

Example 1 - A Simple User Query

In this example we'll program a Preset script which "asks" the user if the new record being created is for a private or commercial flight. The response will be tested and used to control the pre-setting of the Type of Time field.

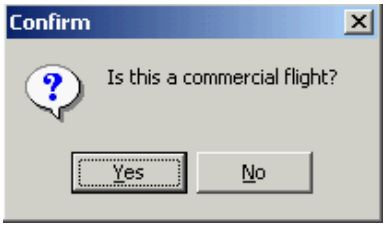
- Open the AutoCalc Script Window, select the Preset script then open the AeroCalc Script Editor. If a script remains from the previous lessons, click  to clear it.
- Enter the script as shown below.



The @MessageBox function takes three parameters. The first is the message to be displayed. The second determines the type of dialog box. In this case the "C" selects an Confirmation type dialog box. The third parameter is a string of letters which determine the buttons to include in the dialog box. In this case, the parameter is "YN" so the box will have **Yes** ("Y") and a **No** ("N") buttons.

@MessageBox() returns a character value which matches the button specifier for the button clicked by the user. In other words, if the user clicks the **Yes** button, a "Y" is returned. If the No button is clicked, a "N" is returned. In the example above, the returned value is tested using an IF...ELSE...ENDIF structure.

- Verify the script, then close the script editor.
- Enable the Preset script.
- In turn, select and disable the Manual and Presave scripts (if enabled).
- Close AutoCalc Script Window and return to the Pilot Logbook Window.
- Add a new flight record. You will immediately see the pop-up message displayed by the @MessageBox call.



- Respond by clicking the **Yes** button. Note that the Type of Time field is pre-set to "Part 121" as per the script.
- Cancel the record and try another. This time select **No** in response to the message. Note the value pre-set into Type of Time.

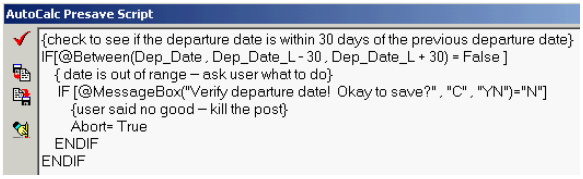
Example 2 - Error Checking and Notification

This example will illustrate using @MessageBox in a Presave script to inform the user of an potential error in the record.

We'll check for a possible mis-entry of the flight departure date by comparing it with the date of the previous flight. If there is more than 30 days between the two dates, a message is displayed asking the user to verify the Departure Date.

If the user rejects the date (as indicated by his or her response to the message dialog), the script sets the special Abort field, to prevent the posting of the suspect flight record.

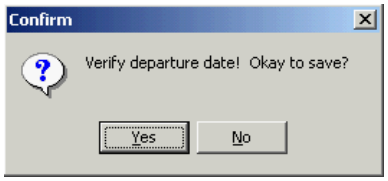
- Select and edit the Presave script. If a script remains from the previous lesson, clear it.
- Enter the script as shown below.



The easiest way to see if a value lies within a certain "range" is to use the @Between function. The function takes three parameters. The first is the value you are testing -- in this case the Dep_Date field. The second and third parameters define the range -- lower end first then upper end. In the example the low-end (second parameter) is the expression (Dep_Date_L - 30). Dep_Date_L holds the departure date from the previous flight. The " - 30" subtracts 30 days from this date. The upper end is also an expression which adds 30 days to Dep_Date_L.

The value of the Abort field is tested by AeroLog prior to its posting a record. If Abort contains a *True* value, the posting is canceled. From the user's point of view, it is as if he or she never tried to post the record. All of the field entries are preserved and the record remains in edit mode.

- Verify the syntax of your script, then close the script editor.
- Enable the Presave script.
- In turn, select and disable the Manual and Preset scripts (if enabled).
- Close AutoCalc Script Window and return to the Pilot Logbook Window.
- Add a new flight record.
- Enter a Departure Date which is more than 30 days from the date of the previous flight.
- Click  to post the record. The Presave script will detect the out-of-range date and display the message shown below.



- Respond by clicking **N**o. Notice that the record is not posted and it remains in edit mode.
- Without changing the date, click  again. This time select **Yes**, and notice how the record is posted.

Language Reference

The AeroCalc language is designed to allow manipulation of flight information within the AeroLog program environment -- specifically, the information contained in the Flight Log portion of the Pilot Logbook. In general, the language is capable of the following...

