Benthic Boundary Layer Project

Kilo Nalu Observatory

Real-time Streaming System

Operations Guide

May 21, 2009

Dr. Margaret McManus
Department of Oceanography
University of Hawaii at Manoa

Table of Contents

1. Operations Guide

1.1. Overview

The Kilo Nalu streaming system is a centralized means of collecting data in near real-time that uses the Open Source DataTurbine software (see http://www.dataturbine.org). The system consists of instruments deployed on the Kilo Nalu cable array (and others via wireless connections) that communicate with the DataTurbine streaming service via customized drivers (written in Java) that understand the stream formats of each instrument. These drivers are known as 'Sources', and the names of the drivers reflect this (e.g. ADCPSource). Software client programs that connect to the DataTurbine to fetch data are known as 'Sinks', and their names reflect this as well (e.g. FileArchiverSink). The primary DataTurbine installation is on the shore station linux server, and mirrors the data streams to the secondary DataTurbine installation at the UH Manoa campus via the wireless VPN connection. At the shore station, data streams are fetched from the DataTurbine on an hourly or daily basis (depending on the specific instrument), and are archived to disk by the FileArchiverSink client for each instrument. These files are also mirrored to the campus Linux server via an hourly process (using the `rsync` command). Web-based graphics are produced using Matlab code that queries the campus DataTurbine on a scheduled basis.

Each of these system components are described in more detail below, along with instructions on how to manage each of them. **Note: This guide assumes some familiarity with the Linux operating system and commands.**

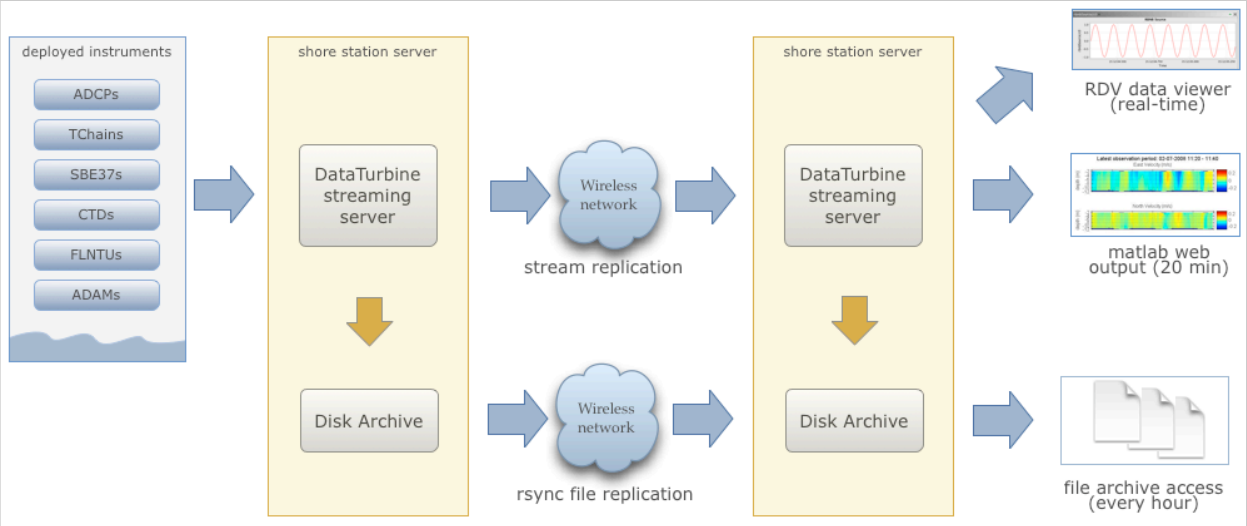


Figure 1. Overview of the Kilo Nalu real-time data software and backup architecture. Data are mirrored to the campus system using the DataTurbine software and via rsync, and are accessed by desktop and web-based applications.

1.2. Managing the DataTurbine Server Software

The DataTurbine software is installed in `/usr/local/RBNB/V3.1a`. It's running on port `3333` both on the shore station Linux server (Internet IP: `168.105.160.139`, VPN IP `192.168.100.60`) and the UH campus server (Internet IP: `bbl.ancl.hawaii.edu`, VPN IP: `192.168.103.50`), and is set up as a standard Linux service installed in the `/etc/init.d` directory, with a run level script called `rbnb`. The DataTurbine is set to start whenever each of the systems is rebooted. The server's event log is located in `/var/log/rbnb/rbnb.log`. The DataTurbine's internal stream archive is located in `/var/lib/rbnb`. In the event that instrument source drivers cannot connect to the DataTurbine, look at the event log to see if there are connection, memory, or file system

errors. If so, the DataTurbine service may need to be restarted, the stream archives reloaded (automatic), and the instrument drivers reconnected.

