Contents

1 Exosense Demo Application Tutorial

2 Documentaion Locations

1. **Building flashable Exosense Device Demo application image** This document.

2. **Building Exosense Device Demo application for desktop use** Please see the README.md file in `https://github.com/Feuerlabs/exodemo` Also check the following chapters in this document for information on getting the demo application up and running

- Setting Device Identity
- Adding Device to Exosense Server
- Sending RPC to Device

3. **Exosense Server Usage** Please see the README.md file at `https://github.com/Feuerlabs/exosense_specs` This repository contains an Exosnese Server User Manual, a JSON-RPC Reference Manual, and sample shell scripts using curl(1) to interface the server.

3 Introduction

Note: An Exosense Server account is needed for the device running the Exosense demo application to communicate with the server. Likewise, a user/password pair, tied to the account, must be setup in order to send JSON-RPC commands to the server.

Send an email to accounts at feuerlabs in order to get a test account setup for free.

This repository contains the Feuerlabs' build system that generates images with erlang and the Exosense Device stack for various hardware targets.

If you want to run the Exosense Device stack on your local box, without having to generate a complete image, please look at the exodemo repo at

`https://github.com/Feuerlabs/exodemo`

The exodemo package is built as a part of the image generated by these instructions, but can also execute stand-alone in an existing Linux environment.

The build system uses yocto (http://www.yoctoproject.org), and provides the following additions:

1. **Erlang** R15B1 is compiled for the given target.
2. **Exosense Device** All Exosense Device components are compiled for the target.
3. **Reference Hardware Support** The reference hardware supported by Feuerlabs for the Exosense Device stack is integrated into the exosense repo. Currently, this is limited to the SBC6845.

4 Further documentation

Yocto quick start

`http://www.yoctoproject.org/docs/1.3/yocto-project-qs/yocto-project-qs.html`

Yocto reference manual

`http://www.yoctoproject.org/docs/current/poky-ref-manual/poky-ref-manual.html`

Yocto developer's guide

`http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html`

5 Directory structure overview

There are five directories that will be referenced throughout the build. These directories can be placed anywhere in the tree, even inside each other. However, since directories are each maintained by their own repositories, it is recommended that they are kept in the same parent directory.

1. **Yocto [$YOCTO]** This is the stock Yocto (poky-danny-8.0) distribution that can generate a Linux distro image for a a wide vaiety of target hardware. It is downloaded from the Yocto project site and will not change during the build process.
2. **Exosense Device [$EXOSENSE]** This is the extensions to Yocto provided by Feurelabs. Included in these build instructions is an erlang build as well as rebar and tetrapak integration, all sourced from Travelping's TPLINO distribution. Also included is the build instructions for all Exosense Device components. The directory is checked out from the Feuerlabs github repo and will not change during the build process.
3. **SBC6845 support [$SBC6845]** ***OPTIONAL*** This directory adds support for the SBC6845 to Yocto through as BSP layer. It is checked out from the Feuerlabs github repo and will not change during the build process.
4. **Exosense Device Application Build [$DEMOBUILD]** Contains build instructions for the application running on top of the Exosense Device stack. The instructions are usually copied from the Exosense Device demo application in the Feuerlabs github repo and are then maintained by the developer. Please note that this directory does not contain the application code itself, which is downloaded from a git/svn/http/whatever repo by the Yocto build instructions in `$DEMOBUILD`
5. **Build [$BUILD]** The directory where the build process is executed. Will swell 20+ GB during the process. The build directory is created through a yocto initialization command. Once created, the template configuration files in the build directories are updated to reference `$YOCTO`, `$EXOSENSE`, `$DEMOBUILD`, and possibly `$SBC6845`. The configuration is also updated to reflect the target hardware and which features (ssh, graphics, etc) to include in the generated image.

In the subsequent instructions, the following directory structure will be used as an example

```
/home/bob/poky-danny-8.0   # The $YOCTO directory
/home/bob/meta-exosense    # The $EXOSENSE directory
/home/bob/meta-exodemo     # The $DEMOBUILD direcotry
/home/bob/meta-sbc6845     # The $SBC6845 direcotry
/home/bob/build            # The $BUILD direcotry
```

6 Building the Exosense Device demo application (exodemo)

6.1 Preparation of build system

An Ubuntu 12.04 / 12.10 should be used as a host system. Install the necessary support packages using the following commands.

