How to add new classes

This document describes everything you need to do to add a new class. We assume that you have the object designed and know how to program the C64 and host behaviors to make it do what it’s supposed to do. This is simply a guide to the various things that need to be updated to incorporate a new class.

General:

1) Pick a name for the new class. This should be short and pithy but descriptive. For purposes of the examples below, we will use the name "foo"

2) Pick a class number for the new class. Of course, this should not belong to any existing class of object. We try to assign class numbers sequentially, but due to the deletion of certain classes as the project has evolved, there are some "holes" in the number space. We are now filling these holes up. We follow a general rule that classes 0-127 are for portable objects or ones which are irremovable features of the universe, while classes 128-255 are non-portable objects that are potentially unique to the initial Habitat world. Unfortunately, this rule has not been applied with absolute rigor; nevertheless, try to follow it anyway. For purposes of the examples below, we will use the number 47. DO NOT USE THIS NUMBER YOURSELF.

C64:

3) Create artwork for the new object. This is usually a matter of bugging Gary until he does it. The imagery should go in files in the directory "/u0/habitat/Beta/Images". These files should have names like "foo0.bin", "foo1.bin", etc.

4) Program any new behaviors that the new class may require (classes can usually be created using existing action code for most behaviors). The new behavior code should be placed directory "/u0/habitat/Beta/Actions". Each new action that an object supports will probably require two routines, one for the synchronous case and one for the asynchronous case. The synchronous cases should go in files with names of the form "class_verb.m" ("class_verb.bin" for the binary). For example, "foo_do.m" or "foo_get.m" would be the DO and GET behaviors for class foo. The asynchronous cases should go in files with names of the form "class_ACTION.m". For example, "foo_ADUMBRATE" would be the asynchronous ADUMBRATE behavior for class foo.

4a) Update the Makefile in "/u0/habitat/Beta/Actions" to incorporate the new behaviors.

5) Create any new sound effects that the new class may require (usually we can use sound effects that we already have, but sometimes new ones are needed). This is usually a matter of negotiating with Chris. The sounds go in "/u0/habitat/Beta/Sounds". There can be several files for any given sound effect. All sounds have a sound file. The source is in a file named "sound.sob" and the binary is in "sound.bin". For example, "loud_crunch.sob" and "loud_crunch.bin". Some complex sounds also have pulse width data associated with them, with source in "sound.spb" and binary in "sound.pwbin". For example, "loud_crunch.spb" and "loud_crunch.pwbin".

5a) Update the Makefile in "/u0/habitat/Beta/Sounds" to incorporate the new sounds.

Muddle:

6) Add a new class entry in the main Muddle file. This is currently the

file "/net/moth/u0/habitat/Beta/beta.mud", though this will change to something new once the beta version is released. The entry should look something like this:

```
        class class_foo 47 {
                action  foo_do                  /* do */
                action  generic_throw           /* reversed do */
                action  generic_goTo            /* go */
                action  generic_cease           /* stop */
                action  generic_goToAndGet      /* get */
                action  generic_goToAndDropAt   /* put */
                action  generic_broadcast       /* talk */
                action  generic_destroy         /* destroy */
                action  foo_ADUMBRATE           /* 8 */
                image   foo_image_0
                image   foo_image_1
                sound   foo_operating
                byte    16                      /* Object instance length */
                byte     8                      /* Initialization count */
                byte     0                      /* Capacity */
        }
```

These entries are in alphabetical order by class name.

The capacity byte value should be the number of objects that objects of this class can contain. If the new class is not a container, this number should be 0.

The initialization count value should be the number of bytes of state data that the host will send to the C64. This number should be 6 plus the number of class-specific state variables that objects of this class possess. In the above example, foo objects have 2 special state variables, so this value is 8.

The object instance length value should be the number of bytes that need to be allocated for instances of this class. This number should be 8 plus the initialization count value plus the capacity value. For objects that have no special state variables and are not containers (a common case), this value will be 14. In our example, since we have 2 special state variables, our value is instead 16.

7) Add new entries in the image, action and sound definition portions of the Muddle file to define the new image, action and sound names used in the class entry. For example, the above class entry would require the image entries:

```
        foo_image_0:            "Images/foo0.bin"
        foo_image_1:            "Images/foo1.bin"
```

the action entries:

```
        foo_do:                 "Actions/foo_do.bin"
        foo_ADUMBRATE:          "Actions/foo_ADUMBRATE.bin"
```

and the sound entry:

```
        foo_operating:          "Sounds/loud_crunch.bin"
```

Tools:

8) Write a Ghu define statement for the class. Assuming that objects of

class foo have two state variables, ’zapcount’ and ’adumbracity’, it would look like this:

```
        define 47 ’foo’
          zapcount:    bin15
          adumbracity: bin15
        enddefine
```

This should be placed in the definition file, "/u0/habitat/defines.ghu", and typed at Ghu on the host.

Host code:

9) Add an entry to the class numbers definition file. This file is in the work directory, "#d010>lucas>microcosm" which can be reached via the command "work". The file is named "defs_class.incl.pl1". You need to add a line of the form:

```
        %replace CLASS_FOO by 47;
```

at the appropriate point in the alphabetical list of class names.

10) Add an entry to the struct names definition file. This file is also in the work directory and is named "defs_struct.incl.pl1". You need to add a line of the form:

```
        %replace struct_foo by ’Structs>struct_foo.incl.pl1’;
```

to the alphabetical list of entries.

11) Create a struct definition for the class in the "Structs" subdirectory of the work directory. This definition should thus be in "Structs>struct_foo.incl.pl1". These struct definitions have a standard form, but we will not go into the details of that form here. Usually one simply copies a similar class’ struct definition and modifies it as needed.

12) Create the class definition itself in the "Classes" subdirectory of the work directory. This definition should thus be in "Classes>class_foo.pl1". These class definitions have a standard form, but we won’t go into the details of that form here. However, the class definition file should define the procedure "initialize_class_foo" as well as any special behavior routines that the class needs. This file can be compiled from the work directory with the Stratus command "pl1 Classes>class_foo" or (alternatively) "plc foo". The resulting object file should be moved into the "Linkable" subdirectory of the work directory.

13) Change the entry for the class number in the list of entries declared for the array of messages in "Actions>actions_help.pl1". For non-existant class number (which this was before you created it) the entry in the help array will be ’-’. If the class uses "generic_HELP" for its help text, the help text should be placed in this entry instead of the "-". If the class has its own help routine, the entry should be changed to ’i’.

Regionproc:

14) In the bind directory, "#d010>quantum>stratus>source>microcosm" or (alternatively) "bindir", reachable via the Stratus command "cdbind", add a line to the file "class.externals.incl.pl1" of the form:

```
        declare initialize_class_foo entry;
```

in the alphabetical list of such entries.

15) Also in the bind directory, add a line to the regionproc itself, in the file "regionproc.pl1". This line should have the form:

```
        call initialize_class_foo;
```

and should be placed in the list of other calls to initialize the other classes.

16) In the bind directory once again, add an entry to the bind control file, "regionproc.bind", of the form:

```
        class_foo,
```

in the alphabetical list of class object modules.

17) Recompile the regionproc from the bind directory with the Stratus command "pl1 regionproc -table". You should then use one of three commands to rebind: "rebind" rebinds the regionproc with the new class included; "redo" rebinds and then moves the resulting executable program into the run directory; "fixup" rebinds, moves the executable program into the run directory, and reboots Habitat. The latter is a little radical and should only be done when you know that you are not going to break somebody else’s running system by bouncing Habitat.