1.2.1. Starting the DataTurbine

The DataTurbine service is started like any other Linux service by calling the run-level script. To do so, as the *kilonalu* user, ssh to the Linux server in question (shore lab or campus lab), and execute the following command in a terminal:

```
$ sudo service rbnb start
```

If prompted, enter the *kilonalu* user's password. This will start the RBNB DataTurbine service and load any existing data stream archives found in the `/var/lib/rbnb` directory.

1.2.2. Stopping the DataTurbine

The DataTurbine service is stopped like any other Linux service by calling the run-level script. To do so, as the *kilonalu* user, ssh to the Linux server in question (shore lab or campus lab), and execute the following command in a terminal:

```
$ sudo service rbnb stop
```

If prompted, enter the *kilonalu* user's password. This will cleanly unload any existing data stream archives and stop the RBNB DataTurbine service.

1.2.3. Troubleshooting the DataTurbine

There may be times when the DataTurbine isn't performing as expected. For instance, client source drivers may not be able to connect, or stream replication from one DataTurbine to another may not continue. There are a few common causes to these sorts of symptoms, including server memory problems, open file problems, or disk space problems. As the *kilonalu* user, use the following command to inspect the DataTurbine's event log to see if any critical errors are being logged:

```
$ tail -f /var/log/rbnb/rbnb.log
```

This will show the most recent log entries that pertain to the DataTurbine service, such as connections, disconnections, or errors. Type Control-c to stop viewing the scrolling log file. Errors such as 'too many open files', or java.lang.OutOfMemoryException indicate resource problems on the server. The 'too many open files' error indicates that the DataTurbine service has exceeded it's operating system-level limits for open files. The best solution to this is to stop and start the DataTurbine, and reconnect the instrument streams. Also, contact Chris (cjones@soest.hawaii.edu) about this, since it can often be due to driver software that isn't closing connections properly. Out of memory errors may be caused by the aggregate memory requests by instrument drivers exceeding the available memory on the server. To mitigate this, the drivers can be tuned to request less memory. As an example, the 20m 1200 kHz ADCP is started by calling the startup script found in `/usr/local/bbl/trunk/bin` called `KN02XX_020ADCP020R00-Source.sh`. The pertinent line of this script calls the Java source driver with a number of command line parameters:

```
java edu.hawaii.soest.kilonalu.adcp.ADCPSource\
 -H 192.168.100.139\
 -P 2102\
 -S KN02XX_020ADCP020R00\
```

```
    -C BinaryPD0EnsembleData\
    -s 192.168.100.60\
    -p 3333\
    -z 50000\
    -Z 31536000
```

The `-z` option requests that `50000` RBNB data frames (in this case ADCP ensembles) be stored in physical memory, whereas the `-Z` option requests that `31536000` RBNB data frames be stored on disk before they are overwritten. For the 20m 1200 kHz ADCP, this equates to one `955` byte ensemble per frame, resulting in a memory request of `(955b x 50000) = 47.75MB`. The on-disk storage request equates to around six months of ensembles, or `(955b x 31536000) = 30.1GB`. These resource request values can be adjusted if the aggregate requests of all of the instrument drivers exceed the limits of the server in terms of memory and disk space. The shore station server currently has `8GB` of physical memory, and `385GB` of disk space available to the DataTurbine. The campus BBL server currently has `12GB` of physical memory, and `50GB` of disk space available to the DataTurbine.

1.3. Managing the DataTurbine Instrument Drivers

Each instrument type in the water has a corresponding instrument driver used to connect it to the DataTurbine. For instance, for ADCPs, there's a Java-based driver called ADCPSource, and for the CTDs, there's a driver called CTDSource. The following table lists the instruments and their associated drivers, along with which DataTurbine they connect to by default. Driver Source naming conventions can be found in the data management plan.

