Abstract

Describes the implementation of hwscan3, a system for displaying a report showing the hardware configurations of publicly available servers and client workstations at the New Mexico Tech Computer Center.

This publication is available in Web form\(^1\) and also as a PDF document\(^2\). Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. Overview ................................................................................................................................ 3
2. Installation .............................................................................................................................. 4
3. Generated XHTML .................................................................................................................. 5
   3.1. XHTML for the start page ............................................................................................... 5
   3.2. XHTML for the room page ............................................................................................. 6
4. Prologue ................................................................................................................................. 8
5. Imports ................................................................................................................................... 8
6. Manifest constants ................................................................................................................... 9
   6.1. HTML_SUFFIX ............................................................................................................... 9
   6.2. TCC_URL ...................................................................................................................... 10
   6.3. TCC_PATH .................................................................................................................... 10
   6.4. BASE_DIR .................................................................................................................... 10
   6.5. BASE_URL .................................................................................................................... 10
   6.6. BASE_PATH ................................................................................................................... 10
   6.7. START_FILE .................................................................................................................. 10
   6.8. START_URL ................................................................................................................... 10
   6.9. START_PATH .................................................................................................................. 11
   6.10. CSS_URL ...................................................................................................................... 11
   6.11. TCC_HOME .................................................................................................................. 11
   6.12. LDAP_URL ................................................................................................................... 11
   6.13. CLIENTS_DN ............................................................................................................... 11
   6.14. NULL_FILTER ............................................................................................................. 11
   6.15. LDAP_TRUE .................................................................................................................. 11
   6.16. CN_ATTR ...................................................................................................................... 12
   6.17. OFFICE_ATTR ............................................................................................................. 12

\(^1\) http://www.nmt.edu/tcc/hw/hwscan3/ims3/
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.18.</td>
<td>OS_ATTR ................................................................. 12</td>
</tr>
<tr>
<td>6.19.</td>
<td>ATTR_LIST ........................................................................ 12</td>
</tr>
<tr>
<td>6.20.</td>
<td>FORGE_SERVER .................................................................. 12</td>
</tr>
<tr>
<td>6.21.</td>
<td>REPORT_FILE ..................................................................... 12</td>
</tr>
<tr>
<td>6.22.</td>
<td>TABLE_ATTRS ..................................................................... 12</td>
</tr>
<tr>
<td>6.23.</td>
<td>L_ALIGN ............................................................................ 13</td>
</tr>
<tr>
<td>6.24.</td>
<td>R_ALIGN ............................................................................ 13</td>
</tr>
<tr>
<td>6.25.</td>
<td>TOP_VALIGN ....................................................................... 13</td>
</tr>
<tr>
<td>6.26.</td>
<td>BOT_VALIGN ....................................................................... 13</td>
</tr>
<tr>
<td>6.27.</td>
<td>CLASS ................................................................................ 13</td>
</tr>
<tr>
<td>6.28.</td>
<td>HANG_CLASS ........................................................................ 13</td>
</tr>
<tr>
<td>6.29.</td>
<td>INST_SUFFIx ......................................................................... 14</td>
</tr>
<tr>
<td>7.</td>
<td>main(): The main program .................................................. 14</td>
</tr>
<tr>
<td>8.</td>
<td>buildPages(): Build all Web pages ........................................ 15</td>
</tr>
<tr>
<td>9.</td>
<td>buildServerTable(): Build the table of servers ......................... 16</td>
</tr>
<tr>
<td>10.</td>
<td>findLoginServers(): Display login server rows .......................... 17</td>
</tr>
<tr>
<td>11.</td>
<td>buildServerRow(): Format one row of the server table .................. 18</td>
</tr>
<tr>
<td>12.</td>
<td>buildOpenLabsList(): Bullet list of open labs ............................ 19</td>
</tr>
<tr>
<td>13.</td>
<td>buildOpenLab(): Build the page for an open lab ....................... 21</td>
</tr>
<tr>
<td>14.</td>
<td>linkToRoomPage(): Build a hyperlink to the room page ............... 21</td>
</tr>
<tr>
<td>15.</td>
<td>roomPageURL(): Generate the URL for a room page .................... 22</td>
</tr>
<tr>
<td>16.</td>
<td>roomPagePath(): Absolute path name of a room page .................. 22</td>
</tr>
<tr>
<td>17.</td>
<td>buildRoomPage(): Make the page with a table of all the clients in a room .................. 23</td>
</tr>
<tr>
<td>18.</td>
<td>buildRoomTable(): Build the table of client configurations .......... 24</td>
</tr>
<tr>
<td>19.</td>
<td>buildRoomRow(): Build one row of the room table ..................... 25</td>
</tr>
<tr>
<td>20.</td>
<td>buildDevices(): Extract relevant device configurations .............. 26</td>
</tr>
<tr>
<td>21.</td>
<td>buildClassroomTable(): Build the table of classrooms ................ 27</td>
</tr>
<tr>
<td>22.</td>
<td>buildClassroom(): Build one classroom table line and page .......... 29</td>
</tr>
<tr>
<td>23.</td>
<td>buildClassroomRow(): Build one line of the classroom table ......... 30</td>
</tr>
<tr>
<td>24</td>
<td>class Singleton: Parent class for singletons .......................... 31</td>
</tr>
<tr>
<td>25.</td>
<td>class ClientSet: Container for client configurations .................. 31</td>
</tr>
<tr>
<td>25.1.</td>
<td>ClientSet.genClients(): Generate all client configurations ........... 32</td>
</tr>
<tr>
<td>25.2.</td>
<td>ClientSet.lookupClient(): Find a specific client by host name .......... 32</td>
</tr>
<tr>
<td>25.3.</td>
<td>ClientSet.init(): Constructor ........................................... 32</td>
</tr>
<tr>
<td>25.4.</td>
<td>ClientSet.__buildLdapMap(): Extract LDAP's client list .............. 33</td>
</tr>
<tr>
<td>25.5.</td>
<td>ClientSet.__processLdapEntry(): Process one LDAP client record .......... 35</td>
</tr>
<tr>
<td>25.6.</td>
<td>ClientSet.__buildClientMap(): Extract configuration data .......... 36</td>
</tr>
<tr>
<td>25.7.</td>
<td>ClientSet.__getConfig(): Look up a system in the GetHardware service .......... 37</td>
</tr>
<tr>
<td>26.</td>
<td>class ClientConfig: Configuration data for one client ................ 38</td>
</tr>
<tr>
<td>26.1.</td>
<td>ClientConfig.<strong>init</strong>(): Constructor .................................. 39</td>
</tr>
<tr>
<td>26.2.</td>
<td>ClientConfig.__deviceCheck(): Check for devices of a given type .... 39</td>
</tr>
<tr>
<td>27.</td>
<td>class ReportInfo: Reading report information ............................ 40</td>
</tr>
<tr>
<td>27.1.</td>
<td>ReportInfo.<strong>init</strong>(): class constructor ................................ 41</td>
</tr>
<tr>
<td>27.2.</td>
<td>ReportInfo.__buildRooms(): Build the room dictionary ............... 42</td>
</tr>
<tr>
<td>27.3.</td>
<td>ReportInfo.__buildSystems(): Build the system map ................. 43</td>
</tr>
<tr>
<td>27.4.</td>
<td>ReportInfo.__makeSystem(): Build a system map entry ............... 44</td>
</tr>
<tr>
<td>28.</td>
<td>class Room: Room information ........................................... 44</td>
</tr>
<tr>
<td>29.</td>
<td>class System: System information ......................................... 45</td>
</tr>
<tr>
<td>30.</td>
<td>Epilogue .......................................................................... 46</td>
</tr>
</tbody>
</table>
1. Overview

This document describes the implementation of the system described in hwscan3 3.0. It includes the entire source code in lightweight literate programming style.

This script is run as a cron job once a day. It generates two kinds of pages:

- The start page contains a table of servers, a bullet list of open labs, and a table of classrooms.
- For each classroom and open lab, it generates a room page with a table showing the hardware configuration of each client in that room as a table.

In general, the execution proceeds in two phases:

1. Extract all the input information from three sources: LDAP, the XML-RPC GetHardware interface, and the scanreport.xml file.
   - TCC’s ldap0 server will provide us with a complete list of all the hosts (servers and clients). For particulars, see LDAP attribute cross-reference.
   
   Note that servers such as login are considered clients by LDAP.
   - The XML-RPC interface GetHardware is the primary source for hardware configuration data. See the TCC internal Wiki under Software/HomeGrown/Forge/ForgeWeb/GetHardware.
   
   Any client in LDAP that is an office machine (tccOfficeMachine is true), or is not known to GetHardware, such as a Macintosh, will not appear in the report.
   - The scanreport.xml file characterizes rooms, and explicitly lists the rooms, services, and peripheral devices of interest.

2. Generate the start page. While generating the start page content for each open lab and classroom, generate the room page for that room.

Before considering what classes we need, let’s look at the order in which we need to access the data.

- In the server table on the start page, we want to present the servers in alphabetical order by server name.
- In the open lab bullet list and also in the classroom table, the order of presentation is alphabetical by room name.
- In a room page, the table row for each client lists the peripheral devices in the same order as the entries in the scanreport.xml file’s deviceList element.

These needs suggest that there be two top-level class instances to represent all the input sources:

- An single instance of a class named ClientSet will hold all the information derived from LDAP and GetHardware. See Section 25, “class ClientSet: Container for client configurations” (p. 31).
  
