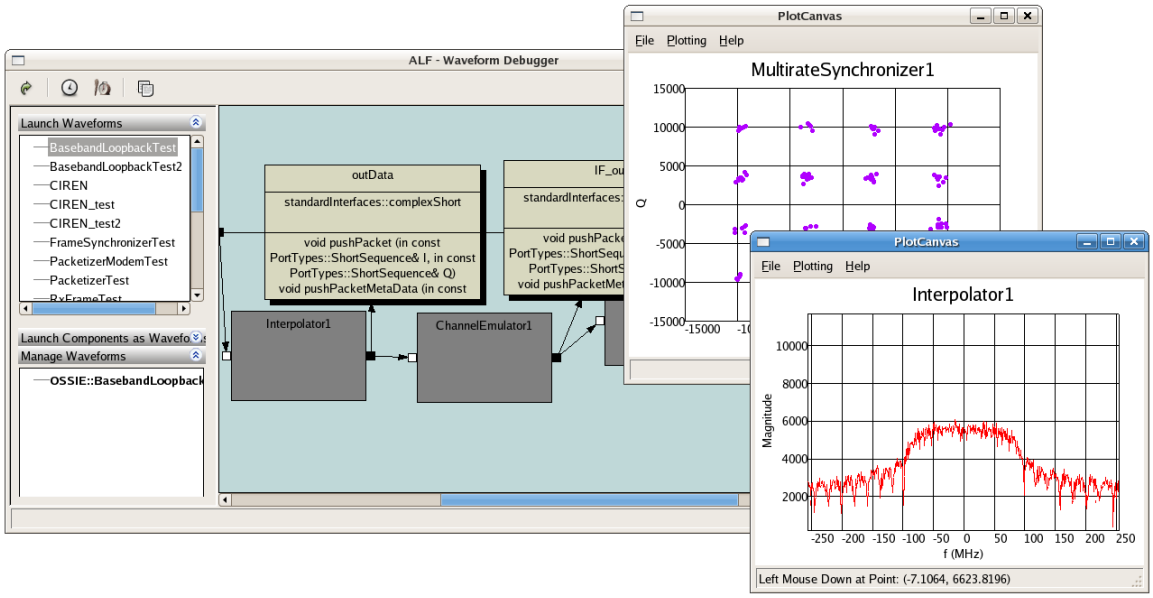
OSSIE 0.7.4 Installation and User Guide

Matt Carrick, Drew Cormier, Christopher Covington, Carl B. Dietrich,
Joseph Gaeddert, Benjamin Hilburn, C. Ian Phelps, Shereef Sayed,
Deepan Seeralan, Jason Snyder, Haris Volos

July, 2009



OSSIE is an open source Software Defined Radio (SDR) development effort based at Virginia Tech. OSSIE is primarily intended to enable research and education in SDR and wireless communications. The software package includes an SDR core framework based on the JTRS Software Communications Architecture, tools for rapid development of SDR components and waveforms (applications), and an evolving library of pre-built components and waveforms. In addition, free laboratory exercises for SDR education and training are being developed in cooperation with the Naval Postgraduate School.

OSSIE is the Open Source SCA Implementation::Embedded

Contents

1 Quick Start Guide **1**

1.1 Using the OSSIE VMware Player Image 1

1.2 Running the OSSIE Demonstration Waveform 1

2 Introduction **5**

2.1 About this Guide 5

2.2 Further Help 5

2.2.1 Wiki 5

2.2.2 Mailing Lists 5

2.2.3 Trac Bug Tracking System 6

2.2.4 The #ossie IRC Channel 6

3 Installation **7**

3.1 Installing OSSIE from Source 7

3.1.1 Notes on OSSIE Installation from Source 7

3.1.2 Installing Dependencies on Fedora 7

3.1.3 Installing Dependencies on Ubuntu 8

3.1.4 Configure omniORB 8

3.1.5 Installing Portions of GNU Radio 9

3.1.6 Install OSSIE 11

3.1.7 Using Installation Scripts 11

3.1.8 Updating System Libraries 12

3.2 Installation of OSSIE Eclipse Feature 12

3.2.1 Installing Java 12

3.2.2 Installing Java on Older Versions of Fedora 13

3.2.3 Installing Java on Ubuntu 13

3.2.4 Installing Eclipse 13

3.2.5 Installing OEF 13

3.3 Using a VMware Image on Any Platform . . . 14

4 Running Waveforms **15**

4.1 Starting the CORBA Naming Service . . . 15

4.2 Running `nodeBooter` . . . 15

4.3 Nodebooter Clean-Up . . . 15

4.4 Loading a Waveform . . . 16

5 Waveform Workshop **18**

5.1 OSSIE Eclipse Feature . . . 18

5.2 OSSIE Waveform Developer . . . 18

5.3 ALF Graphical Debugging . . . 19

5.4 WaveDash . . . 20

6 OSSIE Eclipse Feature **21**

6.1 Creating a New Waveform from Existing Components . . . 21

6.1.1 Adding Components to the Waveform . . . 23

6.1.2 Connecting Components . . . 24

6.1.3 Setting the Assembly Controller . . . 24

6.1.4 Editing Component Properties . . . 24

6.1.5 Deploying Components to a Node . . . 25

6.2 Creating a New Component . . . 25

6.2.1 Adding Ports to the Component . . . 27

6.2.2 Adding Properties to the Component . . . 29

6.2.3 Generating the Source Code . . . 30

6.2.4 Editing C++ Components . . . 31

6.2.5 Editing Python Components . . . 31

6.2.6 Editing the SPD File . . . 32

6.2.7 Making Sure Files are Executable . . . 32

6.2.8 Installing a Component . . . 32

6.3 Importing and Exporting Eclipse Projects . . . 33
6.3.1 Exporting a Project From Eclipse . . . 33
6.3.2 Importing Project in Eclipse . . . 34
6.4 Additional OEF Instruction . . . 36

7 OSSIE Waveform Developer **37**
7.1 Creating a New Waveform from Existing Components . . . 37
7.1.1 Adding an Existing Node to a Waveform . . . 37
7.1.2 Adding an Existing Component to a Waveform . . . 38
7.1.3 Connecting Components . . . 40
7.1.4 Deploying Components and Editing Component Properties . . . 41
7.1.5 Setting the Assembly Controller . . . 43
7.1.6 Generating the Waveform . . . 44
7.1.7 Installing the Waveform . . . 45
7.2 Creating a New Component . . . 45
7.2.1 Adding Ports . . . 46
7.2.2 Adding Properties . . . 47
7.2.3 Selecting Component Generation Options . . . 48
7.2.4 Generating the Source . . . 48
7.2.5 Building a Working Python Component . . . 49
7.2.6 Editing the SPD File . . . 50
7.2.7 Making Sure Files are Executable . . . 50
7.2.8 Building and Installing the Binaries . . . 50
7.3 Custom License Generation . . . 51
7.4 Removing Components, Devices, Nodes and Waveforms . . . 51
7.4.1 Removal Notes . . . 51
7.4.2 Component and Device Removal . . . 52
7.4.3 Node Removal . . . 52
7.4.4 Waveform Removal . . . 53

8 ALF Graphical Debugging Environment **54**

8.1 Running ALF 54

8.2 Running Waveforms 55

8.3 Managing Running Waveforms 55

8.4 ALF Tools 56

8.4.1 The Plot Tool 57

8.4.2 The write_to_file Tool 58

8.4.3 The Arbitrary Waveform Generator 58

8.4.4 The Speaker Tool 58

8.4.5 The Timing Tool 58

8.4.6 The Connect Tool 58

8.4.7 The Automation Tool 59

8.4.8 Automation XML File 59

9 WaveDash **61**

9.1 Running Wavedash 61

9.2 Installing and Un-installing waveforms 62

9.3 Selecting a Waveform 63

9.4 Refresh 64

9.5 GUI Customization 64

9.5.1 Show or Hide Components 64

9.5.2 Re-arrange components 66

9.5.3 Show or Hide Properties 66

9.5.4 Change Property Widget 67

10 Helping With Development **71**

10.1 Giving Feedback and Submitting Bugs 71

10.2 Contributing Code 71

11 Troubleshooting **72**

11.1 nodeBooter Throws an Exception 72
11.2 nodeBooter Fails to Resolve 'TRANSIENT' NameService 72
11.3 OWD Component Editor Does Not Allow me to Select the Device 72

Appendices **74**

A Managing Services on Fedora Core 9 **74**

B Creating `omniNames.sh` **74**

C Configuring `ossie.pth` **76**

D Data Source Component Creation **77**

E Known Bugs **79**
E.1 OSSIE Core Framework 79
E.2 OWD 79
E.3 ALF 79
E.4 OEF 80
E.5 WaveDash 80

F List of Abbreviations **81**

G Change Log **83**
G.1 OSSIE 0.7.4 83
G.2 OSSIE 0.7.3 83
G.3 OSSIE 0.7.2 83
G.4 OSSIE 0.7.1 83
G.5 OSSIE 0.7.0 84

Listings

1 `omniNames.sh` file 74

1 Quick Start Guide

The following is a quick start guide. It is meant to be a concise introduction to OSSIE. While it assumes general computer knowledge, it tries not to assume extensive knowledge of or familiarity with Linux or OSSIE. The quick start guide is a compact version of much of the rest of the guide. If you are interested in more in-depth coverage, we recommend you fully read the pertinent sections of the guide. The rest of this document should be able answer more of the questions that may arise during installation and provide deeper insight into OSSIE. That said, here is a short walkthrough to get you experimenting with OSSIE in short order.

1.1 Using the OSSIE VMware Player Image

The VMware Player application is needed to run the OSSIE demonstration waveform using our VMware image.

Unless VMware Player is already installed on your system, download it from http://www.vmware.com/download/player/ and install it. For detailed instructions, consult the VMWare Player User Guide [9].

Next, download the OSSIE VMware image at http://ossie.wireless.vt.edu/trac/wiki/Downloads. When the download is complete, unzip the image to another directory. We recommend you keep the original zip file. It can serve as a useful backup should you want or need to start from a fresh install and might save time and bandwidth.

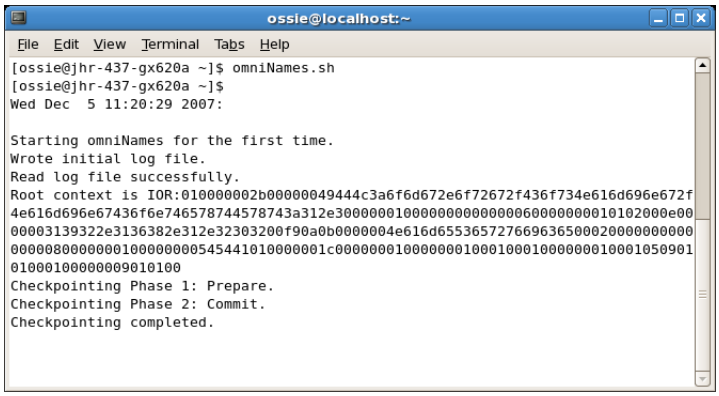
Once the image is unpacked, boot it up in VMware Player. This quick start guide will walk you though the process of running OSSIE and our demonstration waveform. The demonstration waveform simulates a very basic QPSK communication system. It consists of three components: a transmitter, a channel, and a receiver. The transmitter generates bursts of 512 QPSK symbols that are sent through an AWGN channel and decoded by the receiver. The number of errors per burst will be printed out on the screen while the waveform runs.

1.2 Running the OSSIE Demonstration Waveform

Once the OSSIE VMware image has booted up, open a terminal window by navigating to Applications →System Tools→Terminal. The first thing that needs to be started is the naming service. Type as root:

```
# omniNames.sh
```

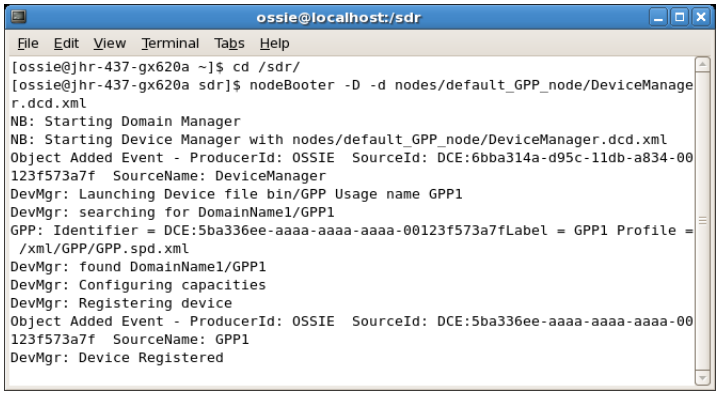
The terminal should look like this:



The next service that needs to be started is the Node Booter. Open a new tab or a completely new terminal and type the commands in the listing below. You can press tab after the first couple of letters of a command, directory or filename to have it automatically completed, saving time and perhaps avoiding typos.

```
$ cd /sdr/
$ nodeBooter -D -d nodes/default_GPP_node/DeviceManager.dcd.xml
```

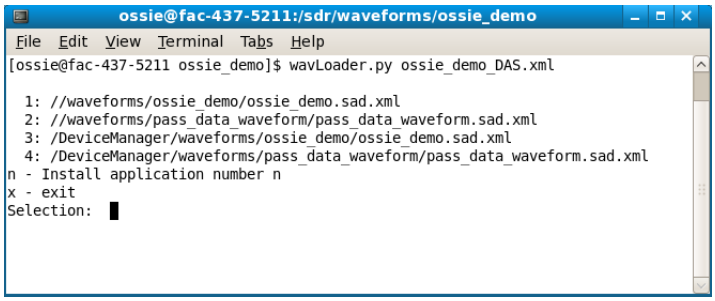
The terminal should look like this:



Now load the waveform using wavLoader. In a third tab or terminal, type in:

```
$ cd /sdr/waveforms/ossie_demo
$ wavLoader.py ossie_demo_DAS.xml
```

The terminal should look similar to the following:

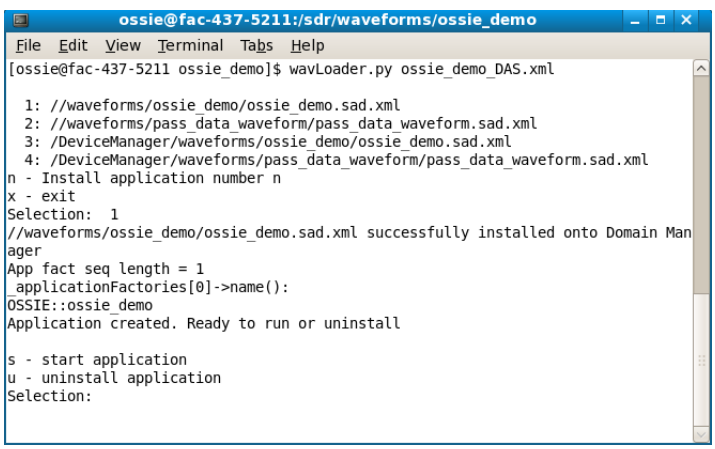


The terminal will list all of the available waveforms to run. Although multiple waveforms may be listed, the waveform that the wavLoader command was started with will only work.

Since the wavLoader command run on `ossie_demo_DAS.xml`, select the first listing of the OSSIE demonstration waveform, `//waveforms/ossie_demo/ossie_demo.sad.xml`

```
Selection: 1
```

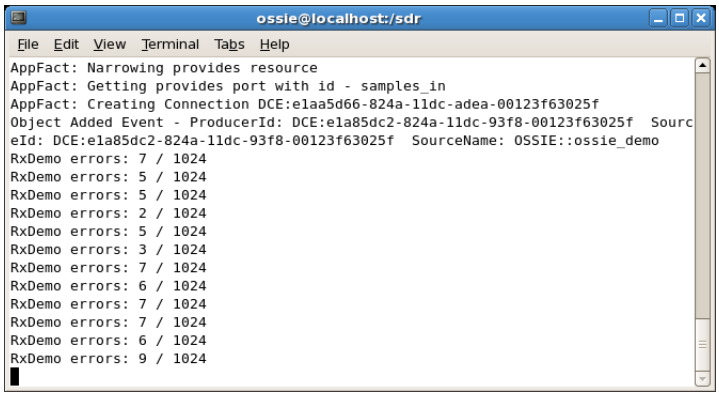
The terminal should look similar to the following:



Now that the waveform is loaded, it must be started. Enter `s` to start the waveform. The tab or terminal window in which nodeBooter was started should now contain the output of the demonstration waveform. The lines will update with output similar to this:

```
RxDemo errors:  X  / 1024
```

The X denotes the number of QPSK bit errors out of 1024 that have been detected by the RxDemo component. Here is an example output:



Congratulations! You have just run the QPSK demonstration waveform for OSSIE version 0.7.4. From here you may be interested in learning how to create your own waveforms using the OSSIE Waveform Developer (OWD) described in Section 7.

2 Introduction

2.1 About this Guide

The aim of the MPRG's Open Source SCA Implementation::Embedded (OSSIE) project is to provide an open source framework for testing the behavior of SDR waveforms and platforms. This software can help achieve a better understanding of software-defined radio and its trade-offs for educational and commercial purposes. This user guide is meant to be a comprehensive introductory and reference source for users of the OSSIE framework and tools, as well as serve as a useful information source for developers. In the guide, we attempt to assume only general computer knowledge and explain how things work along the way. Additionally, this guide is an evolving document. If you have any suggestions for improving it, please submit them to the mailing list (see Section 2.2.2, Trac (see Section 2.2.3) or email the maintainer directly.

2.2 Further Help

There are several ways to obtain help and further information. The online wiki, the Trac bug tracking tickets, the mailing lists and the IRC channel are available to you during moments of breakage, confusion, and inspiration.

2.2.1 Wiki

A wiki is a collaborative website tool. It allows many users to work together to write and revise web pages. Our wiki contains news, documentation and other information. You can visit it at http://ossie.wireless.vt.edu/trac/wiki.

2.2.2 Mailing Lists

We maintain two public mailing lists for OSSIE. *ossie-discuss* is for general help and discussion. *ossie-dev* is meant for more technical discussion and development-related traffic. Both lists can be subscribed to by sending a message to listserv@listserv.vt.edu with the following in the message body, replacing the information in angle brackets with your appropriate information.

```
subscribe <list name>  <your first name>  <your last name>
```

To send a message of all subscribers to the list, address it to ossie-discuss@listserv.vt.edu or ossie-dev@listserv.vt.edu, as appropriate.

2.2.3 Trac Bug Tracking System

The Trac software provides a web-based interface to our bug tracking database. Reporting previously unreported bugs is extremely helpful to the developers, allowing you to inform them of problems not encountered during their testing. Bug trackers are also good places to search for known problems and work-arounds. You will need to register for an account on the website and log in before being allowed to submit new bugs. Visit http://ossie.wireless.vt.edu/trac/newticket to initiate the bug reporting process. Please be sure to tag the bug with version 0.7.4. We will strive to squash all bugs as soon as possible and incorporate usability and feature improvement suggestions into an appropriate future release. If you're interested in crafting very helpful bug reports, consider reading Simon Tatham's essay on the topic [13].

2.2.4 The #ossie IRC Channel

Several OSSIE team members idle in an Internet Relay Chat to aid those interested in real-time communication. Our channel, #ossie, is on the Freenode network. To visit it, try clicking on the channel name in the previous sentence or fire up your favorite IRC client, log onto the Freenode servers and join the #ossie channel. If you have never used IRC before, irchelp.org's IRC Tutorial is a good starting point.

3 Installation

For users that do not have access to a Linux system, or are unfamiliar with Linux, we recommend that you use our pre-built VMWare images. These images have OSSIE pre-installed on them, and are ready to use out-of-the-box. Using an OSSIE VMware image is described in Section 3.3.

For most users on Fedora systems, installing OSSIE and its dependencies via yum is the best route. The only reason not to install via yum is if you plan on doing active development *on* OSSIE, and need access to the source code. Note that this does not include application/waveform developers who are developing *with* OSSIE, as they do not need access to the OSSIE source code.

Although we do not yet provide and support binary installation packages for more Linux distributions, compiling OSSIE from source on them should be possible.

To install OSSIE from source, follow the below instructions to install the OSSIE dependencies, and then compile and install the source code. The latter process is simplified with a provided convenience script, build.py, which compiles and installs OSSIE from source for you.

3.1 Installing OSSIE from Source

3.1.1 Notes on OSSIE Installation from Source

This section assumes that the user is familiar with basic Linux commands such as `cd` and `ls`. Commands run with root permissions are prefixed with `#`, while others are prefixed with `$`.

OSSIE depends on the following software packages:

- Xerces - a C++ XML parser
- omniORB - a CORBA implementation
- omniORBpy - a Python interface for omniORB, necessary for the OSSIE tools and components
- wxPython - a Python interface for the wxWidgets graphical library, used by the OSSIE tools
- numpy - a Python numerical library used by the OSSIE tools

OSSIE 0.7.4 is packaged with a convenient Python script called `build.py` to automatically configure and build the packages necessary for the framework, tools, and other libraries. Use of this script is not mandatory, and you can manually compile and install all/part of OSSIE as you wish.

