1-wire C driver (agranet) Communications Protocol Specification

Status

Current Status: As of 10/15/2011this protocol is considered a work in progress

Current Specification Version: v0.0.a

Communication Channels: Communication will occur via standard input (stdin) and standard output (stdout) using the ACSII character set

Commands

Each command issued will consist of the following:

1. The command name in lower case (TODO: Change to upper case)
2. A list of zero or more arguments. Each argument must be separated by one or more SPACE " " characters. There must also be one or more SPACE "" characters separating the command from the first argument.
3. A NEWLINE "\n" character must occur after the command or argument list and signifies the end of the command

Example: <command> (arg1 arg2 ... argn)\n

Command Listing

list - Displays the serial numbers of all devices present / recognized on the 1-wire network
The response will be a new line with a single integer value, n, ranging from 0 - MAXDEVICES. This value represents the number of devices found. After this line, there will be n more lines of text, each line containing the serial number of a device on the network.

get <arg1:device serial number> - get value(s) from the device associated with the specified device serial number. This command expects a single argument in the form of a 16-character string expressing the 8-byte device serial number in hexadecimal. The response will be sent on a single line of ascii text.

Supported Device Families: 0x1D (counter), 0x10 (temperature)

The server first checks to see if the argument is properly formatted. If the argument is not an 8-byte hex value, the response will be "INVALID_SERIAL". Next the, server will check to see if the specified device family is supported by this command (see Supported Device Families section of this command). If the device is not supported, the server will respond with "DEVICE_NOT_SUPPORTED". Next, the server will examine the serial number of each device present on the network and compare it the serial number specified in arg1. If the device serial number specified in arg1 was not found on the network, the server will respond with: "NO_SUCH_DEVICE". Finally, the server will respond with the value of teh specified supported device or with "DEVICE_READ_FAIL" if the device exists but could not be read.

Server Response based on supported device family:
0x10 - a single floating point value with 1 digit after the decimal and up to 5 digits before the decimal (eg: 51.4).
0x1D - A colon separated list of 4 unsigned long integers (eg: 5345:132456:1:536).

GPS Usage

Our current GPS unit in testing is a BU 353 USB GPS

Basic Specs:

SiRF Star III e/LP

GPS protocol

NMEA 0183

SiRF Binary

Transfer rate

4800,n,8,1 for NMEA

The GPS unit outputs NMEA 0183 binary data for a serial connection. In order to get this to work on our routers, you must have ***kmod-usb-serial*-pl2303** c ompiled and installed. This is for the (Prolific PL2303) chipset.

There are two different ways to capture the data. Either you can gather raw NMEA data and print it. With a few scripts this can be used to send the data to multiple clients or you can use a Net Daemon to host the data locally and a client to grab the data and use it however you want.

Method 1 - Standard Raw Output

Required Packages:

netcat (Busybox's nc will not listen on ports)

coreutils-stty (Setting speed on serial ports)

#Use to set serial port speed

stty -F /dev/ttyS0 4800 sane

#Use to print raw data

cat /dev/ttyS0

After the cat command is run you will see a number of strings print out. The string starting with GPGGA defines the location coordinates. With a script we clean out unnecessary strings and feed coordinate data using "netcat"

Method 2 - gpsd Daemon

#This installs gpsd onto router

opkg update

opkg install gpsd

#This will start the daemon which automatically posts data on localhost:2947

gpsd /dev/ttyUSB0

A gpsd client must be used to gather GPSD data and use it effectively. Their are quite a few clients already that use gpsd protocol. I tested tangogps on my Ubuntu box and it works great.

GPSD clients are generally written in C, but can also be written in python, perl,....etc.

In our case, we will use probably write client code in Python, which I believe will require gps.py library.

Sample Python Code for pulling GPS coordinates (Not yet tested)

****

****

import gps, os, time

session = gps.gps()

```
while 1:
    os.system('clear')
    session.query('admosy')
    # a = altitude, d = date/time, m=mode,
    # o=postion/fix, s=status, y=satellites

    print
    print ' GPS reading'
    print '----------------------------------------'
    print 'latitude    ' , session.fix.latitude
    print 'longitude   ' , session.fix.longitude
    print 'time utc    ' , session.utc, session.fix.time
    print 'altitude    ' , session.fix.altitude
    print 'eph         ' , session.fix.eph
    print 'epv         ' , session.fix.epv
    print 'ept         ' , session.fix.ept
    print 'speed       ' , session.fix.speed
    print 'climb       ' , session.fix.climb

    print
    print ' Satellites (total of', len(session.satellites) , ' in view)'
    for i in session.satellites:
        print '\t', i

    time.sleep(3)
```

Home

This is the home of the Information Technology space.

To help you on your way, we've inserted some of our favourite macros on this home page. As you start creating pages, blogging and commenting you'll see the macros below fill up with all the activity in your space.

Recently Updated

Urbanalta Archive
Aug 05, 2013 • updated by Sumit Khanna

AgraStore Web Service Development
Sep 24, 2012 • updated by Anonymous • view change

jenkins-AgraStore.png
Sep 23, 2012 • attached by Anonymous

DeployMan
Sep 16, 2012 • updated by Anonymous • view change

Router Deployment
Sep 13, 2012 • updated by Anonymous • view change

Deployment
Sep 13, 2012 • created by Anonymous

AgraStore Testing Private Keys
Aug 29, 2012 • updated by Anonymous • view change

Web Structure.png
Aug 16, 2012 • attached by Anonymous

Web - Development.png
Aug 16, 2012 • attached by Anonymous

Web Servers
Aug 16, 2012 • created by Anonymous

Deployment Manager (Requirements)
Aug 06, 2012 • created by Anonymous

Installation
May 28, 2012 • updated by Anonymous • view change

AgraSurvey
May 28, 2012 • updated by Anonymous • view change

future-dir-layout.png
May 27, 2012 • attached by Anonymous

current-dir-layout.png
May 27, 2012 • attached by Anonymous

Navigate space

Analysis

- Sensor Project - Initial Analysis

Sensor Project - Initial Analysis

Green Station Sensor Relay Project

Analysis and Notes – Sumit Khanna
V0.5

Introduction

I was originally brought into the Green Station project by Andrew Retting and Michael Bolan. The Green Station is a research project aimed at analysis of water runoff using remote sensor technology. It involves many individual researchers, non-profits and municipal agencies. My roll in the project is to provide my expertise in Linux programming, embedded devices, web services and service orientated architecture.
The following document provides an initial analysis of the work I've done so far and suggestions for architecture considerations going forward. The scope of this document is limited to my role in installing operating systems on the relay stations, integrating USB hardware and designing software to transmit, receive and aggregate data.

Hardware

The primary hardware required includes sensors and some type of embedded computer to retrieve data from those sensors and transmit back to other computers (databases, 3rd party services, etc.) For cost effectiveness we looked at repurposing standard consumer home routers such as the Cisco /Linksys E2000L.

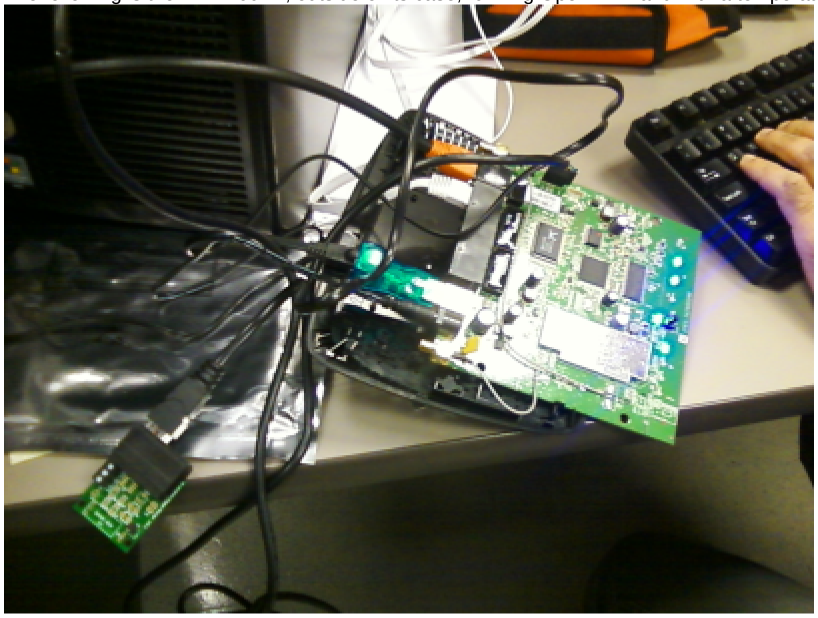
Commodity routers have several advantages over many single board computer systems. Although not as powerful some general purpose embedded boards, they contain a considerable amount of processing power for their price. Mass production for consumer use helps keep that cost reasonable and there is a large base of enthusiasts that repurpose these units using Linux distributions such as DD-WRT, OpenWRT and Tomato.
Initially we attempted to repurpose a Cisco/Linksys E2000L, but its OpenWRT support was still listed as a "Work in Progress" on the community site. Gaowei, A programmer that Michael and Andrew had been working with from China, suggested using the Cisco/Linksys WRT160NL instead and supplied us installation instructions for flashing the router with OpenWRT.
For the sensor, we used a Dallas 1-Wire to USB adapter and a temperature sensor for the initial proof of concept. Other hardware includes breadboards, jumper kits (for prototyping) and USB to TTL-3.3v Serial cable for diagnostics on Cisco/Linksys devices.

