Pimp your Pi
with Eclipse RCP for your Home TV
Who’s Talking?

Jed Anderson
Development Manager
Secure Delivery Center

Timothy Webb
Director of Innovation &
Eclipse Enthusiast
Obligatory Disclaimer

• We’re here because we love Eclipse.
• Yes, we could plug Genuitec, and our awesome products like MyEclipse and Secure Delivery Center…
• But this talk is all about the fun you can have with Pi and open source!

• MyEclipse is your fully loaded IDE, ready to go. Built on Eclipse and optimized for a broad array of development tasks including rich Java EE.
• Secure Delivery Center lets you eliminate engineering overhead by simply packaging up Eclipse with add-ons and configuration, and delivering it to a team, or your end-users.
Agenda

- Introducing PiPlug
- Building your first PiPlug application
- Understanding performance on the Raspberry Pi
- PiPlug architecture and rapid deployment cycles
- Where to start on your own apps
- What’s next

- Raffle: Save your stickers until the end of the talk, we’ll be giving away 15 starter kits for Pi development! (5 more at our booth later)
Pi Meets Eclipse

Raspberry Pi®

- Platform for tinkering
- Exemplifies opportunities of the “Internet of Things” (IoT)
- Based on ARM architecture
- Lower-powered but flexible

+ eclipse

- Rich foundation of components for devices to servers
- Flexible deployment models
- Simple extensibility
- Vibrant community

= An ideal pairing of technologies
Introducing PiPlug

- PiPlug brings the ease and extensibility of Eclipse to the Raspberry Pi
- Runs using a simple RCP front-end with app plug-ins
- Optimized for low overhead and responsive UI
- Remote deployment of apps from Eclipse to Pi

* Daemon can be automatically self-hosted inside Eclipse
What’s Possible?

- What works well with PiPlug
  - Apps using standard SWT and JFace
  - Apps using Java bindings to Pi services and add-on modules
  - Apps using custom canvases for dynamic rendering
  - Apps using remote network resources (off UI thread!)
- What’s not intended for the PiPlug
  - Non-UI services (though not strictly prohibited)
  - Very heavy weight apps that won’t perform well on the Pi
    (example, full Eclipse IDE takes ~10 minutes to start up!)
Seeing PiPlug in Action

Running your first application
App Plug-ins in PiPlug

• Each application...
  – is in an OSGi bundle (plug-in)
  – implements a PiPlug API extension point
  – defines a lifecycle class
  – contributes UI controls for the application display

• Additional service bundles can be deployed
  – Provides services outside of core framework

• Already available to PiPlug apps:
  – JFace, SWT, Jobs, Registry, and other base Eclipse bundles
public class YourFirstApp implements IPiPlugApplication {

    public void installed(IPiPlugServices services) {
        // called upon installation of the plug-in
    }

    public Composite prepare(IPiPlugServices services, Composite parent) {
        // called to initialize the UI on first access or upgrade
    }

    public void resume(IPiPlugServices services) {
        // called as app will be placed in the foreground
    }

    public void suspend(IPiPlugServices services) {
        // called when app is no longer in the foreground
    }

    public void shutdown(IPiPlugServices services) {
        // called to allow cleanup for upgrade or shutdown
    }

}
public void installed(IPiPlugServices services) { ... }

• Prepares services needed for the bundle off UI thread
• Called once per invocation of VM or when upgraded to a new app version
• Should wait until prepare/resume is called for anything heavy

@Override
public void installed(IPiPlugServices services) {
    // nothing needed at installation time
}
public Composite prepare(IPiPlugServices services, Composite parent) { … }

• Initializes the user interface controls for the application
• Called once per invocation of VM or when upgraded to a new app version

@Override
public Composite prepare(IPiPlugServices services, Composite parentStack) {
    composite = new ClockComposite(services, parentStack);
    return composite;
}
public void resume(IPiPlugServices services) { … }  

- Populates the UI with current information  
- Starts background workers or services  
- Called each time application will be in the foreground  
- Resumes previous operations that were otherwise suspended
suspend(...)  

public void suspend(IPiPlugServices services) { ... }  

• Saves off any application state  
• Suspends background jobs or threads  
• Called once the application is in the background  
• Should stop any CPU utilization from the application

```java
@Override
public void suspend(IPiPlugServices services) {
    if (null != job) {
        job.stop();
        job = null;
    }
}
```
public void shutdown(IPiPlugServices services) { … }

- Frees handles to classes in the runtime
  - Make sure to null out references, and clean up services
- Called just before PiPlug front-end shuts down or bundle is uninstalled before upgrade

@Override
public void shutdown(IPiPlugServices services) {
    // nothing needed during shutdown
}
Guidelines for Apps

- Optimize for responsive UI
  - Minimize UI created in prepare(…)
  - Depend on SWT, or at most JFace
  - Minimize # of dependent bundles
- Build for OSGi bundle reloading
  - Cleanly dispose and cleanup services in shutdown(…)
  - Cleanly track Images and other heavy resources
  - If using background threads, respond to shutdown & suspend
Demonstration

Seeing PiPlug in Action
Working with your applications
Performance on the Pi

• Starting Java is slow
  – Java 8 ARM is better!
  – Stay in the same VM

• Your app runs in a shared JVM/OSGi runtime
  – Startup is ~10 seconds
  – You don’t pay for your own JVM/OSGi/SWT Display
  – Follow good UI responsiveness rules (instant feedback)

• Overclocking makes a difference (are you a hardware guy?)
  – A good power supply matters, as do heat sinks!
Architecture of PiPlug

• Daemon manages your collection of bundles
  – Automatic UDP discovery on your local network
  – Supports multiple developers & PiPlugs

• PiPlug UI is intelligent
  – Automatically finds your daemon
  – Automatically adds/removes applications as they’re deployed

• PiPlug Deploy View
  – Detects or automatically starts a daemon
  – Easily deploy or undeploy applications using PDE export
Idiosyncrasies of RCP on Pi

• No SWT fragment
  – ARM LE is not officially supported
  – This makes building products difficult
  – PDE export works if you put in your arch as arm

• Xlib: RANDR missing on display :1.0
  – Can be ignored

• Maven build woes
  – Would love to use tycho but... no SWT fragment monkey-wrenches the build
Demonstration

Seeing PiPlug in Action

Rapid development with PiPlug
Time to Start Plugging!

• It’s all on Github!
  – http://genuitec.github.io/piplug

• In your Eclipse
  – Install the PiPlug Deployment View
  – Create your app or copy an example

• On your Pi
  – Install the PiPlug Frontend (& dependencies)
  – Run it!
Back to the IoT

- Lifecycle of software rollout
- How to get IoT devices out in volume with applications
- How to manage many IoT devices
- How to handle versions of applications
- What security constraints apply to the IoT

- We’ve done this for Eclipse & Eclipse Plugins with Secure Delivery Center but what’s right for the IoT?
Raffle Time!

Seeing PiPlug in Action

Let’s choose our raffle winners!
Thank You!

- Get started with PiPlug at http://genuitec.github.io/piplug/
- Come see us at our booth for another chance to win a Pi, to collect your winnings, or to chat us up about cool things you’re doing with your Pi!