Chapter 13
Expressions

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

This chapter covers these topics:

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

In most cases the simpler flag values – Yes and No – and the three Visual Programmer elements – Make flag yes, Make flag No, and Not flag 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 flags 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.

To maintain compatibility with previous HCA versions, these variables are still called "flags".

Each flag 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

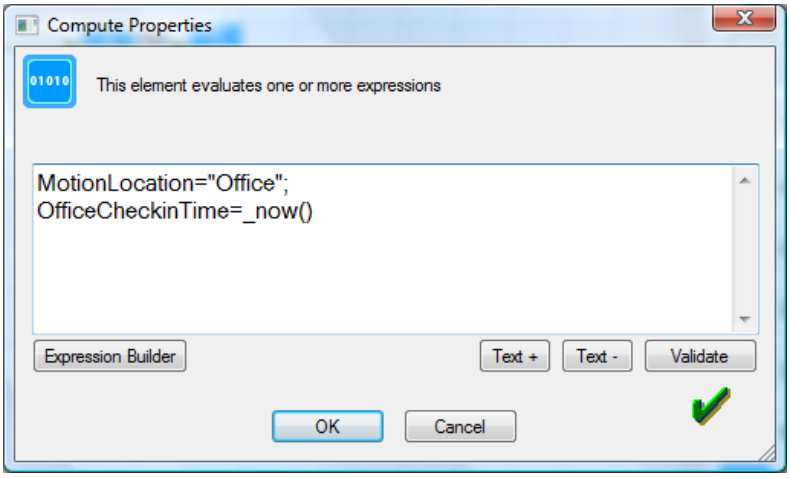
e is a number with value 2

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 in the form:

<flag name> = <expression> ;

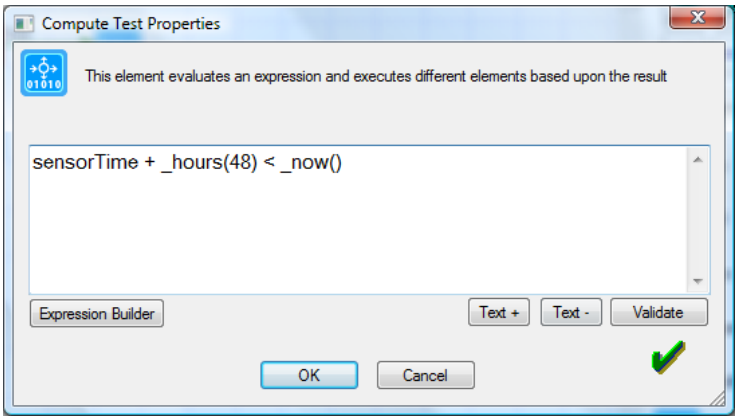
<flag name> = <expression> ;

...

<flag name> = <expression>

When the compute element is executed, the expressions are evaluated and the computed values are assigned to the named flags.

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".



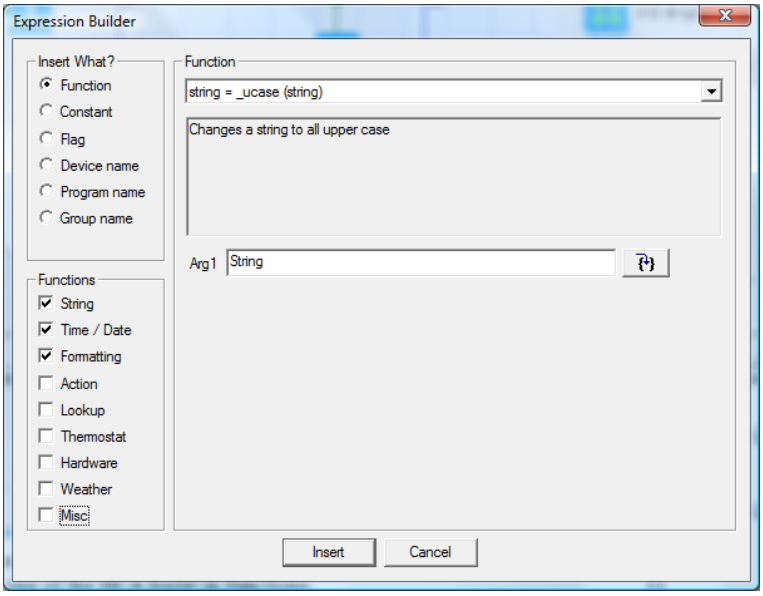
In both these elements the Validate button is used to check that the expression you have entered is correct – it matches the syntax that HCA expects.