- reading the contents of a flight record via references to the field names of fields in these records,
- performing calculations (via AeroCalc functions and operators) on the above mentioned field values,
- modifying fields in individual flight records,
- conditionally performing all of the above based on the contents of a field, or on other system conditions such as the current date or time.

These capabilities are available to various extents in the following parts of the AeroLog program...

- **AutoCalc Scripts** - AeroCalc scripts can be written which add custom calculations and data verification to the Pilot Logbook Window.
- **Global Scan Utility Scripts** - This utility makes use of AeroCalc scripts to perform automated batch modification of flight records.

Language Elements Overview

The elements which make up the AeroCalc language are as follows...

- Expressions -- An expression, as defined in the AeroCalc language, is a combination of fields, constants, functions and operators which evaluate to a single value of a specific type.
- Constants -- Fixed literal values.
- Fields -- AeroCalc expressions can incorporate values from Flight Record fields by referencing the field's name.
- Functions -- Functions simplify the process of writing expressions for scripts by encapsulating commonly used operations which would otherwise require several individual statements. In general, a function accepts as input, one or more values (called parameters) and returns a value. Therefore functions can be used in expressions, much like constants and fields are.
- Operators -- Operators are the "verbs" of an expression. They tell AeroCalc what to do with the constants and the values from fields and functions.
- Statements -- A statement is a request for some action. AeroCalc supports two types of statements: Assignment statements (used to modify the value of a field), and IF statements (used to conditionally execute other statements).
- Scripts -- A script is simply a list of one or more statements, which are executed in sequence starting from the top. AeroLog supports two types of scripts: the AutoCalc Script, used during the entry of flight records, and the Global Scan Script, used in conjunction with the Global Scan Utility.

Expressions

An expression is any combination of fields, constants, functions and operators which evaluate to a single value of a specific type. Expressions can be as simple as a lone constant, or as complex as a mathematical formula. All of the following are examples of valid expressions.

- 3.5
- 2 + 5 / 3
- @Today() - 5
- (Block_Time-PIC_Day)*2

The various elements which can be combined in expressions (ie. fields, constants, functions and operators) are discussed below.

Constants

A constant is simply a literal value, as opposed to a value obtained from a field. For example, in the expression Dep_Date+10, the "10" is a constant since its value will always be the numeric value 10. "Dep_Date", by contrast, is a field an can therefore represent different values at different times (i.e. its value is not constant).

Fields

Flight Record Fields

Fields provide access to the individual pieces of information stored in the pilot's flight records. Each Flight Log entry field in the Pilot Logbook Window has a corresponding named AeroCalc field. Some examples are listed below. *For a complete list of the flight record fields available in AeroCalc, see the AeroCalc Language Reference in the Help file.*

Field Name	Type	Remarks
Flight_No	Character	User defined flight number
Dep_Date	Date	Departure (time-out) date -- local to TimeZone
Leg	Numeric	Leg number. Defines sequence for flights on the same date.
From	Character	Origination airport.
Route	Character	Flight route -- From, Via and To fields combined.
To	Character	Termination airport.
Via	Character	Intermediate stops.

This group of fields are associated with the currently selected flight record. For an AutoCalc script, this is the record currently being edited. In the Global Scan Utility, this is the current record in the scan list.

Previous Record Fields

AeroCalc also includes a group of fields for accessing information from the "previous" flight record. For an AutoCalc script, the "previous" record is the record that was selected just prior to the one currently being edited. In the Global Scan Utility, this is the previous record in the scan list order.

Previous Record fields are "read-only". In other words, a script can read the value in the field and use it in an expression, but cannot assign a new value to the field.

For a complete list of the previous record fields available in AeroCalc, see the AeroCalc Language Reference in the Help file.

The Abort Field

The Abort field is a special purpose field which can be used in an AutoCalc script (specifically a Presave script) to prevent the posting of changes to a flight record. When set to *True* in the Autocalc-Presave script, the posting of field changes is blocked and the flight record remains in edit mode. Setting Abort to *True* in the AutoCalc-Preset or AutoCalc-Manual scripts has no effect.

Typically Abort is used in conjunction with a @MessageBox() function call, and is set to either *True* or *False* based on the user's response (the button clicked) to the message dialog box.

