

BUG Guide

a gentle introduction

Introduction

BUG YT Start Guide

Introduction

Welcome to BUG! This guide will help you get started with BUG Y.T. By the end of this guide, your BUG will be powered on, connected to the network and ready to program!

Intended Audience

The intended Audience for this start guide is new users who are setting up BUG Y.T. the first time. This guide assumes you are using a Linux computer for setup, which is highly recommended for BUG setup. If you are trying to set up with another un-supported platform, please see the Windows and Mac setup pages in addition to this guide.

Connecting Modules the Right Way

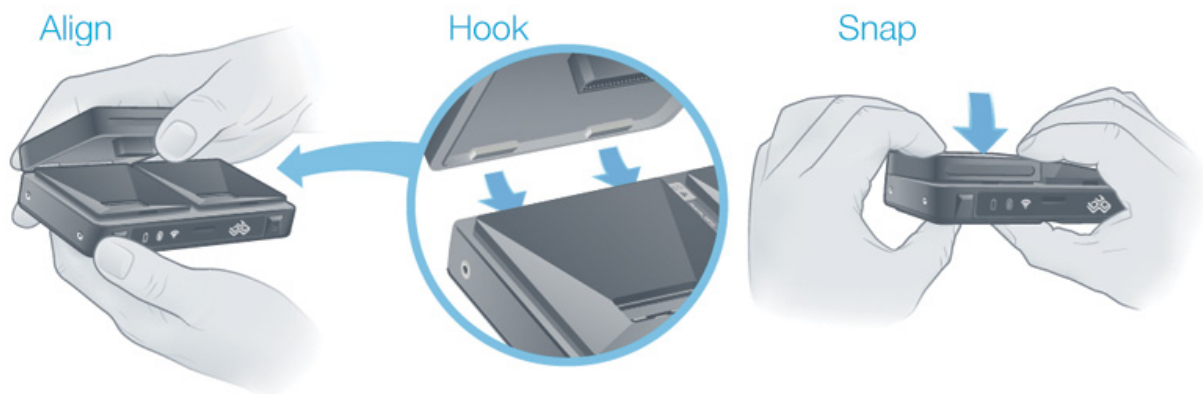
There are 4 slots for plugging in modules, usually identified as Slots 0, 1, 2, and 3.

- As shown in the diagram below, Slot 1 is also the dedicated **Video Slot** where you plug in the LCD module (BUGview2) or the BUGvideo module.
- The other slots are multi-purpose and you can plug any compatible modules into these slots.

The illustration below shows the proper way of connecting modules to BUGbase.

Connecting BUGmodules to the BUGbase

The "Hook and Snap" method.



Safely Removing Modules

To remove BUG modules, reverse the Hook and Snap technique.

Powering Up Your BUG

- Insert A/C power adapter into BUGbase.
- Connect your BUGstinger
- If you have an LCD or Video module, connect it to the dedicated Video slot (Slot 1).
- Press and release the power button on your BUG.
 - The BUG will take approximately 2-3 minutes to boot. The first boot may take a bit longer.
 - While booting you'll see the BUG logo LED light slowly breathing.
- If you'd like to see more information of BUG's boot process in the terminal, follow the Terminal instructions.
 - The default user (**root**) has no set password. If prompted for a password, just hit enter.

Networking

Wired Ethernet Setup

This is the preferred method of getting your BUG on the network. If you don't like our happy path, you can find other ways to connect on the Networking page.

- **Make sure the AC adapter is plugged into your BUG when trying to use the Ethernet port on the BUGstinger.**

Connecting to a wired Ethernet network is fairly simple. This requires a BUGstinger dock which provides an Ethernet port and other connectors to extend the BUG Y.T. base. BUG will auto-detect whether the BUGstinger dock is connected to it and whether there is a live Ethernet cable plugged in.

BUG will try to acquire a network address via DHCP and you should be able to connect to BUG through SSH, VNC, or through Dragonfly SDK which can auto-discover BUG devices on the local area network.

Find your IP address

Run `ifconfig eth0` from a terminal to view your BUG's IP address. This will be helpful when connecting to BUGdash.

Next Step

Once you know your IP address, you can connect to BUGdash

If you prefer to jump into developing with your BUG, you can connect to the SDK

If you already have the SDK Installed, perhaps build your first app following the Getting Started Tutorial or the JavaDoc^[1]

Or simply SSH into your BUG.

Powering Down Your BUG

Hold the power button down for 2 seconds and let go.

The blue BUG light will flash several times. When all lights are off, your BUG is shut down.

Rebooting Your BUG

To reboot your BUG, hold down the power button until the BUG reboots itself.

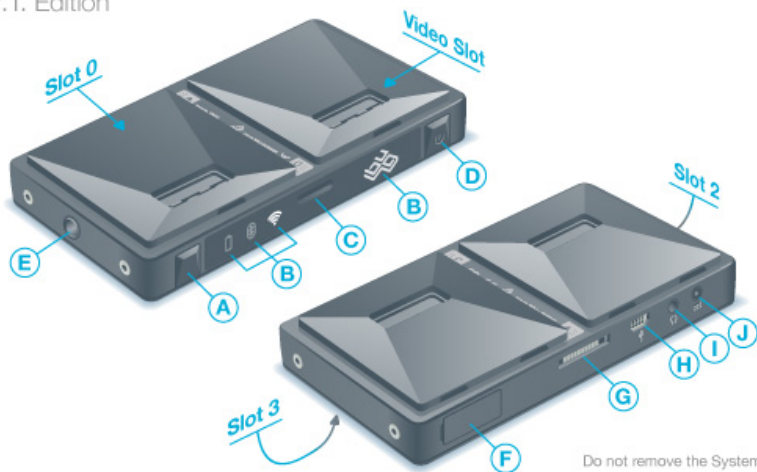
You will know your BUG is rebooting when all LEDs flash on and the BUG logo begins breathing.

BUGbase Diagram

Position your BUGbase so that looks like the first image below. The "Front" image is one which has letters A through D pointing at the set of controls on the front of the BUGbase. You will notice the top and bottom of the BUGbase look the same, so you need to make sure the Tripod mount (identified by E) is also facing you and Slots 0 and 1 are face up.

BUGbase

Y.T. Edition



Front

- A. Select button (user programmable)
- B. Status icons
- C. Open slot for MicroSD user storage
- D. Power button

Side

- E. Tripod mount

Back

- F. System MicroSD card
- G. BUGstinger connector
- H. USB On-The-Go connection
- I. Audio jack
- J. A/C Power adapter

References

- [1] <http://bugcommunity.com/development/javadoc/current/>

BUGdash Admin Tool

Overview

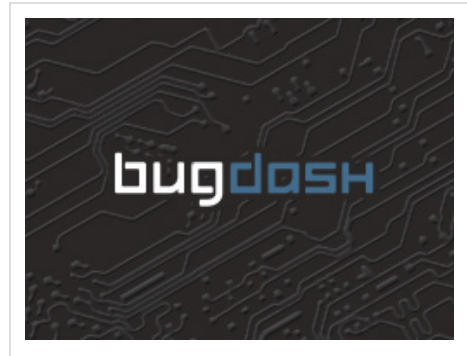
This page introduces **BUGdash**, a web-based admin tool for managing software and viewing device information on **BUG**.

<http://buglabs.net/applications/bugdash2>

Setting it up

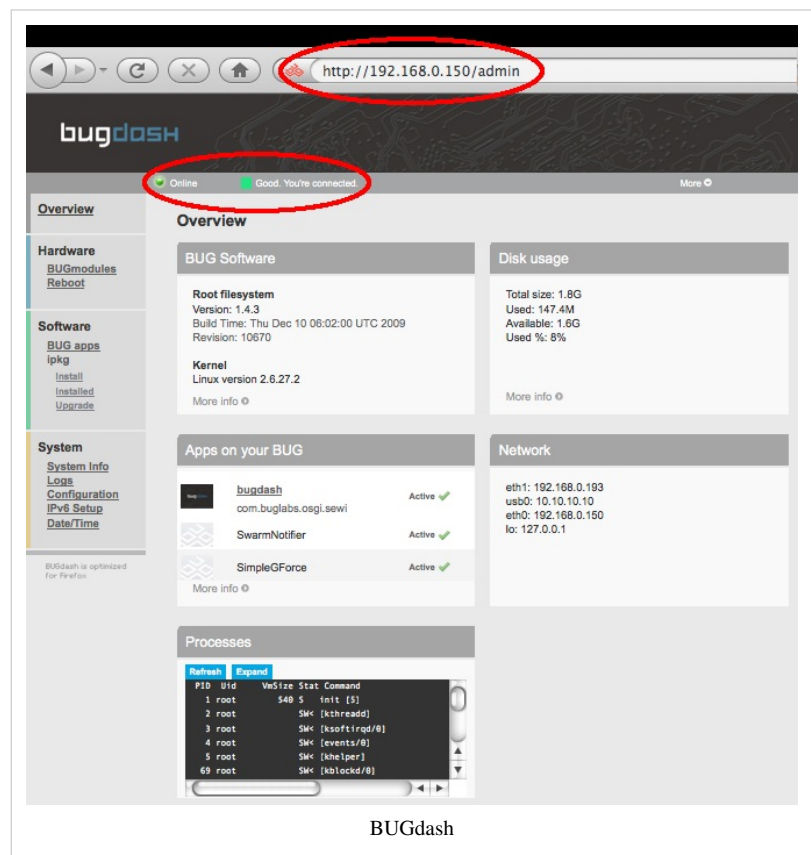
BUGdash comes with BUG Linux, but if you need to upgrade, please follow this instruction:

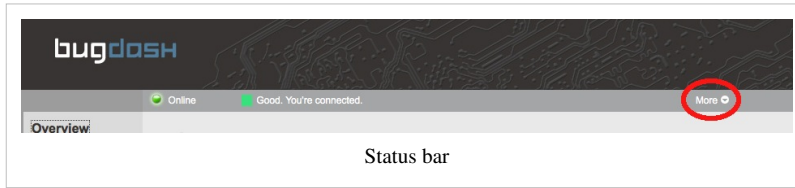
1. Start BUG SDK. If you don't have it installed, please see our Software:SDK Install Guide
2. In the BUGnet view, locate **bugdash2**, download it, and install it on your BUG.



Getting Started with BUGdash

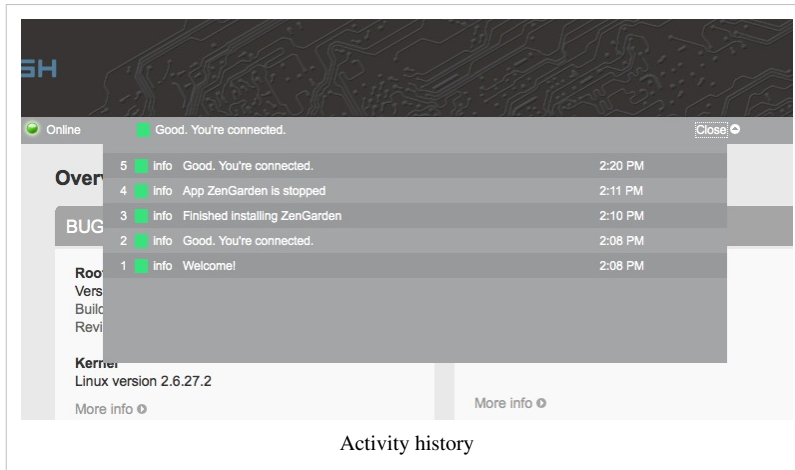
- Make sure your BUG has network connectivity via BUGstinger or WiFi, and determine your BUG's IP. Refer to the Start Page ^[1] if you need help.
- To view your BUGdash page, enter your BUG's IP address/admin into your browser's address bar:
 - **http://BUG_IP_ADDRESS/admin**
- The status bar below the BUGdash logo provides two pieces of information:
 - Network connectivity
 - Green circle with "Online" indicates that your BUG has Internet connection
 - Gray circle with "Offline" indicates that your BUG does not have Internet connection. In this case, those features that require network connectivity are disabled.
 - Clicking the circle will check for connectivity again.
- Activity history
 - The latest activity is displayed in the status bar.
 - Clicking on the "More" link displays the previous activities.





Features

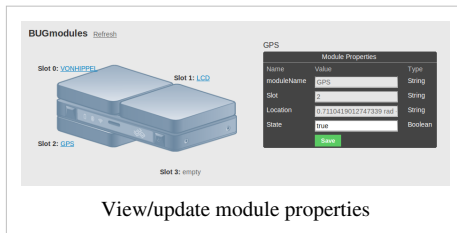
- The Overview page provides a general report of your BUG
 - BUG Software
 - Root filesystem version refers to BUG Linux release version
 - Device Info
 - Rename your BUG
 - Displays disk usage info and battery life
 - Apps on your BUG
 - List of apps currently installed on your BUG and their status
 - Network
 - Processes



Hardware

BUGmodules

- Provides a visualization of which modules are plugged in in which slot
- Allows users to update module properties



Reboot

- Allows reboot remotely. Stay on the same page until a confirmation message appears.

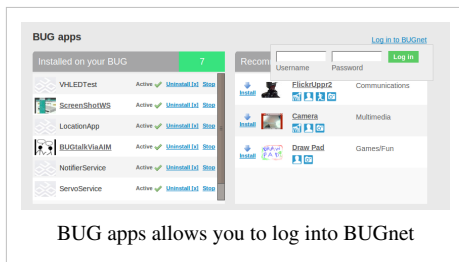
Software

BUG apps

Displays a list of apps currently installed on the BUG and community apps from BUGnet

- Installed apps
 - Displays status and options to uninstall, start, or stop
- Recommended apps
 - Displays a list of featured apps from BUGnet
- Browse apps:

- Allows users to search for apps with a keyword or the current module configuration
- If logged into BUGnet, private apps will be displayed in the list



Known issues

- There is no refresh link for displaying results based on current module configuration
- Program description is displayed in Textile ^[2] markup

opkg

opkg ^[3] is a lightweight package management system and allows us to install, upgrade, configure apps running on BUG easily.

- **opkg - Install**
 - Option 1: Install opkg by name or URL
 - Refer to a list of opkgs in <http://repo.buglabs.net>

```
# single package name
vim

# multiple package names separated by a space
vim tzdata sqlite3

# URL
http://repo.buglabs.net/2.0/armv7a-extras/bzip2_1.0.2-r2.1_armv7a.ipk
```

- Option 2: Provide an opkg file
 - Upload an opkg file to install
- **opkg - Installed**
 - A list of installed opkgs on BUG
- **opkg - Upgrade**
 - Upgrades to the latest BUG software
 - This process can take a few minute. Make sure your BUG is plugged into power.

Known issues

- The percentage doesn't reflect the progress accurately.
- Upgrade may not be successful if you have customized `/etc/opkg.conf`

System

System Info

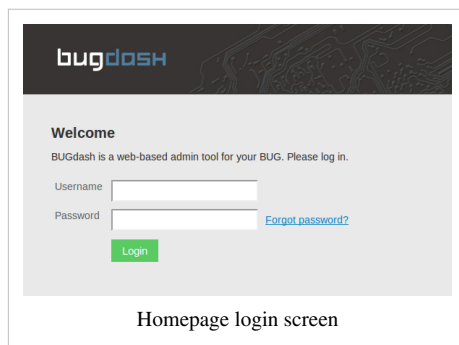
- Displays info regarding kernel, build, drive, memory, CPU, and partition

Logs

- Allows users to monitor concierge and messages logs-- useful for troubleshooting

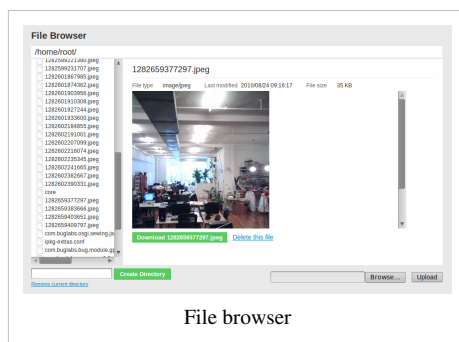
Dash Login

Protect BUGdash by assigning a username and a password. Once you set the credential, you will be prompted to enter your credential in the homepage.



File Browser

Browse file system on your BUG. You can create directories and upload new files.



Configuration

- Allows users to add, update, and delete configuration entries

Date/Time

- Sets date/time and configure timezone on BUG

Contributors

- BugAdmin^[4] by switch
- BUGUpgrade^[5] by jconnolly
- com.buglabs.osgi.sewing^[6] and SimpleRESTClient^[7] by bballantine
- ConfigAdminWeb^[8] by JOcanto
- ShellService^[9] by kgilmer

See Also

- <http://code.google.com/p/opkg/>

References

- [1] <http://buglabs.net/start>
 - [2] http://en.wikipedia.org/wiki/Textile_%28markup_language%29
 - [3] <http://code.google.com/p/opkg/>
 - [4] <http://buglabs.net/applications/BugAdmin>
 - [5] <http://buglabs.net/applications/BUGUpgrade>
 - [6] <http://buglabs.net/applications/com.buglabs.osgi.sewing>
 - [7] <http://buglabs.net/applications/SimpleRESTClient>
 - [8] <http://buglabs.net/applications/ConfigAdminWeb>
 - [9] <http://buglabs.net/applications/ShellService>
-

Connecting

Networking

Overview

NOTE: Make sure the AC adapter is plugged into your BUGbase when trying to use the Ethernet port on the BUGstinger

Your BUGbase has a variety of ways to connect to a network. By default, the bug will connect to the strongest open wireless access point. It will get a dhcp address automatically when connected to any router (both wireless or wired).

By the end of this section, you will be able to connect to the bug via the serial terminal, SSH, VNC, or the Dragonfly SDK. If you would like to set up a manual connection, follow the instructions below, then choose one of the pages above.

Using the Networking Application

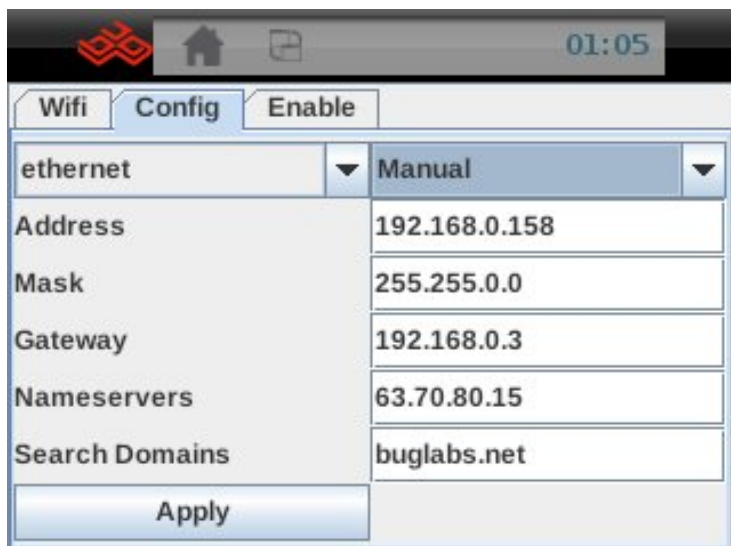
If you have a LCD module, you can use the Network app to configure basic networking. Click the Network icon in the task launcher



Ethernet

Note: Make sure the AC adapter is plugged into your BUGbase when trying to use the Ethernet port on the BUGstinger

Launch the Network app and click on the **Config** tab. Ethernet will be selected by default. Change Auto to Manual and enter the appropriate values



The screenshot shows the Network app interface with the 'Config' tab selected. The 'ethernet' section is active, and the mode is set to 'Manual'. The following fields are visible:

Field	Value
Address	192.168.0.158
Mask	255.255.0.0
Gateway	192.168.0.3
Nameservers	63.70.80.15
Search Domains	buglabs.net

An 'Apply' button is located at the bottom left of the configuration area.

Wireless

Launch the Network App and select an access point from the list.

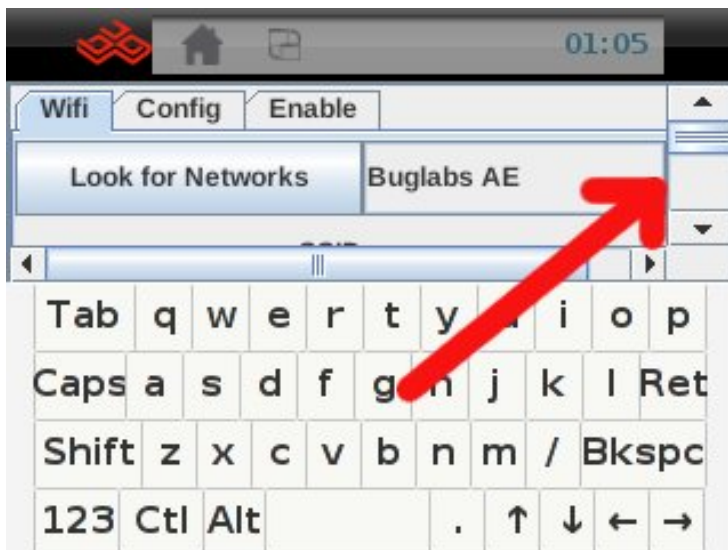


The screenshot shows the Network app interface with the 'Config' tab selected. The 'Wifi' section is active, and the mode is set to 'Buglabs AE'. The following fields are visible:

Field	Value
Look for Networks	Buglabs AE
SSID	
Security	WPA
Passphrase	<input checked="" type="checkbox"/> Show

A 'Connect' button is located at the bottom left, and the status 'Disconnected' is shown at the bottom right.

