Chapter 13
Expressions **(updated 18-feb-2019)**

This chapter describes the expression services in HCA. Expressions are used like in a traditional programming language to change the value of a variable.

This chapter covers these topics:

- Introduction to HCA expressions
- The Visual Programmer Compute and Compute Test elements
- The expression builder
- Managing variables
-
- Important uses of variables besides the Visual Programmer
- Error handling
- Expression syntax and built-in functions

In many cases the simpler variable values – Yes and No – and the three Visual Programmer elements – Make variable yes, Make variable No, and Not variable- are sufficient for applications. The Compute and Compute test elements are used for more sophisticated programming.

Introduction to expressions

As described in the chapter on the Visual Programmer, HCA variables are usually used with simple Yes and No values. But in addition to those you can create and manipulate variables that can store text, numeric, Boolean, or date-time values.

Each variable can contain data of any type. HCA converts the data to the type it needs for the operator being evaluated. For example, consider these expressions:

```
a = 10
b = 20
c = "The result is" + (a + b)
d = #01-01-2001#
e = a - "8"
```

After these expressions are evaluated:

a is a number with value 10

b is a number with value 20

c is a string with value "The result is 30"

d is a date

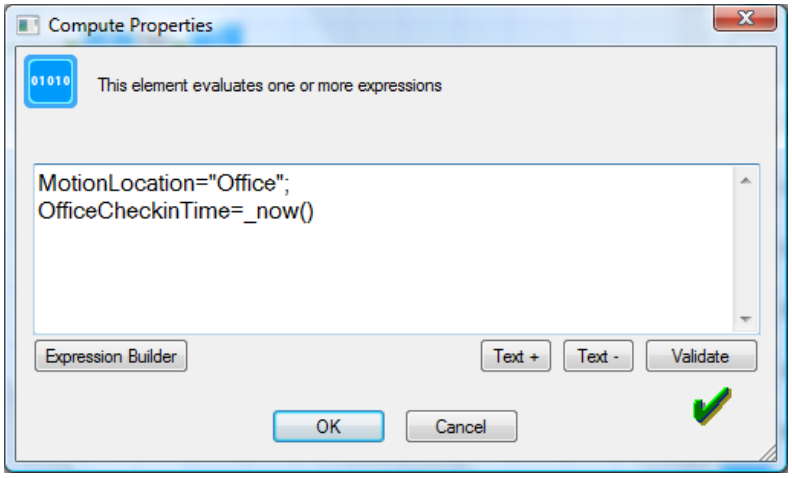
e is a number with value 2

If you understand, or can learn about, how expressions in traditional programming languages like Visual Basic work, you will understand HCA expressions.

Compute and Compute-Test visual programmer elements

To use these expressions two visual programmer elements are available: Compute and Compute Test.

The properties of the Compute element are:



In the Compute element is placed a series of expressions each separated by semicolons.

<variable name> = <expression> ;

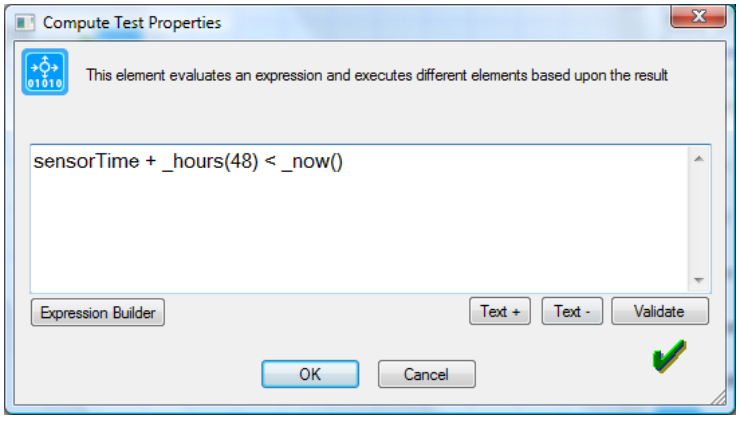
<variable name> = <expression> ;

...

<variable name> = <expression>

When the compute element is executed, the expressions are evaluated and the computed values assigned to the named variables. Expressions are executed in sequential order.

The Compute Test element contains a single expression that is evaluated to determine a yes or no value. If the value is "yes" the path marked "yes" in the program is taken from the Compute Test element, and likewise for "no".



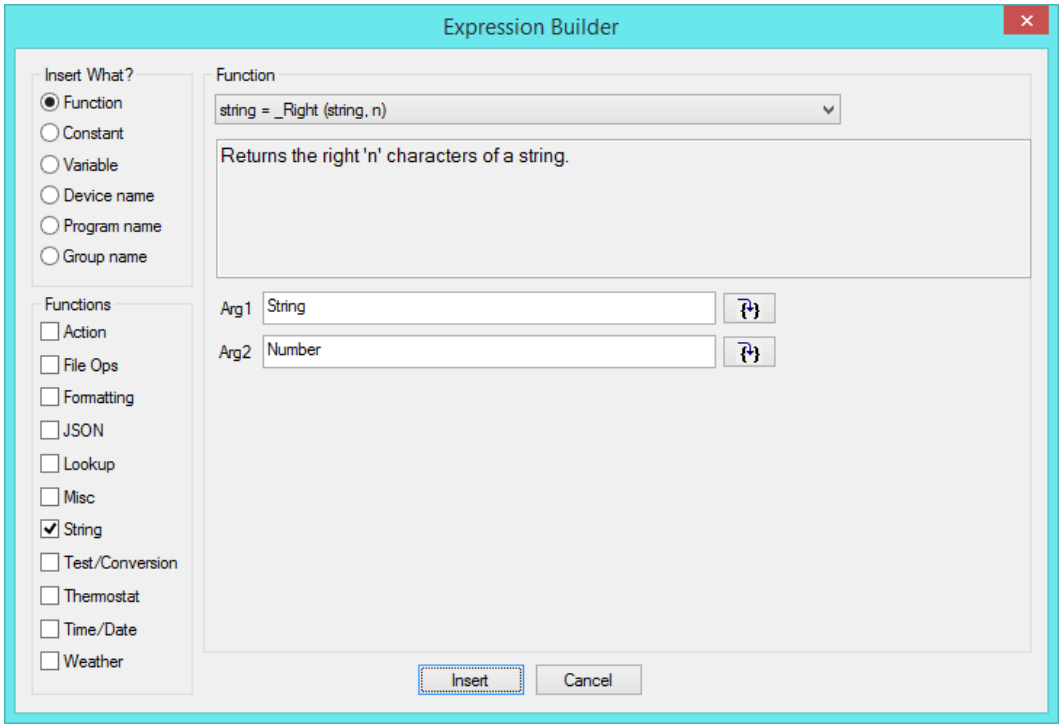
In both these elements the Validate button is used to check that the expression you have entered is correct – it matches the syntax that HCA expects.

A lot of work went into the Visual Programmer to allow HCA users to create programs without all the baggage of existing programming languages – careful syntax, programming terms and concepts. These two elements take a step back from that and leave you in the realm of the programmer. If you have never used, for example, Visual Basic, or all this sounds Greek to you, stick with simple yes and no variables managed with the visual programmer elements for them. You can do many wonderful things with them alone.

Expression builder

To help you create expressions, rather then always having to refer to this documentation, HCA contains a tool called the Expression Builder.