3.1.2 Installing Dependencies on Fedora

On Fedora systems, the dependencies can be installed via `yum`. This is the entire dependency list, so some of these packages may already be installed.

```
# yum -y install xerces-c xerces-c-devel omniORB omniORBpy \
wxPython wxPython-devel numpy rpm-build cabextract glibc-devel \
python-devel openssl-devel gcc gcc-c++ libtool \
boost-1.33.1 boost-devel-1.33.1
```

If you are using Fedora Core 10, you will need to download the RPM for the SDL library in order to user the JPEGVideoViewer component.

```
$ wget http://www.libsdl.org/release/SDL-devel-1.2.13-1.i386.rpm
$ rpm -Ui SDL-devel-1.2.13-1.i386.rpm
```

3.1.3 Installing Dependencies on Ubuntu

This is the entire dependency list, so some of these packages may already be installed. Install all of the following packages:

```
$ sudo aptitude install gcc build-essential
$ sudo aptitude -y install omniorb4 libomniorb4-dev omniidl4-python \
omniorb4-nameserver python-omniorb2 libgtk2.0-dev freeglut3-dev \
libxerces27-dev libxerces27 python-wxgtk2.8 python-wxversion \
python-wxtools python-numpy python-numpy-ext python-numpy-dev g++ \
automake libtool subversion python-dev fftw3-dev libcppunit-dev \
libboost-dev sdcc libusb-dev libasound2-dev libsdl1.2-dev guile-1.8 \
libqt3-mt-dev swig libboost-filesystem-dev python-profiler \
automake1.9 python2.5-dev libboost1.35-dev sdcc-nf guile-1.8-dev \
libqt4-dev ccache python-opengl libgsl0-dev python-lxml doxygen \
qt4-dev-tools libqwt5-qt4-dev libqwtplot3d-qt4-dev
```

3.1.4 Configure omniORB

omniORB must be configured through the modification of a file. This file may be either `/etc/omniORB.cfg` or `/etc/omniORB4.cfg`, depending on the version of the omniORB dependency. Open the file as root, and search for the following line:

```
InitRef = NameService=corbaname::my.host.name
```

Uncomment the line by deleting the pound or hash character '#' and change it to:

```
InitRef = NameService=corbaname::127.0.0.1
```

Before running the framework you will need to start the naming service. `omniNames` will automatically start when you restart your machine. Refer to Appendix A for instructions on how to start the omniNames service without restarting your machine or set whether the service starts automatically at boot. [1]

[1] You may also manually start a source-compiled tarball package of omniNames `omniNames` with the script described in Appendix B

At this time it is a good idea to restart your machine.

3.1.5 Installing Portions of GNU Radio

OSSIE uses a small subset of GNU Radio to communicate with and configure the USRP. The following steps will walk through installing portions of GNU Radio.

First, GNU Radio's dependencies must be installed. If you are using Ubuntu and followed the instructions for installing the dependencies, skip down to checking out the source code from the GNU Radio Subversion repository. If you are installing on Fedora 9, proceed with the following directions.

Install the Engineering and Scientific packages as well as the Software Development packages as root entering the command:

```
# yum groupinstall "Engineering and Scientific" "Development Tools"
```

Additional utilities such as the FFT library and the CPP Test Framework must also be installed. As root, enter the following command

```
# yum install fftw-devel cppunit-devel wxPython-devel libusb-devel  \
  guile boost-devel alsa-lib-devel numpy
```

The Small Device C Compiler, SDCC, must be installed:

```
# yum install sdcc
```

The path for the SDCC must be set. Open the .bashrc file and add the path to the end of the file. To open the file, enter the command:

```
$ vim ~/.bashrc
```

Add the following path to the end of the file:

```
export PATH=/usr/libexec/sdcc:$PATH
```

At this point, all of the GNU Radio dependencies have been installed. Now, the GNU Radio software must be installed.

Download the GNU Radio source code by entering the command:

```
$ svn co http://gnuradio.org/svn/gnuradio/branches/releases/3.1 gnuradio
```

Move into the `gnuradio` directory and start building the source code by entering the following commands:

```
$ cd gnuradio/
$ ./bootstrap
$ ./configure --disable-all-components --enable-gnuradio-core \
  --enable-usrp --enable-gr-usrp --enable-omnithread
```

This sets up the install to only build resources for the USRP which OSSIE requires. Compile the source code by entering the command:

```
$ make
```

Verify that the compile worked by running a check:

```
$ make check
```

Install the portions of GNU Radio by running the following command as root:

```
# make install
```

The libraries installed by GNU Radio need to be linked:

```
# /sbin/ldconfig
```

At this point, GNU Radio and its dependencies have been installed. Now setup the proper permissions for the USRP. As root, create a group which will have access to the USRP:

```
# /usr/sbin/groupadd usrp
```

Add users to the group which need access to the USRP:

```
# /usr/sbin/usermod -G usrp -a USERNAME
```

Now that users will have access to the USRP, read and write access to the device must be created. As root, create the file `/etc/udev/rules.d/10-usrp.rules` in a text editor:

```
# vim /etc/udev/rules.d/10-usrp.rules
```

Add the following text to the file:

```
ACTION=="add", BUS=="usb", SYSFS{idVendor}=="fffe", \
SYSFS{idProduct}=="0002", GROUP:="usrp", MODE:="0660"
```

The text above is displayed on two lines due to the contraints on page size, however the text must appear on a single line, without the backslash, in the file for the access to the USRP to work properly. You may also add the following comment lines to the file for future reference:

```
# rule to grant read/write access on USRP to group named usrp.
# to use, install this file in /etc/udev/rules.d/ as
# 10-usrp.rules
```

The USRP interface has now been created. As an optional test, connect the USRP to the computer and run the following command:

```
$ ls -lR /dev/bus/usb
```

The users root and usrp should now be listed under the user groups.

3.1.6 Install OSSIE

You are now ready to build and install OSSIE. At this point, you can either compile and install everything by hand using the included Makefiles, or use the `build.py` convenience script to do it for you. Note that installing by hand requires some knowledge of OSSIE, Linux, and software development.

Download the latest tarball from http://ossie.wireless.vt.edu/download/tarballs/0.7.4/ and unpack `ossie-0.7.4.tar.bz2`.

```
$ wget http://ossie.wireless.vt.edu/download/tarballs/0.7.4/ \
  ossie-0.7.4.tar.bz2
$ tar -xvjf ossie-0.7.4.tar.bz2
```

If you want to install everything by hand, do so, and then move on to Section 3.1.8.

If you would rather use the script, follow the steps below.

3.1.7 Using Installation Scripts

The included `build.py` and `setup.py` scripts automate the building and installation of the OSSIE packages. Because OSSIE is built as a set of dependent libraries it is necessary to build and install each one separately so that they can link properly. These scripts take care of that for you.

In order for the script to run uninterrupted, the `sudo` command must be able to execute `make install`. As root, edit the `sudoers` file[2]

```
# /usr/sbin/visudo
```

At the end of the file, add the following line:

```
ALL ALL = NOPASSWD: /usr/bin/make install
```

Save and quit (`:wq`). If you made an error, visudo will tell you. We recommend that you comment this line out by putting a pound character ('#') in front of it once you have finished installing OSSIE.

By default, the installation directory of the OSSIE platform is `/sdr`. In order to install new source code and binaries into this directory without root permissions, you need to create and change the ownership of `/sdr`.

```
# mkdir /sdr
# chown -R username.username /sdr
```

where `username` is your user name.

The `setup.py` script installs the tools used for component and waveform development. To install the tools, move into the `tools` directory and run the `setup.py` script as root:

[2] By default the sudoers file must be edited with the `visudo` command

```
$ cd /home/username/path_to_ossie/tools
# python setup.py install
```

The `build.py` script installs the core framework, components, devices and a few demonstration waveforms. This script should not be run as root. To install, first exit out of root, then move into the directory where the OSSIE 0.7.4 tarball has been unpacked and run the script:

```
# exit
$ cd /home/username/path_to_ossie/
$ python build.py
```

Depending on the speed of your system this might take several minutes. You may be asked for a password during the first `make install` command. Because we are using `sudo`, this prompt is asking for your user password, not the root password. If successful, the prompt should say:

```
*********************************************************

  Complete installation of OSSIE 0.7.4 finished!

*********************************************************
```

3.1.8 Updating System Libraries

Once the OSSIE libraries are installed, they need to be linked. As root edit the file `/etc/ld.so.conf`, adding the line

```
/usr/local/lib
```

Now run:

```
# /sbin/ldconfig
```

OSSIE should now be successfully installed on your system. You can skip to Section 4 to learn how to run waveforms.

3.2 Installation of OSSIE Eclipse Feature

Installation of the OSSIE Eclipse Feature (OEF) requires the installation of OSSIE, Java, and Eclipse.

3.2.1 Installing Java

Eclipse is written in Java, so you must have it installed to run Eclipse and OEF. We recommend using Sun's Java Development Kit. As of this writing, the GNU Compiler for Java (GCJ) will not work.

Fedora Core 9 comes with the Sun JDK pre-installed but older versions require manual installation. On other distributions it is advisable to use the package manager to manage the installation, if possible.

3.2.2 Installing Java on Older Versions of Fedora

Go to http://java.sun.com/javase/downloads/index.jsp and click Download next to JDK 6 Update 6. Choose Linux in the platform drop down menu, click the check-box agreeing to the license agreement, and click continue. Download the Linux rpm in self extracting file, `jdk-6u6-linux-i586-rpm.bin`.

Open a terminal and navigate to the file. As root, execute the following command:

```
# sh jdk-u6-linux-i586-rpm.bin
```

Create `java.sh` in `/etc/profile.d/` with the following contents:

```
export JAVA_HOME="/usr/java/latest"
export JAVA_PATH="$JAVA_HOME"
export PATH="$PATH:$JAVA_HOME/bin"
```

Log out and back in to allow the changes to update.

3.2.3 Installing Java on Ubuntu

In a terminal, enter the following lines:

```
$ sudo apt-get update
$ sudo apt-get install sun-java5-jdk
```

3.2.4 Installing Eclipse

Install the Eclipse IDE for Java Developers. Go to the Eclipse Download Center and download an Eclipse distribution for your platform.

Eclipse is distributed as a tarball archive that you can unpack to location of your choice. Pick a location that is appropriate for your platform and simply unpack the contents. There is no self-installer, just unpack the distribution. Do not install Eclipse in a directory that has spaces anywhere in its full path name.

3.2.5 Installing OEF

Now that the dependencies have been installed, OEF can be installed.

Move into the unpacked eclipse directory, and start eclipse:

```
$ cd /path/to/eclipse
$ ./eclipse
```

After Eclipse starts, on the toolbar select `Help`, `Software Updates`. In the new window, select the "Available Software" tab and click the "Add Site" button the right hand side. Enter the URL: `http://ossie.wireless.vt.edu/eclipse/` and select OK. The window will then add the URL, OSSIE, OSSIE Waveform Developer Feature to the list of software to update. Place a check in the box next to OSSIE Waveform Developer Feature and click Install. Eclipse will bring up a window which will resolve any dependencies and then start the install guide.

When the install window opens, make sure OSSIE Waveform Developer Feature is selected and click Next. Accept the GNU license agreement and select Next. Eclipse will then begin to download the necessary files, which may take a few minutes. Allow Eclipse to restart when it prompts to do so. After it restarts, the OSSIE Eclipse Feature will be installed.

Select the OSSIE perspective within Eclipse. On the toolbar, select `Window`, `Open Perspective`, `Other`. In the new window, select OSSIE which will then open the OSSIE perspective. On the toolbar, select `File`, `New`, `OSSIE Waveform`, or `OSSIE Component` to start developing.

These same instructions used for installing OEF can be used later to update it to newer versions.

3.3 Using a VMware Image on Any Platform

A VMware image of a complete Fedora Core 9-based Linux system with all necessary dependencies and a complete install of OSSIE 0.7.4 is available at http://ossie.wireless.vt.edu/trac/wiki/Downloads. All that is needed to run the virtual image is the VMWare Player, available for no fee from http://www.vmware.com/download/player/. Versions of the player are available for both Windows and Linux.

Install VMware Player on your system, unzip the virtual image and open it. For full instructions on installing and using VMware player, please consult the VMWare Player User Guide [9].

It is recommended that you keep a copy of the zipped virtual image so that you do not need to download the image a second time to start with a fresh copy. Changes that you make from within the image will alter it, and in the event of drastic unwanted changes, starting afresh is easy if you have an extra copy of the image on your hard drive.

4 Running Waveforms

4.1 Starting the CORBA Naming Service

If you installed omniORB using `rpm` or your system package manager and have since restarted your machine chances are the naming service is running. Refer to Appendix A for how to manage services under Fedora Core 9. If you chose to install omniNames from source, you will need to run `omniNames.sh` (see Appendix B).

4.2 Running `nodeBooter`

To run `nodeBooter`, open a terminal and execute the following:

```
$ cd /sdr
$ nodeBooter -D -d nodes/default_GPP_node/DeviceManager.dcd.xml
```

It is important that `nodeBooter` be run from the `/sdr` directory because nodeBooter uses paths that are defined relatively to the directory in which it is run.

4.3 Nodebooter Clean-Up

If a waveform crashes or is uninstalled incorrectly, Nodebooter will not be able to shut down all of the processes that it starts. This is currently being addressed by the development team, however in the interim the processes must be stopped by hand. To find which processes were not stopped, enter the following command:

```
$ ps -e
```

This will list all of the currently running processes. Processes with the names of components or devices that are in the waveform need to be shutdown. To stop a process, enter the following command:

```
$ killall USRP GPP <MORE-DEVICES> TxDemo Decimator <MORE-COMPONENTS>
```

If the USRP node (default_GPP_USRP_sound_node) cannot be started, this is typically resolved by killing the USRP, soundCardPlayback and soundCardCapture devices.

To make this process faster, create a script to kill all of the processes created by a certain waveform. For example, create a text file called `killOSSIEDemo`.

```
$ cd ~
$ vim killOSSIEDemo
```

In this text file, press `i` to insert text and enter the following on a single line:

```
killall GPP TxDemo ChannelDemo RxDemo
```

Press <ESC>, :wq, and then <ENTER> to save the file and exit. Now change the permissions so the script can be executed.

```
$ chmod +x killOSSIEDemo
```

Now the script has been created and can be run by entering the command:

```
$ ./killOSSIEDemo
```

If not all processes within the script are running a warning will be printed stating that the process has not been killed, which is fine.

4.4 Loading a Waveform

The 0.7.4 release supports two options for loading a waveform:

1. The command-line `wavLoader.py` script discussed below
2. The graphical tool ALF described in Section 8

Directions here are for running the OSSIE demonstration waveform, `ossie_demo`. The demonstration waveform simulates a very basic QPSK communication system. It consists of three components: a transmitter, a channel, and a receiver. The transmitter generates bursts of 512 QPSK symbols that are sent through an AWGN channel and decoded by the receiver. The number of errors per burst will be printed out on the screen while the waveform runs. To run another waveform, substitute `ossie_demo` with the name of your waveform.

Load the waveform using wavLoader. In a terminal window, execute:

```
$ cd /sdr/waveforms/ossie_demo
$ wavLoader.py ossie_demo_DAS.xml
```

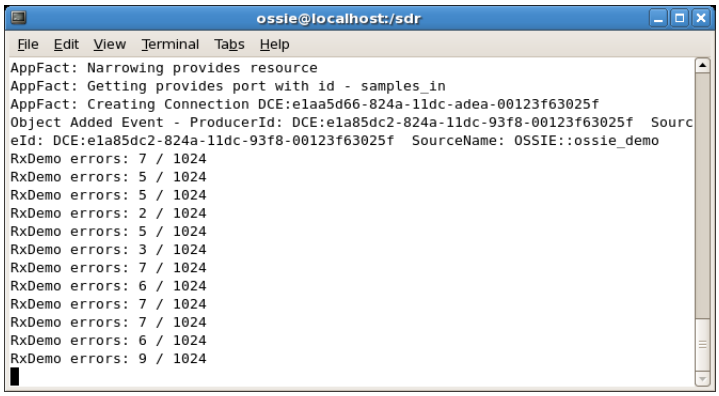
One or more SAD files (ending in `.sad.xml`) will be listed. To load the waveform, enter the number that corresponds to `ossie_demo.sad.xml`. If this file is listed twice choose the first listing, for example: (`//waveforms/ossie_demo/ossie_demo.sad.xml`).

Finally, enter `s` to start the waveform.

The terminal window in which nodeBooter was started will now contain the output of the demonstration waveform. The lines will update with output similar to this:

```
RxDemo errors:  X  / 1024
```

X denotes the number of QPSK bit errors out of 1024 that have been detected by the RxDemo component. The terminal should look like this:



Congratulations! You have just run the QPSK demonstration waveform for version 0.7.4. From here you may be interested in learning how to create your own waveforms and components using the OSSIE Eclipse Feature described in the following section or the OSSIE Waveform Developer described in Section 7.

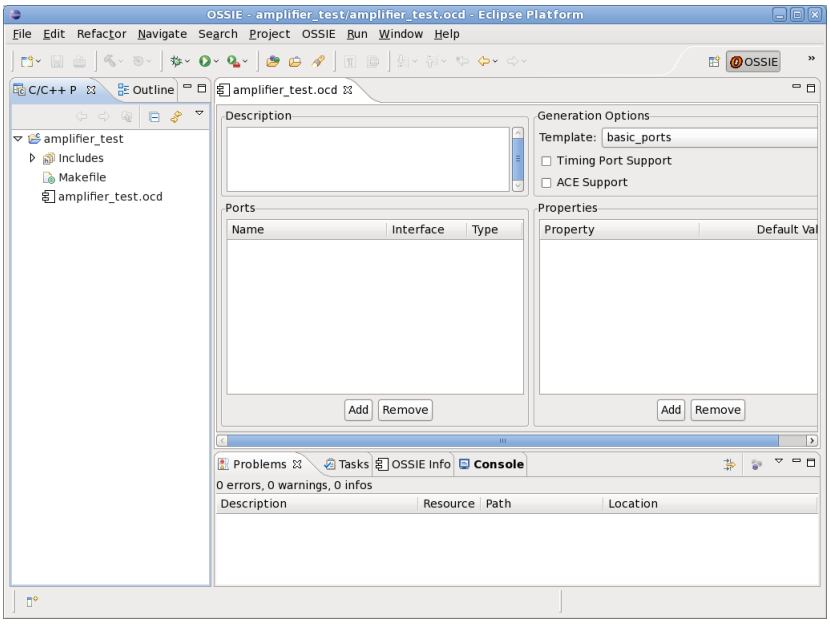
5 Waveform Workshop

The Waveform Workshop is the collection of development and debugging tools associated with the OSSIE Core Framework. The Waveform Workshop is comprised of OSSIE Eclipse Feature (Section 6), OSSIE Waveform Developer (Section 7), ALF Graphical Debugging (Section 8), and WaveDash (Section 9).

5.1 OSSIE Eclipse Feature

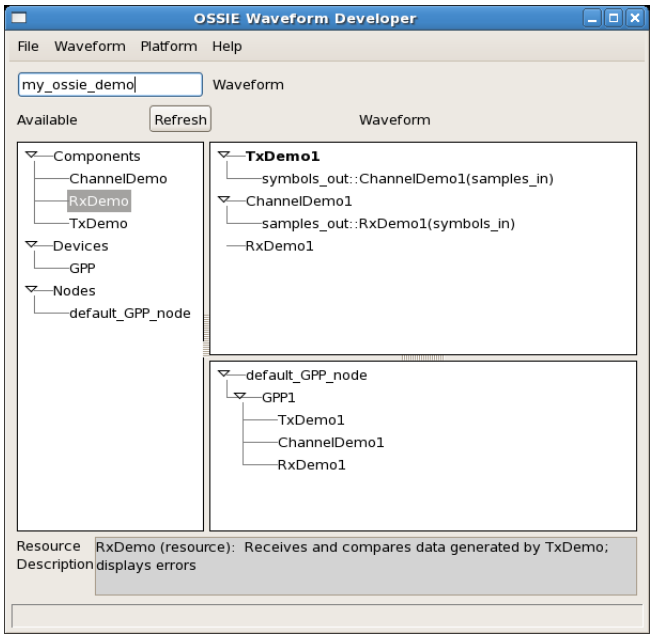
OEF is a development tool and it provides the ability to create signal processing components which the core framework can run, as well as the ability to connect multiple components together to create a waveform.

OEF is an Eclipse plug-in and requires the installation of Java and Eclipse to run.



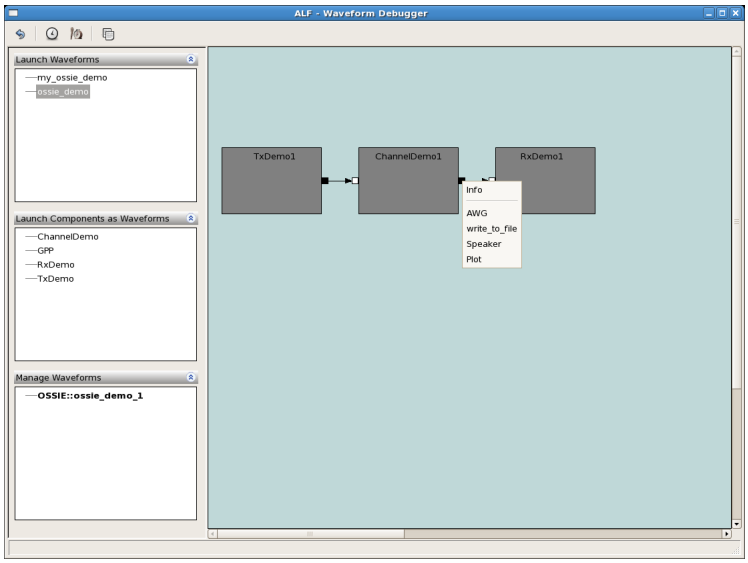
5.2 OSSIE Waveform Developer

OWD is a development tool and it provides the same capabilities of OEF, however the application is standalone and only requires the installation of python to run.



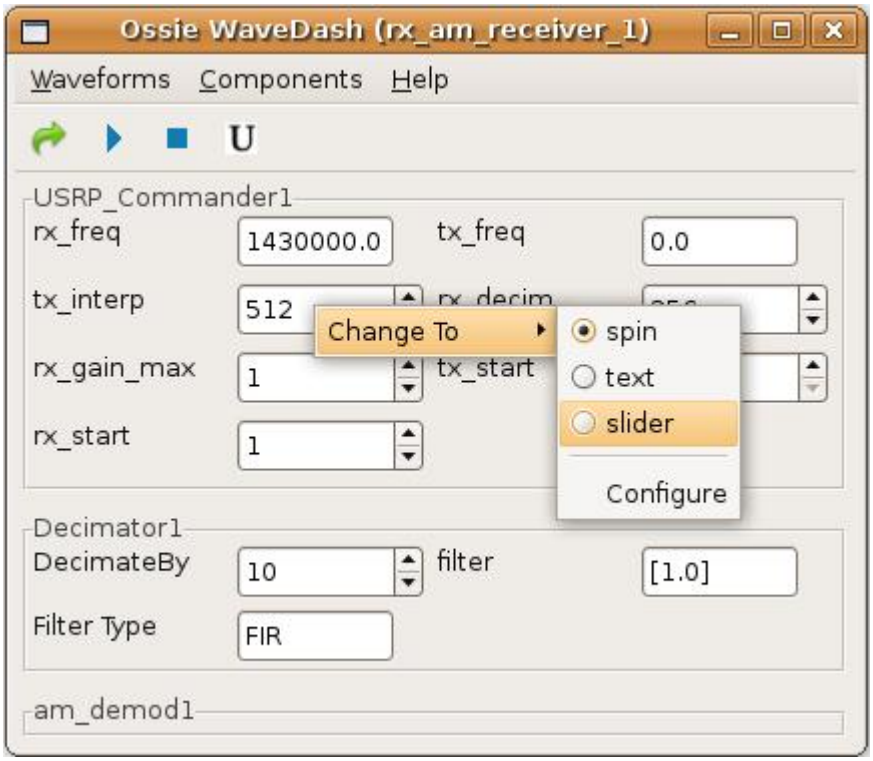
5.3 ALF Graphical Debugging

ALF is a debugging tool which allows waveforms to be debugged visually. It provides the capability to run waveforms and view the spectrum and the I/Q plot at the input or output of any component in a running waveform.



