HomeRun

A User Guide

Document version: 1.1

Software version: 0.4.2

Author: Richard Rodgers

Date: August 10, 2009

Web: http://homerun.sourceforge.net

Table of Contents

Chapter 1 - Introduction

HomeRun is a software application to manage certain interactions with your physical environment. These interactions can take many forms, from automation or orchestration of home automation devices, to monitoring and communicating changes in the environment.

1.2 Who is it for?

HomeRun is for anyone interested in exploring how software can simplify their lives through automation. A common misconception is that you only should consider HomeRun if you possess home automation equipment purveyed by the likes of X10 or Insteon (or intend to buy some). It does support those devices, but even without any specialized hardware, you can reap many benefits. A few simple examples:

Receive a text message when it begins to rain at home, so you can call a neighbor to put in what you had drying on your porch.

Put the task list on the fridge onto a secure, distributed message system or a web page.

Automate delivery of birthday emails to friends and family when you think of it early in the year, or post reminders to be delivered in the future.

Doubtless much of the power of HomeRun is unleashed only with automation equipment, but you can add it at your leisure and as you better understand your needs or can afford it.

1.3 What does it cost to buy and operate?

HomeRun is free software, and there is no licensing cost to operate it. Use of the software is governed by the terms of the Apache 2 license. Services that integrate with the platform and offer additional benefits may be offered in the future could include fees or subscriptions, but these will always be elective and not be required to run HomeRun productively.

1.4 How do I get it?

You can download HomeRun from its website at:

http://homerun.sourceforge.net/

Downloads are also available at the HomeRun development site:

http://code.google.com/p/homerun/

1.5 Scope of this guide: is this all the documentation?

Not at all, it's really just the tip of the iceberg. HomeRun is designed as a modular system, and each module has it's own documentation (bundled with it) describing installation, configuration, and use. You may peruse this documentation online at the HomeRun web site, or from within HomeRun in the Setup program. Moreover, all the user interface programs have online help screens with task-oriented instructions. This guide is intended to get you started with installation, and orient you to the general mode of operation of the software.

Chapter 2 - Requirements

2.1 General Requirements – What do I need to run HomeRun?

HomeRun as a platform has very minimal requirements to operate, but individual packages have their own specific requirements, so check that your system fulfills them before installing them. The platform does have a target environment that allows you to use HomeRun to maximum advantage, which consists of one computer running the server software connected to a network, either a local-area network, or the internet (via a modem or router) or best of all both. Dial-up internet is not currently supported. Any computer that will run HomeRun software (client or server) must have Java (the JRE aka 'runtime') of at least version 5.0 installed and visible to the application. Java is widely supported on modern operating systems, from Windows (Windows 98 2nd Ed through Vista), to Mac OS (distributed by Apple), to Linux (most recent distribution), to Sun Solaris 8 through 10.

2.1.1 Hardware

The server platform should be very modest in terms of compute resources: almost any computer made within the last five years will have ample CPU, memory, disk, etc Again, the actual load will depend on which packages are installed, so read the package requirements.

The same caution applies to computer hardware interfaces – all that the platform requires is an IP network connection, but protocol-specific hardware packages (like, e.g. those supporting X10 automation equipment) may impose additional requirements. The most common would be serial port devices, although USB devices are able to emulate serial interfaces for most operating systems.

2.1.2 Software

The HomeRun platform bundles all the software it depends on with it, so you will not need to obtain or install any other software (except Java as noted above). The same is true for all known packages today, but in the future there may be packages that require external software components to be installed. Similarly, almost all packages (one exception is serial port support) work on all operating systems that the platform does, but this does not rule out OS-specific platforms where they make sense.

Chapter 3 - Installation

3.1 General Considerations

HomeRun has a *client-server* architecture, which in simple terms means it is composed of two independent but cooperating pieces of software. It follows that installing HomeRun is likewise twofold, i.e. a server installation followed by a client installation. In addition to its client-server structure, HomeRun also has a highly modular organization, with almost all the functionality residing in separate 'packages' that are added to the server post-installation. So there really are three phases: server installation, client installation, and then system construction using the installed software. Fortunately, each of these phases is very simple, as the following sections demonstrate.

3.2 Server Identification

The *server* software must run on a single computer in a home network, which should remain on (the computer and the server software), and be connected to the local network. You may in fact use HomeRun with an 'intermittently on' server, but many of the most useful and powerful features would be of very limited value, since the system reacts to the changing environment. Many homes now have a computer attached to the internet router or modem, which generally remains on as a gateway for other computers. This is a good choice for the HomeRun server machine. Another consideration is whether the computer has the necessary interfaces for the HomeRun server: in particular, many automation systems use a controller attached to a serial port, and if the computer lacks one (as MacIntosh computers do, or many recent laptops), it may not make the best server machine. However, adapters can often be installed to emulate the desired interfaces. A final factor to consider is energy consumption: since the server computer will remain on 24 hours a day, if there is a more energy-stingy computer, you might think about using it. The HomeRun server computer doesn't need to be very powerful.

3.3 Server Installation

When you have selected the computer on which you wish to run the HomeRun server, visit the HomeRun website at

http://homerun.sourceforge.net/

and follow the 'download' link on the top of the page. Look for the table labeled as 'current release'. The table contains links to archive ('zip') files for the server and client. Download the server archive file, and unzip the archive to a directory you want the server installation to live in. Now open a terminal window and navigate to the `fwork` directory below the installation directory. Start the server by running the `hrserver.bat` batch file.
If you are using a non-Windows OS, the file used to start the server will instead be a 'shell script' file called hrserver.sh When unzipped on non-Windows systems, this script file may lose permissions to execute it.
In this case, it will be necessary to reset the permissions:

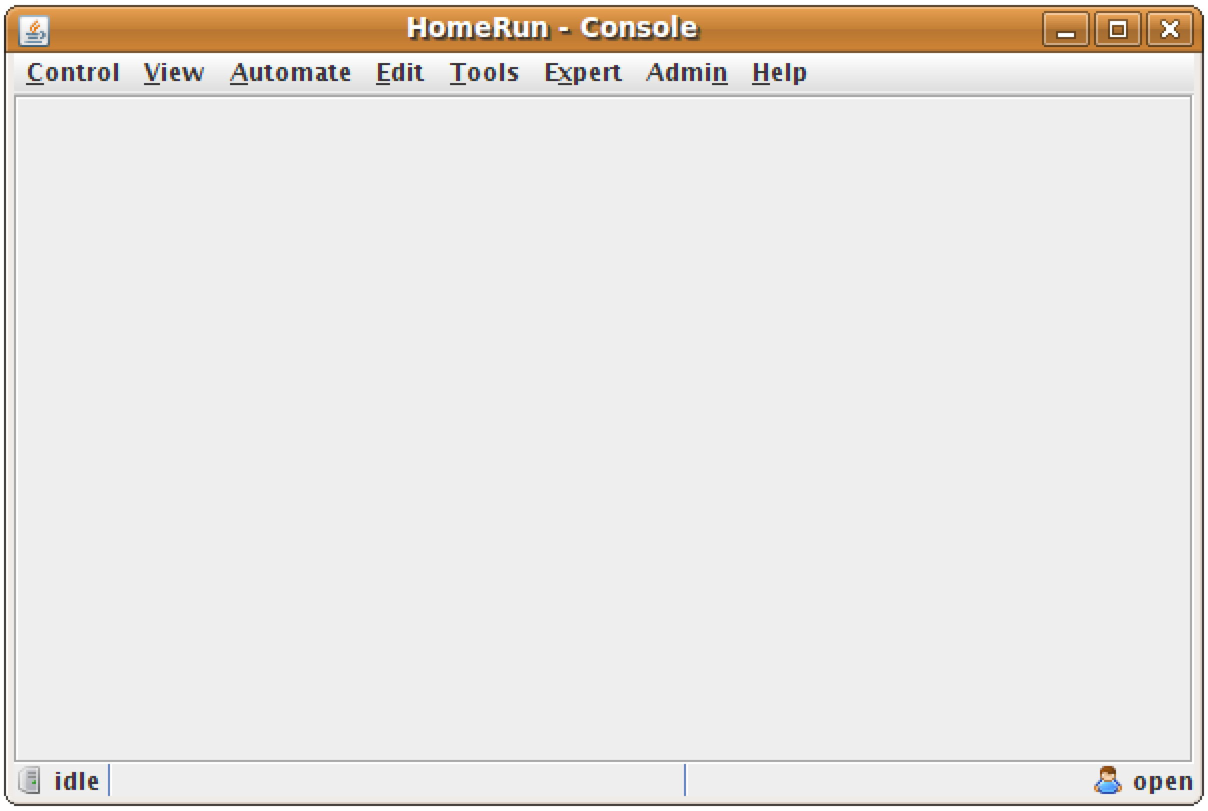
```
$ chmod u+x hrserver.sh
```

would be a typical invocation to accomplish this. If all goes well, you will see output to the terminal window from the server including:

```
HomeRun bootstrap on port: 8070
```

3.4 Client Installation

C*lient* software, by contrast to server, can run anywhere on the network (including the server computer), can run on multiple machines at the same time, and client computers do **not** have to stay running when they aren't needed.
To install a HomeRun client, simply select a computer you want to be a client machine, and download the archive ('zip') file labeled 'Client' from the current release table. You may install clients on as many computers as you wish (though of course only one per computer), but only one is needed to configure the server. Other clients may be installed as needed.
Unzip the archive to a directory you want the client installation to live in, and start the client by running the batch file hrclient.bat, found at the top of the installation directory. Make similar adjustments (hrclient.sh) as above if on a non-Windows OS. If you succeed, a graphical UI known as the 'Console' will appear, looking almost like this:



This admittedly plain UI is a 'cross-platform' Java look, which should appear nearly the same whether your client machine is Windows or Mac OS or Linux. If you want something closer to how your operating system normally appears, then launch the Console as follows:

```
hrclient.bat native
```

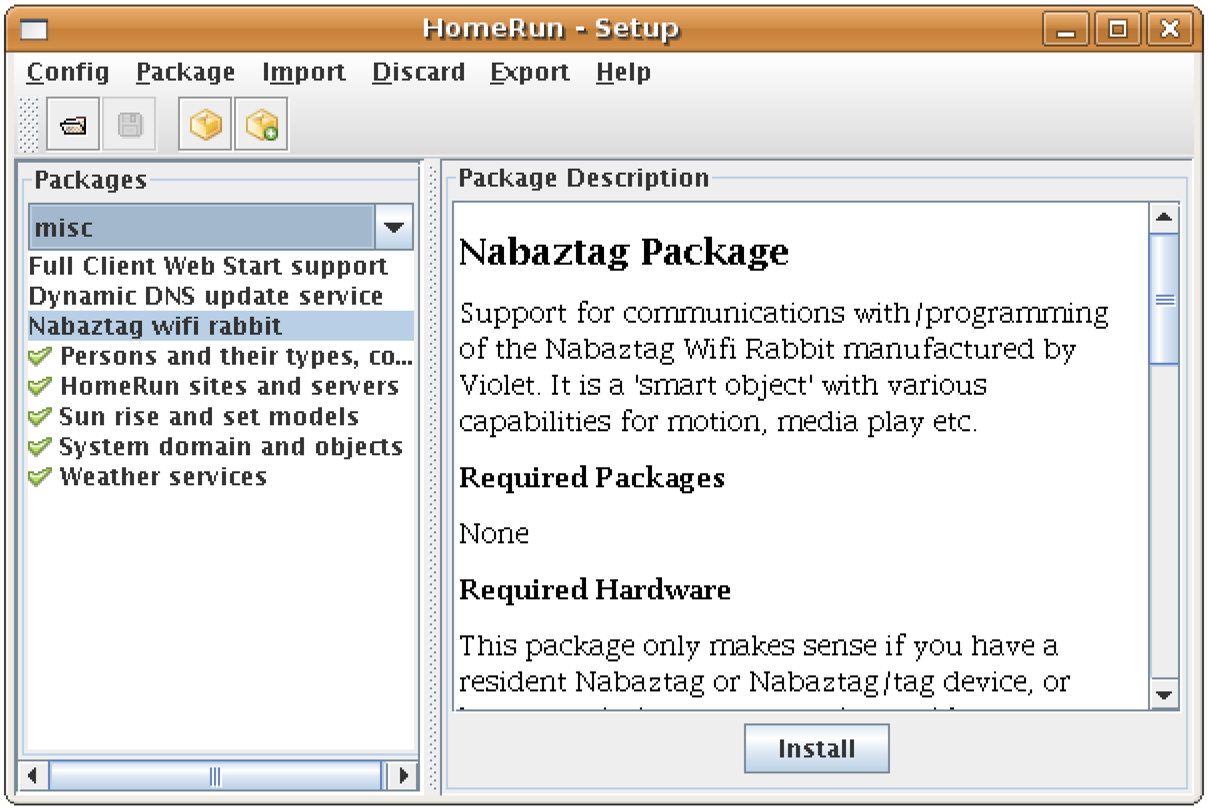
Consult the README.txt file in the installation directory for other looks and options, as well as the latest instructions.