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 flags 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.

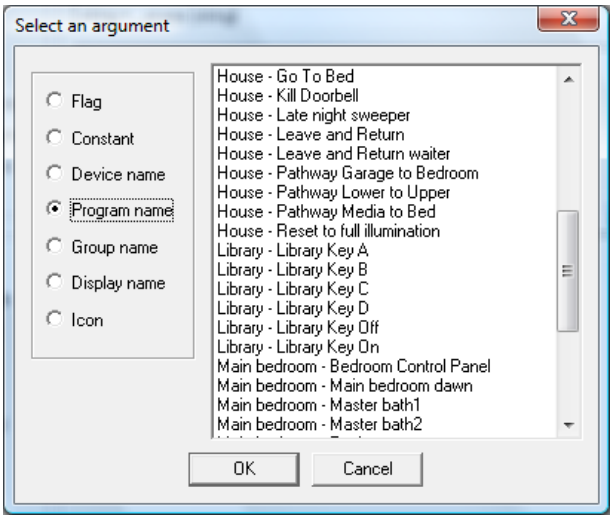
On dialogs where you enter an expression, a button labeled *Expression Builder* helps you compose your expression. Pressing this button opens this dialog:



The upper left box specifies what sort of item you are inserting. The most common case is one of the HCA build-in functions.

The box below that 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, flags, 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 Flags

An important point about flags is that they get created when expressions are evaluated. There is no other way than that to create a flag. That is, there is no "new flag" wizard. When a program is executed any new flags that are used in its expressions appear in the flags list in the flags inventory dialog.

The flags 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%%

These placeholders - % enclosed sections - can be used in these elements:

- Show Message in the text
- Run Program element in the command line
- Speak
- Change Icon
- Add to Log

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.

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 is a Vertical bar |

Binary and is an Ampersand &

Binary not is a circumflex ^

Date and Time constants enclosed in #'s as: #1/15/2001 07:19 AM#

Boolean constants: Yes No

String constants can be enclosed in single or double quotes

If you are using a flag in an expression and the name of that flag has a blank in it, enclose the flag name in square brackets. For example [My Flag]

In creating expressions there are a number of 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. As long as none of your flag names begin with an underscore, if in subsequent versions of HCA new functions are added, none of your flag names will conflict with any new function names.

Hint: Don't start any of your flag names with the underscore character.

String functions

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

string = _ucase (string)
example: _ucase("web")
result: "WEB"

string = _lcase (string)
example: _lcase("WEB")
result: "web"

string = _left (string, n)
example: _left("web", 2)
result: "we"

string = _right (string, n)
example: _right("web, 2")
result: "eb"

string = _mid (string, n, m)
example: _mid("webber", 2, 3)
result: "ebb"

number = _len (string)
example: _len("web")
result: 3

string = _LTrim (string)
example: _LTrim(" example")
result: "example"

string = _RTrim (string)
example: _RTrim("example ")
result: "example"

string = _Trim (string)
example: _Trim(" example ")
result: "example"

string = _chr (number)
example: _chr(65)
result: "A"

number = _asc (string)
example: _asc("A")
result: 65

number = _InStr (string, string)
example: _InStr("webber", "bb")
result: 3

string = _TextPiece (string, number, string)

example: _TextPiece("apple, banana, grape", 2, “,”)
result: “banana”

string = _TextReplace (string, number, string)
example: _TextPiece("Wind speed 10 mph", "mph", “miles per hour”)
result: “Wind speed 10 miles per hour”

number = _HexToDec (string)
example: __HexToDec("1A7”)
result: 423

number = _DecToHex (number, number of digits)
example: __DecToHex(423, 4)
result: 01A7

Bool = _Match (string, pattern string)

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.*”)
result: Yes

General use functions

These functions are generally useful and many are similar to Visual Basic functions.