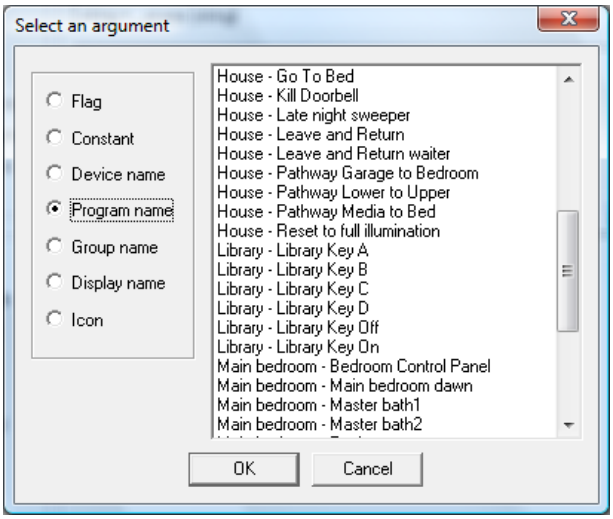
On dialogs where you enter an expression, a button labeled *Expression Builder* helps you compose your expression. Pressing this button opens the builder tool.



The “Insert what?” box specifies what sort of item you are inserting. The most common case is one of the HCA built-in functions.

The “Functions” box lets you limit the number of choices of the possible functions you have to choose from.

The third section of the dialog changes depending upon what you are inserting. In the picture above, a function is being inserted. Choose the name of the function in the dropdown and two things display: a short explanation of what it does, and the parameters to the function. In this example, the _right function takes two arguments. You can simply type in the two arguments, or to get more assistance, press the button next to the argument. This opens this dialog:



This dialog lets you insert common things that you may want to work with. Things like the names of the objects in your design, variables, and constants.

When you close the Expression Builder, the constructed expression is inserted into the text of the element properties at the cursor. Or it replaces the current selection if there is one.

Managing Variables

An important point about variables is that they usually get created when expressions are evaluated. When a program is executed any new variables that are used in its expressions appear in the variables list in the variable inventory dialog.

The variable inventory dialog is described in the chapter on design tools.

Other uses for expressions

In addition to using expressions in the Compute and Compute Test visual programmer elements, you can place expressions in other elements. Just enclose the expression in %'s. When the element is executed, the expression is evaluated and the result in text form replaces the % enclosed section. For example, to show the value of an expression, use this text in the ShowMessage element:

The value of beta is %beta1 + beta2%

If your string wants to display a percent sign, use two in the string:

Inside humidity is %humInside%%%

In these elements, an Embed Expression button appears. This lets you build an expression then encloses it in %'s when it places the expression into the element's properties.

To see in which elements an embedded expression can be used, see technical note titled "Parameter and expression use in program elements".

Error Handling

Because these elements happen at a more complex layer of HCA than most elements, errors can happen that could not be detected in the Visual Programmer. If errors occur while executing these elements, the errors are logged, the Compute or Compute Test element is abandoned, and execution continues with the next element. In the case of the Compute Test element execution follows the Yes path. These errors show up with a red "P" marker and can be filtered as an Error.

Some of the possible errors are:

- Naming a device, thermostat or a magic module as an argument to a function and no device, thermostat, or magic module with that name in your design.
- Divide by zero.
- Using a weather function but no weather provider available.
- Trying to construct a date-time with something out of range. Like a month of 13.

Expression Syntax

HCA expressions are very similar to expressions in any programming language like Visual Basic or VBScript. The usual operators are available:

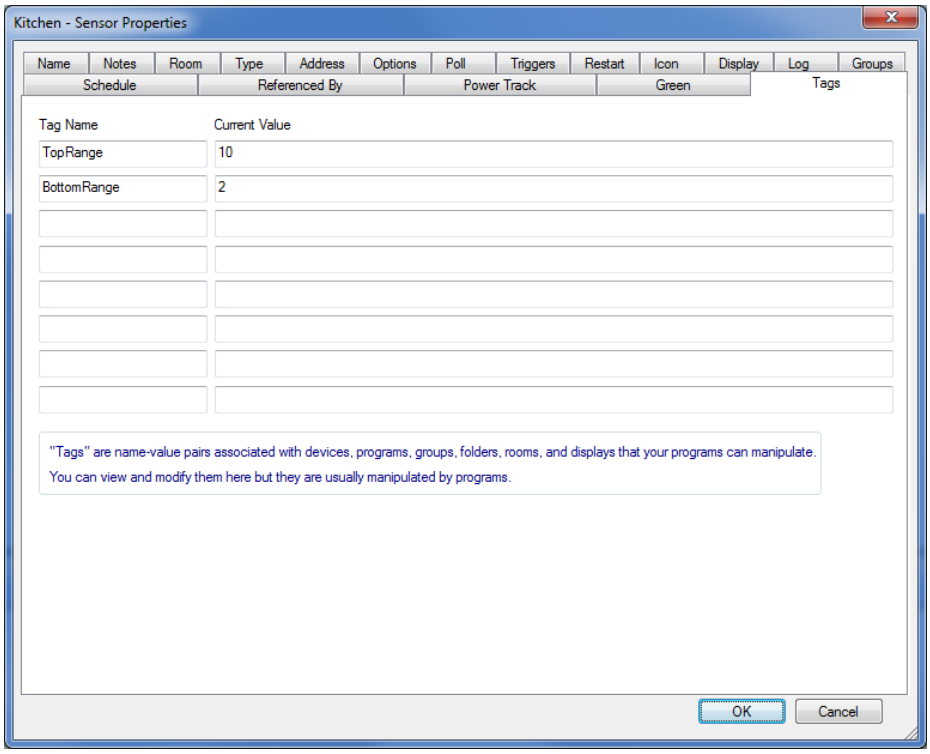
Comparison operators	< > <= >= <> == **Note** that the operator that checks for equality is 2 equal signs not one
Arithmetic operators	+ - * / mod - (unary minus) + (unary plus)
Logical operators	and or not eqv imp xor
Binary operators	Binary or operator is a Vertical bar | Binary and operator is an Ampersand & Binary not operator is a circumflex ^
Date and Time constants	Enclosed in #'s as: #1/15/2001 07:19 AM#
Boolean constants	Yes No True False
String constants	Can be enclosed in single or double quotes
Variables	If the name of that variable has a blank in it, enclose the variable name in square brackets. For example [My Variable]

Tags

A “tag” is simply a name-value pair associated with an object in your design. The name can be any valid HCA name and the value is any text string. You can associate up to 8 “tags” with a room, device, program, group, or keypad.

What are tags good for? That is up to your design. You can use them to associate any piece of data with an object. The key idea behind tags is that programs can manipulate them – adding, deleting, and seeing if a device has a tag or a specific value of a tag. In this way you can have programs save data in a tag of an object and then later another program can read it out. In some ways they are like global variables except that the value is specific to each object.

In the properties dialog there is a “Tags” tab where the current tags assigned to an object are shown with their values. Using this you can view and modify tags and their values from the UI.



In addition to manually viewing and changing, these expression functions are available and described in the Lookup Functions section below:

- ObjectTagGet
- ObjectTagSet
- ObjectTagDelete
- ObjectTagExists

Expression functions

In the expressions used in the Compute and Compute-Test elements, you can use the built-in functions that HCA provides. Some of these are very general and can be found in almost any programming language, and others are specific to HCA.

Some notes on the function-by-function list below:

- In the list of functions below, some have optional parameters. These are designated by showing the parameter in []'s
- All functions return some value and in the Compute element you must assign that value to a variable. In the function list below if the result isn't useful, it is designated as "void".
- In creating expressions there are several functions that HCA provides. Some of these are very general and can be found in almost any programming language, and others are specific to HCA.
- All functions provided by HCA begin with the underscore character. If none of your variable names begin with an underscore, if in subsequent versions of HCA new functions are added, none of your variable names will conflict with any new function names. The case of the name is not important, so the left trim function can be written *Ltrim* or *ltrim*.

String functions

The string functions are identical to the Visual Basic functions of the same name.