We said almost, because the lower left text may actually look a little different the first time you launch the Console. It will read 'find' instead of 'idle' to indicate that it must find a server to communicate with. If you have installed the client and server on the same computer, it will detect this condition and not display the 'find', and lock onto the server. Otherwise, go to the server discovery dialog (Admin->Server->Find). Here you can either enter the IP address of the computer running the server if you know it (be sure to press the 'Check' button after to ensure it can find a server there, then the 'Accept' button to save the selection), or press the 'Probe' button to initiate a process of server discovery (again press 'Accept' when a server has been found). Your client will remember the location of the server, so you should not have to repeat this unless you re-install the server somewhere else.

3.5 System Construction – Package Installation

You now have a complete, but largely empty, HomeRun installation, both server and client. To fill it, you will need to identify and install what HomeRun calls 'packages', which are bundles of software, configuration, and supporting files that allow you to do specific things with HomeRun. These packages live in a repository on an internet server, and you may discover and browse through the available packages in two ways. First you may visit the HomeRun website and look for the link at the top labeled 'packages'. If you identify a package of interest, just remember its name and use the program available from the Console called 'Setup' to install it: it will automatically download the package from the repository and install it to your local server. The website offers a 'faceted browsing' navigational system that makes finding the package you want easy.

You can also explore the packages within the Setup program, but with a more rudimentary interface. However you have discovered a package of interest, you will install it using the Setup program -– which is reachable from the Console via the Admin->Setup menu selection. In Setup, chose Package->List:



Full Client Web Start support
Dynamic DNS update service
Nabaztag wifi rabbit
✓ Persons and their types, co...
✓ HomeRun sites and servers
✓ Sun rise and set models
✓ System domain and objects
✓ Weather services

Nabaztag Package

Support for communications with/programming of the Nabaztag Wifi Rabbit manufactured by Violet. It is a 'smart object' with various capabilities for motion, media play etc.

Required Packages

None

Required Hardware

This package only makes sense if you have a resident Nabaztag or Nabaztag/tag device, or

Simply select a package category – 'misc' in this case – and click on a package name to bring up the description. Packages with the green check-mark are installed, and pressing the 'Install' button installs the selected package.

When installing certain packages, you are given an opportunity to 'localize' some of the text values that appear in the UI or elsewhere. This screen replaces the description, and consists of a table with a localizer field name in the left column, the default value or values of the field, and a blank column where you may enter a localized value. To accept the defaults without modification, press the 'Skip' button; otherwise make any changes and press the 'Done' button.

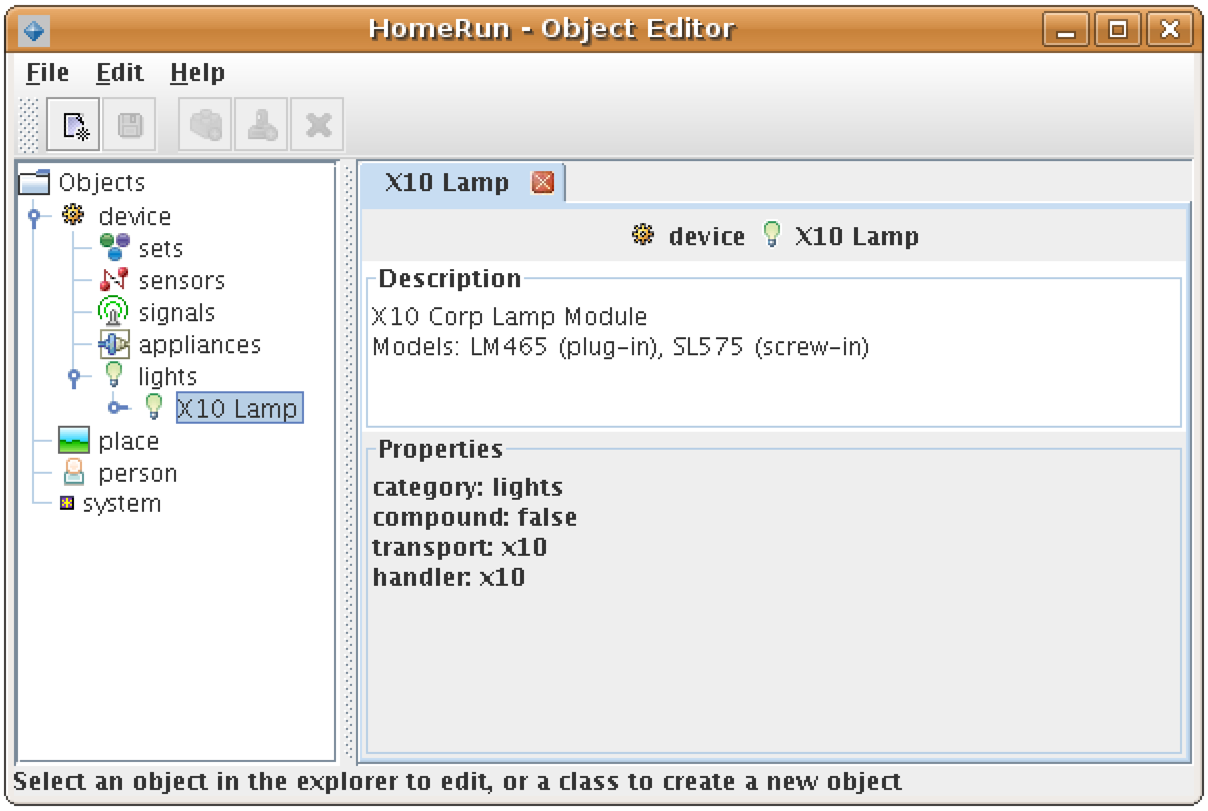
The packages you select will depend entirely on your needs and the sorts of automation hardware you have (if any). Just remember that you can install packages at any time, not just during initial 'population' of your HomeRun system. However, it is strongly recommended that you install at least the 'system' package, which contains indispensable tools for operating your HomeRun system. Also worth knowing is that most unwanted objects installed by packages can be manually deleted without compromising your installation.

Chapter 4 – Objects and Components

4.1 Objects, Types, Categories, and Domains

Almost all operations in HomeRun consist of interacting with what are known as objects, so understanding what they are and how they work is the best way to get to know the platform. Many packages supply objects 'ready made' for you to use, but the more common pattern is for you to create the objects you need.

An object is simply a named resource in your HomeRun universe. Each object is unique, but all HomeRun objects are instances of a class, known as the object's type. We are all unique individuals, but members of the human species. Types are important when we want to create new objects (i.e. individuals or instances) in the system: HomeRun has a UI – User Interface – which means a program that you interact with -- called the Object Editor that displays all the types known to the system, and then gives you an opportunity to create a new object of the type. Here's a screenshot of the Object Editor (to launch: Console Edit->Object):



X10 Corp Lamp Module
Models: LM465 (plug-in), SL575 (screw-in)

category: lights
compound: false
transport: x10
handler: x10

Select an object in the explorer to edit, or a class to create a new object

The tab labeled 'X10 Lamp' displays information about the type, including a description, and various properties. These properties are inherited by any objects you create.

Types are important for object creation, but in most other contexts we care more about what **category** an object is in. Notice above that X10 Lamps (the type) belong to the category (have the property 'category') 'lights'. Category is the way all objects are grouped in the object explorer of all other UI applications. This allows us to treat lights as a logical group, whether they are X10 Lamps, Insteon Lamps, Z-Wave Lamps, etc.

Finally, notice in the screenshot above that the categories and types themselves appear under a top-level description in the tree:
e.g. device, place, etc. These broader groupings are called **domains** in HomeRun. Domains are the highest-level families of types, categories, and objects. All object names, in fact, are relative to the domain they fall in: you may name a person “John” and a lamp (device) “John” and the system will know the difference. There are only a few domains, but they help organize all the objects in your system. In many contexts where a bare name may be ambiguous, HomeRun will ask for, or expect, a domain name as a 'qualifier'.

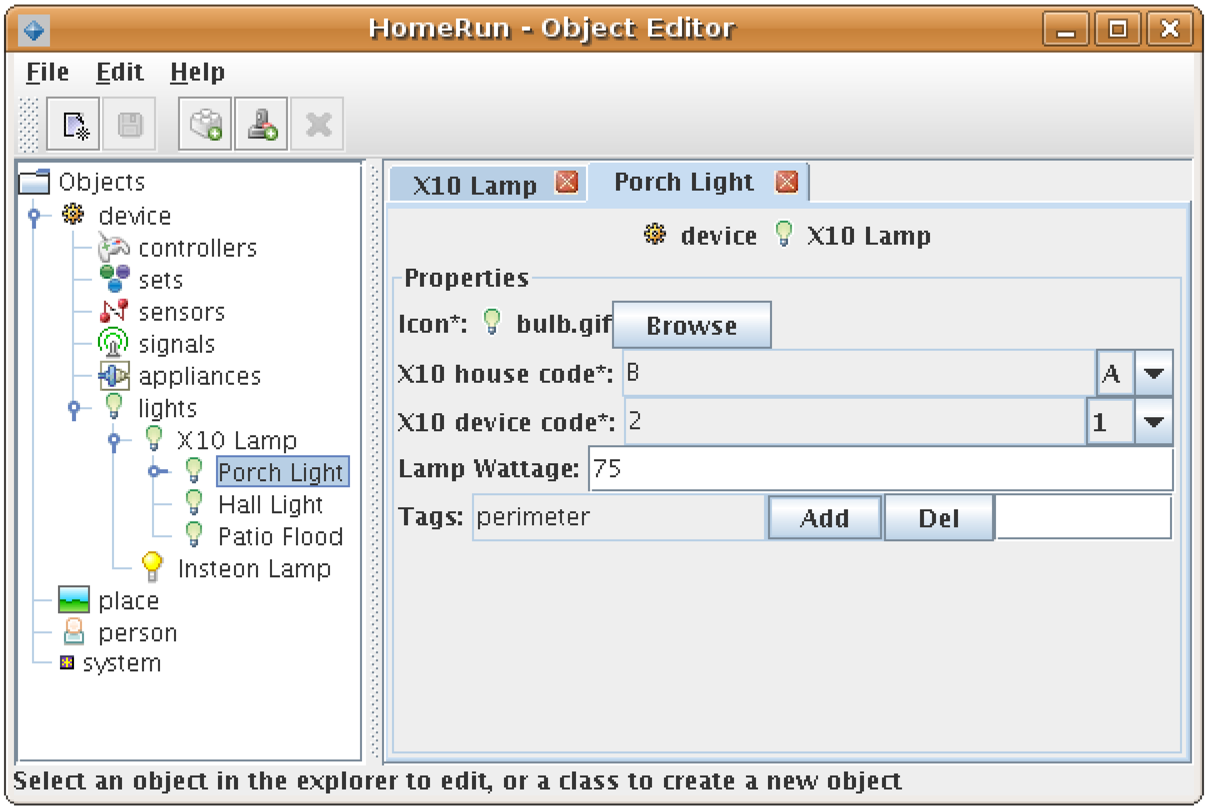
The following sections explore how objects work, and what their components are.