  This instance will be a container for instances of a class named ClientConfig, each of which describes one client (including servers).
- A single instance of class ReportInfo represents the scanreport.xml file. It can generate a list of rooms in alphabetical order. It can generate a list of servers in alphabetical order. It can also generate

---

a list of device types in the same order in which they appear in the \texttt{scanreport.xml} file. See Section 27, “\texttt{class ReportInfo}: Reading report information” (p. 40).

Hence, the two main phases of execution become:

- Instantiate the \texttt{ClientSet} object. Its constructor will query LDAP and \texttt{GetHardware}.
- Generate the start page using this instance, generating room pages as their links are placed onto the start page.

2. Installation

- Make sure that two constants correctly identify the URL and absolute path name of all the files on our Web server.
  - Section 6.2, “\texttt{TCC\_URL}” (p. 10): This is the common prefix of all Web pages served from the TCC server, whose real name is \texttt{infohost.nmt.edu}.
  - Section 6.3, “\texttt{TCC\_PATH}” (p. 10): The absolute path name to the directory corresponding to \texttt{TCC\_URL}. Because the script generates Web pages, it assumes that a web page at some path \texttt{TCC\_PATH + P} corresponds to URL \texttt{TCC\_URL + P}.

- The start page generated by this script is not a standalone page. It is included using a server-side include from the page at \url{http://www.nmt.edu/tcc/hw/homepage.html}. File \texttt{homepage.g} in that directory, the \texttt{PyStyler} \textsuperscript{7} input file that generates the \texttt{homepage.html} file, contains this SSI include:

  \begin{verbatim}
  <!--#include virtual="hwscan.html"-->
  \end{verbatim}

  Therefore, \texttt{hwscan.html} in that directory must be a soft link to the actual start page. To create this link:

  \begin{verbatim}
  cd /u/www/docs/tcc/hw
  ln -l hwscan3/ims3/start/start.html hwscan.html
  \end{verbatim}

- To execute the script once a day, establish a \texttt{cron} job.

  Pick a server that has the \texttt{lxml} package installed, such as \texttt{login}. On that server, edit your \texttt{crontab} by issuing this shell command:

  \begin{verbatim}
  crontab -e
  \end{verbatim}

  In the edit screen that appears, add a line to run the script daily. This example runs the job at 02:30 local time.

  \begin{verbatim}
  30 2 * * * nice /u/www/docs/tcc/hw/hwscan3/ims3/hwscan3.py
  \end{verbatim}

  Because the script writes its name and version to \texttt{sys\_stderr}, the owner of the script should receive an e-mail every day with the output of the script. This will include any error messages about machines that are in LDAP but not known to \texttt{GetHardware()}.

\footnote{\url{http://infohost.nmt.edu/tcc/projects/pystyler3/}}
3. Generated XHTML

Here is the actual XHTML to be generated.

- Section 3.1, “XHTML for the start page” (p. 5)
- Section 3.2, “XHTML for the room page” (p. 6)

3.1. XHTML for the start page

The start page we generate will not be a standalone page. It will be an XHTML fragment that is included via a server-side include from a page that is part of the regular TCC PyStyler structure at http://www.nmt.edu/tcc.

Hence, what we generate will be appropriate child content for the body element. We’ll use a div to package up everything generated here.

Here is the actual XHTML framework. Following a heading, it has three main sections: the table of login servers; the bullet list of open labs; and the table of classrooms.

```xml
<div>
  <h1>TCC server and workstation inventory</h1>
  <h2>Linux servers for remote login</h2>
  <!--Table of login servers-->
  <table rules='all' border='8' cellpadding='10' cellspacing='10'>
    <col valign='top' align='left'/> <!--name-->
    <col valign='top' align='right'/> <!--nCpus-->
    <col valign='top' align='right'/> <!--speed-->
    <col valign='top' align='left'/> <!--arch-->
    <col valign='top' align='right'/> <!--memory-->
    <thead>
      <tr>
        <th>Name</th>
        <th># CPUs</th>
        <th>Speed (MHz)</th>
        <th>Architecture</th>
        <th>Memory (MB)</th>
      </tr>
    </thead>
    <tbody>
      <!--additional rows for each server...-->
    </tbody>
  </table>
  <hr/>
  <!--Bullet list of open labs-->
  <h2>Open labs</h2>
  <p>Click on the room name for a list of the workstations in
```
\[3.2. \text{XHTML for the room page}\]

Each room page displays a table of the configurations of all the client machines in a given room.

Unlike the start page, each room page is a complete standalone page. We'll use the \texttt{tccpage2} module\(^8\) to supply the overall page format including navigational links.

The set of navigational features on these pages will be minimal:

- \textit{Index}: A link back to the start page.
- \textit{TCC home}: A link to the TCC homepage\(^9\).

The page title, in both its \texttt{head} and \texttt{body/h1} elements, will have the form “TCC workstation inventory for \textit{type-of-room room-full}”, where \textit{type-of-room} will be either “open lab” or “classroom”, and \textit{room-full} is the full room name, e.g., “MSEC-III 187”.

\(^8\) \url{http://www.nmt.edu/tcc/projects/tccpage2/}
\(^9\) \url{http://www.nmt.edu/tcc/}
Here is the generated XHTML that will reside inside the body element of the generate page.

```xml
<table rules='all' border='8' cellpadding='4' cellspacing='4'>
    <col align='left' valign='top'/> <!--name-->  
    <col align='right' valign='top'/> <!--nCpus-->  
    <col align='right' valign='top'/> <!--speed-->  
    <col align='left' valign='top'/> <!--arch-->  
    <col align='right' valign='top'/> <!--memory-->  
    <col align='left' valign='top'/> <!--os-->  
    <col align='left' valign='top'/> <!--devices-->  
    <thead>
        <tr>
            <th valign='bottom'>Name</th>
            <th valign='bottom'># CPUs</th>
            <th valign='bottom'><div>Speed</div><div>(MHz)</div></th>
            <th valign='bottom'>Architecture</th>
            <th valign='bottom'><div>Memory</div><div>(MB)</div></th>
            <th valign='bottom'><div>Operating</div><div>system</div></th>
            <th valign='bottom'><div>Peripheral</div><div>devices</div></th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td align='left' valign='top'><tt>client.name</tt></td>
            <td align='right' valign='top'>client.nCpus</td>
            <td align='right' valign='top'>client.speed</td>
            <td align='left' valign='top'>client.arch</td>
            <td align='right' valign='top'>client.memory</td>
            <td align='left' valign='top'>client.os</td>
            <td align='left' valign='top'>
                <div class='hang'>
                    device type: device detail
                </div>
            </td>
        </tr>
        ...additional rows for each client...
    </tbody>
</table>
```

When the `client.os` is the default value ‘ ’, meaning a dual-boot workstation, the contents of that `td` element will be two strings vertically stacked with `div` elements: “Linux/” and “Windows”.

For the “Peripheral devices” column, each device description is set in a `div` element with a class='hang' attribute. This triggers a `div`'s `hang` rule in the TCC stylesheet that uses hanging indentation. That is, if a device description does not fit on one line, successive continuation lines will be...
indented. This allows the reader’s eye to scan down the list of device types without being distracted by continuation lines.

Here is an example of the content of the devices column. This is taken from the information for speare4a-1-2, which has two video cards.

```html
<div class='hang'>
  VIDEO: RV370 5B60 [Radeon X300 (PCIE)]
</div>
<div class='hang'>
  VIDEO: RV370 [Radeon X300SE]
</div>
<div class='hang'>
  MONITOR: 19.1" SAM SyncMaster
</div>
<div class='hang'>
  OPTICAL: TOSHIBA CDW/DVD SD-R1612
</div>
```

4. Prologue

The actual code for `hwscan3` starts with a line to make it self-executing, then the usual opening comments.

```python
#!/usr/bin/env python
#================================================================
# hwscan3.py: Summarize hardware configurations in Web form
# $Revision: 1.17 $ $Date: 2011/02/18 21:35:19 $
#----------------------------------------------------------------
SCRIPT_NAME = 'hwscan3.py'
EXTERNAL_VERSION = '3.0'
```

5. Imports

Here are the modules we’ll need to import. First, everybody needs the `sys` module for standard output and standard error streams, the `time` module for date and time work, and the `os` module and `stat` modules for working with Posix file systems.

```python
#================================================================
# Imports
#----------------------------------------------------------------
import sys
import time
import os, stat
```

Python’s standard `ldap` module handles the extraction of information from the TCC’s LDAP server. See the SourceForge page for this module\(^{10}\).

\(^{10}\)http://python-ldap.sourceforge.net/
Because LDAP identifies machines by host name, but the `GetHardware` interface requires an IP address in dotted quartet form, we'll need the standard Python `socket` interface to translate names to IP addresses.

```
import socket
```

The `GetHardware` interface is an XML-RPC service. Python's `xmlrpclib` handles communications with such interfaces.

```
import xmlrpclib
```

The program needs to read an XML file (`scanreport.xml`), and also to generate Web pages as XHTML. The `lxml.etree` package handles both reading and writing XML; we import it as `et`. For particulars of this package, see `Python XML processing with lxml`\(^\text{11}\).