Instrument Description	DataTurbine Name	Instrument IP	Driver Name	Host DataTurbine
10m CN ADAM control	KN00XX_010ADAM010R00	192.168.100.200	ADAMSource	Shore Station Lab
10m CN ADAM monitor 1	KN00XX_010ADAM010R01	192.168.100.201	ADAMSource	Shore Station Lab
10m CN ADAM monitor 2	KN00XX_010ADAM010R02	192.168.100.202	ADAMSource	Shore Station Lab
10m CN ADAM micronode	KN00XX_010ADAM010R04	192.168.100.203	ADAMSource	Shore Station Lab
10m SN ADAM control	KN01XX_010ADAM010R00	192.168.100.204	ADAMSource	Shore Station Lab
10m SN ADAM monitor 1	KN01XX_010ADAM010R01	192.168.100.205	ADAMSource	Shore Station Lab
10m SN ADAM monitor 2	KN01XX_010ADAM010R02	192.168.100.206	ADAMSource	Shore Station Lab
10m SN ADAM geochem	KN01XX_010ADAM010R03	192.168.100.210	ADAMSource	Shore Station Lab
10m 1200kHz ADCP	KN0101_010ADCP010R00	192.168.100.136	ADCPSource	Shore Station Lab
10m WetLabs FLNTU	KN0101_010FLNT010R00	192.168.100.136	FLNTUSource	Shore Station Lab
10m TChain	KN0101_010TCHN010R00	192.168.100.136	TChainSource	Shore Station Lab
10m Seabird SBE37	KN0101_010SBEX010R00	192.168.100.136	SBE37Source	Shore Station Lab
20m Sub ADAM control	KN0201_010ADAM010R00	192.168.100.220	ADAMSource	Shore Station Lab
20m Sub ADAM monitor 1	KN0201_010ADAM010R01	192.168.100.221	ADAMSource	Shore Station Lab
20m Sub ADAM monitor 2	KN0201_010ADAM010R02	192.168.100.222	ADAMSource	Shore Station Lab
20m Sub ADAM BS48 relay	KN0201_010ADAM010R03	192.168.100.223	ADAMSource	Shore Station Lab
20m 1200kHz ADCP	KN02XX_020ADCP020R00	192.168.100.139	ADCPSource	Shore Station Lab
20m TChain	KN0201_020TCHN020R00	192.168.100.139	TChainSource	Shore Station Lab
20m Seahorse CTD	TBD	TBD	CTDSource	Campus HIG Lab
01m Alawai NS01 CTD	AW01XX_002CTDXXXXR00	68.25.35.242	CTDSource	Campus HIG Lab
01m Alawai NS02 CTD	AW02XX_001CTDXXXXR00	68.25.32.149	CTDSource	Campus HIG Lab
01m Waikiki NS03 CTD	WK01XX_001CTDXXXXR00	68.25.168.134	CTDSource	Campus HIG Lab
01m Waikiki NS04 CTD	TBD	TBD	CTDSource	Campus HIG Lab
JABSOM WX Station	KNWXXX_XXXDVP2XXXR00	168.105.160.135	DavisWxSource	Campus HIG Lab

1.3.1. Starting Instrument Drivers

Each instrument driver can be started by calling a convenience script that has preconfigured startup values for each of the drivers. These convenience scripts are located in `/usr/local/bbl/trunk/bin`, and they all follow the naming pattern of `'Start-SOURCENAME.sh'`. Likewise, the stop scripts follow the naming pattern `'Stop-SOURCENAME.sh'`. **Note: It's a good idea to always stop a driver before starting one, to ensure that two**

drivers aren't running for the same instrument. See Stopping Instrument Drivers below. As an example, to start the 20m 1200kHz ADCP instrument driver, ssh to the Linux server in question (shore lab or campus lab) as the *kilonalu* user, and execute the following commands in the terminal :

```
$ Stop-KN02XX_020ADCPXXXR00.sh
$ Start-KN02XX_020ADCPXXXR00.sh
```

This will cleanly shut down any existing 20m 1200kHz ADCP drivers, start a new driver, and will also start tailing the log file for the specific driver so you can verify that the samples are being sent to the DataTurbine. To stop viewing the log file, type Control-c in the terminal. The table below lists the start and stop scripts:

Instrument Description	Start Script Name	Stop Script Name
10m CN ADAM control	Start-KN00XX_010ADAM010R00.sh	Stop-KN00XX_010ADAM010R00.sh
10m CN ADAM monitor 1	Start-KN00XX_010ADAM010R01.sh	Stop-KN00XX_010ADAM010R01.sh
10m CN ADAM monitor 2	Start-KN00XX_010ADAM010R02.sh	Stop-KN00XX_010ADAM010R02.sh
10m CN ADAM micronode	Start-KN00XX_010ADAM010R04.sh	Stop-KN00XX_010ADAM010R04.sh
10m SN ADAM control	Start-KN01XX_010ADAM010R00.sh	Stop-KN01XX_010ADAM010R00.sh
10m SN ADAM monitor 1	Start-KN01XX_010ADAM010R01.sh	Stop-KN01XX_010ADAM010R01.sh
10m SN ADAM monitor 2	Start-KN01XX_010ADAM010R02.sh	Stop-KN01XX_010ADAM010R02.sh
10m SN ADAM geochem	Start-KN01XX_010ADAM010R03.sh	Stop-KN01XX_010ADAM010R03.sh
10m 1200kHz ADCP	Start-KN0101_010ADCP010R00.sh	Stop-KN0101_010ADCP010R00.sh
10m WetLabs FLNTU	Start-KN0101_010FLNT010R00.sh	Stop-KN0101_010FLNT010R00.sh
10m TChain	Start-KN0101_010TCHN010R00.sh	Stop-KN0101_010TCHN010R00.sh
10m Seabird SBE37	Start-KN0101_010SBEX010R00.sh	Stop-KN0101_010SBEX010R00.sh
20m Sub ADAM control	Start-KN0201_010ADAM010R00.sh	Stop-KN0201_010ADAM010R00.sh
20m Sub ADAM monitor 1	Start-KN0201_010ADAM010R01.sh	Stop-KN0201_010ADAM010R01.sh
20m Sub ADAM monitor 2	Start-KN0201_010ADAM010R02.sh	Stop-KN0201_010ADAM010R02.sh
20m Sub ADAM BS48 relay	Start-KN0201_010ADAM010R03.sh	Stop-KN0201_010ADAM010R03.sh
20m 1200kHz ADCP	Start-KN02XX_020ADCP020R00.sh	Stop-KN02XX_020ADCP020R00.sh
20m TChain	Start-KN0201_020TCHN020R00.sh	Stop-KN0201_020TCHN020R00.sh
20m Seahorse CTD	TBD	TBD
01m Alawai NS01 CTD	Start-AW01XX_002CTDXXXXR00.sh	Stop-AW01XX_002CTDXXXXR00.sh
01m Alawai NS02 CTD	Start-AW02XX_001CTDXXXXR00.sh	Stop-AW02XX_001CTDXXXXR00.sh
01m Waikiki NS03 CTD	Start-WK01XX_001CTDXXXXR00.sh	Stop-WK01XX_001CTDXXXXR00.sh
01m Waikiki NS04 CTD	TBD	TBD
JABSOM WX Station	Start-KNWXXX_XXXDVP2XXXR00.sh	Stop-KNWXXX_XXXDVP2XXXR00.sh

1.3.2. Stopping Instrument Drivers

As above, each instrument driver can be stopped by calling a convenience script. Stop scripts follow the naming pattern `'Stop-SOURCENAME.sh'`. As an example, to stop the 20m 1200kHz ADCP instrument driver, ssh to the Linux server in question (shore lab or campus lab) as the *kilonalu* user, and execute the following command in the terminal :

```
$ Stop-KN02XX_020ADCPXXXR00.sh
```

This will cleanly shut down any existing 20m ADCP driver. The current Stop scripts are listed in the table above in the Starting Instrument Drivers section.

1.3.3. Troubleshooting Instrument Drivers

There may be many reasons why an instrument driver isn't streaming data, but most issues tend to be associated with power outages, network outages, or memory/file issues with the DataTurbine service. The first step in troubleshooting is to view the log file for the given instrument. As an example, to view the 20m 1200kHz ADCP streaming log file, issue the following command as the *kilonalu* user in a terminal on the server in question (either shore station or campus bbl server):

```
$ tail -f /var/log/rbnb/KN02XX_020ADCP020R00-Source.log
```

Each of the log files follow the naming convention of `SOURCENAME-Source.log`, so just substitute the source name string to view the log of that particular instrument driver. To stop viewing the log, type Control-c in the terminal.

As each instrument sample is read over the wire by the instrument driver, the sample will be parsed and inserted into the DataTurbine, and a line will be added to the log file stating so. For instance, for the 20m 1200kHz ADCP, the log file entries are one line per 955 byte ensemble, and should look like:

```
Processed byte # 955 7f - log msg is: 467204614 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467206774 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467208935 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467211100 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467213265 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467215425 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
Processed byte # 955 7f - log msg is: 467217590 [StreamingThread] INFO edu.hawaii.soest.kilonalu.adcp.ADCPSource  - Sent ADCP ensemble to the data turbine.
```

The instrument log entries vary per instrument, but they each say something to the effect of 'Sent sample to the DataTurbine'. If you do not see these messages scrolling by as you tail the file, then either no data are being sent over the wire, or the driver has lost it's connection to the DataTurbine. Try stopping and starting the driver in question, and tail the log file again to see if it has recovered. If not, check to be sure that data are streaming from the instrument through the appropriate Digi portserver. If data are streaming, but not being added to the DataTurbine, look to see if there is a problem with the DataTurbine accepting connections. See section 1.2.3 above. If you continue to have trouble, email Chris at cjones@soest.hawaii.edu.