4.2 Anatomy of an Object

Objects can have four kinds of components, as follows:

4.2.1 Properties

These are basically descriptions (names with values) of attributes that may either be generic (true of the type), or specific (true of the individual). Being bipedal is a generic human property, but hair color is a specific one. We saw in the previous Object Editor screenshot the generic properties of an X10 Lamp in a tab associated with the object type. When you have created an object, the editor asks for (or displays) the specific properties (here we use as an illustration an X-10 Lamp in the device domain):



Icon*: bulb.gif
X10 house code*: B A
X10 device code*: 2 1
Lamp Wattage: 75
Tags: perimeter

Select an object in the explorer to edit, or a class to create a new object

Since X10 devices are addressed by house and device code, these are specific properties. Some properties are required (shown with an asterisk next to their labels in the Object Editor, and some are optional (e.g. Lamp wattage). Some also have default values – for instance, the icon used to represent this light gets its default from it's parent type.

Properties are used for attributes of objects that don't change, i.e. are **static**. OK, so maybe hair color wasn't a great example. For attributes that can change, e.g. whether the light is on or off, HomeRun uses a different component – called a **model** – to capture the value of these variable attributes. Models will be discussed in their own chapter below.

4.2.2 Controls

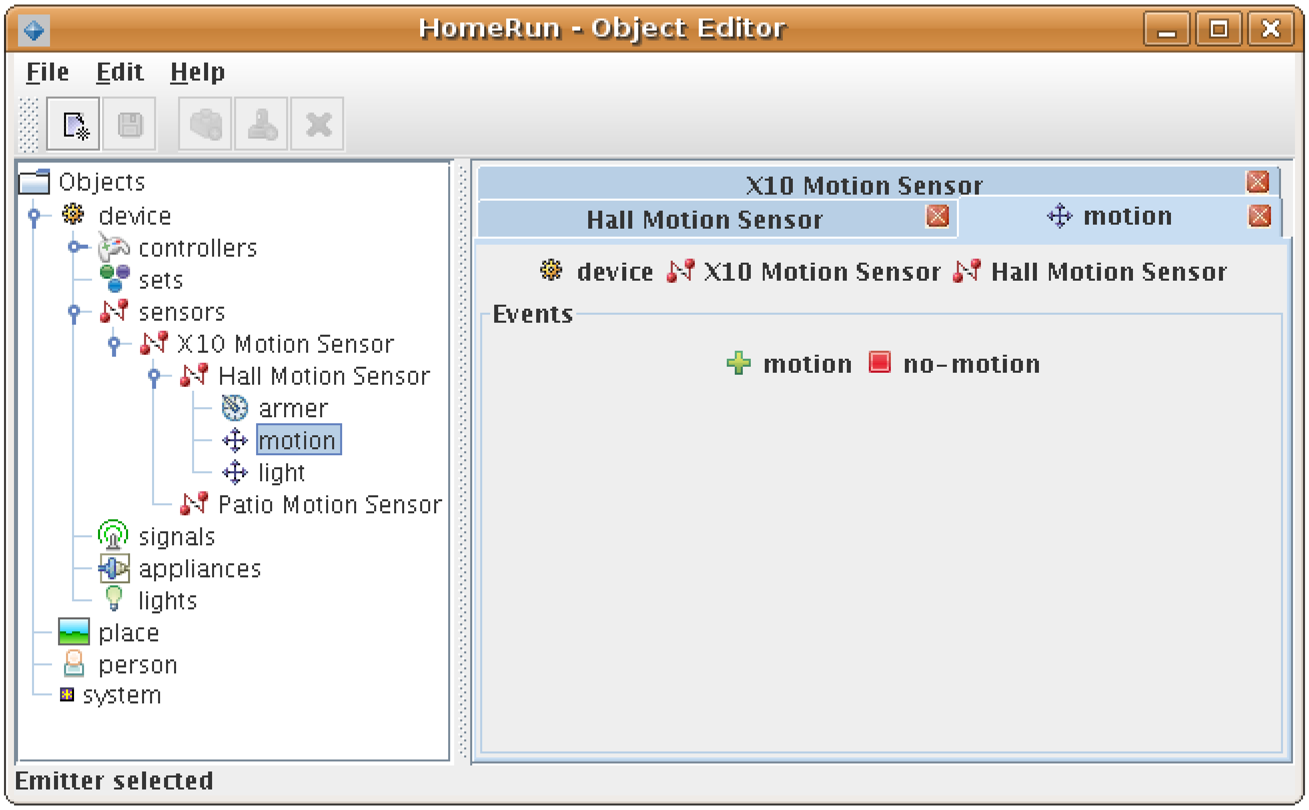
Our chief interest in a large set of objects like lights is that we can control them. A light, for example, has a switch on it that we use to turn it on and off. Depending on its type, an object may have zero, one or more controls. HomeRun has a standard representation and terminology for controls as follows: a control is a connected set of operations we can perform on an object. Connected really means logically related, in the sense that each operation pertains to the same thing. A lamp may have 2 buttons (on and off), but they both affect the power to the light. Each one of these operations is called a **point –** so our light switch has 2 control points. Certain more complex kinds of control points may also have additional values associated with them (like, for example, a reading on a dial), and these are referred to as **inputs.**

For many types of object (again, think of a light), having a particular control is definitive of them. When this is true, the control will automatically be added to the object when you create it. But for others, controls are optional in the sense that either not all objects of the type have them (e.g. a dimmer for a light), or we might not care to use them in HomeRun. For these optional cases, the Object Editor can be used to add a control to an object. Definitive controls will appear already 'attached' to the object when selected in the Object Editor.

4.2.3 Emitters

In contrast to objects with controls, many other objects act as 'observers' or 'reporters' for us. An example might be the smoke detectors we keep in our houses. They don't really have any controls to speak of (maybe a battery test button), but instead have the ability to sound an alarm under certain sensed conditions. HomeRun calls these independently produced activities **events** and calls the facility to produce a related set of events an **emitter**. Again, depending on the type of object, it may possess zero, one or more emitters. But unlike controls, emitters can never be optional: if they belong to an object, they will be added to it when you create one.

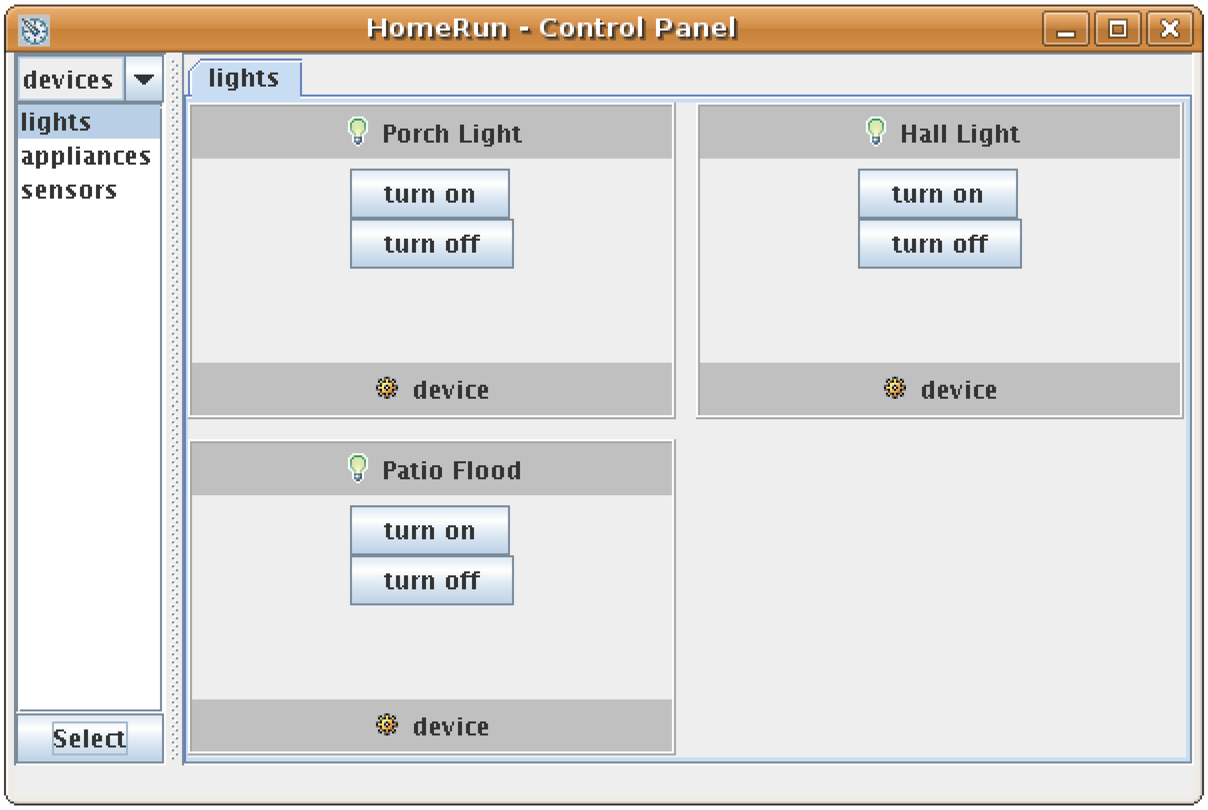
Here's a screen shot of an object with controls and emitters:



The Hall Motion Sensor has a control called 'armer' (since it is used to arm or disarm the sensor), and 2 emitters, one for motion/no-motion events (shown), and one for light/dark events.

4.2.4 Using Controls and Emitters

So far we have only discussed what these components are and how we encounter them in editors. The more important question, of course, is how we use them when we want to interact with objects. We will defer discussion of events until the next chapter, but for controls, there are several UIs we can use to directly manipulate object controls. One is called the Control Panel (Control->Panel from the Console), and it looks like this:



The left drop-down contains domain names, and below them names of panels (for how to create one, see the Advanced Topics chapter), and you can simply press the buttons representing the control point to directly affect the object.

Chapter 5 – Actions and Bindings

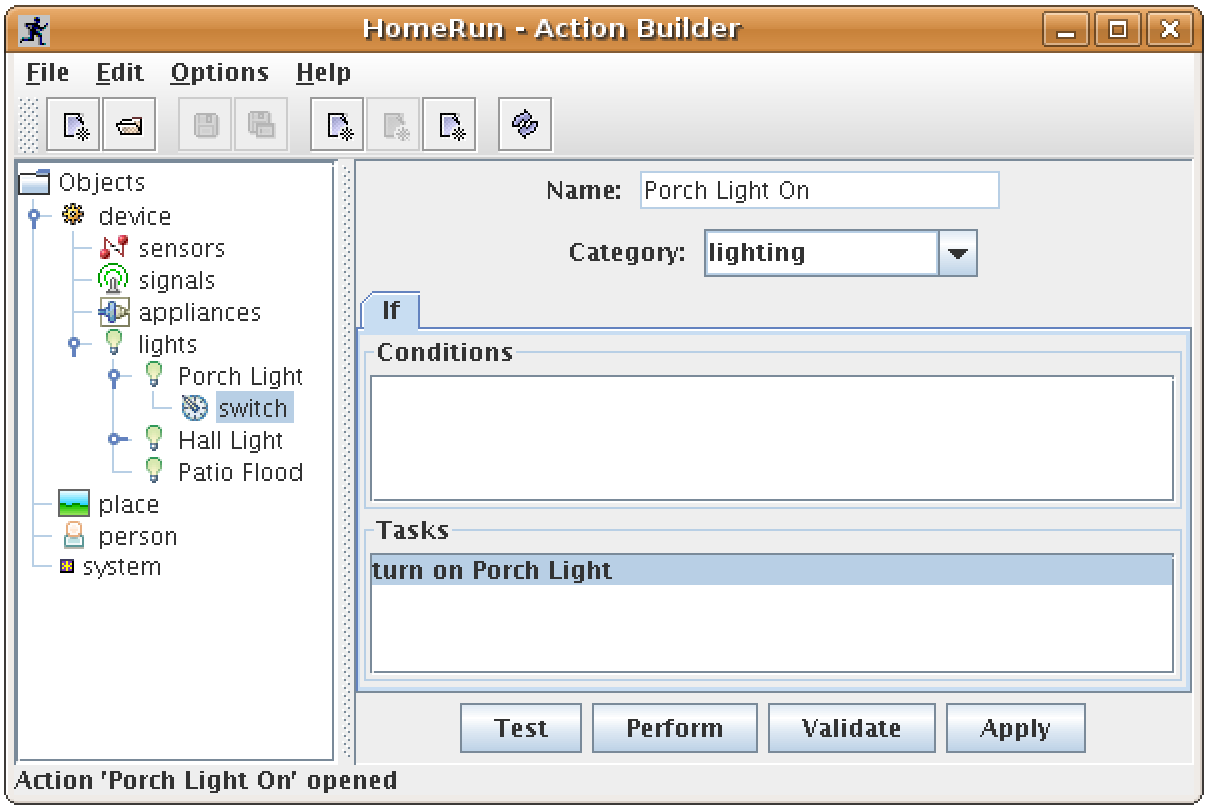
In the last chapter, we explored the world of HomeRun objects, and some of their constituent parts. We also briefly looked at how objects are exposed in the UI so their controls can be manipulated. But what if we want a control to operate when we aren't there, as part of an automated, unattended process?