5.4 WaveDash

WaveDash is a GUI application to interactively configure component properties of SDR waveforms. The GUI can further be customized to show or hide component and their properties on per waveform basis.



6 OSSIE Eclipse Feature

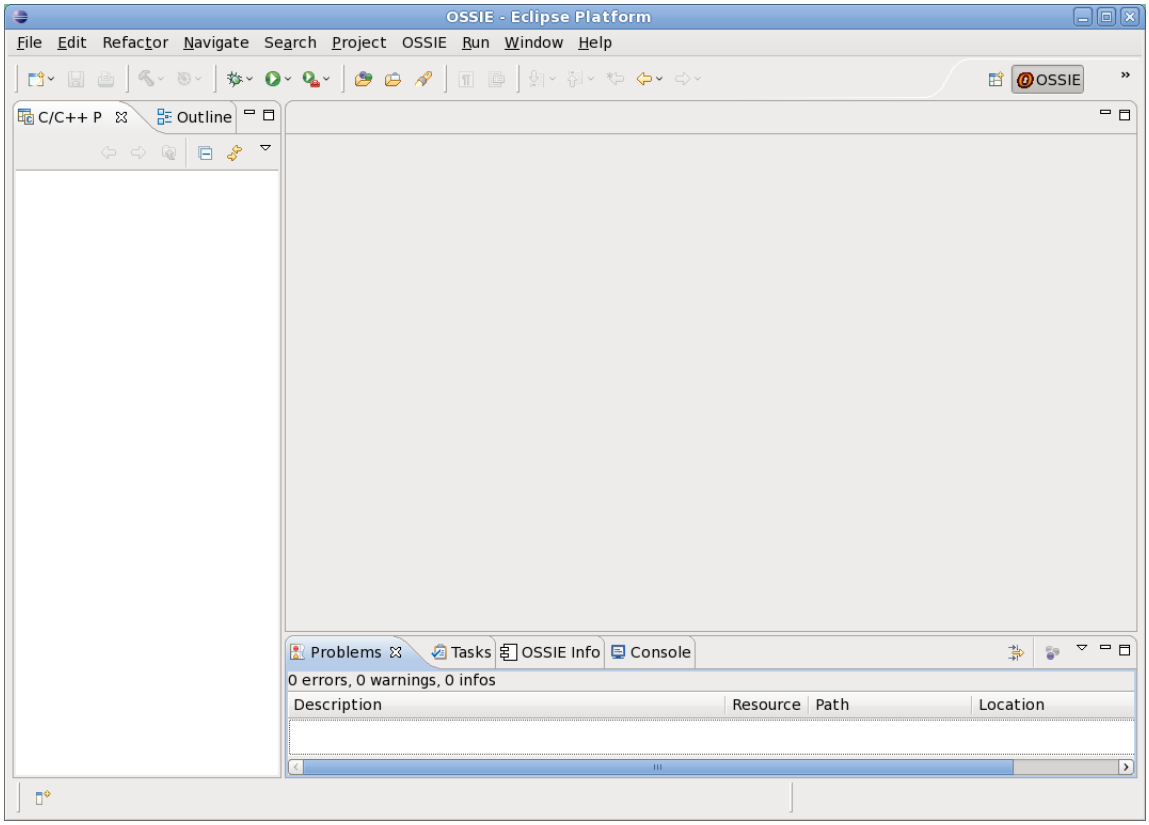
The OSSIE Eclipse Feature (OEF) is an interface for working with OSSIE tools and the core framework. OEF is a drag and drop interface for creating waveforms, as well as creating components and interfacing with the OSSIE core framework.

6.1 Creating a New Waveform from Existing Components

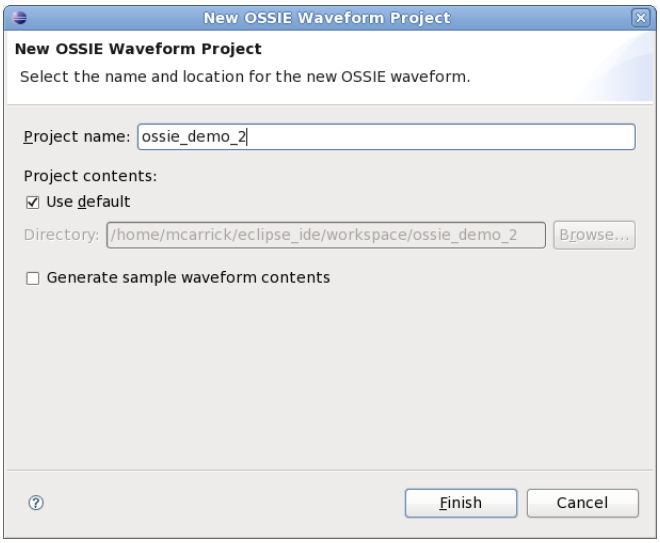
To start Eclipse on the Fedora 9 VMware Image, open a terminal and enter the following command:

```
$ /home/ossie/ossie/src/eclipse/eclipse
```

The main window will appear:

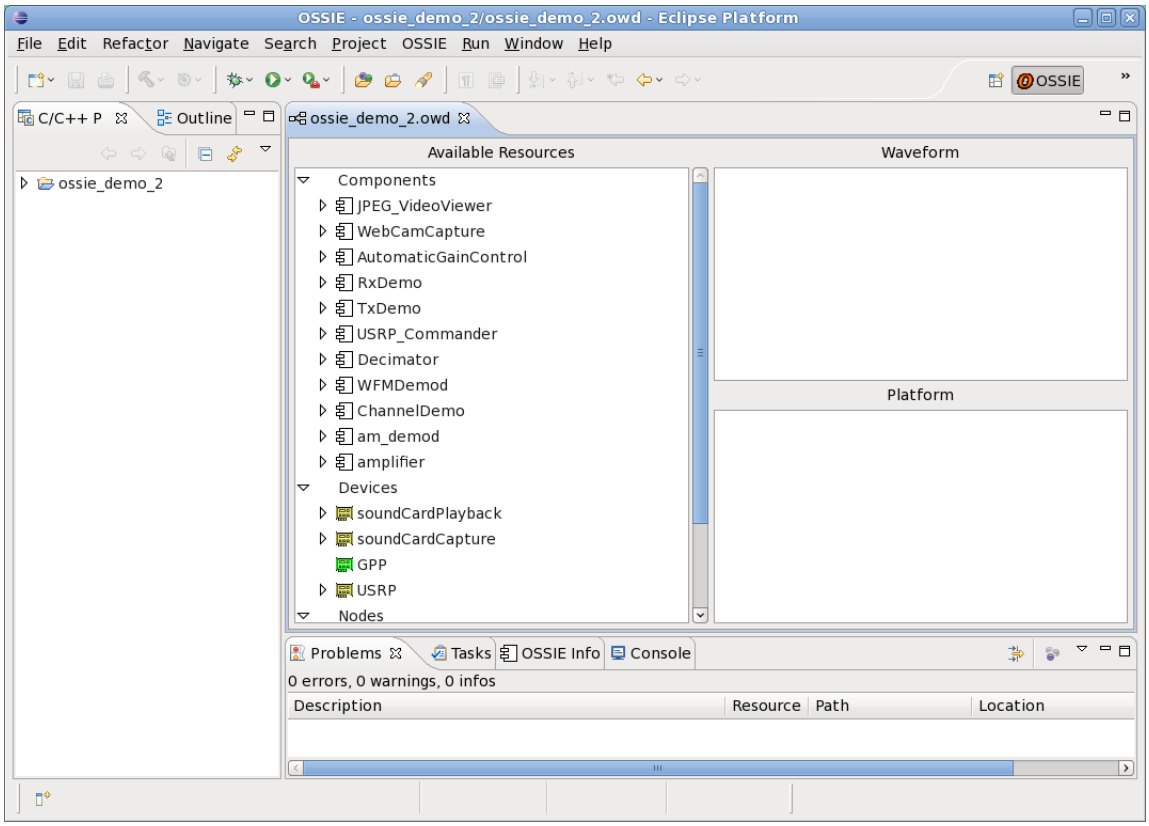


As an example, `ossie_demo` will be recreated using OEF. On the toolbar, start by selecting File : New : OSSIE Waveform. This will bring up the New OSSIE Waveform Project window. Enter ossie_demo_2 for the Project Name and keep Use default box selected. Select Finish.



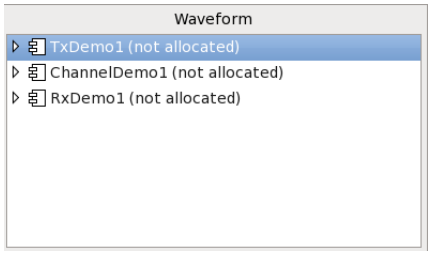
There are two problems to be aware of when naming a waveform. The first is to remove all spaces from a waveform name or replace them with an underscore. The second problem is waveforms cannot start with the text AM_. This prefix is interpreted as an automake macro and will prevent the project from building.

This window displays three panels: Available Resources, Waveform, and Platform. Available resources lists all of the available components, devices, and nodes that a waveform can use. The Waveform panel shows the waveform being built, and the Platform panel shows how the resources are deployed.



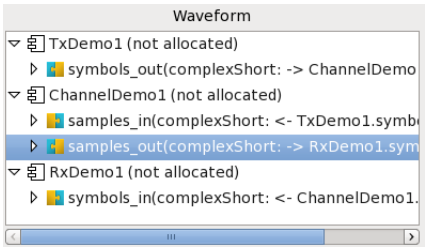
6.1.1 Adding Components to the Waveform

From the Available Resources panel, under Components locate the TxDemo component and drag it into the Waveform panel. Do the same for the ChannelDemo and RxDemo components. The Waveform Panel now shows the three components which will make up the logical and signal processing blocks of the waveform.



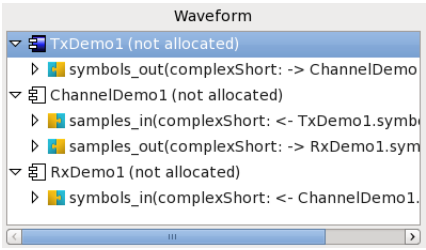
6.1.2 Connecting Components

Connect the components by first selecting the arrow next to each component name in the Waveform panel. This will display all of the provides and uses ports for each component. Under TxDemo, click and drag the symbols_out port to the samples_in port under ChannelDemo. The puzzle pieces should connect next to the port names, denoting a connection has been made. Connect samples_out under ChannelDemo to symbols_in under RxDemo to complete all of the connections.



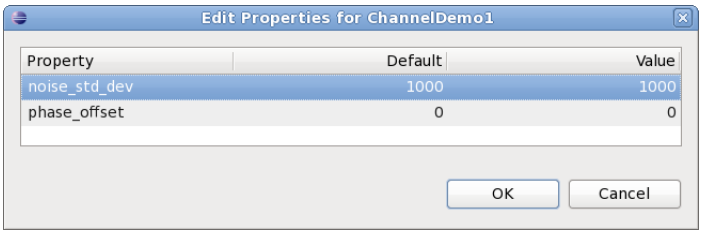
6.1.3 Setting the Assembly Controller

After all of the connections have been made, the assembly controller must be defined. Right click on TxDemo and select Set Assembly Controller. The icon for the component will then turn dark blue indicating it is the Assembly Controller.



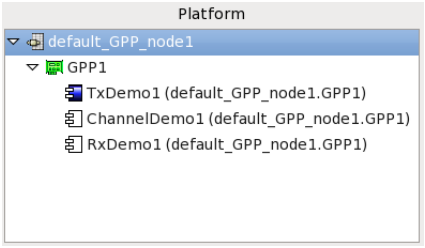
6.1.4 Editing Component Properties

It is possible to change properties of components using OEF. Right click on ChannelDemo in the Waveform panel and select Edit Component Properties. Two properties are displayed, noise_std_dev and phase_offset. To change the value of either, click on the right box under the label Value and enter a new value and select OK. Properties of other components can be changed using this method, however the properties that are available depend on the component.



6.1.5 Deploying Components to a Node

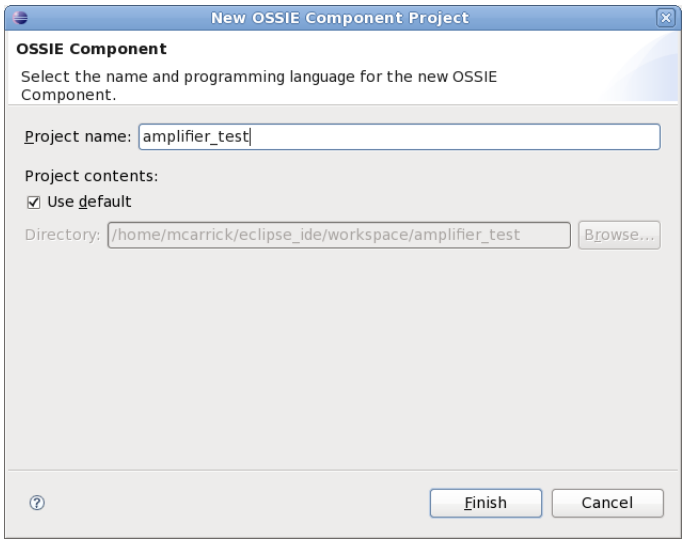
Now the platform must be defined. In the Available Resources panel, under Nodes select default_GPP_node and drag it into the Platform panel. Click on the arrow next to the name to display the GPP device. The waveform will run on the GPP of the local machine, so all of the components must be deployed to this device. In the Waveform panel, select TxDemo and drag it onto the GPP device. Click the arrow next to the GPP device to show that TxDemo has been deployed to it. Repeat the procedure for ChannelDemo and RxDemo.



The waveform should be completely defined and ready to be run. Save the waveform by pressing CTRL+S or File : Save, which will also compile and install it to the `/sdr/waveforms/` directory.

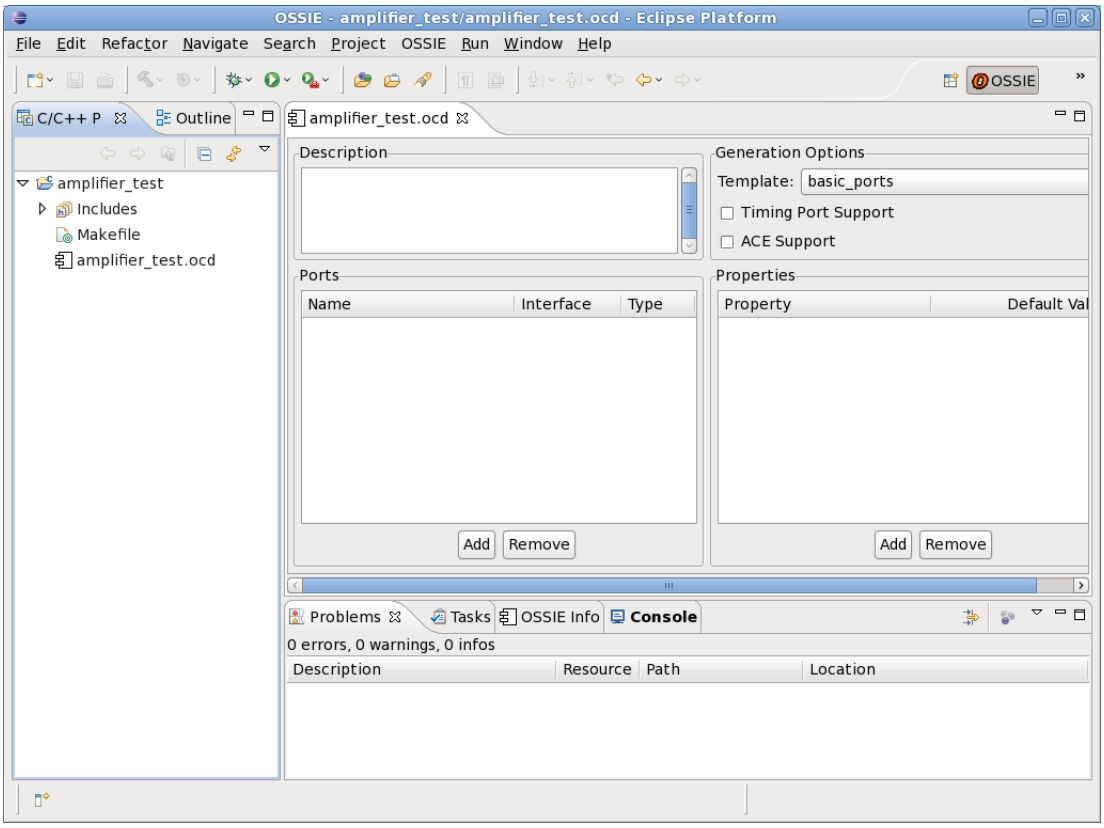
6.2 Creating a New Component

This section will walk through the steps required to build a new component from scratch. To generate a new component, on the toolbar within OEF select File : New : OSSIE Component. Enter amplifier_test for the Project Name, keep the Use default check box selected and select Finish.



There are two problems to be aware of when naming a component. The first is to remove all spaces from a waveform name or replace them with an underscore. The second problem is waveforms cannot start with the text AM_. This prefix is interpreted as an automake macro and will prevent the project from building.

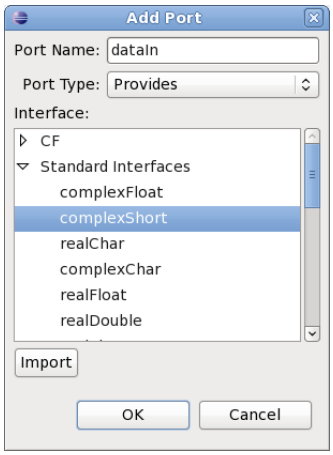
This will bring up the Component Editor workspace which has four main panels: Description, Generation Options, Ports, and Properties. The Description panel is where the developer enters a basic description of the component. The Generation Options panel defines the type of ports to be used and if Timing Port Support and ACE Support are enabled. The Ports panel allows for the addition and removal of ports for the component. The Properties panel allows for the addition and removal of editable properties for the component.



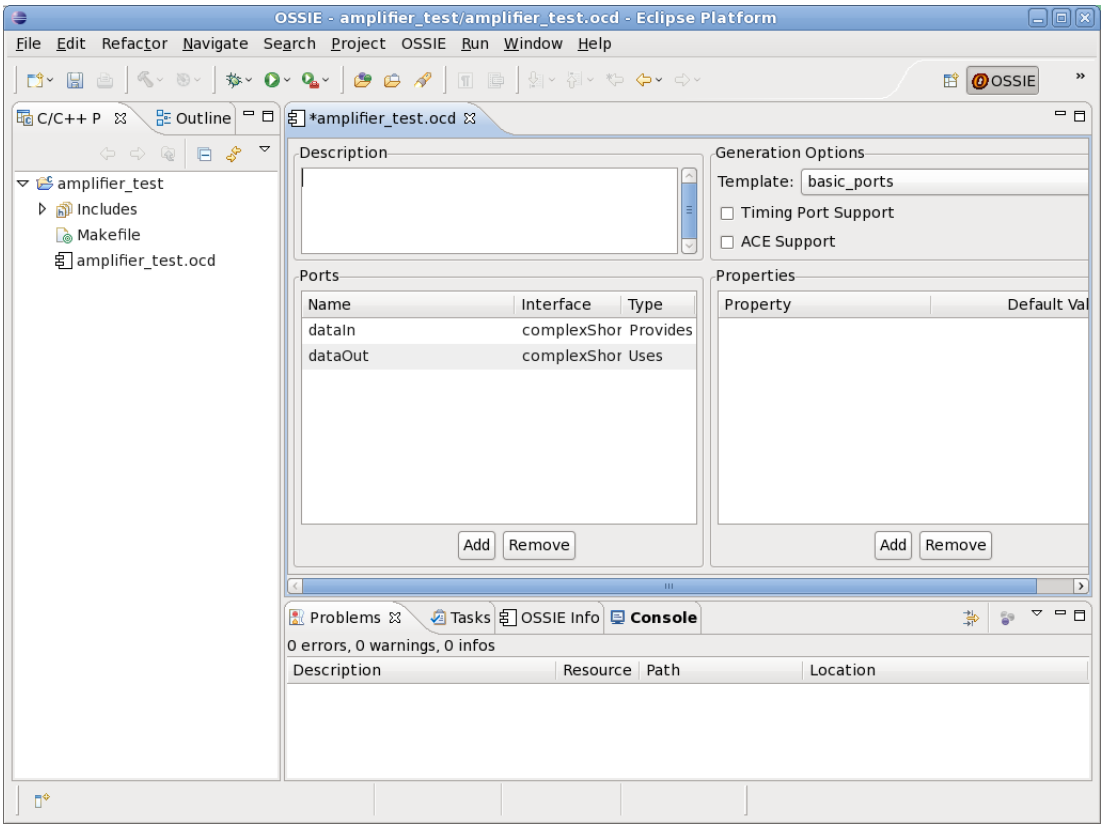
6.2.1 Adding Ports to the Component

To add a port to the component, select Add in the Ports panel. Things brings up the Add Port window, where the name, port type, and interface is defined. Enter `dataIn` as the Port Name, select Provides as the Port Type, and select Standard Interfaces : complexShort as the interface. [3] Select OK to add the port.

[3] Within the SCA, input and output ports are renamed to provides and uses. Depending on how the framework is implemented, the input can be either named as a provides or a uses. The method that OSSIE uses implements input ports as provides, and output ports as uses.