Hit the **Show** button so you can see the passphrase being entered. Tap into the passphrase field and the keyboard will be brought up. The keyboard will cover the passphrase field, scroll down so the field is visible (arrow points at the scroll bar)



Enter in the passphrase and tap Connect



The app will show if you've made the connection successfully.



To see the Bug's IP address , click on the config tab and change the device selection from Ethernet to Wifi



Using connman the Command Line

Connman has various utilities in `/usr/lib/connman/test`. The most useful ones are outlined below (running any utility with no arguments will display that utility's help):

- **test-connman**: scan, list, passphrase <ID> <PASSPHRASE>, connect are the four commands that will help connect to wireless networks
- **connect-service**: to connect to a hidden WAP
- **set-address**: set manual ip address
- **set-domains**: manage DNS
- **set-ipv4-method**: switch between dhcp and manual addresses, good for ad-hoc networking

See more connman documentation ^[1] for other ways to connect.

Restarting

Many common issues can be solved by restarting the BUGbase network by running `/etc/init.d/connman restart`

Further Reading

- <http://en.wikipedia.org/wiki/Iproute2>
- <http://code.google.com/speed/public-dns/docs/using.html>
- https://wiki.archlinux.org/index.php/WPA_supPLICANT
- <https://wiki.archlinux.org/index.php/Connman>

References

[1] <https://wiki.archlinux.org/index.php/Connman>

Serial Terminal

Background

The BUGstinger is equipped with ethernet, USB-host, and a JTAG/serial USB-mini port. The USB-mini port, when plugged into your host computer, will register two ttys, usually ttyUSB0 and ttyUSB1, depending on your setup. The second one is the serial port, the other is for JTAG (YMMV on JTAG). This document describes how to get started using the serial terminal in Linux.

Plug it in

1. Plug the Stinger side into your BUGbase.
2. Plug the USBmini end into your Stinger, regular (USBA) into host PC.

Connect using Screen

screen is the tool of choice for serial terminal use. Developers will likely be using screen for all day-to-day development efforts because the serial port is more fault-tolerant than say, ssh.

```
sudo apt-get install screen
```

From a terminal, run

```
ls /dev/tty*[uU]*
```

The second entry is what you want to connect to, for example, if you see

```
/dev/ttyUSB0 /dev/ttyUSB1
```

You would want to connect to **/dev/ttyUSB1**:

```
screen /dev/ttyUSB1 115200
```

Hit Enter or Return a few times and you should see the sweet Angstrom shell:

```
.-----.  
|         |                               .-.  
|  |  |  |-----|-----|-----|  |  |  |-----|-----|-----| | | | | | | | | | | |
|         |         |  _  |  ---' | '---' |  .-' |         |         |  
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  
'---'---'---'---'---' |-----' |-----'---'---'---'  
                        -' |  
                        '---'
```

```
The Angstrom Distribution bug20 ttyS2
```

```
Angstrom 2009.X-stable bug20 ttyS2
```

```
bug20 login: root  
root@bug20:~#
```

You can now interact with the BUG as you see fit! Let's see what files are in your home directory:

```
root@bug20:~# ls
BUG_Guide.pdf  README
root@bug20:~#
```

Using Screen

Screen has a great manual ^[1]

Quitting Screen

Press "Control+A", then "Q" to quit.

Troubleshooting

Multiple screen sessions will cause the OS to divert the output from /dev/ttyUSB1 to different processes. Don't do it.

There are two tty's that come up with the Stinger/USB-mini combination. One is intended for JTAG the other is simply a serial terminal. The second tty (should be USB1) is the one you care about for the scope of this use.

Sometimes the device comes up as /dev/ttyUSB2. If screen hangs this is probably why. Try either unplugging and plugging back in the USB plug, or check to see where the node is listed by typing **ls /dev/tty*[uU]*** and hitting tab to see what comes up. Reconnect with screen as above, replacing /dev/ttyUSB1 with /dev/ttyUSB#.

See Also

- Windows Setup Guide
- Mac Setup Guide

References

[1] <http://www.manpagez.com/man/1/screen/>

Connecting with VNC

Overview

This page provides a detailed guide to help you connect to your BUG via SSH and with a VNC client. Some of the BUG programs you need to use to configure BUG run as GUI apps on BUG. Sometimes, these GUI apps are hard to use with the BUG Y.T. LCD because of its limited screen size.

Fortunately, BUGbase Y.T. has a built-in VNC server courtesy of x11vnc^[1]. The trick is to connect to your BUG with a VNC client, so you can take advantage of the larger screen area of your workstation.

Prequisites

Hardware Setup

At a minimum, for initial setup, you will need a BUGbase Y.T., a Stinger, and your development workstation running either Linux, MacOS, or Windows.

You also need a local area network and an extra ethernet cable for the BUGbase. To start, connect your BUG device to your local network through the Ethernet port on your Stinger.

Plug your BUGbase into a power source or make sure the device is fully/sufficiently charged. Turn on the BUGbase using the power switch/button. Now that the BUGbase is on and connected to the network, it will try to obtain an IP address via DHCP.

Using the serial port on the Stinger (and your favorite serial terminal program) Get the IP address assigned to the BUG:

```
bug20 login: root
Last login: Fri Sep 24 21:18:29 +0100 2010 on /dev/ttyS2.
root@bug20:~# ifconfig eth0
root@jconnolly-bug20:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:19:D1:55:97:D8
          inet addr:192.168.0.190  Bcast:192.168.255.255  Mask:255.255.0.0
          inet6 addr: fe80::219:d1ff:fe55:97d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1492  Metric:1
          RX packets:178213 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5086 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24583079 (23.4 MiB)  TX bytes:681751 (665.7 KiB)
root@bug20:~#
```


Software Setup

- SSH client: Windows users will need an SSH client like Putty ^[2]. The remainder of this document will provide instructions and examples for a Unix audience.
- VNC client: A number of free VNC client programs are available for Linux, Mac OS X, and Windows.

Unsupported

Windows: RealVNC viewer ^[3] OSX: Chicken of the VNC ^[4]

Supported

Linux: Vinagre ^[5]

- VNC Server: x11vnc This is already installed in BUG, but does not automatically run on startup.

Connecting to BUGbase

Connect via SSH

First, we need to connect to BUG via SSH in order to run x11vnc. Using your SSH client and the IP address obtained previously, connect to the BUGbase by entering the ip address and login credentials.

```
ssh root@192.168.0.190
```

When asked for the password, you can enter any text (or none at all) and press Enter. This step will also verify that you have the correct IP address and that your BUG is accessible over the network.

Starting Up the x11vnc Server

After logging in, we need to start up the x11vnc server.

```
# Verify x11vnc
which x11vnc
# Run it
x11vnc &
```

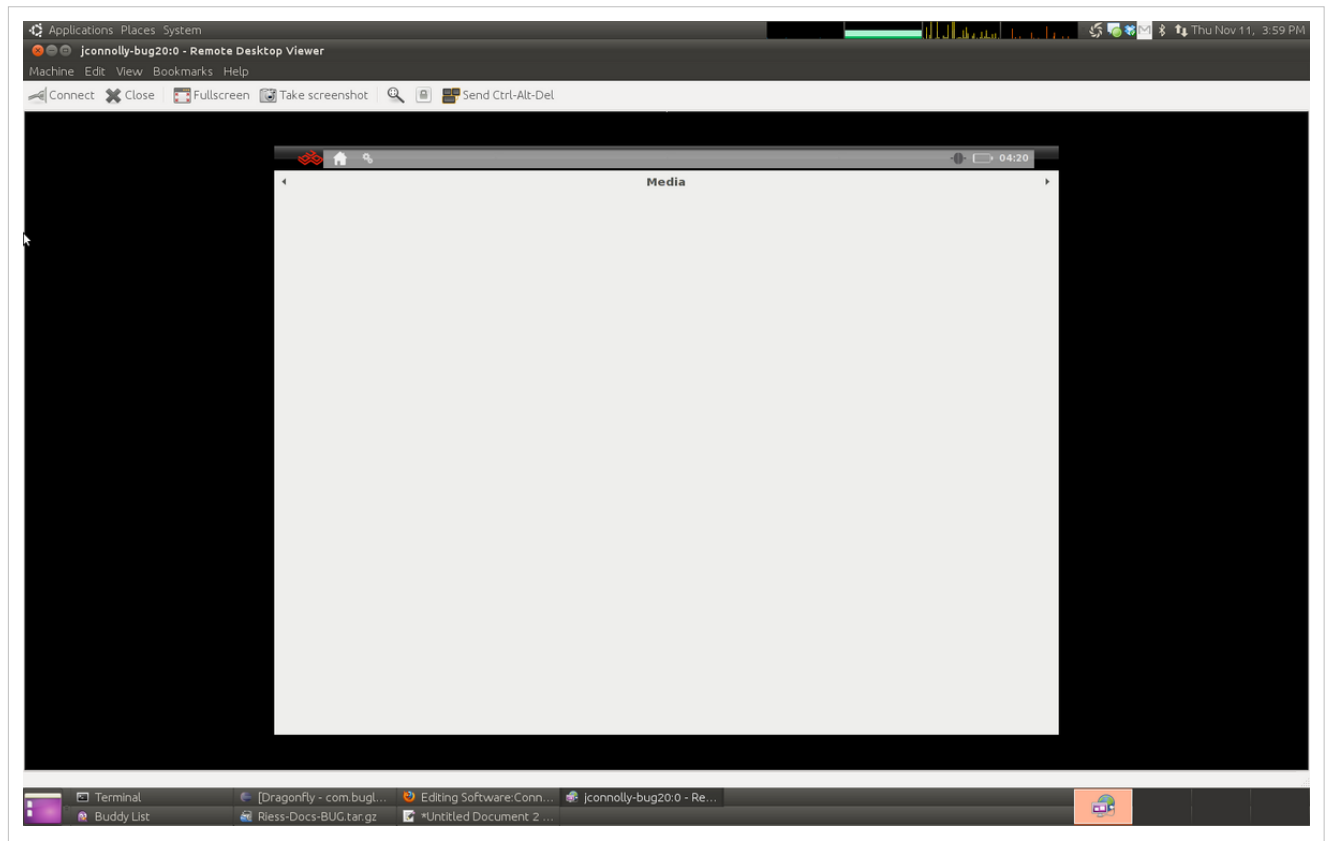
Also, for improved VNC performance, you can try the **ncache** option:

```
x11vnc -ncache 10
```

At this point, you can leave your SSH session open.

Connecting with VNC

Start the VNC client on your workstation and create a new connection to the BUG IP address. If successful, a new window will open showing the BUG desktop as shown below (on Ubuntu using `vinagre vnc` client):



Through VNC, you can control the BUG's keyboard and mouse inputs with your own. With the larger screen on your workstation, you can proceed to configure the Network Manager or use other GUI programs installed on BUG.

References

- [1] <http://www.karlrunge.com/x11vnc/>
- [2] <http://www.chiark.greenend.org.uk/~sgtatham/putty/>
- [3] <http://www.realvnc.com/products/download.html>
- [4] <http://sourceforge.net/projects/cotvnc/>
- [5] <http://projects.gnome.org/vinagre/>

Programming

Develop with Java

Introduction

NOTE: A more comprehensive overview can be found in our OSGi Development Guide

Java is a great language for the BUG. Its large set of available libraries and frameworks, developer interest, and machine independence make it a great platform for building apps. If you are looking to just get started coding, try out the SDK Install Guide.

OpenJDK6

BUG Y.T. ships with `openjdk-6-zero` as the default VM. Its speed and J2SE compatibility and large community make `openjdk` a great choice for an embedded JVM.

OpenJDK represents the future of open source Java. An ongoing community project that aims to port OpenJDK to various ARM-based computers including BUG Y.T. has produced a working jitted JVM. We are actively working on LLVM support (OpenJDK hark project).

More information about `openjdk` can be found here: <http://openjdk.java.net/>

Special thanks to Tarent ^[1] and the JaLiMo ^[2] group for their work in making OpenJDK the stock JVM for BUG Y.T.

BUG Java Libraries

The BUG comes with Java and OSGi libraries for base features and module features. For example, it is possible to program the user button of the BUGbase and toggled LEDs.

Resources

Documentation

BUG javadocs are available online.

Source code

The BUG SVN server (<http://svn.buglabs.net>) contains all of the Java code running on the bug. Most of the project code lives in `/bug/trunk/com.buglabs.bug.*`

Other JVMs

Other JVMs are available for the BUG as packages. Check the BUG package repository ^[3] to find out what is available.

OSGi

Dynamic hardware services and a component-based architecture at the heart of the BUG platform. The standards-based framework we use to make this happen is called Felix ^[4], an OSGi framework implementation.

Writing Java Programs

Since Java programs are machine independent, compiling and running them on a BUG are pretty simple. The JVM on BUG Y.T. is jdk6 compatible, so working against your host computer's JDK should be fine.

Hello World

Compile and Jar this program with your preferred method of building Java programs:

```
public class HelloWorld {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

Once you have your Java application in a Jar, you can copy it to the BUG and run it from a console. This example assumes the BUGbase is connected to your LAN via wifi or the ethernet port on the BUGstinger.

```
pc$ scp hw.jar root@bug20-ip-address:  
pc$ ssh root@bug20-ip-address  
...  
root@bug:~# java -jar /home/root/hw.jar  
Hello world!
```

OSGi applications

The BUG was designed from the beginning to utilize OSGi for easy application development and integration with dynamic hardware services. Thus, building OSGi applications for BUG is very straight forward. As mentioned in many other places, BUG runs an OSGi framework implementation called Apache Felix ^[5]. This framework is executed on boot. The home directory for Felix and the BUG OSGi bundles that provide the base-level functionality of the device reside in `/usr/share/java/bundle/`. User applications typically reside in `/usr/share/java/apps/` and if an osgi bundled is copied into that directory, upon boot, BUG will try to start the bundle.

Dragonfly, the Eclipse-based BUG SDK, represents the easiest path to OSGi bundle development for BUG. Classpath management, application deployment, compiler settings, and other aspects of development are handled by Dragonfly. Dragonfly is based on PDE, Eclipse OSGi tools, known as Plug-in Developer Environment.

Alternatives to Eclipse

We realize not everyone wants to write apps in Eclipse. We use it a lot at Bug Labs but also know that everyone has their own way of doing things. Any method of programming is supported as long as Java Jars can be made with the above-mentioned requirements. Copying OSGi Jars to `/usr/share/java/apps` is all that's required for deployment.

BUG Modules from Java

When creating a BUG application in the SDK, some interesting classpath containers get added to the project. Once called "BUG Libraries" contains the core OSGi bundles that provide access to BUGs modules. These bundles can be found in `/usr/share/java/bundle`. The pattern for hardware support is two bundles for each BUG module. One bundle is used to represent a module's high-level APIs. Typically the name for these bundles is `com.buglabs.bug.module.[module name]`. The second bundle is essentially the Java Native Interface (JNI) representation of the underlying Linux drivers. The naming for these bundles is `com.buglabs.bug.jni`. In most cases, these bundles can be called from regular Java programs as well as OSGi bundles. The high-level API bundles will have some OSGi requirements, but the JNI bundles were written specifically to not require an OSGi runtime.

See Also

References

- [1] <http://www.tarent.de/web/tarent/home>
- [2] <http://wiki.evolvis.org/jalimo/index.php/Hauptseite>
- [3] <http://repo.buglabs.net>
- [4] <http://felix.apache.org/site/index.html>
- [5] <http://felix.apache.org>

Getting Started with OSGi

Introduction

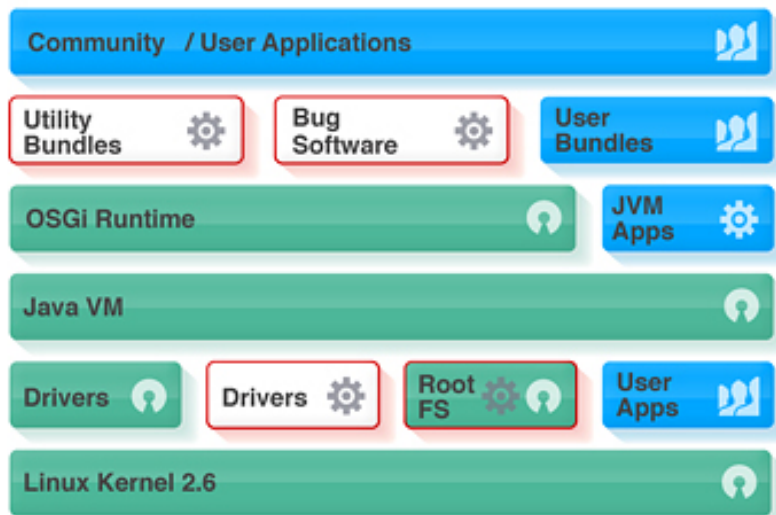
The BUG platform consists of four integrated projects.

- **Hardware** - the BUGbase and dependent modules
- **JVM/OSGi Stack** - the primary software stack on the BUG
- **Dragonfly (the BUG SDK)/Virtual BUG** - a plugin for the popular Eclipse IDE and emulator for the BUG platform
- **BUGnet** - online service for applications

Hardware

The BUG platform consists of the BUGbase, which is a standard mobile GNU/Linux computer, and the associated BUGmodules. Each BUGmodule provides one or more services^[1] to the applications running on the BUGbase. All of this hardware is open source, schematics and detailed documentation are available on the wiki^[2]. Starting out the most important thing for the developer to know are the capabilities of the BUGbase and each module. Note that in contrast to most mobile platforms, applications need not be accessed through a GUI. Though the BUGview module provides a touchscreen LCD service, the base itself allows for users to interact with applications via four hotkeys, a four-way joystick, and two buttons, while information can be output via the base's status bar and speaker. The list of expansion modules is growing, currently this includes GPS, camera, motion sensing, accelerometer, touchscreen LCD, audio, wifi, and low-level input/output (Von Hippel). The base also provides a mini-USB port for other peripherals, and there is a breakout cable for an Ethernet connection for internet connectivity. The entire file system including kernel is on a removable flash memory card for flexibility, on a standard ext3 filesystem.

JVM/OSGi Stack



The recommended platform for BUG applications is the JVM, running with OSGi on top. The stack currently consists of the Java PhoneME Advanced Personal Profile (Java 1.4 based) JVM, and the Concierge OSGi framework (OSGi Spec R3). Though developers are not forced to stay within the JVM/OSGi framework, it provides numerous advantages to writing outside of the BUG stack. First and foremost, all of the device drivers are abstracted out to standard Java interfaces. This allows the details of implementation to be hidden from the application writer. For example, no knowledge is needed of GPS protocols to gain a position fix, simply use the `getLatitudeLongitude()` method of the `IPositionProvider` interface to receive the coordinates at your current position.

This is fairly standard for a Java based mobile application. However when coupled with OSGi, things start to get interesting. The OSGi framework allows for the loading and unloading of *bundles* during runtime. These bundles provide and consume various *services*. This technology enables the modular nature of the BUG. In order to be able to hot plug modules when the BUG is running, you need a way to hold programs from trying to execute before modules are available - and to stop when modules are removed. OSGi handles this life cycle management seamlessly from the application developer. OSGi also enables several applications to use one service, such as the LogService or HttpService, without having to load multiple instances of the service into memory. Perhaps the most important advantage of this approach is that it makes the application dependent on services rather than specific modules. You write an application to use an interface, not any specific module. Today there is the BUGlocate GPS module, but in the future GPS may be integrated to the base, or several service may be integrated into a single module, or an entirely different technology may be used to provide position information. So long as your application is written to the IPositionProvider interface, your application can execute getLatitudeLongitude() and the OSGi framework handles finding the right device. This also allows software and hardware to be accessed on an equal footing. Your IPositionProvider can be implemented by a hardware module, a piece of hardware on the base, or a bundle that reads a log file of positions and streams them out - and the OSGi framework will provide it to the application seamlessly.

Finally, the JVM stack allows for multiple languages. Most of our code is in Java, but Jython, JRuby, Scala, Groovy, and any other language for the JVM should be available with minimum fuss as well. Additionally C/C++ code can be accessed via JNI as usual.

Dragonfly (the BUG SDK)/Virtual BUG

Dragonfly (the BUG SDK) is a plugin for the popular Eclipse IDE, with useful features for both end users and developers. It provides a *perspective* in Eclipse, which is a default set of *views* and toolbars for BUG application development. All the regular tools in Eclipse are present, such as the Debugger and CVS repository browsing. For more information on Eclipse look at this tutorial ([link](#)).

- Virtual BUG - an emulator for the BUG platform. Currently it runs from the VM up, although shortly it will provide full virtualization of BUG via QEMU. It allows for the development of applications without the target hardware, or to debug applications using the Eclipse IDE. It also allows end users to try applications without having to copy them over to their BUG.
- BUGnet view - integrates the BUGnet web service with the BUG SDK. This allows users and developers to download, upload, and search for applications on BUGnet through Eclipse directly, without having to import or export archive files.
- MyBUGs view - shows the currently running Virtual BUGs and physical BUGs on your local machine and your network. For each BUG it lists the currently installed modules, applications, and available web services.
- BUG Application Wizard - manages the OSGi framework for the applications developer

BUGnet

Currently BUGnet is used as a repository for applications. Anyone can upload and download the applications through the web or Dragonfly (the BUG SDK). This allows developers to share applications with other developers and users.

BUG Services

Web Services

The BUG exposes several services through HTTP calls in addition to OSGi interfaces using a standard XML format. This allows programs of all languages running outside of the standard framework to interact with the modules and

services. To access the interface for the Virtual BUG, start the Virtual BUG in Dragonfly (the BUG SDK) and navigate your browser to <http://localhost:8082/>. With a real BUG the default address is <http://10.10.10.10/>. That URL will show the screen below, which allows access to a list of the available web services, modules, and applications on BUG.

For a list of modules, navigate to <http://localhost:8082/module>

```
<modules>
  <module index="1" name="CAMERA"/>
  <module index="3" name="MOTION"/>
</modules>
```

An individual module listing is available by each modules index number <http://localhost:8082/module/3>

```
<module name="MOTION">
  <property value="MOTION" type="String" mutable="false" name="moduleName"/>
  <property value="3" type="String" mutable="false" name="Slot"/>
</module>
```

To get the list of services, navigate to <http://localhost:8082/service>. This will provide a listing similar to the one below. This listing is for a BUG with a motion sensor and a camera.