1.4. Replicating Instrument Data Streams

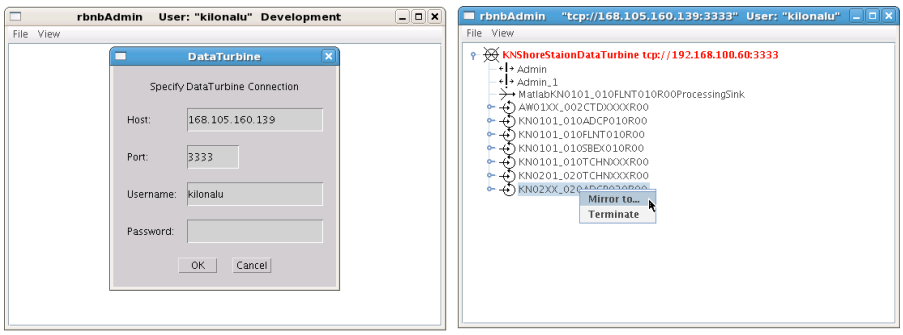
When each of the instrument drivers on the Kilo Nalu array are connected to the shore station DataTurbine, each data stream should then be replicated to the campus DataTurbine. The DataTurbine software ships with a small graphical administrative program called *rbnbAdmin* to manage the data streams. This program can be run from your workstation if you have downloaded it and have installed Java, but these instructions will describe how to use the administrative program on the BBL campus server.

First, connect to the campus server as the *kilonalu* user using VNC as described in section 1.10.1 in this guide. Open a terminal by right-clicking on the red Redhat Linux desktop background, and choosing the *Open terminal* menu item. In the terminal, issue the following command:

```
$ java -jar /usr/local/RBNB/V3.1B4a/bin/admin.jar &
```

This will open up the admin utility within the VNC window. Next, choose the *File --> Open ...* menu item. You will be connecting to the shore station Linux server, and so enter the following into the form:

- Host: `168.105.160.139`
- Port: `3333`
- Username: `kilonalu`
- Password: [leave blank]



Dialog boxes in the rbnbAdmin application used to mirror data sources from the shore station DataTurbine server to the campus DataTurbine server.

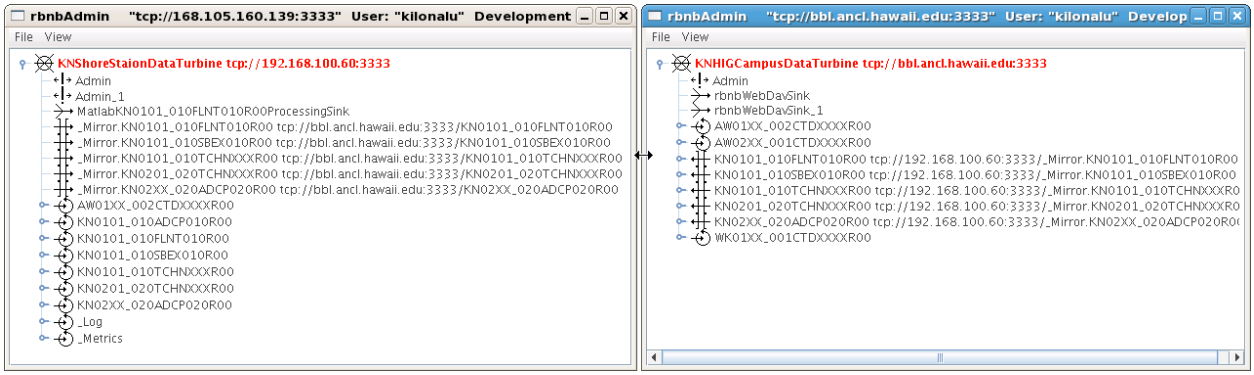
Once you press 'Ok', the application should connect, and show you a list of the data sources on the DataTurbine labeled 'KNShoreStationDataTurbine'. To replicate a data source, first click on a data source name - in this example KN02XX_020ADCP020R00, and copy the name using the Control-c keys. Then, right-click on the same name, and choose *Mirror to ...* in the menu list. Fill in the replication form with the following information. In the *Data Path* field, you can paste the source name in using the Control-v keys.