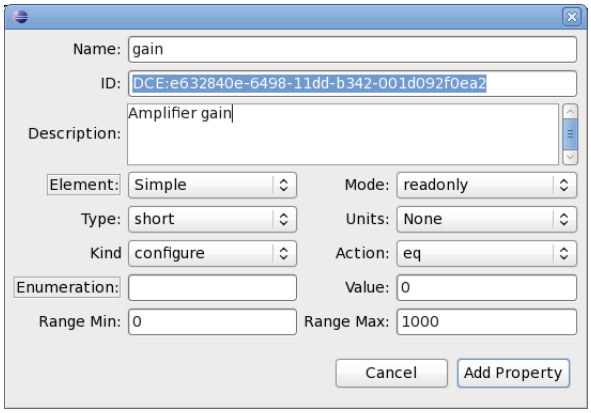


Select Add again, and enter dataOut as the Port Name, Uses as the Port Type and Standard Interfaces : complexShort as the Interface. Select OK to add the port. OEF should now look like the following:

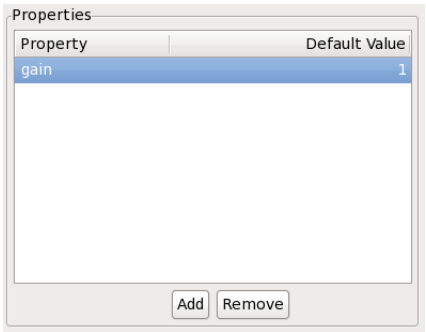


6.2.2 Adding Properties to the Component

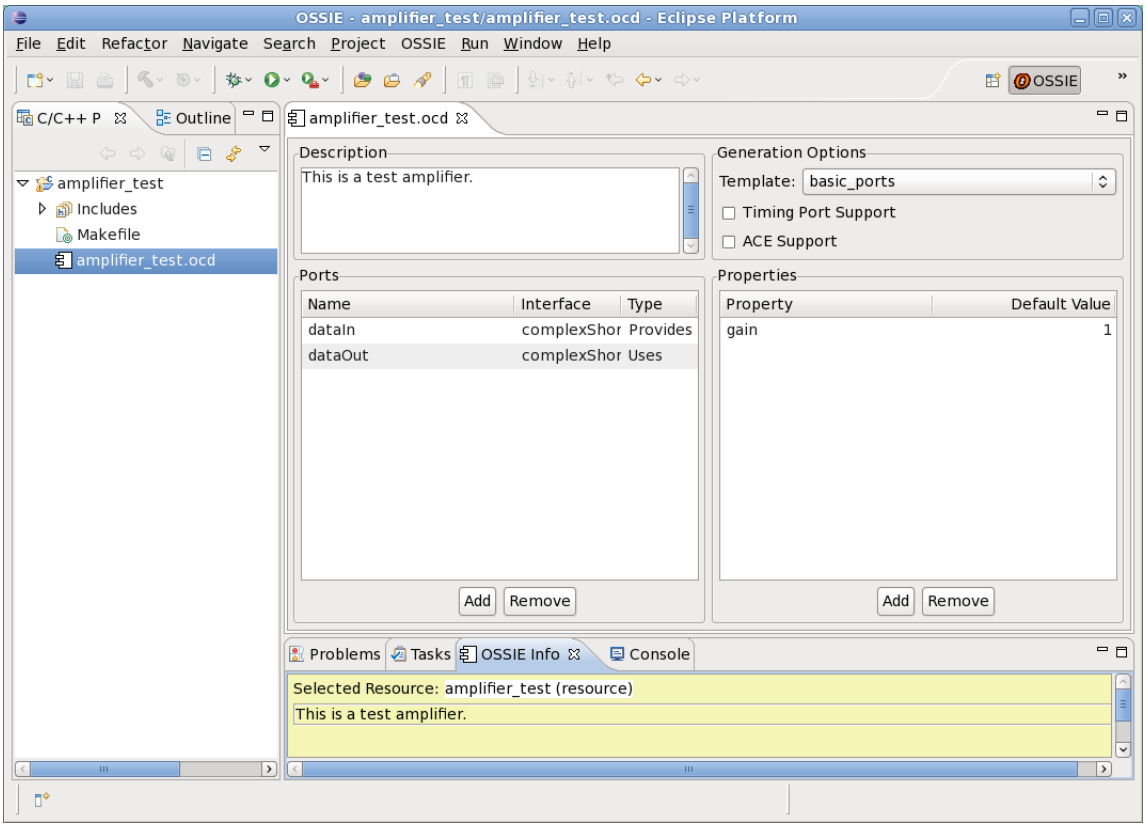
A property, the gain, will be added to the component so it can be set by a user when building a waveform. Add the property by selecting Add in the Properties panel. This will bring up the Property Editor window where various parameters can be set for the property. For this example, enter gain for the Name, "Amplifier gain" for the Description and select Add Property to add it to the component.



Now the default value for the gain must be entered. Click on the clear blank text box under Default value in the Properties panel, and enter 1.



To finish the component, a description must be entered and the source code must be generated. Enter a description, and in the Generation Options panel ensure that basic_ports is selected, and Timing Port Support and ACE Support are both deselected. Before generating the component, OEF should look like the following image:



6.2.3 Generating the Source Code

On the OEF toolbar, select OSSIE : Generate Component. This generates all of the files for the component in the local eclipse workspace directory.

Generating the component will produce multiple files along with the source code.

1. `configure.ac`: This is the script that autoconf uses to generate the configure file. It checks for dependencies such as which compiler to use, as well as the presence of OSSIE libraries.
2. `Makefile.am`: This is the script that automake uses to generate the Makefile. It includes all the source files for the component.
3. `reconf`: This is a script to run the automake tools.
4. `component_name.h`: Component class header file.
5. `component_name.cpp`: Component class implementation definition.
6. `main.cpp`: Contains the mandatory `int main()` definition.
7. `component_name.prf.xml`: Component property file.

8. `component_name.scd.xml`: Software component descriptor.
9. `component_name.spd.xml`: Software package descriptor.
10. `documentation.txt`: File for documenting your component.
11. `Doxyfile`: Doxygen file for generating documentation.

Additional files are generated when using the `pyi_comp` option:

1. `component_name.py`: Python file with port implementations.
2. `WorkModule.py`: Python file where processing is done.
3. `setup.py`: Install script used to copy Python and XML files into the appropriate subdirectories under `/sdr` once the component is edited to provide functionality. This is executed by typing `python setup.py install`.

At this point the signal processing or logical function of the component must be defined. The process for doing so is different depending on whether the component is written in C++ or python.

6.2.4 Editing C++ Components

In this example, by selecting `basic_ports` the component was generated as a C++ component. To edit the component, open `component_name.cpp` and find the `ProcessData` function.

```
$ cd /path/to/OEF/workspace/component_name/
$ nano -w component_name.cpp
```

Within the function, there will be a `while` loop and inside of it will be a line:

```
/*insert code here to do work*/
```

This is where the signal processing function should be implemented. The property that was generated, `gain` will be available in the source code initially as `simplei_0_value`. Reassign this to a new variable named `gain` in the `configure` function in the same file.

6.2.5 Editing Python Components

Generated component template code must be modified to process data. The instructions here apply to a very basic Python component with one *uses* and one *provides* port and no properties. Property values can be used in the processing operations, but this will be the subject of a future exercise.

All data coming into a *provides* port will be loaded into the WorkModule buffer. The data going into this buffer will be loaded into variables called I and Q. See the generated WorkModule.py and look for:

```
    I = new_data[0]
    Q = new_data[1]
```

This implies that if you try to run a component that is getting real data only, Q should be empty, and this may even cause an error. There are comments in various parts of the python files describing how to adjust your code appropriately.

The next two lines initialize two arrays to store your new data:

```
    newI = [0 for x in range(len(I)))]
    newQ = [0 for x in range(len(Q)))]
```

Following these lines are comments with an example of how to process data. If you uncomment the 3-line example, your component will pass the data received by the *provides* (input) port to the *uses* (output) port, assuming you have one *uses* port of type `complexShort` and one *provides* port, also of type `complexShort`. Code can be added to process the data.

The next set of lines following the comment "`# Output the new data`," will send the `newI` and `newQ` vectors to all of your output ports.

6.2.6 Editing the SPD File

You will need to edit `MyComponent.spd.xml` before your Python component will work properly. Find the XML tag below the tag `<code type="Executable">`. By default the next tag is:

```
<localfile name="bin/MyComponent"/>}
```

This needs to be changed to:

```
<localfile name="bin/MyComponent/MyComponent.py"/>
```

6.2.7 Making Sure Files are Executable

After you have installed the component (see below), you will need to make sure that you have permission to execute the Python files. To do this, navigate to `/sdr/bin/MyComponent` and type:

```
$ chmod +x *.py
```

6.2.8 Installing a Component

After the component has been edited, it must be compiled and installed to `/sdr`. To do this, move into the component's workspace directory and run the automake tools.

```
$ cd /path/to/eclipse/workspace/component_name/
$ ./reconf
$ ./configure
```

```
$ make
$ make install
```

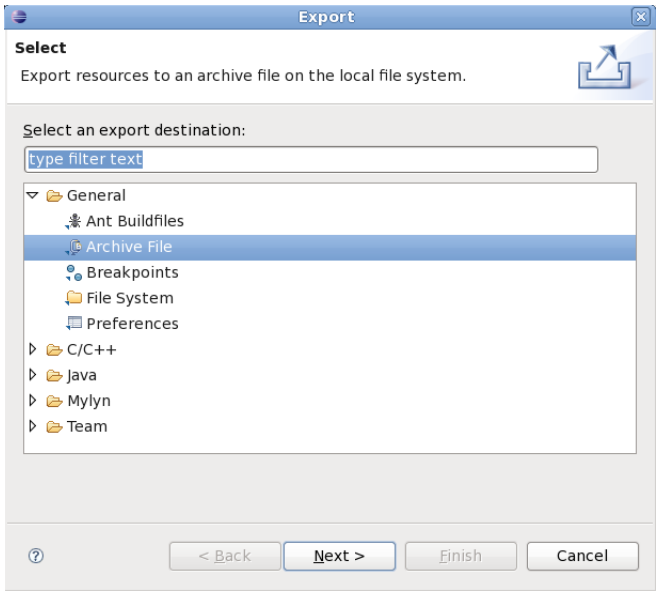
If the commands execute successfully, the component will be installed to `/sdr` and it will be possible to use the component in a waveform. Start a new waveform in OEF to verify that the component has been installed correctly.

6.3 Importing and Exporting Eclipse Projects

Projects developed within Eclipse can be saved to be used as a backup, or to transport to another development platform. This is done by exporting and importing projects.

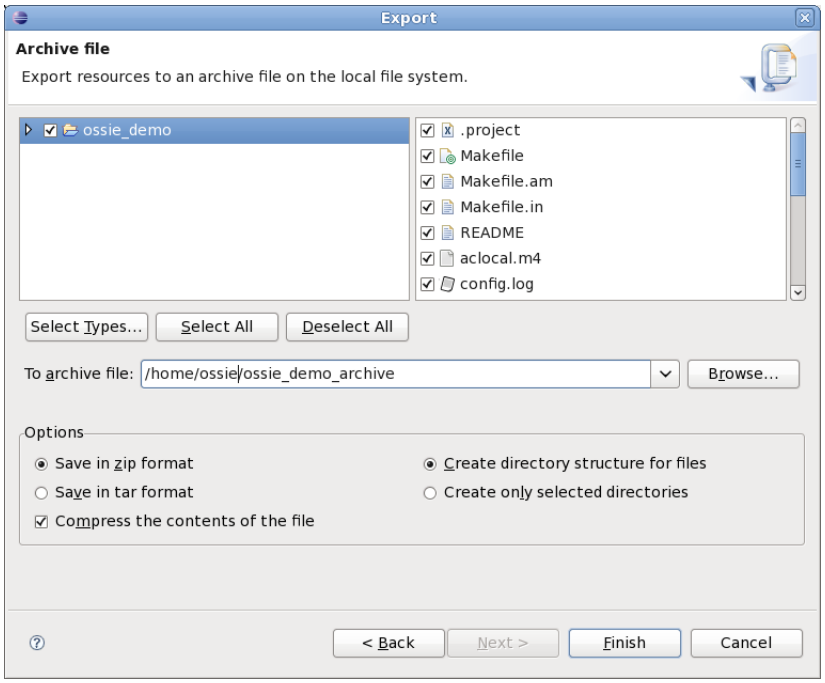
6.3.1 Exporting a Project From Eclipse

To export a project, from the Eclipse toolbar select File : Export. Under the General folder, select Archive File and click Next.



In the next display, in the upper left panel select the boxes next to projects that will be saved. Multiple projects can be saved at one time by selecting more than one project. In the upper right panel, files being saved will have a check mark next to their name. By default, when a project is selected all of the files within the project will be saved. It is recommended to save all of the files however if some files are not needed, they may be deselected by removing the check mark next to their name.

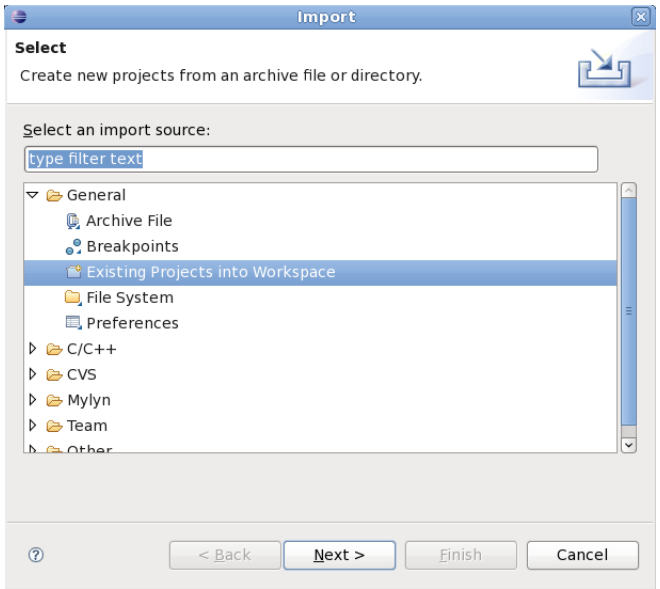
Select Browse to choose a location for the archive to be saved, enter a name for the archive, and select OK. The display should look like the following image:



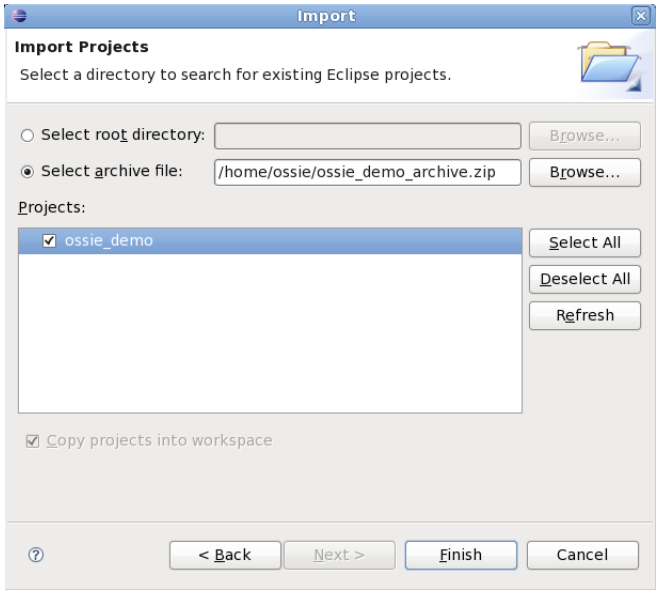
Select Finish to save the archive.

6.3.2 Importing Project in Eclipse

To import a project, from the Eclipse toolbar select File : Import. Under the General folder, select Existing Projects Into Workspace and click Next.



Select the radio button next to Select archive file, and click Browse. Locate the archive and select OK.



The projects saved within the archive file will now be displayed in the Projects pane. To import a project, ensure that the box next to the project name has been selected. Although multiple projects may have been archived, it is not necessary to import all of them. Select Finish to import the project. The project files will now be displayed in the left panel of the Eclipse GUI, and the project has been imported.

6.4 Additional OEF Instruction

For more information on using OEF and other OSSIE tools, please visit the OSSIE website at http://ossie.wireless.vt.edu and select Getting Started : Labs.

7 OSSIE Waveform Developer

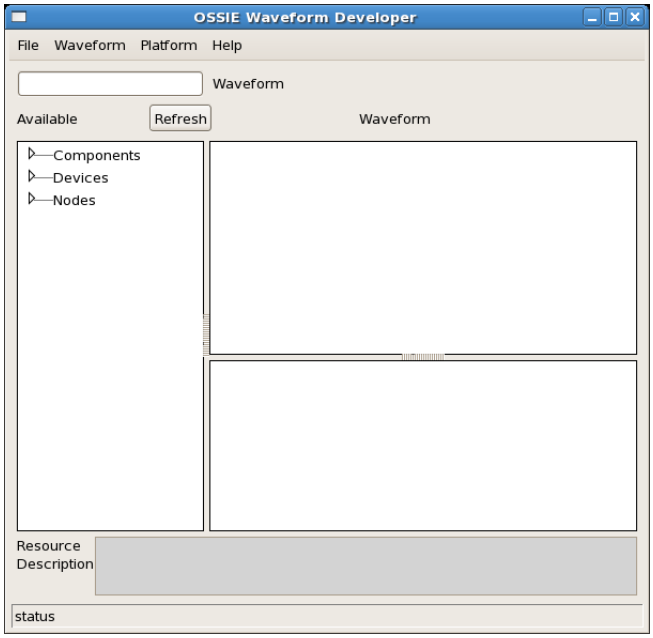
The OSSIE Waveform Developer (OWD) is available for constructing code for waveforms and components based on JTRS's Software Communications Architecture. A user does not need to have detailed knowledge of the SCA or CORBA to use the tool. OWD allows you to draw from a library of available components to construct an SCA-based waveform or create the skeleton code for custom-made C++ or Python components.

7.1 Creating a New Waveform from Existing Components

To start OWD, open a terminal and type:

```
$ OWD
```

The main window should appear without warnings or errors:

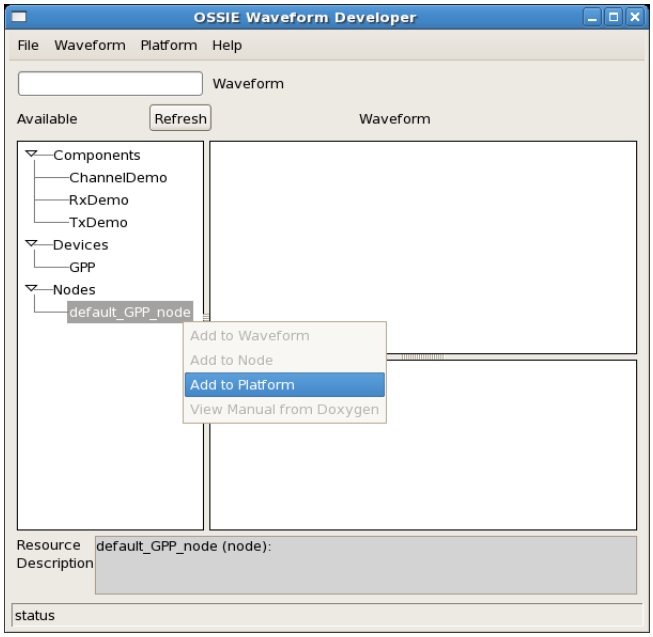


At this time we will re-create the `ossie_demo` waveform packaged with the 0.7.4 release to demonstrate how waveforms can be easily created in OWD.

7.1.1 Adding an Existing Node to a Waveform

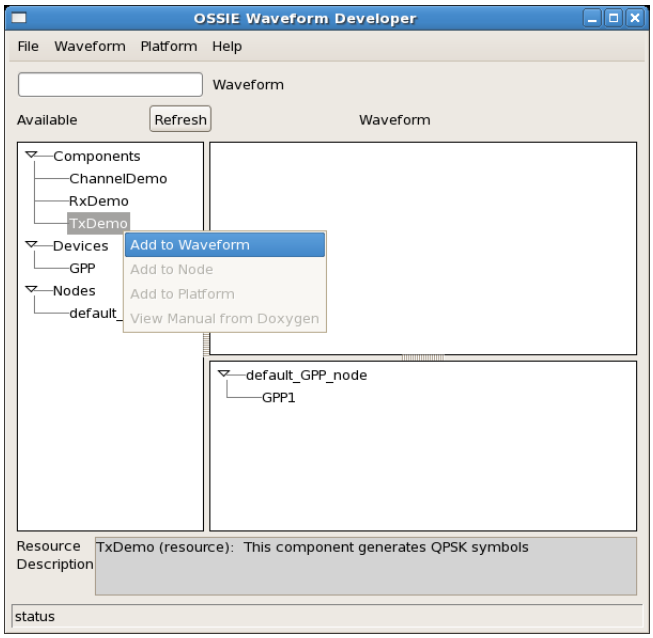
A node is a set of devices upon which components and other resources may be deployed. You can click the arrows next to the components, devices and nodes listed in the pane on the left to reveal the resources installed in your `/sdr` directory.

Under "Nodes," right-click `default_GPP_node` and select "Add to Platform." Click the arrow next to `default_GPP_node` to display all devices associated with the node. In this case, you will see only one device, `GPP1`, which corresponds to the general purpose processor on your computer.