Name	Asc
Result type	Number
Parameters	(string)
Action	Returns the ASCII value of the 1st character of the string
Example	_Asc(“A”) → 65

Name	Chr
Result type	String
Parameters	(number)
Action	Returns a one-character string of the ASCII character number
Example	_Chr(65) → “A”

Name	DecToHex
Result type	String
Parameters	(number, # of hex digits)
Action	Converts a number to a string where the value is expressed in hexadecimal
Example	_DecTohex(100,2) → “64”

Name	HexToDec
Result type	Number
Parameters	(string)
Action	Converts a string containing hex characters into a number. Undefined if there are non-hex characters in the string.
Example	_HexToDec(“64”) → 100

Name	InStr
Result type	Number
Parameters	(string, string)
Action	Finds the location of a string within another string. Returns the character position where the 1st character is 1. Returns -1 if not found.
Example	_InStr("webber", “bb”) → 3

Name	LTrim
Result type	String
Parameters	(string)
Action	Trims white-space characters from the left side of a string
Example	_LTrim(" example") → “example”

Name	Lcase
Result type	String
Parameters	(string)
Action	Make a string lowercase
Example	_Lcase("Webber") → “webber”

Name	Left
Result type	String
Parameters	(string, number)
Action	Returns the leftmost ‘n’ characters
Example	_Left("webber", 3) → “web”

Name	Len
Result type	Number
Parameters	(string)
Action	Returns the number of characters in the string
Example	_Len("webber") → 6

Name	Match
Result type	Bool
Parameters	(string, pattern string)
Action	Performs a regular expression match between the supplied string and pattern. If the expression matches the pattern, then the function returns yes.
Example	_Match(“00773”, “00.*”) → yes

Name	Mid
Result type	String
Parameters	(string, start at #, [stop at #])
Action	Returns characters from the start position to the end position. The first character in the string is 1. If the second argument is omitted the remainder of the start starting from the start position is returned.
Example	_Mid("webber", 2, 3) → “ebb”

Name	RTrim
Result type	String
Parameters	(string)
Action	Trims white-space characters from the right end of the string
Example	_RTrim ("webber ") → “webber”

Name	Right
Result type	String
Parameters	(string, # characters)
Action	Returns the rightmost ‘n’ characters from a string
Example	_Right("webber", 2) → “er”

Name	TextPiece
Result type	String / Bool
Parameters	(string, piece #, delimiter string)
Action	Returns the nth piece of the string that contains sections of text between delimiters. If there is no nth piece in the string a bool of No is returned.
Example	_TextPiece("apple,banana,grape", 2, “,”) → “banana”

Name	TextReplace
Result type	String
Parameters	(string, find string, replacement string)
Action	Replaces one string for another within a string.
Example	_TextReplace("speed 10 mph", "mph", “miles per hour”) → “speed 10 miles per hour”

Name	Trim
Result type	String
Parameters	(string)
Action	Trims white-space characters from the left and right ends of a string.
Example	_Trim(" webber ") → “webber”

Name	Ucase
Result type	String
Parameters	(string)
Action	Uppercases a string.
Example	_Ucase("webber") → “WEBBER”

Test / Conversion functions

These functions are generally useful in creating programs, and many are like the corresponding Visual Basic functions.

Name	Abs
Result type	Number
Parameters	(number)
Action	Returns the absolute value of the number
Example	_Abs(-87) → 87

Name	Choose
Result type	Any
Parameters	(number, any, any, …)
Action	Returns as its result the Nth argument. The 1st argument chooses which argument to be returned. The *any* arguments can be of any types and there can be up to 10 of them
Example	_Choose (3, “Jan”, “Feb”, “Mar”, “Apr”, “May”) → “Mar”

Name	IIF
Result type	Any
Parameters	(bool, any1, any2)
Action	Returns any1 if the 1st argument is yes. Otherwise returns any2.
Example	_IIF (yes, “red”, “blue”) → “red”

Name	Int
Result type	Number
Parameters	(any)
Action	Converts the argument to a number if not already a number and returns it discarding any fractional part
Example	_Int(412.87) → 412

Name	IsBool
Result type	Bool
Parameters	(any)
Action	Returns YES if the argument is a bool or an expression that evaluates to a bool.
Example	_IsBool(“Yes”) → Yes

Name	IsDate
Result type	Bool
Parameters	(any)
Action	Returns YES if the argument is a datetime or an expression that evaluates to a dateTime.
Example	_IsDate(“web”) → No

Name	IsEven
Result type	Bool
Parameters	(number)
Action	Returns YES is the argument is an even number
Example	_IsEven(13) → No

Name	IsNumber
Result type	Bool
Parameters	(any)
Action	Returns YES if the argument is a number or an expression that evaluates to a number.
Example	_IsNumber(“412”) → Yes

Name	IsOdd
Result type	Bool
Parameters	(number)
Action	Returns YES is the argument is an odd number
Example	_IsOdd(13) → Yes

Name	IsText
Result type	Bool
Parameters	(any)
Action	Returns YES if the argument is text or an expression that evaluates to text.
Example	_IsText(“hello there”) → Yes

Name	Max
Result type	Number
Parameters	(number, [number], [number], …)
Action	Returns the maximum value of the arguments given. Up to 10 arguments.
Example	_Max(10, 50, 13, 17) → 50

Name	Min
Result type	Number
Parameters	(number, [number], [number], …)
Action	Returns the minimum value of the arguments given. Up to 10 arguments.
Example	_Min(10, 50, 13, 17) → 10

Name	Num
Result type	Number
Parameters	(any)
Action	Converts the argument to a number if not already a number. Generally not needed because HCA converts between strings and numbers as needed.
Example	_Num(“100”) → 100

Name	Round
Result type	Number
Parameters	(number)
Action	Rounds the number to the nearest integer
Example	_Round(402.6) → 403

Time and date functions

For the examples below, assume that the current time is 02:12:45 pm and the current date is Friday 28-September-2018

Name	Date
Result type	DateTime
Parameters	(year #, month #, day#)
Action	Constructs a datetime value from the supplied year, month, and day. Year is the 4-digit year, month is 1-12, day is 1-31
Example	_Date(2018, 9, 28) → 28-Sep-2018

Name	DateTime
Result type	DateTime
Parameters	(year #, month #, day#, hour#, minute#, second#)
Action	Constructs a datetime value from the supplied year, month, day, hour, minute, second. Year is the 4-digit year, month is 1-12, day is 1-31, hour is 0-23, minute is 0-59, and second is 0-59
Example	_DateTime(2018, 9, 28, 14, 16, 30) → 28-Sep-2018 2:16:30pm

Name	Day
Result type	Number
Parameters	(dateTime)
Action	Returns the day from a dateTime
Example	_Day(_Now()) → 28

Name	DayOfWeek
Result type	Number
Parameters	(dateTime)
Action	Returns the weekday from a dateTime as a number from 1 to 7, where 1 is Sunday
Example	_DayOfWeek(_Now()) → 6

Name	DayOfYear
Result type	Number
Parameters	(dateTime)
Action	Returns the ordinal number of the day of the year
Example	_DayOfYear(_now()) → 271

Name	Days
Result type	DateTimeSpan
Parameters	(#days)
Action	Creates a date time span equal to the number of days
Example	_Now() + _Days(1) → Saturday 28-September-2018 02:12:45 PM

Name	Hour
Result type	Number
Parameters	(dateTime)
Action	Returns the hour of a date time
Example	_Hour(_Now()) → 14

Name	Hours
Result type	DateTimeSpan
Parameters	(#hours)
Action	Creates a date time span equal to the number of hours
Example	_Now() + _Hours(2) → Friday 28-September-2018 04:12:45 PM

