HCA Tech Note 600

User Implemented Device Classes – for Class Creators

User Implemented device classes allow support for devices that are not supported in the base HCA product – like Insteon, UPB, and Phillips Hue are. A device of a user implemented class is controlled by a program created with the Visual Programmer and identified to HCA as the program to use when devices of that class are controlled. It is assumed that program communicates with the target hardware using the HTTP element or the Generic IP or Generic Serial interfaces using the Port I/O element.

The advantage of this mechanism is that once the class is added to HCA then all the “regular” facilities in HCA that work on of the built-in types are now available for objects of the added class. For example, a device of a class implemented as a “dimmable device” can be controlled by the UI with the right-click menu, controlled in the control UI and mobile applications with a tap, used in a schedule, controlled by programs using the ON, OFF, DIM, and MULTI elements. Any place a dimmable device of a built-in protocol can be used, a device of an added class can be used.

Each class implements one of three basic types:

- Dimmable device
- Non-Dim device
- Thermostat

Note: This is a very advanced topic and is intended for experienced HCA users. In HCA you need to be familiar with parameterized programs, local and global variables, and the facilities used for the communication method the devices you are implementing work with. If the devices communicate by HTTP then you must be familiar with the HTTP element. If they communicate over a serial or IP port then you must be familiar with generic interfaces and the Port I/O element.

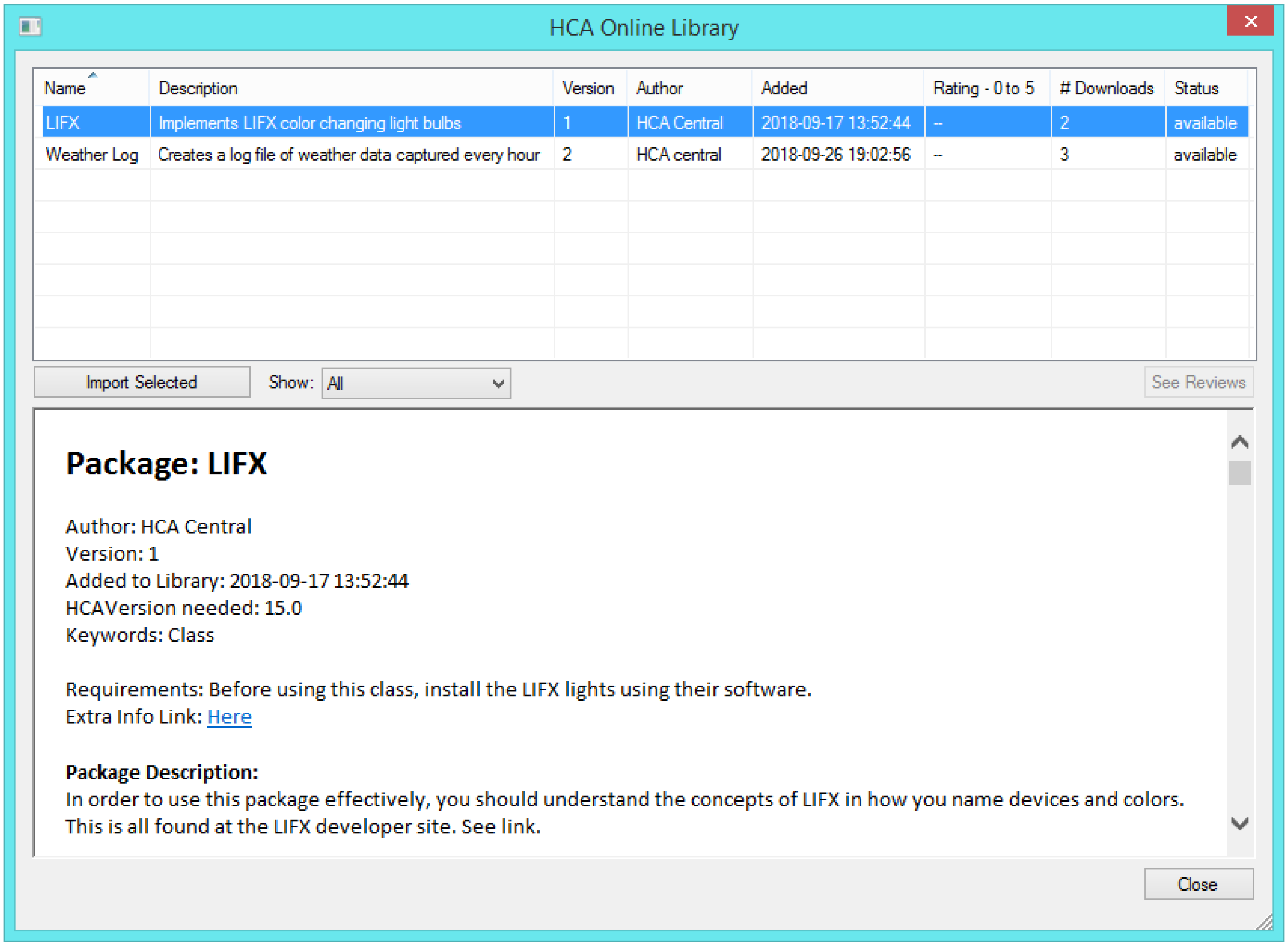
Each class being developed, unless you intend to not share it with anyone, must be documented so that other users know what to do. Documentation should include:

