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 X10, Insteon, UPB, and Phillips Hue is. 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 new 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

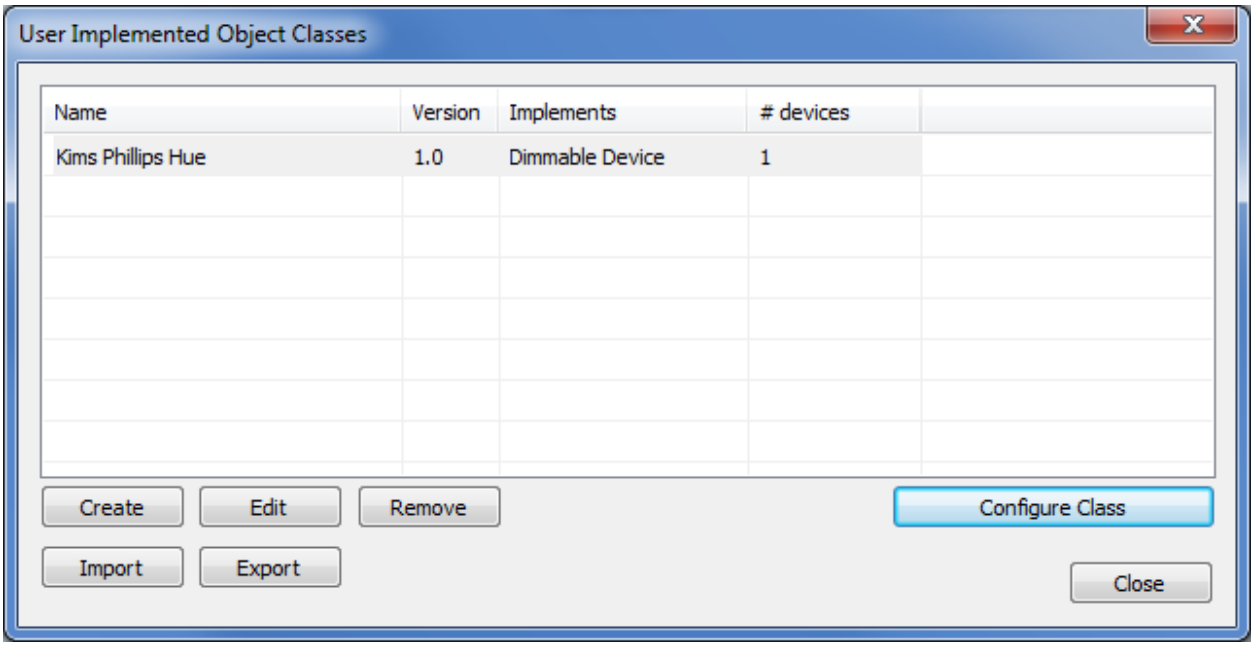
Note: This is a very advanced topic and is intended for experienced HCA users. In HCA you need to be familiar with parameterized programs, local and global variables, and the facilities used for the communication method the devices you are implementing work with. If the devices communicate by HTTP then 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 configure the class – step by step.
- 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, you 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.