Software

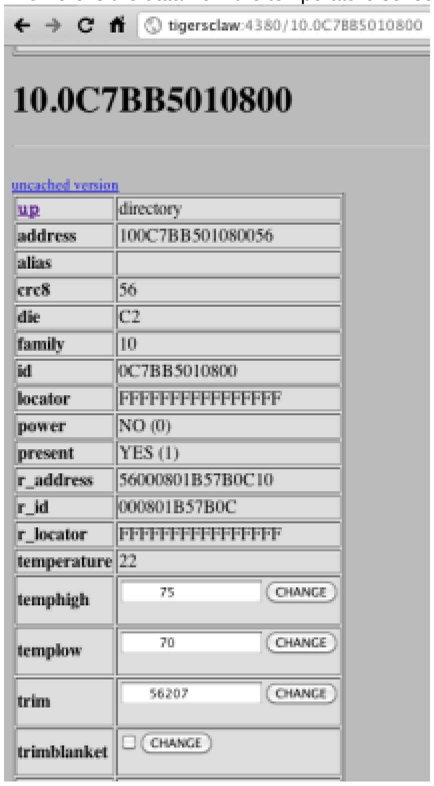
We initially installed DD-WRT on our E2000L router as OpenWRT support was limited. Although DD-WRT is a fully working Linux distribution, it's become more closed in its development and turned into more of a commercial venture. It was difficult to get additional packages for 1-wire software installed and it looked as if packages that did at one time exist for DD-WRT have long since been abandoned.
OpenWRT has a large community of support and a very easy to use build system to cross-compile applications for embedded applications. We were successful in installing OpenWRT on the WRT160NL. We were also able to install One Wire File System (owfs) from the OpenWRT package repository. Once OWFS was running, we were able to plug in our 1-wire USB adapter and immediately read our temperature sensor data.

Working Proof of Concept

The following is the WRT160NL, outside of its case, running OpenWRT and with a temperature sensor plugged into its USB port:



And here is the data from the temperature sensor using the OWFS HTTP server:



tigersclaw:4380/10.0C7BB5010800

10.0C7BB5010800

uncached version

up	directory
address	100C7BB501080056
alias	
crc8	56
die	C2
family	10
id	0C7BB5010800
locator	FFFFFFFFFFFFFFFF
power	NO (0)
present	YES (1)
r_address	56000801B57B0C10
r_id	000801B57B0C
r_locator	FFFFFFFFFFFFFFFF
temperature	22
temphigh	75 CHANGE
templow	70 CHANGE
trim	56207 CHANGE
trimblanket	☐ CHANGE

Terminology

The device we're using was originally a router/gateway. Essentially, a consumer/home router is just a simple embedded Linux device that includes a wireless adapter, two Ethernet ports and a network switch. Although OpenWRT's default configuration is also that of a gateway device (it provides DHCP and Network Address Translation or NAT between the WAN port and the Ethernet switch), this configuration will be changed to turn this device into a specific purpose embedded device.

For clarity, the device, in its repurposed form, will be referred to as a **Sensor Relay** for the remainder of this document.

Data

OWFS provides three ways of accessing the data. One is to use the file system itself; browse to the file representing the sensor and read it. It's important to note these are not real files, but virtual files. Reading from a sensor calls the OWFS API and pulls the data as it is currently from the sensor.
The second way of retrieving data is using the built-in HTTP server. The built in server is designed more for diagnostics or a real time view for users that simply need to read sensors at a given moment. It's not designed for relaying information to other sources or databases.
The third way is to use owserver. OWFS provides a service, with its own protocol, that can be connected to remotely and provide sensor information. Both owfs and owhttp can connect to this server, so it's possible to have owfs running on one computer, pulling remote data from another machine running owserver. It also allows multiplexing as owserver can be the primary system reading from the 1-Wire USB adapter, providing that data to multiple readers.
For our purposes, we will most likely run either owfs or owserver+owfs on the sensor relay. The owserver application isn't part of the existing owfs package we are using, but it is trivial to build and repackage if needed.

Security

Standard Linux security considerations should be put in place on the sensor relays. Although the data isn't very sensitive, precautions should be taken so unauthorized people will not be able to get in and use the devices. The following considerations should be enough to have a reasonable amount of security on our sensor relays:

- Simplified iptables based firewall that allows SSH (port 22) and whatever service protocols we determine are necessary for the sensor data; deny everything else
- Modify the /etc/security/access.conf to only allow SSH connections for our root and user accounts
- Using SSH keys to login to the devices instead of passwords
- Run nessus / nmap on the devices to ensure only the services we need are running

Data Architecture

There is a lot to be considered in regards to handling the actual sensor data. There are two basic ways to handle the data. Data can be processed on the sensor relay itself and transmitted to services straight from the device. Data could also be pushed to a central server (or have the server pull the information) and have a server log and process that information.
The advantage of processing the data on the devices themselves is that each device can be independent and the data immediately available. However, the devices have limited processing power, data can not be aggregated with the data from other devices (unless we make them aware of one another) and adding additional services would mean pushing out updates to the routers.
A central server that either pulls data or has data pushed to it has the advantage of keeping the on-device processing to a minimum, allowing any changes to formatting and data to occur in one central location. However, if centralized services go down or there is network interruption, that data will be unavailable until services are restored.

Sample Rate / Frequency

The rate at which data is sampled is an important factor to software design considerations. The 1-wire system itself is designed for low speed sensors and not for systems requiring high sample rates. Reading the temperature sensor takes less than a second, but there is a significant delay and it is not instantaneous.
Discussions about the sample rate so far have placed it into the one-minute rage. This would allow for more than enough time to process the data on the sensor relay itself. The sample rate would of course be configurable. Benchmarks may need to be preformed to determine the maximum sample rates that can be achieved without a loss of precision.

Timestamps

Timestamps should be in the format 64-bit standard UNIX timestamps (number of seconds since the epoch / January 1st, 1970) for transmission to any of our central servers. Dates and times should always be stored in UTC for our own databases. Time can either be pulled from an external GPS device or using the Network Time Protocol (NTP) to synchronic the sensor relay's system clocks. For external services and formats, they should be translated appropriately.

Programming Language Considerations

More experimentation is needed with the sensor relays to see what languages can be supported. Using a higher-level language such as python would allow a higher degree of maintainability at the cost of some space for the language interpreter and possibly speed. If these become an issue, it is also possible to write applications in C and cross-compile them for OpenWRT on a standard Linux machine.

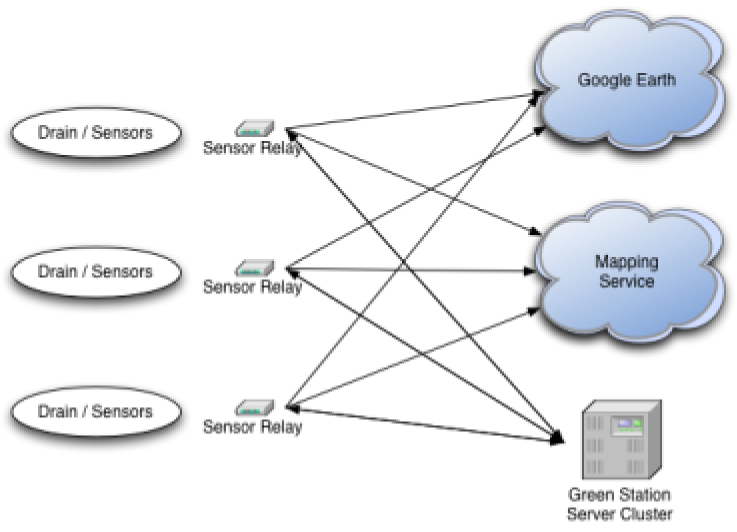
Diagnostic Page

OpenWRT comes with the LuCI web interface intended for standard router configuration. For the sensor relay, this default web server can either be modified to post a customized diagnostic page showing device and sensor information, or it can be removed and replaced with another light weight HTTP service such as lighttpd or nginx.

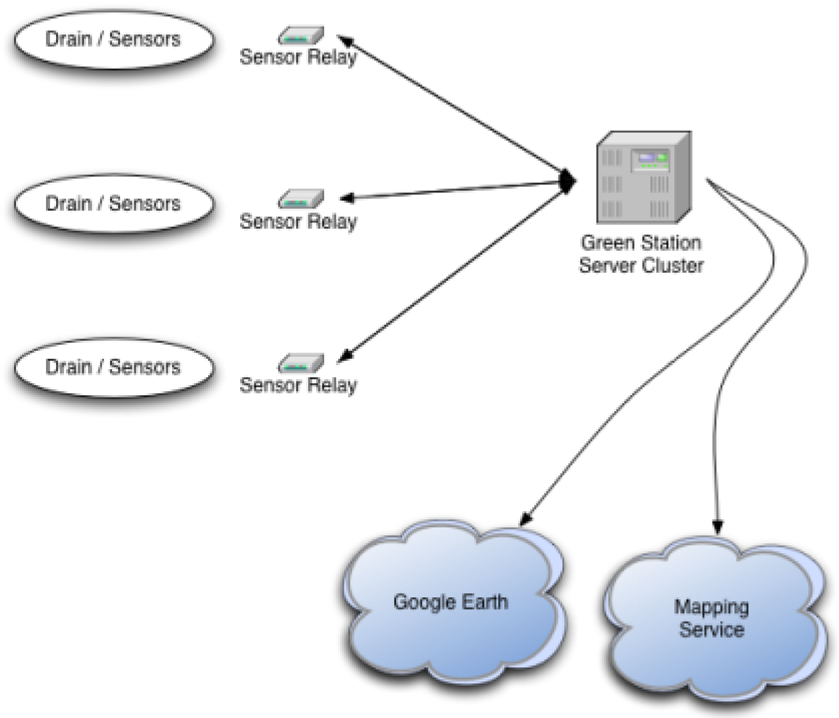
Rolling Logs

If the network connection goes down, information that needs to be pushed or pulled may be lost without some type of buffer. A simple queuing system would place all the data as it comes in to a queue with a given time stamp. As data is processed, pushed or pulled, everything with a timestamp previous to the current would be cleared. Suggestions have included using an sqlite database for the queue.

