jHomeNet

Home Automation using Java

DRAFT User Manual

Written by: Dave Irwin (jhomenet@gmail.com)
Project webpage: http://jhomenet.sourceforge.net

Last updated: November 3, 2007

Acknowledgements

Document revisions

Doc version	Date	Description
0.1	12/20/2006	Updated for version 0.1 of the jHomeNet server
0.2	2/11/2007	Updated for version 0.2 of the jHomeNet server
0.3	3/15/2007	Updated for version 0.3 of the jHomeNet server
0.4	5/28/2007	Updated for version 0.4 of the jHomeNet server
0.5	10/25/2007	Updated for version 0.5.3 of the jHomeNet server.
0.5.1	11/3/2007	Updated for version 0.5.3 of the jHomeNet server.

Table of Contents

Section 1 - Introduction

Section 1.1 – Description

jHomeNet is a suite of open-source software applications that when used with hardware sensors and devices can be used to help monitor and automate systems around your house. Currently the jHomeNet software suite supports Dallas Semiconductor's 1-Wire hardware along with X10 hardware. Future releases of the jHomeNet software suite may support additional hardware.

The suite of software currently includes the following packages:

- jHomeNet commons library: Provides common tools.
- jHomeNet server: Is responsible for maintaining and managing hardware, hardware polling, and the sensor responsive system. Also provides hardware, data, and sensor responsive persistence functionality.
- jHomeNet UI: Provides the user interface classes and tools.
- jHomeNet plugins: A collection of jHomeNet plugins that can be added to the jHomeNet server to provide additional functionality without having to recompile the server.

The jHomeNet server currently provides an operator with a command line interface (CLI) for interacting with the server. However, through the server CLI an operator may also open the jHomeNet graphical user interface (GUI). The GUI provides the same functionality that the CLI does but graphically.

In the future, a jHomeNet client will also be developed that provides an operator the same functionality with the added ability to connect to the server remotely. The client GUI would be nearly identical to the server GUI but could allow remote operators to connect using a TCP/IP based network connection.

Section 2 – System Requirements

Section 2.1 – Software Environment

The purpose of this section is to provide information on the required software environment in order to run the jHomeNet software suite.

Section 2.1.1 – Runtime Environment

To run the jHomeNet software suite (either the jHomeNet server or the jHomeNet client), a compliant Java Runtime Environment (JRE) version 1.5 must be installed and properly configured.

Section 2.1.2 – Operating Systems

Any operating system that supports a version 1.5 JRE should be capable of supporting the jHomeNet server and client. The necessary 1-Wire and X10 communication drivers will also be required if external sensors are to be used.

As of this writing, the jHomeNet software suite has only been developed and tested on a Windows 2000 and XP platform, however there is no reason that the jHomeNet suite shouldn't be able to run on a Linux/Unix platform as well.

Section 2.1.3 – SQL server

In addition to the JDK and/or JRE, the jHomeNet server also requires a compliant SQL server. As of this writing, the MySQL version 4.1 has been used for development and testing. The free MySQL server is available from MySQL's website http://www.mysql.com.

The only manual configuration to the database that is currently required is the creation of the jHomeNet database. Create a new database called `jhomenet_db`. You may leave the database empty as the Ant `init-db` tool will create the necessary database tables and keys.

For database maintenance and configuration, MySQL's administration tools along with a number of third party software applications exist. For example, phpMyAdmin along with an Apache web server could be used. Other SQL front ends used during the development process include MySQL Front (http://www.mysqlfront.de/).

Section 2.2 – Hardware Environment

The current jHomeNet software suite supports both Dallas Semiconductor's 1-Wire network and hardware and X10 network and hardware. For specific details on installing the necessary communication drivers refer to Section 3.3.

Section 3 – System Configuration

The purpose of this section is to provide instructions on building, deploying and configuring a jHomeNet server and client along all of the necessary third-party dependent software applications and drivers.

The jHomeNet suite comes with binary executables as part of the distribution so building the binary executables from the sources should not be required. However, the jHomeNet suite still needs to be configured in order to run. You may skip Section 3.1 if you're not interested in building the jHomeNet suite from the sources.

Section 3.1 – Building the jHomeNet Suite from Source

Each jHomeNet package includes an Apache Ant build file for building the package's source files into executable class files. If Ant is not already installed, download it from the following website: http://ant.apache.org/. (The installation and configuration of Ant itself is beyond the scope of this document. See the Apache Ant website for further details.)

As noted, there are several jHomeNet packages that make up the jHomeNet suite of tools. Some packages have dependencies on other packages. In particular, the following is a list of the jHomeNet package dependencies:

Package Name	*Package Dependencies*
jHomeNet commons	None
jHomeNet UI	jHomeNet commons
jHomeNet server	jHomeNet commons, jHomeNet UI

When a dependent package is rebuilt it must be copied into the library folder (lib in most cases) of the packages that depend on it. Fortunately, there is an Ant task included in the jHomeNet server's `build.xml` file that automates this process. In particular, to build the jHomeNet commons, UI, and server source packages and copy the necessary built package jar files to the necessary folders, run the `build-suite` task (or the `build-clean-suite` to delete any existing class files before building).

Refer to Section 8 (Development Environment) for more details on the steps to properly setting up a development environment and build the jHomeNet suite. For a list of all other available jHomeNet Ant tasks and other information, refer to Appendix A.

Section 3.2 – SQL Server Configuration

The jHomeNet server requires an SQL compliant database to operate. This section describes the steps necessary to setup and configure an SQL database for use with a jHomeNet server.

The jHomeNet server comes with support for persisting information in a database using Hibernate, an object/relational (O/R) tool available from http://www.hibernate.org.

Install a SQL compliant database server (as previously noted, MySQL has been used for development and testing). After installing and setting up the database server, the jHomeNet database has to be created and initialized. Fortunately, an Ant script has been included with the server's `build.xml` file that both creates the jHomeNet database and creates the necessary tables. However, before running the task, both the Hibernate and `build.xml` files need to be updated with the appropriate SQL server connection information.

To configure the `build.xml` to connect to the database server, you'll need to modify the `default.properties` file located in the root directory. Below is a code example showing a default configuration:

```
################################################################
# SQL properties
################################################################
sql.driver     = com.mysql.jdbc.Driver
sql.url        = jdbc:mysql://localhost/
sql.userid     = admin
sql.password   = admin
sql.dbname     = jhomenet_db
```

Code Listing – The `default.properties` SQL properties

To configure Hibernate to connect to the database server, you'll need to modify the Hibernate configuration file, called `hibernate.cfg.xml`, located in the `/resources/conf/` folder. Here you'll need to enter the corresponding Hibernate dialect information, the appropriate SQL server driver, and database URL, username and password. For more information refer to the Hibernate documentation. Below is a code example from a default Hibernate configuration file.

```
<property name="hibernate.dialect"> org.hibernate.dialect.MySQLDialect </property>
<property name="hibernate.connection.driver_class"> com.mysql.jdbc.Driver
</property>
<property name="hibernate.connection.url"> jdbc:mysql://localhost/jhomenet_db
</property>
<property name="hibernate.connection.username">admin</property>
<property name="hibernate.connection.password">admin</property>
```

Code Listing – The Hibernate `hibernate.cfg.xml` database properties

After updating both the `default.properties` and `hibernate.cfg.xml` configuration files, you're ready to create and initialize the jHomeNet database. Fortunately, this process has been automated by running the server's `init-db`

Ant task found in the server's `build.xml` file. This task, as described in Appendix A, will create the necessary database tables, indexes, etc. necessary for the jHomeNet server. When this task completes, verify that the tables have been added to the database.

Section 3.3 – Installing Network Drivers

The purpose of this section is to provide a general overview of the hardware driver installation steps. Please refer to the specific hardware driver vendor documentation for detailed information.

Section 3.3.1 – 1-Wire

The 1-Wire communication network was designed by Dallas Semiconductor and allows small sensors and devices to be interconnected using a very simple communication bus. A 1-Wire network provides half-duplex bidirectional communications between a host/master controller and one or more slaves or hardware objects over a single 1-Wire line. A 1-Wire network requires just three things:

- 1-Wire hardware devices
- Copper plant consisting of either telephone or CAT-5 cable
- A 1-Wire host/master controller for interfacing between a computer and the copper plant

1-Wire has several advantages in that it is extremely easy to deploy and configure and add additional hardware devices. Both power and data communication for slave hardware devices are transmitted over the single 1-Wire line. Each slave hardware device on a 1-Wire network has a unique unalterable (ROM) 64-bit serial number (or address) that will never be repeated by another 1-Wire hardware device.

For the jHomeNet server to communicate with deployed 1-Wire hardware devices, a host/master controller must be installed on the jHomeNet server. Currently there are both serial and USB host/master controllers available. The host/master controllers require software drivers to be installed. For Windows platforms, there is a choice of either the TMEX native drivers or the RXTX serial communications API. The TMEX native drivers are available on the iButton website at the following address: http://www.maxim-ic.com/products/ibutton/software/tmex/index.cfm. The latest version, version 4.00, contains the necessary drivers and support libraries for each of the 1-Wire host/master controllers available from Dallas Semiconductor Maxim, including the 1-Wire USB adapters (DS9490). The current release of 1-Wire drivers supports Microsoft 32-bit Windows ^{TM} including Windows XP, ME, 2000, 98, NT (no USB support), and 95 (no USB support). The installation comes as an executable file and requires no additional configuration beyond running the installation application.