Name	Minute
Result type	Number
Parameters	(dateTime)
Action	Returns the minute of a date time
Example	_Minute(_Now()) → 12

Name	Minutes
Result type	DateTimeSpan
Parameters	(#minutes)
Action	Creates a dateTimeSpan equal to the number of minutes
Example	_Now() + _Minutes(10) → Friday 28-September-2018 02:22:45 PM

Name	Month
Result type	Number
Parameters	(dateTime)
Action	Returns the month of a date time
Example	_Month(_Now()) → 9

Name	MonthName
Result type	String
Parameters	(month#, use full name?)
Action	Returns the name of the month as a text string. If the 2nd argument is yes, the full name is generated otherwise an abbreviation is used.
Example	_MonthName(_now(), yes) → “September”

Name	Now
Result type	DateTime
Parameters	none
Action	Returns as a datetime the current date and time
Example	_Now() → Friday 28-September-2018 02:12:45 PM

Name	ParseTime
Result type	DateTime / Bool
Parameters	(“datetime text”)
Action	Parses the argument into a dateTime. Returns that datetime if the parse worked. Returns a Bool No if not.
Example	_ParseTime(“10/4/2018 5:34:44 AM”) → #10/4/2018 5:34:44 AM#

Name	Second
Result type	Number
Parameters	(dateTime)
Action	Returns the second part of a date time
Example	_Second(_Now()) → 45

Name	Seconds
Result type	DateTimeSpan
Parameters	(#seconds)
Action	Creates a dateTimeSpan equal to the number of seconds
Example	_Now() + _Seconds(10) → Friday 28-September-2018 02:12:55 PM

Name	Sunrise
Result type	DateTime
Parameters	None
Action	Returns the sunrise time for the current location
Example	_Sunrise() → 9/28/2018 7:03 AM

Name	Sunset
Result type	DateTime
Parameters	None
Action	Returns the sunset time for the current location
Example	_Sunset() → 9/28/2018 6:57 PM

Name	Time
Result type	DateTime
Parameters	(hour, minute, second)
Action	Creates a dateTime with the given hour, minute, and second
Example	_Time(14, 20, 0) → 02:20:00 PM

Name	TimeSpan
Result type	DateTimeSpan
Parameters	(days, hours, minutes, seconds)
Action	Creates a dateTimeSpan with the given values
Example	_Now() + _TimeSpan(0, 1, 20, 0) → Friday 28-September-2018 03:32:45 PM

Name	TotalHours
Result type	Number
Parameters	(dateTimeSpan)
Action	Returns the total number of hours represented by the time span
Example	_TotalHours(_TimeSpan(0, 1, 30, 0)) → 1.5

Name	TotalMinutes
Result type	Number
Parameters	(dateTimeSpan)
Action	Returns the total number of hours represented by the time span
Example	_TotalMinutes(_TimeSpan(0, 1, 30, 0)) → 90

Name	TotalSeconds
Result type	Number
Parameters	(dateTimeSpan)
Action	Returns the total number of seconds represented by the time span
Example	_TotalSeconds(_TimeSpan(0, 1, 30, 0)) → 5400

Name	Weekday
Result type	String
Parameters	(#days ago)
Action	Returns the three-letter abbreviation of the weekday. (0) = today, (1) = yesterday, (2) = 2 days ago, etc.
Example	_Weekday(2) → "Wed"

Name	WeekdayName
Result type	String
Parameters	(number, [full name?])
Action	Returns a string of the weekday name. Sunday is numbered 1, and Saturday is numbered 7. If the second argument is supplied and is YES, the full name is returned, otherwise the 3-letter abbreviation is returned.
Example	_WeekdayName(_DayOfWeek(_Now()), No) → "Fri"

Name	Year
Result type	Number
Parameters	(dateTime)
Action	Returns the year of the dateTime
Example	_Year(_Now()) → 2018

There are four major uses of the time functions in the Compute element. These are:

- Determine how long something took. This is done by:

  t = _Now()

  ... do something...

  timeItTook = _Now() - t

- Add or subtract from the current time to generate a date-time in the past or future:

  TwentyFourHoursAgo = _Now() - _days(1)

  SixAndAHalfHoursAgo = _Now() - _timeSpan(0, 6, 30, 0)

- Compose a date-time from its component parts:

t = _DateTime(2018, 9, 28, 14, 12, 45)

- Format a date-time to a string:

  s = _FormatTime(_Now(), "$d-$b-$y $H:$M")

  This would show as "28-Sep-18 14:12"

Formatting functions

Name	FormatInt
Result type	String
Parameters	(#, #digits, [leading zeros?])
Action	Convert a number to a string with no fractional part. If the 3rd argument is supplied as YES, the string is formatted with leading zeros.
Example	_FormatInt(100.5, 4, 1) → "0100"

Name	FormatNum
Result type	String
Parameters	(#, #decimal places)
Action	Converts the number to a string with the given number of digits after the decimal point
Example	_FormatNum(1.6764, 1) → "1.6"

Name	FormatPatten
Result type	String
Parameters	(#, “pattern”)
Action	Convert a number to a string according to the pattern. The pattern uses the same characters as that used for the C programming language sprintf function. Look online for references to sprintf. The only difference is the $ character is used instead of the % character in the pattern.
Example	_FormatPattern (412.543, “$f.1”) → "412.5"

Name	FormatTime
Result type	String
Parameters	(dateTime)
Action	Returns a string of the date-time formatted according to the pattern. The pattern is a string made up of replacements from the following table.
Example	_FormatTime(_now(), “$d-$b $I:$M $p”) → “28-Sep 2:12 pm”

FormatTime pattern characters:

Pattern marker	Meaning
$a	Abbreviated weekday name
$A	Full weekday name
$b	Abbreviated month name
$B	Full month name
$c	Date and time appropriate for locale
$d	Day of month as number (01-31)
$H	Hour in 24-hour format (00-23)
$I	Hour in 12-hour format (01-12)
$j	Day of year as a number (001-366)
$m	Month as a number (01-12)
$M	Minutes as a number (00-59)
$p	Current locale's AM/PM indicator for 12-hour clock
$S	Second as a number (00-59)
$U	Week of year as a number, with Sunday as the first day of the week (00-51)
$w	Weekday as a number (0-6). Sunday is 0.
$W	Week of year as number with Monday as the first day of the week (00-51)
$x	Date representation appropriate for locale
$X	Time representation appropriate for locale
$y	Year without century as a number (00-99)
$Y	Year with century as number
$z or $Z	Time-zone name or abbreviation. Blank if not known
$$	Dollar sign

Action functions

These functions perform actions on your devices, programs, and groups in your design.