Bool = _IsDate (any)
Returns YES if the argument is a datetime or an expression that evaluates to a dateTime.

Bool = _IsText (any)

Returns YES if the argument is a string or an expression that evaluates to a string.

Bool = _IsBool (any)
Returns YES if the argument is a yes/no value or an expression that evaluates to a Yes/No value

Bool = _IsNumber (any)
Returns YES if the argument is a number or an expression that evaluates to a number

number = _Max (number, number, …)
Returns the largest number from any of the arguments provided. You can have up to 10 arguments.

number = _Min (number, number, …)
Returns the smallest number from any of the arguments provided. You can have up to 10 arguments.

number = _Abs (number)
Returns the absolute value of the number. That is, always a positive number.

number = _int (number)
Returns the number as an integer.

bool = _IsOdd (number)
Returns yes if the number is odd

bool = _IsEven (number)
Returns yes if the number is even

number = _Round (number)
Rounds thee number to the nearest integer

any = _Choose (number, any, any, …)
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. For example: _Choose (3, “Jan”, “Feb”, “Mar”, “Apr”, “May”)
Result: “Mar”

any = _iif (bool, any1, any2)
Returns any2 if the 1st argument is NO. Otherwise returns any1.

number = _rand (number, number)
Returns a random number chosen between the two numbers supplied.

bool = _PlaySound (string, number)
Plays a Sound file using the computers sound system. The 1st argument is a path to the sound file. The second 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, at a later time you can stop the sound file playing by using the PlaySound function again with “” for the path.

Time and date functions

For these examples, assume that the current time is 02:12:45 pm and the current date is Friday 14-September-2001

number = _hour (dateTime)
example: _hour(_now())
result: 14

number = _minute (dateTime)
example: _minute(_now())
result: 12

number = _second (dateTime)
example: _second(_now())
result: 45

dateTime = _time (hour, minute, second)
example: _time(14, 12, 45)
result: A time of 02:12:45 pm

number = _day (dateTime)
example: _day(_now())
result: 14

number = _month (dateTime)
example: _month(_now())

result: 9

number = _year (dateTime)
example: _year(_now())
result: 2001

dateTime = _date (year, month, day)
example: _date(2001, 9, 14)
result: A date of 14-September-2001

dateTime = _datetime (year, month, day, hour, minute, second)
example: _datetime(2001, 9, 14, 14, 12, 45)
result: 14-September-2001 02:12:45 pm

number = _totalHours (dateTime)
Returns the number of hours represented by the date-time.
example: _totalHours(_time(7,30,50))
result: 7

number = _totalMinutes (dateTime)
Returns the number of minutes represented by the date-time.
example: _totalMinutes(_time(7,30,50))
result: 450

number = _totalSeconds (dateTime)
Returns the number of hours represented by the date-time.
example: _totalSeconds(_time(7,30,50))
result: 27050

number = _dayOfWeek (dateTime)
Returns the day of the week as a number from 1 to 7, where 1 is Sunday
example: _dayOfWeek(_now())
result: 6

number = _dayOfYear (dateTime)
Returns the day of the year as a number from 1 to 366, where 1 is January 1st
example: _dayOfYear(_now())
result: 257

dateTime = _now ()
Returns the current date-time

dateTime = _sunrise ()
Returns the time of sunrise for today

dateTime = _sunset ()
Returns the time of sunset for today

dateTimeSpan = _Days (number)
Returns a time span of the given number of days. See below for some date time span examples.