The answer in HomeRun is by using 'actions', which are named collections of object control invocations (and much more). You can define actions to suit any purpose, and actions are embedded in many other powerful constructs. Let's begin our tour of actions by making a very simple one.

5.1 Simple Actions

Suppose we have a 'Porch Light' object, and we know it has a 'switch' control. This control in turn has two 'points' – on and off. The most 'atomic', smallest operation, then, that can be expressed is the three-part specification: (Porch Light, switch, on). HomeRun calls these specifications 'tasks' and they constitute the smallest unit of work the system can perform. Smallest here means that it cannot be decomposed into sub-parts. An action in its simplest form is a named list of tasks (i.e. at least one). Let's give our specification (Porch Light, switch, on) the imaginative name “Porch Light On”. We can use this shorthand wherever we have to select an object, control, and point. For example. we could put actions in the Control Panel in lieu of the object representations. A single task action doesn't really buy us much, but if we string several tasks together (after the Porch Light, maybe the Hall Light goes on in anticipation of our entry) in one action, it becomes more powerful. As we will see, this is only the beginning of what can be expressed in actions.

HomeRun provides an intuitive UI for working with actions called the 'Action Builder' (from the Console its Edit->Action). Here's a screen shot of our simple action:



Name: Porch Light On

Category: lighting

Conditions

Tasks

turn on Porch Light

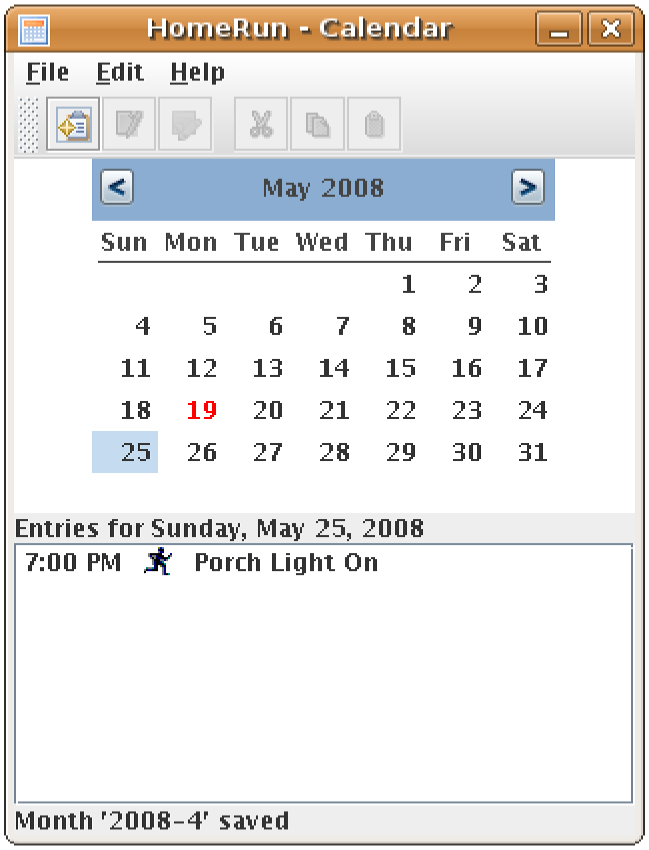
Action 'Porch Light On' opened

Adding tasks is simply a matter of navigating in the object explorer panel on the left to the object of interest, and then clicking on the control you wish to add. A dialog appears to capture the details.

Once actions are defined, they can be invoked manually in the Control Panel, but typically they are invoked automatically in various ways. These ways are often referred to as 'triggers' in that their occurrence fires the action. HomeRun prefers the term 'binding' meaning the action is tied to a circumstance whose occurrence starts it. Of course, the same action can be bound to multiple and different circumstances. Let's briefly examine the primary binding types in HomeRun, and how you manage them.

5.1.1 Calendar Bindings

First, we can bind an action to a specific, non-repeating date and time: “I want the action to take place on June 25th, at 7:30 pm”. These are known as **calendar** bindings since they would naturally be written on your calendar. HomeRun provides a simple program for creating calendar bindings (from the Console: Automate->Calendar):



Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Entries for Sunday, May 25, 2008

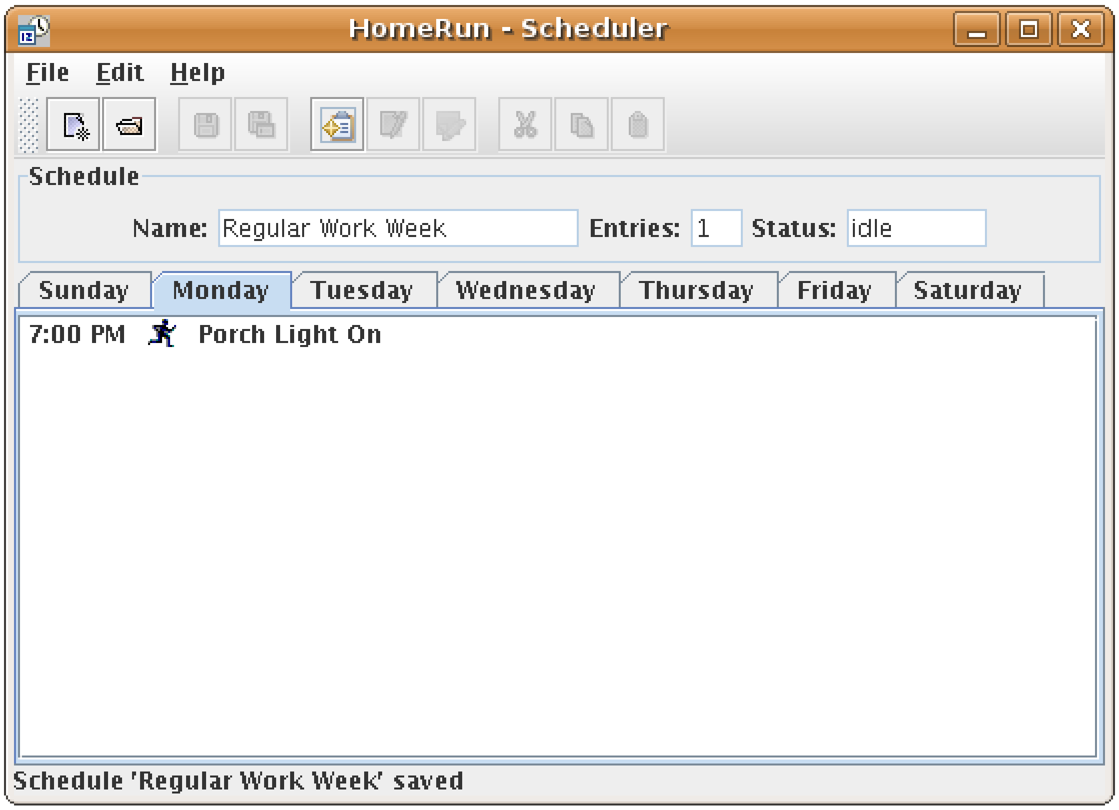
```
7:00 PM    Porch Light On
```

Month '2008-4' saved

Click the date to display any current bindings, and press the 'New Entry' button to add one (they are automatically saved as you add them). You can only add entries in the future, and the action is performed only once. However, the entry remains visible after the action has been performed, so you may review past activity.

5.1.2 Schedule Bindings

When an action is bound to a particular time, but one that **repeats** over a standard interval, it is known as a **schedule** binding. For most of us, the interval that rules our lives is the week, so HomeRun provides a tool to manage weekly repeated actions, called the Scheduler.
Here's a screen shot (from Console: Automate->Schedule):



Name: Regular Work Week Entries: 1 Status: idle

7:00 PM Porch Light On

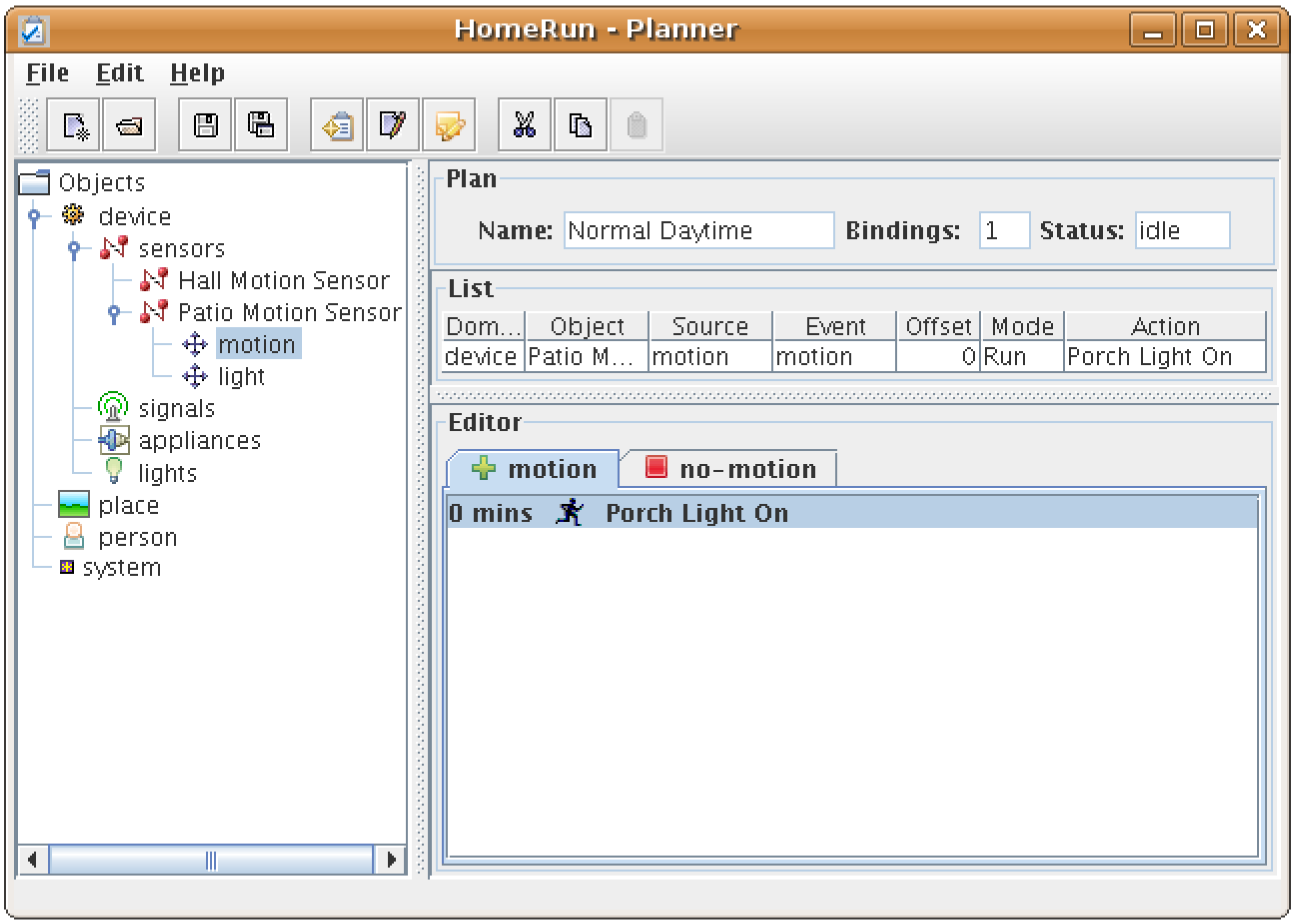
Schedule 'Regular Work Week' saved

Here we have created a schedule named Regular Work Week, and have just added our Porch Light action to Monday's bindings. You can of course have as many bindings as you wish (even at the same time). Unlike the Calendar, we can create many different schedules (that's why we assign a name to each). This is a very powerful feature, since we can design a schedule to suit any occasion (e.g. one for when we are on vacation, etc), and only have to activate that schedule when needed, rather than reprogram the whole system whenever our routine changes. Only one schedule is active at a time, and we see that Regular Work Week now has a status of **idle** (not active).