Name	AutoOffTime
Result type	Date-time / Bool
Parameters	(“device name”)
Action	Returns the date-time when the named device/room auto off timer will expire. If there is no auto off timer running for the named device/room, a boolean value of NO is returned. This can be tested for with _IsBool function. Note: This function only reports the auto off time. You can make changes to the auto off timer and settings for a device using the Auto-Off programmer element.
Example	_AutoOffTime(“Kitchen – Lights”) → No

Name	ChangeSchedule
Result type	Bool
Parameters	(“schedule entry name”, code#, time1, [time2])
Action	Modifies the schedule entry with the given name. The codes are: 0 = Change On Time, 1 = Change Off Time, 2 = Change On and Off Time. With code 2 you must supply the 4th argument. Returns YES if the modification is made, NO otherwise.
Example	_ChangeSchedule(“OutsideSet”, 0, _Time(20,0,0)) → Yes This sets the on time of the schedule entry to 8pm

Name	CurrentScene
Result type	String
Parameters	(“device name”)
Action	Returns the current scene, if known, for the device.
Example	_CurrentScene(“kitchen – lights”) → “Nighttime”

Name	CurrentSchedule
Result type	String
Parameters	none
Action	Returns the name of the current schedule.
Example	_CurrentSchedule() → “Away”

Name	DimDownPercent
Result type	Number
Parameters	(“device name”, percent)
Action	Decreases the named object percentage by the supplied amount. Returns the current percentage before the change is made.
Example	_DimDownPercent(“Kitchen – Lights”, 20) → 100

Name	DimPercent
Result type	Number
Parameters	(“device name”, [request status?])
Action	Returns the dim percentage of the named object. If the 2nd parameter is NO or omitted, the evaluation is based upon internal HCA state. If the 2nd parameter is YES and if the device is 2-way, its state is requested to determine the percent returned.
Example	_DimPercent(“Kitchen – Lights”) → 80

Name	DimToPercent
Result type	Number
Parameters	(“device name”, percent)
Action	Controls the named object the supplied percentage. Returns the current percentage before the change is made.
Example	_DimToPercent(“Kitchen – Lights”, 50) → 80

Name	DimUpPercent
Result type	Number
Parameters	(“device name”, percent)
Action	Increases the named object percentage by the supplied amount. Returns the current percentage before the change is made.
Example	_DimUpPercent(“Kitchen – lights”, 10) → 60

Name	IconChange
Result type	Void
Parameters	(“name”, [“icon name”], [“display name”])
Action	Change the displayed icon for a device, program, group, or display. The 2nd parameter is the name of the icon to change to. If omitted, the original icon selected for the object is restored. The 3rd parameter is the name of the effected display. If omitted, then all displays with an icon for this object are changed.
Example	_IconChange(“Kitchen – lights”, “Appliance”)

Name	IconChangeEx
Result type	Void
Parameters	(“name”, code, [“icon name / label text”], [icon representation code])
Action	Change the displayed icon for a device, program, group, or display. The 2nd parameter is a code. code=0 returns the icon to HCA control code=1 changes the icon code=2 changes the label The 3rd parameter is the name of the icon to change to or the icon label text The 4th parameter is the icon representation code (0:On 1:Off 2:Dim)
Example	_IconChangeEx(“Kitchen – Lights”, 2, “Ceiling Lights”)

Name	InsteonBeep
Result type	Void
Parameters	(“device name”, code)
Action	Sends a command to the named Insteon device that should cause it to beep.
Example	_InsteonBeep (“Kitchen – Lights”, 0)

Name	IsCurrentSchedule
Result type	Bool
Parameters	(“schedule name”)
Action	Returns YES if the named schedule is the current schedule.
Example	_IsCurrentSchedule(“Away”) → Yes

Name	IsDim
Result type	Bool / Number
Parameters	(“Device name”, [request status?])
Action	Returns YES if the named object is at 1% to 99% If the 2nd parameter is NO or omitted, the evaluation is based upon internal HCA state. If the 2nd parameter is YES, and the object supports status, its state is requested to determine the return value. If the device doesn’t respond to the status poll, a -1 is returned.
Example	_IsDim(“Kitchen – lights”) → Yes

Name	IsDisabled
Result type	Bool
Parameters	(“name”)
Action	Returns Yes if the object is disabled.
Example	_IsDisabled(“Kitchen – Lights”) → No

Name	IsInErrorState
Result type	Bool
Parameters	(“name”)
Action	Returns yes if the object is in an error state due to a previous communication failure with the device.
Example	_IsInErrorState(“Kitchen – Lights”) → No

Name	IsOff
Result type	Bool / Number
Parameters	(“Device name”, [request status?])
Action	Returns YES if the named object is OFF, NO otherwise. If the 2nd parameter is NO or omitted, the evaluation is based upon internal HCA state. If the 2nd parameter is YES and the object supports status, its state is requested to determine the return value. If the device doesn’t respond to the status poll, a -1 is returned.
Example	_IsOff(“Kitchen Lights”) → No

Name	IsOn
Result type	Bool / Number
Parameters	(“Device name”, [request status?])
Action	Returns YES if the named object is ON, NO otherwise. If the 2nd parameter is NO or omitted, the evaluation is based upon internal HCA state. If the 2nd parameter is YES and the object supports status, its state is requested to determine the return value. If the device doesn’t respond to the status poll, a -1 is returned.
Example	_IsOn(“Kitchen Lights”) → Yes

Name	IsRunning
Result type	Bool
Parameters	(“Program name”)
Action	Returns YES if the program is currently running
Example	_IsRunning(“Home – Driveway Alert”) → Yes

Name	IsSuspended
Result type	Bool
Parameters	(“name”)
Action	Returns YES if the named object is suspended
Example	_IsSuspended(“Kitchen – lights”) → No

Name	Off
Result type	Number
Parameters	(“name”, [button#])
Action	Controls the named object to OFF. Returns the current percentage before the OFF is done. If the optional 2nd argument is supplied, it designates a keypad button indicator.
Example	_Off (“Kitchen – lights”) → 100

Name	On
Result type	Number
Parameters	(“name”, [button#])
Action	Controls the named object to ON. Returns the current percentage before the ON is done. If the optional 2nd argument is supplied, it designates a keypad button indicator.
Example	_On (“Kitchen – lights”) → 0

Name	SetCurrentState
Result type	Void
Parameters	(“name”, percent#)
Action	Change the internal maintained state of the named object to the percent supplied. The device is not actually communicated with. Also updates the icons for the device. 0% = OFF, 100% = ON, 1%-99% = Dim.
Example	_SetCurrentState(“Kitchen – lights”, 80)

Name	SetHomeMode
Result type	Bool
Parameters	(code#)
Action	Set new home mode using code values: 0 = Home & Awake, 1 = Home & Asleep, 2 = Away, 3 = 4th mode Returns YES if successful, NO if not.
Example	_SetHomeMode(1) → Yes

Name	SetRunAgainTime
Result type	Void
Parameters	(date-time, [“program name”])
Action	Requests a program to start at the specified time. Does, in effect, the same action as the Auto-Start configuration on the program's Advanced Options tab but using a computed time rather than a fixed time as specified there. If the time is given as 0, any previous request for start is cancelled. If the 2nd argument is provided, the named program is the target otherwise it applies to the running program.
Example	_SetRunAgainTime(_now() + _hours(1))

Name	SetToScene
Result type	String
Parameters	(“name”, “scene name”)
Action	Controls a device to a named scene. Returns the current scene before the named scene is set.
Example	_SetToScene(“Kitchen – lights”, “nighttime”) → “”

Thermostat functions

Name	GetThermostat
Result type	Bool / Number
Parameters	(“thermostat device name”, code#)
Action	Retrieves the thermostat setting given by the code from the table below. Returns a number if the operation worked and a bool of No if it didn’t. It is up to program that uses this function to request only settings supported by the thermostat and for the setpoints only when in the correct mode. The return value is the setting retrieved or an error. Use the _IsBool on the result to determine if you have received the requested data or an error.
Example	_GetThermostat(“Home – Thermostat”, 1) → 68

Name	SetThermostat
Result type	Bool / Number
Parameters	(“thermostat device name”, code#, value#, [code#], [value#], …)
Action	Changes a thermostat setting given by the code to the value. Codes are given in the table below. Can change from 1 to 5 settings at once with optional code value pairs. Returns a Yes if the operation worked and No if it didn’t. Use the _IsBool on the result to determine if the operation worked or an error. It is up to program that uses this function to change only settings supported by the thermostat and to change the setpoints only when in the correct mode.
Example	_SetThermostat(“Home – Thermostat”, 2, 68) → Yes

Code	**Setting**	**Returned value**
0	Temperature	Integer value
1	Heat Setpoint	Integer value
2	Mode	Off = 0. Heat = 1, Cool = 2, Auto = 3
3	Fan	0 = On, 1 = Off
4	Economy	0 = On, 1 = Off
5	Aux Heat	0 = On, 1 = Off
6	Humidity	Integer value
7	Cool Setpoint	Integer value
8	Has Leaf (NEST only)	0 = On, 1 = Off
13	Nest Mode (NEST only)	0 = Away, 1 = Home When changing the NEST mode, it changes all thermostats in the structure associated with the thermostat being controlled.