Direct Transmission



Centralized Aggregation



System Updates

System updates can be pushed to the sensor relays using SSH. Once the prototypes have an established base of software, installations for repurposing the routers should be made as automated as possible. A set of installation scripts should be created that can be re-run at any time to put the sensor relay into a default and current state.
For the initial Green Station project, if an update is pushed that breaks the software to the point where the sensor relay cannot be reset remotely, the device will have to be serviced manually. While this is trivial for the initial project, as the project grows it will be important to have a test set of devices which updates will be pushed out to first and be allowed to run for a predetermined amount of time against automated tests. This will ensure updates that could cause problems are not inadvertently pushed to sensor relays that may be difficult to service manually.

What to Do Next

- Plug in multiple sensors on different USB connectors and make sure OWFS supports multiple USB devices
- Test using USB memory sticks for additional store for application packages and data.
- Experiment with our breadboards and pressure sensors (we have an oscilloscope in the lab. We just need a DC power source and alligator clip wires to test them and make sure they give us pressure readings)
- Repurpose Andrew's applications written for this thesis research for this project to get an initial proof of concept that can be used for demos
- Experiment with USB GPS hardware for Linux for pulling in location and time
- Begin work on a solid software architecture going forward.

Appendix A - Installation

Installing OpenWRT on the WRT160NL is fairly straightforward. We used the OpenWRT 10.03-rc3 firmware image for our router profile.

1. Connect a standard desktop computer running Windows or Linux to the switched ports on the WRT160NL (not the WAN port)
2. Either setup the desktop for DHCP and get an IP address or statically set it to 192.168.1.2 on the 255.255.255.0 subnet
3. Login to the admin screen in the routers web interface by going to http://129.168.1.1 and using the default username and password: admin/admin.
4. Go to the Firmware Update section in the Administration menu and upload the image openwrt-ar71xx-wrt160nl-squashfs.bin
5. Wait for the file to upload. The update will fail to install. Wait for it to reboot.
6. When the router reboots, only the right LED on the router will be blinking

7. Use trivial FTP (TFTP) to upload the OpenWRT firmware to the router
    a. On Linux:
        i. tftp> connect 192.168.1.1
        ii. tftp> mode binary
        iii. tftp> put openwrt-ar71xx-wrt160nl-squashfs.bin
    b. On Windows:
        i. tpfp -i 192.168.1.1 PUT openwrt-ar71xx-wrt160nl-squashfs.bin
8. Wait for the router to reboot
9. Open http://192.16.1.1:3001 to view the OpenWRT LuCI console
10. Login as root (password is blank)
11. Set the root password within the menus and enable SSH access

Installing OWFS on OpenWRT is fairly straightforward as well. Once OpenWRT is running on the device:

1. SSH to 192.168.1.1 (using Putty on Windows or the command line SSH in Linux) with the username *root.*
2. Use the password set in the LuCI web console
3. Run "opkg update" to get the latest package list
4. Run "opkg install owfs" to install One Wire File System
5. Run "mkdir /mnt/owfs" to create the OWFS mount point
6. Run "owfs –u –m /mnt/owfs" to start OWFS and have it search for a USB device
7. Navigate to /mnt/owfs to see the sensor data

Deployment

DeployMan

Introduction

DeployMan is a simple set of scripts that allow for deploying packages, configurations and running arbitrary commands on the Sensor Relays. Currently, it pulls most of its data from a set of files in a configuration directory. In the future, I hope to pull most of this configuration form the database instead to avoid duplication. Currently the only data duplicated between the database and the files are the RSA verification keys.

Configuration Folder Structure

Currently, this folder structure exists at pascal.urbanalta.com:/urbanalta/deploy/relays:

```
|-- configs
|   |-- bioswale
|   |-- dev01
|   |-- pavement-altawater
|   |-- pavement-neptune
|   `-- rainGauge
|-- known_hosts
|-- rsaKeys
|   `-- dev01
|-- sites
|   |-- dev01
|   `-- gls
`-- sshKeys
    |-- dev01
    |-- dev01.pub
    |-- gls
    `-- gls.key.pub
```

Configs

The configs directory contains individual relay configurations. These are the files stored in /etc/agrasurvey/gm.conf on the individual routers and used for AgraSurvey Configuration. In these configuration files, the symbol **%RelayName%** is replaced with the identifier in the **sites** configuration listed below

Known Hosts

File containing SSH key fingerprints for hosts. Each relay must have its fingerprint in this file in order for software to be deployed to it.

RSA Keys

This directory contains site keys used to sign sensor data packages

Sites

Files that contain a listing of relays and their configurations for individual sites. They use a bar (|) separated format:

- <host> | <config> | <Relay Name>

```
192.168.150.10|pavement-altawater|1-PermAsphalt
192.168.150.11|pavement-altawater|2-PermConcrete
192.168.150.12|pavement-altawater|3-Concrete
192.168.150.13|pavement-neptune|4-Paver1
192.168.150.14|pavement-neptune|5-Paver2
192.168.150.15|pavement-altawater|6-Paver3
192.168.150.16|bioswale|bioswale
192.168.150.17|rainGauge|rainGauge
```

The <Relay Name> is used to fill in the **%RelayName%** place holder in the AgraSurvey configuration file after deployment to individually identify the router.

SSH Keys

The SSH keys used to connect to the routers and issue commands. Each site has one set of keys. On the router, these keys must be installed in **/etc /dropbear/authorized_keys**. This can be done using the following command:

```
ssh -o UserKnownHostsFile=<config dir>/known_hosts <hostname>
```

Commands

```
Usage: DeployMan <config direcotry> <site> [deploy-config|install <package>|run <command>]
```

- **deploy-config**: deploys both the /etc/agrasurvey/gm.conf configuration file and the /etc/agrasurvey/key.pem RSA key.
- **install**: installs ipkg files by copying them to /tmp and then running *opkg --force-downgrade install*. Supports wildcards for multiple packages
- **run**: Runs command

Adding a new Relay to an Existing Site

Adding a new relay to a site requires the following:

- Adding the new relay's SSH fingerprint to the known_hosts file in the configuration directory (this is shared between all sites)
- Adding the SSH site key to the relay's authorized_keys file so SSH connections can be made without a password (must be done manually)
- Edit the file sites/<site_name> and add the new relay's configuration entry

Adding a new Site

An RSA site key must be generated for each new site and placed in the configuration directory under rsaKeys/<site name>. It must also be added to the Ag raStore database. This is currently done by adding the entries manually to the DDL directory within the project's source code and pushing it back to Version Control. The new DDLs with the keys will then be loaded into the database.

A new ssh key must be generated as well for each site and placed in the configuration directory under sshKeys/<site name> and sshKeys/<site name>.pub respectively. The public SSH key should be installed manually to each of the routers in the new site.

Finally, a configuration file, sites/<site name> should be setup with all the correct information for the site's relays. New configuration files may be necessary. Configuration files can be shared across sites because the **%RelayID%** is what is used to verify the data signatures from a particular site.

Building into Jenkins

Jenkins is used to deploy each site. Development is deployed automatically whenever any packages are pushed to Version Control that are designed to be deployed to the relays including Python modules and AgraSurvey.

Permissions

The SSH keys must be accessible to Jenkins. Run **chown root:jenkins <key>** for each of the SSH keys and **chmod 640 <key>** on the private key files.

Router Deployment

Initial Setup

Deployment can be setup through Jenkins and the DeployMan project, however some initial setup must be done manually for now on the routers. This setup includes installing necessary dependency packages like Python, Python modules, formatting the USB stick, etc.