5.1.3 Event Bindings

Another binding we want to look at is an **event** binding. An event binding causes an action to be performed not at any specific time at all, but rather only when an emitter event occurs. Since we typically don't know when or even **whether** any particular event will happen, event bindings have a conditional nature compared to calendar and schedule bindings, which are guaranteed to occur at the appointed time.

As you might suppose, HomeRun has a UI to help you manage these bindings, which it calls the **Planner** application. The name derives from 'plans', which are groups of related event bindings (in just the same way that groups of schedule bindings are known as schedules). Planner (from Console, Automate->Plan) look like this:



Name: Normal Daytime Bindings: 1 Status: idle

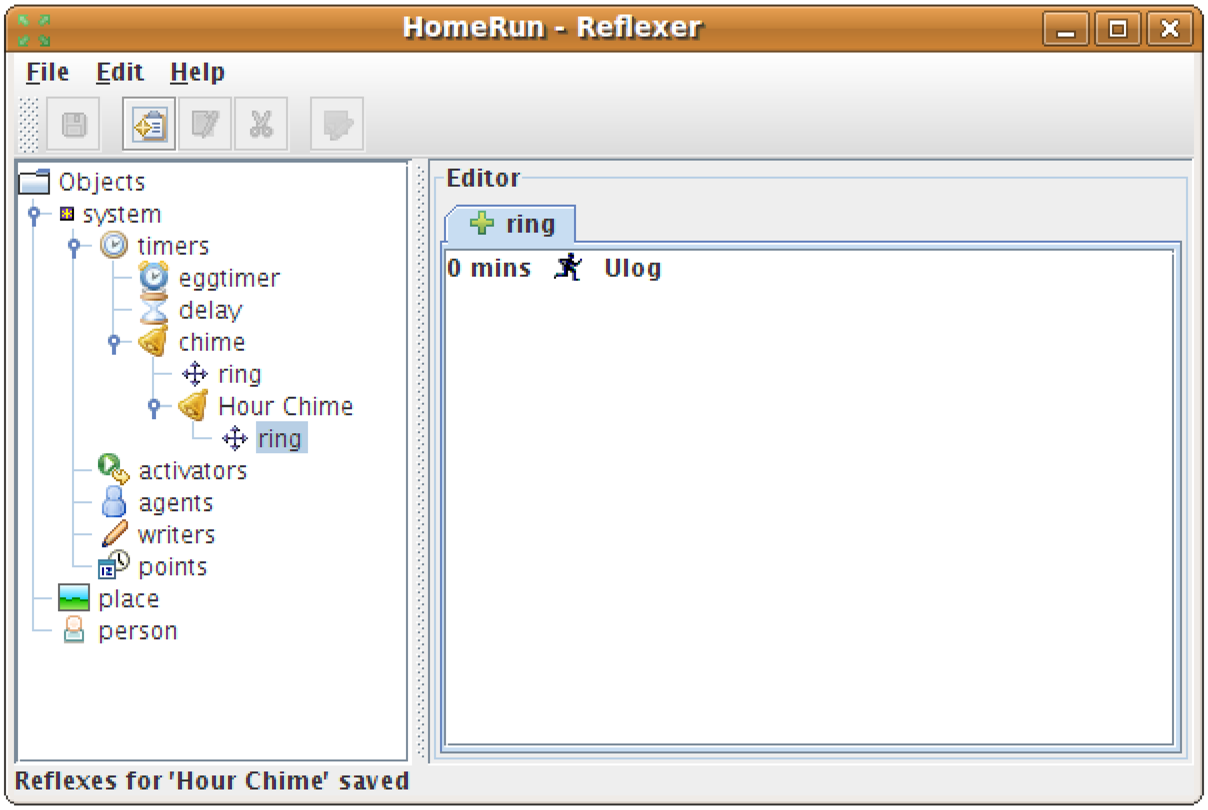
Dom...	Object	Source	Event	Offset	Mode	Action
device	Patio M...	motion	motion	0	Run	Porch Light On

0 mins Porch Light On

You can see the similarity to the Scheduler UI, which is completely intentional. However here, the tab is labeled with the name and icon of the emitter event that has been selected in the object explorer on the left panel. The **0 mins** notation before the name of the action refers to the ability to delay execution of the action for a configurable time. Also notice that like schedules, plans must be activated before their bindings will work: but unlike schedules, you may have multiple plans active at once. You may even interlink schedules and plans in useful ways. For instance, you may have a plan based on motion detection for when you are asleep, but another when you are awake. You can create entries in your schedule to start that plan at night, then stop it and start the other in the morning.

5.1.4 Reflex Bindings

Finally, you can bind an action directly to an object (more precisely, to any event that it emits), not as part of a plan. These bindings are known as an object's **reflexes** since you cannot suspend their occurrence the way you can with a plan (just stop the plan). Here is the managing program, known as the 'Reflexer' (from Console, Automate->Reflex):



```
0 mins   Ulog
```

Reflexes for 'Hour Chime' saved

Here we have defined a chime object to 'ring' every hour – i.e. emit a 'ring' event. We have added a 'reflex' action to log the event every time. A powerful feature of reflexes is that we can attach them not only to objects (as shown here), but to object types: this way, any object created will inherit that reflex behavior.

5.2 Binding Variables (Action Templates)

Recall that an action is a list of tasks, which are specifications of the form: (Object, Control, Point). Some controls, however, as we have noted, require further information to perform, which we called inputs. An example would be a control used to deliver an email, where the input was the message to send. Our task now looks like: (Object, Control, Point, Input). We could of course just create a new action for each message we want to deliver, but they would be single-use, 'throwaway' actions. To provide a more sensible solution, HomeRun supports the idea of binding variables, which are placeholders in an action input that can assume different values in different contexts. In this sense, an action defined with one or more binding variables really becomes an action template, that we can use again and again.

To employ a binding variable, simply substitute a variable name for a real value when defining the action (in the Action Builder editor application) For example, instead of entering a particular message in the email message input field, you would enter the word '$themsg'. The leading '$' tells HomeRun this is a binding variable, not a real message. (You can of course use any variable name you want: $mymessage, etc). Then, whenever the action is 'bound', i.e. added to the Calendar, Schedule, Plan, etc HomeRun will prompt you for the value to use for each binding variable it finds. It's that easy to turn any action with inputs into a reusable template.

Chapter 6 – Models and Scenes

We learned in chapter 4 that objects with variable attributes used HomeRun components called models. Objects can have any number of models associated with them, and we now return for a more in-depth discussion of models and what they can be used for. Recall that a model represents any changeable (mutable) attribute of an object. Most automation software treats this class of data in a fairly limited fashion, e.g. like having a device 'status' with some fixed values (on or off). The problem with this is that it fails to capture the more complex ways we want to represent internal state, and the fact that we often want to track several variables, not just a single 'status'. This deficiency is sometimes addressed by adding support for scripting languages, but this significantly raises the skill-level and complexity required to operate the system.

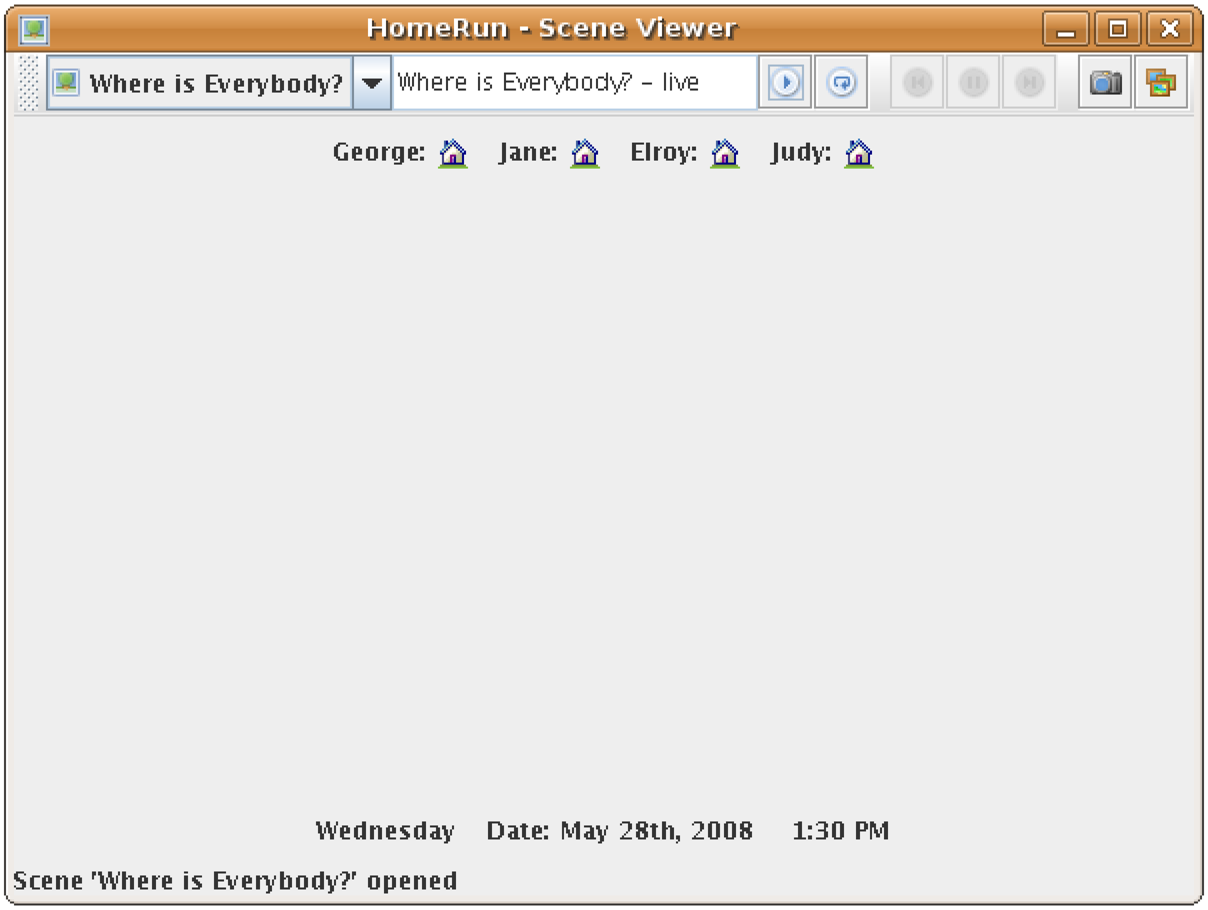
HomeRun gives you a simple, flexible and very powerful alternative in models. You may define your own models, and associate as many of them with an object as you wish. Let's take a simple example, our Porch Light. What attributes or properties might we want? We certainly would expect a model for whether the light is on or off. But maybe we also want to know when the last time it was turned on, or even used. Or maybe we just want a count of how many times the switch was operated, or who the last person to turn it on was. Here then are five simple models we might want, and we can create any or all of them and attach them to any object we wish, all without scripting.