All other platforms will require the RXTX API for communicating with the DS9097U serial port adapter. Visit the Dallas Semiconductor Maxim OneWireViewer website to download the latest RXTX driver files: http://www.maxim-ic.com/products/ibutton/software/1wire/OneWireViewer.cfm. You can also visit the RXTX website at http://www.rxtx.org/ for updates or current releases. Consult the `install` and `readme` files included as part of the distribution for directions on installing the drivers.

Section 3.3.2 – X10

Like a 1-Wire network, an X10 network is an extremely simple network to deploy and configure. An X10 network communicates with hardware devices by making use of existing AC power lines. While X10 provides a number of different capabilities, the for the jHomeNet system it is primarily used with on/off switches.

In order to communicate with the X10 network, the jHomeNet server uses a standard serial connection to an X10 gateway. The Java COMM API software driver, version 2.0, available from Sun has been used.

The most recent release of the COMM API from Sun (version 3.0, update1) does not include support for the Windows platform. Specifically, the latest release only supports Solaris x86, Solaris 8 & 9 (Sparc) and Linux. To download the latest COMM API from Sun, follow the http://java.sun.com/products/javacomm/ link. The older version 2.0 Comm API from Sun is still available for download at the following location: https://jsecom8a.sun.com/ECom/EComActionServlet;jsessionid=9C14E7EB934D26FBA3D49FC5284D2A9B. As mentioned in the 1-Wire driver installation section, other options include serial drivers available from RXTX at the following site: http://www.rxtx.org.

Installing the COMM API has proven to be difficult as the directions included with the library don’t always work. The following is a set of steps previously used to install the COMM API version 2.0 on a Windows ™ 2000 machine.

1. Unzip the contents to a temporary folder.
2. Copy the `win32com.dll` file to `<JRE>\bin` folder where `<JRE>` is the location of the Java Runtime Environment installation.
3. Copy the `javax.comm.properties` file to the `<JRE>\lib` folder.
4. Copy the `comm.jar` file to the `<JRE>\lib\ext` folder.

This installation does not require any edits to the `classpath`. It should be noted, however, that any JRE updates may require a new installation of the COMM API files.

Section 4 – Server Configuration

The purpose of this section is to provide an overview of the jHomeNet server configuration required prior to starting the jHomeNet server.

Section 4.1 – Server Configuration Files

The release includes a default set of configuration files in the `<jhomenet-server folder>\resources\conf` folder.

jhomenet.cfg.xml

This is the main server configuration file. The file is a standard XML file using key/value properties (property name = property). A list of the currently used property names and acceptable inputs is outlined below:

XML node or attribute	***Description***
`<jhomenet-config>` node	The main configuration node. All other nodes must be child of this node.
`<startup-config>` node	Node that is used to define server startup parameters.
`<console>` node	A child node of `<startup-config>`. This node defines the console properties.
`display` attribute	An attribute of the `<console>` node. Set to `true` to display the console at server start. Set to `false` to not display the console at server start.
`<hardware-polling>` node	A child node of `<startup-config>`. This node defines the hardware polling startup properties.
`start` attribute	An attribute of the `<startup-config>`. Set to `true` to start hardware polling at server start. Set to `false` to disable hardware polling at server start.
`<hardware-config>` node	A child node of `<jhomenet-config>`. This node defines hardware configuration properties.
`type` attribute	An attribute of the `<hardware-config>` node. Set to `xml` for an XML file based hardware configuration. Set to `txt` for a text file based hardware configuration.
`filename` attribute	An attribute of the `<hardware-config>` node. Use this to set the XML or text hardware configuration filename to be used.
`<authentication-config>` node	A child node of `<jhomenet-config>` used to set authentication configuration properties.
`<type>`	`<authentication-config>` sub-node used to indicate an authentication type section. Define the authentication service type to be used for user login. The software currently supports three

XML node or attribute	Description
	types of authentication services: JAAS (use `jaas`), JDBC (use `jdbc`), and a simple service (use `simple`). The property may not be left blank.
`<network-config>`	A child node of `<jhomenet-config>` used to indicate a network configuration section.
`<firewall-enabled>`	A child node of `<network-config>` used to indicate whether the network is firewall enabled. Acceptable input values include `true` or `false`.
`<workqueue-config>`	A child node of `<jhomenet-config>` used to set work queue configuration properties.
`<threadsize>`	A child node of `<workqueue-config>` used to set the number of threads allocated to the work queue. Valid values: integers greater than 0.

Hardware.cfg.xml

This file defines the hardware driver names (along with any necessary driver configuration filenames) and the types of hardware available to the server. This file shouldn't require editing unless additional types of hardware are added.

Included with the distribution are a set of dummy hardware configuration files that allows a user to run a server without actually having a 1-Wire or X10 network installed. To use this dummy hardware in the `jhomenet.cfg.xml` configuration file change the hardware configuration file reference from `hardware.cfg.xml` to `hardware_test.cfg.xml` and make sure that the test hardware configuration file is present. You will need to restart the server for the changes to take place. This will start the server with a number of dummy hardware sensors and devices.

X10 configuration file (i.e. x10.cfg.txt)

The current X10 software is unable to automatically detect the X10 hardware gateway to the X10 communication network. Therefore it is necessary to instruct the jHomeNet application what communication interface to use to connect to the X10 network. The x10.cfg.txt file is used to define the X10 network interface parameters. Since the current version of the jHomeNet application only supports a serial X10 interface the only valid parameter is the serial port to be used.

X10 hardware configuration file (i.e. x10-containers.cfg.txt)

The X10 network is unable to scan for hardware unlike the 1-Wire network. Use this file to define the available X10 hardware.

Others

The remaining configuration files should not require any editing unless further development is required.

Included with the distribution are a number of test configuration files. These files can be used to setup and start the server without requiring an actual 1-Wire or X10 communication network.

Section 4.2 – Server Startup Files

When the server's console session is started, it has a built-in feature to look for a `.login` file in the `<root folder>\resources\login` folder. This file is optional and is not required to start the server or a console session. If this `.login` file is found, the contents of this file are executed as if an user were entering console commands. All of the commands are executed in the background (i.e. as if the "&" were included as a command line argument).

The default `.login` file contains a command to start the main jHomeNet GUI. Other console commands may be added to this file. Each command should be placed on a new line. Comments may be added to this file using the # symbol. Refer to the code listing below for the default `.login` file contents.

```
# The jHomeNet server console login script. This file is called
# when the main jHomeNet server console is opened. Known console
# commands may be included in this file along with space separated
# command line parameters.

# Open the main window
main-window
```

Code Listing – Sample `.login` file

Section 5 – Starting the Server

This section provides information on starting the jHomeNet server.

Section 5.1 – Running the server

There are a number of available script files included with the jHomeNet distribution to enable easy server startup. For Windows, use the included `server_jar.bat` batch file (future releases will also include startup scripts for Linux). Note that if you're developing new features in the jHomeNet application you can use the `server_dev.bat` file that is automatically created when the project is built..

As startup, the server goes through a number of different boot procedures during startup. The boot process displays a splash screen that provides server initialization feedback. See Figure 5.1 below for an example splash screen.