Name	TempDecode
Result type	Bool / Number
Parameters	(unitcode#, level#)
Action	Returns the temperature using the RCS X10 decode table for temperature reporting. Useful for legacy thermostats.
Example	

Weather functions

Name	WeatherGet
Result type	Number
Parameters	(“data item name”, [code#], [#hours])
Action	This function retrieves the named weather item in the units the provider is configured for. The item name, provided as a string, is the name of the weather item to retrieve. The item names are the same as used in the weather-test element: “Temperature”, “Apparent Temperature”, “Dew Point”, etc. The code is optional and is specified as: 0=current, 1=max, 2=min, 3=average. The #hours argument is optional and is specified as a positive number for forecast data and as a negative number for historical data.
Example	_WeatherGet(“Wind speed”, 1, -3) → 10.9 Gets the max wind speed in the last 3 hours _WeatherGet(“Wind speed”, 1, +3) → 3.2 Gets the max wind speed expected in the next 3 hours

Name	DarkSky
Result type	String
Parameters	(“path1”, [“path2”], [“path3”], …)
Action	Retrieves data from the dark sky weather provider using the supplied path. There can be from 1 to 10 arguments which are the path through the JSON. The path elements are not case sensitive. See the Dark Sky technical note for more information.
Example	_DarkSky(“currently”, “pressure”) → “1010.34”

Name	BarometerConvert
Result type	Number
Parameters	(number, from units code#, to units code#)
Action	Converts between barometer units. The unit codes are: Inches = 0, Millimeters = 1, Millbars = 2, Hecto Pascals = 3
Example	_BarometerConvert (32, 0, 2) → 1015

Name	BarometerUnits
Result type	String
Parameters	None
Action	Returns a string of the barometer units
Example	_BarometerUnits() → “mb”

Name	HumidityUnits
Result type	String
Parameters	None
Action	Returns a string of the humidity units
Example	_HumidityUnits() → “%”

Name	RainConvert
Result type	Number
Parameters	(Number, from units code#, to units code#)
Action	Converts between rain units. The codes are: Inches = 0, Millimeters = 1
Example	_RainConvert (1, 0, 1) → 25.4

Name	RainUnits
Result type	String
Parameters	None
Action	Returns a string of the rain intensity units
Example	_RainUnits() → “in/hr”

Name	TempConvert
Result type	Number
Parameters	(number, from units code#, to units code#)
Action	Converts between temperature units. The codes are: F = 0, C = 1
Example	_TempConvert(32, 0, 1) → 0

Name	TempUnits
Result type	String
Parameters	None
Action	Returns a string of the temperature units
Example	_TempUnits() → “F”

Name	UVUnits
Result type	String
Parameters	None
Action	Returns a string of the UV units
Example	_UVUnits() → “UV Index”

Name	WindDirUnits
Result type	String
Parameters	None
Action	Returns a string of the wind direction units
Example	_WindDirUnits() → “degrees”

Name	WindDirection
Result type	String
Parameters	(number)
Action	Changes a wind direction in degrees into a string of the form: N, NNE, NE, ENE, etc.
Example	_WindDirection (90) → “E”

Name	WindSpeedConvert
Result type	Number
Parameters	(number, from units code#, to units code#)
Action	Converts between wind speed units. The codes are: Miles per hour = 0, Knots = 1, Kilometers per hour = 2, Meters per second = 3
Example	_WindSpeedConvert (10, 0, 2) → 16.09

Name	WindSpeedUnits
Result type	String
Parameters	None
Action	Returns a string of the wind speed units
Example	_WindSpeedUnits() → “m/s”

File operation functions

This category of functions comprises a set of functions that operate on disk-based files. HCA allows a maximum of 16 files to be open at one time.

Name	FileOpen
Result type	Number
Parameters	(“path”, open option)
Action	Opens a file so that the ReadString and WriteString functions can be used Option 0 = Open for Reading Option 1 = Open for Writing Option 2 = Open for writing and writes append to the end If the file can’t be opened the result is -1
Example	hFile = _FileOpen(“myFile.txt”, 1)

Name	FileClose
Result type	Void
Parameters	(file handle#)
Action	Closes a file previously opened with FileOpen. The argument is the value returned from FileOpen.
Example	Void = _FileClose(hFile)

Name	FileExists
Result type	Bool
Parameters	(“path to the file”)
Action	Determines if a file exists and returns Yes if it does and No if it doesn't.
Example	_FileExists(“myFile.txt) → No

Name	FileLoad
Result type	String / Bool
Parameters	(“path to the file”, code#)
Action	Opens the file at the supplied path, reads the contents and returns the file contents as one string. The options are: 0 = copy CR-LF characters in the file into the result string, 1 = replace CR-LF characters in the file with a single blank in the result string Returns YES if worked, NO otherwise. Test with _IsText or _IsBool to check There is no limit on the size of the file, but a really big file will probably break HCA.
Example	_FileLoad(“my file.txt”, 0) → “The file contents”

Name	FileReadString
Result type	String / Bool
Parameters	(file handle #)
Action	Reads from a file opened by _FileOpen. The handle parameter is the number returned from _FileOpen If there is data remaining in the file, the result is the string read from the file. If there is no data remaining in the file, then the result is a bool value of No. The result can be tested with the IsText or IsBool functions.
Example	_FileReadString (hFile) → "A line from the file"

Name	FileWriteString
Result type	Number
Parameters	(file handle #, "data to write")
Action	Writes to a file opened by _FileOpen. The handle parameter is the number returned from _FileOpen. The number of characters written to the file is returned.
Example	_FileWriteString (hFile, "Hello web") → 9

Name	HCAFolder
Result type	String
Parameters	None
Action	Returns the path to the HCA sub-folder in your documents area.
Example	_HCAFolder() → "c:\users\kimberly\documents\HCA"

JSON functions

JSON is a method of encoding data. HCA has several functions that work with JSON. Refer to the JSON technical note for more information and examples.

Name	Json
Result type	String / Bool
Parameters	(handle, "key1", ["key2"], ["key3"]…)
Action	Retrieves a value from the parsed JSON starting at the current position. There can be from 2 to 10 arguments. arg1 is the handle returned by _JsonOpen arg2 - arg10 are the key names to find at each level. Returns the extracted data if found, NO otherwise. Use _IsBool to check the result.
Example	_Json(handle, "1", "action", "xy", "0") → "0.4573"

Name	JsonOpen
Result type	Number / Bool
Parameters	(“text”)
Action	Parses the JSON text into an internal form and returns a handle to it. If the parse fails, then the result is NO. Use _IsBool to check for failure.
Example	Text = “{ “"Color”" : ”"blue”" , ”"State”" : true}”; hJson = _JsonOpen(text);

Name	JsonClose
Result type	Void
Parameters	(# returned from open)
Action	Releases the parsed form of the JSON. The handle is what was returned by _JsonOpen.
Example	_JsonClose(hJson)

Name	JsonDown
Result type	Bool
Parameters	(# returned from open)
Action	Moves the current position to the first child of the current key in the parsed JSON. The handle is what was returned by _JsonOpen. Returns YES if there if the move was successful, NO otherwise.
Example	_JsonDown(hJson) → Yes

Name	JsonNext
Result type	Bool
Parameters	(# returned from open)
Action	Moves the current position to the next key in the parsed JSON. The handle is what was returned by _JsonOpen. Returns YES if there if the move was successful, NO otherwise.
Example	_JsonNext(hJson) → Yes

Name	JsonPrev
Result type	Bool
Parameters	(# returned from open)
Action	Moves the current position to the previous key in the parsed JSON. The handle is what was returned by _JsonOpen. Returns YES if there if the move was successful, NO otherwise.
Example	_JsonPrev(hJson) → Yes

Name	JsonUp
Result type	Bool
Parameters	(# returned from open)
Action	Moves the current position to the parent of the current key in the parsed JSON. The handle is what was returned by _JsonOpen. Returns YES if there if the move was successful, NO otherwise.
Example	_JsonUp(hJson) → Yes

Lookup functions

This category of functions is useful for working with the elements of your design.