Managing classes

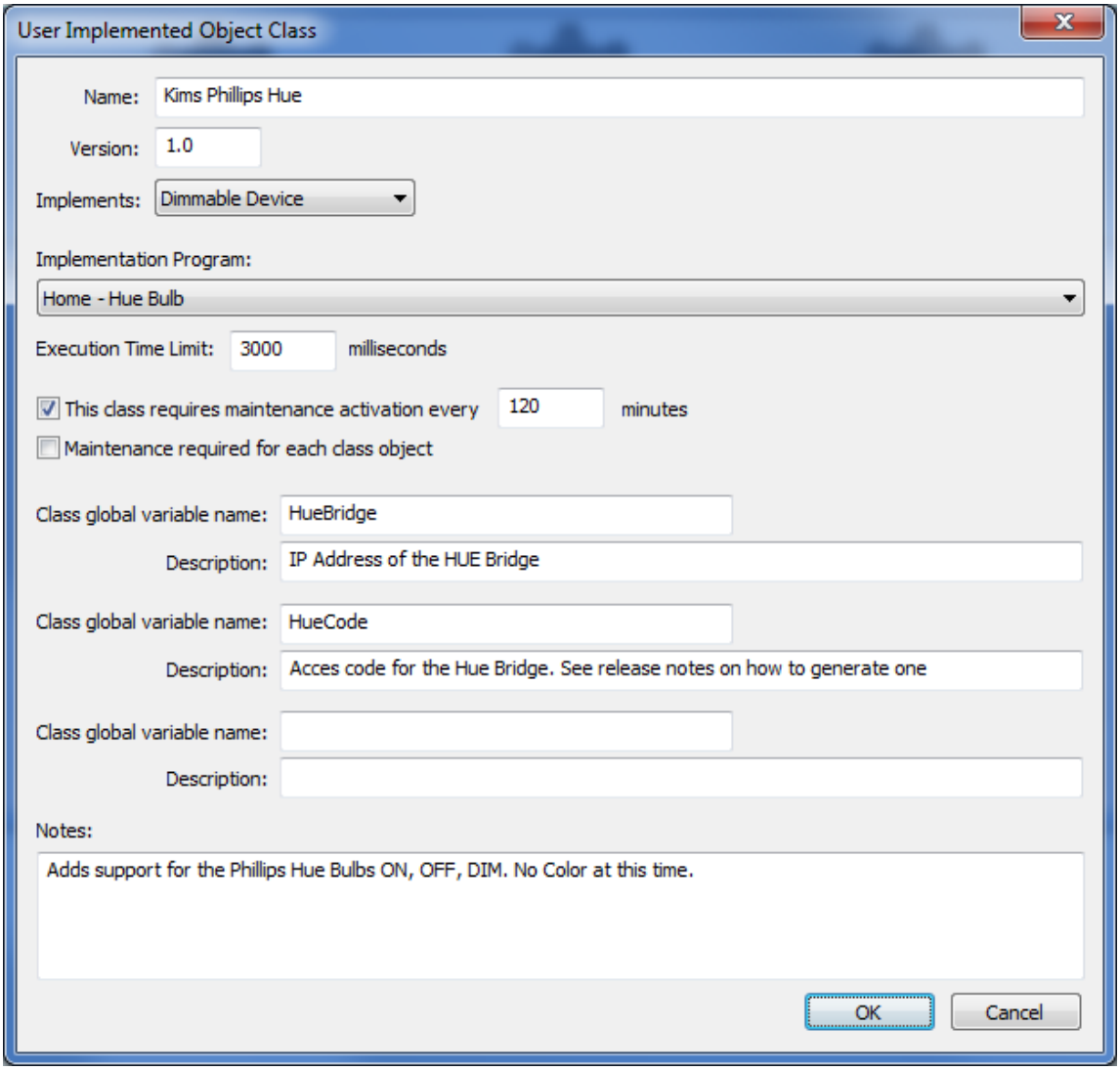
In the *Protocols* ribbon, there is a *User Class Manager* button that provides facilities to add classes to a design either by creation or import.



All the user classes used in the current design are listed showing their name, version, what it implements, and the number of devices of that class currently in the loaded design.

From this dialog, you can create a new class, edit an existing class, or remove it. There is also an export facility which exports the class into a HCLASS file for import into other designs. The Import of a class can also be done from this dialog.

A class contains these properties:



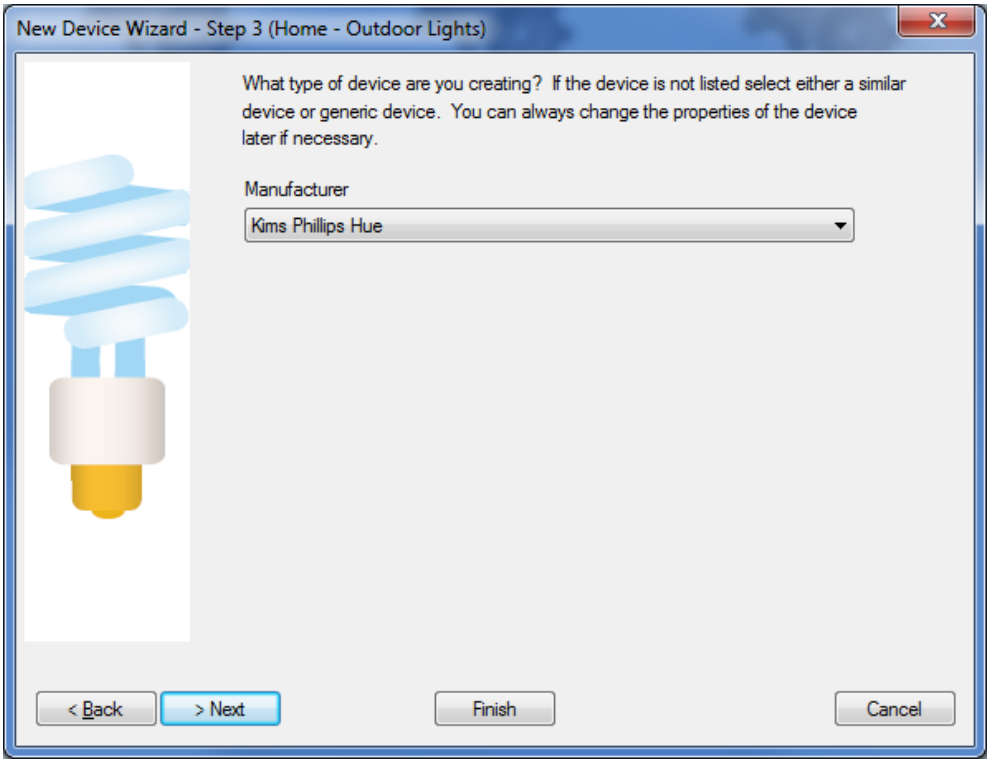
The name can be anything that identifies this class. When the New Device Wizard is used to add objects of this type to the design, the Add Wizard shows this name.

The version is anything you want. If a class is imported that has the same name as an existing class a warning is given showing both the new version and old version and the user can choose to accept or not to accept replacing the existing class implementation with the new one.

The class is implemented by a HCA program the class creator develops. The program accepts four parameters and carries out the requested operation and returns a result. The parameter names and their use is documented later in this note.

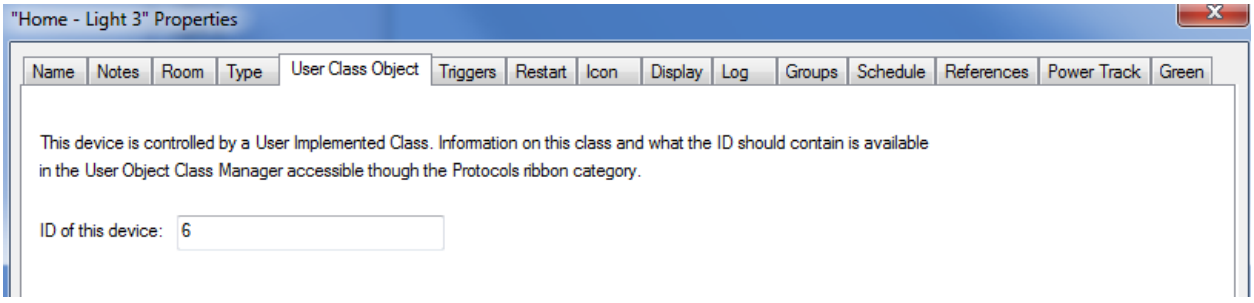
Adding devices of a class

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



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.



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.

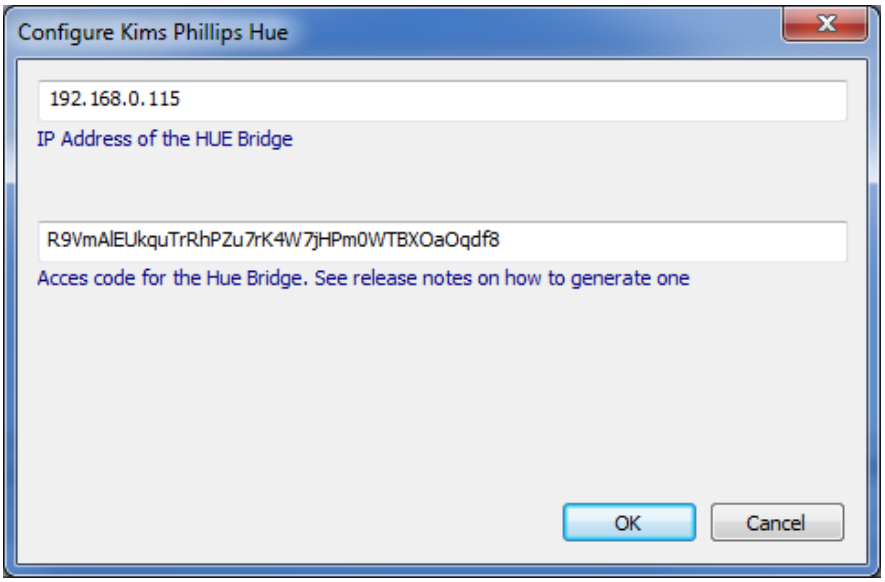
What the ID is completely up to the class implementation – it is just an unstructured string. What the user should put in the ID is also something the class creator must document.