```
<Services>
  <Service description="Returns the last time motion was detected" name="Motion">
    <Get parameters="" returns="text/xml"/>
  </Service>
  <Service description="A web service message board." name="Statusbar">
    <Get parameters="" returns="text/plain"/>
    <Put parameters="[Message]" returns="text/plain"/>
  </Service>
  <Service description="Retrieves image from camera module." name="Picture">
    <Get parameters="" returns="image/jpeg"/>
  </Service>
</Services>
```

Navigating to the Name of the service gives access to that service, for example <http://localhost:8082/service/Picture> will take a picture with the camera and return that in JPEG form to the requester.

A list of programs is available in the same form as the two above, though the listing is large. In that case the individual programs are indexed by their id number, and a GET request will return a .jar archive of the program.

You can also register your own applications as a service on equal footing with those provided by the BUG modules themselves. The interface is the `PublicWSProvider`, and is detailed below.

OSGi Console

OSGi provides a console for handling the bundles running in the framework. The commands available are listed below.

- **bundles** [*filter*]: Returns a list of bundles present in current OSGi runtime environment.
- **consumers** [(*bundle id* | *bundle name*)]: Returns a list of services that each bundle consumes is currently consuming.
- **exit**: Stop all bundles and shutdown OSGi runtime.
- **gc** : Collects the JVM garbage.
- **headers** *bundleId* [*filter*]: Returns the headers for a given bundle.
- **help** [*filter*] : Print table of currently available commands.
- **install** *bundle_URL* : Install a bundle from the given URL.
- **printenv** [*filter*] : Returns the system settings.
- **printlog** [*filter*] : Displays log messages.
- **producers** [(*bundle id* | *bundle name*)]: Returns a list of services that each bundle is currently offering.
- **quit** : Stop the shell bundle.
- **restart** : Stop and restart all active bundles.
- **services** [(*bundle id* | *bundle name*)]: Returns a list of services present in the runtime.
- **start** (*bundle_Id* | *bundle_name*) : Starts the specified bundle.
- **stop** (*bundle_Id* | *bundle_name*) : Stops the specified bundle.
- **uninstall** (*bundle_Id* | *bundle_name*) : Uninstalls the specified bundle.
- **update** (*bundle_Id* | *bundle_name*) : Updates the specified bundle.

Basic Anatomy of a BUG application

Activator

- **Activator.java** - Every application created in Dragonfly (the BUG SDK) by the new project wizard includes an **Activator** class. When the OSGi framework loads, the *Start* method is called, and upon closing of the OSGi framework *Stop* is called. This is the equivalent of the *public static void main(String[] args)* method in most Java programs. Anything called from this method will run for the entire lifetime of the bundle.

ServiceTracker

- **ServiceTracker.java** - Most applications will consume or provide services, in this case there will be a *ServiceTracker* class. In this case the *Activator* class will not have to be modified. Upon initialization the *Activator* class will setup the *ServiceTracker*, which runs for the entire lifetime of the bundle. The *ServiceTracker* lists what services the bundle needs to start in the *initServices()* method. Once those services are available, the OSGi framework calls the *doStart()* method in the *ServiceTracker* class. If one of those services is lost, or the bundle is closed, the *doStop()* method is called. Thus in these type of applications *doStart()* in your *ServiceTracker* class is the equivalent of *public static void main(String[] args)*. Much like the *Activator* class, the *ServiceTracker* class is primarily boilerplate generated by the New Project wizard. Services selected in the wizard will be added to the *ServiceTracker* class appropriately and the *start()* method of the *Activator* class will be setup to start the *ServiceTracker*. The only methods you will have to edit are the *doStart()* and *doStop()* methods of the *ServiceTracker*.

Manifest

- Manifest.mf - the import-package and export-package properties are set by the new project wizard, but any changes to OSGi dependencies will have to be reflected here

Creating a BUG application

Creating a New Project

1. Install the Dragonfly SDK ^[3]
2. Decide what services ^[1] the application needs
3. If necessary, switch Perspective ^[4] to the Dragonfly perspective.
4. Select the **New BUG Project** icon in the tool bar, or File->New->Project->Dragonfly->BUG Application
5. In the New BUG Application window, fill out the Name and Author fields
6. Click **Next**.
7. Click **Start Virtual BUG**
8. Once the Virtual Bug has started, right click on the slots and add the module(s) your application will use
9. Under Target BUG select Virtual BUG
10. Check off the services your application will require
11. Click **Finish**
12. Click Run As and choose Virtual BUG

Almost all applications will consume OSGi services. This is the most direct way to gain access to the various modules, so we begin with a simple example of gaining access to the LCD. In this case select the IModuleDisplay service during the create New Project wizard. The Activator, ServiceTracker, and manifest files will all be created. The Activator and manifest need no changes. Within the ServiceTracker the method that needs to be modified is doStart().

Consuming a Service

To consume any OSGi service, you will need a ServiceTracker class. This is created by the

```
import org.osgi.framework.BundleContext;
import com.buglabs.application.AbstractServiceTracker;

import simplegui.app.*;

import com.buglabs.bug.module.lcd.pub.*;
/**
 *      Service tracker for the BugApp Bundle;
 *
 */
public class MyAppServiceTracker extends AbstractServiceTracker {

    public MyAppServiceTracker(BundleContext context) {
        super(context);
    }

    /**
     * Determines if the application can start.
     */
}
```

```

    public boolean canStart() {
        return super.canStart();
    }

    /**
     * If canStart returns true
     * this method is called to start the application.
     * Place your fun logic here.
     */
    public void doStart() {
        System.out.println("MyAppServiceTracker: start");
        IModuleDisplay display = (IModuleDisplay) getService(IModuleDisplay.class);
        MyApp app = new MyApp(display);
    }

    /**
     * Called when a service that this application depends is unregistered.
     */
    public void doStop() {
        System.out.println("MyApp: stop");
    }

    /**
     * Allows the user to set the service dependencies by
     * adding them to services list returned by getServices().
     * i.e. nl getServices().add(MyService.class.getName());
     */
    public void initServices() {
        getServices().add("com.buglabs.bug.module.lcd.pub.IModuleDisplay");
    }
}

```

The important piece of code is:

```

IModuleDisplay display = (IModuleDisplay) getService(IModuleDisplay.class);
MyApp app = new MyApp(display);

```

This passes access to the LCD to the MyApp class, which is shown below.

```

import com.buglabs.bug.module.lcd.pub.*;
import java.awt.Frame;

public class MyApp {

    /**
     * The BUG's LCD module interface
     */
    private IModuleDisplay display;

```

```
/**
 * This takes the display from the service tracker and passes it in
 * @param display
 */
public MyApp(IModuleDisplay display) {
    super();
    this.display = display;
    Frame frame = display.getFrame();
    frame.show();
}
}
```

In the above class, the application now has access to the service `IModuleDisplay` and can use all of its methods. This is the way in which all the APIs work, simply grab the service using the `getService(class)` method, and then use its methods. You do not have to worry about if the service is unavailable - the framework handles all the life cycle details. You do not need to worry about which slot the module is connected to or even what is providing the service.

Two basic examples of consuming services:

- An example application using the Motion Sensor ^[5]
- An example application using the LCD and the AWT toolkit ^[6]

Providing a Service

Some applications will want to provide an OSGi service to multiple applications. This case is a bit more specialized, but no more difficult. Simply use the `context.registerService(class, class, params)` method.

```
package publishexample;

import java.util.Random;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import org.osgi.framework.ServiceRegistration;

public class Activator implements BundleActivator, ExampleServiceInterface {

    private ServiceRegistration serviceRegistration;

    public void start(BundleContext context) throws Exception {
        serviceRegistration = context.registerService(ExampleServiceInterface.class.getName(), this, null);

        ServiceConsumer consumer = new ServiceConsumer(context);
    }

    public void stop(BundleContext context) throws Exception {
        serviceRegistration.unregister();
    }
}
```

```

    }

    public int getSpecialValue() {
        Random r = new Random();
        return r.nextInt();
    }

    private class ServiceConsumer {

        public ServiceConsumer(BundleContext context) {
            ServiceReference sr = context.getServiceReference(ExampleServiceInterface.class.getName());
            ExampleServiceInterface esi = (ExampleServiceInterface) context.getService(sr);

            System.out.println("Example service value: " + esi.getSpecialValue());
        }

    }

}

```

```

package publishexample;

public interface ExampleServiceInterface {
    public int getSpecialValue();
}

```

Creating a New Web Service

There are multiple ways to create a web service, but the simplest way is to use the `PublicWSProvider` interface. This is specific to the BUG, but has the advantage of hiding many of the details for those new to web programming. Additionally it registers the application in the BUG's list of services, just like the modules' services. Below is a simple "Hello World" example that can be accessed at <http://localhost:8082/service/examples>.

```

/**
 *      Generated by Dragonfly SDK
 *
 */
package examplews.servicetracker;

import org.osgi.framework.BundleContext;
import com.buglabs.application.AbstractServiceTracker;

import com.buglabs.services.ws.PublicWSAdmin;

import examplews.app.ExampleWSApp;
/**
 *      Service tracker for the BugApp Bundle;
 *
 */
public class ExampleWSServiceTracker extends AbstractServiceTracker {

```

```
private PublicWSAdmin wsAdmin;

private ExampleWSApp app;

public ExampleWSServiceTracker(BundleContext context) {
    super(context);
}

/**
 * Determines if the application can start.
 */
public boolean canStart() {
    return super.canStart();
}

/**
 * If canStart returns true
 * this method is called to start the application.
 * Place your fun logic here.
 */
public void doStart() {
    System.out.println("ExampleWSServiceTracker: start");
    app = new ExampleWSApp();
    wsAdmin = (PublicWSAdmin) getService(PublicWSAdmin.class);
    wsAdmin.registerService(app);
}

/**
 * Called when a service that this application depends is unregistered.
 */
public void doStop() {
    System.out.println("ExampleWSServiceTracker: stop");
    wsAdmin.unregisterService(app);
}

/**
 * Allows the user to set the service dependencies by
 * adding them to services list returned by getServices().
 * i.e. nl getServices().add(MyService.class.getName());
 */
public void initServices() {
    getServices().add("com.buglabs.services.ws.PublicWSAdmin");
}

}
```

The registered `PublicWSProvider` will be passed `com.buglabs.services.ws.IWSResponse` objects when a HTTP call is made to the URL of the service. This URL is determined by the string returned by `getPublicName()`. The location of the service in this example will be `http://localhost:8082/service/exampleWS`.

The `execute(int operation, String input)` method runs when a HTTP call is recieved. The operation is either a GET, PUT, POST, or DELETE. The input string the body of the message recieved. For GET calls this will just be the URL of the service. For a POST call, this value be the form data, in the form of `key1=value1&key2=value2` etc.

```
package examplews.app;

import java.util.List;

import com.buglabs.services.ws.IWSResponse;
import com.buglabs.services.ws.PublicWSDefinition;
import com.buglabs.services.ws.PublicWSProvider;

public class ExampleWSApp implements PublicWSProvider {

    public String getDescription() {
        return "Example Web Service";
    }

    public String getPublicName() {
        return "exampleWS";
    }

    public PublicWSDefinition discover(int operation) {
        if (operation == PublicWSProvider.GET) {
            return new PublicWSDefinition() {
                public String getReturnType() {
                    return "text/html";
                }

                public List getParameters() {
                    return null;
                }
            };
        }
        return null;
    }

    public IWSResponse execute(int operation, String input) {
        if (operation == PublicWSProvider.GET) {
            return new IWSResponse() {

                private String message = "";

                public Object getContent() {
                    System.out.println("get");
                }
            };
        }
    }
}
```

```
        StringBuffer strbuf = new StringBuffer();
        strbuf.append("<h2>Hello World!</h2>");
        return strbuf.toString();
    }

    public int getErrorCode() {
        return 0;
    }

    public String getErrorMessage() {
        return null;
    }

    public String getMimeType() {
        return "text/html";
    }

    public boolean isError() {
        return false;
    }

    };
} else {
    return new IWSResponse() {
        private String message = "";

        public Object getContent() {
            return new Object();
        }

        public int getErrorCode() {
            return 405;
        }

        public String getErrorMessage() {
            return "Only GET requests are available on this resource";
        }