Functions

The AeroCalc language provides special-purpose functions which simplify the process of writing expressions for scripts. All of these functions are similar in that they return a value, and therefore can be used in expressions, much like constants and fields are. Most of the functions require one or more values (parameters) as inputs. These input values are used by the function and affect the value that is returned.

Following is a summary of the AeroCalc functions. For a detailed explanation of each function, see the Appendix.

Date Functions

- @AsDate() -- Convert an expression to a date value.
- @BegMonth() -- Return the date of the first day of the specified month.
- @Day() -- Return the day number for the specified date.
- @DMYToDate() -- Convert specified day, month, year values to a date.
- @EndMonth() -- Return the date of the last day of the specified month.
- @Month() -- Return the month number for the specified date.
- @MonthName() – Return the English name of the specified month number.
- @StrToDate() -- Convert a character string to a date value.
- @Today() -- Return today's date from system clock.
- @Year() -- Return the year number for the specified date.

Search Functions

- @BegWith() – Test if character expression begins with the specified character value.
- @Between() -- Test if an expression value lies between two specified values.
- @Contains() -- Test if a character expression contains the specified character value.
- @IIF() -- Return one of two values based on a test condition.
- @Test() -- Return a value if a test condition is true.

Time Functions

- @HMToETime() – Convert specified hours and minutes values to an elapsed time value.
- @HMToTime() – Convert specified hours and minutes values to a clock time value.
- @HrsToETime() -- Convert specified decimal hours value to an elapsed time value.
- @Hours() -- Return the hours portion of a clock time or elapsed time value.
- @Minutes() -- Return the minutes portion of a clock time or elapsed time value.
- @RoundMinutes() -- Round an elapsed time value to the nearest "tenth" boundary.
- @StrToETime() – Convert a character string to an elapsed time value.
- @StrToTime() -- Convert a character string to a clock time value.
- @Tenths() -- Return the tenths portion of an elapsed time value.

Money Functions

- @AsMoney() – Convert an expression to a money value.

Character Functions

- @AsString() – Convert an expression to a character value.
- @BegWith() -- Test if character expression begins with the specified character value.
- @Between() -- Test if an expression value lies between two specified values.
- @Contains() -- Test if a character expression contains the specified character value.
- @Len() -- Return the length of a character value.
- @StrToDate() -- Convert a character string to a date value.
- @StrToETime() -- Convert a character string to an elapsed time value.
- @StrToTime() – Convert a character string to a clock time value.
- @TrimLeft() -- Trim blanks from the left end of a character value
- @TrimRight() – Trim blanks from the right end of a character value

Other Functions

- @Max() -- Return the larger of two values.
- @Min() -- Return the smaller of two values.
- @Not() -- Return the logical inverse (NOT) of a true/false value.
- @Null() – Return an empty or null value. Used to clear logbook fields.
- @MessageBox() – Display various styles of message box and return the user's response.
- @Test() -- Return a value if a test condition is true.

Operators

Following is a summary of the operators use in the AeroCalc language.

Numeric Operators

- Addition/Concatenation (+) – Returns the sum of two numeric expressions, or concatenates two character expressions.
- Subtraction (-) – Returns the numeric difference between two numeric expressions.
- Multiplication (*) – Multiplies two numeric

expressions.

- Division (/) – Divides two numeric expressions.

Comparison Operators

- Greater-Than (>) – Returns *True* if the expression on the left is greater than the one on the right.
- Less-Than (<) – Returns *True* if the expression on the left is less than the one on the right.
- Equal (=) – Returns *True* if two expressions are equal.
- Not Equal (<>) – Returns *True* if two expressions are not equal.
- Less-Than or Equal (<=) – Returns *True* if the expression on the left is less or equal to than the one on the right.
- Greater-Than or Equal (>=) – Returns *True* if the expression on the left is greater than or equal to the one on the right.

Logical Operators

- Logical AND (&) – Returns *True* if both expressions evaluate to a logical *True*.
- Logical OR (|) – Returns *True* if either expression evaluates to a logical *True*.

Operator Precedence

Non-trivial expressions (those with more than one operator) will sometimes yield different results depending on which operator is acted upon first. For example, in the following expression, performing the multiply (*) before the addition (+) yields a result which differs significantly from the result when the addition is done first.

Multiplication First: 3 + 9 * 3 --> 3 + 27 --> 30

Addition First: 3 + 9 * 3 --> 12 * 3 --> 36

The order in which operations are performed is determined by the rules of operator precedence as follows.

