WIND RIVER

Wind River Linux, 3.0

USE CASE

Customizing a LAMP Stack with a Layer

1. Introduction

This use case illustrates a process of developing a lightweight Web application. It provides a simple demonstration of how you can use a layer and templates with LDAT, the Wind River Linux build system, to configure and build a middleware product.

This use case demonstrates how to do the following:

- create a layer
- configure templates in a layer
- modify Web server and Busybox configuration files with a layer
- add directories with a layer
- add a user with a layer
- reconfigure an existing project
- modify a target startup script with a layer
- deploy a simple Web application.

The middleware you will use comprises a "LAMP" model—named after the Linux-Apache-MySQL-Python stack often used to implement it. Some example components of LAMP model stacks are listed in Figure 1.

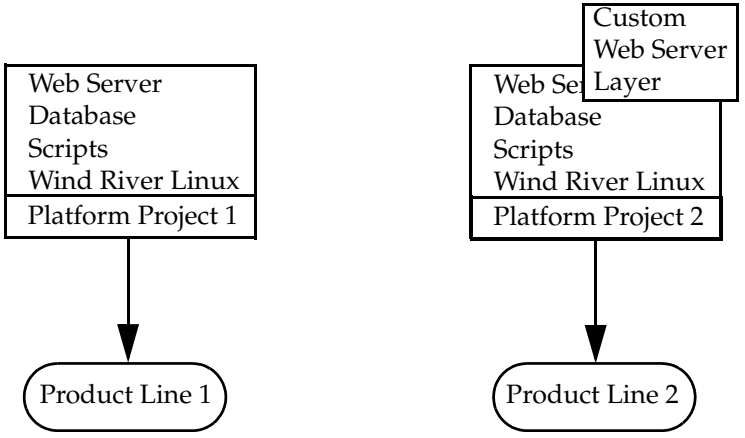
Figure 1 **LAMP Model**

Software	Examples
Scripts	Python, Perl, PHP, shell, other programming languages
Database	MySQL, PostgreSQL, sqlite, other databases
Web Server	Apache, Boa, other Web servers
Operating System	Wind River Linux, other operating systems

You begin by creating a base, default configuration with the kernel, file system, and board support package (BSP) you want to use. You can then create layers and templates to configure the base configuration in various ways, including adding and removing features.

As a simple example, suppose you have two product lines that contain essentially the same middleware but with different Web server configurations. You could use a layer to modify the default Web server configuration as illustrated in Figure 2.

Figure 2 **Customizing a Project With a Layer**



In the second product line, the modified configuration files present in the custom layer take precedence, so the Wind River build system overwrites the default Web server configuration files with the modified configuration files.

NOTE: The layer created in this use case is available for download as a zip file from Wind River Online support along with this PDF file.

Overview of Use Case

This use case describes one way to configure a layer to modify the default Web server configuration and add additional features. With product line 1 you configure a Boa Web server using the default configuration. With product line 2 you expand on this default configuration, modifying the Web server and adding a database and application script. The result is a small-footprint LAMP stack that runs a simple Web server application.

The remainder of this use case consists of the following sections:

2. Creating the Base Project

Product 1 consists of a small kernel and file system project with a Web server added.

1. Configure the basic project:

```
$ configure --enable-board=common_pc \
--enable-kernel=small \
--enable-rootfs=glibc_small
```

2. Add the Boa Web server package:

   ```
   $ make -C build boa.addpkg
   ```

3. Build Product 1:

   ```
   $ make
   ```

   When the build completes, you have a product configured with the default **boa.conf** and **index.html** files. You can view these files in *prjbuildDir*/**export/dist/etc/boa/boa.conf** and *prjbuildDir*/**export/dist/home/httpd/html/index.html**.

You can boot this product and start **boa** to enable a Web server listening at the default port 80. For the purposes of this use case, this default product will only be used to build a customized version.

3. Configuring the Customized Layer

Create a layer with custom Web server configuration files as described in this section.

Creating the Custom Layer for Product 2

Create a custom layer to hold a new configuration, for example:

```
$ mkdir -p $HOME/layers/Product_2/
```

This directory will be the layer that holds all the custom configurations used in this use case.

Configuring the Layer for the Customized Web Server

Add a template for Boa to the layer.