- To: `tcp://bbl.ancl.hawaii.edu:3333`
- Data Path: `KN02XX_020ADCP020R00`
- From: `tcp://168.105.160.139:3333`
- Data Path: `KN02XX_020ADCP020R00`
- Start: `Oldest (Radio button)`
- Stop: `Continous (Radio Button)`
- Buffer Size: `Match Source (Check box)`

Dialog box used to replicate a data source from the shore station to the campus, in this case the 20m 1200kHz ADCP, KN02XX_020ADCP020R00.

Once you press 'Ok', the two DataTurbines will connect and establish the replication for the data source. **Note: In the 'Start' field above, you can choose 'Now' or 'Oldest'. The latter will attempt to replicate all samples in the data source from the oldest point in time stored in the DataTurbine, but due to network performance across the wireless link, this can take days to establish. If you choose 'Now', the most recent samples will begin replicating, and the campus DataTurbine will have a gap in the data time series.**

To see the status of the replicated streams, choose the *Hidden* menu item under the 'View' menu. The following screenshot shows two rbnbAdmin clients open, one connected to each DataTurbine (shore station and campus), and shows all of the data sources, along with their replication links.



If there is a network outage, the replication links should re-establish when the network is restored, and the samples will synchronize across the DataTurbines. You can check the status of the replication by pointing your browser to http://bbl.ancl.hawaii.edu:8080/RBNB. You'll see the list of replicated data source, and by refreshing the browser window, the time stamps for each source should increment based on the sampling rate of the instrument.

1.5. Managing the DataTurbine File Archivers

After each instrument driver is started, a file archiver process should also be started to ensure that data are written to the disk archive directly (either hourly or daily, depending on the archiver configuration). The FileArchiverSink is a Java program that can write any type of data stream to disk. If an instrument driver is

stopped, it doesn't mean that the archiver process is also stopped. An existing archiver should just be idle, and will try to archive any data within it's scheduled time period. The archive directory on the shore station and the campus server is `/data`. For data originating from the Kilo Nalu array, the archivers are set to write files to `/data/kilonalu/[SOURCENAME]`. For data originating wirelessly from the nearshore sensors, the archivers are set to write files to `/data/raw/alawai/[SOURCENAME]`. Infrequently, a file archiver process may be running, but may not archive files correctly, and may need to be restarted.

1.5.1. Starting the Instrument File Archivers

Each instrument file archiver can be started by calling a convenience script that has preconfigured startup values for each of the archivers. These convenience scripts are located in `/usr/local/bbl/trunk/bin`, and they all follow the naming pattern of `'Archiver-Start-SOURCENAME.sh'`. Likewise, the stop scripts follow the naming pattern `'Archiver-Stop-SOURCENAME.sh'`. **Note: It's a good idea to always stop an archiver before starting one, to ensure that two archivers aren't running for the same instrument. See Stopping Instrument Archivers below.** As an example, to start the 20m 1200kHz ADCP instrument file archiver, ssh to the Linux server in question (shore lab or campus lab) as the *kilonalu* user, and execute the following commands in the terminal :

```
$ Archiver-Stop-KN02XX_020ADCPXXXR00.sh
$ Archiver-Start-KN02XX_020ADCPXXXR00.sh
```

This will cleanly shut down any existing 20m ADCP file archiver and start a new archiver. The file archiver start and stop scripts are listed below.