Formatting the USB Stick

The USB stick must be formated before use if it hasn't been so already. The preferred file system in **ext4**. It is not necessary to mount this USB drive as AgraSurvey's startup script will do so automatically and create all the necessary directories on it if they do not already exist.

```
mkfs.ext4 /dev/sda1
```

Installing Packages

```
opkg update
opkg install python
opkg install libusb
opkg install distribute
opkg install python-sqlite3
easy_install rsa
```

Setup Name Server

The file /etc/resolv.conf is a symbolic link to /tmp/resolv.conf. Delete this link and create a static file to the Urbanalta DNS server available at your site. The closest DNS server to your router can be found in the Server Lookup Table

```
rm /etc/resolv.conf
echo "nameserver 192.168.160.100" > /etc/resolv.conf
```

Development

- AgraStore Testing Private Keys
- AgraStore Web Service Development
- Deployment Manager (Requirements)

AgraStore Testing Private Keys

Development

Endpoint URL: http://dev-services.urbantla.com/agrastore/api

To understand how to use these keys, please refer to AgraData XML Format and http://penguindreams.org/blog/signature-verification-between-java-and-python/

Development Site 1

Private Key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAu+zlDLOTvkCtrFK5OGQ/LfuiMCYhpzdjPm2SdYoacYXWncLF
G3AKnOkPJjyRmoCrtJZFk/C73azL6kR41r0XUwdlZw7CuN85D0PZqWDdHX4JWgIO
cGMsuHT0AQmidC/X31aWFPtuuIh8U/CGLY+mhtGvnUSumKluM3aKh1+tNZN3dV7j
YhVBZmNFhXq+nyU9/9xBakHm0cDr/sKqKJu4/Lz8UTrlsLDaqEQoChjvpGUbnNyg
FnvzOgAXOqGa5r3MVVmgNU8EQPzJkVZDR7kPMF+qhlqNffYgTfW+DlWESp6d7Q0B
hSCBNDCvDlc/rkAG84P1npgkwCogxyClKkC0lQIDAQABAoIBAQC3JHZQu42noybe
QrxJjcDY0lvMqsyvRtZMV8KdFBsTOYjftJodVEKzipn9/Jc1yGIOG0jxlXGw5p7P
zy0osZ1SiGm1VvhD7R+RleFBJao3/MWmV0y1EKHbnfbSMA6HRr0N5jdbeXH9Gt53
e3d4XX+/8ghAr1esZaqFMuhsZ7zI8SFeWHTMmaVgDtWziE9KxFqlZN8u66x3EPRl
BVakmWjZ36O+yrP1nCsCBOakok/Rng28CRJI/v1kWEzZ0P7mbstgQeDyCUbV/ZLo
hVLkQkmb5o2xwxB4PFC6pw/bkVdSCq49XJLB5BDV5gVTw1sdcfCOfhlLt+pWKmQ0
4my0Ryj9AoGBAPOuqGkxPR0UFnEDE6np7XYwMXZLE/z2nSNEQIQHgg5riEgg2iQ3
24QDzcUR/8TmTbDb7nLw6EMu2d04QUgieYYCiLNC8Yk98qOR3CPx6XBWFHEA07Rw
A/J0B/UWUBWrfXMhwQbD5KPdEmWr35KvbK+e9zxtfu3zZ2Mwz0w/fV2fAoGBAMVs
uKc2+JennSjVH7pAG4uJbegQ3NR6MAoSBUVLGBCw7MIUZh2flYufQ6DIEXQPlz7I
hMiJXTLWgsRpf/0/+iXSHZwW7l2XMHJ9el+Oaw0bCq6k4UCCUCTBvXvvx5Dg0OXS
HpgZWdXF4KnjFHswtRpticlHBsrJU6bV460uxllLAoGAQMDNwb4y1jJLUFaX+BPQ
ZRKjYiLLOfIyiXeOiUcReVF70mbgcLVjIK5+FHsW8zSbun6G24ZGweuGOzHCry9y
CXlM6A3G70hF3M9apzaWaKKHJgwpNY56jgflQWxfdZKvcFOs3mZZsG2DgP7uFyWE
hRqB6k3SZ9rBQ2tp+oH4h1MCgYEAmUffWhyJCB4gHCrUtmO9vynVhl9JRUMU90yk
gPdb2OG1AL6bxhY4Iq5l8HhFbMoKELnYtmZIUQdRgjOzJqo8io7HZIA9U7bl645W
q8hEf2lmctAa/13t4Yv2lTpGxMp4BmeMT2UnZZ174AspxLe9dKZlWzvlHx8O7rKU
UcrGP18CgYEAoA6ub9DIy4aJSydhloDichGSB1crwGwqQxoVCq1vXVAo022BW7rv
QxBgO9Ww2OOZhhFWfQqJymldRHfV1bCAcj7m6K5QorU56M0VrPbzMmsgwl6yuJWV
H6yNsWHTjQkndfj51vABNobTGuvUT91rDwrg5g24SzjQcBksph7Zcbg=
-----END RSA PRIVATE KEY-----
```

Relay IDs:

- DevSite01
- DevSite02

Development Victor

Private Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAqa2nvOSoxG+DzeEhi5EjgNR9Z+/qch7iPzMzsN1kTyC9ATir
0lsMasWHdkVpfuU5vqJ6kFmqUPBlgzgHZzp+LvOd1CS2lOdvMyQSvHH2E3CHk9P6
Ohdw0OgLcmX7Os5larqA2Zq1aeOy3o9IuzYnYTP8ULjCHjvuN4xN1UCx7vj92227
6D2ohUPEWQdGCp4Dfk3YEC6sXfM7aSLscFhXMBDk0AyXD1OUKx0F/DAirufg5k8B
m6Jj0DDxatS35zqbayt23gacFxtzuCgE6SQGTWA/uj95CX893/1idZLUZ80G3sjC
o9wGr8gjkq+clz2Qnw6jrQhQaTlrf9Olo+czWwIDAQABAoIBAC/UfdYndtN4w+TD
M5uOHD3yqC8mWCY3QnuiSU3v+pi7l8vCV90CQWvPprHWOzjEERUF5BrQy8vpGBR3
NM8KlAtULQdiGf27h4MZBhcM4Nr1+6HWPVEmzmx1HNTwnnMjPz9ot22fyMfCoedO
6KetSkiCSPvQORgguLP73uzgQcgk/ojPdUyfUl/UTfr1Z6fl9fGh3WxiWDh4O2CF
GT0IOjxaR7jRjITN7gGDO39qptrFxME3y19dp7MpBpyr0WK6oz5yA7v0jXn8bPVD
rIBTak27750tc+sYTliGEg1TsPVwGrZsjXau2uDARbVeQsO57Y7R+e1znyhsJyOj
Ds27f3ECgYEA15F8rzjutzWOW6xhBPssiU2dJ7wu9uH2w5soQIbqvKMCwepzBevB
z/WjDr4rlvSAhvQ2m2jJNB6DJV+Q4vClTU/jwV1MIkcHavPr/0eoDNVuRLFfALbi
nf8uMSuJEOiNzlqoR5a59qs/K1N+K9bzdgfycq2K9/QcWubhHYi4As8CgYEAyYDC
4B5aLlKxQRuuyB+UGFXiyD4NVg00XtnctyJqtC3SE+NWXj//lpktLqmalstUJtOE
uj2ywlIi8y5msjOVMdKYTPa0osuXAB+2MPn6bXdN8l9N0RXGbQm88sX7Q9PtmV8V
JZ4hrj74IHAjOJLiIOYab10kwtpyoD7J+qPJGbUCgYAWciexdxQkL4drND/F7QFu
Ko66nHWnb0/qfjVqwqGekrquxmLDsxCzPriDyHlxdnQLmVI2TGrm6mZtdc34U2Zi
7nago75Rs2OqV0uzgRqWe4LH1FA4GtLt8Kw2onnbMNvTKM1s5tzmOl6RfFge6Hrh
R2KPXlKiDuEE9hVqpdnDMQKBgFu1GXws9Dgu9HNAaRy1mo400ID4FNO8MMCWgOhs
ZonYbMsWZWGB2Q4uaqzZDf1hTsxY6ghlOi1MepGY5a5m4ZkKUrx8b5vD8tZgmGfT
7ZDSgowzqdPmCCfqDa+kQYYONw3dg19ah9Oo3v7Maos99mKotxFGIAv6pb6etQg4
cLcdAoGBAJQDMdCQ/GKDZlkdqsMmFAvQmrkEhkMzAz3srRhi5HLJTF7Rdrd3VaY0
wazK+Ni7V4Zd2vogd8PYCsygqnRxsaTBk7vYbqBunnp1akqn/rRqbGsJZOXEOzQI
agRANYCDz6jcpzazkxMmaToF+pf6cRV7Ieyup6XpBhlFnLat6+uO
-----END RSA PRIVATE KEY-----
```

Relay IDs:

- VictorDev01
- VictorDev02
- VictorDev03

Development Sumit

Private Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAxK2GpFZRJs7qmk0+/a62HnHuVT18aK+ceA7rlSJFaIgwS8pw
jQvb6NisZSfyPiJi4TQMb3t0U+NO6A2NH5SrTH4JD9b+g1y3UfAAxgDcquCGx/I6
nPxt5NM2C6OC+9nLple4YIhxB3fT/gZoBV+YGZZw/VeNosBlztlxeSyX9wqDwiPa
bT3FT2rYVTojkel9uMTiBu5AZkEq+sPkFXw3LFi+BI7rrpfFDoKF4FV3uH4TsoQx
qcVoPcTMA+Z8oF+WUT4DNu/NrndeBKZ4c4mk3xWHCMPPFlnmZzohUYCjJtZtB9s7
nb2Gxvv6dotmZJ164/rx0v5ZtzU21cfWeZoBuwIDAQABAoIBAEUWI16kx3rXYkdz
bPYVofmN0cd1grcPQOpXa1+Gm1T9yeFFqkWRbd0cB2q9HnW/BHbRHrEmb5VsGnKf
F/yI76c9+pbq0Zp5Phf1M4BaGymXFyEzMG2mqj+gBbMO69rmBYhX9fdK1lFmQTxR
KccBbl9GnkgPsjwCU4DWluqWxIbNOAjNUcTtNzpCkTGuwtBIszGL7UZzJG5StLdr
mitry5R0rZy98Q02LdoMX8ylklVqzcDFAFSiMkCOA6On6EEjGEv6dztGzjljuxLk
0U8/1ITHHg94rDXDjISY6CIUxZgKLhg1SvgsNbjxYQnOjcHtHjD6+MtMzCIu4xur
o6uHd6ECgYEA5tC82EQ90Pk+JbQbGcEkx+H+QVlu1WPZtHnftgLLbafmKJ9g/quq
3rU/RtEq0rk6pFH0L36CeqAeCefxS11vQQtzsHn6s3WUbCwO7OtwxnI3oVx6gCVW
sE1lZDccowmx/Z+onKaPGGEq1NCGb3rWXpbGfVM0r2G8fzyxfP7DQ8sCgYEA2iM9
UsHZJ3CtMATWWcUTRRDBdW1AT83TcL3kxh7P4/3x5h+xl535EI8JsH87jwYIvYYv
Z80iLiNymhojnAbc98XBCRGxjWG1pYr5Pt4fQST0upj8CX2Np9wY/KIl6I1ThFea
GrT1Ri9hz56078p25g5F/og+Y+BAn4y8SFdz29ECgYBXoAcRU1bkbrrZNXbUYC7G
c1VDGeCpwi50BZCYWJdCGeuWoGsCQ8mqosRS7jWDqi5JE5PQNAb05rSArj08j0wd
NWvGI4i5eHnQVymTaA54SAQ2jhUzcPloFPnZAdMtUhDwaBxq3BCXAMxx3ngq+kdH
wZW5Hk1yB3i4FSKMgWs/4QKBgQCFYNa2K2EkBHlgyxRlf2Lw7/XaXxrbsPaAERw5
J83lkfi+xNJJ6oXH4i4ChUldgksF125VAdDMdVA1eZYcaPXjaj9F1FPEJuJyfi84
iYiCxJ3/GlvBUcuzv5hnoJ2dPAy89vN7MPpoF8CuulPX6uwYbtHNeHDtkMjyxZK6
iP6GgQKBgQCX1rOMbEkp60jtBhmAbnyDehNDqfA6NDLTDNhvZpwmynyvTHgiXAtL
vzPTveU8jA6STqUskVtqIsov7I+3WeWwY5gRdiElNFOa9P71/k6Ck6NW78k+cEd6
tIEuhdgckax+I1UevzyAB66wcpgCuxUFPlr+PH1UVHaGLwClZgQ2Ag==
-----END RSA PRIVATE KEY-----
```

Relay IDs:

- SumitDev01
- SumitDev02
- SumitDev03

AgraStore Web Service Development

Introduction

This document is primarily for Parth who will be taking over development of the AgraStore web service and the web interface to it. However, everyone else involved in development should really look at this document to at least be familiar with how the web service is developed. There was a previous project known as the Altaspace Development Environment that attempted to automate the creation of a development environment, however this project was never really finished and it's probably better to create your own environment anyway. This document will go through the process of setting up a full AgraStore development and testing environment and go over some of the basics of expanding on the existing frameworks.

Code Checkout

Create a workspace. You will need to checkout two different applications, AgraStore and the AgraStoreTester. You will need to be on VPN in order to do this.

```
git clone greenboard.urbanalta.com:/srv/git/AgraStore
git clone greenboard.urbanalta.com:/srv/git/AgraStoreTester
```

Dependencies

You will need to install the following:

- Scala 2.9 SDK (Compiler)
- Tomcat6 or Tomcat7
- Apache Ivy
- Apache Ant

The Apache Ivy jar file needs to be accessible to ant. If you're on Linux, it's best to place it in your home directory under ~/.ant/lib. For Scala ant tasks, set the SCALA_HOME environment variable. The ant build.xml checks that environment variable in order to load the Scala ant tasks.

Building

You'll need a property file. I typically place them in the env directory. There are two examples in env/dist for Tomcat and JBoss. I would use Tomcat as I haven't developed on JBoss in a long time and the last time I did, it would crash when loading some of the AOP stuff.

```
appServerLocation=/opt/apache-tomcat-7.0.19/
servletLib=${appServerLocation}/lib
deployDir=${appServerLocation}/webapps
connectionString=jdbc:jtds:sqlserver://host:port/database
deployFile=war
dbUser=username
dbPass=password
dboUser=username
dboPass=password
securityManager=SignatureSecurityManager
environment=development
dmbs=mssql
```

The *appServerLocation* isn't used by the build.xml itself, but is just a reference for the other attributes. The *servletLib* directory is used to build against the servlet API. Everything else should be pretty self explanatory. The build.xml takes these attributes and alters the spring.xml located in the com.urbanalta.spring package. If you change anything in the property file, be sure to do a *clean* before any other tasks. For further information, please see The Glue (Spring Dependency Injection). To build and deploy your project with the given property file, use the following:

```
ant -propertyfile env/<your property file> deploy
```

Database

The way databases are done is the *dmbs* attribute in the property file is configurable for different DB types. Currently, I've only supported Microsoft SQL (mssql). In theory, the only thing needed to support a new database is a new command file (com.urbanalta.agrastore.db.<dbms.commands>) with new SQL statements. There is a DDL directory in the root that's copied into the final WAR file. In this directory you will need to run the 000-Initialization bootstrap manually on your MS SQL server instance. Change the usernames and passwords in the file appropriately and make sure they match your property file above. It's important that the passwords for the two users match up. The *dboUser need to have full permissions to* CREATE *and* DROP *tables while the* dbUser *only needs to be able to* INSERT, UPDATE, SELECT and DELETE *data.*

The rest of the DDL files, 001, 002, etc will be run automatically in sequence. Note that some DDLs are only run for the given environments as define by the *env* variable in the property file. The DDL loader is run as a Servlet Listener which you'll see in the web.xml. There are two listeners that start in order, the first being the Logger and the second being the DDL loader. They only run once, in listed order, when the web application starts.

Testing

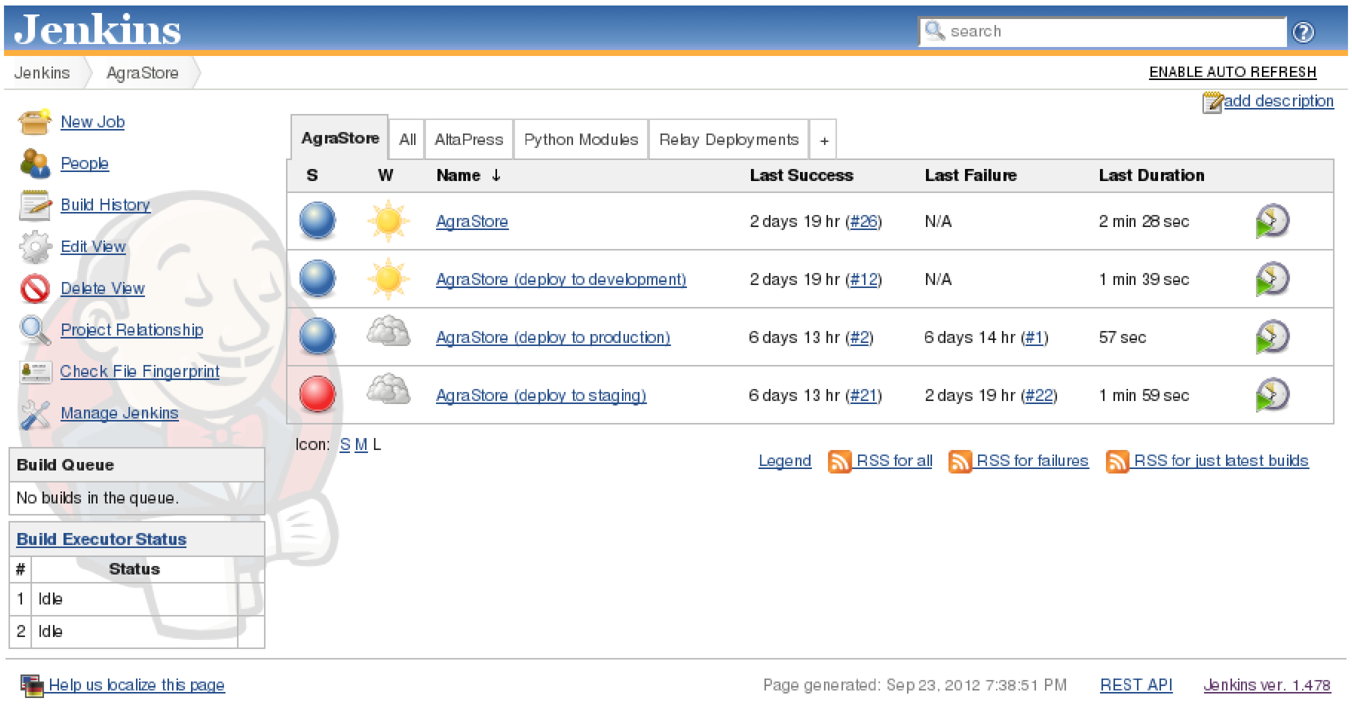
To test your deployment, go to the *AgraStoreTester* project, edit the *config/general.config* file to point to your local Tomcat instance and then run the following:

```
./goc.py AgraStoreMasterTestSet
```

You can also add the *-t* option for trace information. If everything is setup correctly, you should see a slew of OKs. If there are any FAILs, check your logs (by default, located in */tmp/agrastore.log*).

Deployments

Deployments are all automated through Jenkins located at http://greenboard.urbanalta.com:8090 on the VPN. When a git push is used on the AgraStore repository back to the greenboard server, Jenkins automatically starts building AgraStore. It uses property files located on pascal.urbanalta.com:/urbanalta/build/properties for development, staging and production. After it builds, the development version gets pushed to **melbourne**, our development server. If that deploy is successful, a staging deployment tasks begins the staging deploy. Before deploying to staging, it runs the AgraStoreTester on the development web service endpoint http://dev-services.urbanalta.com/agrastore/api. It will **only deploy to staging if all the test cases pass**. The production job must be started manually, but it starts out by running the test cases against staging and will only deploy to production once all the test cases pass.



S	W	Name ↓	Last Success	Last Failure	Last Duration
🔵	☀	AgraStore	2 days 19 hr (#26)	N/A	2 min 28 sec
🔵	☀	AgraStore (deploy to development)	2 days 19 hr (#12)	N/A	1 min 39 sec
🔵	☁	AgraStore (deploy to production)	6 days 13 hr (#2)	6 days 14 hr (#1)	57 sec
🔴	☁	AgraStore (deploy to staging)	6 days 13 hr (#21)	2 days 19 hr (#22)	1 min 59 sec

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Page generated: Sep 23, 2012 7:38:51 PM

Adding Features

Automated testing is essentially for developing a high quality web service of any size such as this one. Whenever adding new functionality, you must add automated tests to deal with both all the success conditions and the error conditions that can be accounted and tested for.

Scala as a Language

The way the current build.xml is configured, both Java and Scala files are compiled within the same build allowing the two languages to work interchangeably. However, it's **highly recommended** to write everything in Scala. Scala adds what has been missing from Java for the past several years and implements many of the things desperately requested in Java 1.7, but that were never delivered. The syntax for properties is better (legacy support can be added, as seen in the code that utilizes Spring dependency injection, with the @BeanProperty annotation), the code is more readable and it's generally just better to work with.

New Data Handler

One of the tasks that has been discussed has been changing out the database backend for a NoSQL data store. The above mentioned environment variable in the property file allows for specifying new DDLs and SQL files for a different RDBMS system. If you want to interact with a non-relational database, you'll have to create a new Database Handler class that extends the existing Data Handler Trait (Interfaces are Traits in Scala; similar but much more powerful). You may need to create a DBO Handler extension as well if your data backend requires scheme to be created or updated (or a blank one if no schema is necessary). These break with the traditional SQL model, so it's best to make these changes directly to the Spring file and add in a new property to switch back and fourth between the traditional SQL backend and a NoSQL backend.

Adding Storage

Another task that has been brought up is adding support for storage backends such as DropBox. There are stubs I have created for this purpose, however I forgot to push them back to the central git repository before I had to send my laptop off for warranty. I will do so once it returns.

Deployment Manager (Requirements)

Introduction

Software needs to be developed for managing and deploying software updates to all our routers. A simple deployment manager was made for the Green Learning Station written in Bash. It was originally installed on *Odyssey*, but has since been moved to the *~~/srv/deploy/DeploymentManager~~* directory on *Sydney* located at the East End Design Centre. It is a very basic set of Bash scripts that preforms the Installation using the old method where all packages and applications are installed to the USB sick. The new Installation is now possible using packages that are automatically built and placed at http://sydney.urbanalta.com/builds/

There are two types of deployments:

1. Deployments made on batches of routers that will be distributed and installed at a given customer site. These would be large scale, multi-unit deployments
2. Deployments to existing routers that are already configured with a unique identifier; that have a known identity.

These two ideas could possibly built into the same application, or they could be two different programs. They are fundamentally different, but may share some common tasks such as deploying and installing ipk files. The following document outlines what I think should be general requirements for both type of deployments. Feel free to make up and edit this document with other requirements that may have been overlooked, or add notes in red for requirements that you think may not be necessary.

General

- SSH keys should be used to securely connect to routers.
- A single public/private key pair can be generated per customer and stored in the database, with the public key being deployed to the routers *authorized_keys* file
- This application or set of applications could be written as a console app or a web app or a web frontend with a service backend.
  - My recommendation would be to write a console python application that uses ncurses and either has its own database or communicates with a REST webservice to data

Mass Deployment

In a university computer lab, deployments are sometimes made to machines by using a disk imagine application and broadcasting an image to ever machine in the lab. Typically, the lab must be disconnected from the university network during this to prevent the rest of the network from getting flooded with broadcast traffic. While this system will not be transmitting the same volume of data or be in broadcast mode, having routes on a separate private network would still be necessary. New routers from Netis come with DHCP enabled. My using a software DHCP server on a Linux deployment machine, each router connected to a closed network could easily be identified by scanning the DHCP logs for assigned IP addresses. One could assume every machine on this closed network was potentially a router that needed to be flashed and configured.

Basic Requirements:

- Have some type of Linux system (small workstation, laptop, embedded device, possibly another router even) have two network cards.
- One network card would be for the Urbanalta network. The second would be a dedicated private network just for attached routers.
- Setup a DHCP server to assign addresses to only the closed network
- Scan the log files for IP addresses
- Attempted to telnet to each router given an IP address to see if it needs an initial password set
- Copy appropriate SSH keys to each router to allow for secure remote administration and updates
- Install ipkg files for all Urbanalta applications and dependencies

Advanced Requirements:

- Have an identification mode where each router is selected and send a command to use GPIO commands to blink an LED.
  - Have a way to label this router (print out a QR code?), assign it a title (or have an option to give it a GUID?) and add the identification information to the database

Standard Deployment Updates.

Standard updates will need to be deployed to existing routers, or we could possibly write scripts to automatically pull updates for routers. Even with automatically updating routers, there still need to be some type of management interface.

Requirements:

- Deploying updates to an entire set of customer routers, pulling the appropriate SSH key from the database for authentication
- Checking the version of all installed software on a given set of routers
- Redeploying an individual router (basic: just our software and packages / advanced: redeploying the core Image and then ssh into it afterwords to complete setup)
  - Note: redeploying the core image has options to protect the configuration files. This will prevent the router from going back into DHCP mode and not being able to connect to it after update

Advanced Requirements:

- In advanced environments where our routers must be on DHCP, query database to learn currently assigned IP address.
- Be able to preform updates in a dynamic IP environment (e.g. preform a full-flash and wait for the router to contact the heartbeat service with new IP and identification information)

Hardware

- Sensors
    - Neptune T-10
    - RAINEW111

Sensors

- Neptune T-10
- RAINEW111

Neptune T-10

References:

http://neptunetg.com/products/meters/t-10-1/

RAINEW111

A tipping bucket rain gauge contains a funnel, two cups, and a switch. The funnel collects water and directs it to one of the cups. The two cups are attached to an axle like a seesaw so that when one of the cups fills and dumps its water, the other takes its place under the funnel. Each time the water dumps, a magnet on the cups activates a reed switch. If the volume of the cups is known, the amount of rainfall for a given funnel size can be determined by counting the number of times that the switch is activated.

Advantages:

- Since tipping bucket gauges empty themselves, they do not need to be manually read.
- Since tipping bucket gauges empty themselves, there is no maximum amount of rainfall that they can measure (i.e. they can not fill up).
- Tipping bucket rain gauges have less "drift" in their measurements. (From the information, it is not clear exactly what this means. Perhaps this is referring to the fact that they do not have to be measured and emptied manually which reduces inconsistencies in the measurements.)

Disadvantages:

- Rain that is gathered but that is not sufficient to tip the cup (e.g. rain collected near the end of a storm or very brief rainfall) is not measured. If this water is not removed or evaporated before the next storm, it will be incorrectly counted as part of that rainfall. This could be mitigated by using a strain gauge to measure the weight of the cups and calculating the amount of water in a partial cup.
- Since it takes a finite amount of time for the cups to switch when one of them empties, during a heavy rain; some water will be lost. This could be mitigated by creating an algorithm that determines the rate of rainfall from the switching rate and accounts for this loss.
- Without calibration and maintenance, measurements will drift overtime as dirt or other substances build up on the cups and other components.
- Snow and ice is a problem since it can quickly cover the funnel and stop any further measurements. When the ice eventually melts, the water will run into the cups and give an inaccurate reading. Heated gauges are available to mitigate this problem. However, we are not interested in measuring snowfall, and melted snow contributes to runoff; therefore, this is not an issue.

In our implementation, we are using the RAINEW111 rain gauge produced by RainWise Inc. The funnel it uses is of the standard size used by most personal weather stations; therefore, we can feed our data to the various associated weather project (Weather Underground, etc.). The gauge measures rainfall in hundredths of an inch increments. A digital counter/indicator comes with the gauge which will give us a good visual reference when setting things up. The number of times the read switch is activated will be monitored using the DS2423 1-Wire Dual Counter.

References:

http://en.wikipedia.org/wiki/Rain_gauge

http://www.usbr.gov/pn/agrimet/precip.html

Software

Our current software technology stack consists of the following

Operating Systems

OpenWRT - Embedded Linux Distribution (used on routers/sensor relays)

openSuse - Linux Server Operating System

Windows 2008 Server

Server Software

ArcGIS - Geographical Mapping

Confluence

Databases

Microsoft SQL Server 2008

MySQL

Programming Languages

Python 2.7

Python 3.0

Scala 2.9 / Java 1.6

Agranet build instructions (1-wire C Drivers)

1. Download and build our latest version of OpenWRT.

When building, make sure that libusb is included in the tool chain libraries

2. Check out the C drivers from our git repo

The 1-wire drivers are in a git repo called agranet

use:

```
git clone <adname>@URBALALTA@odyssey.urbanalta.com/agranet
```

3. Edit agranet/builds/libusblinux/Makefile

You must edit the following makefile variables to the correct paths in your development environment:

- OWPRE = <path to OpenWrt build root>/staging_dir
- PRE = <path to agranet root dir>

For example, the variable defines in the makefile should look similar to this (lines outlined in red MUST be changed to reflect your specific environment):

```
# OpenWrt build paths
OWPRE = /home/ryan/Development/openwrt-trunk1/staging_dir
OWTOOLS = $(OWPRE)/toolchain-mipsel_r2_gcc-linaro_uClibc-0.9.32
OWINCL = $(OWPRE)/target-mipsel_r2_uClibc-0.9.32/usr/include
OWLINK = $(OWPRE)/target-mipsel_r2_uClibc-0.9.32/usr/lib

# Use the OpenWrt Compiler
CC = $(OWTOOLS)/bin/mipsel-openwrt-linux-gcc
AR = $(OWTOOLS)/bin/mipsel-openwrt-linux-ar

# directories
PRE    = /home/ryan/Development/urbanalta/agranet
APPS   = $(PRE)/apps
COMMON = $(PRE)/common
LINK   = $(PRE)/lib/other/libUSB
#SHARED = $(PRE)/lib/general/shared

#CFLAGS = -DDEBUG -Wall -I $(COMMON) -I $(LINK) -DDEBUG -c -static
#LFLAGS = -DDEBUG -lusb -I $(COMMON) -DDEBUG -g -o $@