To simplify the generation of XHTML, we use the `etbuilder` module described in `Python XML processing with lxml`\(^\text{12}\). This package also provides the `lxml.etree` module as `"et"`.

```
from etbuilder import et, E
```

We also use the `pyrag` application to generate Python names for all the XML elements and attributes in `scanreport.xml`. For information on this application, see `pyrag: A single-sourcing tool for Python-XML applications`\(^\text{13}\). The file built by `pyrag` from `scanreport.xml` is named `rnc_scanreport`.

```
from rnc_scanreport import *
```

The `tccpage2.py` module contains the `TCCPage` class, used for generating whole pages that conform to the standard TCC web page layout. See `tccpage.py: Dynamic generation of TCC-style web pages with lxml`\(^\text{14}\).

```
import tccpage2
```

## 6. Manifest constants

Constants used throughout the script have names in all capital letters.

```
#Manifest constants
```

### 6.1. HTML_SUFFIX

File extension for HTML pages.

---

\(^\text{11}\) http://www.nmt.edu/tcc/help/pubs/pylxml  
\(^\text{12}\) http://www.nmt.edu/tcc/help/pubs/pylxml  
\(^\text{13}\) http://www.nmt.edu/tcc/help/lang/python/examples/pyrag/  
\(^\text{14}\) http://www.nmt.edu/tcc/projects/tccpage2/
6.2. TCC_URL

The common prefix of all of the TCC’s web pages. This starts a sequence of interrelated constants regarding the URLs and path names of the pages we generate.

```python
TCC_URL = "http://www.nmt.edu/"
```

6.3. TCC_PATH

The absolute path name corresponding to Section 6.2, “TCC_URL” (p. 10).

```python
TCC_PATH = "/u/www/docs/"
```

6.4. BASE_DIR

This string is the relative directory between the base of the TCC Web and the directory where all the output pages are generated.

```python
BASE_DIR = "tcc/hw/hwscan3/ims3/start/"
```

6.5. BASE_URL

This is the common prefix of all URL of pages generated by this script.

```python
BASE_URL = TCC_URL + BASE_DIR
```

6.6. BASE_PATH

This is the common prefix of all path names of pages generated by this script.

```python
BASE_PATH = TCC_PATH + BASE_DIR
```

6.7. START_FILE

The file name (minus extension) of the generated start page.

```python
START_FILE = 'start'
```

6.8. START_URL

URL of the generated start page.
START_URL = BASE_URL + START_FILE + HTML_SUFFIX

6.9. START_PATH
Path name of the generated start page.

START_PATH = BASE_PATH + START_FILE + HTML_SUFFIX

6.10. CSS_URL
The URL of the standard TCC CSS stylesheet.

CSS_URL = "http://www.nmt.edu/tcc/style.css"

6.11. TCC_HOME
URL of the TCC homepage.

TCC_HOME = TCC_URL + "tcc/"

6.12. LDAP_URL
The URL of the LDAP server.

LDAP_URL = "ldaps://ldap0.nmt.edu:636"

6.13. CLIENTS_DN
The Distinguished Name of the node in the TCC LDAP tree that contains entries for all the client machines.

CLIENTS_DN = "ou=clients,ou=machines,dc=tcc,dc=nmt,dc=edu"

6.14. NULL_FILTER
A filter expression for LDAP searches that passes through all the children, since all LDAP entries have an objectClass attribute, and "*" is a wildcard that matches all values.

NULL_FILTER = "(objectClass=*)"

6.15. LDAP_TRUE
The Boolean true value in LDAP.
6.16. **CN_ATTR**

The “common name” attribute of a `tccHardware` object is its unqualified host name, e.g., “msec187-inst”.

```python
CN_ATTR = 'cn'
```

6.17. **OFFICE_ATTR**

The name of the LDAP attribute that is equal to `LDAP_TRUE` if the machine is in someone’s office, and therefore not considered publicly accessible.

```python
OFFICE_ATTR = 'tccOfficeMachine'
```

6.18. **OS_ATTR**

The name of the LDAP attribute that specifies what operating system the client runs. If not present, the default value is Linux/Windows dual-boot.

```python
OS_ATTR = 'tccSpecialOS'
```

6.19. **ATTR_LIST**

A list of the LDAP attribute names we want to retrieve.

```python
ATTR_LIST = [CN_ATTR, OFFICE_ATTR, OS_ATTR]
```

6.20. **FORGE_SERVER**

The URL of the XML-RPC `GetHardware` service.

```python
FORGE_SERVER = 'https://fedora.nmt.edu/ForgeWeb11/hwinfo/'
```

6.21. **REPORT_FILE**

Name of the `scanreport.xml` file.

```python
REPORT_FILE = '/u/www/docs/tcc/hw/hwscan3/ims3/scanreport.xml'
```

6.22. **TABLE_ATTRS**

A dictionary containing attributes to be applied to all our generated tables.
TABLE_ATTRS = {
    'rules': 'all', 'border': 4, 'cellpadding': 4, 
    'cellspacing': 4 }

6.23. L_ALIGN
For adding align='left' to table cells.
L_ALIGN = { 'align': 'left' }

6.24. R_ALIGN
For adding align='right' to table cells.
R_ALIGN = { 'align': 'right' }

6.25. TOP_VALIGN
For adding valign='top' to table cells.
TOP_VALIGN = { 'valign': 'top' }

6.26. BOT_VALIGN
For adding valign='bottom' to table cells.
BOT_VALIGN = { 'valign': 'bottom' }

6.27. CLASS
This function is lifted straight out of Fredrik Lundh’s example of XHTML generation in his original 
ElementTree builder proposal. It is a function that takes a list of CSS element class names and returns 
a dictionary that will decorate an element with those class names.
def CLASS(*nameList):
    return { 'class': " ".join(nameList) }

6.28. HANG_CLASS
CSS class for a hanging-indent paragraph in the TCC stylesheet\textsuperscript{15}.
HANG_CLASS = 'hang'

\textsuperscript{15} http://www.nmt.edu/tcc/style.css
6.29. **INST SUFFIX**

We know that a client is an instructor machine if it ends with this string.

```
INST_SUFFIX = '-inst'
```

7. **main():The main program**

```python
# - - - - - h w s c a n . p y - - m a i n - - - - -

def main():
    '''Main program.

    [ (ldap0.nmt.edu contains client info) and
      (the GetHardware XML-RPC interface has hardware
      configuration data) ->
      start page := (table of servers) + (list of open lab
      rooms with links to open lab room pages) +
      (table of classrooms with links to classroom room
      pages)
      room pages for all open labs and classrooms :=
      tables of clients in each room ]

    ...
```

The first step is to instantiate the two important data sources, the `ReportInfo` and `ClientSet` instances. We create the `ReportInfo` instance with the miscellaneous report data first, because the `ClientSet` instance needs the list of peripheral devices types from `scanreport.xml`.

```python
#-- 1 --
# [ if REPORT_FILE names a readable, valid file conforming to
#   scanreport.rnc ->
#   return a new ReportInfo object representing that file
# else ->
#   sys.stderr += error message
#   stop execution ]
print >>sys.stderr, ( "=== %s %s ===" %
    (SCRIPT_NAME, EXTERNAL_VERSION) )
try:
    reportInfo = ReportInfo( REPORT_FILE )
except IOError, detail:
    print >>sys.stderr, ( "*** Can't read the report information "
        "file '%s': %s" % (REPORT_FILE, detail) )
    raise SystemExit

#-- 2 --
# [ (ldap0.nmt.edu contains client info) and
#   (the GetHardware XML-RPC interface has hardware
#   configuration data) ->
#   clientSet := the singleton ClientSet() instance
#   representing these data ]
clientSet = ClientSet(reportInfo)
```
For the logic that generates all the Web pages, see Section 8, “buildPages(): Build all Web pages” (p. 15).

```python
#-- 3 --
# [ start page := (table of servers from clientSet) +
#  (list of open lab rooms from reportInfo with links to
#   open lab room pages from clientSet) +
#  (table of classrooms from reportInfo with links to
#   classroom room pages from clientSet)
#  room pages for all open labs and classrooms :=
#  tables of clients in each room from clientSet ]
buildPages( reportInfo, clientSet )
```

8. buildPages(): Build all Web pages

This is the top-level function for building all the output Web pages.

```python
# - - - b u i l d P a g e s
def buildPages( reportInfo, clientSet ):
    '''Build all output Web pages
    [ (reportInfo is a ReportInfo instance) and
      (clientSet is a ClientSet instance) ->
      start page := (table of servers from clientSet) +
      (list of open lab rooms from reportInfo with links to
       open lab room pages from clientSet) +
      (table of classrooms from reportInfo with links to
       classroom room pages from clientSet)
      room pages for all open labs and classrooms :=
      tables of clients in each room from clientSet ]
    ...
    '''
```

Page generation interacts with the existing PyStyler web:

- The actual start page is under the control of the PyStyler system at www.nmt.edu/tcc/hw/index.html. Here, we generate only the content inside the body element. It will be included inside the PyStyler page using a server-side include.

  Eventually, when PyStyler is redesigned so that it uses a separate namespace for PyStyler commands, and it supports dynamic generation of pages using PyStyler templates, this page generation will be reworked to generate the start page directly from the template.

- Room pages will use the tccpage2 module so that they look like PyStyler pages.