1. Create a **boa_mod** template and template infrastructure to hold the modified Boa files:

   ```
   $ mkdir -p $HOME/layers/Product_2/templates/feature/boa_mod/fs/etc/boa/
   $ mkdir -p \
   $HOME/layers/Product_2/templates/feature/boa_mod/fs/home/httpd/html/
   ```

Note that the directory structure below the **fs/** directory mimics the locations of directories in the default configuration of Product 1.

2. Copy the **boa.conf** file from Product 1 to the **boa_mod** template in the custom layer:

```
$ cd prjbuildDir
$ cp export/dist/etc/boa/boa.conf \
$HOME/layers/Product_2/templates/feature/boa_mod/fs/etc/boa/
```

3. Edit the new **boa.conf** file in the template to configure the server to use port 8080.

   Change:

   ```
   Port 80
   ```

   To:

   ```
   Port 8080
   ```

   Save the file.

4. Copy the **index.html** file from Product 1 to the **boa_mod** template in the custom layer:

```
$ cd prjbuildDir
$ cp export/dist/home/httpd/html/index.html \
$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/httpd/html/
```

5. Modify the default greeting. Edit the new **index.html** file in the template.

   Change:

   ```
   <html><body>boa is running</body></html>
   ```

   To:

   ```
   <html><body>boa is listening on port 8080</body></html>
   ```

   Save the file.

Add boa to a pkglist.add File

For Product 1 you added the **boa** package at the command line with the command **make -C build boa.addpkg** command. You can cause **boa** to be added automatically by including it in a **pkglist.add** file in your layer.

Create a **$HOME/layers/Product_2/templates/feature/boa_mod/pkglist.add** file with the following one line:

```
boa
```

Now when you specify the layer and template to the **configure** command, **boa** is added to the project's **pkglist** file.

4. Configuring the Layer for the Customized Busybox Utilities

Modify **busybox** to add some tools to the layer for use later in this use case.

1. In the *prjbuildDir* for Product 1, enter:

   ```
   $ make -C build busybox.menuconfig
   ```

   Add the following utilities by selecting them in the menu:

   - **Coreutils > chmod, chown, pwd, whoami**
   - **Editors > vi**

   NOTE: If you are not familiar with **vi** you can just copy the text that you would enter with it from the provided layer.

   - **Finding Utilities > grep**
   - **Login/Password Management Utilities > su**
   - **Networking Utilities > wget**
   - **Process Utilities >pkill, ps**

   Add any other utilities you may want to use, then exit and save your configuration.

2. Create a new template to hold your customized busybox configuration:

   ```
   $ mkdir -p $HOME/layers/Product_2/templates/feature/my_busybox/busybox/
   ```

3. Copy the new busybox configuration file to the new template:

   ```
   $ cp build/busybox-version/.config \
   $HOME/layers/Product_2/templates/feature/my_busybox/busybox/config
   ```

   Note that the destination file does not have a proceeding "." (dot). It is **config**, not **.config**.

By creating a separate feature template for your **busybox** customizations, you will be able to include the feature template in future projects without necessarily including the **boa** package and its customizations.

5. Creating Product 2

Create a new project that incorporates your custom configurations.

1. Configure a new product like Product 1, but this time specify the custom layer and templates:

```
$ configure --enable-board=common_pc \
--enable-kernel=small \
--enable-rootfs=glibc_small \
--with-layer=$HOME/layers/Product_2 \
--with-template=feature/boa_mod,feature/my_busybox
```

2. Build Product 2:

```
$ make
```

3. Verify that your custom configurations have been included:
   - View the *prjbuildDir*/**export/dist/etc/boa/boa.conf file** to see that it is configured for port 8080
   - View the *prjbuildDir*/**export/dist/home/httpd/html/index.html** file to see that the new message is included.
   - View the *prjbuildDir*/**build/busybox-***version***/busybox.links** file to see that **wget**, **ps**, and the other utilities you added have been included.
   - View the *prjbuildDir***/pkglist** file to see that it includes **boa**.

Test Your New Product

Boot and test Product 2 using QEMU.

1. Boot Product 2:

```
# make start-target
```

2. Start the Boa Web server:

```
# boa
```

3. View the current process list to verify that **boa** is running, for example:

```
# ps
...
924 nobody    1732 S    boa
925 root      1788 R    ps
```

Note that **boa** is running as the default user **nobody**.