Name	AddressForDevice
Result type	String
Parameters	(“device name”)
Action	Returns the “address” of the device. Formatted as per the protocol of the device.
Example	_AddressForDevice(“Den – Plug”) → “192.168.0.182”

Name	CurrentWattage
Result type	Number
Parameters	([“device or room name”])
Action	Returns the current wattage used by the device or room. With no argument supplied it returns the whole home current wattage.
Example	_CurrentWattage (“Kitchen – Lights”) → 400

Name	DesignOpen
Result type	Number
Parameters	(code#, [“room-folder name”], [“tag name”], [“tag value”)
Action	Works with _DesignName and _DesignClose to allow you to operate on each element in your design. Which elements depends upon the code used. Code=1 devices, code=2 programs, code=3 groups, code=4 rooms & folders, code=5 variables, code=6 only rooms, code=7 only folders, code=8 only displays. If the optional 2nd argument is used for codes 1-3, then it limits what _DesignName returns to the contents of the folder or room given by the 2nd argument. The optional 3rd argument limits what _DesignName returns to those objects that have that tag. If you are using the 3rd or 4th argument and don’t want to limit to a specific room, use “” for the 2nd argument. If the optional 4th argument is used, it is further limited to those objects that have the tag with the given value.

Example	hDesign = _DesignOpen(1, “Kitchen”)

Name	DesignClose
Result type	Void
Parameters	(# returned from DesignOpen)
Action	Close the design opened with _DesignOpen. The one argument must be the value returned from _DesignOpen.
Example	Void =_ DesignClose (hDesign)

Name	DesignName
Result type	String
Parameters	(# returned from DesignOpen)
Action	Returns the name of the current design element and moves to the next element. This allows you to use _DesignName until it returns an empty string. This generates the names of all your design elements based upon the arguments to _DesignOpen.
Example	Name = _DesignName (hDesign)

Name	DesignTitle
Result type	String
Parameters	None
Action	Returns the title set in the Home Properties dialog.
Example	_DesignTitle() → “Kimberly’s Villa”

Name	DeviceForAddress
Result type	String
Parameters	(“Protocol name”, “address”)
Action	Locates a device and returns the name of that device by looking for a device of the supplied protocol with the address. The address is formatted differently for each protocol. The protocols are: “X10”, “Insteon”, “UPB”, “Hue”, “Wireless”, or the name of a user class. Returns the empty string if no such device.
Example	_DeviceForAddress(“Insteon”, “02.62.4b”) → “Den – Light”.

Name	GetDeviceKind
Result type	Number
Parameters	(“device name”)
Action	Returns a value that is the kind of device. possibilities are: 0: Other, 1: Switch, 2: Module, 3: Light, 4: Input, 5: Lock, 6: Camera, 7: Keypad with load, 8: Keypad, 9: IR Output, 10: Fan, 11: Thermostat
Example	_GetDeviceKind (“Kitchen-Lights”) → 1

Name	GetDeviceMake
Result type	String
Parameters	(“device name”)
Action	Returns the name of the device manufacturer if known
Example	_GetDeviceMake (“Kitchen – lights”) → “PulseWorx”

Name	GetDeviceModel
Result type	String
Parameters	(“device name”)
Action	Returns the name of the device model if known
Example	_GetDeviceModel (“Kitchen – lights”) → “WS1D Wall switch”

Name	GetDeviceProtocol
Result type	String
Parameters	(“device name”)
Action	Returns the protocol of the device. The protocols are: "X10", "Insteon", "UPB", "Hue", "Wireless" or the name of a user class.
Example	_GetDeviceProtocol (“Kitchen – lights”) → “UPB”

Name	GetMemberCount
Result type	Number
Parameters	(“group name”)
Action	Returns the number of members in the group.
Example	_GetMemberCount (“Outside – lights”) → 5

Name	GetMemberName
Result type	String
Parameters	(“group name”, member#)
Action	Returns the name of the i'th member of the group.
Example	_GetMemberName (“Outside – lights”, 2) → “Left light”

Name	HomeMode
Result type	Number
Parameters	None
Action	Returns current home mode. 0 = Home & Awake, 1 = Home & Asleep, 2 = Away, 3 = 4th mode.
Example	_HomeMode() → 0

Name	IsValidObject
Result type	Bool
Parameters	(“name”, code#)
Action	Returns YES if the design contains an object with the name of the specified type. Codes are: 0:Device, 1:Program, 2:Group, 3:Room/Folder/Display, 4:Schedule, 5:Global Variable, 6:room, 7:folder, 8:display.
Example	_IsValidObject(“Kitchen – lights”, 0) → yes

Name	LastControlTime
Result type	Date-Time
Parameters	(“device or room name”)
Action	Returns the time of the last control of the device. Will be zero if never controlled. If a room name is supplied, it returns the latest control of any device in the room.
Example	_LastControlTime (“Kitchen – Lights”) → 28-Sep-1018 07:03

Name	LastReceptionTime
Result type	Date-time
Parameters	(“device or room name”)
Action	Returns the time of the last reception from the device. Will be zero if never received from. If a room name is supplied, it returns the latest reception from any device in the room.
Example	_LastReceptionTime (“Kitchen – Lights”) → 28-Sep-1018 07:04

Name	ObjectTagClear
Result type	Bool
Parameters	(“object name”)
Action	Removes all tags from a device, program, group, room, folder, or display.
Example	_ObjectTagClear (“Kitchen – Lights”)

Name	ObjectTagDelete
Result type	Bool
Parameters	(“object name”, “tag name”)
Action	Removes from a device, program, group, room, folder, or display the supplied tag. If the tag doesn't exist for that object, NO is returned, YES otherwise.
Example	_ObjectTagDelete (“Kitchen – Lights”, “color”) → yes

Name	ObjectTagExists
Result type	Bool
Parameters	(“object name”, “tag name”, [“tag value”])
Action	Checks if a device, program, group, room, folder, or display has the supplied tag and optionally checks that the tag value matches the supplied value. If the tag isn't assigned to the object, NO is returned. If the tag is assigned to the object and the value argument is omitted, YES is returned. If the value argument is supplied, YES if returned if the tag value matches the supplied value.
Example	_ObjectTagExists (“Kitchen – Lights”, “color”, “blue”) → yes

Name	ObjectTagGet
Result type	String / Bool
Parameters	(“object name”, “tag name”)
Action	Returns the value of the tag assigned to the device, program, group, room, folder, or display with the supplied name. If the object doesn't have that tag returns NO. Use _IsBool to check the result.
Example	_ObjectTagGet(“Kitchen – lights”, “color”) → “blue”