# Modified for OpenWrt
CFLAGS = -DDEBUG -Wall -I $(COMMON) -I $(OWINCL) -I $(LINK) -DDEBUG -c -static
LFLAGS = -DDEBUG -L $(OWLINK) -lusb -I $(COMMON) -I $(OWINCL) -DDEBUG -g -o $@
```

AgraSurvey

AgraSurvey is the Python2 client designed to run on OpenWRT and communicate with the AgraStore web service.

Source Code (Web): http://sydney.urbanalta.com/cgi-bin/cgit/cgit.cgi/AgraSurvey/tree/

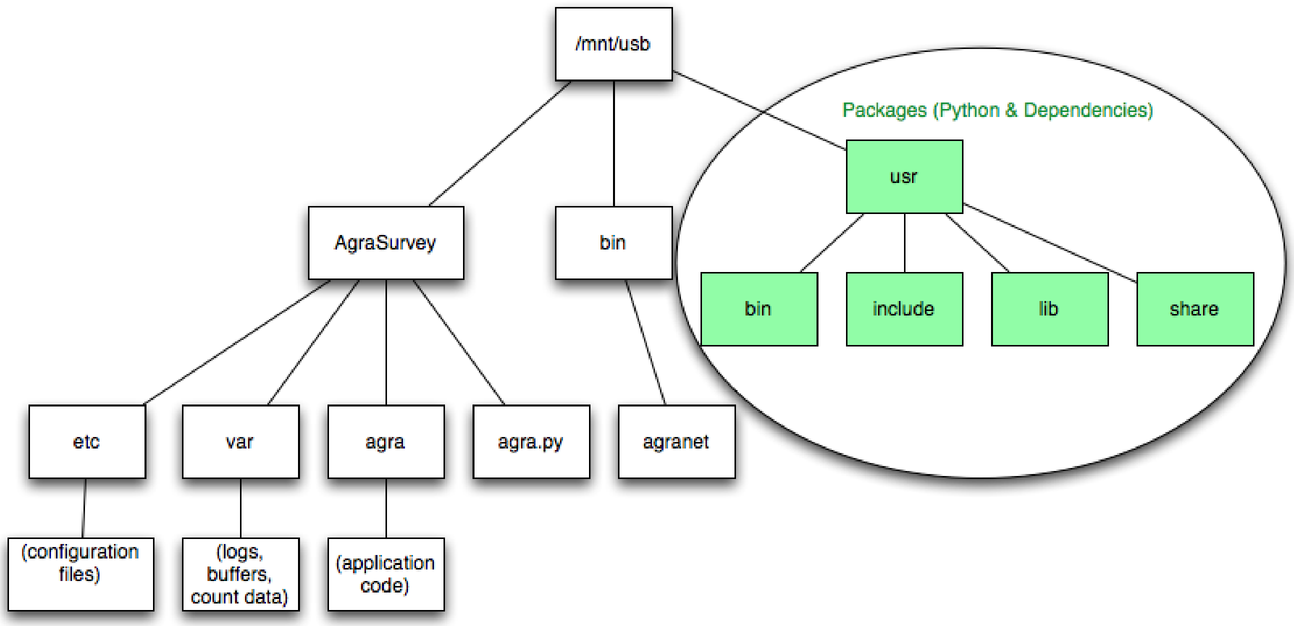
- Installation

Installation

Current

Currently, the production installation of the AgraSurvey application is in a structure based on the limitations of the earlier 4MB file system of the older OpenWRT firmware we used. Because of that, both AgraSurvey and its Python dependencies are located entirely on the USB Flash Drive. The folder layout is as follows:

Please refer to the depreciated documentation for how to install the software using the current setup on the OpenWRT Installation on Netis (NW718) page. Please not that this page is very out of date and is only for reference.

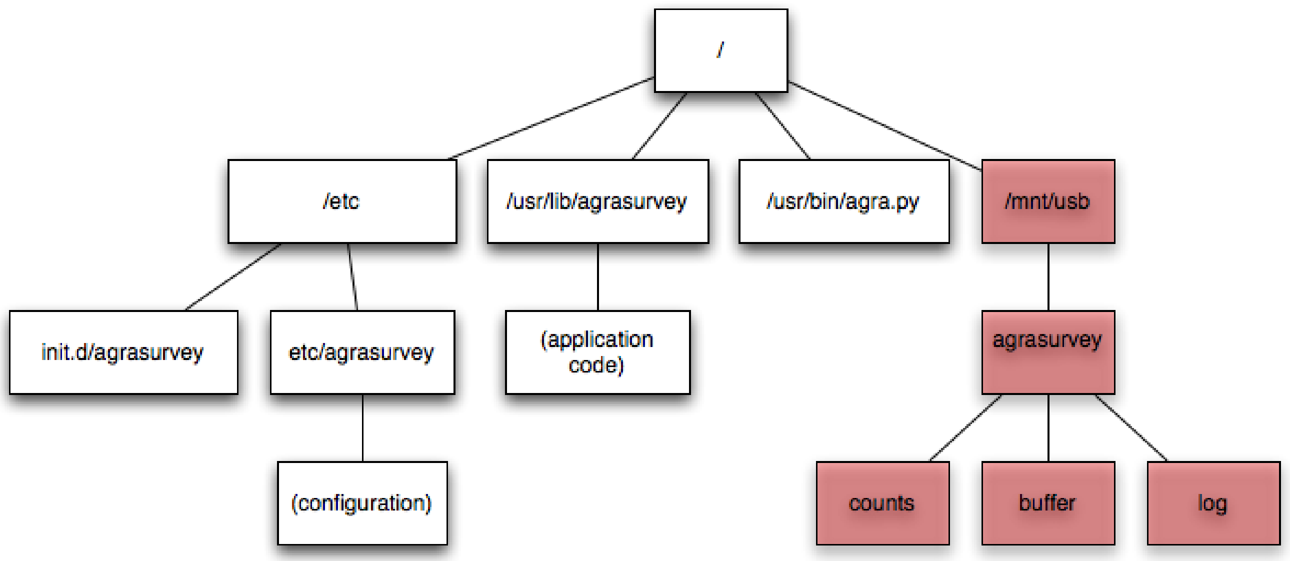


In addition to this structure, there are two ways to start the AgraStore program. One is to call */mnt/usb/AgraSurvey/debug.sh* which loads the application in the foreground with debugging information output to the screen and the second is using the service initialization script */etc/init./d/agrasurvey* which was made by Unknown User (randerso) and mounts the USB drive on router setup, then starts the AgraSurvery application as a background service.

The python and additional dependency packages were originally installed to */mnt/usb* and then archived. This archive is then deployed to the USB drives with the AgraSurvey code. This is not the ideal way to handle installation and was done due to the mentioned size limitation.

Future

In the current OpenWRT firmware, we have 16MB of space to work with. This is more than enough room for AgraSurvey and all of its dependencies on the flash memory of the router itself. The only things that need to be put on the USB storage would be log files and possibly the external buffer file once it's implemented. The following is the idea layout for AgraSurvey, following standard Linux conventions for file locations.



In this structure, the Python packages are not mentioned because they are installed directly to the root filesystem using opkg. Furthermore, the opkg tool should be used to create our own packages of AgraSurvey. We can build these using the Jenkins build tool and have them setup so they can be deployed automatically to development environments.

Smart Vision Analysis

Computer vision, machine learning, statistical modeling, cloud computing.

Web Application Requirements (Draft)

Introduction

This is a basic outline for a requirements and specifications guide for the new Urbanalta web application. Our current application is written in Flex/Flash and works entirely within the client's web browser. We want to move to a server side based application infrastructure that runs on our existing Linux web servers.

Platform

As of now, we (Unknown User (randerso) and Sumit Khanna) are exploring the following frameworks:

- Pyramid/Pylons (Python 3 Web Framework)
- Lift (Scala)
- Spring MVC (Java)
- MVC3 (.NET/Mono for Linux)

Initial Thoughts

Looking at general usability, there are several things I think we should do that diverges from the current web application implementation. Things like live data seem cool, but aren't really necessary and may put and undue burden on the server. Instead we should focus on showing data only on requests. The map is a good idea, but data shouldn't be shown as streaming. Instead each relay should simply be a point and clicking on that point should bring up a popup of the data in a sortable table format within a AJAX box (possibly using the JQuery UI boxes).

Security

Web Servers

Introduction

Web servers can either be accessed directly, through a Load Balancer (HAProxy), or, in the case of a Tomcat Java Application Server, via the AJP protocol typically accessible on port 8009. For Urbanalta, all public access to web servers first goes through the HA Proxy load balancer on Thales. From there it may be connected by proxy to an Apache Web Server or a Tomcat server (typically mapping incoming connections on port 80 transparently to port 8080 on the Tomcat server).

HA Proxy

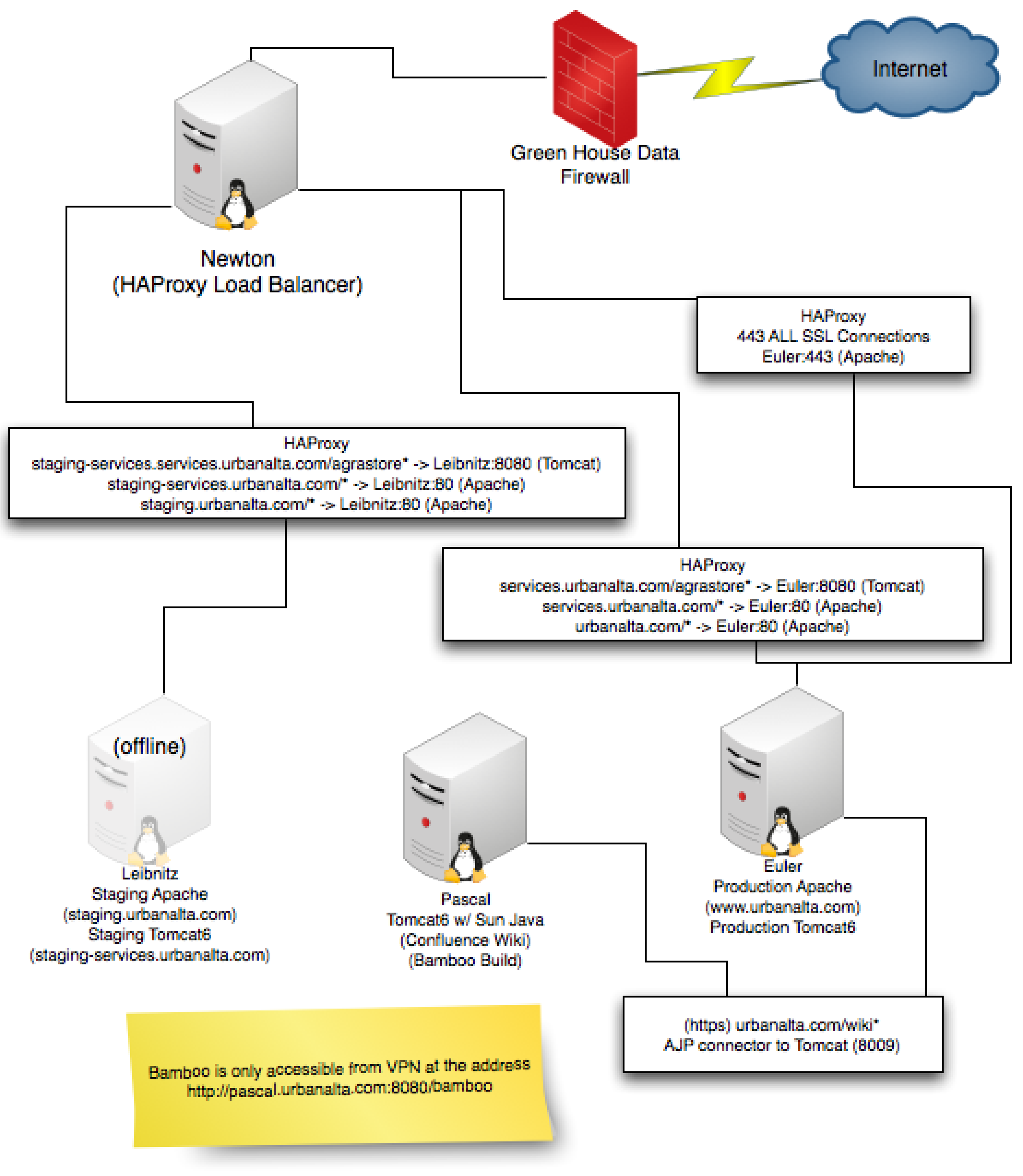
HA Proxy configuration is on *newton.urbanalta.com* and located in ***/etc/haproxy/haproxy.conf***. It's a plain text file and the configuration is very straightforward. Connections are sent to servers baed on their hostname and path specified. All SSL traffic must go to one server as HA Proxy is not capable of SSL off-loading and it passes SSL connections straight through as if they were normal TCP connections.

AJP

Apache Tomcat servers can communicate through their own binary protocol known as AJP. It typically runs on port 8009. Typically, Java servers *"sit behind"* a front end web server. The Java server only runs the application server while the front end web serves up images and content. Typically the front end server maps particular paths, known as *application contexts*, to the Java servlet engine. The apache module is known as *mod_jk*. On the Linux servers, the following must be done for AJP configuraiton:

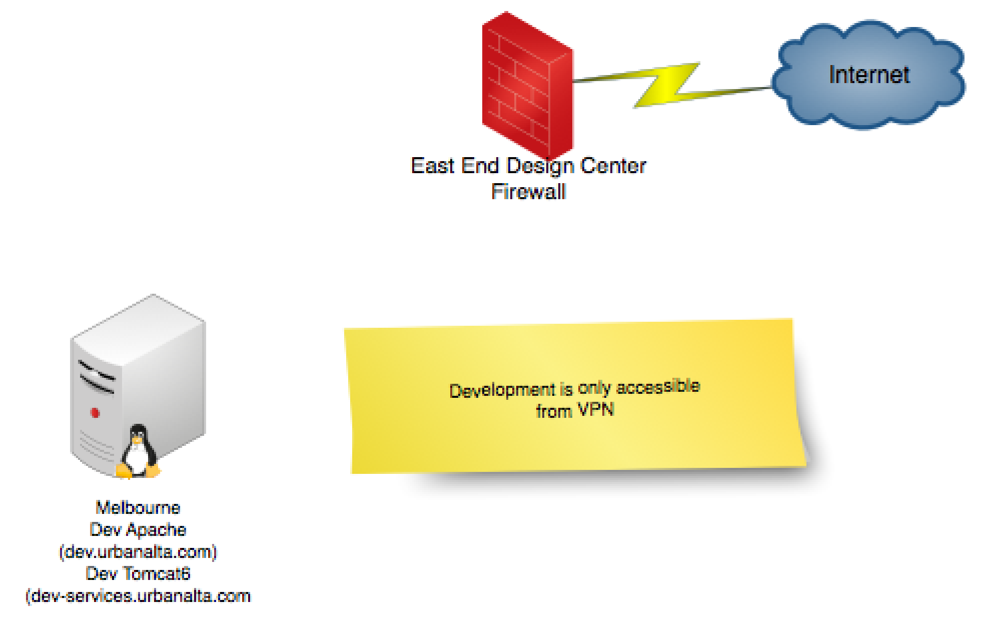
- */etc/apache2/conf.d/mod_jk.conf* - Location of mod_jk configuration file
- */etc/apache2/workers.properties* - List of worker Java servers that the Apache server will forward requests to
- */etc/apache2/vhost.d/*.conf* - The individual virtual host configuration files contain the MountJk

Green House Data Web Systems Diagram


Internet
Green House Data Firewall
Newton (HAProxy Load Balancer)
HAProxy
443 ALL SSL Connections
Euler:443 (Apache)
HAProxy
staging-services.services.urbanalta.com/agrastore* -> Leibnitz:8080 (Tomcat)
staging-services.urbanalta.com/* -> Leibnitz:80 (Apache)
staging.urbanalta.com/* -> Leibnitz:80 (Apache)
HAProxy
services.urbanalta.com/agrastore* -> Euler:8080 (Tomcat)
services.urbanalta.com/* -> Euler:80 (Apache)
urbanalta.com/* -> Euler:80 (Apache)
(offline)
Leibnitz
Staging Apache
(staging.urbanalta.com)
Staging Tomcat6
(staging-services.urbanalta.com)
Pascal
Tomcat6 w/ Sun Java
(Confluence Wiki)
(Bamboo Build)
Euler
Production Apache
(www.urbanalta.com)
Production Tomcat6
(https) urbanalta.com/wiki*
AJP connector to Tomcat (8009)
Bamboo is only accessible from VPN at the address http://pascal.urbanalta.com:8080/bamboo

---

East End Design Center Web Structure


East End Design Center
Firewall
Internet
Development is only accessible
from VPN
Melbourne
Dev Apache
(dev.urbanalta.com)
Dev Tomcat6
(dev-services.urbanalta.com