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

Configuration of the class

The devices the class implements probably requires some supporting hardware and for the class to function it may need to know installation specific information. For example, an implementation of lighting system may need an IP address of a controller and the user account code. This sort of information can be held in global variables that the class consumer – the design that imported the class - must set with the appropriate values for their installation.

In the Class Manager, there is a *Configure Class* button that displays to the user a dialog showing edit controls labeled with the description. When that dialog is closed then global variables are set with the data entered by the user.



The thinking here is that the class consumer would add the class then configure it relying upon documentation provided by the class writer.

Configuration of the class devices

What was described above handles all that is needed for some but not all classes. For example, a class where each device of that class has its own IP address.

In that case the class configuration described above isn't all that needs to be accomplished. What would be done in this case is for the ID information contained in each device – which to HCA is just opaque text – must contain something that identifies that particular device.

For example, suppose that each device is located by an IP address and port # and these are different for each instance of that class. The text entered by the user into the device id field would be that IP address and port number. For example, as "192.168.0.100,5016". As described in the next section, when the class program is invoked for a device, this id string for the device is passed as a parameter to the class program. The program then must use the HCA string functions to extract the IP address and port from the string into local variables and use those where needed.

Remember that it is the job of the class implementer to document the format of the id field that the user must enter when they create a device of that class.

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 in the Class Manager and then can be exported. The key piece of the class is the implementation program. A class 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

Depending upon usage, there may be multiple copies of the program running simultaneously so if any variables use in the program should be local except for the class global variables which would only be read from and not modified.

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.

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 from the program for codes 1 to 5 should be zero if it worked or an error string if not. For code 5 the program must update the device state retrieved using the _SetCurrentState function.

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

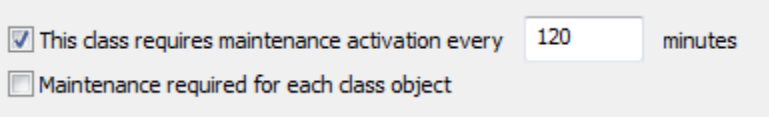
- Code 11: Set fan
  arg4: The fan setting as a number. 0 = auto, any other value = On
  result = 0

- 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. The class definition has two checkboxes for this:



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

Example

There are two 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 in HCA 14). The zip file contains two files:

- User class device example.hca
- User class device example.hclass

The HCA file contains the implementation and some testing functions. The class implementation program is in the folder "Class". In the folder "Test" is an example device of the class are two programs that call upon the class implementation directly - which would only be done during class development and testing. The remaining program "On and Off by program element" is the usual way that user class devices are controlled.

The HCLASS file is the class exported in a way that class users will normally see the class – ready for import into their designs.

This example can be downloaded from here:

www.HCATech.com/download/V14/Classes/ExampleDeviceClass.zip

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. The zip file contains two files:

- User class thermostat example.hca
- User class thermostat example.hclass

The HCA file contains the implementation and some testing functions. The class implementation program is in the folder "Class". In the folder "Test" is an example device of the class and a program that calls upon the class implementation

directly - which would only be done during class development and testing. The remaining program “Change Setpoint” is the usual way that user class devices are controlled.

The HCLASS file is the class exported in a way that class users will normally see the class – ready for import into their designs.

www.HCATech.com/download/V14/Classes/ExampleThermostatClass.zip

##end##