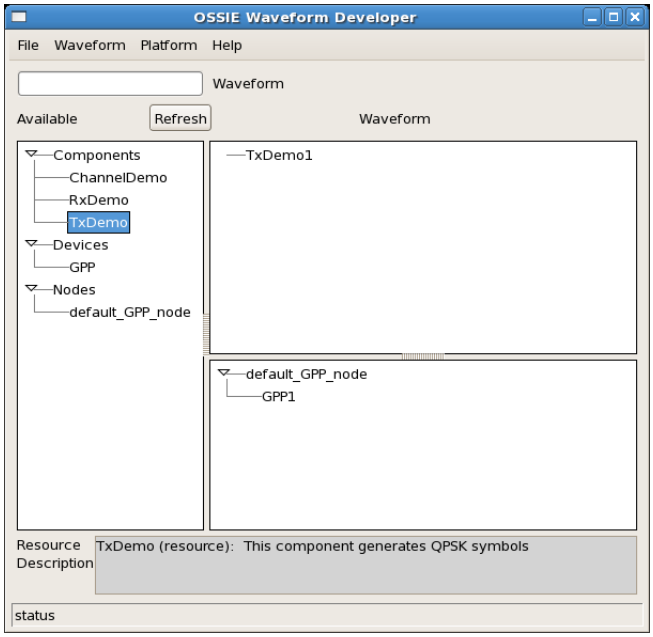


7.1.2 Adding an Existing Component to a Waveform

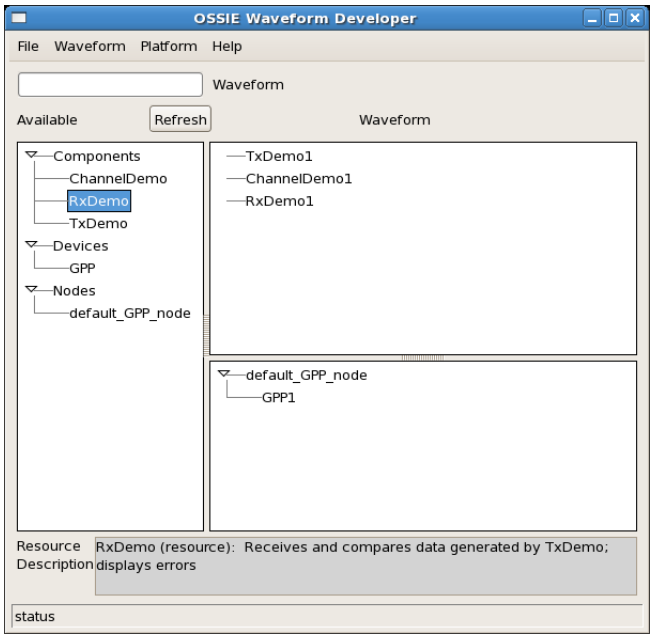
Add an instance of the `TxDemo` component to the waveform by right-clicking and selecting "Add to Waveform."



This will bring up a box requesting an instance name for the component. The default value, "TxDemo1," is usually fine. Select "OK" and you should see an instance called "TxDemo1" on the Waveform pane:



Now add instances of the `ChannelDemo` and the `RxDemo` components to the waveform using the same process as you did for `TxDemo`.



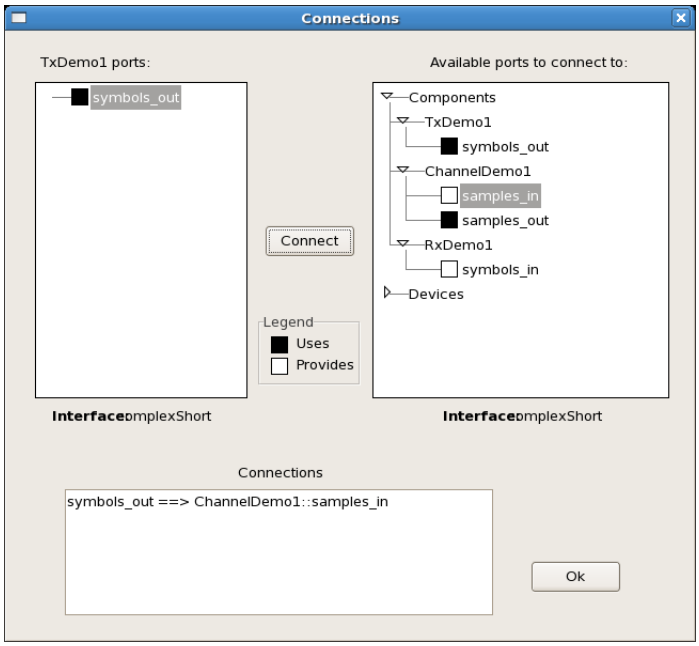
7.1.3 Connecting Components

We are now ready to connect the components. Right-click the "TxDemo1" instance and select "Connect" to open the "Connections" window. The left pane lists all the ports owned by "TxDemo1" which should just include `symbols_out`. The `TxDemo` component itself generates a sequence of QPSK symbols that are pushed by this port.

The right pane lists all the ports associated with the components and devices currently included in the waveform. Click the drop arrow next to "Components." You should see a list of all the component instances and their available ports. As indicated by the legend, a black box indicates a *uses* port while a white box indicates a *provides* port. Generally speaking, a *uses* port is an output interface while a *provides* port is an input interface. There are a few rules to remember when connecting ports:

1. A *provides* port must be connected to a *uses* port, and vice versa
2. Both ports must be of the same interface or data type (e.g. `complexShort`)

From the left pane select the "TxDemo1" *uses* port `symbols_out`. From the right pane, under "Components", select the "ChannelDemo1" *provides* port `samples_in`. Click the "Connect" button in the center of the window to establish the connection. The "Connections" pane should now indicate this connection:

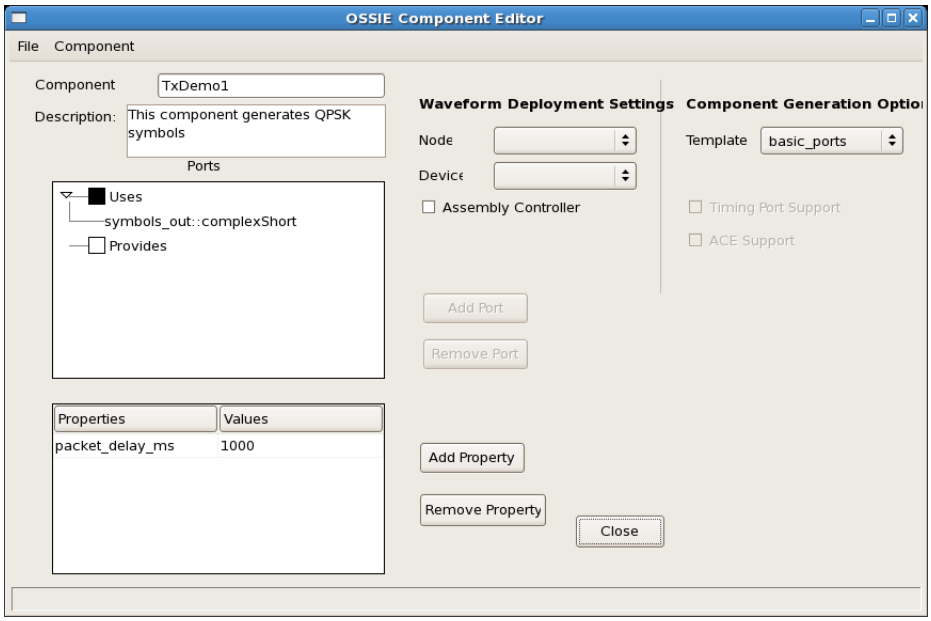


Select "OK" to confirm the connection you specified. This returns you to the main OWD window. You should now see a drop arrow next to the "TxDemo1" component instance. Clicking the arrow reveals all connections made by the component. Repeat these steps to connect the "ChannelDemo1" *uses* port `samples_out` to the "RxDemo1" *provides* port `symbols_in`. Start by right-clicking on the "ChannelDemo1" component and selecting "Connect."

7.1.4 Deploying Components and Editing Component Properties

Each component needs to be deployed to the node and device where it will run. Also, components have properties which are configured during waveform run time. Optionally, for each component instance in a waveform, default property values can be overridden by an XML-format SAD file that describes how the waveform is assembled. This can be done by changing a component's properties within OWD.

To bring up the Component Editor, right-click the "TxDemo1" component instance and select "Edit."

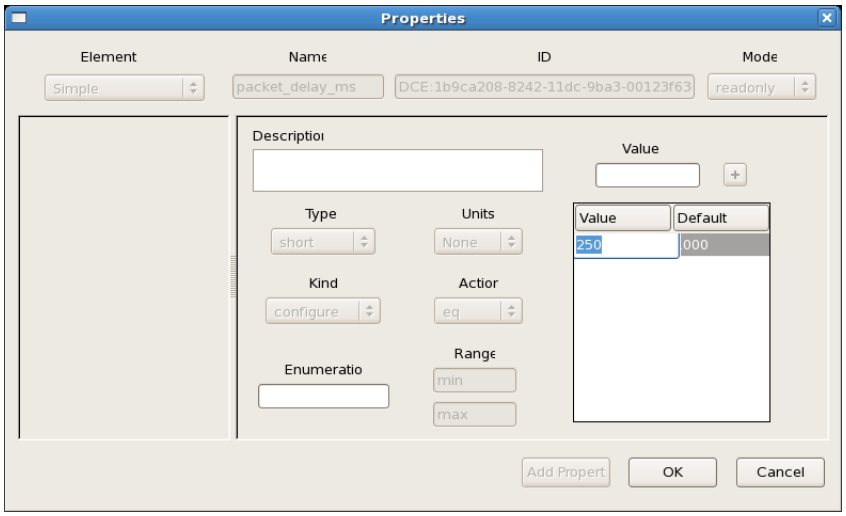


This window contains nearly all the information about the component itself including its ports, properties, and node and device assignments. At the moment the component instance is not assigned to any device. It is necessary to make this assignment for each component in the waveform. For this example, select `default_GPP_node` under the "Nodes" spinbox.[4] Click on the "Device" spinbox and select "GPP1."

Now it is time to set the component properties. For the `TxDemo` example, you should see a *packet_delay_ms* property with a default value of `1000` ms. This particular property is specific to the component and determines how long "TxDemo1" should wait between sending each QPSK packet.

Right-click on the *packet_delay_ms* property and select "Edit." Change the default value from `1000` to `250`:

[4] It is imperative that you actually select the `default_GPP_node` text with the mouse, otherwise you will not see a list of available devices in the "Device" spinbox

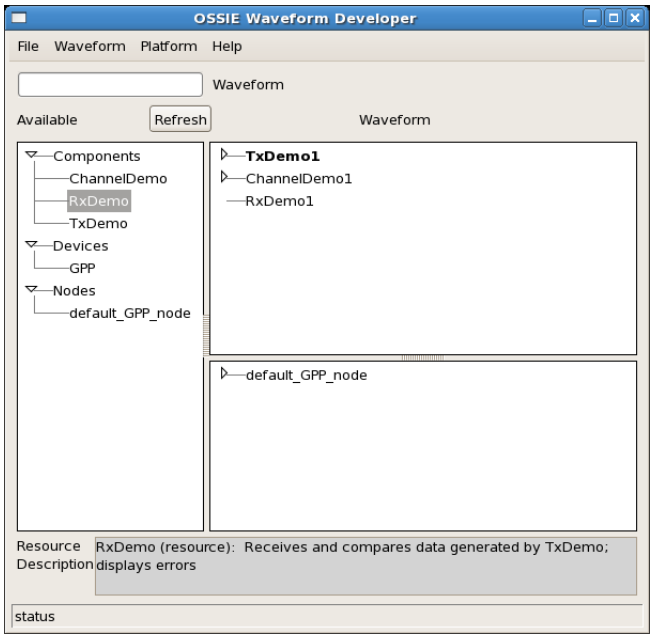


Click "OK" to return the the Component Editor window. Click "Close" to return to OWD's main window.

Repeat the component deployment steps for the "RxDemo1" component. You can see that "ChannelDemo1" has properties, but they can be left unchanged.

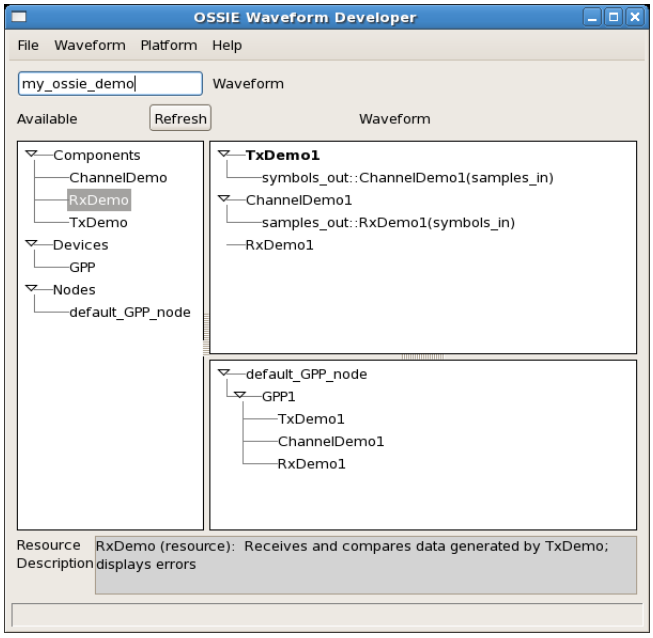
7.1.5 Setting the Assembly Controller

Each waveform must have just one assembly controller. Calling "start" on the waveform actually invokes the `start()` function on the assembly controller component. In this case, the assembly controller is "TxDemo1." Right click the "TxDemo1" component instance and select "Set Assembly Controller." Notice that "TxDemo1" now appears in bold face to indicate its new status:



7.1.6 Generating the Waveform

Before OWD can generate the waveform XML, you need to give the waveform a name. In the upper left textbox type `my_ossie_demo`. If you click on all the drop arrows to display details, your completed waveform should look like this:



Select Waveform→Generate. If everything has worked, you should not get any warnings or error messages. Select the destination for the source (usually something like `~/src/waveforms/`). This can be done by typing the destination path into the text box labeled "Name." Now click "OK" or "Open."

7.1.7 Installing the Waveform

In a terminal, change to the directory in which the source files were generated, and run the included `setup.py` script:

```
$ cd /home/mysername/src/waveforms/my_ossie_demo
$ python setup.py install
```

This should copy the DAS and SAD files to the new directory `/sdr/waveforms/my_ossie_demo`. You can now load and run the waveform using wavLoader (see Section 4) or ALF (Section 8).

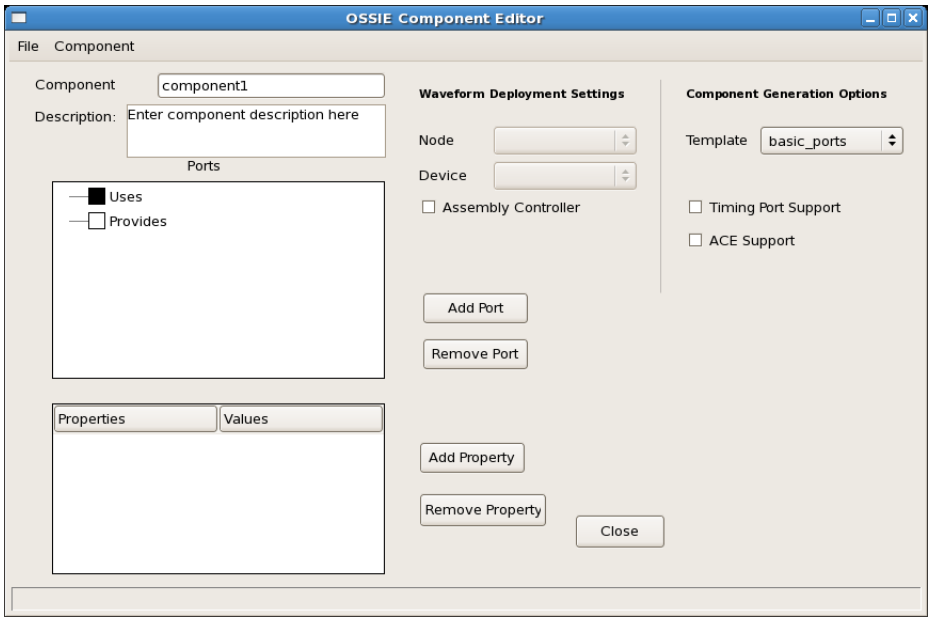
7.2 Creating a New Component

A component is a communications system function that is implemented in software. When modularity is desired, components implement only simple functions. Multiple components are then connected to implement more complex functions. The Component Builder allows the user to develop template code for SCA-based components given a set of properties and ports. In the generated template, a section is created where the user can enter the algorithm.

To start the stand-alone component builder, run the following command from any directory:

```
$ OWDC
```

This should bring up the OSSIE Component Editor window:



There are five major steps to creating an SCA-based component using the OWD Component Editor:

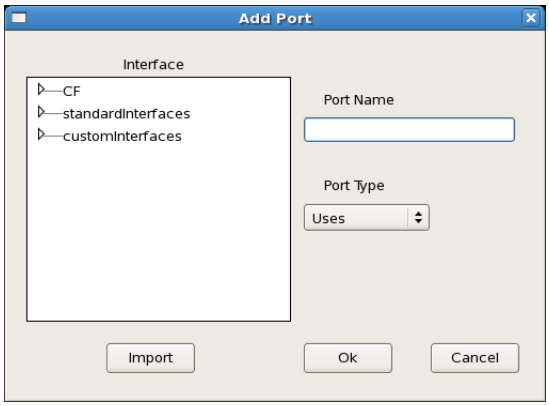
1. Adding ports
2. Adding properties
3. Selecting component generation options
4. Generating the source code
5. Building and installing the binaries

These steps are described in detail in the following sections.

7.2.1 Adding Ports

A port is an interface that allows components to communicate with one another. Generally, ports are used to either pass data or call functions. Release 0.7.4 of OSSIE is packaged with several pre-defined standard interfaces that make development of components simple.

To add one or more ports to your component, click on the "Add Port" button from the OSSIE Component Editor window.

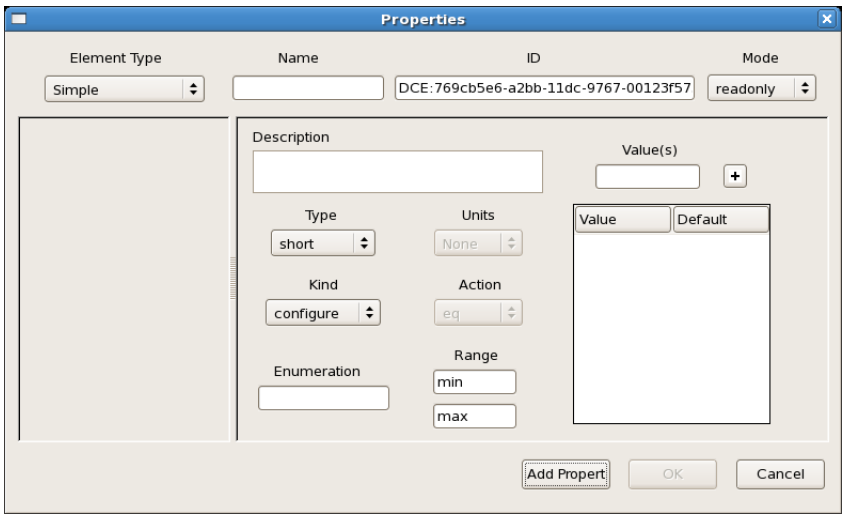


The Add Port dialog box will be opened. You may then select the data type for the component (`complexShort` or `complexFloat` from standardInterfaces is recommended) and its type. Whether or not the port is a *uses* or *provides* port is often a point of confusion for users who are unfamiliar with the SCA. In most simple cases, a *uses* port corresponds to an output interface, while a *provides* port corresponds to an input interface.

Give the port a unique name (e.g. `complex_baseband`) and click "Ok." The new port should now appear in the "Ports" pane of the OSSIE Component Editor window.

7.2.2 Adding Properties

To add one or more properties to your component, click on the bottom "Add Property" button. A properties dialog box will be opened.



A new, unique ID for the property will automatically be generated. The element types "Simple" and "SimpleSequence" are completely supported. Default property values can be added to the

property. Once added, the default value cannot be changed, but the value stored in the "Value" column can be changed by clicking on it. Once all of the configurations are set, click on "Add Property" to add the property to the component. After the property has been generated, the instance values of the property can be changed in the Component Editor.

7.2.3 Selecting Component Generation Options

Component generation options include templates (basic_ports, custom_ports, and py_comp) and optional timing and ACE support. When the basic_ports option is selected, the C++ source code files generated will include port implementations. The custom_ports option generates C++ code with ports implemented in separate files from the functional component code. The custom_ports option allows support for timing but documentation for custom_ports is limited to comments in the source code. The py_comp option generates Python code, suitable for components that do not need to process data at high rates. The py_comp option allows for timing support, which is selected by checking a box below the template selection. The user must edit the generated source code to add functionality to the component. Computer science laboratory format exercises are under development that will include step by step instructions for building simple components using the basic_ports and py_comp options, and for using the timing support feature. The exercises and updates to this guide will be posted at http://ossie.wireless.vt.edu.

7.2.4 Generating the Source

Before generating the source code for the component, you must first give it a name. Type whatever name you would like into the "Component Name" box in the Component Editor. Select Component→Generate Component to select a directory to save the source code that will be generated. The location of the stored source code is arbitrary and many users will create a components directory to store all of their source code. Once you select a directory and click "Open" or "Ok," your code will automatically be generated.

Files Generated Using the basic_ports C++ Option

configure.ac This is the script that `autoconf` [1] uses to generate the `configure` file. It checks for dependencies such as which compiler to use, as well as the presence of OSSIE libraries.

Makefile.am This is the script that `automake` [2] uses to generate the `Makefile`. It includes all the source files for the component.

reconf This is a script to run the automake tools.

MyComponent.h Component class header file

MyComponent.cpp Component class implementation definition

main.cpp Contains the mandatory `int main()` definition

MyComponent.prf.xml Component property file

MyComponent.scd.xml Software component descriptor

MyComponent.spd.xml Software package descriptor

documentation.txt File for documenting your component

Doxyfile Doxygen [3] file for generating documentation

Files Generated Using py_comp Option

MyComponent.py Python file with port implementations

WorkModule.py Python file where processing is done

setup.py Install script used to copy Python and XML files into the appropriate subdirectories under `/sdr` once the component is edited to provide functionality. This is executed by typing `python setup.py install`

MyComponent.prf.xml, MyComponent.scd.xml, MyComponent.spd.xml Same as for C++ components

7.2.5 Building a Working Python Component

Generated component template code must be modified to process data. The instructions here apply to a very basic Python component with one *uses* and one *provides* port and no properties. Property values can be used in the processing operations, but this will be the subject of a future exercise.

All data coming into a *provides* port will be loaded into the WorkModule buffer. The data going into this buffer will be loaded into variables called I and Q. See the generated WorkModule.py and look for:

```
I = new_data[0]
Q = new_data[1]
```

This implies that if you try to run a component that is getting real data only, Q should be empty, and this may even cause an error. There are comments in various parts of the python files describing how to adjust your code appropriately.