Generation of the body content of the start page follows the outline described in Section 3.1, “XHTML for the start page” (p. 5).

```python
#-- 1 --
# [ div := a new et.Element of type 'div' ]
div = et.Element( 'div' )
```

For the generation of the server table, see Section 9, “buildServerTable(): Build the table of servers” (p. 16).
For the generation of the open labs list, see Section 12, “buildOpenLabsList(): Bullet list of open labs” (p. 19).

For the generation of the classroom table, see Section 21, “buildClassroomTable(): Build the table of classrooms” (p. 27).

Finally, write the resulting XHTML to the start page. We must also give it world execute permission so that the server-side include will work; this change was made necessary in January 2009 when the infohost server was upgraded.

9. buildServerTable(): Build the table of servers

This function build the table of servers. It starts by building the constant parts of the table, including an empty tbody element. For a summary of the XHTML we are building, see Section 3.1, “XHTML for the start page” (p. 5).
# - - - b u i l d S e r v e r T a b l e

def buildServerTable ( parent, reportInfo, clientSet ):
    '''Build the table of login servers.
    [ (parent is an et.Element) and
      (reportInfo is a ReportInfo) and
      (clientSet is a ClientSet) ->
      parent += a table showing servers named in reportInfo,
      with configuration data from clientSet ]
    ...

See Section 6.22, “TABLE_ATTRS” (p. 12) for a constant dictionary that defines standard table decorations.

#-- 1 --
# [ parent += an h2 subhead ]
parent.append ( E.h2 ( 'Linux servers for remote login' ) )

#-- 2 --
# [ parent += an XHTML table as an et.Element, with an
#   empty tbody element
# tbody := that empty tbody element ]
table = E.table ( TABLE_ATTRS,
    E.col ( TOP_VALIGN, R_ALIGN ),
    E.col ( TOP_VALIGN, R_ALIGN ),
    E.col ( TOP_VALIGN, L_ALIGN ),
    E.col ( TOP_VALIGN, R_ALIGN ),
    E.thead ( E.th ( 'Name' ),
        E.th ( '# CPUs' ),
        E.th ( 'Speed (MHz)' ),
        E.th ( 'Architecture' ),
        E.th ( 'Memory (MB)' ) ) )
tbody = et.SubElement ( table, 'tbody' )
parent.append ( table )

For the logic that locates, sorts, and displays the login servers, see Section 10, “findLoginServers(): Display login server rows” (p. 17).

#-- 3 --
# [ tbody += tr elements containing one row for each
#   login server named in reportInfo, with configuration
#   data from clientSet ]
findLoginServers ( tbody, reportInfo, clientSet )

10. findLoginServers(): Display login server rows

This function scans the clientSet for systems that are considered login servers, and formats the rows of the server table.
def findLoginServers ( tbody, reportInfo, clientSet ):
    '''Build the rows of the login server table.
    [ (tbody is an et.Element) and 
    (reportInfo is a ReportInfo instance) and 
    (clientSet is a ClientSet instance) ->
    tbody += tr elements containing one row for each
    login server named in reportInfo, with configuration
    data from clientSet ]
    '''

    # The servers are presented in alphabetical order. We'll drive the process by extracting the keys from
    # reportInfo.systemMap and sorting them, and then checking each of the System instances to see if
    # its .isServer attribute is set.

    #-- 1 --
    # [ nameList := a list of the keys of reportInfo.systemMap,
    #   sorted in ascending order ]
    nameList = reportInfo.systemMap.keys()
    nameList.sort()

    #-- 2 --
    # [ tbody += tr elements containing one row for each
    #   login server found in reportInfo.systemMap, in
    #   order by the names from nameList, with configuration
    #   data from clientSet ]
    for systemName in nameList:
        #-- 2 body --
        # [ if (reportInfo.systemMap[systemName].isServer) and
        #   (systemName is known to clientSet) ->
        #   tbody += a tr element for
        #   reportInfo.systemMap[systemName] with
        #   configuration data from clientSet
        # else -> I ]
        system = reportInfo.systemMap[systemName]
        if system.isServer:
            #-- 2.1 --
            # [ if system.nodename is known to clientSet ->
            #   tbody += a tr element for system with
            #   configuration data from clientSet
            # else ->
            # sys.stderr += message about unknown server ]
            buildServerRow ( tbody, system, clientSet )

11. buildServerRow(): Format one row of the server table

Given a System instance and a ClientSet with configuration data, this function attempts to build
one row of the server table.
**It may happen that a server is unknown to the GetHardware interface. In that case, a message is written to the standard error stream.**

```python
# - - - b u i l d S e r v e r R o w

def buildServerRow ( tbody, system, clientSet ):
    '''Build one row of the server table on the start page.

    [ (tbody is an et.Element) and
      (system is a System instance) and
      (clientSet is a ClientSet instance) ->
        if system.nodename is known to clientSet ->
          tbody += a tr element for system with
          configuration data from clientSet
        else ->
          sys.stderr += message about unknown server ]

    ...

    #-- 1 --
    # if clientSet knows about (system) ->
    #    clientConfig := a ClientConfig instance that
    #      describes system's configuration
    # else ->
    #    sys.stderr += message about unknown server
    # return 
    try:
        clientConfig = clientSet.lookupClient ( system.nodename )
    except KeyError:
        print >>sys.stderr, ( "*** GetHardware does not know about 
        " + system.nodename )
    return

    Formatting the table row is straightforward. For the XHTML, see Section 3.1, “XHTML for the start page” (p. 5).

    #-- 2 --
    # [ tr := a new tr et.Element representing the hardware
    #   configuration data for clientConfig ]
    tr = E.tr ( E.td ( L_ALIGN, E.tt ( clientConfig.hostName ) ),
               E.td ( R_ALIGN, clientConfig.nCpus ),
               E.td ( R_ALIGN, clientConfig.speed ),
               E.td ( L_ALIGN, clientConfig.arch ),
               E.td ( R_ALIGN, clientConfig.memory ) )

    #-- 3 --
    # [ tbody += tr ]
    tbody.append ( tr )
```

**12. buildOpenLabsList(): Bullet list of open labs**

This function builds the bullet list of open labs on the start page. It also builds the room pages for each open lab, which are linked from the bullet list.
# -- - -  b u i l d  O p e n  L a b s  L i s t

def buildOpenLabsList ( parent, reportInfo, clientSet ):
    '''Build open lab pages and the bullet list that links to them.

    (parent is an et.Element) and
    (reportInfo is a ReportInfo) and
    (clientSet is a ClientSet ) ->
    parent +:= bullet list of open labs named in reportInfo,
    with configuration data from clientSet, as
    links to room pages for those labs
    room pages for open labs := tables of clients in
    those labs from clientSet ]

    ...

If we sort the keys of reportInfo.roomMap, that will put the rooms more or less into ascending order.
Note that, for instance, speare4 sorts between speare23 and speare5. If that ever bothers anyone,
we could split the room number into a string room name and an integer room number and sort those.
However, this is probably sufficient for now.

#-- 1 --
# [ prefixList := keys of reportInfo.roomMap, sorted into
#   ascending order ]
prefixList = reportInfo.roomMap.keys()
prefixList.sort()

Next comes the subhead and the bullet list.

#-- 2 --
# [ parent +:= (hr element)+(h2 subtitle) ]
parent.append ( E.hr() )
parent.append (E.h2 ('Open labs') )
parent.append (E.p ( 'Click on the room name for a list of the'
                  'workstations in that room.' ) )

#-- 3 --
# [ parent +:= a new ul child element
#   ul := that element ]
ul = et.SubElement ( parent, 'ul' )

For each room prefix that is identified as an open lab in reportInfo.roomMap, we'll generate a room
page for that lab, and also add a bullet linking to that page. See Section 13, “buildOpenLab(): Build
the page for an open lab” (p. 21).

#-- 4 --
# [ room pages for open labs in reportInfo := tables of
#   clients in those labs
#   ul := ul with new 'li' elements added for open labs
#   in reportInfo, linking to those corresponding
#   room pages ]
for prefix in prefixList:
    #-- 4 body --
    # [ if reportInfo.roomMap[prefix] is an open lab ->
    #   room page for that lab := table of clients in that lab from clientSet, with devices from reportInfo
    #   ul += a new 'li' element linking to that room page ]
    room = reportInfo.roomMap[prefix]
    if room.roomType == Room.ROOM_OPEN:
        buildOpenLab ( ul, room, reportInfo, clientSet )

13. `buildOpenLab()`: Build the page for an open lab

This function does two things. It builds the page showing a table of all the clients in a given open lab room. It also adds a link to that page to the open-labs bullet list on the start page.

```python
def buildOpenLab ( ul, room, reportInfo, clientSet ):
    '''Build an open lab room page and the link to it.
    
    [ (ul is an et.Element) and
      (room is a Room instance) and
      (reportInfo is a ReportInfo) and
      (clientSet is a ClientSet) ->
      room page for room := table of clients in that room from clientSet, with devices from reportInfo
      ul += a new 'li' element linking to that page ]
    '''
    First add a bullet to the bullet list, containing the room name, with a link to the room page.
```

```python
    #-- 1 --
    # [ ul += a new 'li' element with a link to the room page for (room) ]
    ul.append ( E.li ( linkToRoomPage ( room ) ) )
```

For the logic that builds the room page, see Section 17, “`buildRoomPage()`: Make the page with a table of all the clients in a room” (p. 23).

```python
    #-- 2 --
    # [ room page := table of clients in room (room)
    #    from clientSet, with devices from reportInfo ]
    buildRoomPage ( room, reportInfo, clientSet )
```

14. `linkToRoomPage()`: Build a hyperlink to the room page

This function returns an `et.Element` that links to the room page, using the room’s full name as its link text.
# - - - l i n k T o R o o m P a g e

def linkToRoomPage ( room ):
    '''Generate a hyperlink to the page for a given room.
    