4. Download and view the **index.html** file from the Web server:

```
# wget http://localhost:8080
Connecting to localhost:8080 (127.0.0.1:8080)
index.html           100% |******************************|    56
--:--:-- ETA
# cat index.html
<html><body>boa is listening on port 8080</body></html>
#
```

The file provided by **boa** on port 8080 is your customized **index.html** file.

You can terminate the emulation by entering **CTRL+A**,**X**.

6. Adding a User and Group

Boa requires root privileges to run when listening on any port less than 1024, for example, the default port 80. Since you are using port 8080 it is possible to run Boa as another user.

In this section you will add the user **webapp** and the group **webapp**.

The build system executes **lua** script commands found in spec files, so you can use an existing spec file or create a new one to add the **webapp** features. In this example, you will create a new spec file, **target_admin.spec**, with the appropriate **lua** scripting.

Step 1: Create the target_admin package infrastructure.

1. Create the directory infrastructure for the **target_admin** package in your layer:

```
$ mkdir $HOME/layers/Product_2/packages/
$ mkdir -p $HOME/layers/Product_2/dist/target_admin/
```

2. Add the **target_admin** package to your **$HOME/layers/Product_2/templates/feature/boa_mod/pkglist.add** file so that it looks like this:

```
boa
target_admin
```

Step 2: Create the target_admin package.

You may wish to create a **target_admin** package that in itself adds functionality to your installation, but in this example you will just create an empty package and then use the package's spec file to add the desired features.

Use an editor to create the empty package, or just:

```
$ touch $HOME/layers/Product_2/packages/target_admin-12.0.src.rpm
```

Step 3: Create the target_admin.spec file.

Create (or copy from the provided layer) a **$HOME/layers/Product_2/dist/target_admin/target_admin.spec** file with the following contents:

Example 1 **A Sample target_admin Spec File**

```
Summary: A package for performing target administration.
Name: target_admin
Version: 1.0
Release: 1%{?dist}_WR%{?_ldat_rel}
License: Public Domain
Group: System Environment/Base
Buildroot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
Requires(Pre): setup >= 2.5.4-1

%description
The target_admin package implements a structure for target administration
tasks, such as adding users and groups.

%install
rm -rf %{buildroot}
mkdir %{buildroot}

cd %{buildroot}

%post -p <lua>
wrs.groupadd('-g 200 webapp')
wrs.useradd('-c "webapp user account" -u 200 -g 200 -s /bin/sh -d
/home/webapp webapp')

%files
%defattr(0755,root,root)

%changelog
* Tue Jun  2 10:59:21 PDT 2009 John Doe <john.doe@windriver.com>
- created
```

The **lua** scripting commands to add the group and user and the user home directory are shown in bold.

Step 4: Create the makefile.

Create (or copy from the provided layer) a **$HOME/layers/Product_2/dist/target_admin/Makefile file** with the contents shown in Example 2.

Example 2 **A Sample target_admin Makefile**

```
# Copyright (c) 2008 Wind River Systems, Inc.

PACKAGES               += target_admin

target_admin_TYPE               = SRPM

target_admin_NAME               = target_admin
target_admin_RPM_NAME     = target_admin
target_admin_RPM_DEFAULT  = target_admin
target_admin_DEVEL        =
target_admin_RPM_ALL      = target_admin target_admin_debuginfo

target_admin_VERSION    = 1.0

target_admin_SUMMARY      = Adminster target features with a
target_admin.spec file.
target_admin_DESCRIPTION  = The target_admin package supplies build \
     infrastructure to add users and groups. By use of a (possibly empty) \
     <package>.src.rpm file and associated Makefile and spec file, you can \
     add lua script commands to the spec file. \

target_admin_GROUP        = System Environment/Base
target_admin_LICENSE      = Public Domain

target_admin_ARCHIVE      = target_admin-$(target_admin_VERSION).src.rpm
target_admin_MD5SUM     =  d41d8cd98f00b204e9800998ecf8427e
target_admin_UPSTREAM   = windriver.com

# %changelog
# Tue Jun  2 10:55:06 PDT 2009 John Doe <john.doe@windriver.com>
# created
```

Step 5: Reconfigure and build the project with the modified custom layer.