dateTimeSpan = _Hours (number)
Returns a time span of the given number of hours

dateTimeSpan = _Minutes ()
Returns a time span of the given number of minutes

dateTimeSpan = _Seconds()
Returns a time span of the given number of seconds

dateTimeSpan = _TimeSpan (days, hours, minutes, seconds)
Returns a time span of days, hours, minutes, seconds

string = _weekday (number)

Returns a string of the three letter abbreviation of the weekday.
The argument is the number of days to go back. So if today is Monday then

- _weekday(0) is "Mon"
- _weekday(1) is "Sun"
- _weekday(2) is "Sat"

string = _weekdayName (number, bool)

Returns a string of the weekday. If the 2nd parameter is 0, the three letter abbreviation is used.

- _weekdayName(1, 0) is "Sun"
- _weekdayName(1, 1) is "Sunday"
- _weekdayName(7, 0) is "Sat"

string = _monthName (number, bool)

Returns a string of the month. If the 2nd parameter is 0, the three letter abbreviation is used.

- _monthName(1, 0) is "Jan"
- _monthName(1, 1) is "January"
- _monthName(12, 0) is "Dec"

string = _FormatTime (dateTime, pattern)

Returns a string of the date-time formatted according to the pattern. The pattern is a string made up of these replacements:

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

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(2003, 3, 15, 9, 8, 3)

- Format a date-time to a string:

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

  This would show as "15-Mar-03 09:08"

Numeric formatting functions

string = _FormatNum (number, # of decimal places)

Converts the number to a string with the given number of digits after the decimal point.

_FormatNumber(1.6764, 1) evaluates to "1.6"

string = _FormatInt (number, # digits, leading zeros?)

Converts the number to a string with no fractional part. If the third parameter is supplied and if it evaluates to Yes, the string contains leading zeros.

_FormatInt(100.5,4) evaluates to " 100"

_FormatInt(100.5,4,1) evaluates to "0100"

string = _FormatPattern(number, string pattern)

This function is intended for users familiar with programming languages. The pattern uses the same pattern characters as the C language printf function. Refer to C runtime documentation or books on the programming language for full info. This documentation is not included here.

Device / Program / Group / Schedule functions

string = _status ("name")

Returns text that contains the status of the device, program, group, or controller. The text is designed to be displayed to the user rather than operated on by subsequent Compute expressions. What is returned depends upon the type of object. For example, a device returns "ON, "OFF", or "Dim xx%" where a program would show "Running" or "Not running". In no case is the device queries to fund out it's true status – the status retuned is based upon HCA's internal state.

bool = _isOn ("name", send status?)

Attempts to look up the supplied name as a device, program, group, or controller. If it is On, or is running if a program, then return Yes, otherwise No. If the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result.

bool = _isOff ("name", send status?)

Attempts to look up the supplied name as a device, program, group, or controller. If it is Off, or if a program the program is not running, then return Yes, otherwise No. If the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result

bool = _isDim ("name", sendStatus?)

Attempts to look up the supplied name as a device, group, or controller. If it is Dim then return Yes, otherwise No. f the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result

Additional note on _IsOn, _IsOff, and _IsDim functions:

If the device supports status requests, and if the optional parameter doesn't say to not use a status request, then if the device doesn't respond, a numeric code is returned rather than a BOOL.

You can test for this:

X = _IsOn("Home – Lights");

In a Compute Test element you could decide if it was not responded to by:

_IsBool(x)

If that function returns "Yes" then the status request was answered and "x" contains yes or no. If _IsBool returns FALSE then the device didn't answer the status request.

bool = _isRunning ("programName")

Attempts to look up the supplied name as a program. If it is running then return Yes, otherwise No.

bool = _isDim ("name", sendStatus?)

Attempts to look up the supplied name as a device, group, or controller. If it is Dim then return Yes, otherwise No. f the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result

number = _dimLevel ("name", sendStatus?)

Attempts to look up the supplied name as a device, group, or controller. Returns the illumination level. If it is off, then zero is returned. If On then the maximum illumination level is returned. f the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result

number = _dimPercent ("name", sendStatus?)