    [ room is a Room instance ->
        return an 'a' et.Element that links to the room page
        for room, using room.roomFull as the link text
    ]

See Section 15, “roomPageURL(): Generate the URL for a room page” (p. 22).

#-- 1 --
return E.a ( room.roomFull, href=roomPageURL ( room ) )

15. roomPageURL(): Generate the URL for a room page

Given a Room instance, this function returns the absolute URL of the room page for that room. Be sure
to keep this logic in sync with:

• Section 16, “roomPagePath(): Absolute path name of a room page” (p. 22).
• Section 6.5, “BASE_URL” (p. 10).
• Section 6.1, “HTML_SUFFIX” (p. 9).

# - - - r o o m P a g e U R L

def roomPageURL ( room ):
    '''Returns the URL for a given room's page.
    
    return "%s%s%s" % (BASE_URL, room.roomPrefix, HTML_SUFFIX)

16. roomPagePath(): Absolute path name of a room page

This function returns the absolute path name of the room page for a given room. Be sure to keep this
logic in sync with:

• Section 15, “roomPageURL(): Generate the URL for a room page” (p. 22).
• Section 6.6, “BASE_PATH” (p. 10).
• Section 6.1, “HTML_SUFFIX” (p. 9).
17. buildRoomPage(): Make the page with a table of all the clients in a room

This function builds a room page, consisting mainly of a table displaying the configurations of all the clients in a given room. For the XHTML generated here, see Section 3.2, “XHTML for the room page” (p. 6).

```python
#--- 1 ---
# [ page := a new tccpage2.TCCPage instance with navigational
#    links pointing to the parent page and the TCC
#    homepage, and using the TCC stylesheet ]
title = ( "TCC workstation inventory for %s %s" %
    (room.roomFull, room.roomFullType) )
roomURL = roomPageURL ( room )
navList = [
    tccpage2.NavLink ( "TCC hardware",
        [["Server and workstation inventory", START_URL]] ),
    tccpage2.NavLink ( "TCC home",
        ["New Mexico Tech Computer Center", TCC_HOME]] ) ]
page = tccpage2.TCCPage ( title, navList, url=roomURL,
    cssUrl=CSS_URL )
```

For the generation of the page’s XHTML content, see Section 18, “buildRoomTable(): Build the table of client configurations” (p. 24).

```python
#--- 4 ---
# [ page.content += table of clients in room room.roomPrefix
#     from clientSet, with devices from reportInfo ]
buildRoomTable ( page.content, room, reportInfo, clientSet )
```

The tccpage2 module builds a basic page in the TCC standard format. For a discussion of the navigational features used, see Section 3.2, “XHTML for the room page” (p. 6). The page’s file name will be the room prefix, e.g., “speare116.html”; see Section 16, “roomPagePath(): Absolute path name of a room page” (p. 22).

For the definitions of relevant constants, see Section 6.8, “START_URL” (p. 10); Section 6.2, “TCC_URL” (p. 10); and Section 6.10, “CSS_URL” (p. 11).

---

16 http://www.nmt.edu/tcc/projects/tccpage2/
Finally, write the room page to its destination.

```python
#-- 5 --
# [ if we can create a new file named clientPath ->
#   file clientPath := page as XHTML
# else ->
#   sys.stderr += error message ]
roomFileName = roomPagePath ( room )
try:
    roomFile = open ( roomFileName, 'w' )
    page.write ( roomFile )
    roomFile.close()
except IOError, detail:
    print >>sys.stderr, ( "*** Can't create room page "
                      "'%s': %s" % (roomFileName, detail) )
```

### 18. buildRoomTable(): Build the table of client configurations

This function builds just the content part of the room page for one room. For the generated XHTML, see Section 3.2, “XHTML for the room page” (p. 6).

```python
#-- 1 --
# [ parent += a new table element defining the layout of
#   the client table, with an empty tbody element
# tbody := that empty tbody element ]
table = E.table ( TABLE_ATTRS,
                E.col ( TOP_VALIGN, L_ALIGN ),
                E.col ( TOP_VALIGN, R_ALIGN ),
                E.col ( TOP_VALIGN, R_ALIGN ),
                E.col ( TOP_VALIGN, L_ALIGN ),
                E.col ( TOP_VALIGN, R_ALIGN ),
                E.col ( TOP_VALIGN, L_ALIGN ),
```

For the constants used here, see Section 6.22, “TABLE_ATTRS” (p. 12); Section 6.23, “L_ALIGN” (p. 13); Section 6.24, “R_ALIGN” (p. 13); Section 6.25, “TOP_VALIGN” (p. 13); and Section 6.26, “BOT_VALIGN” (p. 13).
To find all the clients in this room, we generate the ClientConfig instances in `clientSet`, and look for those whose `hostName` attribute starts with `room.roomPrefix`. For the formatting of one row, see Section 19, “buildRoomRow(): Build one row of the room table” (p. 25).

```python
#-- 2 --
# [ tbody += rows describing the configurations of clients
# in clientSet whose names start with room.roomPrefix ]
for clientConfig in clientSet.genClients():
    fullPrefix = room.roomPrefix + "-"
    if clientConfig.hostName.startswith ( fullPrefix ):
        #-- 2 body --
        # [ tbody += a tr element describing the
        #   configuration of clientConfig with devices
        #   from reportInfo ]
        buildRoomRow ( tbody, reportInfo, clientConfig )
```

### 19. buildRoomRow(): Build one row of the room table

This function builds an XHTML `tr` element that describes one client's configuration. For the XHTML generated, see Section 3.2, “XHTML for the room page” (p. 6).

```python
# --- buildRoomRow ---
def buildRoomRow ( tbody, reportInfo, clientConfig ):
    """Build one row of the client configuration table.

    [ (tbody is an et.Element) and
      (clientConfig is a ClientConfig instance) ->
      tbody += a tr element describing the configuration
      of clientConfig ]
```

First we build a snippet of XHTML to form the content of the “Operating system” column. For dual-boot systems, this is two stacked div elements containing the strings “Linux/” and “Windows”.

```python
#-- 1 --
# [ if clientConfig.os == '' ->
#   osContent := vertically stacked strings "Linux/" and
#   "Windows"
# else ->
#   osContent := clientConfig.os ]
if clientConfig.os:
    osContent = clientConfig.os
else:
    osContent = E.div(E.div("Linux/"),
                      E.div("Windows"))
```

The other nontrivial cell content is the device listing. For the construction of this content, see Section 20, “buildDevices(): Extract relevant device configurations” (p. 26).

```python
#-- 2 --
# [ deviceContent := a div containing divs for each
#   device named in reportInfo that matches a device
# from clientConfig ]
deviceContent = buildDevices(reportInfo, clientConfig)
```

All that remains is to assemble the pieces.

```python
#-- 3 --
tbody.append(E.tr(E.th(L_ALIGN, TOP_VALIGN, E.tt(clientConfig.hostName)),
                  E.th(R_ALIGN, TOP_VALIGN, clientConfig.nCpus),
                  E.th(R_ALIGN, TOP_VALIGN, clientConfig.speed),
                  E.th(L_ALIGN, TOP_VALIGN, clientConfig.arch),
                  E.th(R_ALIGN, TOP_VALIGN, clientConfig.memory),
                  E.th(L_ALIGN, TOP_VALIGN, osContent),
                  E.th(L_ALIGN, TOP_VALIGN, deviceContent)))
```

### 20. buildDevices(): Extract relevant device configurations

Given a reportInfo instance that specifies which devices we care about, and a clientConfig instance that specifies a given client configuration, this function builds an XHTML fragment containing a stack of div elements, one for each matching device. For the generated XHTML, see Section 3.2, “XHTML for the room page” (p. 6).

```python
# -- -- buildDevices

def buildDevices(reportInfo, clientConfig):
    '''Build the contents of the "Devices" cell in the client table.
    [ (reportInfo is a ReportInfo) and
```
We'll start by building a `div` element that will hold all the device `div` elements.

Next we'll step through the device types of interest from `reportInfo`. For each type, we will then look through the device list in `clientConfig`, and for each matching device type, we add another `div` to `result`. For the definitions of the constants, see Section 6.27, “CLASS” (p. 13) and Section 6.28, “HANG_CLASS” (p. 13).