        public String getMimeType() {
            return "text/html";
        }

        public boolean isError() {
            return true;
        }

    };
}
```



```
}  
  
}
```

Where to go from here

This is only a brief introduction, and not meant to be a definitive guide. This Wiki contains a great deal of reference documentation, from Javadocs and schematics to step by step tutorials. Additionally, all of the applications on BUGnet include their source. Several of these applications are examples. Also, check the Developer Resources page for links to the forums, mailing list, and IRC channel.

- [Software:Develop_with_Java](#)
- [Javadocs](#) ^[7]
- [Application How-TOs](#) ^[8]
- [Developer Resources](#) ^[9]

References

- [1] http://bugcommunity.com/wiki/index.php/Module_Services
- [2] <http://bugcommunity.com/wiki>
- [3] http://bugcommunity.com/wiki/index.php/SDK_Install_Guide
- [4] http://bugcommunity.com/wiki/index.php/Switch_Perspectives
- [5] http://bugcommunity.com/wiki/index.php/Create_a_basic_application_with_the_Motion_Sensor_module
- [6] http://bugcommunity.com/wiki/index.php/Create_a_basic_GUI
- [7] <http://bugcommunity.com/wiki/index.php/Javadocs>
- [8] http://bugcommunity.com/wiki/index.php/Application_HOWTOs
- [9] http://bugcommunity.com/wiki/index.php/Developer_Resources

SDK Install Guide

Overview

The Dragonfly software development kit (SDK) is an Eclipse-based development environment that enables you to develop BUG applications in Java to run on BUG.

BUG Development Tools

The BUG SDK provides a comprehensive environment for coding, running and testing BUG apps. When creating new apps, the SDK helps you select the hardware modules and services to use and it generates the base code you will start from. This provides a framework for interacting with the OSGi services that represent the pluggable BUG hardware modules.

BUG Simulator

The built-in BUG Simulator is an emulator that allows you to quickly test-drive your applications in a controlled environment without BUG hardware. You can deploy your apps to BUG Simulator and debug them within Eclipse.

BUGnet Community

The SDK also provides access to the BUGnet community, where you can browse, download and share BUG apps. BUGnet provides a rich repository of BUG apps and code that you can try out and re-use in your BUG apps. When you have a BUG app to share, you can upload it to BUGnet and share it with the BUG community.

Dragonfly Perspectives

In Eclipse, a perspective is an organized view of the tabs and windows for a specific development task. For BUG programming, there is a Dragonfly perspective that presents the views that often need in BUG development.

Opening the Dragonfly Perspective

- In Eclipse, go to **Window > Open Perspective > Other...**
 - In the Open Perspective window, click on **Dragonfly** to select it.
- The recently-used perspectives in Eclipse are listed in the upper-right corner. If you switched to the Dragonfly perspective recently, you should see Dragonfly and its icon listed there. Click on it to quickly switch to Dragonfly.

Installation

To use the Dragonfly SDK, you need to download and install Eclipse, then give Eclipse a simple setting to download and turn on the Dragonfly SDK. Bug Labs has (and will again) provide one click installers for Eclipse & Dragonfly SDK, but for the moment we are using a two step install system.

Prerequisites

If you have these prerequisites, skip ahead to 'Installing Dragonfly SDK'

- Dragonfly is compatible with Eclipse version 3.6 (Helios) or greater.
- Java 1.6 -- this can be tricky in Linux. See the troubleshooting section if you are unsure.
 - On Linux, you may need to install xulrunner for the browser view to work inside Eclipse.
 - If you are running on a 64-bit CPU, make sure your JDK and Eclipse versions are also 64-bit

Installing Eclipse

Hopefully, you have Eclipse already installed. If not, visit www.eclipse.org and look for the latest **Eclipse Classic** or **Eclipse for Java Developers** downloads. The latest version is currently 3.6 Helios. Download and install it, using these instructions if needed ^[1].

Install the Dragonfly SDK Plugin in Eclipse

These instructions assume you already have Eclipse version 3.6.

- In Eclipse, go to **Help > Install New Software**.
- On the right side, select the **Add...** button to open the **Add Site** dialog.
- Enter the Dragonfly Update Site url into the Location box (in this case, it's the production release. For more information, see <http://buglabs.net/sdk> ^[2]):
 - <http://www.buglabs.net/sdk/production/current/updatesite/>

- Click **OK**.
- An item will appear in the **Available Software** listing the "Dragonfly SDK".
- Select the checkbox to the left of Dragonfly SDK.
- Click the **Next >** button on the bottom of the dialog.
- A progress bar should appear.
- After a few seconds, the **Install Details** dialog will appear. One item, Dragonfly SDK, should be listed.
- Click **Next**
- The licenses associated with the SDK are displayed for review.
- Once you have reviewed them to your satisfaction select the **I accept the terms of the license agreements** radio button.
- Click **Finish**.
- Eclipse will detect and install a dependency called the **Graphical Editing Framework (GEF)** if you don't already have it.
 - If you need to install GEF manually, you can search for "gef" in the Helios update site: <http://download.eclipse.org/releases/helios>
- A progress bar will appear as Eclipse downloads and installs the plugins.
- You will then receive a security warning **Warning: You are installing software that contains unsigned content...**
- Pretend you're running Vista and just click OK. We thank you for your donation from your bank accounts
- OK, that last bit isn't true, you can click the **<<Details** button and verify that everything is a com.buglabs... jar file, then click OK.
- When installation is complete, Eclipse will ask if you would like to restart.

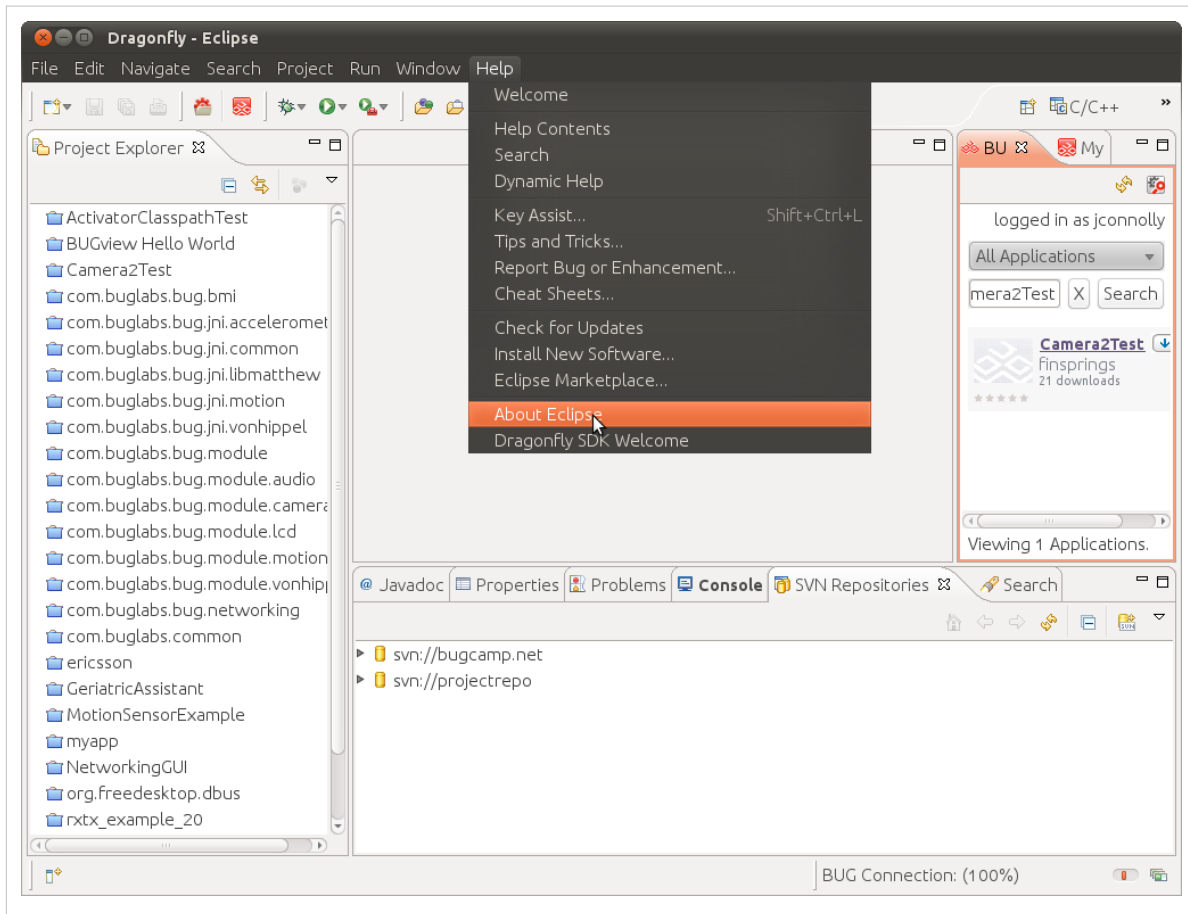
Troubleshooting

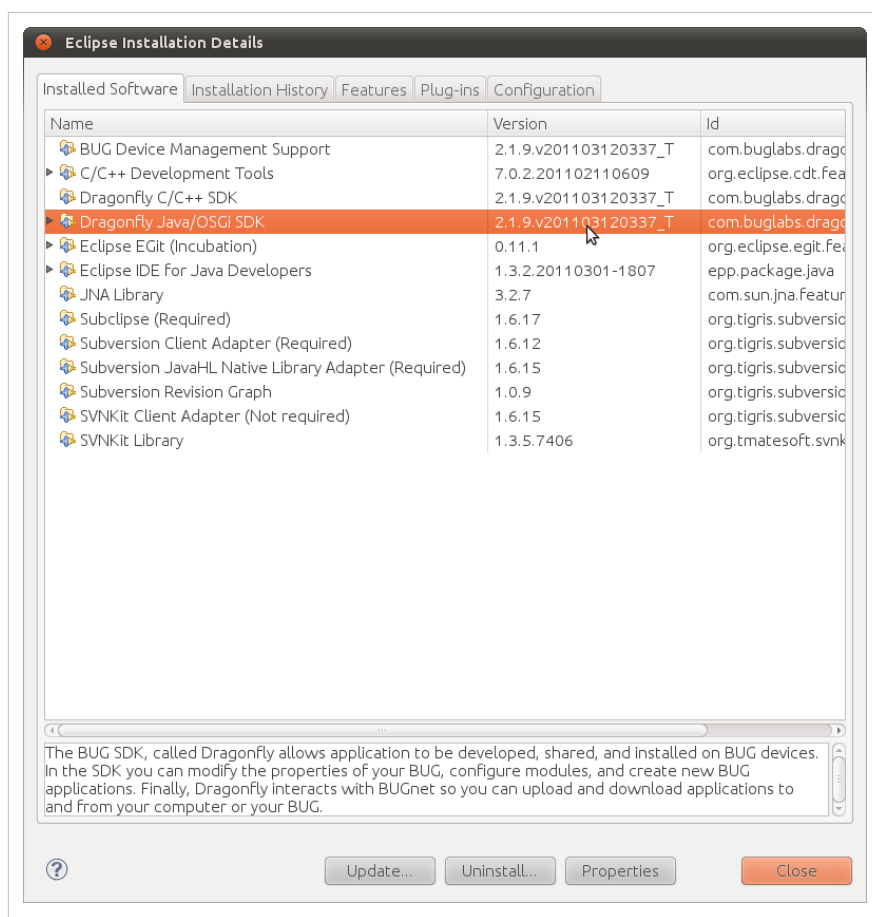
If you are using Linux and the Dragonfly perspective is not available after a seemingly successful installation, it is possible that Eclipse is not using the correct version of Java. Determining the version of Java Eclipse is using can be frustrating.

One user fixed this problem by editing the eclipse.ini file in the root directory of his Eclipse installation to force the use of Java 1.6. The information at <http://wiki.eclipse.org/Eclipse.ini> explains how to do this.

SDK Version

When popping into IRC or filing a defect on redmine.buglabs.net ^[3], please attach information about the version of Dragonfly (and Eclipse that you're using) Installation Information dialog:





Next Step

Check out the Getting Started Tutorial to write your first program using the Dragonfly SDK

See Also

- Contact Us

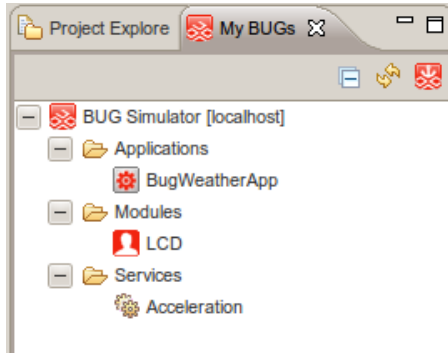
References

- [1] http://wiki.eclipse.org/FAQ_Where_do_I_get_and_install_Eclipse%3F
- [2] <http://buglabs.net/sdk>
- [3] <http://redmine.buglabs.net>

Interacting with BUGs in Dragonfly SDK




My BUGs View


The Eclipse view called **My BUGs** is usually found in the Dragonfly perspective. If it is not visible, you can display it by going to: **Window > Show View > Other > Dragonfly > BUGnet**.



Connection Types

The My BUGs view displays any BUG Connections that are available to the SDK and allows you to interact with them. There are 3 connection types you will see in this window:


-  BUG Simulator Connection.
-  Static Connection (Manual).
-  Discovered BUG Connection (Physical BUG).


Sometimes, My BUGs view has not refreshed to show new connections. To refresh My BUGs, click on the refresh icon  near the top of My BUGs view.

Actions in My BUGs View

- Browse any BUG Connection's Applications, Modules, or Services nodes.
- Double click the node on a BUG Connection to bring up a Physical Editor window for the Connection.
- Right click a BUG Connection to bring up a menu (see the Right Click Menu section below for details and options).
- Right click on an application in the BUG Connection Applications folder (see the Right Click Menu section below for details and options).

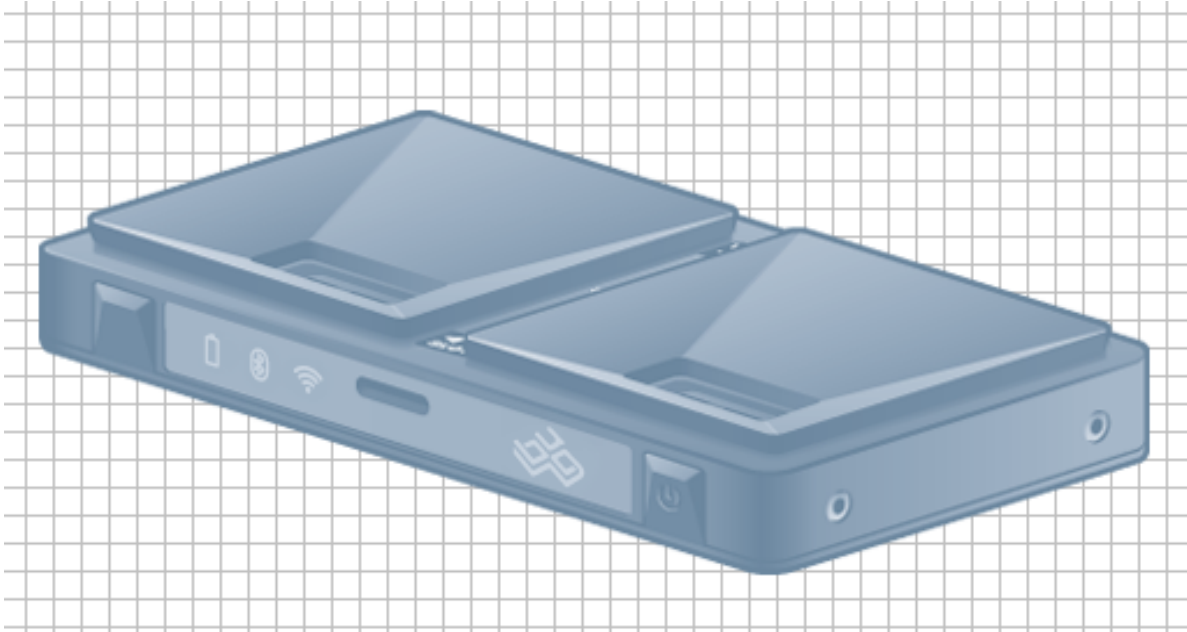
Launching BUG Simulator

If BUG Simulator is currently running, you will see it listed in My BUGs view. The BUG Simulator button  also shows whether the BUG Simulator is running. If the button icon is red, this means that the simulator is probably not running. If it is blank, the simulator is already running.

Hint: To stop a running instance of BUG Simulator, go to the Console view in Eclipse and click on the red **Terminate** button .

Interacting with BUG Simulator

As noted above, you can open the BUG Simulator window from My BUGs view. Double click on the **BUG Simulator** element at the top of the My BUGs view to open the BUG Simulator window. You should see something like this.




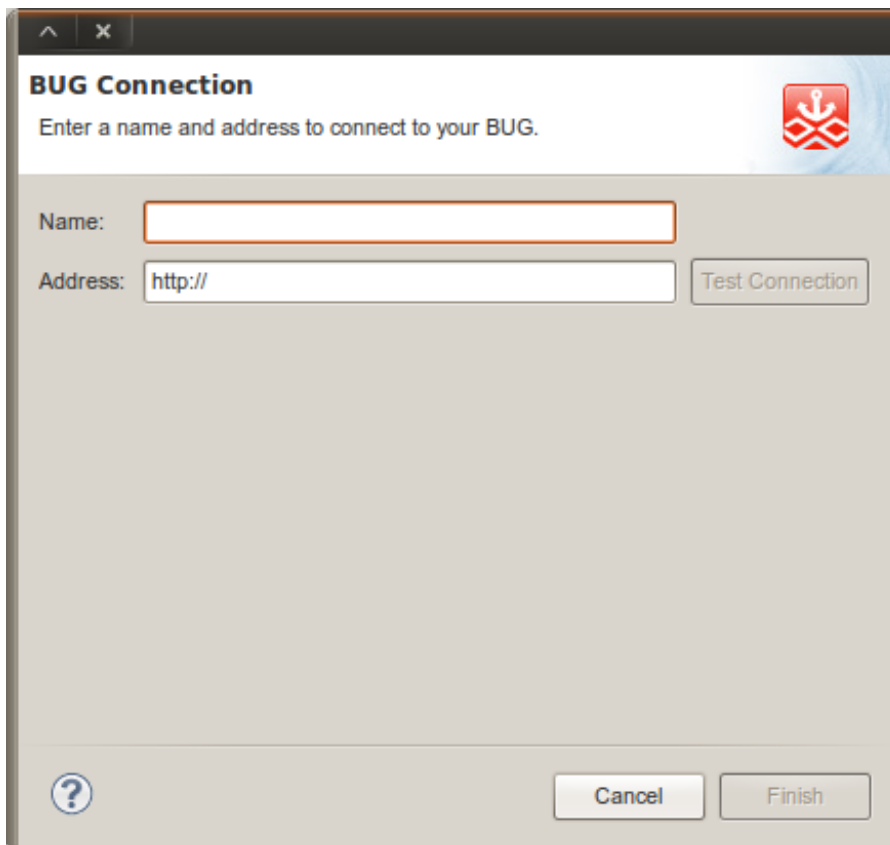
In the simulator, you will see an image of a BUGbase. There are slots at the top and bottom that are designed to hold external modules that are plugged in. Right now, the simulator does not have anything plugged in. And that's a lot like how your physical BUG hardware starts.

BUG Connection

When BUG Simulator is launched, it will automatically appear in the My BUGs view. The same should happen for a physical BUG when it is connected to your Network. In some cases, it is necessary to create the connection manually. (If you are having trouble please see <http://buglabs.net/start>).

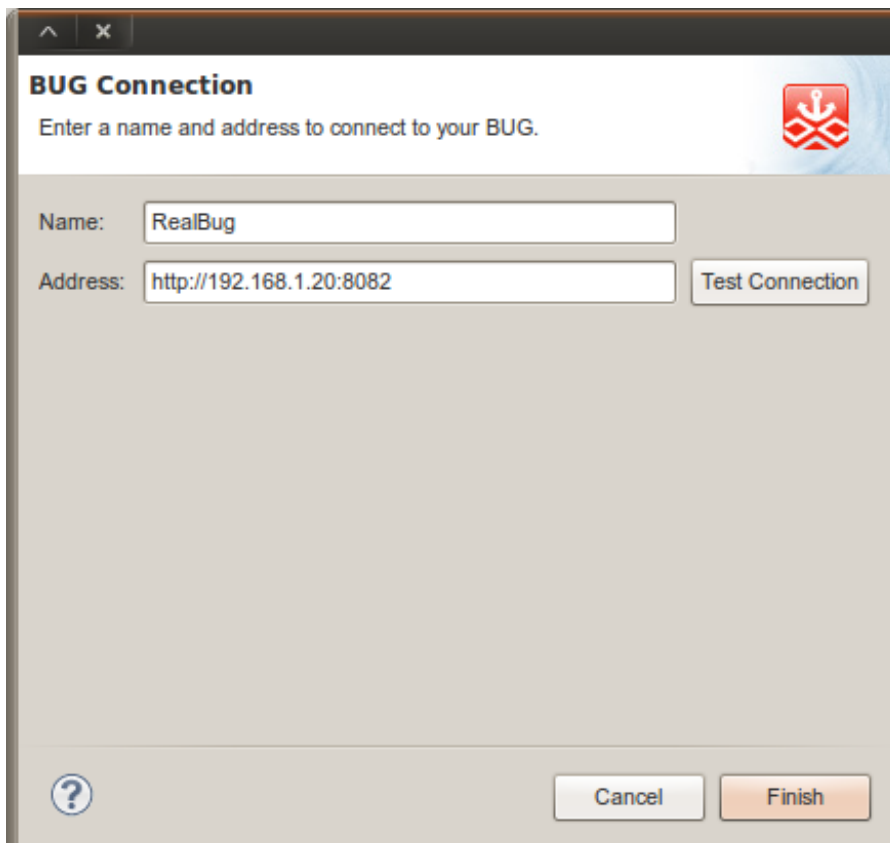
Creating a New BUG Connection

1. Click the New BUG Connection button  to manually create a BUG Connection. This button is available via the My BUGs view title bar, and by right clicking in the My BUGs view.



The image shows a 'BUG Connection' dialog box. At the top, it says 'Enter a name and address to connect to your BUG.' There is a red icon with a white anchor and a cross. Below the text, there are two input fields: 'Name:' and 'Address:'. The 'Address:' field contains 'http://'. To the right of the 'Address:' field is a 'Test Connection' button. At the bottom, there is a question mark icon, a 'Cancel' button, and a 'Finish' button.






2. After you enter a Name and Address, the Test Connection and Finish buttons will become active.



The image shows the same 'BUG Connection' dialog box, but now the 'Name:' field contains 'RealBug' and the 'Address:' field contains 'http://192.168.1.20:8082'. The 'Test Connection' button is now active (highlighted in orange). The 'Cancel' and 'Finish' buttons are still present at the bottom.

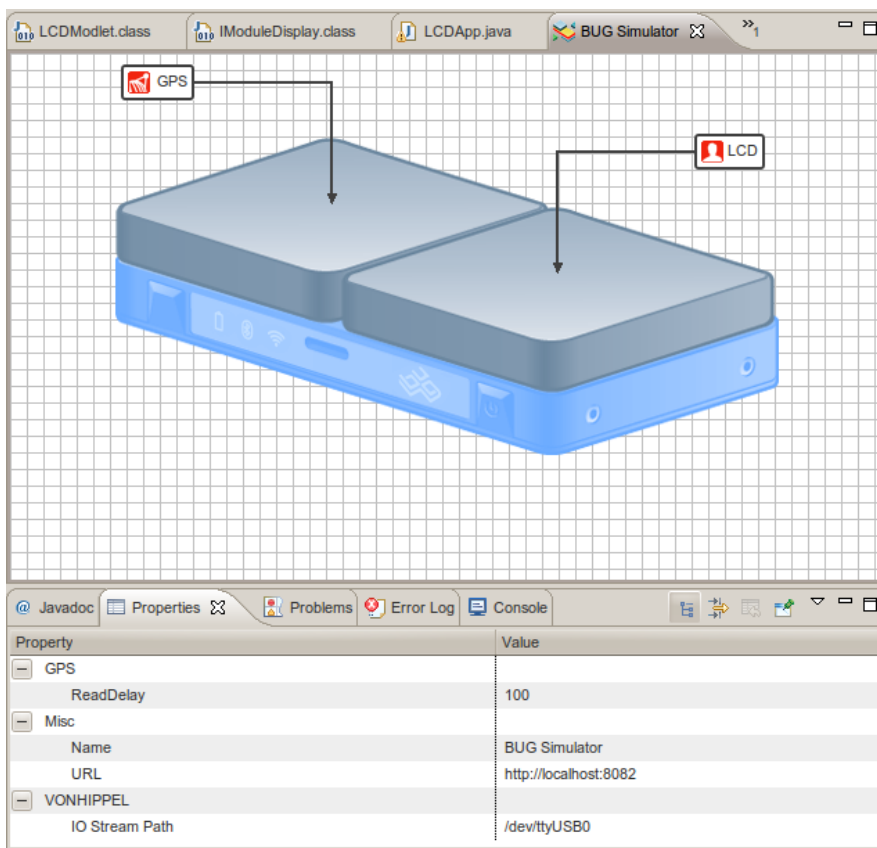
3. Click Test Connection to validate the Connection or click Finish to add it to the My BUGs view.

Right Click on BUG Connection Menu

-  New BUG Connection button - Opens New BUG Connection Wizard
-  Delete BUG Connection button - Only active for Manually created Connections
-  Refresh BUG Connection button
- Properties - Update BUG Connection Address
- Additional menu options are available when right clicking on an application in the Applications folder.
 -  Remove application button (Note: If you right click on the Applications folder this button is "Remove All applications")
 -  Import into Dragonfly SDK button

Physical Editor Properties View