Attempts to look up the supplied name as a device, group, or controller. Returns the illumination level as a percentage of 100%, where 0 is Off and 100 is full bright. f the second parameter is not given, or evaluates to No, the status of the device is determined from the internal HCA state. If the parameter is supplied and evaluates to Yes, and if the device supports status requests, a status request is sent to the device and the response back determines the result

bool = _isCurrentSchedule ("scheduleName");

Returns Yes if the current schedule matches the name supplied.

string = _currentSchedule ()

Returns a string of the name of the current schedule. If no schedule is current returns an empty string.

number = _on ("name")

Turns the named device or group on. Returns the current dim level before the On command is sent.

number = _off ("name")

Turns the named device or group off. Returns the current dim level before the Off command is sent.

number = _dimToLevel ("name", level)

Adjusts the illumination level of the named device to the supplied level. Returns the current dim level before any commands are sent.

number = _dimToPercent ("name", percent)

Adjusts the illumination level of the named device to the supplied percent. Returns the current dim level before any commands are sent.

string = _CurrentScene ("name")

If the named device or controller supports stored scenes, returns the name of the scene set at the device. This function does not query the device so the result is based upon HCA internal state.

string = _SetToScene ("name", "scene name")

Changes the current scene in a switch or controller to the scene named. The current scene in the device before the new scene is established is returned.

string = _ComposeScene ("name", level)

Returns the name of the scene for the given Compose command (preset dim level) in the named device or controller. This only applies to Lightolier CI and CP devices in Compose mode.

Number = _GroupMemberCount ("group name")

Returns the number of members in a group. Used in conjunction with the GroupMemberName function to get the name for each group member.

String = _GroupMemberName ("group name", number)

Returns the name of the Nth member of the named group. To get the first member name the 2nd parameter is a 1.

String = _Trigger ()

Returns a string which represents the trigger that started the program.

NOTE: If not used in the context of a program, for example in an expression used in a Status Export it returns ""

NOTE: The function is no longer supported and should not be used in new designs

The first character of the string is the kind of trigger. These are:

X = X10
M = Magic Module
W = Weather
F = Flag
S = Special
E = Expression

I = Wireless

M, W, F, S, and E triggers are not defined in HCA 5

The X triggers are: XHUCCMDXLV

X: Always start with 'X'

H: One character House Code 'A' .. 'P'

UC: Two characters of unit code '01' .. '16'

CMDX: 4 characters of command. Right padded with blanks to get 4 characters

"ON "
"OFF "
"DIM "
"BRT "
"AUF "
"ALO "
"ALF "
"HRQ "
"HRPL"
"PDIM"
"SON "
"SOFF"
"SREQ"

LV: 2 characters of preset dim level '00' .. '31'

If any section doesn't apply it is left blank. So a status request trigger to housecode G would look like (blanks shown as '-' for clarity)

XG--SREQ--

The I triggers are: IIDXMMMMM

I: Always start with 'I'

IDX = the id in the range 1-255

MMMMM: 5 characters of message. Right padded with blanks to get 5 characters

AAMIN
AAMAX
AHMIN
AHMAX
PANIC
ALERT
NORML
LTON
LTOFF
DSARM

String = _X10NameOf (string)

This returns the name (text) of the device or controller with a supplied HC/UC.

The argument string must be formatted as a valid house and unit code. For example "A1" or "A16" or "G5", etc.

This first tries to find a device with that X10 address. If that fails controllers are searched for a match.

NOTE: For multi-unit devices / controllers only the base address is considered. If there are more than one device / controller with the HC/UC, one is found. No guarantees which one will be found.

String = _X10AddressOf (string)

This returns the primary address of the named device or controller.

The returned string is formatted as "A1" or "A16" or "G5", etc.

String = _WirelessName (number)

This returns the name (text) of a wireless component with that hardware id number.

Number = _IconChange (“name”, “icon name”, “display name”)

The first argument must be the name of a device, program, group or room. The second argument is the name of an icon from the icon gallery. The third argument is the name of a display.