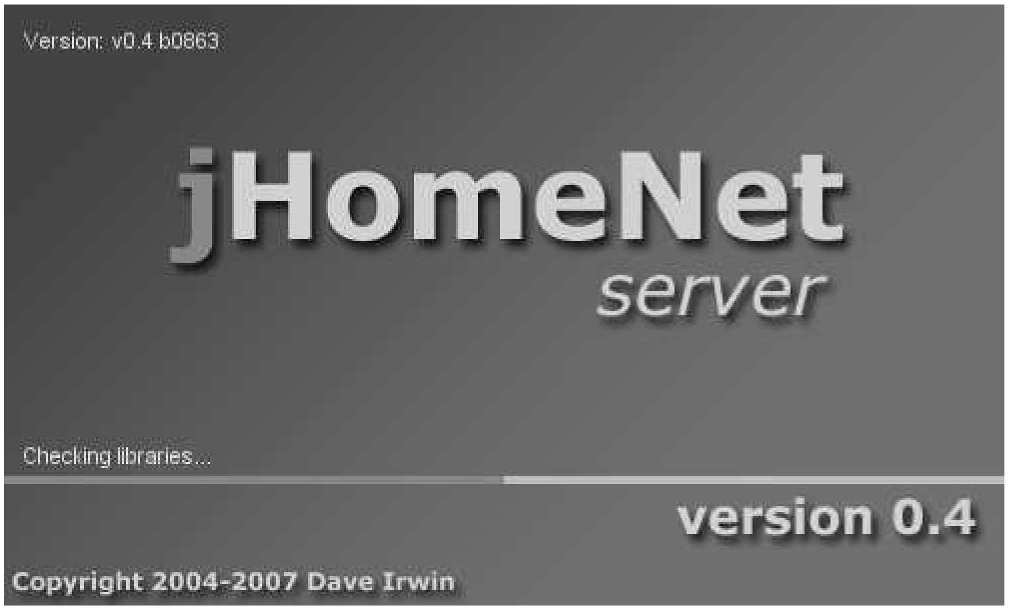
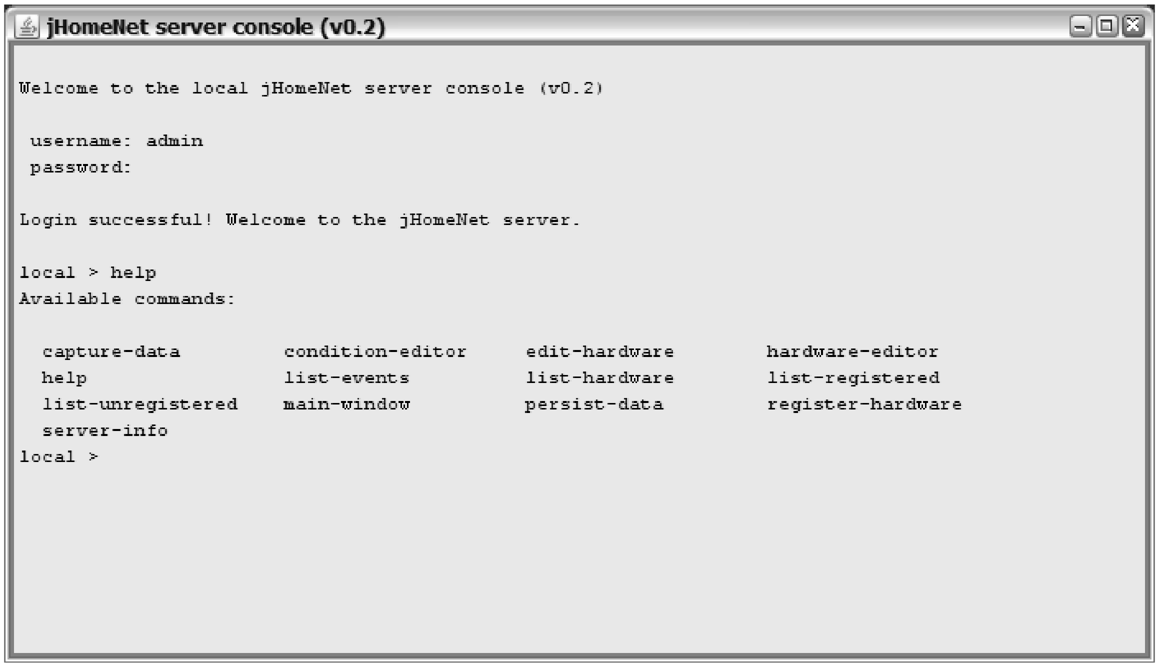


Figure 5.1 – Sample jHomeNet server splash screen

When the server initialization is complete, a local console session will be started. This console provides a basic interface to the server for the user. The user is first prompted for a username and password. By default, the administrative username and password is `admin/admin` respectively.

As discussed in Section 4.2, after a successful user login the console session startup process will look for an available `.login` file. If the `.login` file is found the commands in this file are executed as if the user entered them at the console. The default `.login` file has a command that starts the main GUI.

Once the user has successfully logged into the jHomeNet server, they may enter a number of different commands. For a listing of available jHomeNet console commands, type `help` at the prompt. See Figure 5.2 below for an example console.



```
Welcome to the local jHomeNet server console (v0.2)

 username: admin
 password:

Login successful! Welcome to the jHomeNet server.

local > help
Available commands:

  capture-data        condition-editor     edit-hardware        hardware-editor
  help                list-events          list-hardware        list-registered
  list-unregistered   main-window          persist-data         register-hardware
  server-info
local >
```

Figure 5.2 – Example jHomeNet server console

Section 6 – Server Features

The purpose of this section is to provide an overview of the features included with the jHomeNet server.

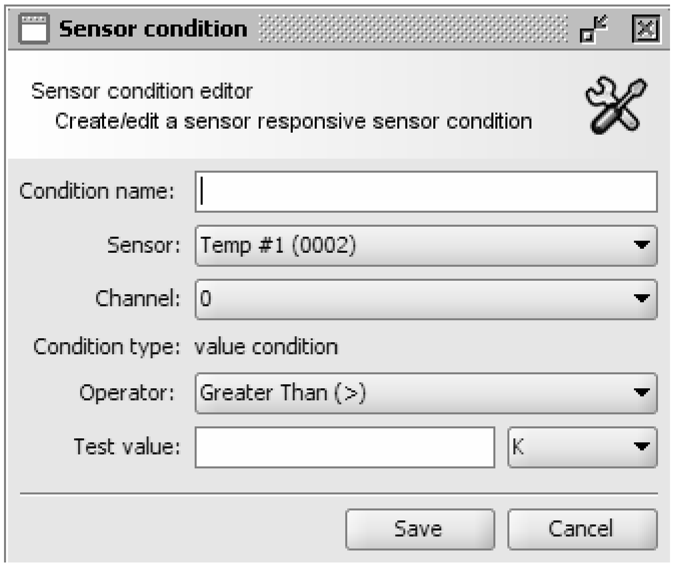
Section 6.1 – Sensor Responsive Programming

Sensor responsive programming (SRP) is a feature that adds dynamic control functionality to the jHomeNet suite of applications. Without SRP, jHomeNet is just a data acquisition and manual control program. Still quite useful, but even more so when SRP is included.

Section 6.1.1 – Conditions

The first thing you'll probably want to create in developing your SRP system is a condition. A SRP condition object encapsulates a logical conditional statement. As an example, the simplest condition is simply a Boolean condition that has a desired state equal to either `true` or `false`. That is, a user can set the desired state and when the condition is evaluated it simply returns the desired state. A more practical condition might make use of a temperature sensor and could be used to check if the measured temperate is less than a certain test temperature.

The jHomeNet suite currently supports two types of conditions: single sensor value based conditions (i.e. is a measured value greater than or less than some test value) and two sensor difference value based conditions (i.e. is the measured difference between two sensors greater than or less than a test value).

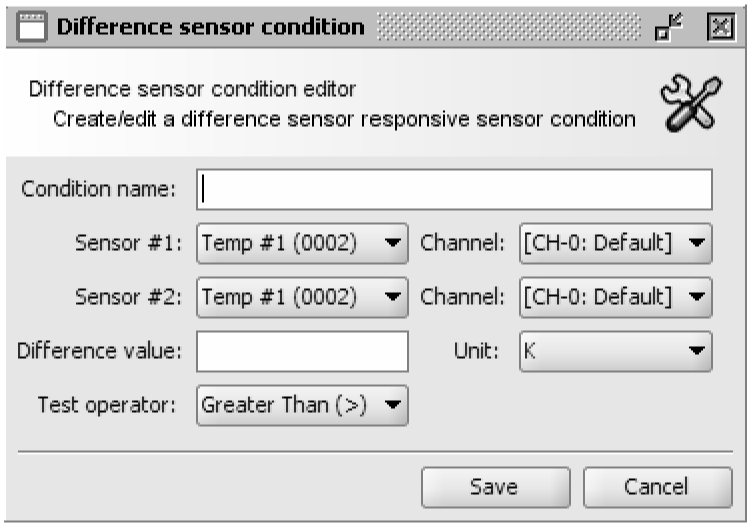

Sensor condition

Sensor condition editor
Create/edit a sensor responsive sensor condition

Condition name:
Sensor: Temp #1 (0002)
Channel: 0
Condition type: value condition
Operator: Greater Than (>)
Test value: K

An operator may create a new condition by right-clicking on the Conditions node in the main jHomeNet tree and selecting either the New Sensor Condition option or the New Difference sensor condition option. Selecting the former will open a new sensor condition editor window as seen in the figure at left. Here a user creates a unique condition name, selects the desired sensor and I/O channel, the operator (either greater than or less than), the test value and the test value unit.

Alternatively, the user may choose to create a sensor difference condition. Selecting the latter option will open the editor window as seen in the figure at right. Again, the user creates a unique condition name but then must also select two different sensors and their respective I/O channels, the difference value, the difference value unit, and the test operator (either greater than or less than).



Difference sensor condition editor
Create/edit a difference sensor responsive sensor condition

Condition name:
Sensor #1: Temp #1 (0002) Channel: [CH-0: Default]
Sensor #2: Temp #1 (0002) Channel: [CH-0: Default]
Difference value: Unit: K
Test operator: Greater Than (>)

There's also a Boolean condition where the user may set the desired state that the condition will return. This type of condition is primarily used for testing and development.

In all instances when the user clicks on the Save button the condition is permanently stored in the jHomeNet database for later use.

Users may edit existing conditions by selecting the condition tree object, right clicking on the particular condition, and selecting the Edit option. This opens the respective condition editor window which allows the user to edit some (but not all) of the condition fields.

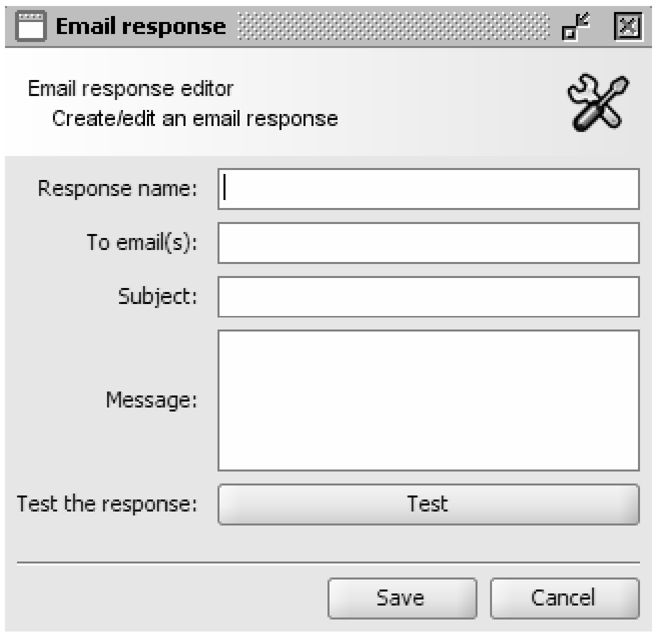
<u>Section 6.1.2 – Expressions</u>

An expression is really just a combination of one or more conditions using Boolean operators such as AND, OR, or NOT. For example, an expression could contain two conditions combined with the Boolean AND operator. In this case both conditions must evaluate to true in order for the expression to evaluate to true.