The **reconfig** target will cause your changes to be picked-up:

```
$ make reconfig
```

Build and verify your changes:

```
$ make
```

After the build is complete, you should be able to see a **webapp** user in the *prjbuildDir***/export/dist/etc/passwd** file:

```
$ grep webapp export/dist/etc/passwd
webapp:x:200:200:webapp user account:/home/webapp:/bin/sh
```

and there should be group **webapp** in the *prjbuildDir***/export/dist/etc/group** file

```
$ grep webapp export/dist/etc/group
webapp:x:200:
```

7. Modifying Boa to Run as Non-root User

By default, **root** starts the Boa Web server which then runs as user **nobody**. In this section you change the user that **boa** runs as to **webapp**. User **webapp** is able to start **boa** because the port **boa** binds to (8080) is greater than 1024.

You will also make the document root of the Web server to be the **webapp** user's home directory. This will allow **webapp** activities to run without root privileges.

NOTE: In the following section you will manually launch the Web server as user **webapp**. Later you will modify the system startup script so that root automatically launches the Web server at startup, changing ownership to user **webapp**.

Step 1: Change ownership of the Web server.

Edit **boa.conf** in your **boa_mod** feature template to change the user/group from **nobody/nobody** to **webapp/webapp**.

Change:

```
User nobody
Group nobody
```

To:

```
User webapp
Group webapp
```

Step 2: Change server default locations.

Also edit **boa.conf** to change the location of documents and CGI scripts to subdirectories of the **webapp** home directory.

Change:

```
DocumentRoot /home/httpd/html
```

To:

```
DocumentRoot /home/webapp/html
```

Change:

```
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
```

To:

```
ScriptAlias /cgi-bin/ /home/webapp/cgi-bin/
```

Step 3: Create the new subdirectories.

You can create the **webapp** home directory and the subdirectories for the document root and CGI scripts with a **changelist.xml** file. Create the file **$HOME/layers/Product_2/templates/feature/boa_mod/changelist.xml** with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<layout_change_list version="1">
    <change_list>
        <cl action="adddir" name="/home/webapp" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/html" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/cgi-bin" uid="200" gid="200"/>
    </change_list>
</layout_change_list>
```

This will create the home directory and subdirectories with correct permissions for user **webapp**.

Step 4: Create a new index.html file.

The **index.html** file that you created earlier will no longer be accessed by the Web server, because you have changed the document root, which is the location that the server looks for the HTML files.

In your **boa_mod** template, create the new location and then copy the earlier **index.html** file to it:

```
$ mkdir -p \
~/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/html/
$ cp \
~/layers/Product_2/templates/feature/boa_mod/fs/home/httpd/html/index.html \
~/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/html/
```

Edit the file.

Change:

```
<html><body>boa is listening on port 8080</body></html>
```

To:

```
<html><body>boa is running as webapp user and listening on port
8080</body></html>
```

Step 5: Make index.html writable by webapp.

By default, the permissions of the **index.html** file do not allow the **webapp** user to write it. You can use the **pseudo** command to view the permissions settings that the file will have on the running target:

```
$ host-cross/bin/pseudo ls -l export/dist/home/webapp/html
pseudo: Warning: PSEUDO_PREFIX unset, defaulting to
/home/user/Builds/new-p2/host-cross.
total 4
-rwxr--r-- 1 root root 83 Jun  5 12:11 index.html
```

Add an entry to the **changelist.xml** file as shown in bold:

```
<?xml version="1.0" encoding="UTF-8"?>
<layout_change_list version="1">
    <change_list>
        <cl action="adddir" name="/home/webapp" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/html" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/cgi-bin" uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/html/index.html"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/html/index.html"
uid="200" gid="200"/>
    </change_list>
</layout_change_list>
```

NOTE: The text in bold is all entered on one line.

7.1 Test the Usermode Web Server

Configure and build your modified project, and then verify that it works as expected.

Step 1: Reconfigure and build your project.

The **reconfig** target will pick up your changes:

```
$ make reconfig
```

Then build:

```
$ make
```

Step 2: Boot the target and test your changes.

Perform the following procedure to test your changes:

1. To boot the target in an emulation, enter:

   ```
   $ make start-target
   ```

2. At the system prompt, become the new **webapp** user:

```
# su - webapp
$ whoami
webapp
$
```

Verify that you are in the **webapp** home directory and have the appropriate subdirectories and permissions:

```
$ pwd
/home/webapp
$ ls -l
drwxr-xr-x    2 webapp   webapp 4096 Apr 27 23:16 cgi-bin
drwxr-xr-x    2 webapp   webapp 4096 Apr 27 23:16 html
$
```

3. As the **webapp** user, start the Web server:

```
$ boa
```

and verify that it is running:

```
$ ps
  PID USER       VSZ STAT COMMAND
 ...
  924 webapp    1732 S    boa
  925 root      1788 R    ps
```

The **boa** Web server is now running as user **webapp**.

4. Test the Boa server. Download the **index.html** file:

```
$ wget http://localhost:8080
Connecting to localhost:8080 (127.0.0.1:8080)
index.html 100% |******************************|   83 --:--:-- ETA
```

and view it:

```
$ cat index.html
<html><body>boa is running as webapp user and listening on port
8080</body></html>
```

The **boa** Web server is listening on port 8080, and serving the **/home/webapp/html/index.html** file that you modified.

7.2 Starting Boa at Boot Time

You can add a startup script to your layer so that Boa is started at boot time.

1. Create the startup script directory in your layer:

   ```
   $ mkdir  ~/layers/Product_2/templates/feature/boa_mod/fs/etc/rcS.d
   ```

2. Create the startup script. Name it, for example, **S1boa**:

   ```
   $ edit ~/layers/Product_2/templates/feature/boa_mod/fs/etc/rcS.d/S1boa
   ```

   and give it these contents:

   ```
   #!/bin/sh

   boa
   ```

3. Make the script executable:

   ```
   $ chmod +x \
   ~/layers/Product_2/templates/feature/boa_mod/fs/etc/rcS.d/S1boa
   ```

4. Reconfigure and build:

   ```
   $ make reconfig
   $ make
   ```

5. Reboot and test:

   ```
   $ make start-target

   # ps
   ...
   918 webapp    1732 S    boa
   919 root      1764 S    -/bin/sh
   926 root      1792 R    ps
   #
   ```

The Boa server was started automatically and is running with user **webapp** permissions.

8. Adding sqlite

In this example, you will use **sqlite** as the database engine. You may prefer another database, for example **mysql**, but **sqlite** has the advantage of requiring virtually no configuration and is small.

A Note on Using Different Database Engines

Other database utilities require more configuration than **sqlite**, but you can set them up in much the same way as you configured Boa. For example, MySQL is configured by the **/etc/my.cnf** file, which you could modify and place in a **$HOME/layers/Product_2/templates/feature/mysql_mod/fs/etc/my.cnf**. Your custom **my.cnf** file would then override the default **my.cnf** file when you configured a new product with your **mysql_mod** feature template.

Adding sqlite

Create a template for your database customizations.

Step 1: Create a new template.

Create a new template for **sqlite**:

```
$ mkdir $HOME/layers/Product_2/templates/feature/my_sqlite
```

Step 2: Create a pkglist.add file.

Create a **pkglist.add** file in the **feature/my_sqlite** directory with the following contents:

```
sqlite
```

Step 3: Create a new project.

You must create a new project so that you can include the new feature template:

```
$ configure --enable-board=common_pc \
--enable-kernel=small \
--enable-rootfs=glibc_small \
--with-layer=$HOME/layers/Product_2 \
--with-template=feature/boa_mod,feature/my_busybox,feature/my_sqlite
```

The **sqlite** package should be included in the *prjbuildDir*/**pkglist** file after configuration.

Step 4: Build the new project.

Build the project:

```
$ make
```

Start the Emulator in TUN/TAP Mode

Boot your actual target or, if you are using QEMU, boot QEMU in TUN/TAP mode.