Name	ObjectTagSet
Result type	Bool
Parameters	(“object name”, “tag name”, “tag value”)
Action	Assigns to the supplied device, program, group, room, folder, or display a tag with the supplied value. If the tag doesn't exist for that object one is added to it. If there is no room for a new tag for that object - it already has the maximum number - a bool NO is returned.
Example	_ObjectTagSet(“Kitchen – lights”, “color”, “blue”) → yes

Name	SetCurrentWattage
Result type	Void
Parameters	(“device name”, wattage#)
Action	Change the current wattage used by the device when at 100% to the value supplied.
Example	_SetCurrentWattage(“Kitchen – lights”, 400)

Name	Statistics
Result type	Number
Parameters	(code#, [“name”)
Action	Returns statistics since HCA was started based upon the supplied code. [Code = 1] Total number of programs executed, unless a program name is supplied and then the count is for only that program [Code = 2] Total number devices controlled, unless a device name is supplied and then the count is for only that device [Code = 3] Total number of client connections [Code = 4] Total number of messages from all interfaces unless interface number provided then the count for only that interface If the code has a negative value it clears the statistics, either the total or for a specific object. Examples: _Statistics (2) → the total count for all devices _Statistics (2, "Kitchen") → the total for all devices in the kitchen _Statistics (2, "Kitchen - Lights") → the count for only that single device _Statistics (-2) → clear counter for every device _Statistics (-2, "Kitchen") → clear counter for every device in the kitchen _Statistics (-2, "Kitchen - Lights") → clear counter for kitchen-Lights
Example	_Statistics(2) → 26

Name	Status
Result type	String
Parameters	(“object name”)
Action	Returns a readable string which shows the status of the device, program, or group. What is returned depends upon the type of the object, and if a device, the type of the device.
Example	_Status(“Kitchen – lights”) → “75%”

Miscellaneous functions

This category of functions comprises a set of generally useful things that don't fit into any other category.

Name	AlertAdd
Result type	Void
Parameters	(userAlert#, "text to put in the alert log")
Action	Raises a user alert. How alerts are configured determines the effect of this. User alert numbers are 1, 2, 3, or 4. The Alert Manager lets you configure "user alerts" for just this purpose.
Example	_AlertAdd(1, "Check pump")

Name	AlertCount
Result type	Number
Parameters	(alert#)
Action	Returns the # of alerts for the alert with that code. Codes are: 1: Unknown reception 5: Missing UPB sequence 7: Status poll fails 8: Confirm receipt of command failed 9: All attempts at confirm receipt of command failed 10: Interface error 11: Weather observation failed 13: Power out (interface disconnected) 14: Power restored (interface reconnected) 15: Program error 16: No ACK from device (Confirm receipt failed) 17: Client disconnected abnormally 18: No reception group 1 19: No reception group 2 20: No reception group 3 21: No reception group 4 22: User alert type 1 23: User alert type 2 24: User alert type 3 25 User alert type 4 26: Interface disconnect 27: Cloud update failed
Example	_AlertCount(7) → 1

Name	Assign
Result type	Void
Parameters	("variable name", any)
Action	Assigns to the named variable – given by a string - the value given by the second argument. If the variable doesn't exist a global variable is created.
Example	_Assign("My variable", 17)

Name	Delay
Result type	Number
Parameters	(time-span1, [time-span2])
Action	If one argument is supplied, then delays for that amount of time. If two arguments are supplied, then delays for a time somewhere between the two time spans. Returns the number of seconds delayed.
Example	_Delay(_Minutes(1), _Minutes(10)) → 117

Name	DelayShort
Result type	Void
Parameters	(milleconds#)
Action	Delays for at least the specified number of milliseconds.
Example	_DelayShort(1500)

Name	DesignSave
Result type	Void
Parameters	([code#])
Action	Saves the current design file. The codes are: 0 or not supplied = Always save. 1 = Only save if modified.
Example	_DesignSave ()

Name	InterfaceName
Result type	String
Parameters	(interface-number)
Action	Returns the name of the interface with the supplied number (1-8). The numbers are in the order that the interfaces appear on the HCA Options hardware tab.
Example	_InterfaceNamel() → “Insteon”

Name	InterfaceStatus
Result type	Bool
Parameters	(interface-number)
Action	Returns the status of the interface with the suppled number (1-8). The numbers are in the order that the interfaces appear on the HCA Options hardware tab. Returns TRUE if working and FALSE if not
Example	_InterfaceStatus(1) → TRUE

Name	PlaySound
Result type	Bool
Parameters	(“path to sound file”, code#)
Action	Plays a Sound file using the computers sound system. The 1st argument is a path to the sound file. The code# argument is as follows: 1: The sound file starts playing and HCA moves to the next element 2: The sound file starts playing and HCA moves to the next element. When the sound file finishes, it starts playing again. 3: The sound file starts playing and HCA waits until it is complete before moving to the next element. If you used option #2, later you can stop the sound file playing by using the PlaySound function again with "" for the path.
Example	_PlaySound (“c:\files\beep.wav”, 1) → yes

Name	ProblemLevel
Result type	Number
Parameters	None
Action	Returns current problem level as shown by the HCA status bar lights. 0 = Green, 1 = Yellow, 2 = Red.
Example	_ProblemLevel() → 0

Name	RGB
Result type	Number
Parameters	(red#, blue#, green#)
Action	Returns the encoded color for the red, green, blue values specified.
Example	_RGB(51, 51,255) → hex value 3333FF (a nice blue)

Name	Rand
Result type	Number
Parameters	(number1, number2)
Action	Returns a random number chosen between the two numbers supplied.
Example	_Rand (100, 200) → 119

Name	ReportAdd
Result type	Void
Parameters	(“text”)
Action	Adds a message to be sent in the next Daily Report.
Example	_ReportAdd (“Check battery levels this week”)

Name	SetSunriseDelta
Result type	Number
Parameters	(sunrise delta #)
Action	Changes the sunrise delta value that is set in the home properties on the Location tab. This is the number of minutes to add or subtract from the computed sunrise time to be more accurate for your location. Returns the value it was set to before the change was made.
Example	_SetSunriseDelta (20) → 8

Name	SetSunsetDelta
Result type	Number
Parameters	(sunset delta #)
Action	Changes the sunset delta value that is set in the home properties on the Location tab. This is the number of minutes to add or subtract from the computed sunset time to be more accurate for your location. Returns the value it was set to before the change was made.
Example	_SetSunsetDelta (20) → 6

Name	ThisProgram
Result type	String
Parameters	None
Action	Returns the name of the running program.
Example	_ThisProgram() → “Garden - Watering”

<table>
<tr><td>Name</td><td>TileUpdate</td></tr>
<tr><td>Result type</td><td>Bool</td></tr>
<tr><td>Parameters</td><td>(“tile name”, code#, [x], [y])</td></tr>
<tr><td>Action</td><td>Updates the names tile based upon the parameters
<table>
<tr><td>Code</td><td>Use</td><td>Arg3</td><td>Arg4</td></tr>
<tr><td>0</td><td>Change label</td><td>Label text</td><td>Not used</td></tr>
<tr><td>1</td><td>Change colors</td><td>Background color</td><td>Text color</td></tr>
<tr><td>2</td><td>Change image path</td><td>Image path</td><td>Not used</td></tr>
<tr><td>3</td><td>Change text</td><td>Text</td><td>Not used</td></tr>
<tr><td>4</td><td>Refresh</td><td>Not used</td><td>Not used</td></tr>
</table></td></tr>
<tr><td>Example</td><td>_TileUpdate (“StatusTile”, 0, “Good”) → yes</td></tr>
</table>

Name	VarValue
Result type	Any
Parameters	(“variable name”)
Action	Returns the value of the named variable given as a text string. If the variable doesn't exist a global variable is created with default value of NO.
Example	_VarValue(“Counter”) → 10