- The name of the class.
- What the class is for – what hardware is now supported by HCA once this class is added.
- How to import the class.
- How to get started with the class, that is, any configuration needed by a user.
- How to add devices of the class and how to identify them.
- What HCA operations work as expected and which may not. For example, all user classes assume that they support some form of status pooling but your device may not and that would be documented.
- A description of any programs that are part of the class that a user can use when implementing more than what is required by HCA for the class. For example, while every device class must implement an ON and OFF, the class might have an operation that sets a light’s color. Having a program to do that would be useful and should be part of the class import. You must document what those programs are and how the user can use them.

Much of this documentation is captured in the notes of each program that is part of the class package.

Managing classes

All classes are stored in the online library. To access the library, from the *Tools* ribbon category, press the *Library Browse/Import* button.



Name	Description	Version	Author	Added	Rating - 0 to 5	# Downloads	Status
LIFX	Implements LIFX color changing light bulbs	1	HCA Central	2018-09-17 13:52:44	--	2	available
Weather Log	Creates a log file of weather data captured every hour	2	HCA central	2018-09-26 19:02:56	--	3	available

Package: LIFX

Author: HCA Central
Version: 1
Added to Library: 2018-09-17 13:52:44
HCAVersion needed: 15.0
Keywords: Class

Requirements: Before using this class, install the LIFX lights using their software.
Extra Info Link: Here

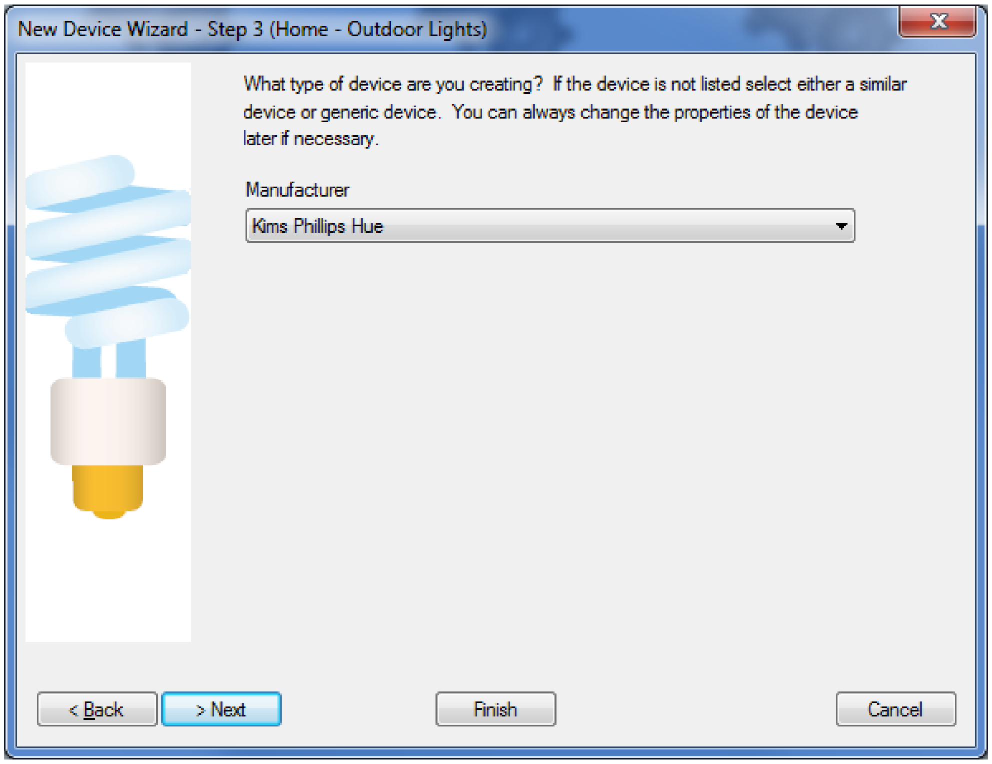
Package Description:
In order to use this package effectively, you should understand the concepts of LIFX in how you name devices and colors. This is all found at the LIFX developer site. See link.