21. `buildClassroomTable()`: Build the table of classrooms

This function builds the table of classrooms, and also the room pages for each room.
We'll start by adding the horizontal rule and subhead, then the table without the content of its `tbody` element.

```python
#-- 1 --
# [ parent += (horizontal rule) + (subhead) ]
parent.append(E.hr())
parent.append(E.h2('Classrooms'))
parent.append(E.p('Click on the room name for a list of the ' 
                   'workstations in that room.'))
```

For relevant constants, see Section 6.22, “TABLE_ATTRS” (p. 12), Section 6.23, “L_ALIGN” (p. 13), and Section 6.24, “R_ALIGN” (p. 13).

```python
#-- 2 --
# [ parent += a new table element with col, thead, and 
#    an empty tbody element 
#    tbody := that tbody element ]
table = E.table(TABLE_ATTRS,
               E.col(L_ALIGN),
               E.col(R_ALIGN),
               E.col(R_ALIGN),
               E.thead(
                   E.tr(
                    E.th('Room'),
                    E.th('Workstations'),
                    E.th('Instructor station'))))
parent.append(table)
tbody = et.SubElement(table, 'tbody')
```

The overall ordering of the classrooms in the classroom table is determined by `reportInfo.roomMap`. We'll generate the rooms in ascending order by prefix. The logic that checks to see if it's a classroom, and builds each classroom line in the table as well as the room page for that classroom, is in Section 22, “buildClassroom()::Build one classroom table line and page” (p. 29).

```python
#-- 3 --
# [ tbody += rows for classrooms named in reportInfo, with 
#    configuration data from clientSet, as links to room 
#    pages for those rooms 
#    room pages for those rooms += tables of clients in 
#    those rooms, with configuration data from clientSet 
#    and devices from reportInfo ]
prefixList = reportInfo.roomMap.keys()
prefixList.sort()
for prefix in prefixList:
    #-- 3 body --
    # [ if reportInfo.roomMap[prefix] is a classroom -> 
    #    tbody += a row for that classroom, with 
```
22. buildClassroom(): Build one classroom table line and page

This function builds the room page for a classroom. It also constructs that room’s row in the classroom table on the start page.

```python
# - - - b u i l d C l a s s r o o m

def buildClassroom ( parent, room, reportInfo, clientSet ):
    '''Build one classroom table row and classroom page.

        [ (parent is an et.Element) and
          (room is a Room instance) and
          (reportInfo is a ReportInfo) and
          (clientSet is a ClientSet) ->
            parent += a row for that classroom, with
            workstation and instructor station counts from
            clientSet, and a link to a room page for that room
          room page for that room := table of clients in
              that room, with configuration data from
              clientSet and devices from reportInfo ]

    ...
```

Building the classroom table line is a bit complex because we have to count the workstations and instructor stations in that room. For that logic, see Section 23, “buildClassroomRow(): Build one line of the classroom table” (p. 30).

```python
#-- 1 --
# [ parent += a tr with the room name and a link to its
#   room page, and count of workstations and instructor
#   stations from clientSet ]
buildClassroomRow ( parent, room, clientSet )
```

For the logic that builds the room page, see in Section 17, “buildRoomPage(): Make the page with a table of all the clients in a room” (p. 23).

```python
#-- 2 --
# [ room page for room += table of clients in room
#   room.roomPrefix from clientSet, with roomType in
#   the page title, with devices from reportInfo ]
bUILDroomPage ( room, reportInfo, clientSet )
```
23. buildClassroomRow(): Build one line of the classroom table

This function counts the number of workstations and instructor stations and builds the tr element displaying those numbers, with the room name as a link to the room page.

```python
# -- - - b u i l d C l a s s r o o m R o w

def buildClassroomRow ( parent, room, clientSet ):
    '''Build one row of the classroom table.'''
    [ (parent is an et.Element) and
      (room is a Room) and
      (clientSet is a ClientSet) ->
      parent += a tr with the room name as a link to the
      room page for (room), and count of workstations
      and instructor stations in room from clientSet ]
```

The first order of business is to generate all the clients from clientSet and count the number of workstations and instructor stations whose names start with `room.roomPrefix`.

```python
#-- 1 --
[# seatCount := number of clients in clientSet whose names
# start with room.roomPrefix but don't end with INST_SUFFIX
# instCount := number of clients in clientSet whose names
# start with room.roomPrefix but do end with INST_SUFFIX ]
seatCount = instCount = 0
for clientConfig in clientSet.genClients():
    fullPrefix = room.roomPrefix+"-"
    if clientConfig.hostName.startswith ( fullPrefix ):
        if clientConfig.hostName.endswith ( INST_SUFFIX ):
            instCount += 1
        else:
            seatCount += 1
```

We have everything we need now to build the actual table row.

```python
#-- 2 --
[# parent += a new tr element containing a link to
# pageURL with room.roomFull as the link text,
# seatCount, and instCount ]
parent.append ( E.tr ( E.td ( L_ALIGN, linkToRoomPage ( room ) ),
                       E.td ( R_ALIGN, seatCount ),
                       E.td ( R_ALIGN, instCount ) ) )
```
24. class Singleton: Parent class for singletons

This class implements the logic of the Singleton design pattern\(^\text{17}\). It is cribbed directly from Guido van Rossum’s document, *Unifying types and classes in Python 2.2*\(^\text{18}\).

```python
# - - - - - c l a s s S i n g l e t o n

class Singleton(object):
    def __new__(cls, *args, **kwds):
        it = cls.__dict__.get('__it__')
        if it is not None:
            return it
        cls.__it__ = it = object.__new__(cls)
        it.init(*args, **kwds)
        return it

    def init(self, *args, **kwds):
        pass

Inheriting classes should override method init(), not __init__."

25. class ClientSet: Container for client configurations

The singleton\(^\text{19}\) class ClientSet holds the configuration data for all TCC client systems (including login servers). Here is the interface:

```python
# - - - - - c l a s s C l i e n t S e t

class ClientSet(Singleton):
    '''Represents all available public client configuration data.

    Exports:
    ClientSet(reportInfo):
        [ (ldap0.nmt.edu contains client info) and
          (the GetHardware XML-RPC interface has hardware
          configuration data) and
          (reportInfo is a ReportInfo instance) ->
          return the singleton ClientSet() instance representing
          the publicly accessible clients from ldap0.nmt.edu
          known to GetHardware with peripheral configurations
          as described by reportInfo ]
    .genClients():
        [ generate a sequence of ClientConfig instances representing
          the clients in self, in alphabetical order by host name ]
    .lookupClient(hostName):
```
25.1. `ClientSet.genClients()`: Generate all client configurations

To generate the clients in alphabetical order by host name, we extract the keys from the `self.__clientMap` dictionary and sort them.

```python
# - - - ClientSet.genClients

def genClients(self):
    """Generate are ClientConfig children in hostname order."
    hostNameList = self.__clientMap.keys()
    hostNameList.sort()
    for hostName in hostNameList:
        yield self.__clientMap[hostName]
    raise StopIteration
```

25.2. `ClientSet.lookupClient()`: Find a specific client by host name

```python
# - - - ClientSet.lookupClient

def lookupClient(self, hostName):
    """Find a client by host name."
    return self.__clientMap[hostName]
```

25.3. `ClientSet.init()`: Constructor

For this class, the constructor is called `init()`, rather than the customary `__init__()`, because of the way the Singleton parent class works—it guarantees that the constructor will be called only once.

That's an especially good idea here, because the constructor has a lot to do. It has to rummage through the LDAP clients tree to find all the publicly available TCC clients (including login servers). Then it has to query the GetHardware XML-RPC interface for each client in order to extract and store that client's hardware configuration data. Both operations combined take, at this writing, on the order of half a minute.
The work of the constructor will proceed in two phases. In the first phase, we will extract the client list from LDAP. In the second phase, we will use that client list to extract configuration data from GetHardware. We need reportInfo because it tells us which peripheral devices are of interest to users.

So, what data structure should be passed between these two phases? If we use the LDAP tccOfficeMachine attribute to filter out office machines, we are left with only two items of information for each client: its name, and its tccSpecialOS attribute, if there is one—we’ll use an empty string as the default value.

At this point, we can represent the client list as a dictionary, where each key is the client name and the value is the tccSpecialOS attribute.

That dictionary, plus the information from GetHardware, is sufficient to set up our instance’s __clientMap.

--- 2 ---
# [ self := self with ClientConfig instances added for
# clients whose hostnames are keys of ldapMap and
# for which the GetHardware XML-RPC interface has
# client configuration data, including device types
# from reportInfo ]
self.__buildClientMap ( reportInfo, ldapMap )

25.4. ClientSet.__buildLdapMap(): Extract LDAP’s client list

This function encapsulates all our dealings with the LDAP server.

--- 1 ---
# [ ldap0.nmt.edu contains client info ->
#   ldapMap := a dictionary whose keys are the
#   non-office machine names from LDAP’s client tree,
#   and each corresponding value is its tccSpecialOS
#   attribute (defaulting to '') ]
ldapMap = self.__buildLdapMap()
First we create the dictionary to be returned, and the anonymous LDAP binding. The first two arguments to the .bind() method are the login name and password; we pass empty strings to them, giving us an anonymous binding. See Section 6.12, “LDAP_URL” (p. 11).

```python
#-- 1 --
# [ ldapMap := a new, empty dictionary
#   binding := an anonymous LDAP binding ]
ldapMap = {}
binding = ldap.initialize ( LDAP_URL )
binding.bind ( "", "", ldap.AUTH_SIMPLE )
```

The DN of the clients tree in LDAP is given by Section 6.13, “CLIENTS_DN” (p. 11) to find all client systems. The second argument, ldap.SCOPE_ONELEVEL, restricts the search to return only immediate children of the clients tree. The third argument is a trivial filter expression that does not actually filter out any children; see Section 6.14, “NULL_FILTER” (p. 11). The fourth argument is a list of the attribute names we want to retrieve; see Section 6.19, “ATTR_LIST” (p. 12).