This operation changes the displayed icon for an object to the named icon on the named display.

If the display name argument is not given then any displays the object appear on are affected. If the icon name argument isn’t given then the icon is restored to the one chosen in the object properties icon tab.

Number = _IconChangeEx (“name”, "code", “text", “number”)

The first argument must be the name of a device, program, group or room. The second argument is code:

- 0 = Return control of the object's icon back to HCA
- 1 = Change the object's icon to the named icon (3rd argument) with specified representation (4th argument: 0 = ON, 1 = OFF, 2 = DIM)
- 2 = Change th object's icon lavel to be the supplied text given by argument 3.

Unlike the ChangeIcon function, this function changes the icon for the named object where it appears and not just on a single display. This function always returns zero.

string = _ChangeSchedule (“name”, code, dateTime, dateTime)

Modifies the named schedule entry. The code is as follows:
0 = Change on time
1 = Change off time
2 = Change on and off time
The first dateTime argument is the time used for code 0 and 1. The 4th argument is only needed for code 2

DateTime = _AutoOffTime (device-or-room-name)

Returns the time when a device or room will auto-off. If there is no auto off timer running returns a bool of zero. Use the _IsBool on the result before assuming it is a date-time.

Thermostat

These elements are useful only when a bi-directional thermostat device has been added to your design. See the thermostat appendix for information on this.

NOTE: The next 7 functions are still available but should not be used in new designs. The _GetThermostat and _SetThermostat functions should be used instead.

number = _temperature ("thermostat name")

Returns the current temperature read from the named thermostat. The thermostat must have two-way capabilities. The returned value is in the units selected in the properties for the thermostat.

number = _setPoint ("thermostat name")

Returns the current setpoint read from the named thermostat. The thermostat must have two-way capabilities. The returned value is in the units selected in the properties for the thermostat.

string = _mode ("thermostat name")

Returns the current mode read from the named thermostat. The thermostat must have two-way capabilities. The returned value is a string of "Heat", "Cool", "Auto", or "Off"

bool = _isFanOn ("thermostat name")

Returns Yes or No if the fan is On or Off when the named thermostat is queried.

bool = _isEconomy ("thermostat name")

Returns Yes or No if the thermostat is in economy or setback mode when the named thermostat is queried.

bool = _isAuxHeat ("thermostat name")

Returns Yes or No if the thermostat is in aux heat mode when the thermostat is queried.. Not supported by all thermostats..

number = _humidity ("thermostat name")

Returns the current humidity read from the named thermostat. The thermostat must have two-way capabilities. Not supported by all thermostats.

number = _TempDecode (Unitcode, level)

Uses the standard X10 RCS temperature decode table (also used by the SmartHome TempLinc), to change a unit code and preset dim level into a temperature. For example:

_TempDecode(15, 2) = 70

The 1st argument is the unit code given as a number from 11 to 16 corresponding to UC 11 to UC 16. The 2nd argument is the preset dim level between 0 and 31, inclusive.

Value = _GetThermostat (“thermostat name”, code)

The “Thermostat name” is the two part name for the thermostat.

The code is the setting to be retrieved. These are:

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

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. You should use the _IsBool on the result to determine if you have received the requested data or an error.

Bool = _SetThermostat ("thermostat name", code, value, code , value, …)

You can supply up to 11 arguments. The 1st is the two part name for the thermostat device. The next two arguments are the code and value of the setting to change. The next arguments are up to 4 other code-value pairs. The valid codes are:

Code	Setting	Data
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
7	Cool Setpoint	Integer value
13	NEST mode (NEST only)	0 = Away, 1 = Home

Note: When changing the NEST mode it changes all thermostats in the structure associated with the thermostat being controlled.

Weather

The weather functions are useful only if you have set up a weather provider using the weather provider setup dialog. See the weather appendix for information on this.