All the user classes used in the current design are listed showing their name, version, and other library info. To see just the classes, you can select *Only Classes* from the *Show* dropdown.

When the class was created, the author provided the documentation needed for you to use it. If you import the class that documentation – a HTML file – is saved in the library folder for your later use. A popup message shows you where that is.

Adding devices of a class

Once a class is added to HCA, then the Add Wizard allows for adding objects of that class. For example:



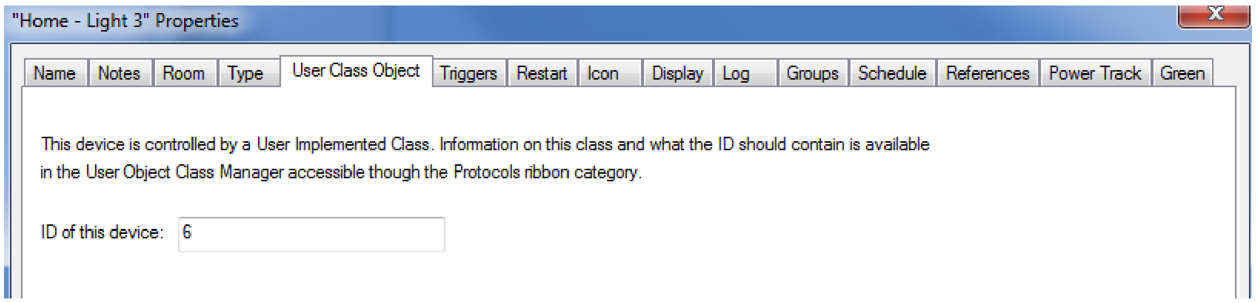
New Device Wizard - Step 3 (Home - Outdoor Lights)

What type of device are you creating? If the device is not listed select either a similar device or generic device. You can always change the properties of the device later if necessary.

Manufacturer: Kims Phillips Hue

Under "Manufacturer" are all the built-in manufactures and the names of any classes that have been added to the design.

In the properties of a device that is an object of a class, there is a tab where information that identifies this particular device is entered.



"Home - Light 3" Properties

This device is controlled by a User Implemented Class. Information on this class and what the ID should contain is available in the User Object Class Manager accessible though the Protocols ribbon category.

ID of this device: 6

For example, suppose you created a class that implements a device which is a light that has an IP address. You may have many of these lights in your home and HCA needs some way to identify each one. Again, depending upon how these devices work you could put as the ID the IP address of that particular device.

What the ID is completely up to the class implementation – it is just an unstructured string. What the user should put in the ID is also something the class creator must document. Class users should read and follow the class documentation on the contents and format of the id.

And that's it for the class user. Once added there is nothing else they need do except use the devices.

Creating Classes

It is assumed that only sophisticated user will undertake creation of classes. But with the advent of so many new device types many using a HTTP interface it is hoped that users can be nudged into doing this and then sharing.

A class is created as one or more HCA programs and then exported as a package for addition to the online library. In the package is a class implementation program that HCA uses to control devices of the class on, off, set to a percent, and retrieve state. In the package there may also be other programs that supplement the class and would be used by the user in programs that they create using the Start-Program element.

A class implementation program takes 4 parameters. These are:

1. Device Name. This is the 2-part name as in *room name – device name*.
2. Device Id. This is the ID that identifies a specific device as was entered in the device properties
3. Code. A code that tells what to do
4. Data. Any data needed by the operation

The parameters can be named anything wanted but they must appear in this order.

Important notes about class programs

In general, any variables used by any of the class package programs should be local except for when needed for configuration. For example, a global variable that holds connection information for an interface that will be different on each users system.