- Operations with a higher precedence are performed before those with a lower precedence.
- When two operators have the same precedence, they are acted on in sequence from left to right.

The following table lists the operator precedence order used in AeroCalc language.

Operators	Precedence Level
<, >, =, <>, <=, >=	Lower
+, -, |	...
*, /, &	Higher

Parenthesis can be used to override the above rules. Enclosing a portion of an expression in parenthesis forces that portion to be evaluated before the rest. The following examples illustrate.

Expression	Result	
3+9/3	6	"/" has precedence over "+"
(3+9)/3	4	"(3+9)" is evaluated first
5*3+2*3	21	"*" has precedence over "+"
(5*3+2)* 3	51	"()" forces "* 3" to be done last
(5*(3+2))* 3	75	inner-most "()" is evaluated first

Type Mixing

AeroCalc is designed to manipulate the following types of values: character, elapsed time, count, money, date, clock time and logical (*True/False*). When two different types of values are encountered in a single expression, AeroCalc will attempt to resolve the difference by doing automatic type conversion, where possible. However, an awareness of how AeroCalc deals with type mixing will speed the script writing process and result in more reliable scripts. The primary rule is as follows:

When an expression is evaluated, the value to the left of an operator has "type-dominance" -- that is, AeroCalc will always try to convert the value to the right of an operator to a type compatible with the value to the left of the same operator.

The following examples illustrate...

Expression	Result Type
Dep_Date+10	date -- Dep_Date is dominant.
10+Dep_Date	count -- Dep_Date is converted to #days

Statements

Assignment Statements

Assignment statements alter the contents of a field. Typically they are used in AutoCalc or Global Scan scripts to alter fields in the flight record. The syntax of an assignment statement is as follows:

{field} = {expression}

where {field} is any valid field name, and {expression} is a valid AeroCalc expression. The type of value produced by the {expression} must be compatible with the {field} type in order for the statement to be valid. The following are examples of valid and invalid assignment statements.

Dep_Date = @AsDate("1/13/92") { sets Dep_Date field to 1/13/92 }
Block_Time = @Hrs_Time(1.5) { sets Block_Time field to 1.5 hrs }
Block_Time = "SEL" { invalid - "SEL" cannot be converted to a numeric value }
PIC_Night = Block_Time - PIC_Day {calculate Night as Bock_Time - Day }

Note that the "=" symbol can also represent an Equals comparison operator. AeroCalc uses the context to determine if the "=" is being used as a comparison operator or an assignment operator.

IF Statements

The IF statement allows a block of one or more statements to be conditionally executed, based on the value of an expression. The IF statement can take two possible forms. In the following, {expTF} is an expression which evaluates to either *True* or *False*. {block1} and {block2} represent blocks of one or more assignment statements.

Form 1

```
IF[{expTF}]
   {block1}
ENDIF
```

If {expTF} is *True*, statements in {block1} are executed, otherwise they are skipped.

Form 2

```
IF[{expN}]
   {block1}
ELSE
   {block2}
ENDIF
```

If {expTF} is *True*, statements in {block1} are executed, otherwise statements in {block2} are executed.

Example 1

This example compares the TO and FROM fields, and if they are not the same, the PIC_XC field is set to the total Block_Time.

```
IF[TO <> FROM]
   PIC_XC = Block_Time
ENDIF
```

Example 2

This example is an extension of example 1 which will work with "out and back" flights. In this situation TO and FROM fields are the same, but an intermediate stop is entered into the VIA field.

```
IF[TO <> FROM]
  PIC_XC = Block_Time
ELSE
  IF[@Len(VIA) > 0]
    PIC_XC = Block_Time
  ENDIF
ENDIF
```

Appendix - Functions

@AsDate

Syntax

@AsDate({exp})

Parameters

{exp} is a character or numeric expression.

Result

Attempts to convert {exp} to a date value. If {exp} is numeric, it is interpreted as the number of days since December 30, 1899. A syntax error is generated if the conversion fails.

Examples

@AsDate("11/4/01") {returns 11/4/2001}

@AsDate(100) {returns 4/9/1900}

See Also

- @StrToDate()
- @AsCount()
- @AsString()

@AsCount

Syntax

@AsCount({exp})

Parameters

{exp} is a character, date or numeric expression.