6.1 Model Use

Before delving more deeply into models, we must address the obvious first question: how you use them in HomeRun? There are two main ways: you can inspect them in a UI, or you can use their values to affect the behavior of actions (without needing to examine them directly).

6.1.1 Scenes

Scenes are UI constructions in HomeRun that visualize models. You can select any number of models, and place them on a 'canvas' with a few standard layouts. Then using a separate program, you select the scene and it displays the formatted visualization of each model using its current value. Here's a screen shot of the Scene Viewer application (View->Scenes from the Console):



George: 🏠 Jane: 🏠 Elroy: 🏠 Judy: 🏠

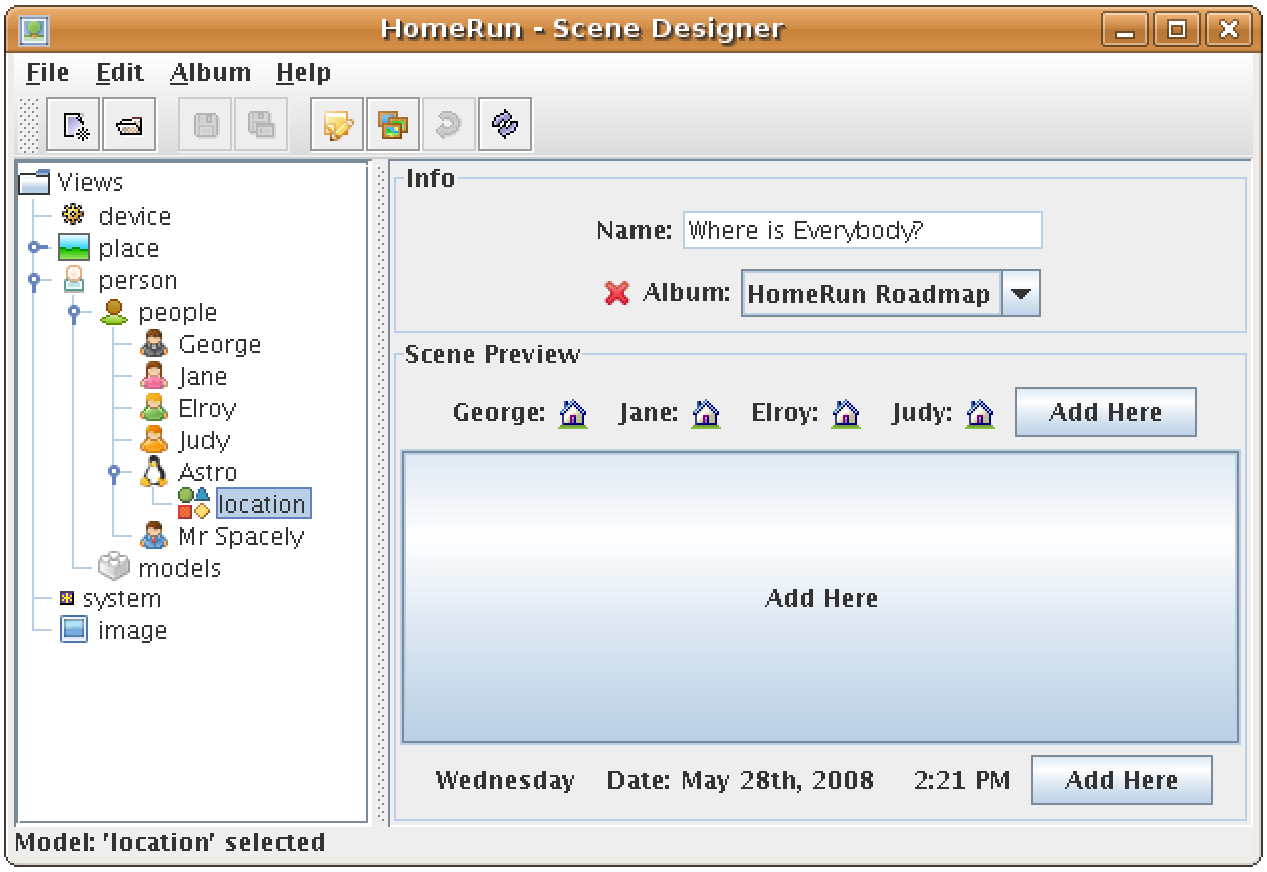
Wednesday Date: May 28th, 2008 1:30 PM

Scene 'Where is Everybody?' opened

This very simple scene named “Where is Everybody?” consists of seven distinct models: on the bottom row are a model for the day of the week, the date and the current time. On the top row, there is a 'location' model for each of our four imaginary residents (from the Jetsons). The visualization here is an icon representing their current location – which in all cases is 'home'. In plain English, all the Jetsons are home.

A few other UI controls to notice in Scene Viewer: the circular arrow, which means 'refresh' allows you to update the scene with the latest model values. Scenes represent a model evaluation at a particular time (when you hit the 'play' button), but you can update them with refresh. The button with the camera icon (“capture”) lets you record a scene's values for later inspection - think of it as a 'data snapshot' of the scene. These snapshots can be viewed using the button with multiple images (“snapshots”) - which will open a dialog for you to choose among the available snapshots of the selected scene. Note also that you can define an action to take the snapshot, so you could add that to a schedule (e.g who is home at 5:00 pm every day?).

Here's a peek at the UI for creating scenes (Edit->Scene from the Console):



Name: Where is Everybody?

Album: HomeRun Roadmap

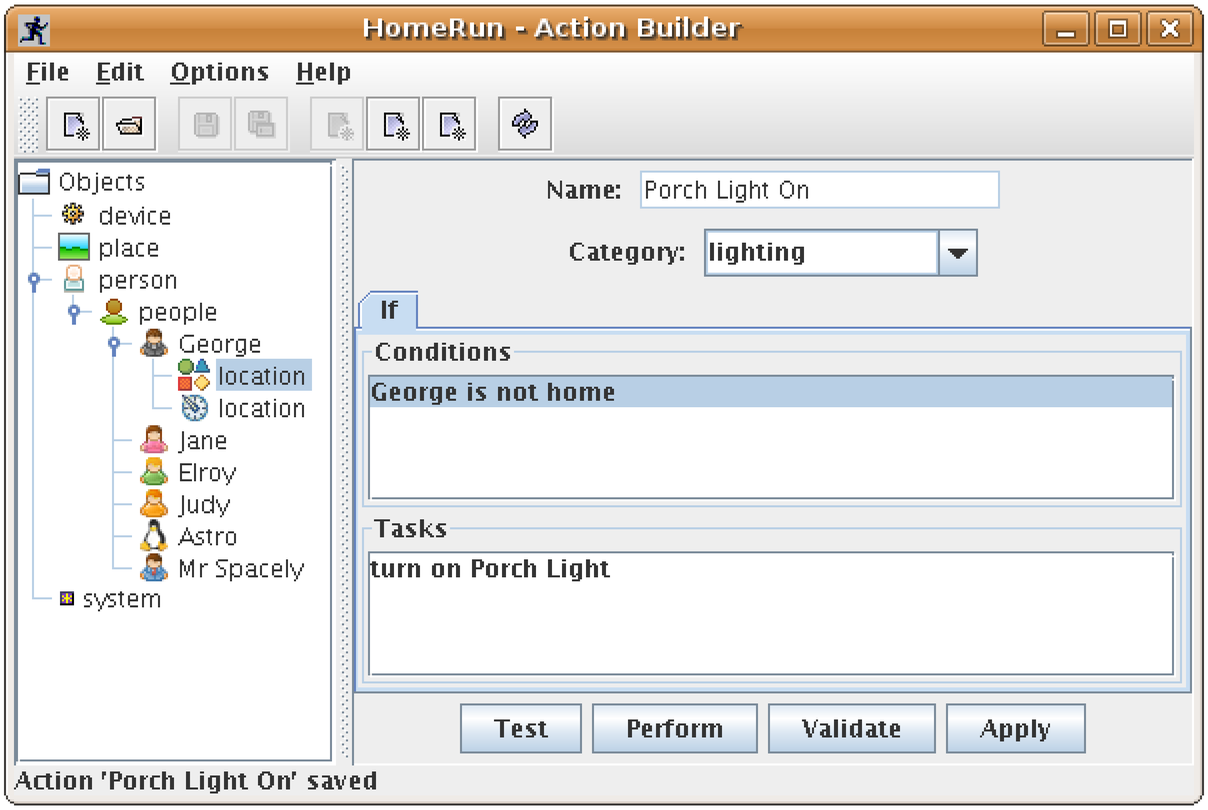
Wednesday Date: May 28th, 2008 2:21 PM

Model: 'location' selected

Scene Designer has the familiar layout for HomeRun editors, with an object explorer in the left panel. Here we are about to add Astro's location to the scene.

6.1.2 Action Conditions

The other way we use models is to add conditional logic to our actions. That is, we can make the performance of the action depend on the status of models at whatever time the action is performed. You might have noticed a section in the Action Builder labeled 'Conditions'. Here we can add any number of model tests to control how the action works. Here's an example in the editor UI:



Name: Porch Light On

Category: lighting

Conditions

George is not home

Tasks

turn on Porch Light

Action 'Porch Light On' saved

We just added a 'condition' to our action. A condition is a test of a model that evaluates to true or false. In this case, the condition in the action means that the Porch Light will only be turned on if George isn't home. Action Builder enables very sophisticated logic, where we can have multiple conditions, conditions that alternate (this condition OR that condition), and whole other clauses (ELSE IF conditions, tasks). If you press the 'Test' button, the UI will display whether the conditions are true.

6.2 Types

HomeRun makes models easy to work with by establishing families, or **types** of models, and supports many built-in operations based on type. Let's see how this works in practice. There are 6 main model types in HomeRun, and here is a quick tour of them.

6.2.1 Number Models

Perhaps the simplest model to understand is the number model, which represents a simple number without any units. We would use this model when we want to capture a single whole number. Even though it lacks a unit, each number has a type, such as 'counter', or 'variable' which indicates how we would use it. You select the type when you create the model. For example, a 'counter' can only be changed (HomeRun prefers the term 'informs') using 3 verbs: 'increment', 'decrement' and 'reset'.

Another handy feature of a number model is a means of associating reference values against which the model value can be compared. These are called **limits** in HomeRun, and a model can have any number of limits defined for it. Limits each have a value (of course), a name and an icon. Having limits makes it very easy and intuitive to add action conditions like this: let's suppose we have an object with a duty-cycle of 100, meaning that we should repair or check a part after 100 uses. We would create a limit called 'check' with a value of 100, and then in an action, we could easily express our restriction as: 'if the use count is below the check limit, proceed with the tasks,
 else notify Bob via email'.

6.2.2 Value Models

In contrast to number, value models have a unit -- like degrees Fahrenheit -- and typically represent **measured** physical quantities, rather than the computed or assigned values typical of number models. The units are described by value types, which include names, symbols, icons, etc. so that visualizations of these models can expose the unit information succinctly.

Because there is often imprecision or fluctuation in the physical measurements that underpin value models, they do not use limits. Imagine, for example if you used an action condition like 'if temperature > 68', and your readings came in at 67, 69, 68 on successive samples due to random fluctuation. A more natural way to handle this case is using **ranges** of values, which is what HomeRun uses with value models. Thus you might define a 'comfort zone' range of 67-72 degrees. Then the action condition could read: 'if temperature remains in comfort zone, do x, else y', etc.