Instrument Description	Archiver Start Script Name	Archiver Stop Script Name
10m CN ADAM control	Archiver-Start-KN00XX_010ADAM010R00.sh	Archiver-Stop-KN00XX_010ADAM010R00.sh
10m CN ADAM monitor 1	Archiver- Start-KN00XX_010ADAM010R01.sh	Archiver- Stop-KN00XX_010ADAM010R01.sh
10m CN ADAM monitor 2	Archiver- Start-KN00XX_010ADAM010R02.sh	Archiver- Stop-KN00XX_010ADAM010R02.sh
10m CN ADAM micronode	Archiver- Start-KN00XX_010ADAM010R04.sh	Archiver- Stop-KN00XX_010ADAM010R04.sh
10m SN ADAM control	Archiver- Start-KN01XX_010ADAM010R00.sh	Archiver- Stop-KN01XX_010ADAM010R00.sh
10m SN ADAM monitor 1	Archiver- Start-KN01XX_010ADAM010R01.sh	Archiver- Stop-KN01XX_010ADAM010R01.sh
10m SN ADAM monitor 2	Archiver- Start-KN01XX_010ADAM010R02.sh	Archiver- Stop-KN01XX_010ADAM010R02.sh
10m SN ADAM geochem	Archiver- Start-KN01XX_010ADAM010R03.sh	Archiver- Stop-KN01XX_010ADAM010R03.sh
10m 1200kHz ADCP	Archiver- Start-KN0101_010ADCP010R00.sh	Archiver- Stop-KN0101_010ADCP010R00.sh
10m WetLabs FLNTU	Archiver- Start-KN0101_010FLNT010R00.sh	Archiver- Stop-KN0101_010FLNT010R00.sh
10m TChain	Archiver- Start-KN0101_010TCHN010R00.sh	Archiver- Stop-KN0101_010TCHN010R00.sh
10m Seabird SBE37	Archiver- Start-KN0101_010SBEX010R00.sh	Archiver- Stop-KN0101_010SBEX010R00.sh
20m Sub ADAM control	Archiver- Start-KN0201_010ADAM010R00.sh	Archiver- Stop-KN0201_010ADAM010R00.sh
20m Sub ADAM monitor 1	Archiver- Start-KN0201_010ADAM010R01.sh	Archiver- Stop-KN0201_010ADAM010R01.sh
20m Sub ADAM monitor 2	Archiver- Start-KN0201_010ADAM010R02.sh	Archiver- Stop-KN0201_010ADAM010R02.sh
20m Sub ADAM BS48 relay	Archiver- Start-KN0201_010ADAM010R03.sh	Archiver- Stop-KN0201_010ADAM010R03.sh
20m 1200kHz ADCP	Archiver- Start-KN02XX_020ADCP020R00.sh	Archiver- Stop-KN02XX_020ADCP020R00.sh
20m TChain	Archiver- Start-KN0201_020TCHN020R00.sh	Archiver- Stop-KN0201_020TCHN020R00.sh
20m Seahorse CTD	TBD	TBD
01m Alawai NS01 CTD	Archiver- Start-AW01XX_002CTDXXXXR00.sh	Archiver- Stop-AW01XX_002CTDXXXXR00.sh
01m Alawai NS02 CTD	Archiver- Start-AW02XX_001CTDXXXXR00.sh	Archiver- Stop-AW02XX_001CTDXXXXR00.sh
01m Waikiki NS03 CTD	Archiver- Start-WK01XX_001CTDXXXXR00.sh	Archiver- Stop-WK01XX_001CTDXXXXR00.sh
01m Waikiki NS04 CTD	TBD	TBD
JABSOM WX Station	Archiver- Start-KNWXXX_XXXDVP2XXXR00.sh	Archiver- Stop-KNWXXX_XXXDVP2XXXR00.sh

1.5.2. Stopping the Instrument File Archivers

As above, each instrument file archiver can be stopped by calling a convenience script. Stop scripts follow the naming pattern `'Archiver-Stop-SOURCENAME.sh'`. As an example, to stop the 20m 1200kHz ADCP instrument driver, ssh to the Linux server in question (shore lab or campus lab) as the *kilonalu* user, and execute the following command in the terminal :

```
$ Archiver-Stop-KN02XX_020ADCPXXXR00.sh
```

This will cleanly shut down any existing 20m ADCP driver. The current Stop scripts are listed in the table above in the Starting the Instrument File Archivers section.

1.6. Understanding File-based replication

In addition to replicating data streams to the campus DataTurbine, we also mirror the archived data files in the shore station `/data` directory using a Linux mirroring tool called `rsync`. This ensures that all archived data are synchronized with the campus directory, and the data directories on the campus server are backed up to disk on a nightly, weekly, and monthly schedule. The *kilonalu* user has a scheduled cron job that mirrors the data files hourly. The cron command that is called is:

```
rsync -avt /data/kilonalu bbl.ancl.hawaii.edu:/data/raw
```

If data files that are present on the shore server are not present on the campus server within an hour, check to be sure that the cron service is running on the shore station server. To do so, issue the following command as the kilonalu user:

```
$ sudo service crond status
```

If the service is not running, either contact Chris at cjones@soest.hawaii.edu, or start the service using:

```
$ sudo service crond start
```

If file-based replication is working properly, you should be able to view the newest data files in the `/data/raw` directory by pointing your browser to https://bbl.ancl.hawaii.edu/kilonalu-data. Likewise, the web directory can be mounted on your Mac or PC if you want to drag-and-drop files to your workstation. See the screencasts at http://bbl.ancl.hawaii.edu/share/WebDAV-medium-mac.html (Mac) and http://bbl.ancl.hawaii.edu/share/WebDAV-medium-windows.html (PC) to see how to mount the Kilo Nalu data archive directory.