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo build-essential chrpath li
```

Please check the Yocto quick start guide for further details if another host environment is desired.

6.2 Download and install the latest Yocto release

1. **Download Yocto 1.3** The URL for this Yocto release is cd /tmp wget http://downloads.yoctoproject.org/releases/yocto/yocto-1.3/poky-danny-8.0.tar.bz2 This will download the yocto release to /tmp.

2. **Unpack the Yocto release into `$YOCTO`** The downloaded file can be unpacked using the following commands:

   ```
   cd /home/bob
   tar xf /tmp/poky-danny-8.0.tar.bz2
   ```

This will create the Yocto directory structure under `/home/bob/poky-danny-8.0`.

6.3 Download and install the latest Exosense Device release

1. **Download Exosense 1.0** Clone the git repository into `$EXOSENSE` with:

   ```
   cd /home/bob
   git clone --bare -b 1.0 https://github.com/Feuerlabs/meta-exosense.git
   ```

This will create the Yocto directory structure under `/home/bob/meta-exosense`.

6.4 Setting up an application build system

1. **Download the Exosense Device demo application build layer**
   Checkout the template application with into `$DEMOBUILD` with:

   ```
   git clone --bare https://github.com/Feuerlabs/meta-exodemo.git
   ```

The git command creates a Yocto layer under `/home/bob/meta-exodemo` (`$DEMOBUILD`) containing recipes (build instructions) used to compile and package the Exosense Device Application.

Please note that this directory only contains Yocto build instructions ; the actual application code itself is managed by its own repository that is referenced by the recipes.

6.5 Generating a build directory

The build directory, under which the entire build process takes place, is created through a yocto script. Please see the Yocto quick start guide for more information.

Generate the build directory, `$BUILD`, using the following commands:

```
cd $YOCTO
. oe-init-build-env $BUILD
```

This will create the `/home/bob/build/conf` (`$BUILD`) directory containing two files: `bblayers.conf` and `local.conf`. Once this script has executed, all builds should be executed from the build directory.

If git, svn, p4, or other version control system is used, you should add the `$BUILD` directory and its content to it at this point. Once the build has started $BUILD will get populated with additional directories and files that should not be version managed.

6.6 Binding $YOCTO$,EXOSENSE, $SBC6845and$DEMOBUILD together

All directories to use during the build are specified by `$BUILD/conf/bblayers.conf` The default file, generated by the `oe-init-build-env` script, looks like this:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"
```

```
BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/bob/poky-danny-8.0/meta \
  /home/bob/poky-danny-8.0/meta-yocto \
  /home/bob/poky-danny-8.0/meta-yocto-bsp \
"
```

The `$EXOSENSE`, `$DEMOBUILD` and `$SBC6845` directories should be added to the BBLAYERS entry in bblayers.conf, yielding the following file:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/bob/poky-danny-8.0/meta \
  /home/bob/poky-danny-8.0/meta-yocto \
  /home/bob/poky-danny-8.0/meta-yocto-bsp \
  /home/bob/meta-exosense \
  /home/bob/meta-sbc6845 \
  /home/bob/meta-exodemo \
"
```

If the SBC6845 is not to be used by the app, the `/home/bob/meta-sbc6845 \` line can be omitted.

6.7 Configuring the target image features

The setup of the target image, to be flashed to the device, is specified by `$BUILD/conf/local.conf`.

The default file, generated by the `oe-init-build-env` script, contains several entries. Please see the comments of the file for details.

6.7.1 BB_NUMBER_THREADS

This entry specifies how many threads to be used during a build. Uncomment this and set this to the number of cores that the host system has:

```
BB_NUMBER_THREADS = "8"
```

6.7.2 PARALLEL_MAKE

This entry is provided as an argument to make to specify the number of parallel compiles to run. Uncomment and set to the number of cores that the hos system has.