When you click on the BUGbase or any Module name tag in the Physical Editor, it will be highlighted blue and the Properties view will populate with specific information for the component.



Using Modules in BUG Simulator

Introduction

The modules that you add in BUG Simulator are important, since they define the services that are available when creating applications AND when running them. Basically, that means:

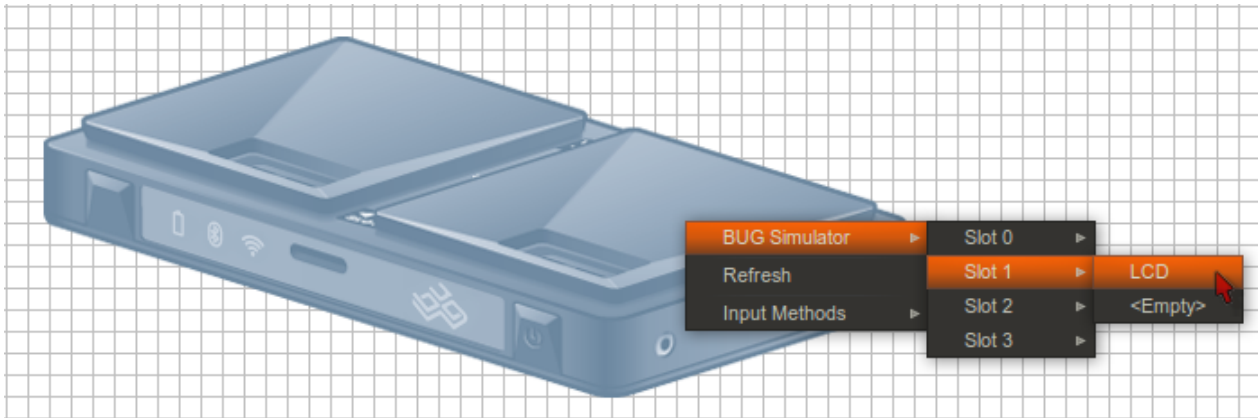
- When you create a new BUG project, it is a good idea to add the modules in BUG Simulator first.
- When running and testing an app in BUG Simulator, your app will wait until you add the modules it expects to use.

Starting the BUG Simulator

The following tutorial covers starting the BUG simulator: Tutorial:Interacting_with_My_BUGs_in_Dragonfly_SDK

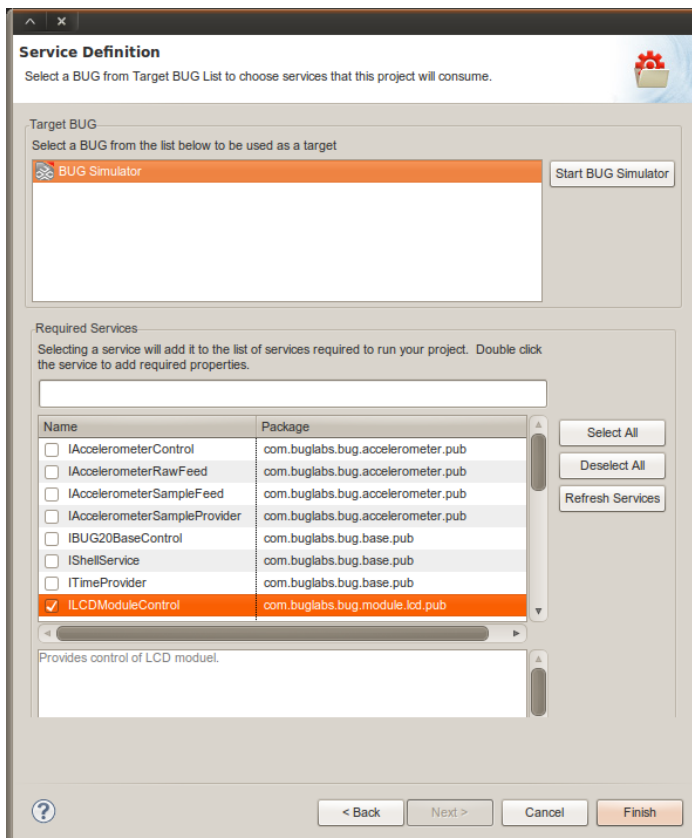
Adding a Module

The next step is to plug in some hardware. In this example, we will use the LCD module. To plug in the virtual LCD, right click on the BUGbase image and navigate to the menu item: **BUG Simulator > Slot 1 > LCD**



Using the BUG Simulator When Creating a New BUG Project

When you are creating a new BUG project to run on the BUG Simulator, only the currently installed modules are listed in the Service Definition window. If you added the LCD module in BUG Simulator, then you will see the **ILCDModuleControl** in the list of available services like in the example below. By adding the modules services in the Service Definition window, you get the benefit of the SDK writing some of the bootstrap classes and code for calling each service.



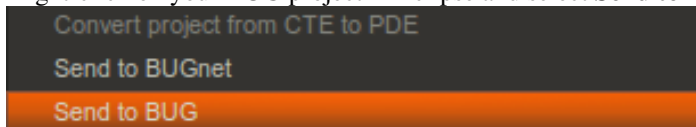
Adding Services Manually in Code

If you did not add the LCD module or other module(s) that you need or if you used the **Start BUG Simulator** button in this window, then these modules will not appear in this list. However, this does not prevent you from manually adding the services you require.

Using the BUG Simulator When Testing an App

When you are building and testing apps in BUG Simulator, you often have to start or restart the BUG Simulator and use the **Send to BUG** command to deploy the app. Your code that depends on hardware modules will only execute when the modules are present. So you will frequently need to perform the following steps to run and test your app:

1. If BUG Simulator is already running, you may need to terminate the running instance first. To do this, look for the **Console** view in Eclipse and click on the red **Terminate** button . (Note: You can often redeploy an app into a running BUG Simulator without stopping and starting the simulator)
2. Click on the **Launch BUG Simulator** button
3. Right-click on your BUG project in Eclipse and select **Send to BUG**



4. The BUG Connections window will open to let you choose the BUG to send to. Select the BUG Simulator and click OK.
5. Open **My BUGs** view and open the **BUG Simulator** window by double-clicking on the BUG Simulator listed in My BUGs.
6. Right-click on the BUG base and add the required modules (as shown above)

Interacting with BUGnet

Introduction

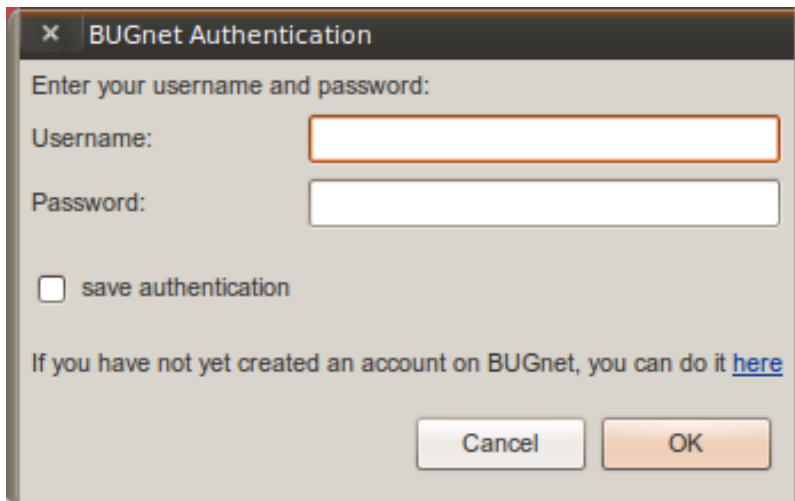
BUGnet is our online community, including the repository for sharing applications. In the Dragonfly SDK, you can access BUGnet and the apps repository through the **BUGnet view**, which is an Eclipse view that allows you to interact with the BUG community. You can search, upload, or download applications through BUGnet view.

The BUGnet view in Dragonfly SDK interacts with the Bug Labs API located at: <https://api.buglabs.net>. Through the API, BUGnet helps you find the apps and code samples that you are looking for.

- **Hint:** If you do not see the BUGnet view, go to **Window > Show View > Other** and look for **Dragonfly > BUGnet**.

Login to BUGnet

If you are not logged in yet to BUGnet, you will see a link to "login to BUGnet". Click the link to open the BUGnet Authentication window.

A screenshot of the 'BUGnet Authentication' dialog box. The dialog has a title bar with a close button and the text 'BUGnet Authentication'. Inside, it says 'Enter your username and password:'. There are two text input fields: 'Username:' and 'Password:'. Below the password field is a checkbox labeled 'save authentication'. At the bottom, there is a line of text: 'If you have not yet created an account on BUGnet, you can do it [here](#)'. At the very bottom are two buttons: 'Cancel' and 'OK'.

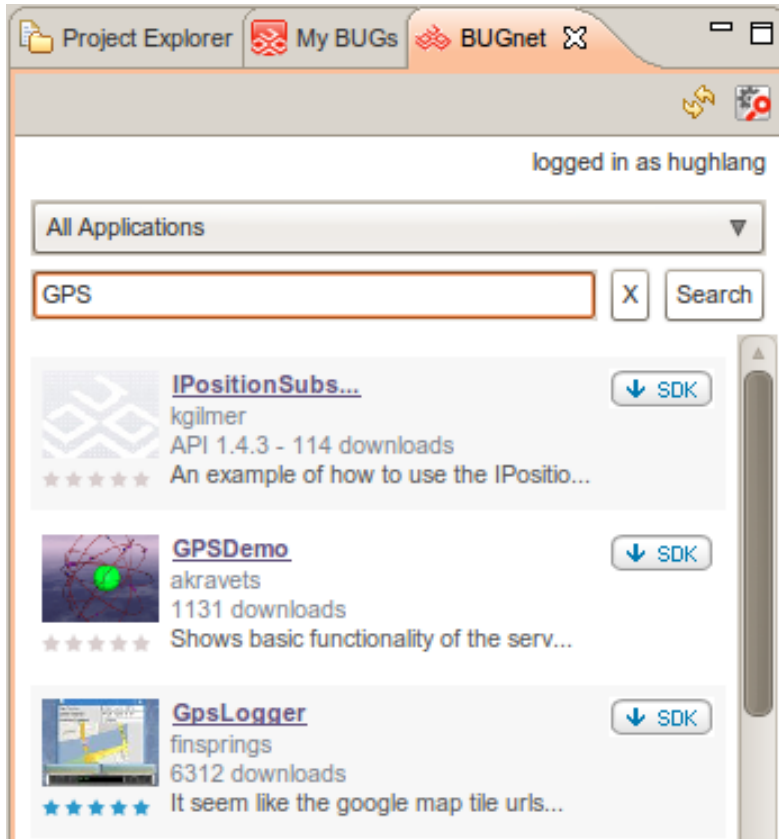
After a successful login, the application list will refresh, showing both public applications and the private applications that you have permission to view.

Signing Up for BUGnet

If you don't have BUGnet account you can click on the link in the login window. Or go the signup page: <http://www.buglabs.net/signup>

Searching BUGnet

To search for apps on BUGnet, enter your search text in the form and press 'Enter' or click on the Search button.



Search Terms

Here are some ways you can search BUGnet:

- Search by topic (like "GPS", "LCD") to find apps that use specific BUG modules or technologies
- Search by app name, if you already know the application you are looking for.
- Search by username, if you know someone in the BUGnet community. Or to find your apps, you can search by your own username.

Search Results

1. The number of apps in the current search results is displayed in the bottom-left. Example: **Viewing 30 Applications**
2. If there are more apps to display in the current search results, there will be a **View more...** link in the lower right to fetch the next page of results.

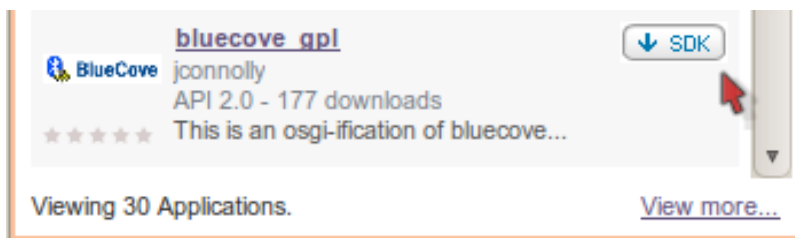
Search Filter

The Search Filter has the following choices:

- All Applications: Shows all applications on BUGnet that match the search terms. If the search box is empty, it will return all applications.
- My Applications: Shows all applications for which you are an administrator or collaborator. If you select this filter prior to logging in, you will be presented with the BUGnet Authentication window and asked to log in.
- Applications for My BUGs: If you have a real or virtual BUG connected to the SDK, they will appear as a list here. If you have no BUGs connected, then you will not see this filter. This filter allows you to search only for applications that will run on the connected BUG chosen, based on the modules attached to the connected BUG.

Discovering Apps

As you can see from the image below, each app listing displays the app name, the author(s), the number of downloads, and the start of the description. As you move your mouse over the description field, an info layer will appear to show you the entire description. This is a nice way to quickly browse and learn about various apps.



If you click on the application name (which looks like a hyperlink), the application web page will display in a new tab. From this page, you can learn more about the app, browse the source code, review the history and comments, etc.

Downloading Apps

In the app web page, you can also click the **Download Latest Version to SDK** button to download the application and add it to your Eclipse workspace.


- **Note:** The app is downloaded in the background. Look in your Eclipse workspace directory for the downloaded project directory.
- You can also click on the SDK download button next to the listing in the search results.

BUGnet Preferences

A number of settings for the BUGnet view can be accessed via the Eclipse Preferences. Go to: **Window > Preferences > BUGnet**

- Enable BUGnet - checkbox that allows you to go into offline mode.
- Server Name - The web service URL to buglabs.net.
- Number of Applications to display - Accepts 0 to 100. This changes the number of applications displayed in the My Apps and Top Apps sections.
- Clear Authentication Data - Clears saved BUGnet Authentication and logs you out of your current session.

Finding Applications on the Bug Labs Website

Click the 'Search Applications' button  to load the Community Applications page in a new tab.

- You can also access this page in a browser: <http://www.buglabs.net/applications>

The Community Applications page has additional search options, such as the ability to browse by category and/or filter by development status.

Getting Started Tutorial

BugWeather Introduction




This tutorial will take you through some of the basic development tasks in Dragonfly SDK by building your first application, BugWeather. BugWeather will fetch the local weather and display it on the LCD. First we will create a basic application that can display information on the LCD, then we will test it in the BUG Simulator, and finally fetch the weather from the internet. This assumes you have already followed the Dragonfly SDK Install Guide ^[1]

Topics Covered

- Creating an app with Dragonfly SDK
- Understanding the OSGi Activator and ServiceTracker
- Displaying a GUI using Swing

The first step is to create a Dragonfly project in Eclipse that has the basic code for creating and running a Dragonfly app.

Start a new project

1. Make sure you are in the Dragonfly perspective in Eclipse. (Windows > Open Perspective > Other > Dragonfly)
2. Click  in the toolbar to launch the simulator.
3. Double-click the simulator  **Bug Simulator [localhost]** in the 'My BUGs' panel
4. Add an LCD module to the simulator (Right click > BUG Simulator > Slot 1 > LCD)
5. Add an GPS module to the simulator (Right click > BUG Simulator > Slot 0 > GPS)
6. Click  in the toolbar to create a new project.
7. Name the project **BugWeather** and click "Next"

New BUG Project




Enter a project name and optionally provide other information.

Name:

Author:

Description:



Choose services

The service definition window will display a list of OSGi services that are available for each module. Note that we don't need any special service to draw to the BUGview module, since swing takes care of that automatically.


1. Select the **locate** module and check the **Position Service**, which will allow reading gps coordinates.
2. Hit *Next*. This page is for selecting services that aren't related to a physical module. We will ignore it for now.
3. Hit *Next*. This page lets you customize the code generation. The defaults are for your safety, so let's leave them be.
4. Hit *Finish*. Time to start writing code!

BUG Module Services



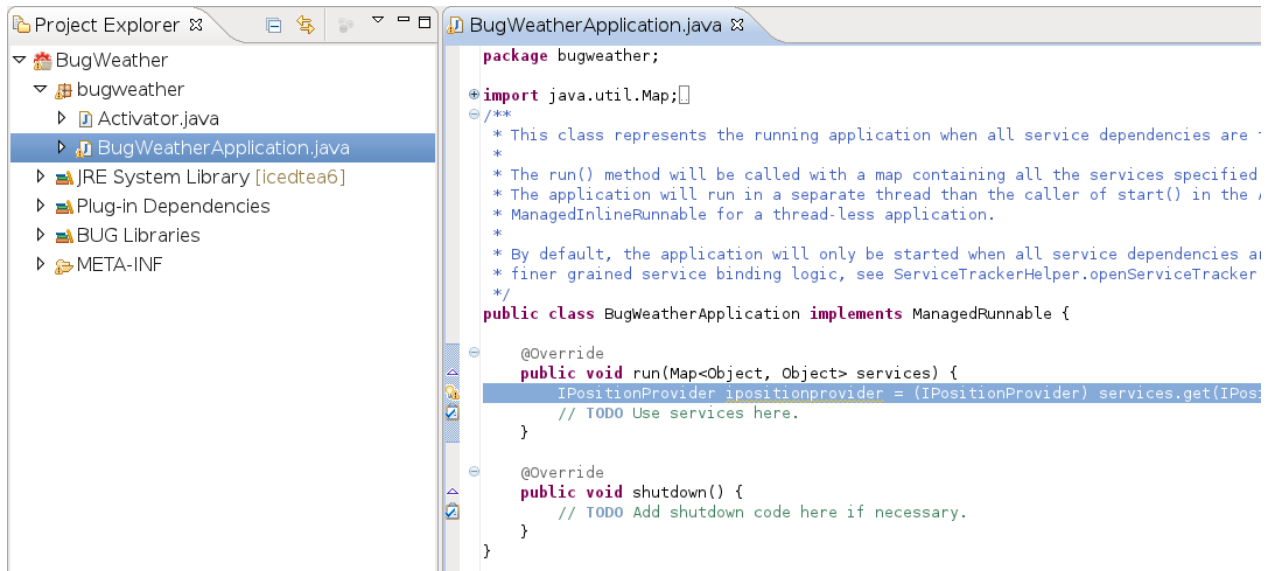
Select BUG module services that will be used in the application.

Name	OSGi Service
<input checked="" type="checkbox"/> Position Service	com.buglabs.bug.module.gps.pub.IF
<input type="checkbox"/> GPS Control	com.buglabs.bug.module.gps.pub.IF
<input type="checkbox"/> NMEA Sentence Provider	com.buglabs.bug.module.gps.pub.IF
<input type="checkbox"/> NMEA Data Provider	com.buglabs.bug.module.gps.pub.IF



Create BugWeatherApplication

The new project wizard will automatically create `Activator.java`. The `Activator` class implements `org.osgi.framework.BundleActivator`, which means it handles the starting and stopping of OSGi services (when hardware is added or removed). Through the `Activator`, the `ServiceTracker` instance will handle start and stop events for each device it manages. In the `run` method exposes `ipositionprovider` which will be used to get the position of the BUG and draw on the screen.



And inside of the `run` method, we need to instantiate our application like this:

```
// Warning, this method will be called from within the same thread as the OSGi framework. Long running operations should be avoided here.

// Implement application here.

System.out.println("BugWeatherActivator: start");

BugWeatherApplication application = new BugWeatherApplication();
```

Now lets make `BugWeatherApplication` do something interesting.

Write BugWeatherApplication

You can add more bells and whistles if you're really awesome with Swing/awt programming.

Add a `JFrame` to `BugWeatherApplication` so we can draw stuff to the screen:

```
private JFrame f;
```

Next let's set some text labels on that `jframe`, inside the `BugWeatherApplication` `run` method. Add the following code:

```
Label l = new Label();
l.setBounds(10, 60, 300, 30);
l.setText("Current Weather");
l.setVisible(true);
l.setAlignment(Label.CENTER);

f = new JFrame();
f.setTitle("BUG Weather App");
f.setLayout(new BorderLayout());
```

```
f.add(l, BorderLayout.NORTH); // add the label to the new frame
f.setVisible(true);
```

Finally, let's dispose of our window nicely:

```
public void shutdown() {
    f.dispose();
}
```

Fix any errors by importing the correct classes. Now we should be ready to test if our code works in the simulator. Fingers crossed!

Testing in the simulator

Now that we have written some code, let's learn how to bundle the application and test it in the simulator.

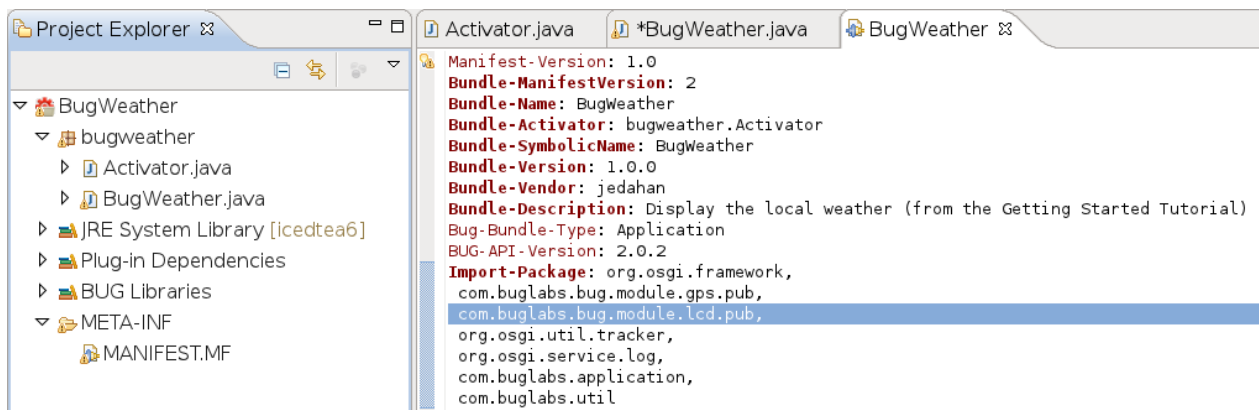
Topics Covered

- Editing the MANIFEST.MF file
- Running an app in the BUG Simulator
- Sending an app to BUG

Editing the Manifest File

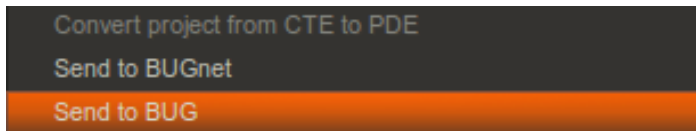
One of the essential things you need to do before running or deploying an app is to declare the packages to import. This is done in the **MANIFEST.MF** file which can be found in the **META-INF** folder in the Eclipse Project Explorer. (see below)

Above the highlighted line there is an entry for "com.buglabs.bug.module.gps.pub" which was automatically inserted when we chose to add the BUGlocate Position Provider service. Since we want to use the LCD, we also need to add an entry here for **com.buglabs.bug.module.lcd.pub**.



Sending to a BUG

Now we are ready to send the app to the simulator. If you right-click on the BugWeather project in the Project Explorer, a context menu will appear. Near the bottom is **Send to BUG**, which does exactly what it says. By executing this command, the compiled application code is copied over to the running BUG Simulator. If you are updating an app that you had previously run, the "Upload to BUG" dialog appears asking you to confirm the action. Click Yes to overwrite the existing app.

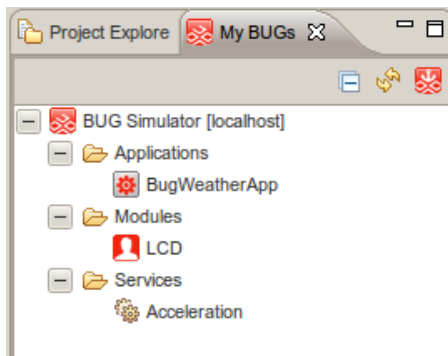


Oh no, some red lines in the console! Read them carefully, and think back to the `manifest.mf` section to fix the errors!

Your source code should look something like BugWeather Part1

My BUGs View

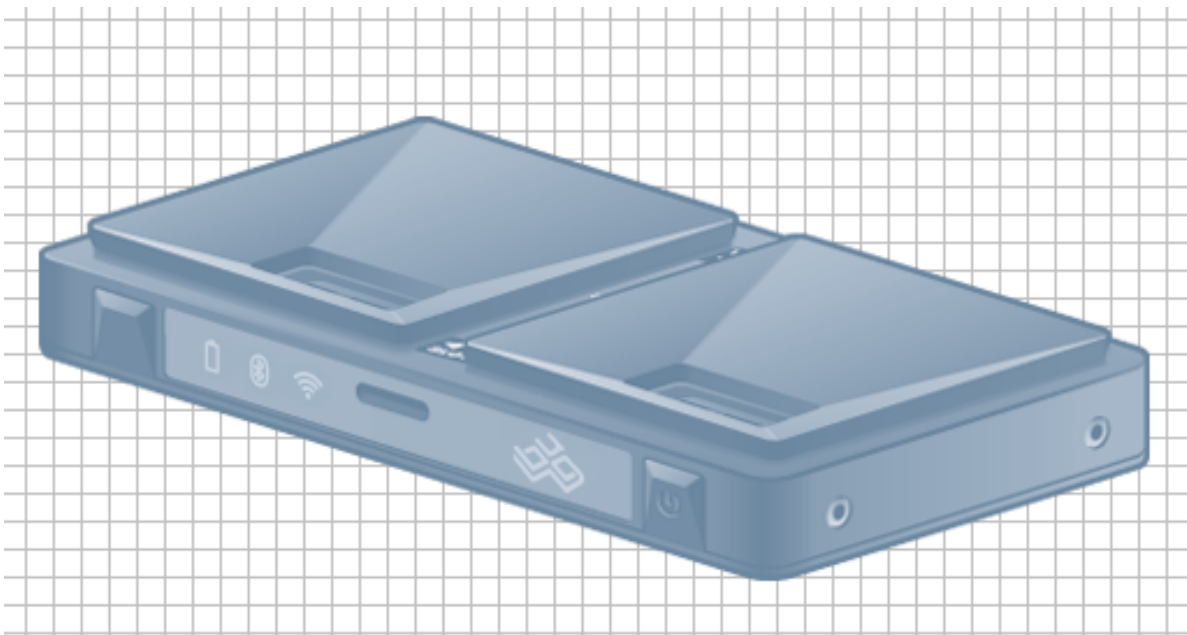
In the Dragonfly perspective in Eclipse, there usually is a tab called "My BUGs" somewhere. If you don't see it, go to the menu bar and select **Window > Show View > Other...** and look for the **Dragonfly SDK... My BUGs**. When it is opened and visible, you will see something like this:



Since you executed the "Send to BUG" command only a moment ago, it is possible that you will not see the "BugWeather" listed in the Applications folder. If so, click on the refresh icon 🔄 near the top of My BUGs view.

BUG Simulator View

While you can see the different components and apps that are installed on the BUG Simulator, what you really want to see is the BUG in action (or at least, simulated action). Double click on the **BUG Simulator** item at the top of the My BUGs view to show the BUG Simulator view:

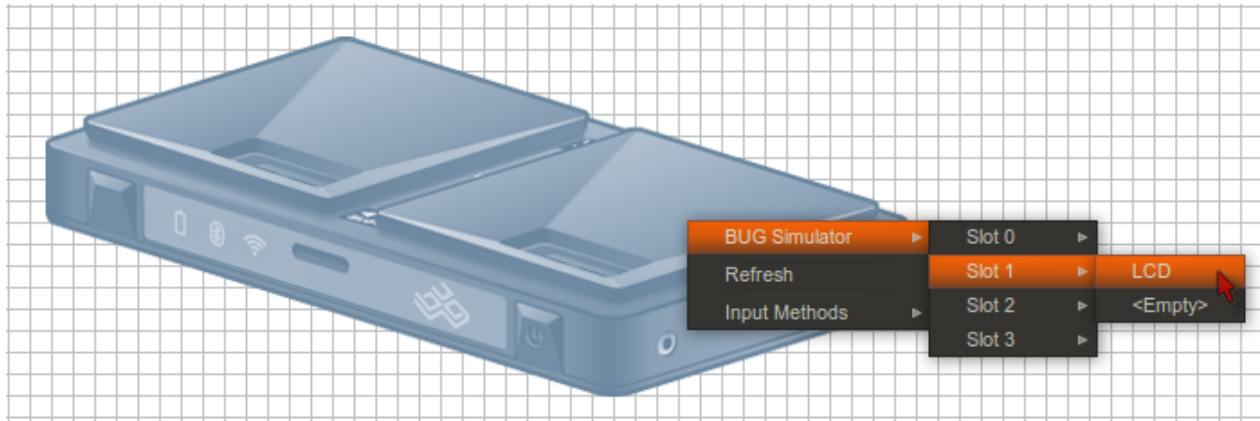


In the simulator, you will see an image of a BUGbase. There are slots at the top and bottom that are designed to hold external modules that are plugged in. Right now, the simulator does not have anything plugged in. And that's a lot

like how your physical BUG hardware starts. So, the next step is to plug in some hardware. Specifically the LCD module.

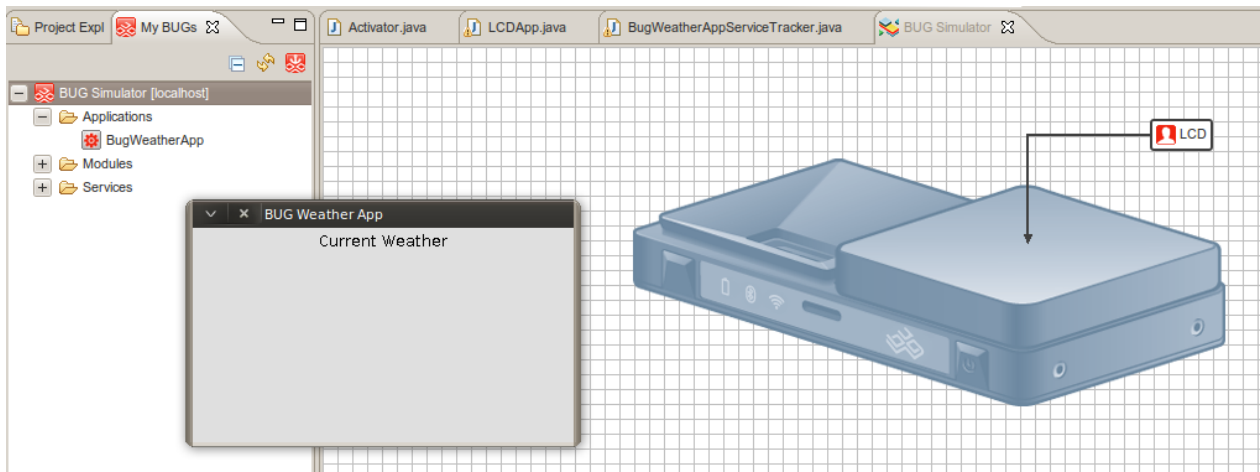
Adding the LCD Module

To plug in the virtual LCD, right click on the BUGbase image and navigate to the menu item: **BUG Simulator > Slot 1 > LCD**



LCD Display Window

Recall for a moment that the BUG Simulator is already running and the code we wrote relies on OSGi to detect and handle hardware that is added (or removed). As soon as you add the LCD module to Slot 1, the Simulator will execute the BugWeather code. In the BugWeatherApplication class, we told it to create a new frame and add a Label to display "Current Weather". You should see something like the image below. The AWT window will run as a separate Java program on your computer and render the content.



Getting the weather

We have gone through all the basic steps of creating an app and writing the basic code needed to interact with the BUGview module. We tested it in the simulator. Now the final step is to get some real weather data in our app and display it in the BUG Simulator. To do this, we are going to use Google's secret and unofficial Weather API ^[2]. Just because it's there. You can take a sneak peek at Google's weather API by hitting this URL: <http://www.google.com/ig/api?weather=New+York,+NY>

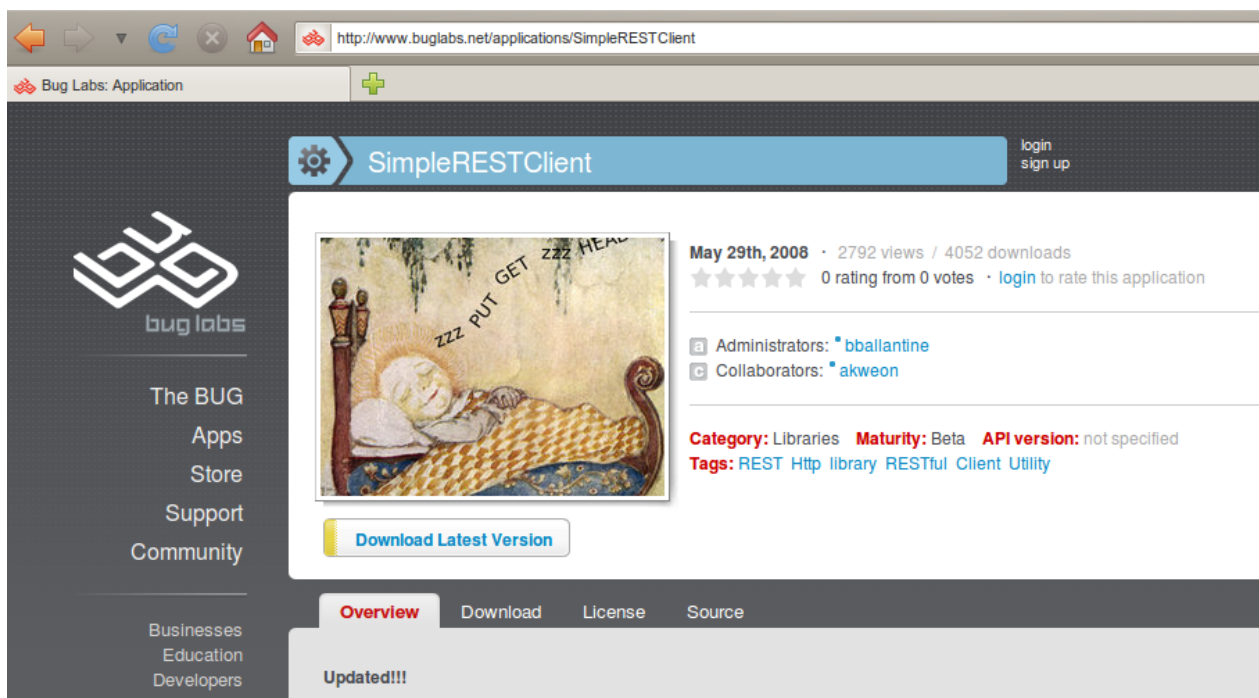
Topics Covered

- Using the SimpleRESTClient to call remote web services
- Parsing XML using XPath
- More GUI Programming

SimpleRESTClient

Our first objective is to call a remote web service, so we need a client library for making URL connections. Before you start rolling your own `java.net.URLConnection` utilities, you should know that BUG already has a nice SimpleRESTClient library. It's pretty full-featured, and it's not heavy or obtuse like the Apache HttpClient.

The SimpleRESTClient^[3] is built into the current SDK. It is useful to review the online docs for SimpleRESTClient to understand how to use it. While you are there, check out the other apps and libraries shared by the BUG community.



Create fetchWeatherDataXml method

Now it's time to implement the SimpleRESTClient. If you recall, the BugWeatherApplication class was responsible for displaying content on the screen. So far, we have a bland title "Current Weather" displayed on-screen.

For convenience, we will add all of the code we need in this tutorial in the BugWeatherApplication class. The following is a copy of our new method. Note how the connection instance is created as a DefaultConnectionProvider.

```
protected String fetchWeatherDataXml(String location) throws Exception {

    String xml = null;
    String url = WEATHER_HOST + WEATHER_LOOKUP_URL + location;
    System.out.println("url=" + url);

    try {
        IConnectionProvider connection = new DefaultConnectionProvider();
        HTTPRequest request = new HTTPRequest(connection);
        HTTPResponse response = request.get(url);
    }
}
```

```
        xml = response.getString();

        System.out.println(xml);
        return xml;

    } catch (HttpException e) {
        throw e;
    } catch (IOException e) {
        throw e;
    } catch (Exception e) {
        throw e;
    }
}
```

The static variables `WEATHER_HOST` and `WEATHER_LOOKUP_URL` are declared at the top of the class as:

```
private static final String WEATHER_HOST = "http://www.google.com";
private static final String WEATHER_LOOKUP_URL = "/ig/api?weather=";
```

Other Notes

- If you need to send username and password credentials in the request, take a look at the **BasicAuthenticationConnectionProvider** class.
- If you are expecting a larger volume of data in the `HTTPResponse`, you can also get an `InputStream` using `response.getStream()` instead of `response.getString()`

Parse the Weather XML Response

Assuming our `fetchWeatherDataXml` method works, we can go ahead and write the code to parse the XML.

Weather XML Format

Here's a truncated sample of the weather XML response returned by this url: <http://www.google.com/ig/api?weather=New+York,+NY>. Note that the primary information we are interested in is located in `current_conditions` element.

```
- <xml_api_reply version="1">
- <weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0" section="0">
- <forecast_information>
  <city data="New York, NY"/>
  <postal_code data="New York, NY"/>
  <latitude_e6 data=""/>
  <longitude_e6 data=""/>
  <forecast_date data="2010-10-09"/>
  <current_date_time data="2010-10-09 16:49:21 +0000"/>
  <unit_system data="US"/>
</forecast_information>
- <current_conditions>
  <condition data="Sunny"/>
  <temp_f data="72"/>
  <temp_c data="22"/>
  <humidity data="Humidity: 41%"/>
  <icon data="/ig/images/weather/sunny.gif"/>
  <wind_condition data="Wind: NW at 8 mph"/>
</current_conditions>
- <forecast_conditions>
  <day_of_week data="Sat"/>
  <low data="45"/>
  <high data="69"/>
  <icon data="/ig/images/weather/sunny.gif"/>
  <condition data="Sunny"/>
</forecast_conditions>
```

Create Weather class

Next, we should create a new class to hold the weather data that we extract from the XML. I chose to create an inner Weather class (at the bottom of `BugWeatherApplication.class`) that looks like this.

```
private class Weather {
    // public fields because we don't need the noise
    public String condition;
    public String temp_f;
    public String temp_c;
    public String humidity;
    public String icon;
    public String wind;
}
```

Create parseWeatherXml method

Our parseWeatherXml method will take a String xml parameter and return a Weather object, so this is where the XML parsing happens. Again, we are happy to discover that the BUG SDK provides a simple implementation of a XML DOM parser. You can read more about it in the BUG XML Howto.

Here's the code:

```
protected Weather parseWeatherXml(String xml) throws Exception {
    Weather weather = new Weather();
    try {
        XmlNode root = XmlParser.parse(xml);
        XmlNode node = root.getChild("weather").getChild("current_conditions");
        XmlNode dataNode = null;

        if (node != null) {
            // TODO: create helper method to read nodes, check nulls, get attribute
            dataNode = node.getChild("condition");
            weather.condition = dataNode.getAttribute("data");
            dataNode = node.getChild("temp_f");
            weather.temp_f = dataNode.getAttribute("data");
            dataNode = node.getChild("temp_c");
            weather.temp_c = dataNode.getAttribute("data");
            dataNode = node.getChild("humidity");
            weather.humidity = dataNode.getAttribute("data");
            dataNode = node.getChild("icon");
            weather.icon = dataNode.getAttribute("data");
            dataNode = node.getChild("wind_condition");
            weather.wind = dataNode.getAttribute("data");
        } else {
            System.out.println("current_conditions node not found");
        }
    } catch (IOException e) {
        throw e;
    } catch (Exception e) {
        throw e;
    }
    return weather;
}
```


Displaying the Weather Data

Previously, we had all the AWT code in the constructor and that is no longer practical. We need to do a bit of refactoring so that we can simply call a new method called **displayWeather** after we have fetched and parsed the weather xml data.

Create displayWeather method

Our new displayWeather method will take a Weather object as a parameter and execute the AWT code to render the content on the LCD display. We also want to organize the content better and make it somewhat pretty. Here, we chose to use a GridLayout instead of BorderLayout.

```
private void displayWeather(Weather weather) {

    // Frame declared at beginning of the class
    frame = display.getFrame();
    frame.setBackground(Color.WHITE);
    frame.setTitle("BUG Weather App");
    frame.setLayout(new GridLayout(6,2));

    Label label1 = new Label();
    label1.setText("Current Weather");
    label1.setBackground(Color.ORANGE);
    label1.setForeground(Color.BLACK);
    frame.add(label1);

    Label label2 = new Label();
    label2.setText(weather.condition);
    frame.add(label2);

    Label label3 = new Label();
    label3.setText("Temperature:" + weather.temp_f + " F");
    frame.add(label3);

    Label label4 = new Label();
    label4.setText(weather.humidity);
    frame.add(label4);

    Label label5 = new Label();
    label5.setText(weather.wind);
    frame.add(label5);

    frame.setVisible(true);
    frame.pack();
}
```

Refactor the LCDApp constructor

Finally, it's time to revisit the LCDApp constructor and integrate our new methods. We can rip out all of the AWT code and just call our new methods now. Like this:

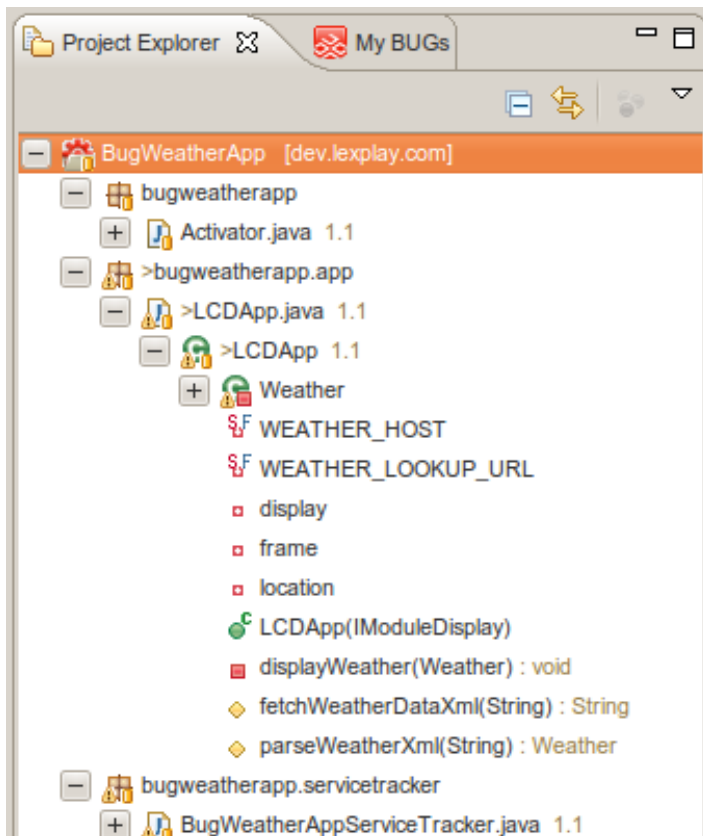
```
public LCDApp(IModuleDisplay display) {
    super();
    this.display = display;

    try {
        location = "New York, NY";
        location = URLEncoder.encode(location, "UTF-8");
        String xml = this.fetchWeatherDataXml(location);

        if (xml != null) {
            Weather weather = this.parseWeatherXml(xml);
            this.displayWeather(weather);
        }
    } catch (Exception e) {
        // TODO Display or log error
        e.printStackTrace();
    }
}
```

Build, Deploy and Test

At long last, we are ready to run the app in BUG Simulator. So far, we have added 3 methods and one inner class. In the Eclipse Project Explorer, your code base should look like this.



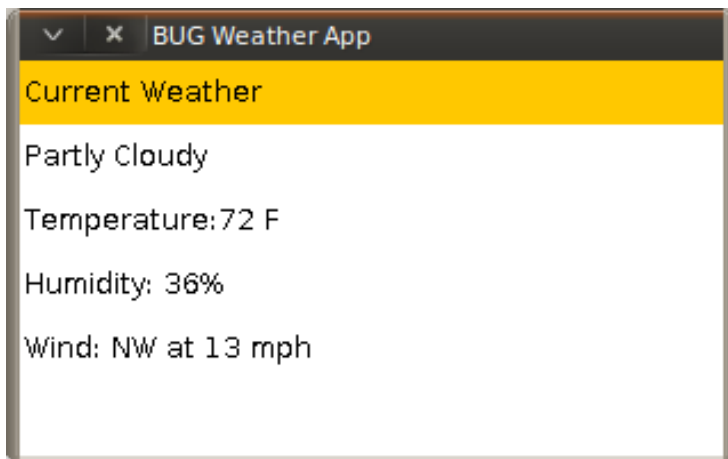
Updating MANIFEST.MF

If you went ahead and tried to run the app on the BUG Simulator, you may have ~~crashed and burned~~ discovered an OSGi configuration problem. Hint, it's one of the library dependencies that we added. Below is a copy of the updated and corrected MANIFEST.MF file. A new entry for **com.buglabs.util.simplerestclient** must be added (*see last line*). After you add this, you can deploy to the BUG Simulator using the **Send to BUG** menu command.