<u>Section 6.1.3 – Responses</u>

In addition to conditions, SRP allows the creation of responses. Responses are what is executed when an expression is evaluated. The current jHomeNet implementation has four types of responses:

- Message response: this type of response records a text message in the jHomeNet log
- Event response: this type of response creates and records a new event in the event logger
- Device response: this type of response allows a hardware device (such as a switch) to be controller
- Email response: this type of response creates and sends an email with a given subject and message



Response name:

To email(s):

Subject:

Message:

Test the response: Test

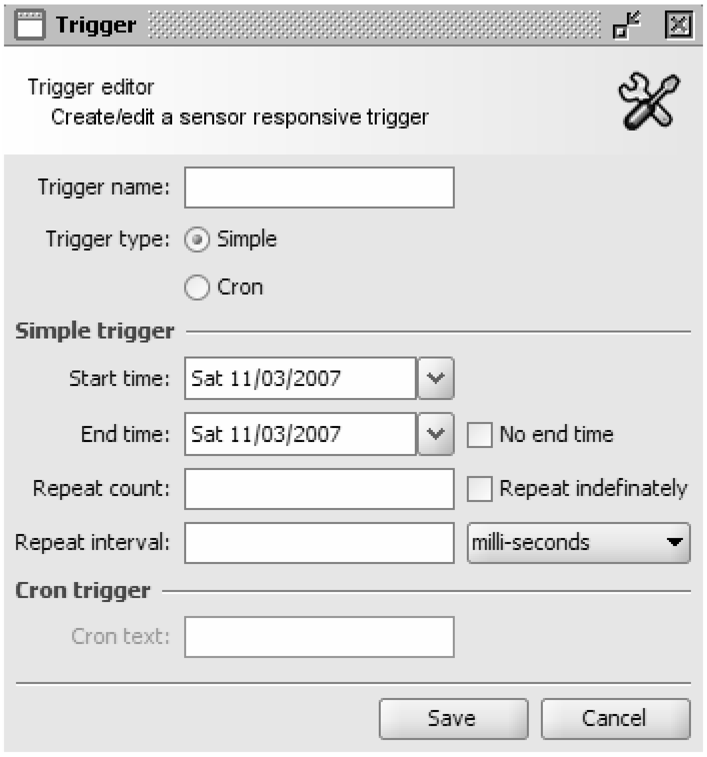
Save Cancel

Each response type has a different editor that allows a user to create and customize the responses.

To create a new response, right-click on the Response node in the Sensor Responsive tree on the left side of the main jHomeNet application. From there a user may select one of four response types as highlighted above. At left is the current email response editor. All in all cases the user must enter a unique response name. When finished, click on the Save button. This will permanently save the response in the jHomeNet database for later use.

Users may edit existing responses by selecting the response tree object, right clicking on the particular response, and selecting the Edit option. This opens the respective response editor window which allows the user to edit some (but not all) of the response fields.

<u>Section 6.1.4 – Triggers</u>



Trigger name:

Trigger type: [✓ Simple] Cron

Simple trigger

Start time: Sat 11/03/2007

End time: Sat 11/03/2007 ☐ No end time

Repeat count: ☐ Repeat indefinately

Repeat interval: milli-seconds

Cron trigger

Cron text:

Save Cancel

Triggers are used to define how often an expression should be evaluated. Triggers are a very powerful feature and provides a lot more detailed control beyond simply setting the trigger to fire every 30 seconds.

To create a new trigger right-click on the Triggers tree object and select New. This will open the trigger editor window as seen to the right.

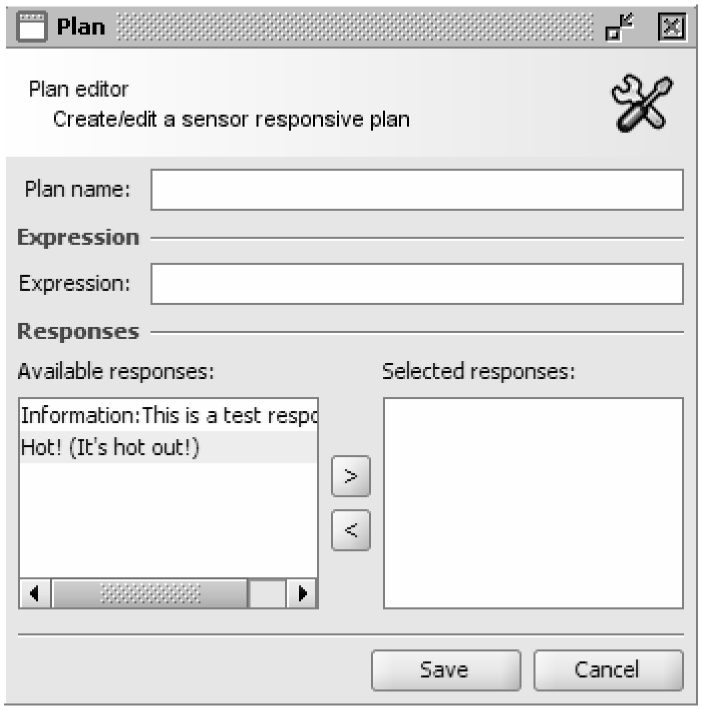
All triggers must have a unique trigger name. Then the user must select whether to create a simple trigger or a cron trigger. A simple trigger allows the user to define the start and end time of the trigger (or no end time), how many times the trigger should repeat (or to repeat indefinitely), and what the interval should be between trigger firings. The cron trigger allows the user to define a firing schedule using Unix like cron definitions. For example, this means being able to create a firing schedule such as: "At 8:00am every Monday through Friday" or "At 1:30am every last Friday of the month". Since the jHomeNet application uses the Quartz scheduling library for all the SRP scheduling the cron expression is simply passed to a Quartz CronTrigger object. The components of a valid cron expression are beyond the scope of this document. Refer to the Quartz documentation on CronTriggers for more information.

When finished creating the trigger, click on the Save button. This will permanently save the trigger in the jHomeNet database for later use.

Users may edit existing triggers by selecting the trigger tree object, right clicking on the particular trigger, and selecting the Edit option. This opens the trigger editor window which allows the user to edit some (but not all) of the trigger fields.

<u>**Section 6.1.5 – Plans**</u>

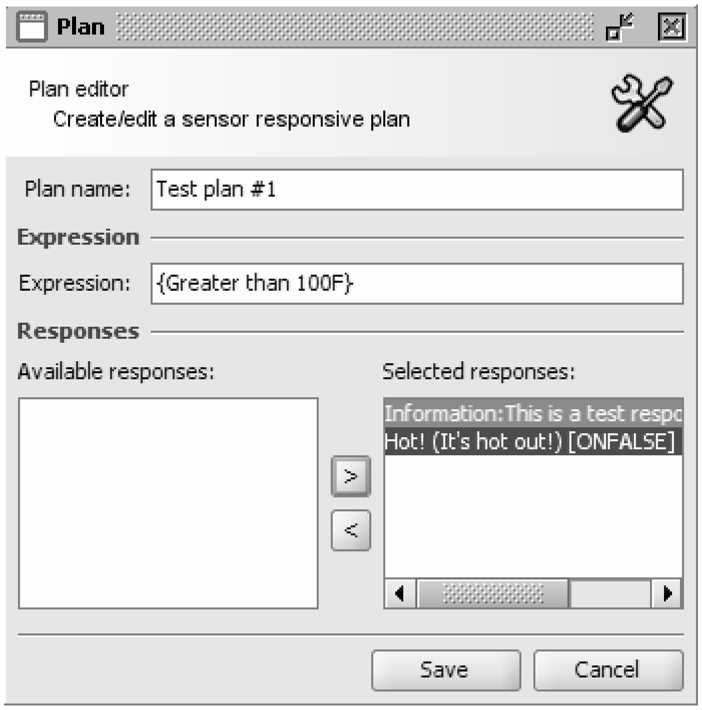
A SRP plan is the glue that pulls everything together. In particular, a SRP plan consists of an expression, a trigger, and a set of both on-true and on-false responses. When a plan created and scheduled with the SRP manager, every time the plan's trigger fires the plan's expression is evaluated. If the expression evaluates to `true` then all of the on-true responses are executed. If the expression evaluates to `false` then all of the on-false responses are executed.



To create a new plan, right-click on the Plans tree object and select New. This will open the plan editor window as seen to the left.

All plans must have a unique plan name. A user may create an expression but right-clicking on the expression textfield and may to choose to clear the current expression or insert expression elements (where an expression element can be a condition or a Boolean operator such as AND, OR, or NOT). A user may also choose to enter either an opening or closing parenthesis.