The first local variable – name it whatever you want - is the result from the program that communicates back a result to the HCA facilities using the class. As all HCA variables are un-typed, the type of data placed into it is important. If it is a number, the op is assumed to have worked. If it is text, it is an error message that can be shown to the user or put in the log.

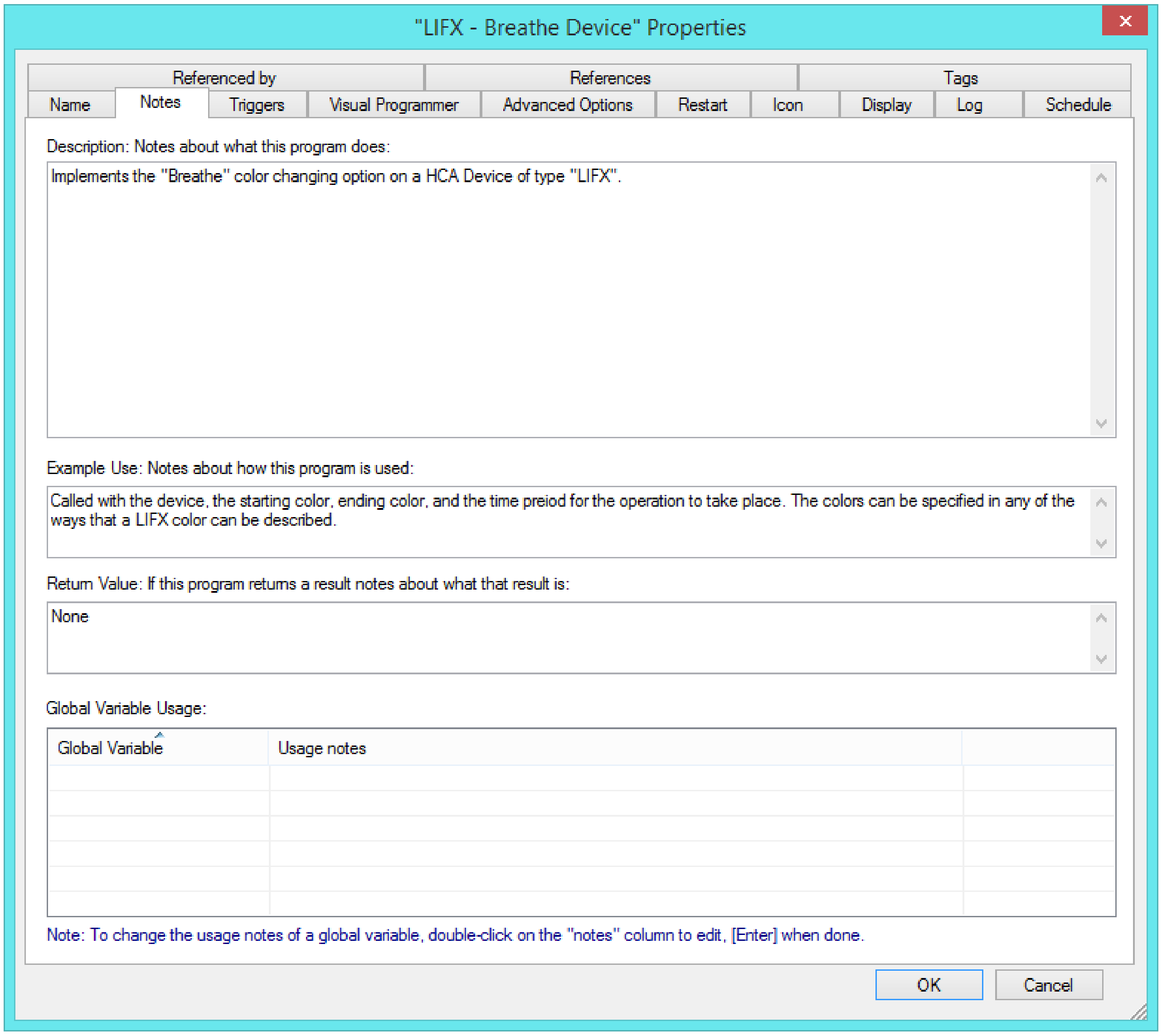
The class program executes in the same thread as whatever is using it – the scheduler, another program, or, most importantly, the user interface. As such the program cannot execute for very long. Part of the class definition contains the time limit for the implementation program execution. If the program runs for longer than this, HCA assumes that it has failed and will kill it.