```
PARALLEL_MAKE = "-j 8"
```

6.7.3 MACHINE

This entry specifies the target machine (and its architecture) to generate the image for. See the Yocto Developer's Guide for more information on this. Set this entry to the target machine:

```
MACHINE ??= "sbc6845"
```

6.7.4 IMAGE_INSTALL_append

This entry specifies additional packages (generated from recipes), to include in the image. No default is created in the `local.conf` file. Add the following line anywhere in `local.conf` to include the Exosense Device demo application built by the recipes located under `$DEMOBUILD`.

```
IMAGE_INSTALL_append = " erlang-exodemo exosense exodemo-boot exodemo-app"
```

The exodemo-boot, found in `meta-exodemo/recipes-exodemo/exodemo/exodemo-boot.bb` sets up boot scripts to be executed at system start.

The exodemo-ppp, found in `meta-exodemo/recipes-exodemo/exodemo/exodemo-ppp.bb` sets up ppp options and call scripts for mobile PPP links. Please note that these ppp scripts will have to be adapted to local carriers. All files are available under `meta-exodemo/recipes-exodemo/exodemo`.

6.7.5 EXTRA_IMAGE_FEAUTURES

The image built by the `bitbake core-image-minimal` command below is a barebone distribution.

To add ssh-access, add `ssh-dropbear-server` to the list specified by `EXTRA_IMAGE_FEATURES`.

To add automatic start of the demo application when the device is powered up, add `exodemo-boot` to `EXTRA_IMAGE_FEATURES`.

To add automatic GPRS/3G/4G connectivity, add `exodemo-ppp` to `EXTRA_IMAGE_FEATURES`.

An example of a fully populated `EXTRA_IMAGE_FEATURES` would be:

```
EXTRA_IMAGE_FEATURES = "debug-tweaks ssh-server-dropbear exodemo-boot exodemo-ppp"
```

Please note that the ppp scripts most likely will require customization prior to use.

See the "Device Boot Sequence" chapter for details on how automatic startup of the exosense demo application is done.

See the "Device PPP setup" chapter for details on how PPP is setup.

6.8 Build the image

Ensure that the oe-build-env script has been executed

```
cd $YOCTO
. oe-init-build-env $BUILD
```

Since the `$BUILD` directory already exists, the script will only setup the environment and `cd` to it.

An image is generated using

```
bitbake core-image-minimal
```

See the Yocto quick start guide for a primer on the various images available. Please note that `$BUILD/conf/local.conf` can be updated to add features and packages.

7 Image Flash Instructions for the SBC6845

Setup a serial connection to the SBC6845 debug DB9 port. 115200 8N1

Power on the SBC6845.

Interrupt the boot process by pressing space immediately after power on

Connect an ethernet cable and setup the device IP address and boot params

```
setenv ipaddr 192.168.10.55      # Device IP
setenv netmask 255.255.255.0
setenv serverip 192.168.10.200  # TFTP server ip
setenv bootargs console=ttySAC6,115200 ubi.mtd=2 root=ubi0:sbc6845-rootfs rootfstype=ubifs
setenv bootcmd 'nand read.i 0x72000000 0x00000000 0x260000; bootm'
saveenv
```

Note: The third bootcmd read size parameter, 0x2600000, may need to be adjusted upward to be equal to or greater than the loaded uImage file size. See the result of the tftp command used to retrieve uImage to see the minimum read size parameter value.

Note: Please see "Setting Device Identity" for instructions on how to specify exosense server address, device ID, and device keys in the Linux boot params.

Wipe the entire nand area

```
nand erase
```

Load and flash the ubi volume image with the rootfs ubifs

```
tftp 0x70000000 core-image-minimal-sbc6845.ubi
nand write.i 0x70000000 0xba0000 $(filesize)
```

Load the boot uImage