After creating the expression the user may also select the desired responses to be executed. In the left list are all the available responses. A user may select an available response and then click on the ">" button. The user will be prompted whether this should be designated as an on-true or on-false response. After making a choice the selected responses will now be in the right list color coded appropriately (green for on-true responses, red for on-false responses). At right is the same plan editor window with an on-true and an on-false response.



Plan name: Test plan #1

Expression: {Greater than 100F}

Information:This is a test respo

Hot! (It's hot out!) [ONFALSE]

When finished creating the plan, click on the Save button. This will permanently save the plan in the jHomeNet database for later use.

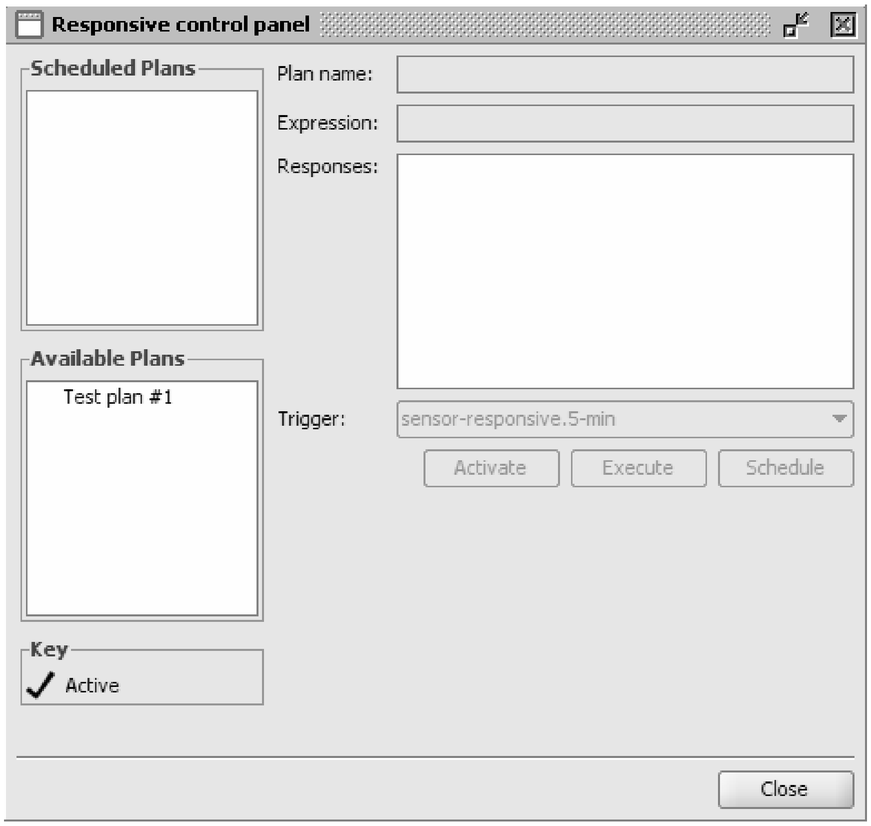
Users may edit existing plans by selecting the plan tree object, right clicking on the particular plan, and selecting the Edit option. This opens the plan editor window which allows the user to edit some (but not all) of the plan fields.

NOTE: Saving SRP plans in the jHomeNet database is not currently supported. Look for future releases to add this support.

Section 6.1.6 – SRP Control Panel

Now that we've created all of these SRP responses and triggers and conditions and plans, we need to be able to schedule and control the plans with the SRP manager. This is done using the SRP control panel. To open the control panel, select the Control Panel option from the Responsive drop down menu.

The control panel consists of two lists on the left: available SRP plans and currently scheduled SRP plans. To schedule a plan with SRP manager select a plan from the Available Plans list. This will automatically update the field on the right side of the window. (Note that none of these fields are editable. If you want to edit any of the plan's features close the control panel and open the respective editor.) Once a plan has been selected a user may then select the desired trigger to use to control when and how often the plan's expression is evaluated. A user may also activate or deactivate a plan. Note that by default when plans are created they are not set to active so a user must activate a plan otherwise its expression won't be evaluated when the respective trigger fires.



Once a plan has been selected and activated, a user may test the plan by clicking on the Execute button. This will cause the plan's expression to be evaluated and the corresponding on-true or on-false response to be executed as well. It's a way to test a plan without having to actually schedule the plan with the SRP manager.

If the plan has been tested it now may be scheduled with the SRP manager. Select the desired trigger from the drop down box and click on the Schedule button. Once this happens the plan will move from the Available Plans list to the Scheduled Plans list on the left side. The plan will also be scheduled with the SRP manager to be evaluated based on the selected trigger. Scheduled plans may be unscheduled by selecting the plan from the Scheduled Plans list and clicking on the Unschedule button. This will unschedule the plan with the SRP manager and move the plan from the Scheduled Plans list to the Available Plans list.

<u>Section 6.2 – Plug-ins</u>

The jHomeNet server has built-in plug-in support to enable third parties to add additional functionality through plug-ins. In particular, the latest release of the jHomeNet server software includes plug-in support to allow third parties to add additional CLI commands. Additional plug-in support may be added in future releases of the software. The purpose of this section is to provide a brief overview of plug-in development and installation.

Section 6.2.1 – Console Plug-ins

The built-in plug-in support is enabled through the Java Plugin Framework (JPF) open-source project.

To create a new command for the jHomeNet server console, create a new command class that implements the `jhomenet.server.console.command.Command` interface (refer to the API documents for further information). Alternatively, you may also subclass the `jhomenet.server.console.command.AbstractCommand` class to create a new command. The `AbstractCommand` class provides a few additional utility methods that are helpful while printing information to the console. You will need to include the jHomeNet server Jar file on your classpath in order to successfully compile the new command.

Once the new command has been developed and compiled, a new console command `plugin.xml` file needs to be developed in order to import the new command into the jHomeNet server plug-in manager. Included with the jHomeNet server distribution is a main console plug-in file, `plugin.xml`. Refer below for the included `plugin.xml` main console plug-in file.

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.5"
       "http://jpf.sourceforge.net/plugin_0_5.dtd">
<plugin id="jhomenet.server.plugin.console" version="1.0">
       <extension-point id="main-console-plugin">
              <parameter-def id="class" />
       </extension-point>
</plugin>
```

The main console plug-in ID is `jhomenet.server.plugin.console`, the plug-in version is `1.0`, and the plug-in extension point is `main-console-plugin`. These parameters will be required for any new console command plug-ins.

A sample new CLI command plug-in file is included below for a plug-in class called `jhomenet.addon.plugin.console.TestCommand`. The new `TestCommand` plug-in class extends the `AbstractCommand` class in the jHomeNet server distribution.

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.5"
       "http://jpf.sourceforge.net/plugin_0_5.dtd">
<plugin id="jhomenet.addon.plugin.console" version="1.0">
       <requires>
              <import plugin-id="jhomenet.server.plugin.console" />
       </requires>
       <runtime>
              <library id="test-console-plugin" path="classes/" type="code" />
       </runtime>
       <extension plugin-id="jhomenet.server.plugin.console" point-id="main-console-
plugin"
              id="test-console-plugin">
              <parameter id="class" value="jhomenet.addon.plugin.console.TestCommand"
/>
       </extension>
</plugin>
```

Once the new CLI command `plugin.xml` file has been developed and the class files built, put them in the jHomeNet `plugins` folder. In particular, create a new folder in the `plugins` folder that matches the extension plugin ID defined in the `plugin.xml` file. In the case of the example above, a `jhomenet.addon.plugin.console` folder was created. In that folder place the new console command `plugin.xml` file and create a new `classes` folder. In the `classes` folder place the new console command compiled Java class files. For an example, refer to the folder layout below.

```
<root jHomeNet server folder>
|-- conf
|-- data
|-- images
|-- jhomenet
|-- lib
|-- persistence
|-- plugins
        |-- jhomenet.addon.plugin.console
                |-- classes
                        |-- jhomenet
                                |-- addon
                                        |-- plugin
                                                |-- plugin
                                                        |-- console
                                                                |-- TestCommand.class
                |-- plugin.xml
        |-- jhomenet.server.plugin.console
                |-- plugin.xml
```

Section 7 – Development Environment

Section 7.1 – My Setup

My biggest hope is that others will find the jHomeNet project useful and will want to contribute to the project. This section explains the development environment that I currently use to develop the jHomeNet suite of applications.

The jHomeNet suite actually consists of several smaller sub-projects (or packages). At the time of this writing they include:

- jhomenet-commons
- jhomenet-ui
- jhomenet-server

These three sub-projects are required in order to run the jHomeNet suite. Lately I've been working on a remote client for the jHomeNet suite. It's in the jhomenet-client sub-project.

The first thing required in order to develop the jHomeNet application is a Java compiler. While the JRE is all you need to actually run the jHomeNet application you'll need a Java compiler to compile and build the project from source. I have been using the latest 1.6 release of the JDK from Sun Microsystems. You'll need a 1.6 compliant JDK in order to compile and build the project. You can download the latest JDK from http://java.sun.com.

I also use Apache Ant (http://ant.apache.org) to build the sub-projects. Each sub-project includes a `build.xml` build file. I've been using version 1.7.0 of Apache Ant.