Don’t forget that this is just a regular HCA program. You might want to disable logging for it when it is fully developed before passing on to others.

Class documentation

Much of information displayed in the library browser, what becomes the documentation of the class - comes from information input during development of the class programs. This information is contained in the *Notes* tab and from the program Begin-Here element’s properties.

The *Notes* tab provides unstructured text you can use to describe what the program does and how it can be used.



Description: Notes about what this program does:

Implements the "Breathe" color changing option on a HCA Device of type "LIFX".

Example Use: Notes about how this program is used:

Called with the device, the starting color, ending color, and the time preiod for the operation to take place. The colors can be specified in any of the ways that a LIFX color can be described.

Return Value: If this program returns a result notes about what that result is:

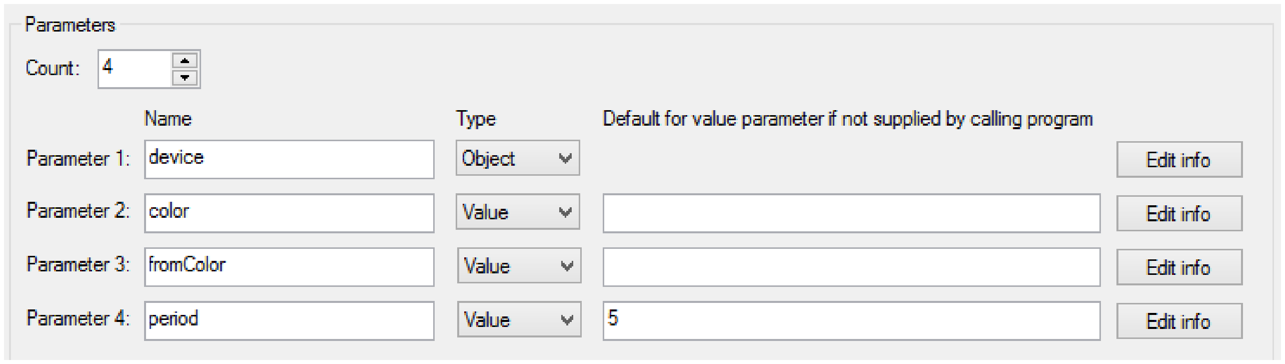
None

Global Variable Usage:

Global Variable	Usage notes

Note: To change the usage notes of a global variable, double-click on the "notes" column to edit, [Enter] when done.

The Begin-Here element of the program shows any parameters that the program uses and their default values.

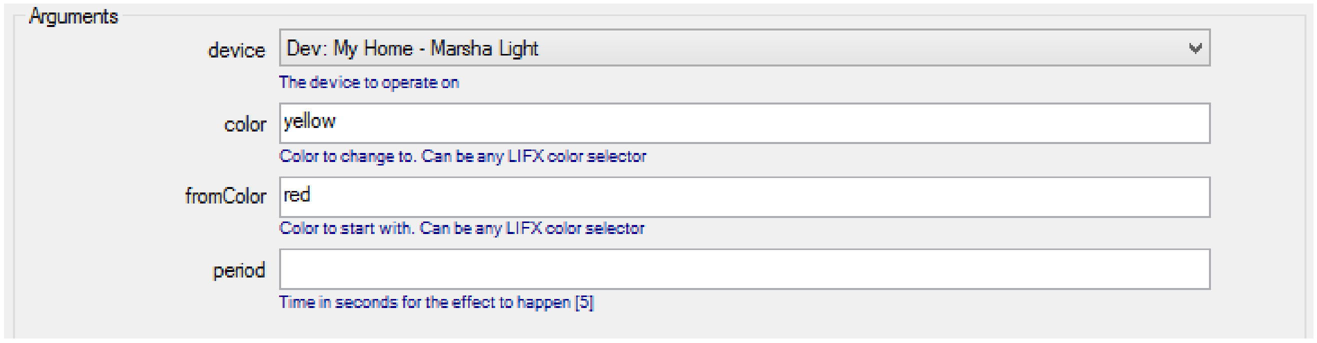


Parameters

Count: 4

	Name	Type	Default for value parameter if not supplied by calling program	
Parameter 1:	device	Object		Edit info
Parameter 2:	color	Value		Edit info
Parameter 3:	fromColor	Value		Edit info
Parameter 4:	period	Value	5	Edit info

By adding info about each parameter, when a user works with the class and uses the Start-Program element, they will see that "helpful" info in blue below where the argument values are entered. Default values show in square brackets at the end of the text.