```
tftp 0x72000000 uImage
nand write.i 0x72000000 0x0 $(filesize)
```

8 Setting Device Identity

The device identity, consisting of a device id, an exosense server address, and device/server keys, are needed in order for the device to act as a unique unit in a network. The device identity needs to be setup both in the device and in the server before the device can connect.

The chapter "Adding Device to Exosense Server" describes how to setup the device in the server.

8.1 Setting the device identity through Linux boot parameters

By setting the boot parameters passed to the Linux kernel by the boot loader (u-boot, grub, lilo, etc), the device ID can be set outside the Linux environment. This allows reflashing of the entire system (over the air), withouth having to generate a custom image for each device.

In a mass production environment, the boot parameter option also allows for a simpler setup since there will be no need to boot each flashed unit in order to customize it.

The following boot parameters are detected by the Exosense Device stack

1. **exo_host** Specifies the fully qualified domain name (if DNS is availble) or IP address (if DNS is not available) of the Exosense Server that the device should use when it has a connection.
2. **exo_account** Specifies the account in the Exosense Server that the device shall authenticate itself with. The account must have been created in the server with the create-account JSON-RPC command. **Note:** The account is different from the user, specified in `~/.exodmrc`, used to authorize JSON-RPC commands sent to the Exosense Server. When an an account was
3. **exo_device_id** Specifies the device-id string to use when authenticating the device toward the Exosense Server. The device must have been provisioned in the server with the create-device JSON-RPC command.
4. **exo_device_key** Specifies the device key that the device will use to authenticate itself with toward the server. Please note that this is a 64 bit key used only for authentication, not encryption. If a stronger security is needed, OpenSSL is recommended.
5. **exo_server_key** Specifies the server key that the server must use to authenticate itself with toward the device. Please note that this is a 64 bit key used only for authentication, not encryption. If a stronger security is needed, OpenSSL is recommended.n

An example of how to setup the sbc6845 boot parameters is given below, with artificial line breaks.

```
setenv bootargs console=ttySAC6,115200 ubi.mtd=2 \
    root=ubi0:sbc6845-rootfs rootfstype=ubifs \
    exo_host=test.feuberlabs.com exo_account=my_account exo_device_id=tst-1 \
    exo_server_key=1234567890 exo_device_key=0987654321
```

When Linux boots, all boot parameters not recognized by the kernel will be passed on to init as environment variables. Init will launch the exodev_ctl script, which will start the exodemo Erlang application. The application will, through the exoport app, examine its environment variables and use those above.m

The exo_ boot parameters (but not their values) are case insensitive and can be provided in either upper or lower case.

8.2 Setting the device identity through environment variables.

Since the Exosense Stack uses environment variables to set device identity, these variables can be set and exported by the shell (or other process) that launches Exosense.

The variable names are the same as specified by the boot parameter chapter. The variable names (but not their values) are case insensitive and can be provided in either upper or lower case.

8.3 Setting the device identity through a configuration file.

The exoport configuration file, found in `exodemo/priv/exoport.config`, can contain the device identity variables. Please note that the exo_ prefix is to be removed from the names. An example of a configuration file is given below.

```
{ host, "test.feuerlabs.com }.
{ account, "my_account" }.
{ 'device_id', "demo-1" }.
{ 'server_key', 1234567890 }.
{ 'device_key', 0987654321 }.
```

Please note that any values found in the exoport.config file will override values specified by boot parameters and environment variable.

When the image is generated by Yocto, the configuration file will be placed on the target system at `/usr/lib/erlang/lib/exoport*/priv/exoport.config`.

When the `exodev_ctl` script executes the first time during boot, it will use `setup_gen` to collect all config files into a single directory that is used as a configuration directory by the exosense stack. Please see the Exosense Device Boot Sequence for details.

9 Device Boot Sequence

If the `EXTRA_IMAGE_FEAUTRES` list in `${BUILD}/conf/local.conf` has the `exodemo-ppp` entry in it, Yocto will setup a control script in the target as `/etc/init.d/exodev_ctl`. A symlink will be created from this file to `/etc/rc5.d/S90exodev_ctl`.

All these actions are carried out by the recipe in `${DEMOBUILD}/recipes-exodemo/exodemo/exodemo-boot.bb`. The same directory also contain the source `S90exodev_ctl` script that will be copied to the target.

The `S90exodev_ctl` script, automatically executed by /sbin/init as a part of the boot process, will go through the following three steps:

1. **Start ppp (if applicable)** If ppp is installed, by providing the `exodemo-ppp` option to the `EXTRA_IMAGE_FEATURE` list in `${BUILD}/conf/local.conf`, it will be started to setup a mobile link to the Internet. Please see the "Device PPP setup" chapter for details on how PPP is setup.

2. **Create an erlang environment** The first time the image is booted, an exosense root directory is created ($ROOT). In this directory all configuration files are collected by `setup_gen` into a `setup` directory under the root. Please see the following repository for details on this process:

   https://github.com/esl/setup

3. **Launch erlang** If ppp is used to communicate, 15 second sleep is done to give the ppp link time to come up. After that erlang is started with the boot scripts created by `setup_gen` under `$ROOT/setup/start`, with the collected configuration under `$ROOT/setup/sys`. All output from the process are redirected to `/tmp/exodev.log`.

10 Device PPP setup

The ppp files are installed in the target imagea by the recipe in `${DEMOBUILD}/recipes-exodemo/exodemo/exodemo-boot.bb`. This directory also contains the following files that are copied to `/etc/ppp` on the target system.

1. **ppp-start** Simple script that launches pppd with the `gprs-options` configuration file.
2. **gprs-connect** Chat script to connect the modem to the mobile carrier and its APN.
3. **gprs-disconnect** Chat script to disconnect an established mobile session from the carrier.
4. **pap-secrets** A default file with pap authentication passwords.
5. **chap-secrets** A default file with chap authentication passwords.
6. **connect-errors** Not used

The `ppp-start` script is launched by the `exodev_ctl` script. See the "Device Boot Sequence" for details.

See the `pppd(8)` manpage for details on how to setup ppp links in Linux.

11 Adding Device to Exosense Server through JSON-RPC

All communication with the Exosense Server is conducted through its JSON-RPC interface.

In the subsequent chapters, shell scripts will be used for convenience.

The shell scripts are located at:

```
https://github.com/Feuerlabs/exosense_specs
```

Please see the "Exosense Server Usage" item at the top of this document for references to documentation that describes the exact mechanics of each step of adding a device to the Exosense Server.

11.1 Setting up the .exodmrc file to access Exosense Server

In order for the shell scripts to operate, they must know the URL where the Exosense Server resides, and how to authenticate to it. This is done through the ~/.exodmrc file that looks like the following

```
URL=https://test.feuerlabs.com:8000/exodm/rpc
USER_AUTH=user:password
```

The `URL` in the example above points to the Feuerlabs test server provided for free for up to ten devices. Please email us at accounts at Feuerlabs to get an account setup.

The USER_AUTH contains the user name and password assigned to you by Feuerlabs.

11.2 Uploading RPC Yang Specification to Exosense Server

The yang specification for the exodemo application is located in the yang directory of this repository:

```
https://github.com/Feuerlabs/exodemo
```

Use the `create-yang-module.sh` script to upload the `exodemo.yang` file to the Exosense Server:

```
sh create-yang-module.sh user ~/work/exosense_demo/exodemo/yang/exodemo.yang
```

Replace the `~/work/exosense_demo/exodemo` path with the path to the checked out exodemo repository.

The server will send the following reply to indicate success.

```
{"result":{"result":"ok"},"id":"1","jsonrpc":"2.0"}
```

11.3 Adding a device type to the Exosense Server

A new device type needs to be created for the demo device. Since the exodemo application uses the default bert-rpc protocol (and not a plugin for a custom protocol), the following command can be used:

```
sh create-device-type.sh demotype exodm_bert
```

The server will send the following reply to indicate success.

```
{"result":{"result":"ok"},"id":"1","jsonrpc":"2.0"}
```

11.4 Adding a configuration set to the Exosense Server

A configuration set needs to be created in order to associate the uploaded Yang specification with the device:

```
sh create-config-set.sh demoset exodemo.yang http://myurl.com:1234
```

- `demoset` The name of the configuration set.
- `exodemo.yang` The yang specification previously uploaded to the Exosense Server.
- `http://myurl.com:1234` The address to send notifications and incoming RPCs to when they are received from a device.

Note: Configuration sets will be deprecated in favor of packages in Exosense 2.0. Please see the Exosense Server Manual (reference at top of this document) for more information.

The server will send the following reply to indicate success.

```
{"result":{"result":"ok"},"id":"1","jsonrpc":"2.0"}
```

11.5 Adding a device to the Exosense Server

Create the device with the create-device.sh script:

```
sh create-device.sh demo-1 demotype demoset 1234567890 0987654321
```

- `demo-1` The device ID, which must be unique for the user.
- `demotype` The device type, as specified by the create-device-type.sh script above.
- `demoset` The configuration set, as specified by the create-config-set.sh script above.
- `1234567890` The server key that the Exosense Server will use to authenticate itself toward the device.
- `0987654321` The device key that the demo app will use to authenticate itself toward the Exosense Server.

Please note that the device ID, server key, and device key must match those setup in the demo application in the "Setting Device Identity" chapter.

The server will send the following reply to indicate success.