6.2.3 State Models

State models represent situations that reside in one of a fixed number of states, which are described by names (and icons, if desired). The simplest state models are what would ordinarily be called **boolean** variables (I.e. true/false states). For example, a state model could represent your location with respect to your home: you are home or not. A state model would express this as 2 states: home and away, and further provide names for the transitions from one to the other: 'arrival' is the transition from away to home, 'departure' from home to away.

State models can, of course, have more than 2 states, but you can go quite a long way with binary (2-state) models.

6.2.4 Set Models

When you have a list of one or more values, consider a set model. Remember that state models can only be in one of a fixed set of states; sets can contain from zero to an unlimited number of things. Each thing is represented as a name, and the only restriction in a set is that the same name cannot appear twice. For example, you might create an 'occupancy' set model containing the names of the people who are home at the measured time.

6.2.5 Calendar Models

These models are used to represent times as expressed in a calendar. Calendar models have a few sub-types, as follows. 'day of week' : Monday – Sunday 'day of month': 1st – 31st. 'month of year': January – December. Typical use in a condition would be: 'if today is a Thursday, do x'.

6.2.6 Time Models

Time models also express a time, but as seen by a clock rather than a calendar. These models can also be used to represent time intervals, like countdown timers, etc. A typical condition: 'if it's earlier than 7:30, do y'. You note in these examples, that the language of model conditions is tailored to the model type (“earlier than” for time models) – this results in very natural or 'idiomatic' condition language.

6.3 Informing Models

One very important topic that we have glossed over in our tour of models is the simple question of how the model values get updated and maintained? How does George's location model know when he comes home? HomeRun uses the term 'inform' a model for the update, and has a system whereby a model is given an 'informer'. Here are the basic mechanisms at work:

6.3.1 Internal Informer

An object may have a model that is closely bound to it's identity: a good example is the 'Sun' object with models 'rise' and 'set' for the sunrise and sunset time. These values may be calculated (if longitude and latitude are known), so there needs to be no external information for the model value to be determined. These are internally informed models, and there is nothing you need to do to ensure their value (except the initial configuration including setting coordinates).

6.3.2 Control Informer

When you add a model to an object in the Object Editor, there are 3 properties you may assign, and one is labeled 'Expose Control'. If you check this box, HomeRun creates a 'virtual control' whose purpose is simply to update the model. It is given the same name as the model. This allows you in Control Panel or in an action to update a model yourself.

6.3.3 Projected Informer

Probably the most common case, again set with a property named 'Project Informer' in the Object Editor checked, this will allow you to assign a value to a model in the context of an action in the Action Builder.

6.3.4 Wired Informer

One advanced feature of HomeRun is the ability to 'wire up' components in an object. A good example would be our Porch Light power status model – it makes sense that whenever the switch is turned on, the model should be updated to be 'on'. We can legislate this by setting up a 'circuit' between the control and the model. The UI for this is covered in the Advanced Topics chapter.

Chapter 7 - Advanced Topics

There are many powerful features in HomeRun that you might have rare occasion to use, but are invaluable if needed. Here's a very cursory look.

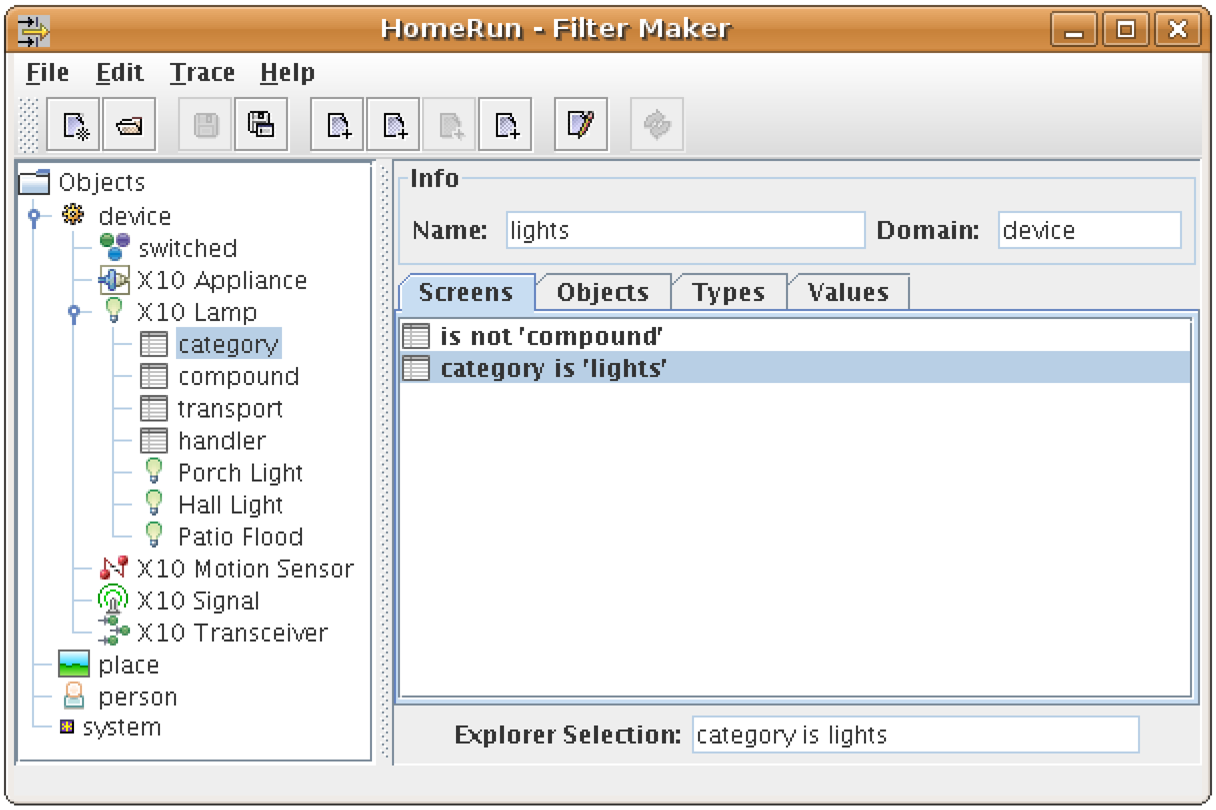
7.1 Object Filters

When you add, for example, an X10 Lamp device to your HomeRun system, it appears in the Control Panel in the 'lights' tab without any additional configuration. This is due to the use of a construct in HomeRun known as a filter. A filter is a description that 'picks out' some number of objects that match it. In this case the description is roughly: “is in category: lights”, but far more complex descriptions are possible.

A filter works much like a water filter on a tap: it is composed of one or more **screens**, and the objects that the filter identifies are those that can *pass through* all of the screens. A screen is simply a description of a characteristic of an object - if the object has that characteristic – it passes through that screen. These characteristics can vary during the life of an object; for this reason, filters can also be **dynamic** - a new object created after the filter was defined can have the characteristic or a former member lose it.

Filters are used 'under the covers' in many places in HomeRun, so you should try to familiarize yourself with their construction and use.

As always, there is a editing UI for filters – Filter Maker (Expert->Filter from Console) -- shown here with the filter above:



Name: lights
Domain: device

is not 'compound'
category is 'lights'

Explorer Selection: category is lights

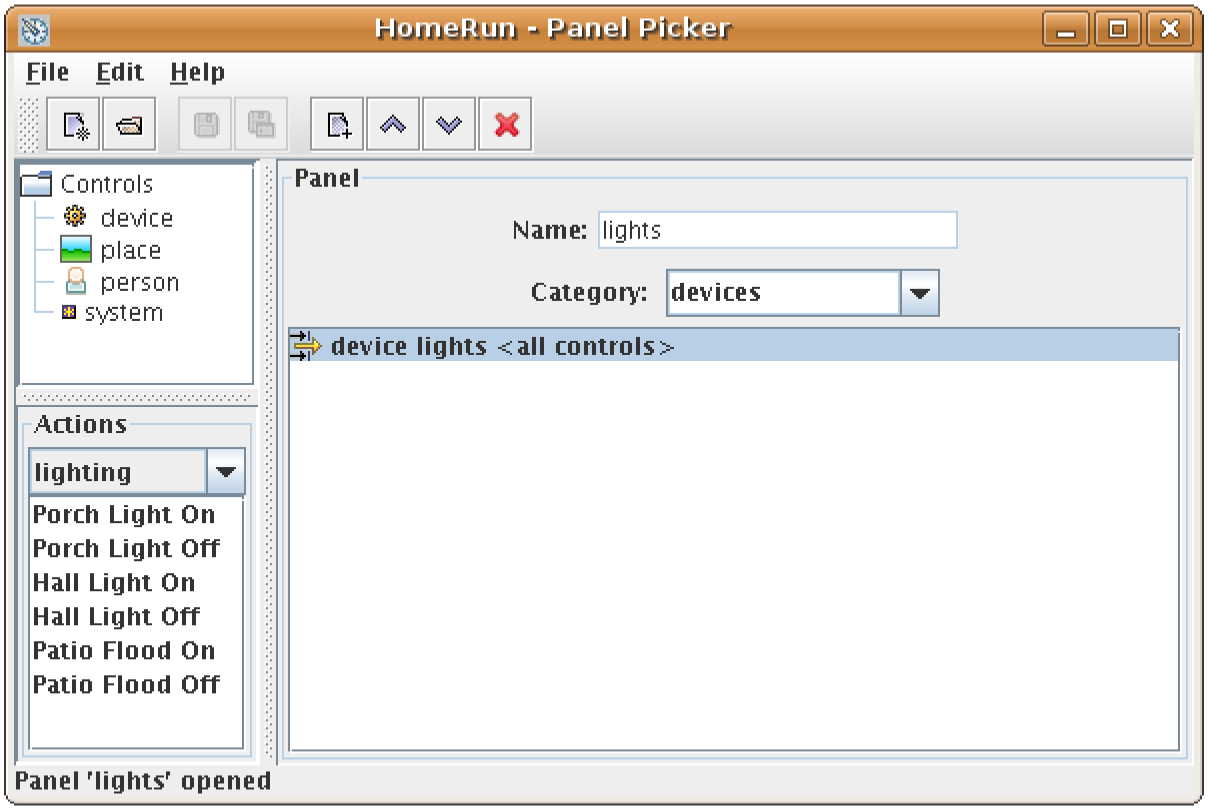
We have given the filter the name 'lights' and it contains 2 screens. The Objects tab (not shown open) displays the list of currently filtered objects, so you can interactively verify the selection is what you want.

Filter Maker employs a 'build by example' style to construct the screens: here we have used the object explorer to navigate to the X10 Lamp type, and selected the property 'category'. That selection becomes the screen highlighted on the list.

7.2 Control Panels

The Control Panel UI displays groups of objects and their controls, as we have seen. What is less obvious is that the definition of these groups is a HomeRun construct that you can create or edit. This gives you the ability to make arbitrary groups appear together: you might want all the lights on the second floor, e.g.