```
Manifest-Version: 1.0
Bundle-Name: BugWeather
Bundle-Activator: bugweather.Activator
Bundle-SymbolicName: BugWeather
Bundle-Version: 1.0.0
Bundle-Vendor: hlang
Bug-Bundle-Type: Application
BUG-API-Version: 1.4.4
Import-Package: com.buglabs.bug.base.pub,
               com.buglabs.bug.module.lcd.pub,
               org.osgi.framework,
               org.osgi.util.tracker,
               com.buglabs.application,
               com.buglabs.util,
               com.buglabs.util.simplerestclient
```

The Results

After adding the LCD module in the BUG Simulator (using the right-click menu and selecting **BUG Simulator > Slot 1 > LCD**), you will hopefully see something like this:



Not too shabby.

Extra Credit

If you feel like showing off, try out some of these extra credit features:

- Display the weather image icon for current weather conditions
- Display the weather forecast for the next few days
- Add form inputs to the display to let the user set the location.
- Periodically update the weather data.

Next Steps

- You may want to check out the JavaDoc ^[1] to start doing advanced stuff.

References

[1] http://wiki.buglabs.net/index.php/Software:SDK_Install_Guide

[2] <http://blog.programmableweb.com/2010/02/08/googles-secret-weather-api/>

[3] <http://www.buglabs.net/applications/SimpleRESTClient>

Resources

Changelog

Changelog for release 2.1.0

Features

- Added bugdash2 admin interface
- Added networking gui application
- Added BUG User Guide
- Added sample camera application
- Added ppp connection scripts

Enhancements

- Improved New Project Wizard GUI
- Improved touchscreen accuracy
- Improved wireless and sd card throughput
- Updated kernel to 2.6.35
- Updated angstrom to 2011.03
- Enabled opkg update / opkg install for third party-applications
- Switched to connman for networking
- Video module uses monitor EDID for mode setting
- USB midi support enabled
- ppp configurations for common 3/4g modems included by default

Bugfixes

- Fixed keyboard display
 - Fixed application upload (un)reliability
 - Fixed BUG Simulator launching from the wizard
 - Fixed openembedded build scripts to respect local.conf
 - Fixed openembedded build of ecj-bootstrap
 - Removed unused matchbox apps
 - Pass more OSGi unit tests
 - Pass some checkstyle tests
 - Bluetooth now uses device MAC address instead of generic
 - Fixed wireless driver watchdog timeout messages
 - Fixed ethernet register write errors
 - Fixed BMI initialization corner case error
 - Suppressed BQ27500 (battery fuel gauge) debug messages
-

Known Issues

- When booting with the stinger without ethernet and power plugged in, wireless does not autoconnect. Boot with the power off, then plug it in once wireless is connected, or use `/usr/lib/connman/test-connman`
- Ad-hoc wireless mode is not supported in the networking gui
- Some monitors return broken EDID strings. This may result in the video driver choosing a lower setting. If completely corrupt EDID data is returned, the display will default to 1024x768.
- When configuring wireless in the Networking configuration app: We recommend you click the show password button before entering the access point passphrase. When you click in the passphrase entry field, the on screen keyboard will come up. Scroll down to see the passphrase entry field (Use the scroll bar in the upper right hand side of the screen, under the close window X)

Contact Information

General Contact Info

- Get more information about BUG by emailing us at info@buglabs.net
- Get addresses and phone numbers at [<http://buglabs.net/contact>]
- Email us about buying in bulk at bulksales@buglabs.net
- Sign up for updates ^[1]

Community & Development Support

- Sign up ^[2] for our Developer Mailing List
- Meet the BUG team! ^[3]
- Email our support staff at support@buglabs.net
- Post to our community forums ^[4]
- Chat live on IRC channel #buglabs...
 - using irc.freenode.net ^[5]
 - via a web client ^[6]
 - or using your favorite IRC client.

BUG+EDU

- Sign up ^[7] for our Educators Mailing List
- BUG+EDU Information ^[8] about discounts, student programs, and campus tours
- Email us about BUG+EDU at edu@buglabs.net

Press & Marketing

- Press ^[9] – Press images, contact info, logos, etc
 - Marketing – Email our marketing team at marketing@buglabs.net
-

More BUG Links

- BUG home: <http://buglabs.net> ^[10]
- BUG blog: <http://bugblogger.com> ^[11]
- Community Website: <http://community.buglabs.net> ^[12]
- Test Kitchen ^[13]

References

- [1] <http://buglabs.net/subscribe>
- [2] <http://lists.buglabs.net/mailman/listinfo/bug-dev/>
- [3] <http://buglabs.net/team>
- [4] <http://community.buglabs.net/forums/>
- [5] <irc://irc.freenode.net/buglabs>
- [6] <http://java.freenode.net//index.php?channel=buglabs>
- [7] <http://lists.buglabs.net/mailman/listinfo/bug-edu/>
- [8] <http://buglabs.net/edu>
- [9] <http://buglabs.net/press>
- [10] <http://buglabs.net>
- [11] <http://bugblogger.com>
- [12] <http://community.buglabs.net/>
- [13] <http://www.buglabs.net/testkitchen/>

Javadocs

Current Version

This page links to Javadoc HTML pages that were generated from source files created by Bug Labs and others that are part of the BUG Java/OSGi system, the SDK, or other BUG tools.

The most current version of the Javadocs can always be found in the **/development/javadoc/current/** path. *This link currently points to the 2.0 release of the SDK*

- <http://bugcommunity.com/development/javadoc/current/>

Prior Versions

The Javadocs for prior versions of the BUG SDK can be found here:

- <http://bugcommunity.com/development/javadoc/>
-

Glossary

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

ADC

Analog-to-Digital Converter: An analog-to-digital converter (abbreviated ADC, A/D or A to D) is a device that converts a continuous quantity to a discrete digital number. The reverse operation is performed by a digital-to-analog converter (DAC). See also: http://en.wikipedia.org/wiki/Analog-to-digital_converter

ARM

The CPU architecture in the BUG. Many PDAs and phones use ARM CPUs. The BUG uses the TI OMAP processor. See also: http://en.wikipedia.org/wiki/Texas_Instruments_OMAP

Angstrom

Ångström is a stable Debian-like Linux distribution for embedded devices like handhelds, set top boxes and network-attached storage devices. <http://www.angstrom-distribution.org/>

AWT

The Abstract Window Toolkit (AWT) is Java's original platform-independent GUI toolkit. http://en.wikipedia.org/wiki/Abstract_Window_Toolkit

B

bitbake

BitBake is a simple tool for the execution of tasks. It is derived from Portage, which is the package management system used by the Gentoo Linux distribution. It is most commonly used to build packages, and is used as the basis of the OpenEmbedded project. <http://en.wikipedia.org/wiki/BitBake>

BMI

BUG Module Interface: the four connectors on the BUGbase unit that connect to BUGmodules. The BMI connector acts as a hot-pluggable bus in Linux similar to USB. The pinouts from the BMI connector map to common generally available interfaces such as USB, GPIO, and I2C.

BOM

A bill of materials (BOM) is a list of the raw materials, sub-assemblies, intermediate assemblies, sub-components, components, parts and the quantities of each needed to manufacture an end product. You may see references to "BOM" in the hardware module specs. http://en.wikipedia.org/wiki/Bill_of_materials

Bootloader

A bootloader is the first software that runs on a computer. On a desktop PC this is normally referred to as a BIOS. The BUG originally shipped with Redboot ^[1] bootloader but has since moved to Uboot ^[2] to load the Linux operating system.

BUG Application

Application created in Dragonfly SDK. Application can be transferred for execution to BUG. It can also be transferred to BUGnet so it can be shared with the community.

BUGbase

The BUGbase is the foundation of your BUG device, you can think of it as the "brains" of your gadget. More info within the products section of our website ^[3].

BUGmodules

Each BUGmodule represents a functional component of a BUG device. A BUGmodule could be a digital camera, a GPS receiver, a keyboard, a bar code reader, etc. More info within the products section of our website ^[3].

BUGnet

BUGnet ^[4] is our online community, including the repository for sharing applications.

BUG Simulator

BUG Simulator is a virtual representation of BUG in the Dragonfly SDK. When developing BUG applications, users can deploy a compiled application onto the BUG Simulator to test basic functionality.

Bundle

A bundle is an OSGi term for an application or component. It is a discreet software component. The BUG application layer is a set of OSGi bundles.

C

CAN

Controller–area network (CAN or CAN-bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other within a vehicle without a host computer. CAN is a message based protocol, designed specifically for automotive applications but now also used in other areas such as industrial automation and medical equipment. http://en.wikipedia.org/wiki/Controller_area_network

Concierge

An R3 OSGi implementation used in the BUG. Not implemented in BUG Y.T. OSGi services now provided by Apache Felix. See also: <http://concierge.sourceforge.net>.

Configuration Admin

Upon unregistering a bundle from the OSGi Framework all configuration data that might have been set to define its state is gone. To relieve user of pains of reconfiguring the bundle once its active, Configuration Admin can be used to persist configuration data so that upon the bundle's being active again its states can be set to those when they were

before bundle was unregistered.

D

DAC

A **digital-to-analog converter** is a device that converts a digital (usually binary) code to an analog signal (current, voltage, or electric charge). An analog-to-digital converter (ADC) performs the reverse operation. http://en.wikipedia.org/wiki/Digital-to-analog_converter

E

Eclipse

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Dragonfly (the BUG SDK) is a collection of plug-ins that contribute to Eclipse Platform. More info about the Eclipse Platform can be found here ^[5].

EDGE

Enhanced Data rates for GSM Evolution (EDGE) is a 3G digital mobile phone technology that allows improved data transmission rates as a backward-compatible extension of GSM. <http://en.wikipedia.org/wiki/EDGE>

EEPROM

Electrically Erasable Programmable Read-Only Memory. <http://en.wikipedia.org/wiki/EEPROM>

Ethernet via USB

A method of networking which uses the USB bus as though it were ordinary Ethernet hardware.

F

Felix

Apache Felix is an open-source implementation of the OSGi R4 Service Platform and other OSGi-related technologies under the Apache license. <http://felix.apache.org/site/index.html>

FTDI

Refers to the company FTDI which specializes in converting legacy peripherals to Universal Serial Bus (USB). <http://www.ftdichip.com/>

G

Gerber File

A file format used by printed circuit board(PCB) manufacturing machines to lay out electrical connections. http://en.wikipedia.org/wiki/Gerber_File

GPIO

General Purpose Input Output: An embedded bus used on the BUG. The BMI connector exposes GPIO ports. See also: <http://en.wikipedia.org/wiki/GPIO>

GPS

Global Positioning System. The BUGlocate module provides GPS services for BUG

GSM

GSM is the most popular standard for mobile telephony systems in the world. GSM is a cellular network, which means that mobile phones connect to it by searching for cells in the immediate vicinity. There are five different cell sizes in a GSM network—macro, micro, pico, femto and umbrella cells. <http://en.wikipedia.org/wiki/GSM>

gspca

Generic Software Package for Camera Adapters: For working with USB webcams. <http://mxhaard.free.fr/sview.html> and <http://www.linuxjournal.com/video/get-your-webcam-working-gspca>

GUI

Graphical User Interface: A way to interact with a system visually through icons, menus, and pictures.

H

HSDPA

High-Speed Downlink Packet Access (HSDPA) is an enhanced 3G (third generation) mobile telephony communications protocol in the High-Speed Packet Access (HSPA) family, also dubbed 3.5G, 3G+ or turbo 3G, which allows networks based on Universal Mobile Telecommunications System (UMTS) to have higher data transfer speeds and capacity. http://en.wikipedia.org/wiki/High-Speed_Downlink_Packet_Access

I

I2C

Refers to I²C (Inter-Integrated Circuit). This serial bus is an interface that is exposed in the BMI connector. See also: <http://en.wikipedia.org/wiki/I%C2%B2C>

ioctl

In computing, ioctl, short for Input/output control, is a system call for device-specific operations and other operations which cannot be expressed by regular system calls. It takes a parameter specifying a request code; the effect of a call depends completely on the request code. Request codes are often device-specific. For instance, a CD-ROM driver which can get a physical device to eject a disc would provide an ioctl request code to do that. Device-independent request codes are sometimes used to give userland access to kernel functions which are only used by core system software or still under development. <http://en.wikipedia.org/wiki/Ioctl>

IOX

I/O Expander found in BUGvonHippel

ipkg

(Replaced with opkg) ipkg, or the Itsy Package Management System, was a lightweight package management system designed for embedded devices that resembled Debian's dpkg. <http://en.wikipedia.org/wiki/Ipkg>

IPU

Image Processing Unit: a component in the BUG that handles image processing.

J

JAE

Referenced in BUGstinger module specs as "JAE Pin Number". Probably refers to: <http://www.jae.com/jaehome.htm>

Jar

A jar is essentially a compressed zip file that contains Java code. The Dragonfly SDK packages BUG applications in jar files.

JNI

The Java Native Interface (JNI) is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly. <http://en.wikipedia.org/wiki/JNI>

jSLP

A SLP implementation used in the BUG and Dragonfly (the BUG SDK). SLP allows Dragonfly (the BUG SDK) to discover BUGs on the local network. <http://jslp.sourceforge.net>

JTAG

JTAG is a serial protocol, similar to SPI in some respects, that is used for boundary scan testing, in circuit emulation, and flash programming. <http://www.freelabs.com/~whitis/electronics/jtag/>

JVM

Java Virtual Machine: a program that executes Java programs. BUG uses OpenJDK 6 ^[6]

K

Kernel

Short for the Linux Kernel. BUG runs Linux version 2.6.19.

KProbes

Referenced here: `Software:KProbes_on_BUG_YT` See also: <http://www.mjmwired.net/kernel/Documentation/kprobes.txt>

L

LLVM

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. <http://llvm.org/>

M

Manifest File

In an OSGi bundle, the **MANIFEST.MF** file is where metadata is stored. This file is in the `/META-INF` directory of the jar.

minicom

Minicom is a text-based modem control and terminal emulation program for Unix-like operating systems. <http://en.wikipedia.org/wiki/Minicom>

Modlet

A modlet is a piece of Java code that gets executed each time a module is inserted or removed from a BUGbase.

Mongrel

Mongrel is an application server for Ruby web applications. It is often used behind more general web servers such as Apache or nginx in a clustered manner. In this arrangement, the general web server typically serves static content and hands off requests for dynamic content to the mongrel cluster.

MMCcard

The original BUG used an MMC card that contained the Linux Root files system (rootfs). The current version of the BUGbase uses SDmicro cards which contains the linux kernel (for uboot) and rootfs. User applications and data is also stored on this card.

N

NMEA

NMEA is a combined electrical and data specification for communication between marine electronic devices such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the U.S.-based National Marine Electronics Association. The NMEA standard uses a simple ASCII, serial communications protocol that defines how data is transmitted in a "sentence" from one "talker" to multiple "listeners" at a time. http://en.wikipedia.org/wiki/NMEA_0183

nginx

A lightweight, fast web server that has become very popular for hosting Ruby on Rails. It is pronounced "engine X". [Nginx wiki](#) ^[7]

O

OBD

On-Board Diagnostics, or OBD, in an automotive context, is a generic term referring to a vehicle's self-diagnostic and reporting capability. OBD systems give the vehicle owner or a repair technician access to state of health information for various vehicle sub-systems. http://en.wikipedia.org/wiki/On-board_diagnostics

OMAP3

Open Multimedia Application Platform is a category of proprietary microprocessors that has capabilities for portable and mobile multimedia applications and that is developed by Texas Instruments. Many mobile phones use OMAP microprocessors. <http://en.wikipedia.org/wiki/OMAP3#OMAP3>

OpenEmbedded (OE)

OpenEmbedded, the build framework for embedded Linux. OpenEmbedded offers a cross-compile environment and allows developers to create a complete Linux Distribution for embedded systems. http://wiki.openembedded.org/index.php/Main_Page

openjdk6

OpenJDK 6 is an open source implementation of the Java SE 6 specification. <http://openjdk.java.net/projects/jdk6/>

opkg

Opkg is a lightweight package management system. It is written in C and resembles apt/dpkg in operation. It is intended for use on embedded Linux devices and is used in this capacity in the OpenEmbedded and OpenWrt projects. <http://code.google.com/p/opkg/>

OSGi

Open Services Gateway Initiative ^[8]. A Java-based specification that describes a runtime and component model. In BUG Y.T., BUG applications run on an OSGi implementation called Apache Felix ^[4].

OSGi Console

A console interface to the OSGi runtime. This console lets a user inspect the current state of the runtime system, start and stop bundles, and see what services are available. In addition, developers can add their own commands.

P**poky**

Refers to the build tool used in previous BUG software. <http://www.pokylinux.org/about/>

pppd

PPP is the protocol used for establishing internet links over dial-up modems, DSL connections, and many other types of point-to-point links. The pppd daemon works together with the kernel PPP driver to establish and maintain a PPP link with another system (called the peer) and to negotiate Internet Protocol (IP) addresses for each end of the link. <http://linux.die.net/man/8/pppd>

Q**QSE/QTE**

Refers to Samtec QSE/QTE connectors. <http://www.samtec.com/ProductInformation/TechnicalSpecifications/Overview.aspx?series=QTE>

R**RDAC**

A digital potentiometer is a digitally-controlled electronic component that mimics the analog functions of a potentiometer. It is often used for trimming and scaling analog signals by microcontrollers. Sometimes this device is also referred to as an RDAC, Resistive Digital-to-Analog Converter. <http://en.wikipedia.org/wiki/RDAC>

REST

Stands for Representational State Transfer. A style of web service architecture that uses HTTP verbs for CRUD (Create, Read, Update, Delete) actions on addressable resources. Resources are addressable via a uri. Typically these uri's are descriptive and easy to read. REST style resources are often called RESTful and they tend to be simpler than RPC-style web services, such as those that depend on SOAP.

Rootfs

Shorthand for Root Filesystem. This is the data structure where all files are stored on BUG.

RS232

Commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors. <http://en.wikipedia.org/wiki/RS-232>

RxTx

RXTX is a native lib providing serial and parallel communication for the Java Development Toolkit (JDK). All deliverables are under the gnu LGPL license. <http://www.rxtx.org/>

S**Samtec**

Refers to the Samtec connectors which allow BUG modules to plug into the BUG base. <http://www.samtec.com/ProductInformation/TechnicalSpecifications/Overview.aspx?series=QTE>

Sewing

Simple Embedded Web framework for OSGi and the BUG. It simplifies the creation of dynamic web applications. See also `com.buglabs.osgi.sewing` on BUGnet ^[9]

SDK

Software Development Kit. The BUG SDK, also known as Dragonfly, is what you use to write applications for BUG.

SDmicro Card (Secure digital)

The current version of the BUGbase uses SDmicro cards which contains the linux kernel (for uboot) and rootfs. User applications and data is also stored on this card.

SLP

Service Location Protocol. A discovery protocol that allows various networked services to know about each other. http://en.wikipedia.org/wiki/Service_Location_Protocol

SoC

Short for "System on a Chip". The BUG CPU, Freescale's i.MX31, is a SoC. It combines several devices and interfaces that are typically found on separate chips in traditional designs.

SPI

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a "four wire" serial bus. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

SSH

Secure Shell: Network protocol used between two networked devices to directly access shell accounts (command-line user account).

Swing

Swing is the primary Java GUI widget toolkit and is included in OpenJDK6, the standard Java environment on BUG. Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). http://en.wikipedia.org/wiki/Swing_%28Java%29

SWT

The Standard Widget Toolkit (SWT) is a graphical widget toolkit for use with the Java platform. It was originally developed by IBM and is now maintained by the Eclipse Foundation. It is an alternative to the AWT and Swing Java GUI toolkits provided by Sun Microsystems as part of the Java Platform, Standard Edition. http://en.wikipedia.org/wiki/Standard_Widget_Toolkit

SystemTap

SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel. <http://sourceware.org/systemtap/>

T

TCP/IP over USB

Linux allows a special USB kernel module (g_ether) to be loaded that causes a USB connection between BUG and a computer to be used as a network connection.

toolchain

A toolchain is the set of programming tools that are used to create a product (typically another computer program or system of programs). The tools may be used in a chain, so that the output of each tool becomes the input for the next. A simple software development toolchain consists of a text editor for editing source code, a compiler and linker to

transform the source code into an executable program, libraries to provide interfaces to the operating system, and a debugger. <http://en.wikipedia.org/wiki/Toolchain>

U

u-boot

U-Boot is a universal boot loader for embedded boards based on PowerPC, ARM, MIPS and several other processors. <http://www.denx.de/wiki/U-Boot> <http://www.linux-mips.org/wiki/U-Boot>

UMTS

Universal Mobile Telecommunications System (UMTS) is one of the third-generation (3G) mobile telecommunications technologies, which is also being developed into a 4G technology. http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System

USB

Universal Serial Bus. BUG has one USB OTG (on the go) port. This port can be used to connect external peripheral such as hard drives and web cams. Additionally when connected to a computer with the correct settings, the USB connection can be used as a network device. http://en.wikipedia.org/wiki/Universal_Serial_Bus

uvc

The USB video device class (also USB video class or UVC) is a USB device class that describes devices capable of streaming video like webcams, digital camcorders, transcoders, analog video converters, television tuners, and still-image cameras. http://en.wikipedia.org/wiki/USB_video_device_class

V

Virtual BUG

BUG Simulator was formerly known as Virtual BUG. It is a virtual representation of BUG in Dragonfly (the BUG SDK).

VNC

Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the RFB protocol to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network. <http://en.wikipedia.org/wiki/VNC>

W

Web Service

Web API, remotely accessible over a network.

X

x11vnc

x11vnc allows one to view remotely and interact with real X displays (i.e. a display corresponding to a physical monitor, keyboard, and mouse) with any VNC viewer. The BUG operating system includes x11vnc, which you can use for connecting to your BUG with a VNC client. See also: [Software:Connecting to BUG with VNC](#)

References

- [1] <http://en.wikipedia.org/wiki/Redboot>
 - [2] http://en.wikipedia.org/wiki/Das_U-Boot
 - [3] <http://buglabs.net/products>
 - [4] <http://buglabs.net/login>
 - [5] <http://eclipse.org>
 - [6] <http://openjdk.java.net/projects/jdk6/>
 - [7] <http://wiki.codemongers.com/Main?action=show&redirect=Nginx>
 - [8] <http://osgi.org>
 - [9] <http://www.buglabs.net/applications/com.buglabs.osgi.sewing>
-

Troubleshooting

BUG Troubleshooting

Bug Support IRC

The easiest way to get real-time help from bug developers is to visit our IRC channel.

- **Hostname:** irc.freenode.net
- **Channel:** #buglabs

If you are unfamiliar with IRC, you can connect through the Freenode webchat at <http://webchat.freenode.net/>. After entering a nickname to identify yourself and the **#buglabs** channel, you will be able to get support help from the Bug Labs team.

Normal BUG Boot Behavior

1. Engage the power button
2. Front LEDs will flash white (go on and off) once
3. FIXME waiting on file from Mateo
4. BUGbase BUG logo will illuminate blue, and appear to be breathing:
[[Image:]]
5. When BUG is booted fully the BUG logo will illuminate blue and steady.
6. If the LCD module is attached, it will power up at kernel/Uboot load time, so the LCD's backlight will be on. There will be no display on the LCD module. This is normal.
7. Provided you have calibrated BUGview, the LCD module should display this:



The sequence of the BUG booting:

1. Power on->Uboot->Load Kernel->mount rootfs on Memory Card->load OS->load OSGi.
2. Step 1 regards power. If there is a failure here, it is likely a fail in power.
3. Other steps regard Uboot and the Kernel loading. If BUG hangs, it is likely that the Root file system on your Memory Card is corrupted.
4. If you have BUGstinger plugged in, you can view the boot sequence in minicom

Network

Problem: I don't know the password for telnet/ssh on my BUG.

Solution: By default, the root login is not set and will accept anything including blank.

Problem: What is my BUG's IP?

There are a number of ways to lookup your BUG's IP address.

Solution 1: If you have connected your BUG to an Ethernet network through the BUGstinger, it will automatically try to obtain an IP address via DHCP. On startup, BUG also broadcasts its available services to your local network, which means you might be able to ping it using the default host name **bug20.local**.