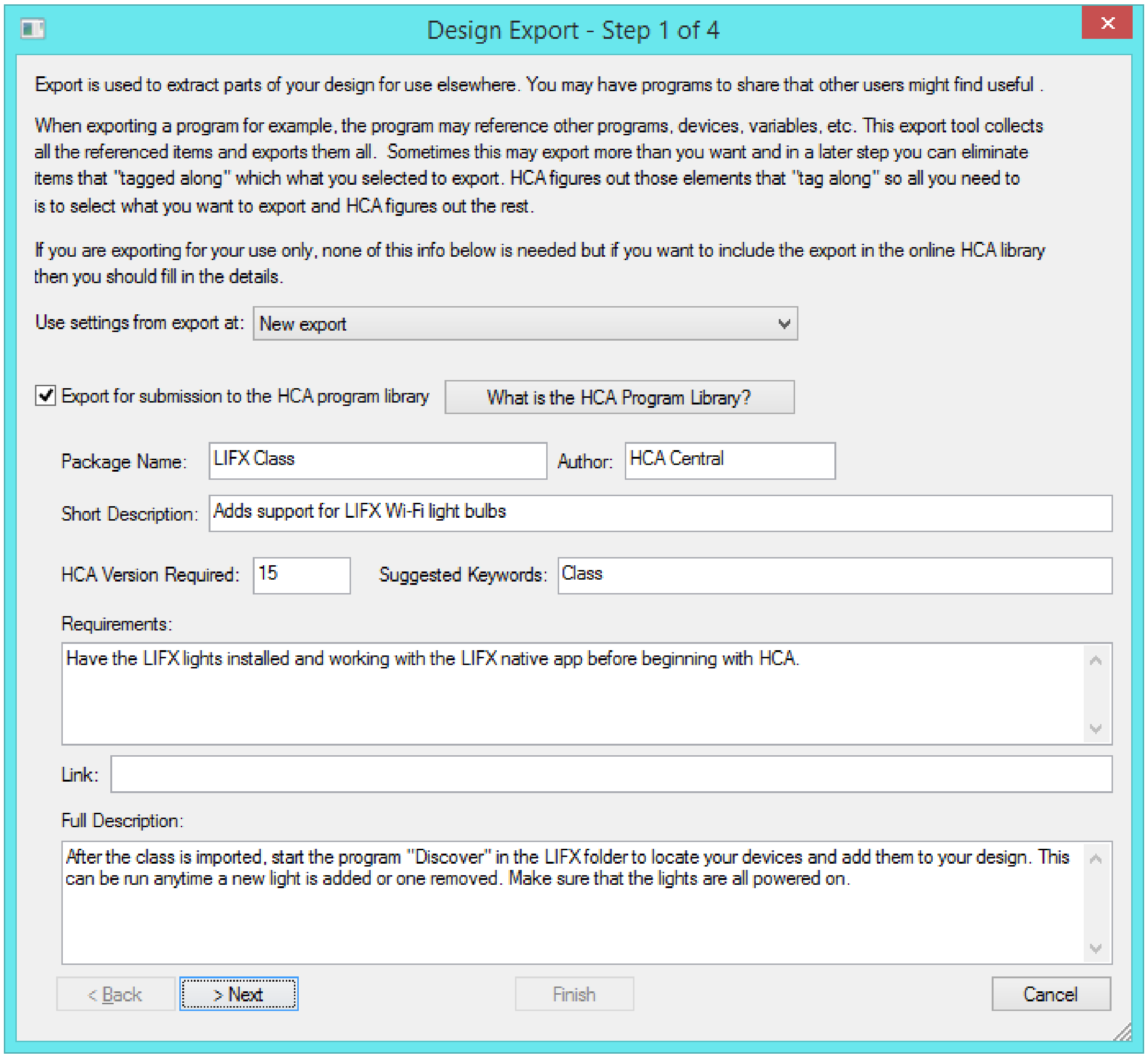


Arguments

device	Dev: My Home - Marsha Light The device to operate on
color	yellow Color to change to. Can be any LIFX color selector
fromColor	red Color to start with. Can be any LIFX color selector
period	 Time in seconds for the effect to happen [5]

Tip: Use these facilities for adding information about the program, parameters, and any global variable information.