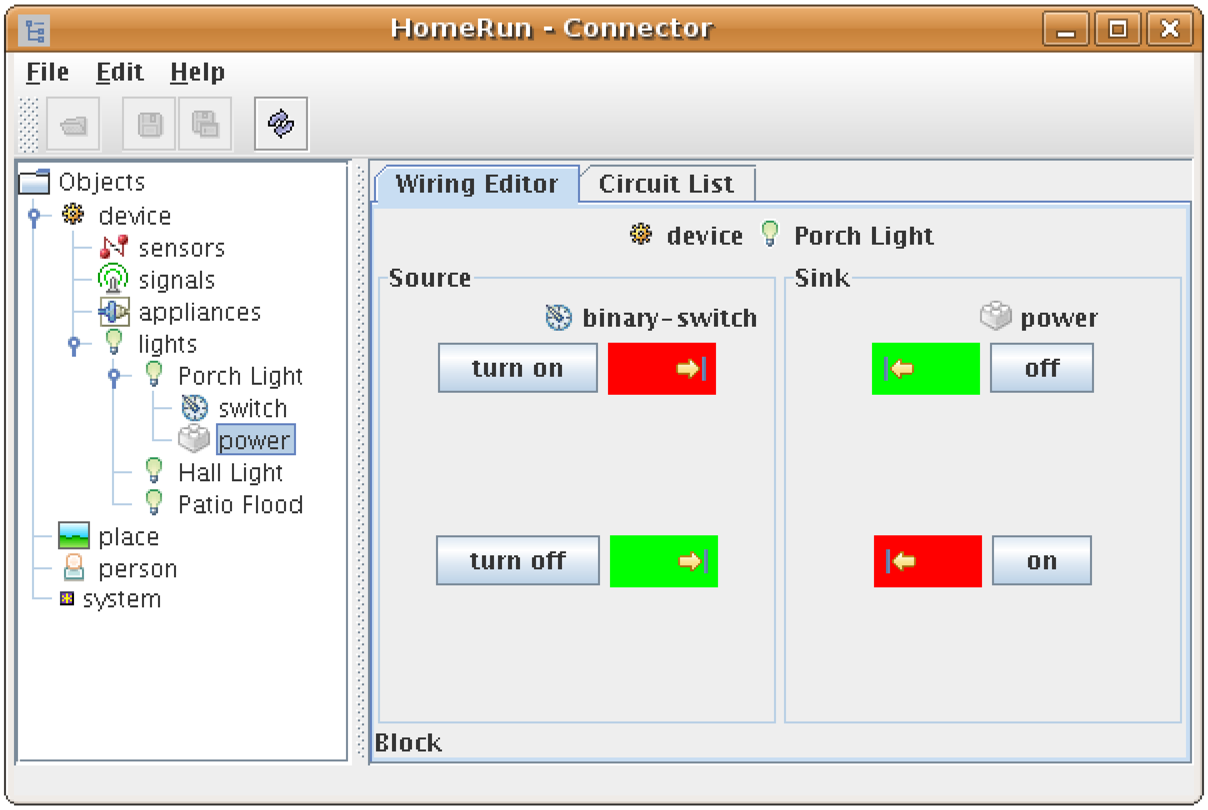
The UI for managing **panels,** as they are known, looks like this (Expert->Panel from the Console):



Here we see a list containing one specification, using the filter we encountered in the previous section called 'lights', followed by the instruction 'all controls' which means that the panel will display all controls the objects have. You can limit the controls to any type if desired. Also notice that actions may be added to a panel, using the drop-down lists in the lower-left panel.

7.3 Circuits

In Chapter 4 when discussing how models are informed, we mentioned a facility to 'wire' connections between a source component (control or emitter) and a model. A collection of wires is known as a **circuit** and although you may not have much occasion to use it, there is a UI application for managing circuits. Called the 'Connector' (from the Console Expert->Wire), it looks like this:



Here the editor tab is displayed, showing how wires are represented as matched colors from the 'source' component to the 'sink' component. The circuit list tab displays all the defined circuits. One particularly powerful feature of circuits is the ability to define **templates** which are circuits that apply to whole classes of objects, not individual ones. The circuit above is in fact such a template, which means that whenever a the 'turn on' control point of an object with a switch is executed, if the object also contains a 'power' model, it will be updated to the 'on' state, as one would expect.

Templates may be overridden on a per object basis by simply adding a circuit for the object.

Chapter 8 – Administration

HomeRun strives to be very easy to run and manage, and provides several tools and Uis for helping you administer your system. Before getting a tour, a brief overview of HomeRun's architecture as it pertains to administration.

8.1 Server Life-Cycle

As previously mentioned, HomeRun software is divided into a server and client software that communicates with it. The server has an internal life-cycle, in the sense that it can be in one of two states that you control. When you start the server with **hrserver.bat** batch file, it is not completely running, but only in a ready state known as the bootstrap state. When in this state, most client applications cannot operate. In fact, when the Console first launches, you will notice that almost all the menu choices are disabled, except those in the Admin group. Fortunately one of the enabled choices under Admin is Server->Start which brings the server to the fully operational state. Most of the time it will remain in this state, and you normally will not have to stop it. There are, however a few cases where after installation of new packages, you will be instructed to restart the server (I.e. stop and start it), as you would with many software installations.

8.2 Users and Authentication

Another important concept to understand is the 'all-or-nothing' model of users taken by HomeRun. When first installed, HomeRun is considered to be in the **open** state, which means simply this: it has no notion of distinct users of the system, so it does not ask you to identify or authenticate yourself. In consequence, anyone using the software can do anything. In fact, you might want to leave the system open for some period of time – especially while learning it: you won't have the bother of logging in to use the system. If all users are known and trusted, one could make the more extreme case that it remain open always for simplicity.

However, you often want to reserve certain powers to certain people, and HomeRun supports a role-based authorization model. As soon as you define a single user (from Console Admin->User->Manage), then the system is considered to be in the **closed** state and users must identify and authenticate themselves to operate the system. This also creates a potential Catch-22: what if the first user created lacks administrative authority? Then how would we create that administrator? To avoid this trap, HomeRun insists that the first user defined be an administrator, so that he or she can then define users of lesser powers. You can play the whole process in reverse too: make sure that if you are deleting all users to return to the open state, the last user left is an administrator.

8.3 Roles

HomeRun understands a very simple set of roles, and limits access to certain programs and Uis based on those roles. When creating a user, you simply assign one of more roles to him/her. The roles are:

admin – can administer the system.
editor – can use the editor applications to create new objects and components
user – can run the control panel or messaging
other – also used for anonymous users (= not logged in)

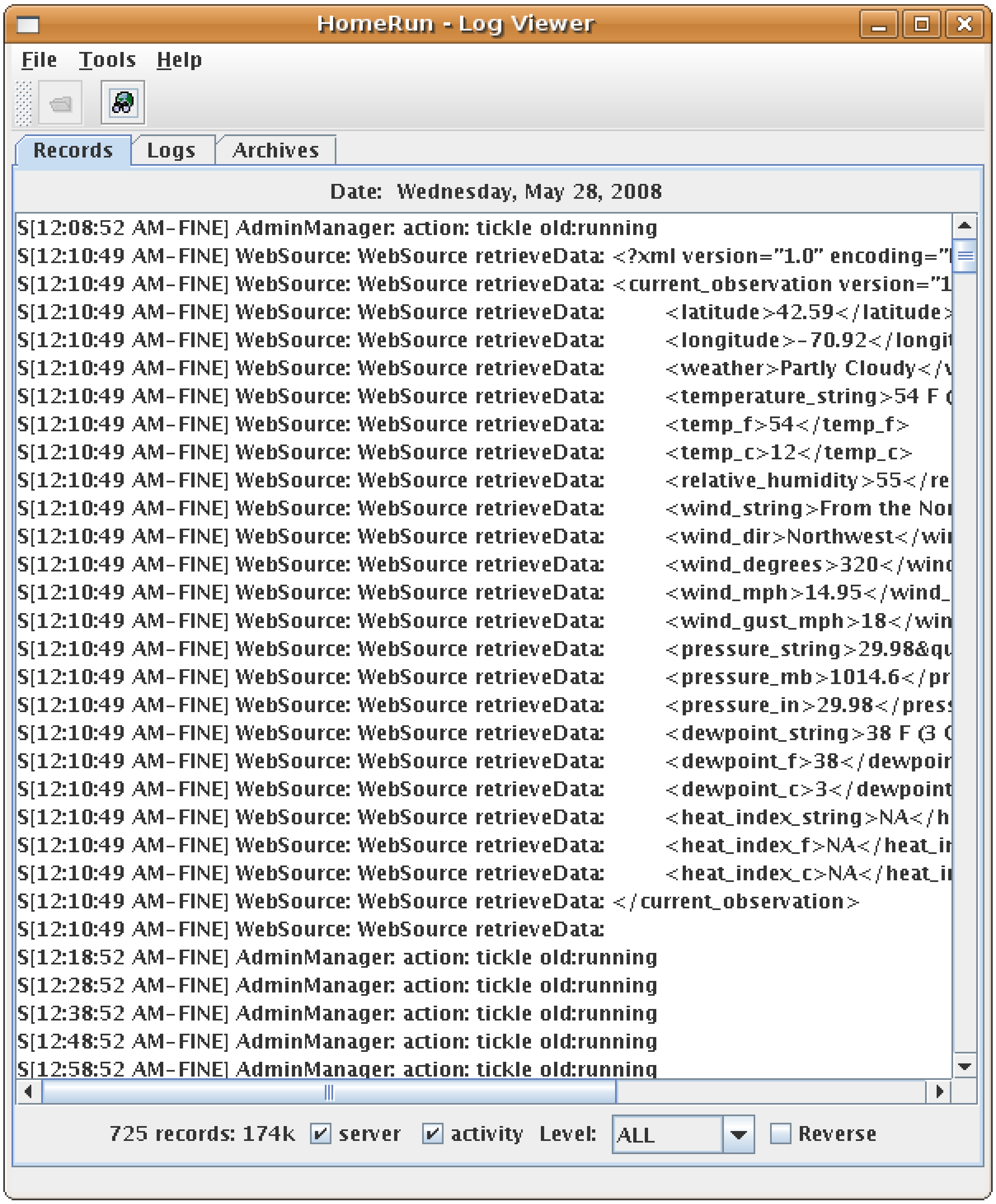
You should note that roles are not encompassing – meaning that an admin is not automatically also an editor, etc. You must assign each role to each user. Users are managed in a UI application called Manager (Admin->User->Manage from the Console) where all you do is assign the user name, and set the roles above. New users are all given the default password 'homerun', but they should quickly select their own password in a dialog (Admin->User->Change Password).

8.4 Administrative Duties

We already discussed one of the major duties, which is selection and installation of HomeRun packages (Chapter 3), which determine the functionality of the system. Most packages require further configuration, which is accessed through the Setup program. You may also import both images (gifs and jpegs) for use in scenes, and icons for use in various places, e.g. objects, model states, models, etc Setup also has options to delete old scene snapshots.

8.5 Logging

HomeRun has a flexible system of logging, and provides a tool, the Log Viewer, for managing log data. Normally three different logs are kept: the **system** log keeps data about the internal operation of the server. The **activity** log has a record of the use of the HomeRun system, when actions are taken, or events occur, etc. Finally a **user** log is maintained for the sole purpose of recording data that you supply in actions. Here's a snapshot of the Log Viewer, (Admin->Logs from Console):



Date: Wednesday, May 28, 2008

```
S[12:08:52 AM-FINE] AdminManager: action: tickle old:running
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData: <?xml version="1.0" encoding="
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData: <current_observation version="1
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <latitude>42.59</latitude>
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <longitude>-70.92</longit
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <weather>Partly Cloudy</
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <temperature_string>54 F (
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <temp_f>54</temp_f>
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <temp_c>12</temp_c>
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <relative_humidity>55</re
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <wind_string>From the No
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <wind_dir>Northwest</wi
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <wind_degrees>320</wind
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <wind_mph>14.95</wind_
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <wind_gust_mph>18</win
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <pressure_string>29.98&qu
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <pressure_mb>1014.6</pr
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <pressure_in>29.98</pres
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <dewpoint_string>38 F (3 C
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <dewpoint_f>38</dewpoi
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <dewpoint_c>3</dewpoint
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <heat_index_string>NA</h
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <heat_index_f>NA</heat_i
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:      <heat_index_c>NA</heat_i
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData: </current_observation>
S[12:10:49 AM-FINE] WebSource: WebSource retrieveData:
S[12:18:52 AM-FINE] AdminManager: action: tickle old:running
S[12:28:52 AM-FINE] AdminManager: action: tickle old:running
S[12:38:52 AM-FINE] AdminManager: action: tickle old:running
S[12:48:52 AM-FINE] AdminManager: action: tickle old:running
S[12:58:52 AM-FINE] AdminManager: action: tickle old:running
```

725 records: 174k ☑ server ☑ activity Level: ALL ☐ Reverse

The Log viewer has three tabs, one for a listing of log files (one per day per type), the log records (shown), and a listing of archived logs. Tools include creating archives (which compress the logs and save disk space), and searching log data for keywords. At the bottom are filters, including severity of log entry (Level).