I also have been using Eclipse as my IDE. The latest stable version that I have been using is version 3.3.0. You can download the latest version from http://www.eclipse.org. I haven't installed any special plug-ins other than an XML editor to help edit some of the XML configuration files but you can do this by simply using Wordpad or VI.

All the source code for the jHomeNet project is freely available from the Sourceforge website. In particular, I use Subversion as the software revision control system. I primarily use two Subversion clients: Tortoise SVN (http://tortoisesvn.tigris.org) which is a Windows Explorer client and Subclipse (http://subclipse.tigris.org) which is an Eclipse Subversion plug-in client. Both I have found work very well and I generally use them interchangeably.

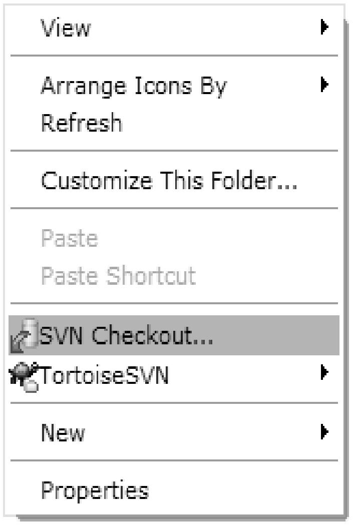
Once you've got your favorite Subversion client installed it's time to checkout the jHomeNet projects (actually, you'll be checking out as a minimum the three sub-projects listed above). Unless you have developer access the Subversion repository is read only but that won't stop us from getting started. First, start by creating a development directory. I usually like to have the following directory layout although your personal preferences will dictate your particular setup:

```
c:\dev\ref
c:\dev\workspace
```

I typically like to keep copies of interesting projects in my `ref` folder and keep all active projects (including the checked out source code for the jHomeNet sub-projects) in my `workspace` folder.

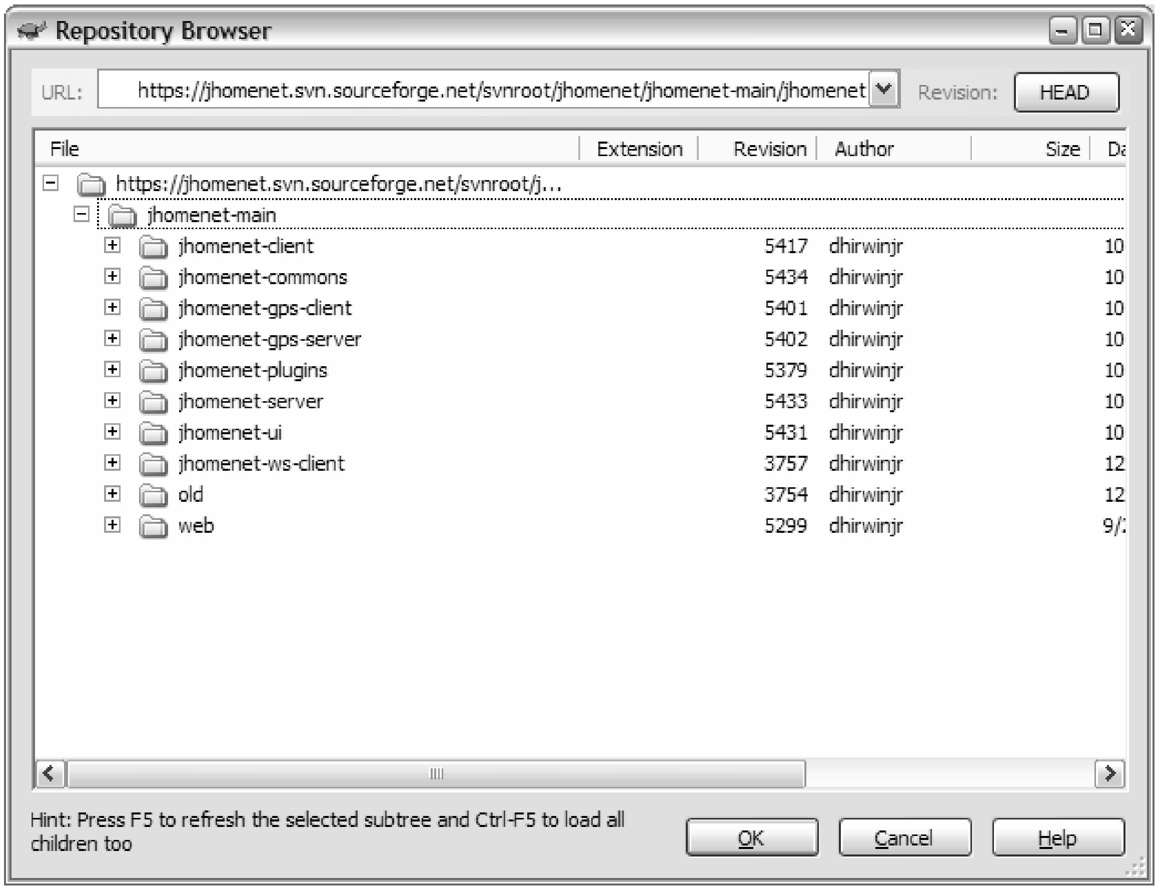
Once you've chosen your working directory it's time to checkout the actual jHomeNet source. As provided on the Sourceforge website, access to the main jHomeNet Subversion repository uses the following URL:

```
https://jhomenet.svn.sourceforge.net/svnroot/jhomenet jhomenet
```



We'll start by checking out the jhomenet-commons sub-project. For the purpose of this demonstration I'll be using the Tortoise SVN tools. In your working directory right-click and you should get a drop down window (see left). Select the "SVN Checkout…" option.

Next you'll be presented with a checkout window that allows you to enter the repository URL (see above) and the desired output folder (for example, C:\dev\workspace\jhomenet-commons). Enter the repository URL from above and then click on the "…" button next to the URL text field to choose the jHomeNet sub-projects. Below is a screen shot of the current list of available sub-projects. As you can see there are several sub-projects beyond those listed above but for the time being we'll only be concerned with those three.


Repository Browser

URL: https://jhomenet.svn.sourceforge.net/svnroot/jhomenet/jhomenet-main/jhomenet Revision: HEAD

File	Extension	Revision	Author	Size	Da
https://jhomenet.svn.sourceforge.net/svnroot/j...					
jhomenet-main					
jhomenet-client		5417	dhirwinjr		10
jhomenet-commons		5434	dhirwinjr		10
jhomenet-gps-client		5401	dhirwinjr		10
jhomenet-gps-server		5402	dhirwinjr		10
jhomenet-plugins		5379	dhirwinjr		10
jhomenet-server		5433	dhirwinjr		10
jhomenet-ui		5431	dhirwinjr		10
jhomenet-ws-client		3757	dhirwinjr		12
old		3754	dhirwinjr		12
web		5299	dhirwinjr		9/:

Hint: Press F5 to refresh the selected subtree and Ctrl-F5 to load all children too

Select the jhomenet-commons and expand the folder. You'll see three sub-folder: branches, tags, and trunk. Click on the trunk folder and then click the Ok button. In total the jhomenet-commons repository URL should now be set to:

```
https://jhomenet.svn.sourceforge.net/svnroot/jhomenet/jhomenet-
main/jhomenet-commons/trunk
```

Next click on the Ok button to actually checkout the source code. You may be prompted that the folder doesn't already exist. Click on the Yes button to continue. The SVN client now takes care of checking out all of the source code, images, and build files required to build the jhomenet-commons sub-project. Depending on your network connection it may take a few minutes as there's about 9.3 MB of data to download. Once it's finished click on the Ok button. Now you're free to explore the contents of the jhomenet-package sub-project.

But checking out the jhomemet-commons sub-project really won't get you very far as it's only a piece to the larger puzzle. You'll also need to check out the jhomenet-ui and the jhomenet-server sub-projects by following similar steps that I just outlined.

Once you have all three sub-projects checked out onto your workstation you're now ready to begin developing. To start the Ant `build.xml` file that's included with the jhomenet-server sub-package actually can take care of building all three sub-projects using a single call. This is quite helpful because you don't need to go into each sub-project and compile the projects separately, then copy the newly built Jar file into one or more of the dependent sub-projects and then build that sub-project and so on (remember, the jhomenet-ui sub-project depends on the jhomenet-commons package, and the jhomenet-server package depends on both the jhomenet-commons package and the jhomenet-ui package). However, in order to do this you need to edit the included `build.properties` file in the jhomenet-server root folder. In particular, open the file in your favorite text editor and scroll down to the `jHomeNet suite full-build dependency properties` section towards the bottom of the file. You'll need to edit the `commons-package.dir` and `ui-package.dir` values to point to your actual jhomenet-commons and jhomenet-ui sub-project locations.

Depending on whether or not you've also checked out the jhomenet-client package you'll also want to set the `client-package.build` value to either `true` or `false` (and the `client-package.dir` values). This controls whether the jhomenet-client package is built. If this is set to `true` and you don't have the jhomenet-client package checked out the build will fail. Once you've made these changes you should now be ready to begin compiling.

You can compile the full jHomeNet suite by typing at the command line in the jhomenet-server root directory:

```
D:\workspace\jhomenet-server>ant build-suite
```

This of course assumes that you have Apache Ant properly installed and configured. This Ant call will build the jhomenet-commons, jhomenet-ui, and the jhomenet-server sub-projects. It will also create a Windows batch (`server_dev.bat`) file that you can use to run the newly compiled source files. Once the compilation is complete and now errors were reported you can start the development server by type `server_dev` at the command line.