Result

Attempts to convert {exp} to an integer count value. If {exp} is a date, it is converted to the number of days since December 30, 1899. A syntax error is generated if the conversion fails.

Examples

@AsCount(335.5) {returns 335 fractional portion is dropped}

@AsCount("RXX55") {syntax error}

@AsCount(@AsDate("3/15/2002")) {returns 37330 - days since 12/30/1899}

See Also

- @AsString()
- @AsDate()
- @AsETime()
- @AsMoney()

@AsETime

Syntax

@AsETime({exp})

Parameters

{exp} is a character or numeric expression.

Result

Attempts to convert {exp} to an elapsed time value. If {exp} is numeric, it is interpreted as a decimal hours (i.e. hours.tenths) value. If {exp} is character, it must be formatted as "hrs.tenths" or "hrs:minutes". A syntax error is generated if the conversion fails.

Examples

@AsETime(5.5) {returns 5 hrs and 30 min}
@AsETime("4.5") {returns 4 hrs and 30 min}
@AsETime("3:54") {returns 3 hrs and 54 min}
@AsETime("SEL") {syntax error}

See Also

- @HMToETime()
- @HrsToETime()
- @StrToETime()
- @AsCount()
- @AsDate()
- @AsMoney()
- @AsString()

@AsMoney

Syntax

@AsMoney({exp})

Parameters

{exp} is a character or numeric expression.

Result

Attempts to convert {exp} to a currency (money) value. A syntax error is generated if the conversion fails.

Examples

@AsMoney("12.5") {returns 12 dollars and 50 cents}
@AsMonay("RXX55") {syntax error}
@AsMoney(9) {returns 9 dollars and 0 cents }

See Also

- @AsString()
- @AsDate()
- @AsCount()
- @AsETime()

@AsString

Syntax
@AsString({exp})

Parameters
{exp} is a numeric or date expression.

Result
Attempts to convert {exp} to a string (character) value. A syntax error is generated if the conversion fails.

Examples
@AsString(Dep_Date) {converts Dep_Date value to string}
@AsString(PIC_Day) {converts PIC_Day value to string}
@AsString(444) {returns "444"}

See Also
- @AsDate()
- @AsCount()
- @AsMoney()
- @AsETime()

@BegMonth

Syntax
@BegMonth({expD},{expN})

Parameters
- {expD} is a date expression.
- {expN} is a numeric expression.

Result
Returns a date value equal to the first day of the month indicated in {expD} plus or minus {expN} months.

Examples
@BegMonth(DEP_DATE,2) { if DEP_DATE=3/14/92, returns 5/1/92 }

@BegMonth(@TODAY(),0) { returns first of this month }

@BegMonth(@TODAY(),-1) { returns first of last month }

See Also
- @EndMonth()

@BegWith

Syntax
@BegWith({expC1},{expC2})

Parameters
{expC1} and {expC2) are character expressions.

Result
Returns a *True* value if {expC1} begins with {expC2}. Otherwise *False* is returned. The comparison is case insensitive.

Examples
@BegWith("VORTAC","VOR") { returns *True* }
@BegWith("VORTAC","vor"){ returns *True* }
@BegWith("VORTAC","TAC") { returns *False* }
@BegWith(AC_Tags,1) { syntax error -- 1 is not char type }

See Also
- @Contains()

@Between

Syntax
@Between({exp1},{exp2},{exp3})

Parameters
{exp1}, {exp2}, {exp3} are expressions of any one type. All parameters must have the same type.

Result
Returns *True* if {exp1} is greater-than or equal-to {exp2} and {exp1} is less-than or equal-to {exp3}. Otherwise *False* is returned. If the expressions are character, the comparison is case insensitive.

Examples
@Between(3,1,6)
@Between(Dep_Date,@TODAY(),@TODAY()+10)
@Between(AC_Class,"SEL","SES")

@Contains

Syntax

@Contains({expC1},{expC2})

Parameters

{expC1} and {expC2} are character expressions

Result

Returns a *True* if {expC2} is contained in {expC1}. Otherwise *False* is returned. The comparison is case insensitive.

Examples

@Contains("VORTAC","rt") { returns *True* }

@Contains("RT","VORTAC") { returns *False* }

@Contains(AC_TAGS,"RG")

@Contains(AC_TAGS,1) { invalid - 1 is not character }

See Also

- @BegWith()

@Day

Syntax

@Day({expD})

Parameters

{expD} is a date expression

Result

Returns a numeric value equal to the day number indicated in {expD}.