The final set of information about the class comes during the class export process.



Design Export - Step 1 of 4

Export is used to extract parts of your design for use elsewhere. You may have programs to share that other users might find useful .

When exporting a program for example, the program may reference other programs, devices, variables, etc. This export tool collects all the referenced items and exports them all. Sometimes this may export more than you want and in a later step you can eliminate items that "tagged along" which what you selected to export. HCA figures out those elements that "tag along" so all you need to is to select what you want to export and HCA figures out the rest.

If you are exporting for your use only, none of this info below is needed but if you want to include the export in the online HCA library then you should fill in the details.

Use settings from export at: New export

[✓] Export for submission to the HCA program library

Package Name: LIFX Class

Author: HCA Central

Short Description: Adds support for LIFX Wi-Fi light bulbs

HCA Version Required: 15

Suggested Keywords: Class

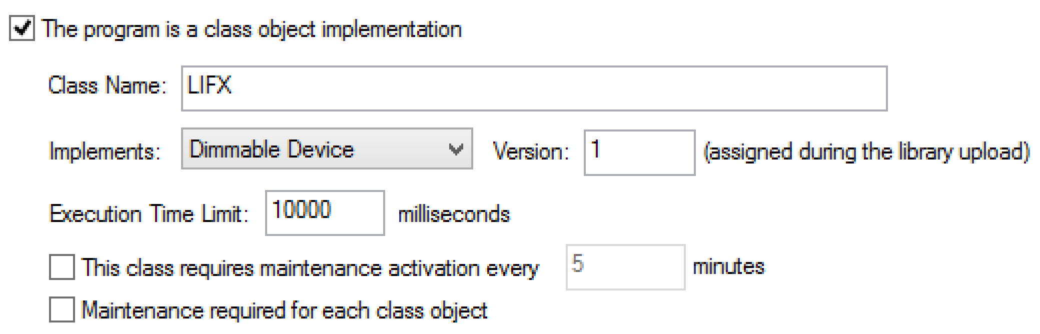
Requirements: Have the LIFX lights installed and working with the LIFX native app before beginning with HCA.

Link:

Full Description: After the class is imported, start the program "Discover" in the LIFX folder to locate your devices and add them to your design. This can be run anytime a new light is added or one removed. Make sure that the lights are all powered on.

The Class Implementation program

As described above, a class implementation program is only a HCA program with a fixed set of parameters and a defined set of actions when the program is executed. To turn a “regular” HCA program into a class program, settings on the Advanced Options tab of the program’s properties is used. These options are:



☑ The program is a class object implementation

Class Name: LIFX

Implements: Dimmable Device Version: 1 (assigned during the library upload)

Execution Time Limit: 10000 milliseconds

☐ This class requires maintenance activation every 5 minutes

☐ Maintenance required for each class object

The class name is the most important item as that is the class name users will work with. As described in the next sections, a class can support a dimmable device, a non-dimmable device, or a thermostat.

The execution time limit is the time that HCA allows the program to run to carry out the requested action. If this is set too long then the user's use of the class with seem to "stall" if the action can't eb completed in that time.

Class maintenance is described below.

Dimmable and Non-Dim Device Classes

A Generic device is implemented with these action codes:

- Code 0: Class maintenance
- Code 1: On
- Code 2: Off
- Code 3: Set percent
  The data parameter supplies the percent (0 – 100)
- Code 4: Change percent
  The data parameter supplies the delta percent (+ or -) to change the level
- Code 5: Get Status
  Queries the device and returns its state.

The program must keep the HCA state of the device up to date as it changes it. There are expression facilities for this: _SetCurrentState (<device name>, <percent>). A side effect of that function is that the device icon is updated to reflect the new state.

The return value – the value assigned to the first local variable - from the program for codes 1 to 4 should be zero if it worked or an error string if not. For code 5 the return value is the device percentage (0 – 100) or an error string if the status request failed.