And that should be it for getting you up and developing.

Section 7.2 – Other Issues

There are times when you may want to clear away all of the previously built Java class files and start from scratch. Fortunately the jhomenet-server sub-project's `build.xml` file includes a task that does just this. At the command line you can run the following:

```
D:\workspace\jhomenet-server>ant build-clean-suite
```

This first cleans all of the build folders for the three sub-projects and then builds the full jHomNet suite from scratch.

If you’re getting errors during compiling and it looks like either the jhomenet-commons or jhomenet-ui Jar files don’t contain the correct API methods, it’s possible that an older copy of the jhomenet-commons or jhomenet-ui Jar file is in the jhomenet-server sub-project’s `lib` folder. This may especially happen between releases when a sub-project’s version is incremented and the sub-project’s Jar file changes (for example, from `jhomenet-commons-0.5.2.jar` to `jhomenet-commons-0.5.3.jar`). Delete all the jhomenet-commons and jhomenet-ui jar files and then run the `build.xml` file again. This should fix any compilation issues.

Section 8 – FAQs

Section 8.1 –FAQs

When either compiling or running the application, because it's still under active development, errors may occur. Below is a list of possible errors a user might receive.

Compiling and initializing database
The following error may be received while compiling the source and generating the necessary database using the `schemaexport` Ant task.

```
Schema text failed: Error reading resource: com/foo/SomeFile.hbm.xml
```

This may mean that not all the properties defined in the `SomeFile.hbm.xml` file are present in the actual Java file. For example, if the primary key ID is defined in the hbm file but not in the Java file (along with the appropriate getter/setter methods), then this error will result.

Section 9 – Sample Configuration Files

This section contains some sample configuration files. The most up-to-date copies are found in the respective Subversion repository.

Section 9.1 – server.cfg.xml

Sample `server.cfg.xml` configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================== -->
<!-- jHomenet server configuration file.                            -->
<!-- ================================================== -->
<jhomenet-config>

        <!-- ============================================================= -->
        <!-- Startup configuration information.                            -->
        <!--                                                               -->
        <!-- Known nodes include:                                          -->
        <!--    console        : defines status of console view at startup  -->
        <!-- Known attributes include:                                     -->
        <!--    display        : define the type                           -->
        <!-- Known display values include:                                 -->
        <!--    true/false     : true for console startup, false for GUI    -->
        <!-- ============================================================= -->
        <startup-config>
                <console display="false" />
                <hardware-polling start="false" />
        </startup-config>

        <!-- ============================================================= -->
        <!-- Hardware configuration definition. If nothing is defined, it   -->
        <!-- defaults to an XML configuration using hardware.cfg.xml as     -->
        <!-- the hardware configuration filename.                          -->
        <!--                                                               -->
        <!-- Known attributes include:                                     -->
        <!--    type           : define the type (see below)               -->
        <!--    filename       : for XML or TXT based configurations only   -->
        <!-- Known hardware configuration types include:                   -->
        <!--    XML                    : For XML based hardware configuration-->
        <!--    TXT                    : For TXT based hardware configuration-->
        <!--                             (not yet developed)               -->
        <!--    hibernate      : For Hibernate based hardware configuration  -->
        <!--                             (not yet developed)               -->
        <!-- ============================================================= -->
        <hardware-config type="XML" filename="hardware_test.cfg.xml" />

        <!-- ================================================== -->
        <!-- User authentication configuration                      -->
        <!-- ================================================== -->
        <authentication-config>
                <type>simple</type>
        </authentication-config>

        <!-- ================================================== -->
        <!-- Network configuration                          -->
        <!-- ================================================== -->
        <network-config>
                <firewall-enabled>false</firewall-enabled>
        </network-config>

        <!-- ================================================== -->
        <!-- Work queue configuration                        -->
        <!-- ================================================== -->
        <workqueue-config>
                <threadsize>5</threadsize>
        </workqueue-config>
```

```
</jhomenet-config>
```

Section 9.2 – hardware.cfg.xml

Sample `hardware.cfg.xml` configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
        jHomenet server hardware configuration file

        Use this file to define and configure jHomenet server hardware properties.
        This includes:

        1) Hardware container manager definitions
        o hardware container manager configuration filename (optional)
        2) Hardware definitions including the following properties:
        o hardware classname (required)
        o description (optional)
        o icon filename (optional)
        o compatible driver hardware (optional)

        This file is parsed at the server start and any hardware not present in this
        configuration file will not be available to the server.

        Id: $Id: hardware.cfg.xml 1078 2005-12-21 22:58:45Z dhirwinjr $
-->

<hardware-configuration>

        <!-- ======================================================== -->
        <!-- Define the available hardware container loaders          -->
        <!-- classnames.                                             -->
        <!-- ======================================================== -->
        <hardware-container-loader
               classname="jhomenet.server.hw.driver.onewire.OneWireContainerLoader" />
        <hardware-container-loader
               classname="jhomenet.server.hw.driver.X10.X10ContainerLoader" />

        <!-- ============================================================== -->
        <!-- Define the hardware persistence layer classname                       -->
        <!--                                                            -->
        <!-- Known hardware persistence types include:                            -->
        <!--    XML                   : For XML based hardware persistence         -->
        <!--    hibernate      : For Hibernate based hardware persistence          -->
        <!-- ============================================================== -->
        <hardware-persistence-layer type="hibernate" />

        <!-- ============================================================== -->
        <!-- Define the hardware data persistence layer.                -->
        <!--                                                            -->
        <!-- Known attributes include:                                  -->
        <!--    type           : define the type (see below)            -->
        <!--    filename       : for XML or TXT based configurations only   -->
        <!-- Known hardware data persistence types include:                       -->
        <!--    XML                   : For XML based hardware data persistence    -->
        <!--    TXT                   : For text based hardware data persistence   -->
        <!--    hibernate      : For Hibernate based hardware data persistence     -->
        <!-- ============================================================== -->
        <data-persistence-layer type="hibernate" />

        <!-- ======================================================== -->
        <!-- Sensor hardware definitions                              -->
        <!-- ======================================================== -->

        <!-- Door sensor -->
        <!--
               <hardware
```

```
                classname="jhomenet.server.hw.sensor.DoorSensor"
                description="Door sensor"
                icon-filename="contact.png"
                compatible-driver-hardware="DS2406">
                </hardware>
        -->

        <!-- Humidity sensor -->
        <!--
                <hardware
                classname="jhomenet.server.hw.sensor.HumiditySensor"
                description="Humidity sensor"
                icon-filename=""
                compatible-driver-hardware="">
                </hardware>
        -->

        <!-- Light sensor -->
        <hardware classname="jhomenet.server.hw.sensor.LightSensor"
                description="Light sensor" >
                <compatible-driver-hardware type="DS2438"
                        driver-
classname="jhomenet.server.hw.driver.onewire.OneWireVoltageDriver">
                </compatible-driver-hardware>
        </hardware>

        <!-- Lightning sensor -->
        <hardware classname="jhomenet.server.hw.sensor.LightningSensor"
                description="Lightning sensor" >
                <compatible-driver-hardware type="DS2423"
                        driver-
classname="jhomenet.server.hw.driver.onewire.OneWireCountDriver">
                </compatible-driver-hardware>
        </hardware>

        <!-- Barometric pressure sensor -->
        <!--
                <hardware
                classname="jhomenet.server.hw.sensor.BarometricSensor"
                description="Barometric pressure sensor"
                icon-filename=""
                compatible-driver-hardware="">
                </hardware>
        -->

        <!-- Rain sensor -->
        <!--
                <hardware
                classname="jhomenet.server.hw.sensor.RainSensor"
                description="Rain sensor"
                icon-filename=""
                compatible-driver-hardware="DS2423">
                </hardware>
        -->

        <!-- Temperature sensor -->
        <hardware classname="jhomenet.server.hw.sensor.TempSensor"
                description="Temperature sensor" >
                <compatible-driver-hardware type="DS1822"
                        driver-
classname="jhomenet.server.hw.driver.onewire.OneWireTempDriver">
                </compatible-driver-hardware>
                <compatible-driver-hardware type="DS18B20"
                        driver-
classname="jhomenet.server.hw.driver.onewire.OneWireTempDriver">
                </compatible-driver-hardware>
                <compatible-driver-hardware type="DS18S20"
                        driver-
classname="jhomenet.server.hw.driver.onewire.OneWireTempDriver">
                </compatible-driver-hardware>
                <compatible-driver-hardware type="DS1920"
```

```
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireTempDriver">
               </compatible-driver-hardware>
       </hardware>

       <!-- Wind direction sensor -->
       <hardware classname="jhomenet.server.hw.sensor.WindDirectionSensor"
               description="Wind direction sensor">
               <compatible-driver-hardware type="DS2450"
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireWindDirectionDriver">
               </compatible-driver-hardware>
       </hardware>

       <!-- Wind speed sensor -->
       <hardware classname="jhomenet.server.hw.sensor.WindSpeedSensor"
               description="Wind speed sensor" >
               <compatible-driver-hardware type="DS2423"
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireWindSpeedDriver">
               </compatible-driver-hardware>
       </hardware>

       <!-- HVAC monitor sensor -->
       <hardware classname="jhomenet.server.hw.sensor.HvacSensor"
               description="HVAC sensor" >
               <compatible-driver-hardware type="DS2450"
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireADDriver">
               </compatible-driver-hardware>
       </hardware>

       <!-- ====================================================== -->
       <!-- Device hardware definitions                            -->
       <!-- ====================================================== -->

       <!-- Switch device -->
       <hardware classname="jhomenet.commons.hw.device.SwitchDevice"
               description="Light switch device" >
               <compatible-driver-hardware type="SWITCH"
                      driver-classname="jhomenet.server.hw.driver.X10.X10SwitchDriver">
               </compatible-driver-hardware>
               <compatible-driver-hardware type="DS2405"
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireSwitchDriver">
               </compatible-driver-hardware>
               <compatible-driver-hardware type="DS2406"
                      driver-
classname="jhomenet.server.hw.driver.onewire.OneWireTwoChannelSwitchDriver">
               </compatible-driver-hardware>
       </hardware>

</hardware-configuration>
```