Examples

@Day(Dep_Date) { if Dep_Date=3/14/92, returns 14 }

@Day(@Today()) { returns today's day number }

@Day(@AsDate("4/5/92")) { returns 5 }

See Also

- @Month()
- @Year()
- @MonthName()

@DMYToDate

Syntax

@DMYToDate({expN1},{expN2},{expN3})

Parameters

- {expN1} is a numeric expression representing the day.
- {expN2} is a numeric expression representing the month.
- {expN3} is a numeric expression representing the year.

Result

Returns an equivalent date value.

Examples

@DMYToDate(6,9,1985) { returns 9/6/85 }

@DMYToDate(1,@Month(@Today()),@Year(@Today()) { returns 1st of this month }

See Also

- @Day()
- @Month()
- @Year()

@EndMonth

Syntax

@EndMonth({expD},{expN})

Parameters

- {expD} is a date expression
- {expN} is a numeric expression

Result

Returns a date value equal to the last day of the month indicated in {expD} plus or minus {expN} months.

Examples

@EndMonth(Dep_Date,2}
{ if Dep_Date=3/14/92, returns 5/31/92 }

@EndMonth(@TODAY(),0)
{ returns last of this month }

@EndMonth(@TODAY(),-3)
{ returns last of third previous month }

See Also

- @BegMonth()
- @Today()

@HMToETime

Syntax

@HMToETime({expN1},{expN2})

Parameters

- {expN1} is a numeric expression representing hours.
- {expN2} is a numeric expression representing minutes.

Result

Returns an equivalent elapsed time value.

Examples

@HMToETime(3,55) { returns 3:55 }

See Also

- @AsETime()
- @HrsToETime()
- @Hours()
- @Minutes()

@HMToTime

Syntax

@HMToTime({expN1},{expN2})

Parameters

- {expN1} is a numeric expression ranging in value from 0 to 23 representing hours.
- {expN2} is a numeric expression ranging in value from 0 to 59 representing minutes.

Result

Returns an equivalent clock time value.

Examples

@HMToTime(17,55) { returns 17:55 }

@HMToTime(18,66) { syntax error, minutes value out of range }

See Also

- @Hours()
- @Minutes()

@HrsToETime

Syntax

@HrsToETime({expN})

Parameters

{expN} is a numeric expression representing decimal hours.

Result

Converts {expN} to an equivalent elapsed time value.

Examples

@HrsToETime(1.3) { returns 1:18 }

@HrsToETime(45.5) { returns 45:30 }

See Also

- @AsETime()
- @HMToETime()

@Hours

Syntax
@Hours({exp})

Parameters
{exp} is an elapsed time or clock-time expression.

Result
Returns a numeric value equal to the hours portion of the time indicated in {exp}.

Examples
@Hours(Block_Time) {e.g. if Block_Time= 2:30, returns 2 }
@Hours(@AsETime(3.7)) { returns 3 }
@Hours(Time_Out) {e.g. if Time_Out= 17:45, returns 17}

See Also
- @Minutes()
- @Tenths()

@IIF

Syntax
@IIF({expTF},{exp1},{exp2})

Parameters
- {expTF} is a true/false expression.
- {exp1} and {exp2} are expressions of any type.

Result
If {expTF} is *True*, value of {exp1} is returned, otherwise value of {exp2} is returned.

Examples
@IIF(False,"VORTAC",1.4) { returns 1.4 }
@IIF(True,"VORTAC",1.4) { returns "VORTAC" }
@IIF(AC_Class="SEL",Block_Time,0)
{e.g. if AC_Class is equal to "SEL", returns Block_Time otherwise 0}

@Len

Syntax
@Len({expC})

Parameters
{expC} is a character expression.

Result
Returns a numeric value equal to the length of the character value.

Examples
@Len("Approach") { returns 8 }
@Len("SEL"+"MEL") { returns 6 }

See Also
- @TrimLeft()
- @TrimRight()

@Max

Syntax
@Max({exp1},{exp2})

Parameters
{exp1} and {exp2} are expressions of any one type. Both expressions must have the same type.

Result
Returns a value equal to the larger of the two parameters.