```python
#-- 2 --
# [ resultList := an LDAP query result for children of
#   CLIENTS_DN ]
resultList = binding.search_s ( CLIENTS_DN, ldap.SCOPE_ONELEVEL, NULL_FILTER, ATTR_LIST )
```

The resultList is a list of 2-tuples. In each tuple, the first element is the DN of the entry, and the second element is a dictionary of attributes. Within this dictionary, the key of each entry is the attribute name, and the corresponding value is a list of attribute values. Once the ldapMap is built, it is returned to the caller. See Section 25.5, “ClientSet.__processLdapEntry(): Process one LDAP client record” (p. 35).

```python
#-- 3 --
# [ resultList is an LDAP search result ->
#   ldapMap := ldapMap with entries added for
#   results from resultList that are not
#   office machines, each key is the host name,
#   and each corresponding value is the
#   tccSpecialOS attribute value, defaulting to '' ]
for dn, attrMap in resultList:
  #-- 3 body --
  # [ attrMap is a dictionary whose keys are LDAP
  #   attribute names from RESULT_LIST, and each
  #   corresponding value is a list of the values of
  #   that attribute ->
  #   if attrMap[OFFICE_ATTR][0] != LDAP_TRUE ->
  #     ldapMap := ldapMap with an entry added
  #     whose key is the CN_ATTR attribute from
  #     attrMap and whose corresponding value is
  #     the OS_ATTR value from attrMap,
  #     defaulting to ''
  #   else -> I ]
  self.__processLdapEntry ( ldapMap, attrMap )

#-- 4 --
return ldapMap
```
25.5. ClientSet.__processLdapEntry(): Process one LDAP client record

This function examines one of the results returned from the LDAP search that describes a client machine. If the result describes an office machine, it is ignored. Otherwise an entry is added to the ldapMap whose key is the host name and whose value is the tccSpecialOS attribute.

```python
# - - - ClientSet._._processLdapEntry

def __processLdapEntry ( self, ldapMap, attrMap ):
    '''Examine one client record.
    [ attrMap is a dictionary whose keys are LDAP attribute names from RESULT_LIST, and each corresponding value is a list of the values of that attribute ->
    if attrMap[OFFICE_ATTR][0] != LDAP_TRUE ->
    ldapMap := ldapMap with an entry added whose key is the CN_ATTR attribute from attrMap and whose corresponding value is the OS_ATTR value from attrMap, defaulting to ''
    else -> I ]
    ...
    #-- 1 --
    if attrMap[OFFICE_ATTR][0] == LDAP_TRUE:
        return
    #-- 2 --
    # [ if attrMap has an entry for key OS_ATTR ->
    #    os := attrMap[OS_ATTR][0]
    # else ->
    #    os := '' ]
    try:
        os = attrMap[OS_ATTR][0]
    except KeyError:
        os = ''
    #-- 3 --
    # [ ldapMap := ldapMap with a new entry whose key is attrMap[CN_ATTR][0] and whose values is os ]
    cn = attrMap[CN_ATTR][0]
    ldapMap[cn] = os
```

First we eliminate the office machines. See Section 6.17, “OFFICE_ATTR” (p. 12) and Section 6.15, “LDAP_TRUE” (p. 11).

At this point, we know that the client is publicly accessible. Find its tccSpecialOS attribute, if it has one; the default value is the empty string. Finally, add an entry to ldapMap.
25.6. ClientSet.__buildClientMap(): Extract configuration data

This function handles all calls to the GetHardware XML-RPC service. For clients that are known to GetHardware, we build a ClientConfig instance containing the client's hardware configuration data, and add an entry to self.__clientMap whose key is the client's unqualified host name and whose value is that ClientConfig instance.

```python
def __buildClientMap ( self, reportInfo, ldapMap):
    '''Transform LDAP client entries into ClientConfig instances.
    
    [ (reportInfo is a ReportInfo instance) and
    (ldapMap is a dictionary whose keys are unqualified
    client names, and each corresponding value is that
    client's tccSpecialOS value (defaulting to '')) ->
    self := self with ClientConfig instances added for
    clients whose hostnames are keys of ldapMap and
    for which the GetHardware XML-RPC interface has
    client configuration data ]

    ...
```

First we instantiate an XML-RPC server to handle all the queries to GetHardware. See Section 6.20, “FORGE_SERVER” (p. 12).

```python
#-- 1 --
# [ rpcServer := an XML-RPC server at FORGE_SERVER
# self.__clientMap := a new, empty dictionary ]
rpcServer = xmlrpclib.ServerProxy ( FORGE_SERVER )
self.__clientMap = {}
```

For each entry in ldapMap, we query this server and, if a result is returned, add a corresponding entry to self.__clientMap.

```python
#-- 2 --
# [ self.__clientMap +:= entries for clients from
#    the host names from the keys of ldapMap that
#    are known to rpcServer, where each entry's key
#    is the host name and the corresponding value
#    is a ClientConfig instance made from ldapMap[host name]
#    and the result returned from rpcServer ]
for hostName in ldapMap:
    #-- 2 body --
    # [ (hostName is a host name string) and
    #   (ldapMap[hostName] is a tccSpecialOS value) ->
    #   if rpcServer returns a value for hostName ->
    #   self.__clientMap[hostName] := a ClientConfig
    #   instance made from ldapMap[hostName], os,
    #   and the value returned from rpcServer
    #   else -> I ]
    self.__findConfig ( reportInfo, rpcServer, hostName,
    ldapMap[hostName] )
```
25.7. ClientSet.__findConfig(): Look up a system in the GetHardware service

This function calls GetHardware to retrieve the configuration data for one client.

```python
# - - - ClientSet.__findConfig

def __findConfig(self, reportInfo, rpcServer, hostName, os):
    '''Look up one client in the GetHardware interface.

    [ (reportInfo is a ReportInfo instance) and
      (rpcServer is an xmlrpclib.ServerProxy for FORGE_SERVER) and
      (hostName is a client host name string) and
      (os is a tccSpecialOS value) ->
        if rpcServer returns a value for hostName ->
          self.__clientMap[hostName] := a ClientConfig instance made from ldapMap[hostName], os,
          and the value returned from rpcServer,
          using device types from reportInfo
        else -> I ]
    ...
```

Because GetHardware expects a dotted IP address and not a host name, we use socket to translate the qualified host name to an IP value. It should never happen that socket doesn't know about one of our clients, but in case it does, we log a message and return.

```python
#-- 1 --
# [ fullName := fully qualified version of hostname ]
fullName = "%s.nmt.edu" % hostName

#-- 2 --
# [ if socket.gethostbyname() knows about fullName ->
#   ip := fullName's dotted IP address
# else ->
#   sys.stderr += error message
# return ]
try:
    ip = socket.gethostbyname(fullName)
except socket.gaierror, detail:
    print >>sys.stderr, ( "socket(%s): system unknown" % fullName )
    return

Next we see if GetHardware has configuration information about this IP. If not, it returns an empty list, and we return. For a successful call, a dictionary is returned, whose keys are attributes of the client.

```python
#-- 3 --
# [ if rpcServer has a record for ip ->
#   hardMap := a dictionary containing hardware configuration information for ip
# else -> return ]
```
hardMap = rpcServer.GetHardware(ip)
if type(hardMap) is list:
    print >>sys.stderr, ("*** GetHardware has no record
            "for host '%s'." % hostName)
    return

All that remains is to package up the host name, the os value, and the contents of hardMap into a
ClientConfig instance, and add it to self.__clientMap().

#-- 4 --
# [ clientConfig := a ClientConfig instance made from
#     hostName, os, and hardMap, with device classes
#     from reportInfo ]
clientConfig = ClientConfig(reportInfo, hostName, os, hardMap)

#-- 5 --
self.__clientMap[hostName] = clientConfig

26. class ClientConfig: Configuration data for one client

An instance of this class represents all the pertinent information about a particular client.

# - - - - - c l a s s C l i e n t C o n f i g

class ClientConfig:
    '''Describes the hardware configuration of one client.

    Exports:
    ClientConfig ( reportInfo, hostName, os, hardMap ):
        [ (reportInfo is a ReportInfo instance) and
        (hostName is a client host name string) and
        (os is the client's tccSpecialOS value) and
        (hardMap is the result returned by GetHardware for
        the client) ->
        return a new ClientConfig instance representing
        those values, with device types from reportInfo ]
    .hostName: [ as passed to constructor, read-only ]
    .os: [ as passed to constructor, read-only ]
    .nCpus: [ number of processors, an int ]
    .speed: [ speed in MHz, an int ]
    .arch: [ architecture, a str ]
    .memory: [ memory in MB, an int ]
    .deviceList:
        [ a list of two-element lists [device type, description] ]
    ...
    
These class constants define the keys in hardMap.