```
$ make start-target TOPTS='-t'
```

Note that this requires root privilege on your development host. If you do not have root permission or do not have a networked target to test this on, you will not be able to perform the steps in the section *Accessing the Output with a Graphical Web Browser*, p.21. You can, however, still boot QEMU without the **TOPTS='-t'** option and perform the rest of this use case.

Set-up the SQLite Database

In this section you create a sample database "by hand" for demonstration purposes. Your data may come from Web forms and CGI scripts or other means.

NOTE: The sample product layer includes the **sqlite** database file (**employees**) so you can copy it as described below rather than entering it manually.

Step 1: Change user and directory.

Become the **webapp** user:

```
# su - webapp
```

Move to the **cgi-bin** subdirectory:

```
$ cd cgi-bin
```

Step 2: Use sqlite3 to populate the database.

Supply a name along with the **sqlite3** command to create the database, for example:

```
$ sqlite3 employees
SQLite version 3.6.7
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

In this case, the name of the database is “employees”.

Enter the following at the **sqlite** prompts:

```
sqlite> create table offices(name varchar(20), office smallint);
sqlite> insert into offices values('Mary Smith', 101);
sqlite> insert into offices values('Jose Lopez', 102);
sqlite> insert into offices values("Hu Hwang", 201);
sqlite> insert into offices values("Lab",211);
sqlite> .exit
```

(Alternatively, you can use the **employees** file from the provided layer located in the **cgi-bin/** subdirectory of user **webapp**’s home directory.)

The result is a database file named **employees** that contains a table named **offices** with two columns, named **name** and **office.**This is the database that you will access in the rest of this use case.

Step 3: Test your database.

Enter some **sqlite3** query commands to verify that you can read your database, for example:

```
$ sqlite3 employees 'select name from offices'
Mary Smith
Jose Lopez
Hu Hwang
Lab
$ sqlite3 employees 'select office from offices where office > 200'
201
211
```

The database queries are shown working as expected.

Generating HTML

This example uses basic shell commands to create and read a simple database for demonstration purposes. You may want to use a different programming language such as PHP, Python, or Perl to create the applications that populate your database entries and generate output. Note that you would have to add those packages as well as any package dependencies, which may considerably enlarge the size of your project.

Creating a Script to Access the Database

In this section, you write a CGI application to access the SQLite database. The application is a shell script that produces a Web page that prompts for search input, searches for the input in the database, and then outputs a Web page with the results. Example 3 shows the script.

Example 3 **The employee-search Script**

```
#!/bin/sh

echo Content-type: text/html
echo ""
if [ $# = 0 ]
then
/bin/cat << EOM1
  <HTML>
  <HEAD><TITLE>Search by Employee or Office</TITLE>
  </HEAD>
  <HR SIZE=5>
  <H1>Text search </H1>
  <P>
  <ISINDEX prompt="Enter name or office number to search for: "
action="http://localhost/cgi-bin/employee-search
  <P>
  </BODY>
  </HTML>
EOM1
else
/bin/cat << EOM2
  <HTML>
  <HEAD><TITLE>Search results for $* </TITLE>
  </HEAD>
  <HR SIZE=5>
  <H1>Search results for $* </H1>
  <HR SIZE=5>
  <P>
  <PRE>
EOM2

sqlite3 employees 'select * from offices' | grep  -i $*

/bin/cat << EOM3
  </PRE>
  <P>
  </BODY>
  </HTML>
EOM3
fi
```

You can create the shell script shown in Example 3 by entering it with the **vi** editor in the **cgi-bin/** directory of the running target and saving it as employee-search.

Alternatively, create it on your development host by doing the following:

1. Make the **cgi-bin** directory:

   ```
   $ mkdir \
   $HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/
   ```

2. Then either copy it from this document to that location or type in the **employee-search** file in that location:

   ```
   $ edit \
   $HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/employee-search
   ```

   Or copy it from the provided layer:

   ```
   $ cp location/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/
   employee-search \
   $HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/
   ```