Examples
@Max(6,9) { returns numeric value of 9 }
@Max("SEL","MEL") { returns character value of "SEL" }
@Max(Dep_Date,@TODAY()) { returns later of the dates }

See Also
- @Min()

@Min

Syntax
@Min({exp1},{exp2})

Parameters
{exp1} and {exp2} are expressions of any one type. Both expressions must have the same type.

Result
Returns a value equal to the smaller of the two parameters.

Examples
@Min(6,9) { returns numeric value of 6 }
@Min("SEL","MEL") { returns character value of "MEL" }
@Min(Dep_Date,@TODAY()) { returns earlier of the dates }

See Also
- @Max()

@Minutes

Syntax
@Minutes({exp})

Parameters
{exp} is an elapsed time or clock-time expression.

Result
Returns a numeric value equal to the minutes portion of the time indicated in {exp}.

Examples
@Minutes(Block_Time) {e.g. if Block_Time= 2:30, returns 30 }
@Minutes(@AsETime(3.7)) { returns 42 }
@Minutes(Time_Out) {e.g. if Time_Out= 17:45, returns 45}

See Also
- @Hours()
- @Tenths()
- @HMToETime()

@Month

Syntax
@Month({expD})

Parameters
{expD} is a date expression

Result
Returns a numeric value equal to the month number indicated in {expD}.

Examples
@Day(Dep_Date) { if Dep_Date=3/14/92, returns 3 }
@Day(@Today()) { returns today's month number }
@Day(@AsDate("4/5/92")) { returns 4 }

See Also
- @Day()
- @Year()
- @MonthName()

@MonthName

Syntax
@MonthName({expN})

Parameters
{expN} is a numeric between 1 and 12.

Result
Returns the name of month number {expN} as a character value.

Examples
@MonthName(1) { returns "January" }
@MonthName(@Today()) { returns name of current month }
@MonthName(@Month(Dep_Date))
{ returns Dep_Date month name }

See Also
- @Month()
- @Year()
- @Day()

@MessageBox

Syntax

@MessageBox({msg},{boxtype},{buttonlist})

Parameters

- {msg} is a character expression containing the message text
- {boxtype} is a character expression indicating the type of dialog box to display (see Results below)
- {buttonlist} is a character expression indicating the button(s) to include in the dialog box (see Results below)

Result

@MessageBox displays a message in a pop-up dialog box on the screen, and returns the user's response (the button clicked). The message text is passed in the {msg} parameter. The type of dialog box is specified by the single-character {boxtype} parameter. There are four choices for the type of dialog box to display:

Value of {boxtype}	Type of dialog displayed
"W"	Warning
"E"	Error
"I"	Information
"C"	Confirmation

The buttons displayed in the dialog box are specified using the {buttonlist} parameter. {buttonlist} can contain one or more of the following {buttontype} values

{buttontype}	Button caption
"Y"	Yes
“N”	No
"O"	OK
"C"	Cancel
"A"	Abort
"R"	Retry
“I”	Ignore

To specify more than one button, include each of the desired {buttontype} letters in {buttonlist}. For example, to display both a Yes and a No button, pass a {buttonlist} value of "YN"

@MessageBox returns a character value equal to the {buttontype} of the button the user clicked. For example, if the user clicks the No button in response to the dialog, a "N" is returned.

Examples

```
{display an Error dialog box with Retry and Ignore buttons}
IF["R"=@MessageBox("Block time exceeds limit. Click Retry to correct.","E","RI")]
    {if user clicks Retry, set Abort to True -- cancels record save}
    Abort=True
ENDIF

{ display a confirmation dialog with OK and Cancel buttons.}
IF["C"=@MessageBox("Click Cancel to continue editing.","C","OC")]
    {if user clicks Retry, set Abort to True -- cancels record save}
    Abort=True
ENDIF
```

@Not

Syntax
@Not({exp})

Parameters
{exp} is a numeric or True/False expression

Result
Returns the logical inversion (NOT) of the argument. If {exp} is *True*, *False* is returned. If {exp} is a numeric value, *False* is returned if the {exp} is non-zero, otherwise *True* is returned.

Examples
@Not(True) { returns *False* }
@Not(False) { returns *True* }
@Not(AC_Category="AIR")
{ returns *True* if AC_Category field does not equal "AIR" }
@Not(34) { returns *False* }
@Not(0) { returns *True* }

@Null

Syntax
@Null()

Parameters
None.