    N_CPUS = 'numCpus'
    SPEED = 'cpuSpeed'
26.1. ClientConfig.__init__(): Constructor

```python
# - - - C l i e n t C o n f i g . __ i n i t _ _

def __init__( self, reportInfo, hostName, os, hardMap ):
    '''Constructor for ClientConfig.'''
    #-- 1 --
    self.reportInfo = reportInfo
    self.hostName = hostName
    self.os = os

    #-- 1 --
    self.nCpus = hardMap[self.N_CPUS]
    self.speed = hardMap[self.SPEED]
    self.arch = hardMap[self.ARCH]
    self.memory = hardMap[self.MEMORY]
```

The next group of instance attributes to be set up come from the `hardMap` dictionary entries for various keys. Most of the values are strings, but the CPU count, speed, and memory are returned as integers.

```python
#-- 1 --
self.nCpus = hardMap[self.N_CPUS]
self.speed = hardMap[self.SPEED]
self.arch = hardMap[self.ARCH]
self.memory = hardMap[self.MEMORY]
```

Device configurations are a second-level dictionary contained in `hardMap[DEVICES]`. In each case, there may be no entry for that device type. If there is an entry for a given device type, add to `self.deviceList` a list with this structure:

```
[devType, detail]
```

where `devType` is the device type and `detail` is one of the descriptions found in `hardMap`.

```python
self.deviceList = []
deviceMap = hardMap[self.DEVICES]

#-- 2 --
# [ self.deviceList := as invariant ]
for deviceType in reportInfo.deviceList:
    #-- 2 body --
    # [ if deviceMap has a key equal to deviceType ->
    #   self.deviceList += a list [deviceType, detail]
    #   where detail is the detail part of the
    #   corresponding value from deviceMap ]
    self.__deviceCheck( deviceMap, deviceType )
```

26.2. ClientConfig.__deviceCheck(): Check for devices of a given type

The `deviceMap` passed to this function is a dictionary obtained from `GetHardware` in which the keys are device types such as `'VIDEO'` and the associated value is a list of two-element lists `[detail, ...]`.
If the `deviceMap` does have an entry for the desired device type, add to `self.deviceList` a two-element list `"[devType, [detail]]"` for each instance of that device type. For example, if a client has two video cards, there will be two elements in `self.deviceList` of the form `"["VIDEO", "model-info"]"`.

```python
# Client Config._.deviceCheck

def _deviceCheck ( self, deviceMap, deviceType ):
    '''Extract from deviceMap any device details for type deviceType

    [ (deviceMap has the format of a value from hardMap[self.DEVICES]) and
    (deviceType is a string) ->
    if deviceMap has a key equal to deviceType ->
    self.deviceList += a list [deviceType, detail] for each device of that type in
    deviceMap
    else -> I ]
    '''

First we check to see if this client has any devices of type `deviceType`.

```python
#-- 1 --
# [ if deviceMap has a key that matches deviceType ->
#    multiList := deviceMap[deviceType]
#   else -> return ]
try:
    multiList = deviceMap[deviceType]
except KeyError:
    return

At this point, `multiList` is a list with one member for each device of the right type. Each member is itself a two-element list `[detail, other]`, where `detail` is a string describing the device’s details, and `other` is a dictionary whose contents don’t interest us.

Our job is to add to `self.deviceList` a two-element list of the form `[devType, detail]` for each instance of this device type in `multiList`.

```python
#-- 2 --
# [ self.deviceList += two-element lists [devType, detail] for each element of multiList ]
for detail, other in multiList:
    self.deviceList.append ( [deviceType, detail] )
```

## 27. class ReportInfo: Reading report information

The purpose of this class is to make available the information from the `scanreport.xml` file. Here is the interface:
# - - - - - class ReportInfo - - - - -

class ReportInfo:
    '''Represents the XML file containing report parameters."
    
    Exports: ReportInfo ( fileName ):
        [ fileName is the name of the report file ->
        if fileName names a readable, valid file
        conforming to scanreport.rnc ->
        return a new ReportInfo object representing
        that file
        else -> raise IOError ]
    .fileName: [ as passed to constructor, read-only ]
    .roomMap: [ a dictionary whose values are Room objects, with the
    corresponding key its .prefix attribute ]
    .systemMap: [ a dictionary whose values are System objects, with the
    corresponding key its .nodename attribute ]
    .deviceList: [ a list of device types as strings ]

27.1. ReportInfo.__init__(): class constructor

    For general information on reading an XML file in Python, see Python XML processing with lxml20. Note that the et.parse() function may raise either of two exceptions. If the file is unreadable, it will raise IOError. If the file is readable but not well-formed, it raises et.XMLSyntaxError.

    To find all the ROOM N nodes in the document, we can use the XPath expression 
    
    To find all the ROOM N nodes in the document, we can use the XPath expression 
    “//room”. The resulting node set is converted into a set of entries in self.roomMap by Section 27.2, “ReportInfo.__build-Rooms(): Build the room dictionary” (p. 42).

20 http://www.nmt.edu/tcc/help/pubs/pylxml/
Similar techniques find all the SYSTEM_N nodes, and use those nodes to set up self.systemMap. See Section 27.3, “ReportInfo.__buildSystems(): Build the system map” (p. 43).

Finally, we go through the list of device classes and build self.deviceList.

27.2. ReportInfo.__buildRooms(): Build the room dictionary

The input to this method is a node list containing ROOM_N nodes. The purpose is to create the .roomMap attribute, mapping room names onto Room objects representing those rooms.

First we set up the empty dictionary, then iterate over the room nodes, adding the information from each to the dictionary.
27.3. ReportInfo.__buildSystems(): Build the system map

This routine takes a set of SYSTEM_N nodes and uses them to build the self.systemMap dictionary.

```python
#-- 1 --
self.systemMap = {}

#-- 2 --
# [ self.systemMap += entries mapping system names
#   from systemNodeSet ]
for systemNode in systemNodeSet:
    #-- 2 body --
    # [ systemNode is a SYSTEM_N et.Element ->
    #   self.systemMap := self.systemMap with an entry that maps
    #   the system name from systemNode |-> a new System object made
    #   from systemNode ]
    self.__makeSystem ( systemNode )
```

See Section 27.4, “ReportInfo.__makeSystem(): Build a system map entry” (p. 44).
27.4. ReportInfo.__makeSystem(): Build a system map entry

Given a SYSTEM_N node, this routine creates a new System object using the information in that node, and adds it to self.systemMap.

```python
# - - - R e p o r t I n f o . __ m a k e S y s t e m - - -

def __makeSystem ( self, systemNode ):
    '''Add a new system map entry.

    [ systemNode is a SYSTEM_N et.Element ->
      self.systemMap := self.systemMap with an
      entry that maps the system name from
      systemNode |-> a new System object made
      from systemNode ]
    ...

    #-- 1 --
    # [ systemName := NODENAME_A attribute from systemNode
    #  servAttr := SERVER_A attribute from systemNode,
    #  or '' if missing
    #  linuxAttr := LINUX_ONLY_A attribute from systemNode,
    #  or '' if missing ]
    systemName = systemNode.attrib [ NODENAME_A ]
    servAttr = systemNode.attrib.get ( SERVER_A, '' )
    linuxAttr = systemNode.attrib.get ( LINUX_ONLY_A, '' )

    #-- 2 --
    if len(servAttr) == 0:
        systemIsServer = 0
    else:
        systemIsServer = int(servAttr)

    #-- 3 --
    if len(linuxAttr) == 0:
        systemIsLinux = 0
    else:
        systemIsLinux = int(linuxAttr)

    #-- 4 --
    system = System ( systemName, systemIsServer,
      systemIsLinux )

    #-- 5 --
    self.systemMap [ systemName ] = system
```

See Section 29, “class System: System information” (p. 45) for this constructor.

28. class Room: Room information

A small object that holds room information.
class Room:
    '''Represents data about one room in the report parameters file.
    Exports:
    Room ( roomType, roomPrefix, roomFull ):
        [ (roomType is the room's type as a string) and
          (roomPrefix is the prefix for clients in that room) and
          (roomFull is the full name of the room) ->
          return a new Room object with those attributes ]
    .roomType: [ as passed to constructor, read-only ]
    .roomPrefix: [ as passed to constructor, read-only ]
    .roomFull: [ as passed to constructor, read-only ]
    .roomFullType: [ text corresponding to self.roomType, e.g.,
                     "open lab" or "classroom" ]
    .ROOM_OPEN: [ roomType for open labs ]
    .ROOM_CLASS: [ roomType for classrooms ]
    ...
    ROOM_OPEN = 'o'
    ROOM_CLASS = 'c'
    ROOM_TYPE_MAP = { ROOM_OPEN: "open lab", ROOM_CLASS: "classroom" }
    def __init__ ( self, roomType, roomPrefix, roomFull ):
        '''Constructor for Room'''
        self.roomType = roomType
        self.roomPrefix = roomPrefix
        self.roomFull = roomFull
        self.roomFullType = self.ROOM_TYPE_MAP[roomType]

29. class System: System information

Another small object to hold information about a system from the report parameters file.

class System:
    '''Represents one system from the report parameters file.
    Exports:
    System ( nodename, isServer, isLinux ):
        [ (name is the system's nodename as a string) and
          (isServer is 1 iff it is a server, else 0) and
          (isLinux is 1 iff it is a Linux-only system, else 0) ->
          return a new System object with those attributes ]
    .nodename: [ as passed to constructor, read-only ]
    .isServer: [ as passed to constructor, read-only ]
    .isLinux: [ as passed to constructor, read-only ]
```python
def __init__ ( self, nodename, isServer, isLinux ):
    '''Constructor for the System object.'''
    self.nodename = nodename
    self.isServer = isServer
    self.isLinux = isLinux
```

30. Epilogue

These last lines invoke the main program, Section 7, “main() : The main program” (p. 14).

```python
if __name__ == '__main__':
    main()
```