Generic Thermostats

A Generic Thermostat is implemented with these action codes:

- Code 0: Class maintenance
- Code 1: Get capabilities
- Code 2: Get mode
- Code 3: Get heat set point
- Code 4: Get cool set point
- Code 5: Get temperature
- Code 6: Get humidity
- Code 7: Get fan
- Code 8: Set mode
- Code 9: Set heat set point
- Code 10: Set cool set point
- Code 11: Set Fan
- Code 12: Get Status

Code 1 – Get Capabilities – returns an CSV encoded string that provides the overall capabilities of the thermostat. In most cases this string can be assembled as a constant. The pieces of the string are:

- # modes
- Mode string 1, Mode string 2, … Mode string n
- # fan settings
- Fan String1, Fan string 2, … Fan string n
- Heat range min
- Heat Range Max
- Cool range min
- Cool range max
- F or C
- Has humidity

For example, suppose a thermostat has these capabilities:

- Modes: “off”, “heat”, “cool”, and ”auto”.

- Fan: “auto”, “on”.
- Heat range min: 40
- Heat range max: 100
- Cool range min: 50
- Cool range Max: 90
- The thermostat is showing temperatures in degrees F
- The thermostat is capable of reporting humidity

The encoded string would be:

4,off,heat,cool,auto,2,auto,on,40,100,50,90,F,1

For all the other codes besides code zero, when the program completes the 1st local variable contains the result. That result is either a string, in which case it is an error message, or a number representing the data.

- Code 2: Get mode
  result = The mode as a number. Can be translated to a string using the capabilities string. For his example, if 1 was the result that would be the “heat” mode

- Code 3: Get heat set point
  result = temperature of heat set point

- Code 4: Get cool set point
  result = temperature of cool set point

- Code 5: Get temperature
  result = temperature of room

- Code 6: Get humidity
  result = humidity of the room

- Code 7: Get fan
  result = The fan setting as a number. Can be translated to a string using the capabilities string. For this example, if 0 was the result that would be the “auto” fan setting, 1 would be the “On” setting.

- Code 8: Set mode
  arg4: The mode as a number
  result = 0

- Code 9: Set heat set point
  arg4: The heat set point as a number
  result = 0

- Code 10: Set cool set point
  arg4: The cool set point as a number
  result = 0

- Code 11: Set fan
  arg4: The fan setting as a number. 0 = auto, any other value = On
  result = 0

- Code 12: Get Status
  This code is invoked if the HCA device has the “poll for status” setting enabled. What exactly is polled – what this code does – is up to the class implementation. It is invoked based upon the device “poll time” properties as set by the user.

Any specific issues relating to how the thermostat functions must be implemented in the class. For example, some thermostats only let you change the set-point for the mode that the unit is in – that is, you can’t change the heat set-point unless in heat mode. HCA does no error checking for these sorts of limitations and the class must either detect the limitation and provide an error message or if the unit itself reports error then pass that error back into HCA.

Class Maintenance

Depending upon the hardware being used it may be necessary for the class to be invoked periodically, for example, to update communication parameters. These settings are among the class options on the program’s *Advanced Options* tab.

If class maintenance is enabled, then the class program is executed by HCA periodically every ‘n’ minutes. If maintenance is required but not for each object, then when the class program is invoked the device name is set to “” and the device id is zero.

If maintenance is required for each class object, then the class program is invoked once for each object in turn and supplied with the device name and id of each device as usual.

Bigger Classes

There is no reason why a class writer couldn’t implement other aspects of the hardware in other programs beyond the facilities needed by the built-in HCA operations. When a class is imported it really is just a HCA import with some special handling.

In the example class – see below - which implements Phillips Hue bulb control it does nothing with color as there are no HCA built-in operations for color – you can click on a light and turn it ON but there is no UI to turn it red. Other programs could be included in the class import that implements controlling color in a device. A design that imports this class could use those other programs by using the start-program on it from their own programs.

Handling Errors

Error handling is totally up to the class implementation. How the controlling hardware reports back an error is something that the class creator needs to understand and implement. The JSON functions available in the Compute element, might be useful here if that is how the hardware reports errors.

If the implementing program uses the Port I/O or HTTP element these are facilities in that element to set timeouts and handle timeout conditions. Well implemented classes will use those facilities

Examples

There are several example classes that you can download and look at for more information on how this all works.

Device example

The example for a device implements HUE Bulbs (even though they are supported as a built-in protocol). This is available in the online library.

Thermostat example

The example of a thermostat implementation has no actual hardware support. All the class does is to set various variables with the setpoints. It is available in the online library.

##end##