3. Add the following line in bold to your **changelist.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<layout_change_list version="1">
    <change_list>
        <cl action="adddir" name="/home/webapp" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/html" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/cgi-bin" uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/html/index.html"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/html/index.html"
uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/cgi-bin/employee-search"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/employee-sear
ch" umode="744" uid="200" gid="200"/>
    </change_list>
</layout_change_list>
```

NOTE: The text in bold is all entered on one line.

Note that in the added line, the **source** field identifies the location of the existing **employee-search** script from the provided layer. If you unzipped the provided layer someplace else, you must supply the correct path to your file script location.

4. To keep from having to re-create the employees database every time you start a new QEMU instance, add the following line shown in bold to your **changelist.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<layout_change_list version="1">
    <change_list>
        <cl action="adddir" name="/home/webapp" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/html" uid="200" gid="200"/>
        <cl action="adddir" name="/home/webapp/cgi-bin" uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/html/index.html"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/html/index.html"
uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/cgi-bin/employee-search"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/employee-sear
ch" umode="744" uid="200" gid="200"/>
        <cl action="addfile" name="/home/webapp/cgi-bin/employees"
source="$HOME/layers/Product_2/templates/feature/boa_mod/fs/home/webapp/cgi-bin/employees"
umode="600" uid="200" gid="200"/>
    </change_list>
</layout_change_list>
```

NOTE: The text in bold is all entered on one line.

Note that in the added line, the **source** field identifies the location of the existing **employees** database from the provided layer. If you unzipped the provided layer someplace else, you must supply the correct path to your database location.

5. Reconfigure, build, and boot:

```
$ make reconfig
$ make
$ make start-target TOPTS="-t"
```

Accessing the Output with a Graphical Web Browser

If you are using an actual target, you should be able to start Firefox or another Web browser and view the files served by the Boa Web server at port 8080 of the IP address of your target.

If you are using QEMU, access the emulation at 192.168.200.15 from a Web browser on your development host.

1. Access the following URL for your **index.html** page:

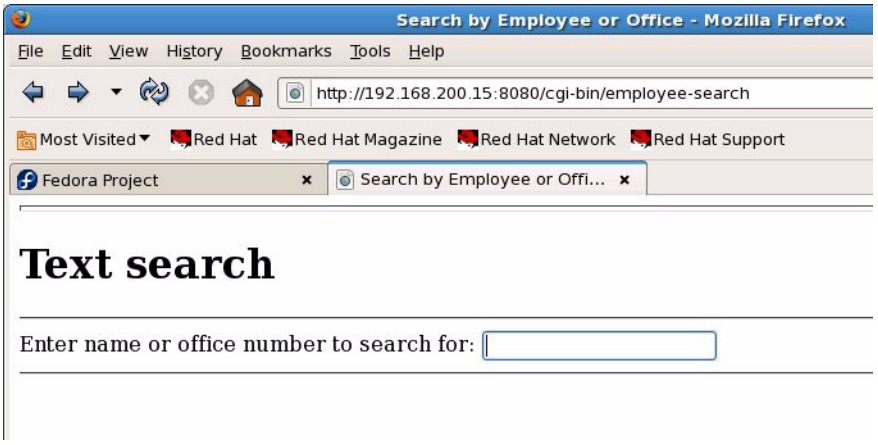
   ```
   http://192.168.200.15:8080
   ```

2. Access the following URL to start your application:

   ```
   http://192.168.200.15:8080/cgi-bin/employee-start
   ```

You will see something similar to Figure 3.

Figure 3 **Web Browser Input to Application**



3. Enter a string to search for—a name or part of a name or number. Then press **ENTER**. For example, if you enter **mary**, the output will appear similar to Figure 4.

Figure 4 **Web Browser Output from Application**

9. Using a default Template

Whenever you include a layer in a configuration, its **default** template, if it exists, is automatically included in the configuration.

1. Create the **default** template and an **fs/** directory with an **etc/** subdirectory:

   ```
   $ mkdir -p $HOME/layers/Product2/templates/default/fs/etc
   ```

2. Create a hosts file that includes your local host information:

   ```
   $ edit $HOME/layers/Product2/templates/default/fs/etc/hosts
   ```

3. Add a **README-Product_2** file to **template/fs/** with the following contents:

   ```
   This installation was produced using the Product_2 layer.
   ```

If you now configure and build a project using the **Product_2** layer, the **README-Product_2** file will appear in the root ("/") directory, and your custom **hosts** file will replace the default **/etc/hosts** file.

10. Additional Resources

The following Web sites are among those that provide additional information on some of the open-source tools mentioned in this use case:

- Apache—**http://www.apache.org/**
- Boa—**http://www.boa.org/**
- MySQL—**http://www.mysql.com/**
- Perl—**http://www.perl.org/**
- Python—**http://www.python.org/**
- PHP—**http://www.php.net/**
- SQLite—**http://www.sqlite.org/**