The next two lines initialize two arrays to store your new data:

```
newI = [0 for x in range(len(I)))]
newQ = [0 for x in range(len(Q)))]
```

Following these lines are comments with an example of how to process data. If you uncomment the 3-line example, your component will pass the data received by the *provides* (input) port to the

uses (output) port, assuming you have one *uses* port of type `complexShort` and one *provides* port, also of type `complexShort`. Code can be added to process the data.

The next set of lines following the comment "`# Output the new data`," will send the `newI` and `newQ` vectors to all of your output ports.

7.2.6 Editing the SPD File

You will need to edit `MyComponent.spd.xml` before your Python component will work properly. Find the XML tag below the tag `<code type="Executable">`. By default the next tag is:

```
<localfile name="bin/MyComponent"/>}
```

This needs to be changed to:

```
<localfile name="bin/MyComponent/MyComponent.py"/>
```

7.2.7 Making Sure Files are Executable

After you have installed the component (see below), you will need to make sure that you have permission to execute the Python files. To do this, navigate to `/sdr/bin/MyComponent` and type:

```
$ chmod +x *.py
```

C++ components

For C++ components, within the generated `your_component_name.cpp` file you may enter the code for processing data. Look for `/*insert code here to do work*/` within the process_data function. Editing the SPD file should not be necessary. The installed executable file, `/sdr/bin/your_component_name`, needs to be executable, but this should be done by default.

7.2.8 Building and Installing the Binaries

Once the component has been generated, use a terminal to navigate to the directory of the created source code (e.g. `src/components`). Four commands must be executed (in order) in order to install a C++ component to the `/sdr` directory:

```
$ ./reconf
$ ./configure
$ make
$ make install
```

If you do not have ownership of your target install directory, (by default `/sdr`), you will need to run `make install` as root.

To install a Python component, from the directory that contains the generated and edited source code, simply execute the following command:

```
$ python setup.py install
```

Just as with the C++ installation, if you do not have ownership of your target install directory you will need to run `python setup.py install` with root privileges.

7.3 Custom License Generation

OWD allows for customization of the license headers that proceed component or waveform source files. By changing some simple parameters, these licenses can be tailored to a specific need.

The configuration file that must be modified to customize the header is `tools/WaveDev/wavedev.cfg`. There are multiple XML tags within this file, but the three that need to be considered are `sourcepreamble`, `licensefile`, and `developer`.

The user will need to configure this header, as a meaningful default header will not be used. By default, the license header will display as:

```
Copyright $YEAR by __DEVELOPER__, all rights reserved.
```

The `developer` option is simply the name of the person(s), company, or organization to whom the file should be copywritten. The `sourcepreamble` option should contain an absolute path to a file (readable by the user) that contains the preamble to be placed at the top of all source files. The `sourcepreamble` file is parsed as follows:

__COMP_NAME__ is replaced with the name of the component or waveform, __YEAR__ is replaced with the current year, and __DEVELOPER__ is replaced with the `developer` value from `wavedev.cfg` file. Finally, `licensefile` is an absolute path to a full license file (e.g. the GPL) that will accompany the generated component or waveform.

7.4 Removing Components, Devices, Nodes and Waveforms

7.4.1 Removal Notes

The files that need to be removed all exist within the `/sdr` directory. To remove a component or device, the executable must be removed from `/sdr/bin/` as well as the xml files in `/sdr/xml/`. To remove a node the xml files must be removed from `/sdr/nodes/`, and to remove a waveform the xml files must be removed from `/sdr/waveforms/`. The following steps will walk through the procedure required to uninstall a component or device, a node and a waveform.

7.4.2 Component and Device Removal

Files in both `/sdr/bin/` and `/sdr/xml/` must be deleted to uninstall a component or a device. Move into the `/sdr/bin` directory.

```
$ cd /sdr/bin
```

Now delete the executable. The file is write protected, so the `-f` option will not ask you to verify that you want to delete the file.

```
$ rm -f COMPONENT_OR_DEVICE_NAME
```

Now move into the `xml/` directory.

```
$ cd ../xml/
```

Remove the directory containing all of the xml files. This command must be executed as root, however exit out of root after the folder has been removed.

```
# rm -rf COMPONENT_OR_DEVICE_NAME
# exit
```

The component has now been removed. Open OWD to verify the component is no longer available.

```
$ OWD
```

Within OWD, click the arrow next to `Components` if you are removing a component, or `Devices` if you are removing a device, in the left pane. Verify that the component is no longer listed. If OWD was running while the component or device was removed, click the `Refresh` button above the component list to refresh the list of available components and devices.

7.4.3 Node Removal

To remove a node only a single directory within `/sdr/nodes/` needs to be removed. Move into the `nodes/` directory.

```
$ cd /sdr/nodes/
```

Remove the directory containing the xml files. This must be done as root, however exit out of root after entering the command.

```
# rm -rf NODE_NAME
# exit
```

The node has now been removed. Open OWD to verify the node is no longer available.

```
$ OWD
```

Within OWD, click the arrow next to `Nodes` in the left pane. Verify that the node is no longer listed. If OWD was running while the node was removed, click the `Refresh` button above the component list to refresh the list of available nodes.

7.4.4 Waveform Removal

To remove a waveform, only a single directory within `/sdr/waveforms/` needs to be removed.

To remove a waveform, first move into the `waveforms/` directory.

```
$ cd /sdr/waveforms/
```

Remove the directory containing the xml files. This should not be done as root.

```
$ rm -rf WAVEFORM_NAME
```

The waveform has now been removed.

8 ALF Graphical Debugging Environment

Our waveform application visualization and debugging environment, ALF, provides a graphical user interface for waveform management and debugging. Not only is it useful for installing and starting waveforms, but it also provides a selection of debugging tools. When a waveform is running, the developer is able to monitor the throughput of components, plot data in real time, and input or output data to components in a waveform. ALF also gives the user the ability to run components that are not part of a host waveform in the framework.

8.1 Running ALF

To start ALF, nodeBooter must be started as usual. In one tab or terminal type:

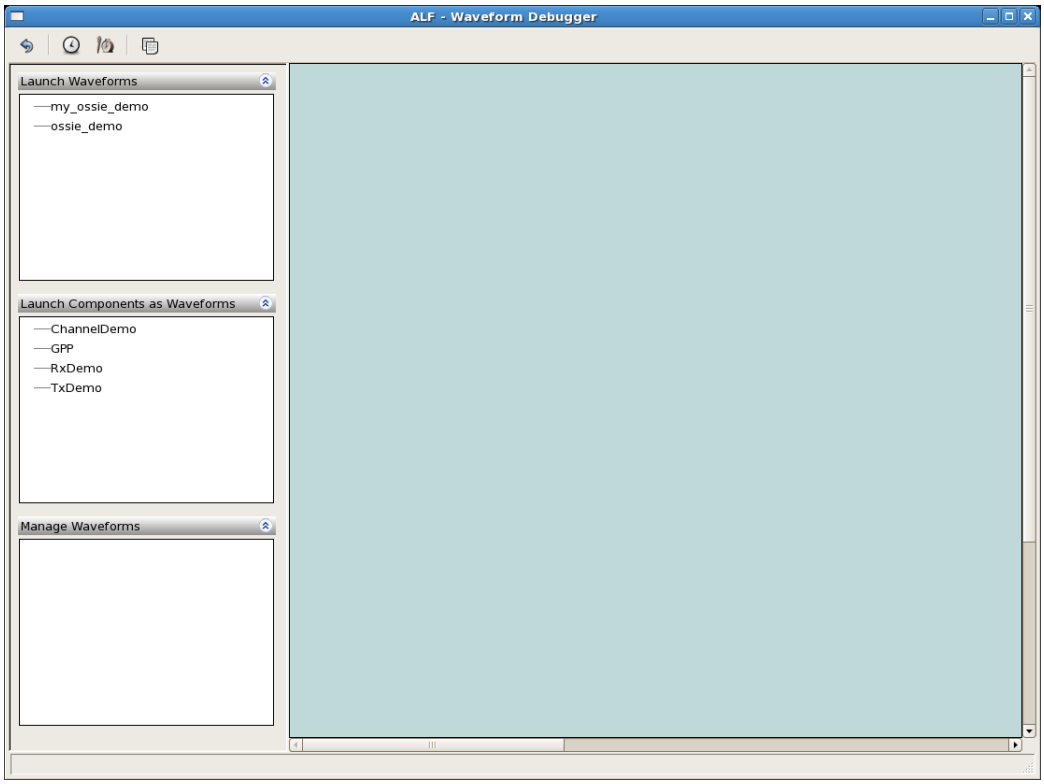
```
$ nodeBooter -D -d nodes/default_GPP_node/DeviceManager.dcd.xml
```

`default_GPP_node` can be changed if your waveform requires it.

Once `nodeBooter` is running, open a second tab or terminal and type:

```
$ ALF
```

This should open the ALF main window without errors or warnings. You will be presented with a dialog box to select the IP address hosting the NameService. If you are running the NameService locally (i.e. 127.0.0.1), then select 'OK' and continue using ALF. You can also point ALF to a different IP address that is hosting the NameService, for example, an embedded platform.



8.2 Running Waveforms

The top left pane, Launch Waveform Applications, displays a list of installed waveforms. To install and run an instance of the waveform, simply double-click on the waveform name.[5] The terminal in which `nodeBooter` was started should display the output of the OSSIE frameworks ApplicationFactory as it installs the waveform. You may easily run multiple waveforms, or multiple instances of the same waveform in this manner.

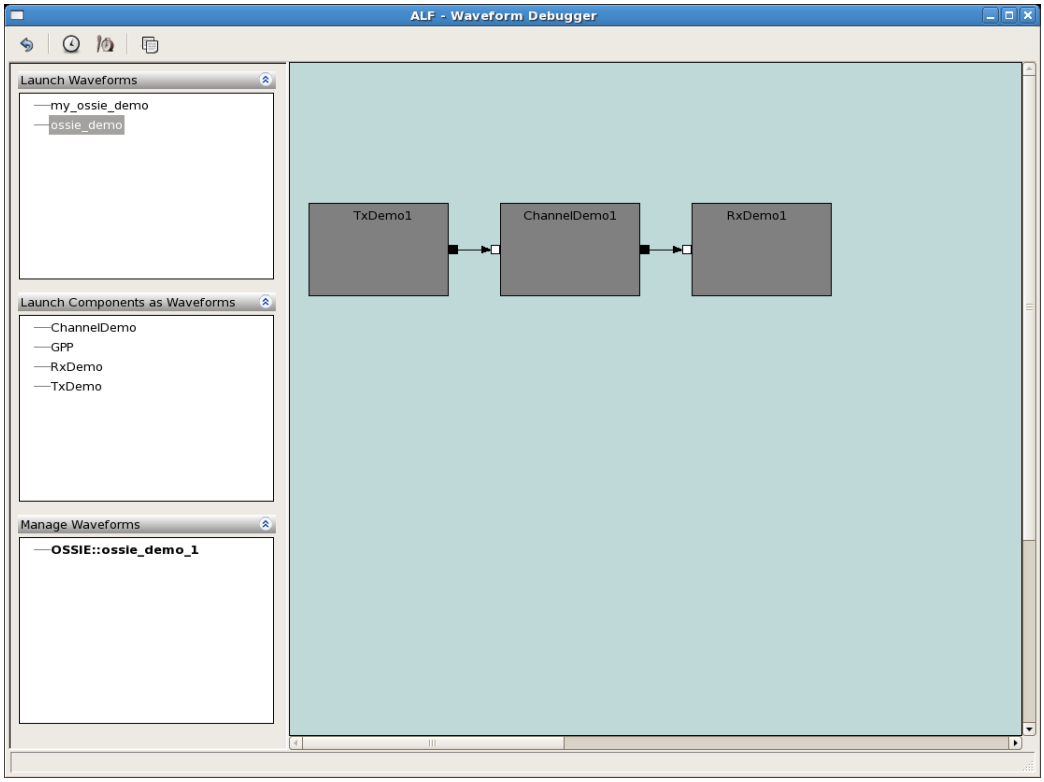
It is also possible to install a waveform and start it manually. In the Launch Waveform Applications window, right click on a waveform to select Install or Install and Start. If Install is selected, the waveform will be installed and will be visible in the Manage Applications window. To start the installed waveform, right click and select start. To stop the waveform, right click on stop.

8.3 Managing Running Waveforms

The Manage Applications pane in the lower-left corner of the main ALF window displays a lists of all instances of running waveforms. Try running the `ossie_demo` waveform using the instructions

[5] Note that for a waveform to run, you must have started the correct node for that particular waveform in nodeBooter. For a resource to run, you must have started one of the default nodes in `nodeBooter`

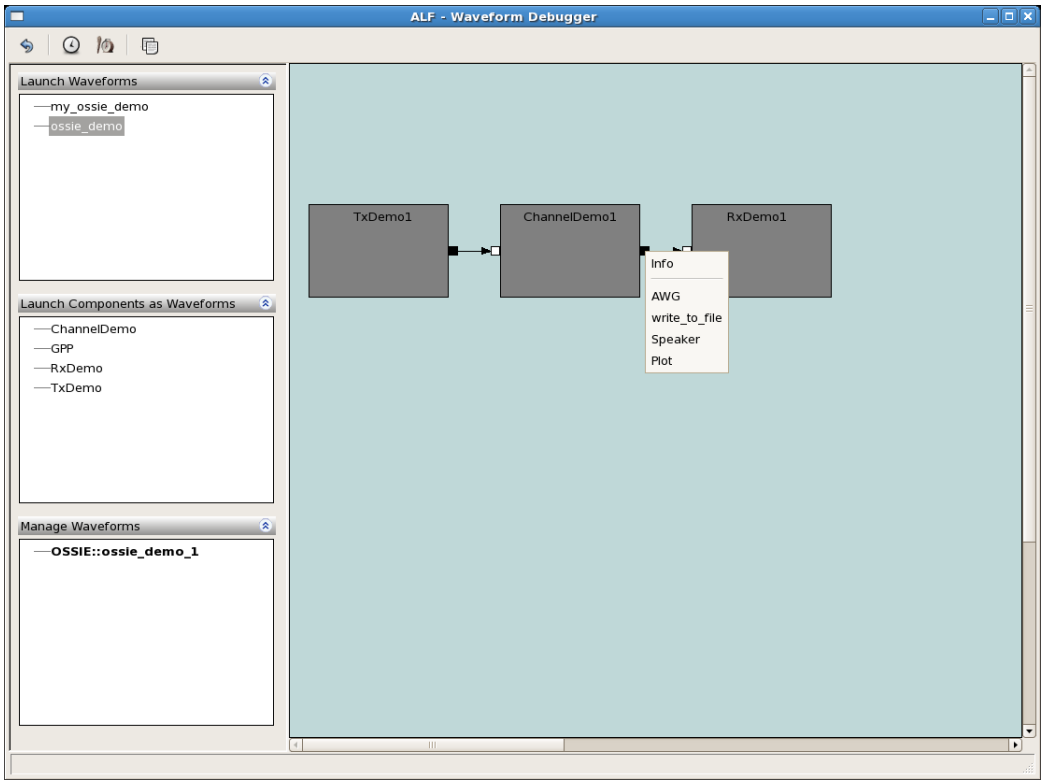
above. You should see –OSSIE::ossie_demo_1 listed in the Manage Waveforms pane. Double-click on –OSSIE::ossie_demo_1 to display the waveform's components in the main window:



Notice that –OSSIE::ossie_demo_1 is now in bold face to indicate that this waveform instance is the one currently being displayed.

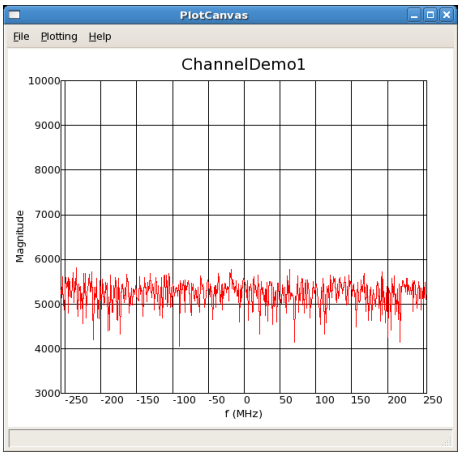
8.4 ALF Tools

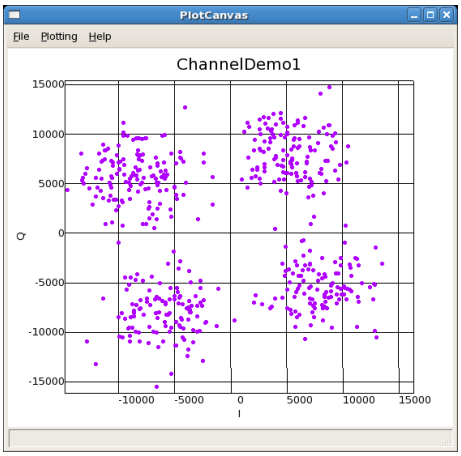
ALF uses a set of tools (or plugins) to assist in waveform and component development. To see a list of available tools, display a waveform and right-click on one of the ports:



8.4.1 The Plot Tool

Right click on an existing *uses* port on the graphical interface and select "Plot" in order to plot data. The data must be either of type `complexShort` or type `complexFloat` for this tool.





8.4.2 The write_to_file Tool

In order to write data to a file, right click on an existing *uses* port and select "write_to_file." In the write to file window you may specify where the data will be written. Click on the "Write Packet" button to write the last incoming packet to file.

8.4.3 The Arbitrary Waveform Generator

To activate the Arbitrary Waveform Generator (AWG), click on an existing provides port and select "AWG." Either select a pre-existing signal type from the drop down menu (e.g., "random"), or select "File" in order to specify a file to read data from. The data will be sent when the "Push Packet" button is pressed. The data being sent will be of type `complexShort`.

8.4.4 The Speaker Tool

To activate the speaker tool, click on an existing *uses* port and select "speaker." Audio should start playing automatically. The input to this tool must be of type `complexShort`.

8.4.5 The Timing Tool

The main toolbar of the ALF display contains buttons that allow you to toggle the timing displays. In order to view the throughput of a port, right click on the desired port and select "Get Info." Note that the port must support timing. This feature can be added when creating the component. Timing support is available only for the custom_ports and py_comp component generation options. Currently no documentation exists for the custom_ports option beyond the comments in the generated source code, so it is recommended for advanced developers only.

8.4.6 The Connect Tool

On the main toolbar of the ALF display, to the right of the timing buttons, there is a button to start the connect tool. To connect two ports, specify the *uses* port on the left half of the connect tool display and specify the *provides* port on the right side of the connect tool display. To specify a port, select the appropriate waveform from the first drop-down menu. Once the waveform is selected, the second drop-down menu will allow you to select a component. Once the component is selected, select the name of the port you wish to connect in the drop-down men below the component selected. Once both the *uses* and *provides* port have been selected, press the "Connect" button.

8.4.7 The Automation Tool

The Automation Tool is a feature which allows aggregate waveforms to be built. Through the use of XML files, additional components can be added or removed while the waveform is running. To start the tool, select the "Connect Tool" button on the main toolbar within ALF. This will display the Connect Tool window, which at the bottom has a path to an automation file and a "Load Automation File" button. Double click on a waveform in the Launch Waveform Applications panel in the main ALF window to start the waveform. Browse to the XML file and select" Load Automation File." For example, to run the `pass_data_waveform`, browse to `/usr/lib/python2.5/site-packages/alf/config/example1.xml`, then select "Load Automation File."

This will bring up the Arbitrary Waveform Generator window. The AWG allows components to be added to the waveform through the Packet Headers box, and determines which kind of data will be sent to the components. In this example, the waveform has two consumers, labeled 1 and 2. These are components which can be installed and uninstalled through setting the appropriate packet headers. To send a packet to a consumer enter its number into the Packet Headers box, separated by commas. For example, to send two packets to consumer 1, and one to consumer 2, enter the following:

```
1, 1, 2
```

To uninstall a consumer, enter a negative sign before the consumer number. For example, to send three packets to consumer 1 and then uninstall it, enter:

```
1, 1, 1, -1
```

By selecting Push Packet at the bottom, a single packet will be pushed to the corresponding consumer listed in the Packet Headers box. Each time the button is selected, it will advance to the next consumer listed and when it reaches the end of the list it will loop back to the beginning. Selecting the Start button sends the sequence continually until the Stop button is selected.

8.4.8 Automation XML File

The Automation Tool determines which components are available to be installed. The XML file lists producers and consumers, which provide and consume packets, respectively. The XML file also defines the names of the producers and consumers, what their number designation is, when they are installed, which waveform they are assigned to, the name of the component and the input or output port. For example, in `example1.xml`:

```
<consumer>
    <name>My first consumer</name>
    <header>1</header>
    <install_at_startup>True</install_at_startup>
    <waveform>pass_data_waveform</waveform>
    <componentInstance>pass_data</componentInstance>
```

```
    <port>cshort_in</port>
```

The name of the consumer is “My first consumer” and it is designated 1, which means to send a packet to the consumer using the AWG the number 1 must be entered in the Packet Headers text box. The consumer is set to be installed at start up, denoted by the install_at_startup tag. If this was set as False, the consumer would only be installed when the first packet is pushed to it. The component is `pass_data`, and the input port is defined as `cshort_in`.