The _Weather function allows you to access all data in a weather observation. For observation data that have units – temperatures for example - the values returned are in the units selected in the weather provider setup dialog. The _Weather function format is:

value = _weather (name)

The "name" is the name of data from the weather observation. Since the names change as more weather providers are supported and as those providers change, the documentation of those names is on the HCA web site in the weather technical note.

Hint: The support web site is at www.HCATech.com. Look for the technical notes link on the main page.

In addition to the _weather function, other functions compute their results by examining data in the historical weather log files. If you don't have a log file setup in the weather setup dialog none of these functions work.

The "minutes, hours, days" arguments tell how far back in time to go to determine a high, low, or average.

The units used are the same as the analogous current weather item. That is, all temperatures are returned in the same units.

When using these functions it is not necessary to supply all three arguments. For example, you can determine the high outside temperature in the last 90 minutes from this expression

temp = _weatherHigh(OutsideTemp, 90)

The functions are:

Value = _WeatherHigh(name, minutes, hours, days)

Value = _WeatherLow(name, minutes, hours, days)

Value = _WeatherAvg(name, minutes, hours, days)

In addition to these functions, there are two other sets of weather functions. These also compute their result by looking at historical weather data. But in this case the argument is not a length of time but rather the number of days to go back in time.

For example if today is Monday:

_weatherDayHigh(Barometer, 0) is the barometer high for Monday

_weatherDayHigh(Barometer, 1) is the barometer high for Sunday

These functions are:

Value = _WeatherDayHigh(name, day)

Value = _WeatherDayLow(name, day)

Value = _WeatherDayAvg(name, day)

The final set of functions is like the Day function except the parameter is an hour. For example is it is 3:15pm:

_weatherHourHigh(Barometer, 0) is the high barometer from 3pm to 4pm

_weatherHourHigh(Barometer, 1) is the high barometer from 2pm to 3pm

string = _BarometerUnits()

Returns a string of the current barometer units. For example "in" or "mm"

string = _humidityUnits()

Returns a string of the current humidity units. Always "%"

string = _tempUnits()

Returns a string of the current temperature units. For example "° F" or "° C"

string = _windSpeedUnits()

Returns a string of the current wind speed units. For example "m/s" or "knots"

string = _windDirUnits()

Returns a string of the current wind direction units. Always "°"

string = _wunderground([1 to 10 strings])

Retrieves data from the Weather Underground internet site. It must be configured as your weather provider for this function to operate correctly. This function is somewhat complex and is described in the Weather user guide appendix.

string = solarRadiationUnits()

Returns a string of the current solar radiation units. Always "W/sq m"

string = _UVUnits()

Returns a string of the current UV units. Always "UV index"

string = _soilUnits()

Returns a string of the current soil units. Always "c"

number = _tempConvert (number, fromUnits, toUnits)

Converts a temperature between F and C. The first parameter is the temperature. The 2nd parameter is the current units and the 3rd parameter is the units wanted. The encodings of the units parameters are:

F = 0
C = 1

number = _barometerConvert (number, fromUnits, toUnits)

Converts a barometer measurement between the four supported units. The first parameter is the barometric reading. The 2nd parameter is the current units and the 3rd parameter is the units wanted. The encodings of the units parameters are:

Inches = 0
Millimeters = 1
Millbars = 2
Hecto Pascals = 3

number = __windSpeedConvert (number, fromUnits, toUnits)

Converts a barometric reading between the four supported rates. The first parameter is the wind speed. The 2nd parameter is the current units and the 3rd parameter is the units wanted. The encodings of the units parameters are:

Miles per hour = 0
Knots = 1
Kilometers per hour = 2
Meters per second = 3

number = _rainConvert (number, fromUnits, toUnits)

Converts a rain amount between the two supported units. The first parameter is the rain amount. The 2nd parameter is the current units and the 3rd parameter is the units wanted. The encodings of the units parameters are:

Inches = 0
Millimeters = 1

string = _windDirection (number)

Changes a wind direction in degrees into a string of the form, N, NNE, NE, ENE, E, etc.

string = _wunderground(…)

Gets information from Weather underground. See the weather underground technical note for use.