```
HUGH:etc hlang$ ping bug20.local
PING bug20.local (10.7.203.59): 56 data bytes
64 bytes from <red>10.7.203.59</red>: icmp_seq=0 ttl=64 time=4.723 ms
```

Solution 2: If that does not work, you should connect to your BUG with a serial connection. This requires a BUGstinger dock and a USB cable connecting your computer and the BUGstinger.

- Linux users, please refer to Software:Ubuntu-Debian_Serial_Terminal.
- Mac users, try this: Software:Mac Setup Guide
- Windows users, you're out of luck. Just kidding. Read this: Software:Windows Setup Guide

Problem: I can't connect to my BUG via wifi.

This may be the result of any of the following:

- The **wlan0** network interface was not loaded successfully
- Your wireless network has not been configured or requires a password

Solution: Please refer to the Software:Networking page and review the information there.

BUGbase:

Problem: My bug won't boot.

Solution: Try to find out at what state the boot failure is occurring.

1. If BUG freezes before the kernel boots, make sure the System SD card is in the System slot. We do not recommend taking out the SD card out of the System slot unless you fully understand the BUG kernel and rootfs. Ensuring the SD card is in the System slot, your Memory card is corrupted and needs to be flashed.
1. If BUG freezes at any other point, it means that the Memory card may have been corrupted in the /opt/concierge folder. You can telnet/ssh into BUG and cat /var/log/concierge.log. If there are many exceptions, this most likely indicates that the Memory card is corrupted. Please follow these instructions to upgrade your BUG Memory card.

Problem: My BUG won't charge.

Solution: More than likely this is a problem with the AC adapter. If the battery status indicator (located on the front of your BUG) intermittently illuminates, this is an indication that you have a bad AC adapter. If your BUG is showing a full battery with a green battery status indicator and your BUG shuts down when the AC adapter is unplugged, this may be an indication that the BUGbase has a power problem. Either way you should contact our support team ^[1].

Problem: I can connect to BUG via serial terminal, but I am having problems with hardware/software/networking

Solution: Review the log files and look for errors. Log files are found in /var/log and you can review the contents of these to help troubleshoot. If you know key words you are looking for (such as "lcd" in the example below), try using grep like this:

```
grep lcd /var/log/*
```

Also, you can try this command:

```
dmesg grep lcd
```

Problem: BUG is not recognizing a new hardware module

Solution: You probably need to "program the EEPROM" so that your BUG sees the correct vendor/product identifiers. Please read: Programming EEPROM on BUG YT

Problem: A software package I need is missing

Use **opkg** to search for and install the needed packages. You can browse the available packages are here: <http://repo.buglabs.net/2.0/>

Sample commands:

```
opkg update
```

```
opkg search libcrypto
```

```
opkg install
```

BUGstinger:**Problem: The ethernet port is not working**

Solution: Make sure that the BUG AC adaptor is plugged into the BUG and is providing power to the BUG. You should verify that the power plug is seated firmly. The ethernet port on the BUGstinger will not operate unless the BUG is plugged into a power source. When it is working, you will see the standard blinking lights next to the ethernet port.

You can also try disconnecting and reconnecting the BUGstinger 30-pin connector in the BUG base.

Problem: The BUGstinger is not working and I cannot connect via serial terminal

Solution: There's a reset button on the under side of the BUGstinger. It looks like a tiny hole. Use the tip of a pen or paperclip to press the reset button.

BUGlocate:

Problem: I can't get a GPS fix.

Solution: This is due to the large amount of EM noise generated by the BUGbase. We are in the process of choosing an appropriate external antenna to ship to all users who have bought the Hiro P edition of the BUGbundle.

BUGview

Problem: My LCD module doesn't clip into slots 0, 2 or 3 and the connector is backwards.

Solution: This is by design. On our new BUG Y.T. 2.0 bases, there is a special video slot (Slot 1) which is the only slot BUGview2 can attach to. If you have an older BUGview, you will need to purchase the new BUGview2 ^[2] for your BUG Y.T. The BUGview2 modules will not work with older editions of BUG (BUGwifi or Hiro P.). support team ^[1].

BUGmotion

Problem: BUG Y.T. does not recognize my BUGmotion module from an older BUG

Solution: BUGmotion is no longer supported by BUG Y.T. Please use the accelerometer in BUGview2.

Problem: I need high-resolution accelerometer data. The API doesn't support it.

Solution: The old BUG API (R1.1, SDK 1.0.0.5) had an elementary Accelerometer API. It is now much more robust. Please refer to BUGlevel application ^[3] for an example of how to access the hardware at very high resolutions. Also see: Accelerometer API Specification ^[4]

30-pin Connector

Problem: What is the 30-pin connector for on my BUG Y.T?

Solution: The 30-pin connector is for BUGstinger ^[5]. BUGstinger exposes a serial interface and some other peripherals. We strongly recommend BUGstingers for developers.

Applications:

Problem: My application won't start.

Solution: Ensure that you have networking set up, that you sent the application to your BUG's IP in the Dragonfly (the BUG SDK), and that the required modules for the application are attached to your BUG. A great resource for any application or programming questions is to check out our discussion forums ^[6].

Eclipse

Problem: I installed or upgraded Firefox 3 on my Linux machine and now my Eclipse Browser doesn't work.

Solution: From the command line execute: `sudo apt-get install xulrunner`

VNC

Problem: x11vnc crashes on the bug

Solution: try 'export DISPLAY=localhost:0.0' on the bug and run x11vnc again

References

- [1] <mailto:support@buglabs.net>
- [2] <http://store.buglabs.net/Product-Catalog/BUGview2>
- [3] <http://www.buglabs.net/applications/BUGLevel>
- [4] http://bugcommunity.com/wiki/images/b/b3/AccelerometerAPI_v1_1.pdf
- [5] <http://store.buglabs.net/Product-Catalog/BUGstinger>
- [6] <http://bugcommunity.com/forums/login.php>

Article Sources and Contributors

BUG YT Start Guide *Source:* <http://wiki.buglabs.net/index.php?oldid=3063> *Contributors:* Alicia, Aturley, FarMcKon, Hlang, Jason.gourley, Jedahan

BUGdash Admin Tool *Source:* <http://wiki.buglabs.net/index.php?oldid=3007> *Contributors:* Akweon, Hlang, Jedahan

Networking *Source:* <http://wiki.buglabs.net/index.php?oldid=3196> *Contributors:* Hlang, Jconnolly, Jedahan, Mgrundy

Serial Terminal *Source:* <http://wiki.buglabs.net/index.php?oldid=3064> *Contributors:* Hlang, Jconnolly, Jedahan

Connecting with VNC *Source:* <http://wiki.buglabs.net/index.php?oldid=3021> *Contributors:* Hlang, Jconnolly, Jedahan

Develop with Java *Source:* <http://wiki.buglabs.net/index.php?oldid=3046> *Contributors:* Hlang, Jconnolly, Jedahan

Getting Started with OSGi *Source:* <http://wiki.buglabs.net/index.php?oldid=3031> *Contributors:* Aturley, Hlang, Jedahan

SDK Install Guide *Source:* <http://wiki.buglabs.net/index.php?oldid=3010> *Contributors:* Bugvish, FarMcKon, Hlang, Jason.gourley, Jconnolly, Jedahan, Mgrundy

Interacting with BUGs in Dragonfly SDK *Source:* <http://wiki.buglabs.net/index.php?oldid=3034> *Contributors:* Bcruskie, Hlang, Jason.gourley, Jedahan

Using Modules in BUG Simulator *Source:* <http://wiki.buglabs.net/index.php?oldid=3036> *Contributors:* Hlang, Jason.gourley, Jedahan

Interacting with BUGnet *Source:* <http://wiki.buglabs.net/index.php?oldid=3037> *Contributors:* Hlang, Jedahan

Getting Started Tutorial *Source:* <http://wiki.buglabs.net/index.php?oldid=3066> *Contributors:* CaptainMcCrank, Jedahan

Changelog *Source:* <http://wiki.buglabs.net/index.php?oldid=3119> *Contributors:* Jedahan, Mgrundy

Contact Information *Source:* <http://wiki.buglabs.net/index.php?oldid=3042> *Contributors:* Hlang, Jedahan

Javadococs *Source:* <http://wiki.buglabs.net/index.php?oldid=3040> *Contributors:* Hlang, Jedahan

Glossary *Source:* <http://wiki.buglabs.net/index.php?oldid=3018> *Contributors:* Alicia, Hlang, Jconnolly, Jedahan

BUG Troubleshooting *Source:* <http://wiki.buglabs.net/index.php?oldid=3065> *Contributors:* Alicia, Hlang, Jedahan

Image Sources, Licenses and Contributors

File:BUG_YT_Hook-n-Snap.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:BUG_YT_Hook-n-Snap.jpg *License:* unknown *Contributors:* Hlang

File:BUG_YT_callouts.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:BUG_YT_callouts.jpg *License:* unknown *Contributors:* Hlang, Jconnolly

Image:BUGdash_appThumb.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:BUGdash_appThumb.jpg *License:* unknown *Contributors:* Hlang

Image:Bugdash overview sm.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:Bugdash_overview_sm.jpg *License:* unknown *Contributors:* Hlang

Image:More tab.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:More_tab.jpg *License:* unknown *Contributors:* Hlang

Image:Morehistory.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:Morehistory.jpg> *License:* unknown *Contributors:* Hlang

Image:Bugdash20-hardware-bugmodules.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Bugdash20-hardware-bugmodules.png> *License:* unknown *Contributors:* Akweon

Image:Bugdash-bug-apps-login.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Bugdash-bug-apps-login.png> *License:* unknown *Contributors:* Akweon

Image:Bugdash-home-login.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Bugdash-home-login.png> *License:* unknown *Contributors:* Akweon

Image:Bugdash-filebrowser.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Bugdash-filebrowser.png> *License:* unknown *Contributors:* Akweon

File:NetApp01.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp01.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp10.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp10.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp03.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp03.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp04.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp04.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp05.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp05.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp08.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp08.jpg> *License:* unknown *Contributors:* Mgrundy

File:NetApp11.jpg *Source:* <http://wiki.buglabs.net/index.php?title=File:NetApp11.jpg> *License:* unknown *Contributors:* Mgrundy

Image:vnc-example.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Vnc-example.png> *License:* unknown *Contributors:* Jconnolly

Image:JVM_OSGi_stack.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:JVM_OSGi_stack.jpg *License:* unknown *Contributors:* Jconnolly

Image:Help-about-sdk.png *Source:* <http://wiki.buglabs.net/index.php?title=File:Help-about-sdk.png> *License:* unknown *Contributors:* Jconnolly

Image>About-sdk.png *Source:* <http://wiki.buglabs.net/index.php?title=File:About-sdk.png> *License:* unknown *Contributors:* Jconnolly

Image:Eclipse_Installation_Details_004.png *Source:* http://wiki.buglabs.net/index.php?title=File:Eclipse_Installation_Details_004.png *License:* unknown *Contributors:* Jconnolly

File:BUGWeatherApp_MyBUGs.png *Source:* http://wiki.buglabs.net/index.php?title=File:BUGWeatherApp_MyBUGs.png *License:* unknown *Contributors:* Hlang

File:virtual_bug.gif *Source:* http://wiki.buglabs.net/index.php?title=File:Virtual_bug.gif *License:* unknown *Contributors:* Hlang

File:staticConnection.gif *Source:* <http://wiki.buglabs.net/index.php?title=File:StaticConnection.gif> *License:* unknown *Contributors:* Hlang

File:slpBUG.gif *Source:* <http://wiki.buglabs.net/index.php?title=File:SlpBUG.gif> *License:* unknown *Contributors:* Hlang

File:refresh.gif *Source:* <http://wiki.buglabs.net/index.php?title=File:Refresh.gif> *License:* unknown *Contributors:* Hlang

File:Bug_simulator_button.gif *Source:* http://wiki.buglabs.net/index.php?title=File:Bug_simulator_button.gif *License:* unknown *Contributors:* Hlang

File:Eclipse_terminate_button.png *Source:* http://wiki.buglabs.net/index.php?title=File:Eclipse_terminate_button.png *License:* unknown *Contributors:* Hlang

File:BUGsimulator_base.png *Source:* http://wiki.buglabs.net/index.php?title=File:BUGsimulator_base.png *License:* unknown *Contributors:* Hlang

File:New_bug_connection.png *Source:* http://wiki.buglabs.net/index.php?title=File:New_bug_connection.png *License:* unknown *Contributors:* Hlang

File:New_bug_connection_address.png *Source:* http://wiki.buglabs.net/index.php?title=File:New_bug_connection_address.png *License:* unknown *Contributors:* Hlang

File:deleteBug.gif *Source:* <http://wiki.buglabs.net/index.php?title=File:DeleteBug.gif> *License:* unknown *Contributors:* Hlang

File:app_remove.gif *Source:* http://wiki.buglabs.net/index.php?title=File:App_remove.gif *License:* unknown *Contributors:* Hlang

File:app_import.gif *Source:* http://wiki.buglabs.net/index.php?title=File:App_import.gif *License:* unknown *Contributors:* Hlang

File:BUGsimulator_base_properties.png *Source:* http://wiki.buglabs.net/index.php?title=File:BUGsimulator_base_properties.png *License:* unknown *Contributors:* Hlang

File:BUGSimulator_Slot1_LCD_menus.png *Source:* http://wiki.buglabs.net/index.php?title=File:BUGSimulator_Slot1_LCD_menus.png *License:* unknown *Contributors:* Hlang

File:NewProject_ServiceDef_LCD.png *Source:* http://wiki.buglabs.net/index.php?title=File:NewProject_ServiceDef_LCD.png *License:* unknown *Contributors:* Hlang

File:Menu_sendtobug.png *Source:* http://wiki.buglabs.net/index.php?title=File:Menu_sendtobug.png *License:* unknown *Contributors:* Hlang, Jedahan

File:BUGnet_Auth_window.png *Source:* http://wiki.buglabs.net/index.php?title=File:BUGnet_Auth_window.png *License:* unknown *Contributors:* Hlang

File:SDK_BUGnet_view.png *Source:* http://wiki.buglabs.net/index.php?title=File:SDK_BUGnet_view.png *License:* unknown *Contributors:* Hlang

File:SDK_BUGnet_listing_download.png *Source:* http://wiki.buglabs.net/index.php?title=File:SDK_BUGnet_listing_download.png *License:* unknown *Contributors:* Hlang

File:SDK_BUGnet_websearch.png *Source:* http://wiki.buglabs.net/index.php?title=File:SDK_BUGnet_websearch.png *License:* unknown *Contributors:* Hlang

File:Bug_project_button.gif *Source:* http://wiki.buglabs.net/index.php?title=File:Bug_project_button.gif *License:* unknown *Contributors:* Hlang

File:BugWeather_new.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeather_new.png *License:* unknown *Contributors:* Jedahan

File:module_services.png *Source:* http://wiki.buglabs.net/index.php?title=File:Module_services.png *License:* unknown *Contributors:* Jedahan

File:BugWeather_Activator.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeather_Activator.png *License:* unknown *Contributors:* Hlang, Jedahan

File:BugWeather_manifest.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeather_manifest.png *License:* unknown *Contributors:* Jedahan

File:Menu_sendtobug.png *Source:* http://wiki.buglabs.net/index.php?title=File:Menu_sendtobug.png *License:* unknown *Contributors:* Hlang, Jedahan

File:BugWeatherApp_simulator_gui.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeatherApp_simulator_gui.png *License:* unknown *Contributors:* Hlang

File:SimpleRESTClient.png *Source:* <http://wiki.buglabs.net/index.php?title=File:SimpleRESTClient.png> *License:* unknown *Contributors:* Hlang

File:Google_weather_xml.png *Source:* http://wiki.buglabs.net/index.php?title=File:Google_weather_xml.png *License:* unknown *Contributors:* Hlang

File:BugWeatherApp_part3_project_explorer.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeatherApp_part3_project_explorer.png *License:* unknown *Contributors:* Hlang

File:BugWeatherApp_LCD_beta3.png *Source:* http://wiki.buglabs.net/index.php?title=File:BugWeatherApp_LCD_beta3.png *License:* unknown *Contributors:* Hlang

Image:Gui_for_R1.4.3.jpg *Source:* http://wiki.buglabs.net/index.php?title=File:Gui_for_R1.4.3.jpg *License:* unknown *Contributors:* Hlang