9 WaveDash

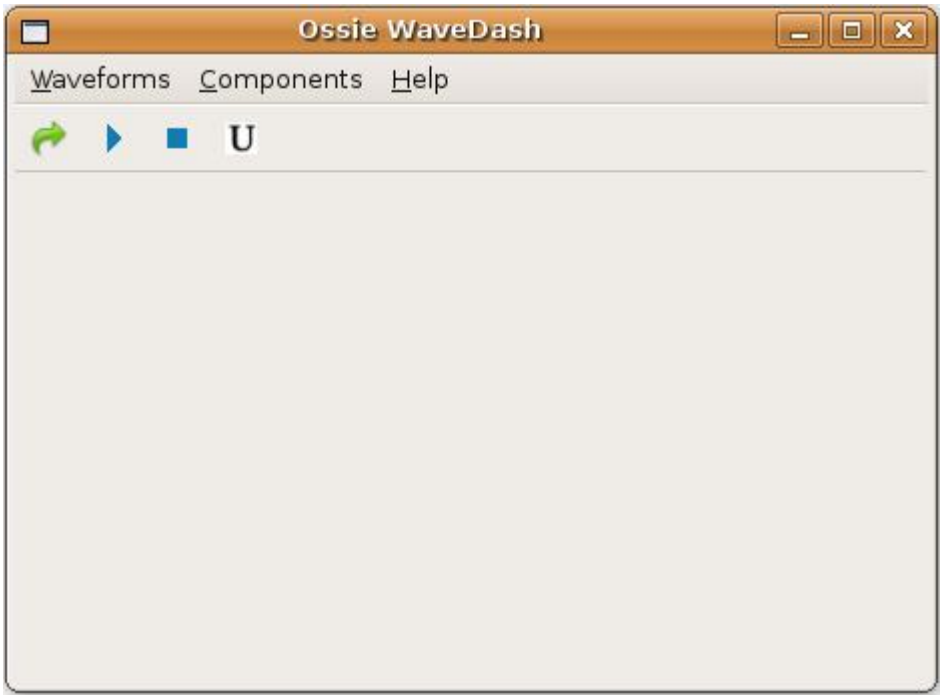
Wavedash is an interactive configurable GUI used to work with OSSIE SDR waveforms. Through this tool, users can install/uninstall or start/stop waveforms and configure the component properties at run time. This eliminates the need to restart or rebuild the waveform every time when component properties need to be changed. Users can also customize the GUI to view only the component and properties of their interest. Further, it also allows the users to change the widget type of a property that makes the configuration process more interactive.

9.1 Running Wavedash

Wavedash is a part of OSSIE Tools and hence the tools setup should install this application as well. Make sure that the naming service and nodeBooter are running before starting Wavedash . To run Wavedash, open a terminal and type

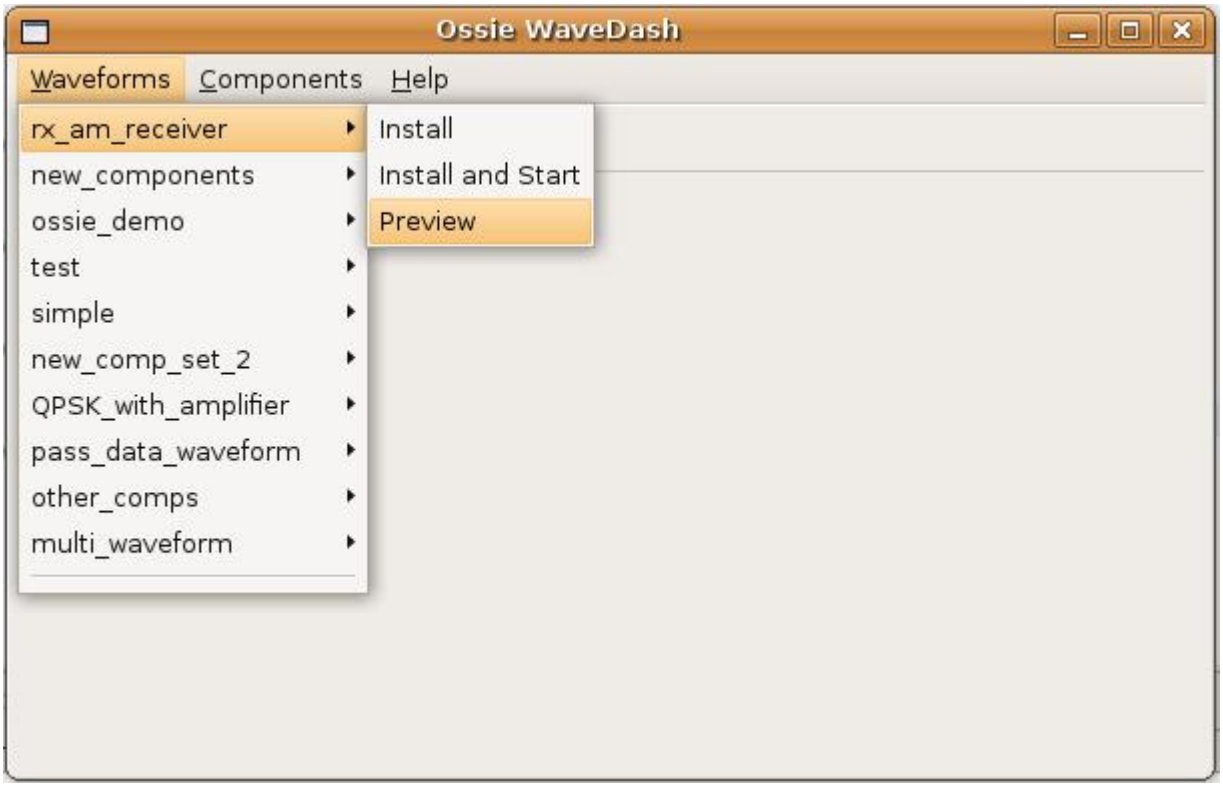
```
$ WAVEDASH
```

This should open the Wavedash initial window without any errors.

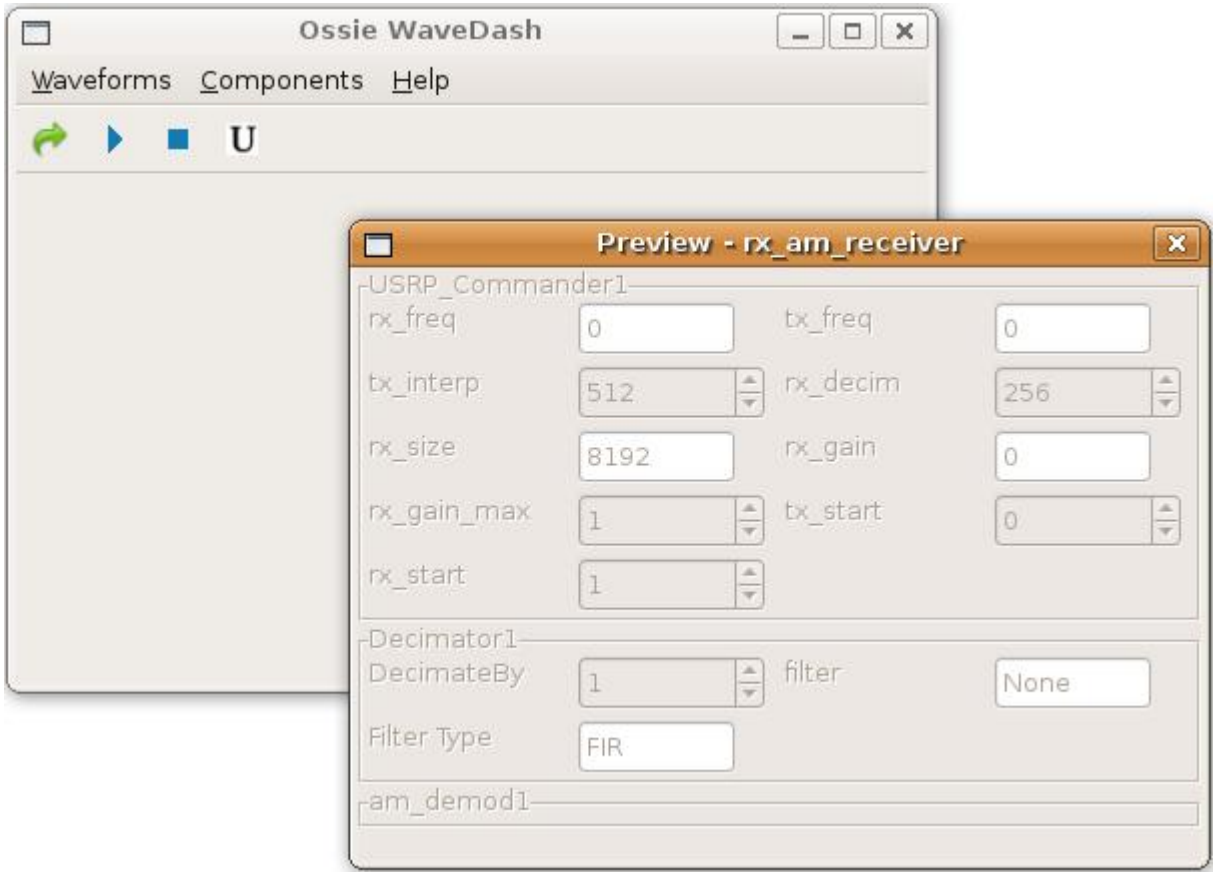


9.2 Installing and Un-installing waveforms

The Waveforms menu in the menu bar will lists the SDR waveforms available in the /sdr/waveforms directory. This menu displays both the waveforms in /sd/waveforms directory and the waveform applications (if any) running in the nodeBooter. These two types of waveforms are separated by a line and the application can show only one waveform at a time.



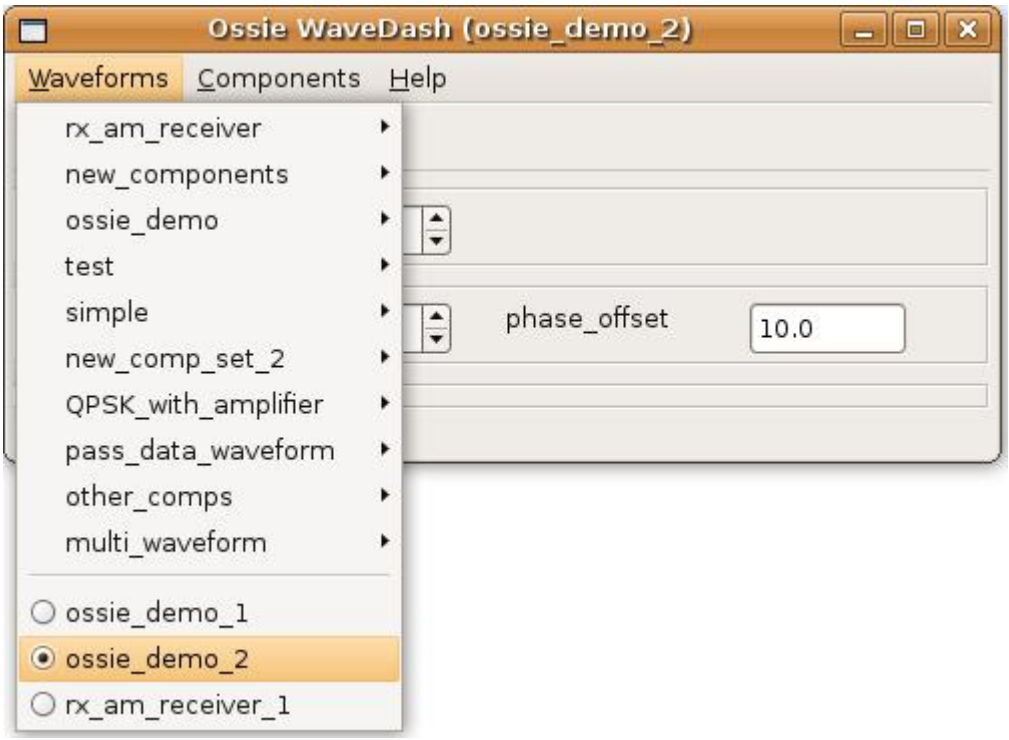
To install a waveform, pull down the Waveform menu and highlight the waveform to be installed. This should open a submenu where you can choose to just Install the waveform or install and start the waveform. You can also preview the waveform to know the waveforms components and their properties.



Once a waveform is installed, it can be started, stopped or uninstalled by using the appropriate buttons in the tool bar.

9.3 Selecting a Waveform

Pull down the waveform and choose one of the installed waveforms.



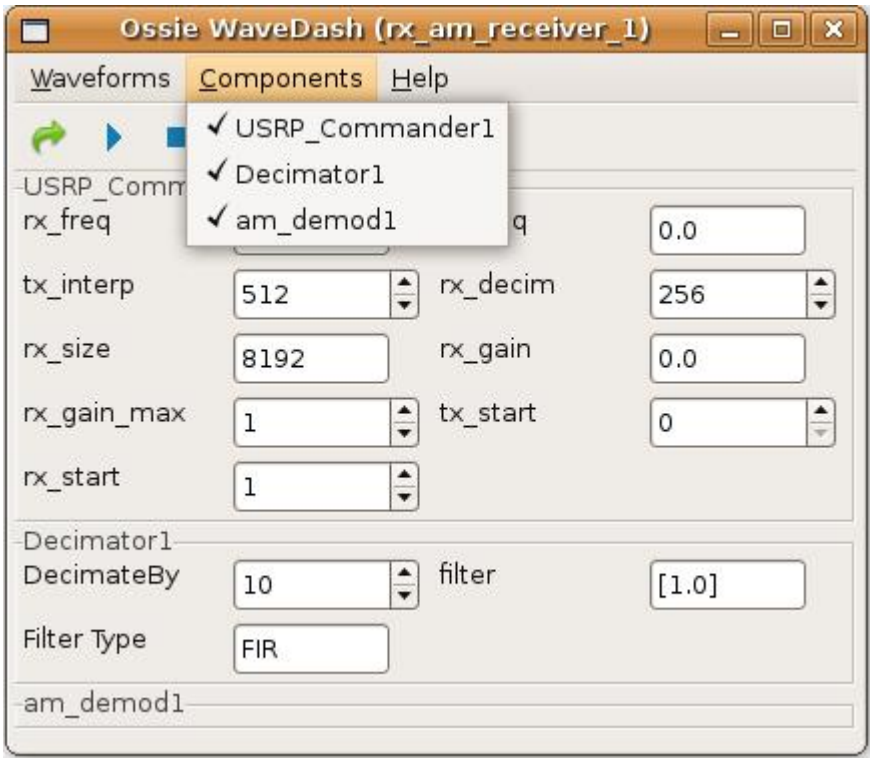
9.4 Refresh

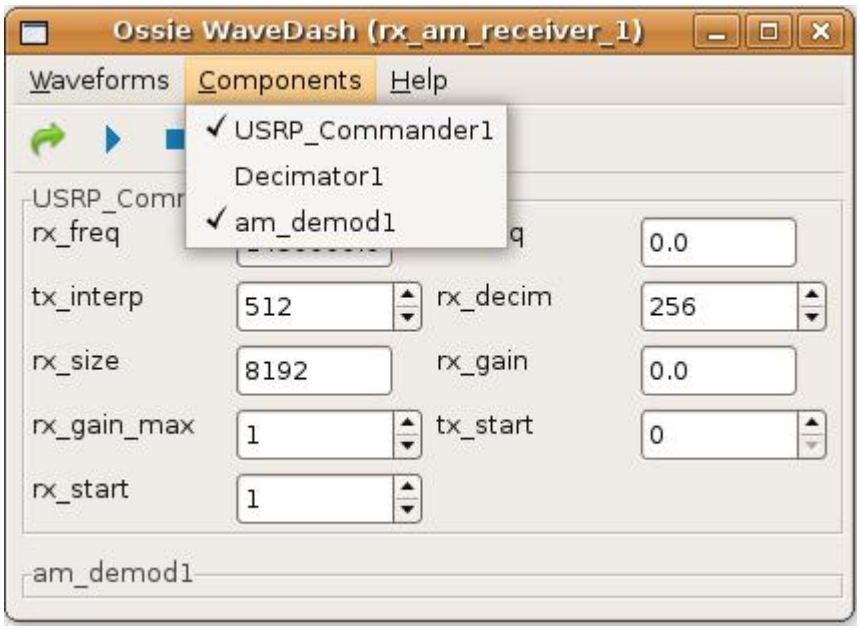
Since waveforms can be installed or uninstalled from ALF also, Wavedash can be refreshed to sync with the current state of waveforms running in the nodeBooter. Click on the Refresh button in the toolbar to refresh.

9.5 GUI Customization

9.5.1 Show or Hide Components

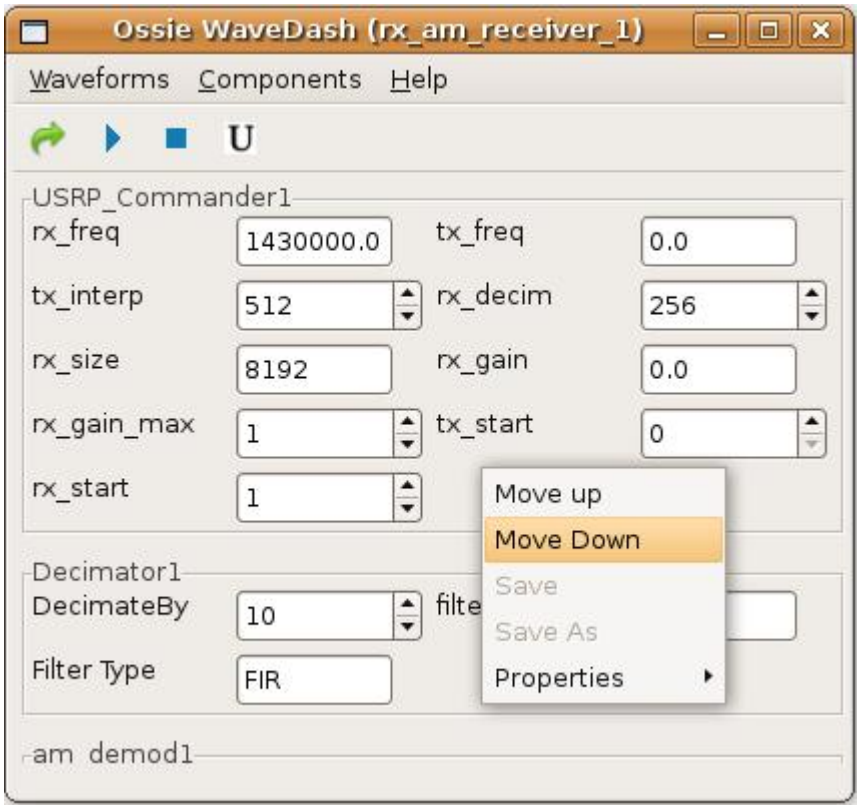
Waveforms may contain different components of which only few may be used frequently. In such case, users can select only the components they wish to see in the application. The Components menu is updated whenever a waveform is selection. Pull down the components menu and select/deselect the components that you wish to show/hide.


Ossie WaveDash (rx_am_receiver_1)
Waveforms
Components
Help
USRP_Commander1
Decimator1
am_demod1
USRP_Comm
rx_freq
0.0
tx_interp
512
rx_decim
256
rx_size
8192
rx_gain
0.0
rx_gain_max
1
tx_start
0
rx_start
1
Decimator1
DecimateBy
10
filter
[1.0]
Filter Type
FIR
am_demod1


Ossie WaveDash (rx_am_receiver_1)
Waveforms
Components
Help
USRP_Commander1
Decimator1
am_demod1
USRP_Comm
rx_freq
0.0
tx_interp
512
rx_decim
256
rx_size
8192
rx_gain
0.0
rx_gain_max
1
tx_start
0
rx_start
1
am_demod1

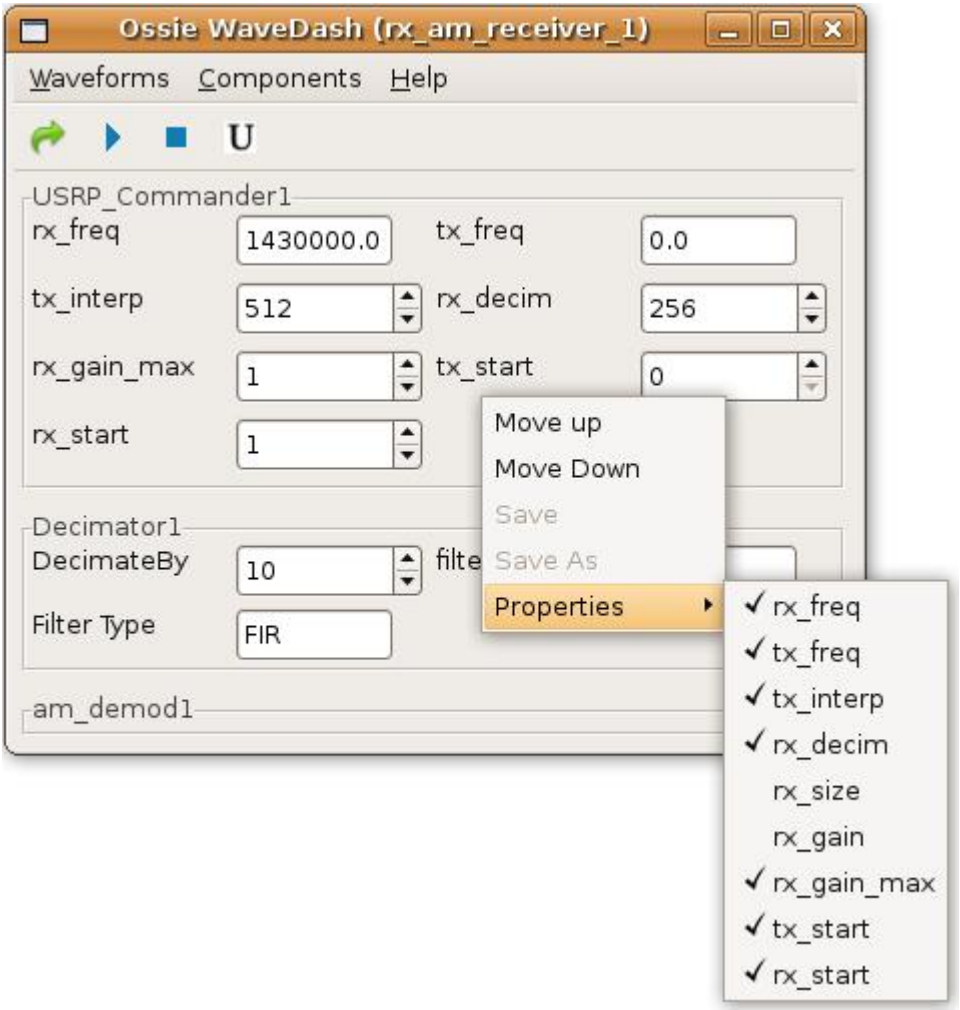
9.5.2 Re-arrange components

Components can also be stacked differently from the default order. To rearrange, right click on the component area and move it up or down using Move Up/Move Down choice in that context menu.