Result
Returns a "null" or empty value. When assigned to a logbook field, the value returned by @Null() will clear the field.

Example
PIC_Day=@Null() {clear the PIC_Day field}

@RoundMinutes

Syntax
@RoundMinutes({expN})

Parameters
{expN} is a numeric expression containing an elapsed time value

Result
Rounds the elapsed time value in {expN} to the nearest "tenth" boundary as per the "Minutes to Tenths Rounding Method" currently selected in the Program Options window.

Example
Block_Time = @RoundMinutes(Block_Time)
{ forces Block_Time to the nearest tenth boundary}

@StrToDate

Syntax
@StrToDate({expC})

Parameters
{expC} is a character expression containing a valid date character string

Result
Converts {expC} to an equivalent date value. {expC} must be formatted to match the current date format as set in the Program Options window. A syntax error is generated if the conversion fails.

Examples
@StrToDate("½/92") { returns ½/1992 }
@StrToDate(4/5/92) { syntax error - need quotes (") }

See Also
- @AsDate()
- @DMYToDate()

@StrToETime

Syntax
@StrToETime({expC})

Parameters
{expC} is a character expression containing a valid elapsed time character string.

Result
Attempts to convert {expC} to an elapsed time value. {exp} must be formatted as "hrs.tenths" or "hrs:minutes".
A syntax error is generated if the conversion fails.

Examples
@StrToETime("4.5") {returns 4 hrs and 30 min}
@StrToETime("3:54") {returns 3 hrs and 54 min}
@StrToETime("SEL") {syntax error - not a valid elapse time}

See Also
- @AsETime()
- @HMToETime()
- @HrsToETime()

@StrToTime

Syntax
@StrToTime({expC})

Parameters
{expC} is a character expression containing a valid clock time character string.

Result
Attempts to convert {expC} to a clock time value. {exp} must be formatted as "hrs:minutes". A syntax error is generated if the conversion fails.

Examples
@StrToTime("4.5") {syntax error - invalid format}
@StrToTime("3:54") {returns 03:54}
@StrToTime(3.54) {syntax error - not a character expression}

See Also
- @HMToTime()

@Tenths

Syntax
@Tenths({exp})

Parameters
{exp} is an elapsed time

Result
Returns a numeric value equal to the tenths portion of {exp}.

Examples
@Tenths(Block_Time) {e.g. if Block_Time= 2:30, returns 5 }
@Tenths(@AsETime(3.7)) { returns 7 }

See Also
- @Hours()
- @Minutes()

@Test

Syntax
@Test({expN},{exp1})

Parameters
- {expTF} is a True/False expression.
- {exp1} is an expressions of any type.

Result
If {expTF} is True, returns value of {exp1}, otherwise returns a null value of the same type as {exp1}.

Examples
@Test(AC_Class="SEL",Block_Time)
{e.g. if AC_Class is SEL, returns value of Block_Time, otherwise null }
@Test(AC_Class="SEL",Notes)
{e.g if AC_Class is SEL, returns contents of Notes, otherwise null}

See Also
- @IIF()

@Today

Syntax
@Today()

Parameters
(none)

Result
Returns the current system date.

Examples
@Today()+10 { returns date ten days from today }

@TrimLeft

Syntax
@TrimLeft({expC})

Parameters
{expC} is a character expression.

Result
Returns {expC} with leading blanks removed.

Examples
@TrimLeft(" Approach ") { returns "Approach " }
@TrimLeft(" SEL"+"MEL") { returns "SELMEL" }

See Also
- @TrimRight()
- @Len()

@TrimRight

Syntax
@TrimRight({expC})

Parameters
{expC} is a character expression.

Result
Returns {expC} with leading blanks removed.

Examples
@TrimRight(" Approach ") { returns "Approach" }
@TrimRight(" SEL"+"MEL") { returns " SELMEL" }

See Also
- @TrimLeft()
- @Len()

@Year

Syntax
@Year({expD})

Parameters
{expD} is a date expression

Result
Returns a numeric value equal to the year indicated in {expD}.

Examples
@Day(Dep_Date) { if Dep_Date=3/14/92, returns 1992 }
@Day(@Today()) { returns the current year }
@Day(@AsDate("4/5/92")) { returns 1992 }

See Also
- @Day()
- @Month()
- @MonthName()