```
{"result":{"result":"ok"},"id":"1","jsonrpc":"2.0"}
{"result":{"result":"ok"},"id":"1","jsonrpc":"2.0"}
```

12 Invoking Device RPC through Exosense Server JSON-RPC commands

12.1 Device running in desktop environment

The exosense demo application is located in the exodemo repository at:

`https://github.com/Feuerlabs/exodemo`

Instructions for building and running the application in desktop linux environment are provided in the README file of the exodemo repository. Please ensure that the `local_start_demo.sh` script has its `exo_xxx` environment variables updated to reflect the device id given in the "Setting Device Identity" chapter.

12.2 Device running on target hardware

If a target device (sbc6845 or similar) has been flashed with a Yocto build, as described in the "Building the Exosense Device demo application" chapter, then powering on the device with the new image should automatically have it connect to the server.

Check that the `exo_xxx` Linux kernel boot parameters are matching their corresponding values setup with the create-device.sh script.

See the "Setting the device identity through Linux boot parameters" Chapter for details.

12.3 Creating shell script sending JSON-RPC to Exosense Server

The exodemo.yang specification (provided by the exodemo repo), contains a single command that can be sent from the server to the device. The yang fragment looks as follows:

```
rpc beep {
  description
    "Sent from Server to Device. Beep the on-board buzzer, " +
    "turn on the LED, or a similar visible operation.";

  input {
    uses exo:std-request;

    leaf duration {
```

```
      description "Number of milliseconds to run the buzzer for.";

      type uint32;
    }
  }
  output {
    description "Output sent in response to a beep request";
    uses exo:std-callback;
  }
}
```

A valid JSON-RPC script sent to the server would be:

```
{
  "jsonrpc": "2.0",
  "method": "exodemo:beep",
  "id": "1",
  "params":
  {
    "device-id": "demo-1",
    "duration": 500
  }
}
```

- `exodemo:beep` The method to be invoked on the device, where `exodemo` is the namespace of the spec, and `beep` is the actual method.
- `device-id` Specifies the target device, as previously created with the create-device.sh scripot.
- `duration` The single argument provided to the `beep` method on the device.

Finally, this method can be integrated into a shell script that uses curl:

```
#!/bin/sh
. ~/.exodmrc
curl -u $USER_AUTH -k -X POST $URL -d @- << EOF
{
  "jsonrpc": "2.0",
  "method": "exodemo:beep",
  "id": "1",
  "params": {
    "device-id": "$1",
    "duration": $2
  }
}
EOF
```

Invoking this shell script will parse the ~/.exodmrc file to retrieve $USER_AUTHand$URL (the Exosense Server address), and then invoke curl to send the command to the device id given in the first command line argument, with `dutaion` set to the second argument.

See the "Setting up the .exodmrc file" chapter for details on $USER_AUTHand$URL.

12.4 Invoking the shell script

When the shell script above is invoked, it will send the JSON-RPC command to the Exosense Server.

```
sh beep.sh demo-1 1234
```

The server will respond with the following receipt.

```
{
  "result": {
    "transaction-id":"411",
    "rpc-status":"accepted",
    "rpc-status-string":"Operation has been accepted and is in progress.",
    "final":false
  },
  "id":"1",
  "jsonrpc":"2.0"
}
```

The demo application will, when it receives the call, emit a message from its `exodemo_rpc:beep()` method:

```
beep([{'device-id',<<"demo-1">>},{duration,1234}])
```

13 Changing the RPC interface to the demo application.

If the `beep` method is to return a value to the server, to be forwarded to the URL associated with configuration sets that the device is a member of, a notification must be sent from the device to the server.

The following chapters shows how to implement such a notification.

13.1 Cloning the repos

Since only Feuerlabs has push access to the demo code at `https://github.com/Feuerlabs/exodemo` and the Yocto build layer at `https://github.com/Feuerlabs/meta-exodemo`, these two repos should be cloned by the developer prior to modifying the code.

See github documentation for detail on how to clone repos.

Once cloned, check out meta-exodemo repo and modify `meta-exodemo/recipe/exodemo/erlang-exodemo.bb` so that its URL refers to the cloned `exodemo` repo as well.

13.2 Modifying the Yang specification

The notification has the following format in Yang:

```
notification beep-notification {
  description "Notificaiton sent back in response to beep RPC.";

  leaf extra {
    description "Extra information";
    type string;
  }
  uses exo:std-callback;
}
```

Insert it anywhere in the `exodemo/yang/exodemo.yang` file.

13.3 Validating the edited Yang specification

A changed yang specification should be validated prior to being uploaded to the server. The pyang validator is the standard tool to accomplish this:

`http://code.google.com/p/pyang/`

Since the `exodemo.yang` specification in the `exodemo` repository references the core `exosense.yang` specification, this file should be checked out from the exosense_specs repository.

```
https://github.com/Feuerlabs/exosense_specs
```

Run the pyang validator (after it has been intalled) as follows:

```
pyang -p ~/work/exosense_specs/yang/ exodemo.yang
```

Replace the `~/work/exosense_specs/yang` path with the path to the checked out exosense_specs repo.

If pyang provides no output, the specification is valid and can be uploaded to the Exosense Server.

13.4 Uploading the modified Yang specification

Use the `create-yang-module.sh` script to upload the `exodemo.yang` file to the Exosense Server:

```
sh create-yang-module.sh user ~/work/exosense_demo/exodemo/yang/exodemo.yang
```

Replace the `~/work/exosense_demo/exodemo/yang/exodemo.yang` path with the path to the modified specification.

The new specification will replace the old version in the Exosense Server.

13.5 Updating the Exosense demo application

All code for the Exosense demo application is in the `exodemo/src`.

Notifications are sent back from the device by replacing the `ok` return value from `exodemo_rpc:beep` with a `{ notification, ...}` tuple.

The original beep code is as follows:

```
'beep'(Duration) ->
  io:format("beep(~p)~n", [Duration]),
  ok.
```

To send back a notification, use this code instead:

```
'beep'(Duration) ->
  io:format("beep(~p)~n", [Duration]),
  { notify, "exodemo:beep-notification",
   [ {'rpc-status', 1 },
     {'final', true },
     {'extra', "Beep! Beep!" } ] }.
```

- `rpc-status` [mandatory] Has a status value taken from the `status-code` enum in `exosense.yang`, which can be found in the `https://github.com/Feuerlabs/exosense_specs` repository.
- `final` [mandatory] Can be `true` or `false` and indicates if more notifications are to follow for this transaction.
- `extra` Specifies the string value of the `extra` argument to the `beep-notification` call added to the `exosense.yang` file.

13.6 Rebuilding the Exosense demo application on desktop

Please see the `https://github.com/Feuerlabs/exodemo` repository and its README.md file for instructions on how to recompile and launch the Exosense demo application in a desktop environment.

13.7 Installing recompiled Erlang beam files on Device

If only erlang beam files have been changed, they can be copied out to the target hardware through scp:

```
scp exodemo/ebin/exodemo_rpc.beam \
    192.168.0.10:/usr/lib/erlang/lib/exodemo-*/ebin/
```

Replace the IP address with the address of the device.

The beam files are architecture independent and can be freely transported between machines of different endianess, word length, etc.

13.8 Rebuilding the flashable Exosense demo image

13.8.1 Committing the changes to the repo

Before a new image can be built, the changes to the exodemo code must be commited and pushed to the repository:

```
git commit -m "Added notification" -a
git push
```

13.8.2 Cleaning out the yocto recipe

In order for yocto to pull the changed code from the repo, it must first be cleaned out.

```
cd $YOCTO
. oe-init-build-env $BUILD
bitbake -c cleanall erlang-exodemo
```

Any other code modifications/pushes should be cleaned by bitbake as well .

13.8.3 Rebuilding the image

Please follow the build instructions provided in “Build the image” chapter.

13.8.4 Flashing the image

Please follow the f lash instructions provided in “Image Flash Instructions for the SBC6845”.