File Operations

This category of functions comprises a set of functions that operate on disk based files. HCA allows a maximum of 4 files to be open at one time.

Bool = _FileExists(string)

Determines if the supplied file path exists. Returns true or false.

number = _FileOpen(string, number)

Opens the file at the supplied path and returns a "handle" to the opened file that can be used in the Read, Write, and Close functions. The second argument is a code:

Code	Use
0	Open file for reading
1	Open file for writing. If the file already exists delete it's contents.
2	Open file for writing. If the file already exists, append new data to the end of the file

number = _FileClose(number)

Close the file. The argument supplies the handle returned by the FileOpen function.

number = _FileWriteString(number, string)

Writes the string to the file. The first argument is the handle of the file returned by FileOpen. The return value is the length of the string written.

string = _FileReadString(number)

Reads a string file from a file and returns it. The first argument supplies the handle of the file returned by FileOpen.

JSON

JSON is a method of encoding data. HCA has several functions that work with JSON. These are defined in a technical note. Refer to that for complete information.

Miscellaneous

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

number = _problemLevel ()

Returns the alert level as displayed by the red-yellow-green status indicator on the status bar. Red is 2, yellow is 1 and green is zero.

number = _SetProblemLevel (number)

Changes the alert level as displayed by the red-yellow-green status indictor on the status bar to the value given. The existing level is returned.

number = _delay (time1, time2)

Delays execution of the program for a given time. If one argument is given the delay is the total number of seconds in time1. If two arguments are given a random time is chosen between the two given times. The return value is the number of seconds delayed.

number = _DelayShort (millseconds)

Delays execution of the program for a given time. The time is in milliseconds. The return value is the number of milliseconds delayed.

number = _SetHomeMode(number)

Changes the home mode to the mode supplied. The home mode is given as in index into the modes you select in the *Control* ribbon category *Current Home Mode* dropdown. The first made in the list is zero, the second is 1, etc. The return value is the mode that was changed from using the same numbering.

number = _HomeMode()

Returns the current home mode. Uses the same numbering as in SetHomeMode described above.

Number = _AlertAdd (alert #, text)

Raises an alert in the Alert Manager. The Alert manager lets you configure for user alerts. These four “user alerts” are there to tie into this compute function. What happens with the alert – does it log, does it change the alert level – is all part of the alert configuration. The _AddAlert function only causes the alert to be raised - what happens is up to the alert configuration. Always returns 0.

string = _DesignTitle ()

Returns the design name as set in Home Properties.

string = _DesignTitle ()

Returns the name of the design as set in Home Properties

string = _DesignSave (number)

Saves the current design. The single argument is optional. If not supplied the design is saved unconditionally. If supplied and is zero, the design is saved unconditionally. If supplied and set to 1, the design is saved only if marked as modified.

string = _CurrentWattage (string)

Returns the wattage currently used by the named object. That name could be a device name or room name. To get the current wattage of the whole home, don't supply any argument.

number = _CurrentWattage (string, number)

Sets wattage of the named object to the value. That name could be a device name or room name. Always returns 0.

dateTime = _LastReceptionTime (string)

Returns the time of the last reception from the named device. If the supplied string names a room then the time returned is the last reception for any device in the room./

number = _InsteonBeep (string, number)

Sends a command to an Insteon device that could cause it to "beep". Not all Insteon devices support this command.

number = _RGB (number, number, number)

Returns a number that is the RGB value of the 3 color components.

number = __InsteonBeep (string, number)

Sends a command to an Insteon device that could cause it to "beep". Not all Insteon devices support this command.

Bool = _TileUpdate (string, number, any, number)

Update the tile named by the first argument. The second argument is a code and that determines the use of the next two arguments. Returns Yes if the operation worked.

Code	Use	Arg3	Arg4
0	Change label	Label text	Not used
1	Change colors	Background color	Text color
2	Change image path	Image path	Not used
3	Change text	Text	Not used
4	Refresh	Not used	Not used