Section 9.3 – x10.conf

Sample X10.conf configuration file:

```
#----------------------------------------------------------------------
# X10 configuration file
# Written by: David Irwin (jhomenet at gmail dot com)
# Last updated: July 23, 2007
#----------------------------------------------------------------------

#----------------------------------------------------------------------
# Define X-10 network configuration details.
#----------------------------------------------------------------------
port = COM1
```

Section 9.4 – x10-containers.cfg.txtf

Sample X10 containers configuration file:

```
#----------------------------------------------------------------------
# X10 container configuration file
# Written by: David Irwin (jhomenet at gmail dot com)
# Last updated: July 23, 2007
#----------------------------------------------------------------------

#----------------------------------------------------------------------
# Define the X-10 hardware. Because the current implementation
# of the X-10 network doesn't support scanning for hardware
# on startup, the hardware must be known before hand. This file
# serves as the hardware setup.
#
# Each hardware object should be defined on a single line in this
# configuration file. The exact format is as follows:
#       hardware-<identifier> = <hardware address>:<hardware type>
# where
#       <identifier>:         this can be any unique identifier; this is
#                                used internally so it does not need to
#                                correspond to anything in particular
#       <hardware address>:      this is the hardware address of the X10
#                                device in (<house code>,<device code>) format
#                                (e.g. A,1)
#       <hardware type>:         this determines the hardware type; current
#                                options include SWITCH. See the server
#                                documentation for more information.
#
# See the JavaDocs for the different types of hardware.
#----------------------------------------------------------------------
hardware-001 = A,1:SWITCH
```

Section 9.5 – sensor-responsive.cfg.xml

Sample sensor responsive configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ================================================== -->
<!-- jHomenet sensor responsive configuration file.         -->
<!-- ================================================== -->
<responsive-config>
        <!-- The persistence type -->
        <property name="persistence.type">hibernate</property>

        <!-- Email configuration -->
        <property name="email.name">Dave Irwin</property>
        <property name="email.hostname">smtp.sbcglobal.yahoo.com</property>
        <property name="email.username">dhirwin@sbcglobal.net"</property>
        <property name="email.sender">dhirwin@sbcglobal.net"</property>
        <property name="email.replyto">dhirwin@sbcglobal.net</property>
</responsive-config>
```

Section 9.6 – hibernate.cfg.xml

Sample Hibernate configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
               "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
               "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>

         <!-- Settings for a MySQL database. -->
```

```
               <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
               <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
               <property
name="hibernate.connection.url">jdbc:mysql://localhost/jhomenet_db</property>
               <!--
               <property name="hibernate.connection.username">root</property>
               <property name="hibernate.connection.password">ek</property>
               -->
               <property name="hibernate.connection.username">root</property>
               <property name="hibernate.connection.password">xtnm21a</property>

               <!-- Use the Hibernate built-in pool for tests. -->
        <property name="connection.pool_size">3</property>

        <!-- Enable auto-commit mode for special cases (integration testing) -->
        <!-- <property name="connection.autocommit">true</property> -->

        <!-- Disable the second-level cache  -->
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <property name="cache.use_query_cache">false</property>
        <property name="cache.use_minimal_puts">false</property>

        <!-- In eager fetching, only join three tables deep if joins are used -->
        <property name="max_fetch_depth">3</property>

        <!-- Print SQL to stdout, format it nicely  -->
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="use_sql_comments">true</property>

        <!-- Drop and then re-create schema on SessionFactory build, for testing  -->
        <!-- <property name="hbm2ddl.auto">create</property> -->

        <!-- Use thread-bound persistence context propagation, scoped to the transaction
-->
        <property name="current_session_context_class">thread</property>

               <!-- ===================================================== -->
               <!-- jHomeNet mapping files                                -->
               <!-- ===================================================== -->

               <!-- User type definitions (these need to be before all other mappings) --
>
               <mapping
resource="jhomenet/commons/persistence/hibernate/UserTypes.hbm.xml" />

               <!-- jHomeNet configuration related mapping files -->
               <mapping resource="jhomenet/server/ServerDatabaseVersion.hbm.xml" />
               <mapping
resource="jhomenet/commons/weather/WeatherGatewayConnectionInfo.hbm.xml" />
               <mapping resource="jhomenet/commons/auth/User.hbm.xml" />
               <mapping resource="jhomenet/commons/event/Event.hbm.xml" />

               <!-- jHomeNet hardware related mapping files -->
               <mapping resource="jhomenet/server/hw/Hardware.hbm.xml" />
               <mapping resource="jhomenet/commons/hw/data/AbstractHardwareData.hbm.xml"
/>

               <!-- jHomeNet sensor response related mapping files -->
               <mapping
resource="jhomenet/commons/responsive/condition/Condition.hbm.xml" />
               <mapping resource="jhomenet/commons/responsive/response/Response.hbm.xml"
/>
               <mapping
resource="jhomenet/commons/responsive/trigger/TriggerWrapper.hbm.xml" />

       </session-factory>
</hibernate-configuration>
```

Resources & Links

Java

[http://java.sun.com] – Official home of Java. Free downloads of Sun Microsystem's JRE and JDK.

[http://swinglabs.org/index.jsp] – SwingLabs provides an excellent source of Java UI tools incuding the SwingX library heavily used in the jHomeNet UI library.

1-Wire information

[http://www.maxim-ic.com/1-Wire.cfm] – Official home of 1-Wire network. Good resource for 1-wire hardware datasheets and application notes.

[http://www.1wire.org] – Excellent resource for 1-Wire information including installation suggestions and details.

Development Tools

[http://www.eclipse.org] – The Eclipse IDE used to assist in most of the software development.

[http://www.hibernate.org] – The Hibernate O/RM persistence tool.

[http://jpf.sourceforge.net] – The Java Plugin Framework open source project website.

Appendix A – jHomenet Ant build

Build configuration

Most of the jHomenet build parameters are not actually set in the `build.xml` file. Instead, most build parameters are set in the included `default.properties` file. Here the user can define the project name, version information, and all the project folders including all the build and distribution filenames and folders.

Ant tasks

A few of the defined Ant tasks require additional Jar files not included in the default Ant distribution.

To run any of the defined Ant tasks, from the root jHomenet directory type at a command prompt:

```
ant [task name]
```

The default task is `build`. In most instances when developing the server, a user should only have to type `ant` at the command line.

A description of the currently defined tasks are listed below.

prepare
Creates the necessary directories.

clean-classes
This task deletes all the files and folders in the build/classes and dist folders.

init-build
This task is used to initialize the build script including creating all the necessary build folders and copying the library and configuration files. This task must be called prior to most other defined tasks.

build
This is the default task and is responsible for building the source code and putting the compiled class files in the `/build/classes` folder.

This task depends on the `prepare` and `init-build` tasks. This is the default task.

build-clean

Same as the build task except that it deletes all the existing files in the `/build/classes` file (including class and any configuration files). Note: the actual configuration files are stored in the `/resources/conf` folder.

This task depends on the `prepare`, `clean-classes`, and `build` task.

init-db
This task is used to create the database using Hibernate's schema export tool (dmb2dll). This requires that the Hibernate3.jar file is located in the `/resources/lib` directory. The Hibernate Jar file is included with the jHomenet distribution.

This task depends on the `build` task.

jar
This task creates a distribution folder (`dist`) and creates two Jar files in this folder. One Jar file contains all the compiled class files and the second Jar file contains all the necessary library files.

This task depends on the `clean-all` and `build` tasks.

javadocs
This task calls the java-docs tool to create the Java documentation from the source files. The files are created in the `/docs/api` folder.

This task depends on the `prepare` task.

package
This task creates the distribution folder and copies all the necessary components for an official distribution. This includes creating and copying the API documentation, copying all the source files, copying all the configuration and image files, and copy all the necessary library files.

This task depends on the `clean-all`, `build`, and `javadocs` tasks.

package-zip
This task zips the contents of the distribution folder (`/dist/jhomenet-server-[versionid]`) and creates a Zip file (`/dist/jhomenet-server-[versionid].zip`).

This task depends on the `package` task.

Appendix B – Acronyms & Other Information

Acronyms

API	Application Programming Interface
CLI	Command Line Interface
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
IDE	Integrated Development Environment
JDK	Java Development Kit
JPF	Java Plugin Framework
JRE	Java Runtime Environment
O/RM	Object/Relational Mapping
POJO	Plain Old Java Object
SDK	Software Development Kit
SRP	Sample Responsive Programming