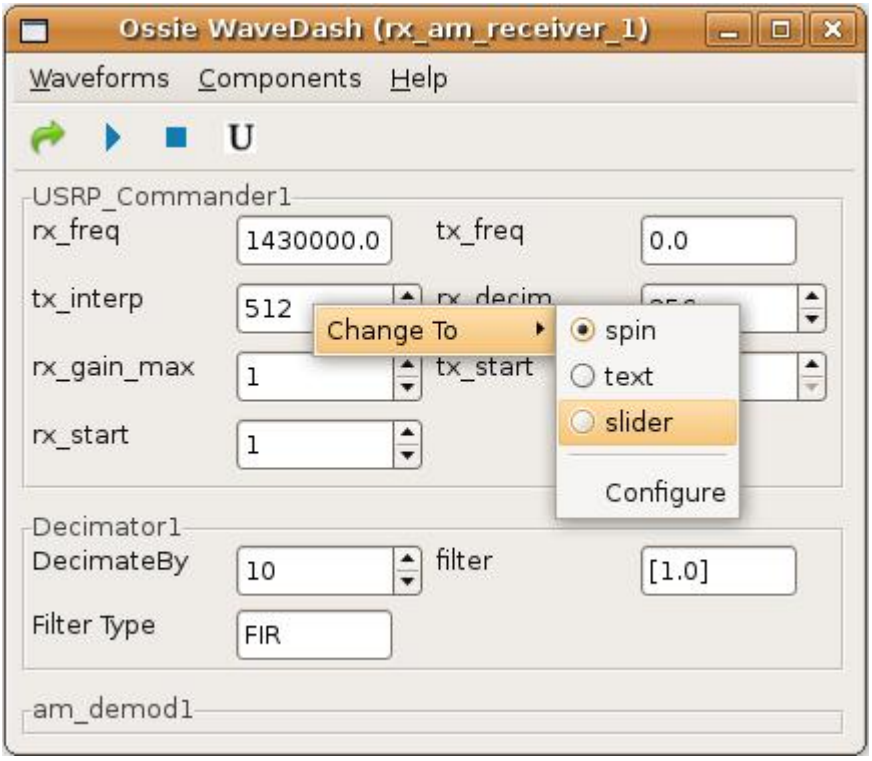
9.5.3 Show or Hide Properties

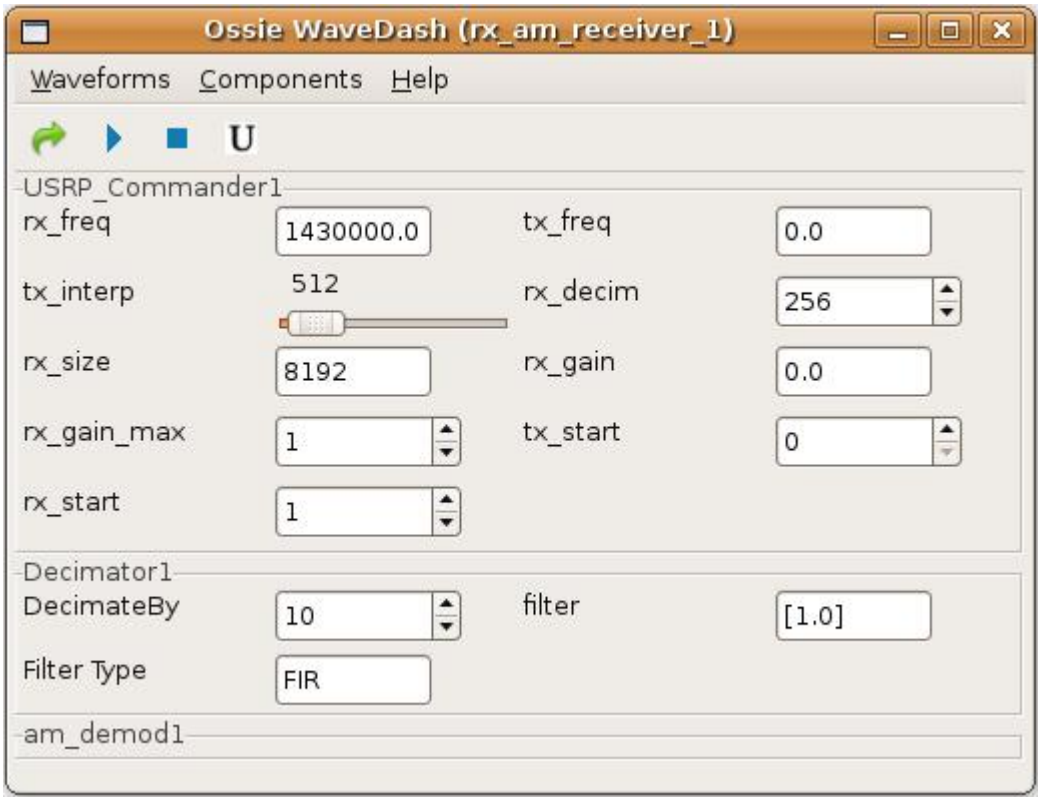
Similar to components, properties of a component can also be customized. The component box context menu has a submenu Properties which lists the properties of a component. Select/Deselect the properties from the context menu to show/hide the properties on the screen.



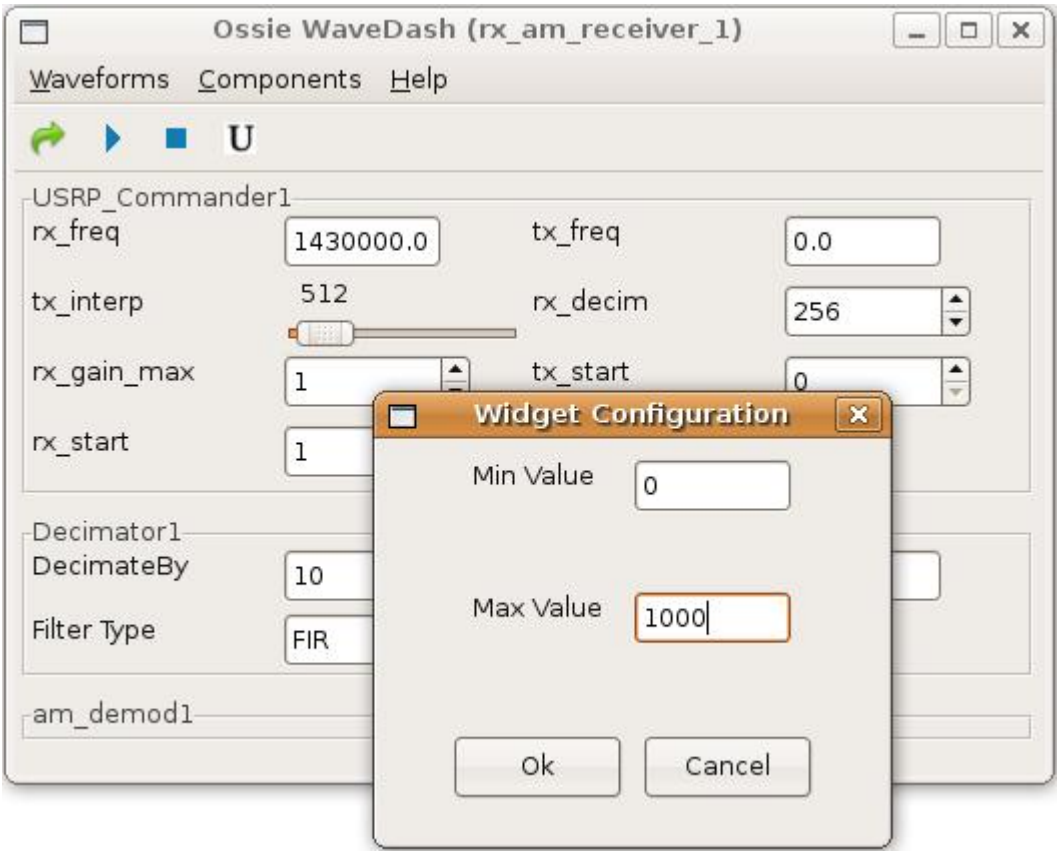
9.5.4 Change Property Widget

Each property of a component has its own data type associated. The application provides a default mapping of the property data type to a GUI widget. For e.g. integer values are shown in a text box, short values are shown in a spin box, etc However, user may change the widget for a property by right clicking on the current widget and choose the alternate widget compatible to the property datatype. The compatibility here refers to the right mapping between datatype and the GUI widget. For e.g. a property of type string cannot be represented by a spin box or slider. Hence, no alternative widget can be seen when you open the context menu of a property.


Ossie WaveDash (rx_am_receiver_1)
Waveforms
Components
Help
U
USRP_Commander1
rx_freq
1430000.0
tx_freq
0.0
tx_interp
512
Change To
spin
text
slider
Configure
rx_gain_max
1
tx_start
rx_start
1
Decimator1
DecimateBy
10
filter
[1.0]
Filter Type
FIR
am_demod1



The widgets Spinbox and Slider have a minimum and maximum values associated with them. When spin box or slider is chosen as a widget, the application sets the default minimum and maximum as 0 and 10000 respectively. Should the user wish to change these values for better access, right click on the widget and choose Configure. This would open a new dialog box where you can configure minimum and maximum values.


Ossie WaveDash (rx_am_receiver_1)
Waveforms
Components
Help
U
USRP_Commander1
rx_freq
1430000.0
tx_freq
0.0
tx_interp
512
rx_decim
256
rx_gain_max
1
tx_start
0
rx_start
1
Decimator1
DecimateBy
10
Filter Type
FIR
am_demod1
Widget Configuration
Min Value
0
Max Value
1000
Ok
Cancel

10 Helping With Development

OSSIE is an open source project and we encourage everyone interested to contribute to the project.

10.1 Giving Feedback and Submitting Bugs

The *ossie-discuss* mailing list and the #ossie IRC channel on Freenode are probably the best methods of discussing and collaboratively solving end-user problems. For development-oriented issues, Trac tickets and the *ossie-dev* mailing list are probably best. See Section 2.2.4 for more information on the IRC channel and Section 2.2.2 for details on the mailing lists.

10.2 Contributing Code

Contributions are heartily welcomed. The best way to get to know the developers and really get involved is submit patches that fix bugs or add features. Patches can generally be sent to us through Trac or the mailing list. Write access to the subversion repository can be given to contributors who have proved their reliability.

In order to make sure OSSIE stays open, we have licensed the source code under the GNU General Public License Version 2 and the Lesser General Public License. Contributions to the project must be made under these licenses. Also, the preferred coding style is that devised for the Linux kernel [7]. See [4, 5] for more information about the licenses.

11 Troubleshooting

This section includes several common problems you may encounter when installing and running OSSIE. It is by no means complete. If you cannot find your problem here, please feel free to contact the team (refer to Section 2.2)

11.1 nodeBooter Throws an Exception

When I try to start `nodeBooter`, I get this output:

```
$ nodeBooter -D -d nodes/default_GPP_node/DeviceManager.dcd.xml
NB: Starting Domain Manager
Fatal: An exception occurred! Type:RuntimeException, Message:The primary
  document entity could not be opened.
Id=/home/ossieuser/domain/DomainManager.dmd.xml
PublicId :
SystemId :
Line number : 0
Column number : 0
terminate called after throwing an instance of 'int'
Aborted
```

Solution: You need to run `nodeBooter` from the `/sdr` directory.

11.2 nodeBooter Fails to Resolve 'TRANSIENT' NameService

When I try to start `nodeBooter` I get this output:

```
$ nodeBooter -D -d nodes/default_GPP_node/DeviceManager.dcd.xml
Failed to resolve NameService: TRANSIENT
terminate called after throwing an instance of 'CORBA::TRANSIENT'
Aborted
```

Solution: You need to start the naming service. If you installed omniORB via your system's package manager, `omniNames` will start automatically when you restart your machine, if the service is configured properly. Refer to Appendix A for instructions on managing services under Fedore Core 9. You can manually start the `omniNames` by creating and running the shell script described in Appendix B.

11.3 OWD Component Editor Does Not Allow me to Select the Device

I am trying to edit a component to assign it to a device. I have selected the `default_XXX_node` under "Waveform Deployment Settings" but nothing exists under the "Device" spinbox.

Solution: This is a bug in OWD. You have probably not selected `default_XXX_node` with the mouse, even though it is visible on the screen. Click on the "Node" spinbox again and re-select `default_XXX_node`, making sure that you actually click on the text. Now when you click on the "Device" spinbox you should see the device you want.

Appendices

A Managing Services on Fedora Core 9

Controlling which services are currently running on your computer and which start up with your computer is very easy with Fedora Core's Service Configuration interface. From the GNOME taskbar, click System→Administration→Services. You will need to enter your root (system administrator) password to use the dialog. The window that pops up shows you all of the available services on your computer. Services that have a check in the box next to them are currently running. To turn a service on or off, simply click the box, or select it and press the 'Start' or 'Stop' button at the top of the list.

In order to edit which services start with your computer when it boots, you need to know a little about runlevels. You computer has several different runlevels. The one you are most concerned with is runlevel 5. Runlevel 5 means multiuser X-server mode. Generally speaking, if you are using a graphical interface then you are in runlevel 5.

To have omniNames start with your computer, select it in the services list, click 'Edit Runlevel' on the top menubar, and select runlevel 5. Click the save button to save your configuration.

That's it! The next time you start your computer, omniNames will start running automatically.

B Creating `omniNames.sh`

Installing omniORB from RPM automatically starts the naming service when the system boots. If omniORB was *not* installed using RPM, the naming service will probably need to be started manually. To do so, create a file called `omniNames.sh` containing the following

Listing 1: `omniNames.sh` file

```
#!/bin/sh

killall omniNames
rm /sdr/logs/omninames*

omniNames -start -logdir /sdr/logs &
```

Once this file is created, the file must be given executable permissions:

```
$ chmod 755 omniNames.sh
# cp omniNames.sh /usr/local/bin
```

Create a directory for the logs:

```
$ mkdir /sdr/logs
```

The script can now be run by executing `omniNames.sh` in any terminal. You can check to see if `omniNames` is running by executing:

```
$ nameclt list
```

If the naming service is *not* running you will see an error like:

```
Unexpected error when trying to narrow the NamingContext.
```

C Configuring `ossie.pth`

In order for the Python tools to link properly, a Python path file must be created to point to the shared modules. The default `ossie.pth` file is located under `ossie-0.7.4/src/system/ossie/idl/python/` and is installed by default in `/usr/lib/pythonX.X/site-packages` when you install the ossie core framework libraries.

If you did *not* install omniORBpy using an RPM, you might need to add `/usr/local/lib/pythonX.X/site-packages` to `ossie.pth`, where *X.X* corresponds to the specific version of Python installed on your system. Now reinstall by running `python setup.py install` as root within the `ossie-0.7.4/src/system/ossie/idl/python/` directory:

```
# cd ossie-0.7.4/src/system/ossie/idl/python/
# python setup.py install
```

D Data Source Component Creation

The component TxDemo will be used as an example on how to create a data source. When creating a component which is not a data source, typically only the function `ProcessData` needs to be modified to include the desired operation. This is not the case for a source component, however. The code that needs to be updated in this example resides in the `start`, `stop`, and `ProcessData` functions. Locate the directory `components/TxDemo` and open the files TxDemo.h and TxDemo.cpp.

In TxDemo.h, near the end of the file the following code will be listed:

```
bool continue_processing(void);
volatile bool thread_started;
```

The flag `thread_started` holds the state of the component; whether or not the component should continue sending data. A function `continue_processing` is also declared which will be used to check `thread_started` and determine if the component should continue sending packets.

In TxDemo.cpp, locate the `start` function and the following code will be listed:

```
omni_mutex_lock l(processing_mutex);
if (false == thread_started){
  thread_started = true;    //initialize
  //Create the thread for the writer's processing function
  processing_thread = new omni_thread(Run, (void *) this);

  //Start the thread containing the writer's processing function
  processing_thread->start();
  }
```

This code creates a new thread and starts processing when the `thread_started` variable is set to true.

Now locate the `stop` function, and the following code will be listed:

```
omni_mutex_lock l(processing_mutex);
thread_started=false;
```

This code stops the processing on the thread by setting `thread_started` to false.

Now locate the `ProcessData` function, and the following code will be listed:

```
while(continue_processing()) {
  // push data to output
  dataOutPort->pushPacket(I_out, Q_out);

  // wait
  usleep(packet_delay*1000);
  }
```

This code checks the function `continue_processing` to determine if packets should continue to be sent using `pushPacket`. The `usleep` function call is specific to TxDemo, and is not necessarily required in other data sources.

The last section of code that must be created is the `continue_processing` function. The code for the function is listed as the following:

```
bool TxDemo_i::continue_processing(void)
{
  omni_mutex_lock l(processing_mutex);
  return thread_started;
}
```

All of the code listings provided above are not created by default when a new component is generated, since the code is only required for source components. Instead, all of the code must be added by hand. The addition of the code will create the appropriate environment for the data source, and afterwards only the data creation must be added to the `ProcessData` function.

E Known Bugs

E.1 OSSIE Core Framework

1. After running a waveform, `nodeBooter` must be killed using `Ctrl-C` and does not exit gracefully.
2. Running `wavLoader.py` results in two copies of each SAD file, for example:

   ```
   1: //waveforms/ossie_demo/ossie_demo.sad.xml
   2: /DeviceManager/waveforms/ossie_demo/ossie_demo.sad.xml
   ```

   This is an artifact of the filesystem. You should always choose the first one and ignore the second.

E.2 OWD

1. Generating components with certain interfaces result in code that cannot be compiled directly. OWD generates the source, but will display a warning in the terminal:

   ```
   Interfaces other than complex and real Short, Float, Char, long and Double are
   not supported yet in the process data function generation!!
     See writeProcessDataDeclaration in genStructure.
   ```

2. When trying to assign a component to a device in the "Waveform Deployment Settings" section in the OSSIE Component Editor window, if the "Node" is not properly selected, no devices will appear in the spinbox.
3. When generating a waveform, the `configure.ac` file includes tests for source code instead of just installation.
4. After opening ComponentFrame from OWD for a new component, Python will not quit when you quit OWD.

E.3 ALF

1. Waveforms with more than about fifteen components will not display properly and are cut off from view.
2. Aside from components, the Launch Components as Waveforms pane includes devices. NodeBooter doesn't seem to crash when trying to launch a device as a waveform, although an error appears in the nodeBooter window.
3. Pointing ALF to an IP address does not fully recognize the available components or available applications. We still need to implement support for the CF::FileSystem.

E.4 OEF

1. OEF does not support generation of new node definitions. Currently, we still rely on OWD for that functionality.

E.5 WaveDash

1. Display of dialog does not expand to support large words.

F List of Abbreviations

ALF A waveform application visualization and debugging environment for OSSIE. The name was initially derived from the "Alien Life Form" television series and character.

DAS Device Assignment Sequence

JTRS Joint Tactical Radio System

OSSIE Open-source SCA Implementation::Embedded

OWD OSSIE Waveform Developer

QPSK Quadrature phase-shift keying

SAD Software Assembly Descriptor

SCA Software Communications Architecture

References

[1] Autoconf. http://www.gnu.org/software/autoconf/.

[2] Automake. http://www.gnu.org/software/automake/.

[3] Doxygen. http://www.stack.nl/~dimitri/doxygen/.

[4] GNU General Public License. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html#TOC1.

[5] GNU Lesser General Public License. http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html#TOC1.

[6] IRC Tutorial. http://www.irchelp.org/irchelp/irctutorial.html.

[7] Linux kernel coding style. http://pantransit.reptiles.org/prog/CodingStyle.html.

[8] The JTRS Software Communications Architecture. http://sca.jpeojtrs.mil/.

[9] VMWare Player User Manual. http://www.vmware.com/pdf/vmware_player200.pdf.

[10] E. Buracchini. The Software Radio Concept. *IEEE Commun. Mag.*, 38(9):138–143, 2000.

[11] J. H. Reed. *Software Radio: a Modern Approach to Radio Engineering*. Prentice Hall, Upper Saddle River, 2002.

[12] John Bard and Vincent J Kovarik. *Software Defined Radio: The Software Communications Architecture*. Wiley, Hoboken, NJ, 2007.

[13] Simon Tatham. How to Report Bugs Effectively. http://www.chiark.greenend.org.uk/~sgtatham/bugs.html.

G Change Log

G.1 OSSIE 0.7.4

1. Addition of new waveform: Lab 5 example AM application.
2. AutomaticGainControl component will output ones when threshold is not met.
3. ALF ConnectTool has improved support for port connections.
4. WaveDash can now control single-component waveforms created and launched in ALF.
5. ALF allows user to specify IP address for CORBA naming service.

G.2 OSSIE 0.7.3

1. Introduction of Waveform Dashboard (WaveDash).
2. Addition of new waveforms: OSSIETalkLoopBack, OSSIETalkUSRP.
3. Addition of new components: Channel, Conv_Dec, Conv_Enc, DigitalModem, FrameAssembler, OSSIETalk, PacketResizer, SymbolSyncPoly.
4. Decimator has the ability to use either IIR or FIR.
5. OWD provides generation of custom license headers.
6. More removal of vestigial project files.

G.3 OSSIE 0.7.2

1. Improved start/stop capability within TxDemo component.
2. Corrected XML Template files for OWDC component generation.
3. Removal of incorrect entry point tags in default_GPP_node SPD.
4. Deletion of vestigial project files.
5. Addition of Interpolator component.

G.4 OSSIE 0.7.1

1. Removal of duplicate profile templates.
2. Added start, stop capability to TxDemo component.
3. Fixed USRP_Commander rx_freq property typo.
4. Fixed capability of components being run as waveforms.

G.5 OSSIE 0.7.0

1. Start of change log.
2. Allows for non /sdr install.
3. Installation through RPMs.
4. ALF can start and stop waveforms.
5. Reduced plot tool's refresh rate.
6. Removed "sample waveform" option in OWD.
7. Removed duplicate DOCTYPE descriptors in generated XML files.
8. Fixed path in USRP's SCD file.
9. Fixed soundCardCapture device name to prevent OWD error.
10. Added Ubuntu 8.04 as supported platform.
11. OWD default save directory is now home directory.
12. ALF's plot tool has normalized frequency axis.
13. Debug statements use DEBUG macro instead of std::cout.
14. Tools now installed to site-packages.
15. ALF, OWD, and OWD Component Editor can be run from any directory.
16. More than two nodes can be run through a single domain manager.
17. Added webcam components.
18. Installation from source broken into two steps.