1.7. Managing the Matlab Instrument Plotting Code

Data streaming into the DataTurbines are queried every twenty minutes using Matlab, and the plotting code is run on the BBL campus server (`bbl.ancl.hawaii.edu`). The code can be run from just a terminal, or from the graphical version of Matlab. Either way, it is convenient to be able to view the plots within Matlab for troubleshooting, and so we run a service on the campus server called VNC (Virtual Network Computing) which allows us to connect to the server's remote desktop as the *kilonalu* user. At the moment, the 20m ADCP plotting code is run using Matlab's full desktop window, whereas the 10m SBE37, 10m FLNTU, and NS01, NS02, and NS03 CTDs call matlab from within a terminal in order to reduce the memory load on the server. Once connected to

The four Linux 'virtual desktops' are selectable in the bottom right corner of each desktop.

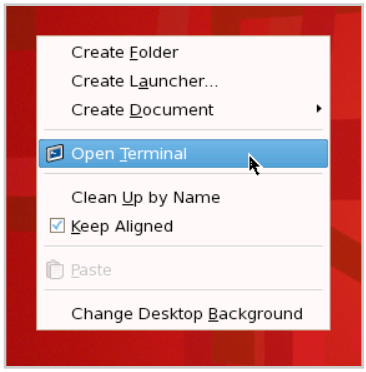
the server via VNC, you should see the Matlab window for the ADCP processing, and a terminal window with multiple tabs that are running the the plotting code for the other instruments. **Note: Linux supports 'virtual desktops', and in the bottom right corner of each desktop is a 'switcher' application. Clicking on each of the four square boxes will move you to each of the four virtual desktops.**

1.7.1. Connecting to the Server via VNC

VNC is a remote desktop application that runs as a server on Linux, Windows, and Mac OS (GoToMyPC uses it as it's foundation). You can connect to the BBL campus server using a VNC client application that runs on your workstation. Due to the way the Kilo Nalu network is configured for security, you must first create a secure 'tunnel' using an SSH client program, and then connect to the VNC server via the tunnel. Instructions for doing so using Windows XP are shown in the Quicktime screencast at http://bbl.ancl.hawaii.edu/share/Media/VNC-SSH-tunnel-BBL.m4v. Follow the instructions in this screen cast to 1) Download and install both Putty SSH and RealVNC, 2) create the tunnel using Putty SSH, and 3) Connect to the server using RealVNC. **Note: TODO - These instructions will also be updated for connections via Mac OS.**

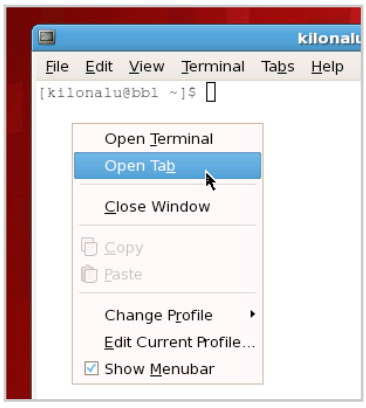
1.7.2. Starting the Instrument Plotting Code

Once connected to the *kilonalu* user's remote desktop on the BBL campus server, the Matlab plotting code can be started for each instrument stream from the Matlab source code installed in `/usr/local/bbl/trunk/src/matlab`. The following instructions apply to the 10m SBE37, 10m FLNTU, and Ala Wai/Waikiki CTDs. The 20m 1200kHz ADCP plotting will be handled differently in the instructions below.



First, each instruments plotting code will be started in a separate terminal window. Look at the virtual desktops, and find the white terminal window that has multiple tabs open. If there isn't one (e.g. after a server reboot), right-click on the red desktop and choose the 'Open terminal' menu item.

Once the terminal is open, right-click on the white terminal background and choose the 'Open Tab' menu item. Create a tab for each of the instruments that you'll be starting the Matlab plotting code.



In each of the terminal tabs, change directories to the location of the Matlab plotting code scheduler scripts, and start matlab without the graphical interface using the following two commands:

```
$ cd /usr/local/bbl/trunk/src/matlab
$ matlab -nosplash -nodesktop
```

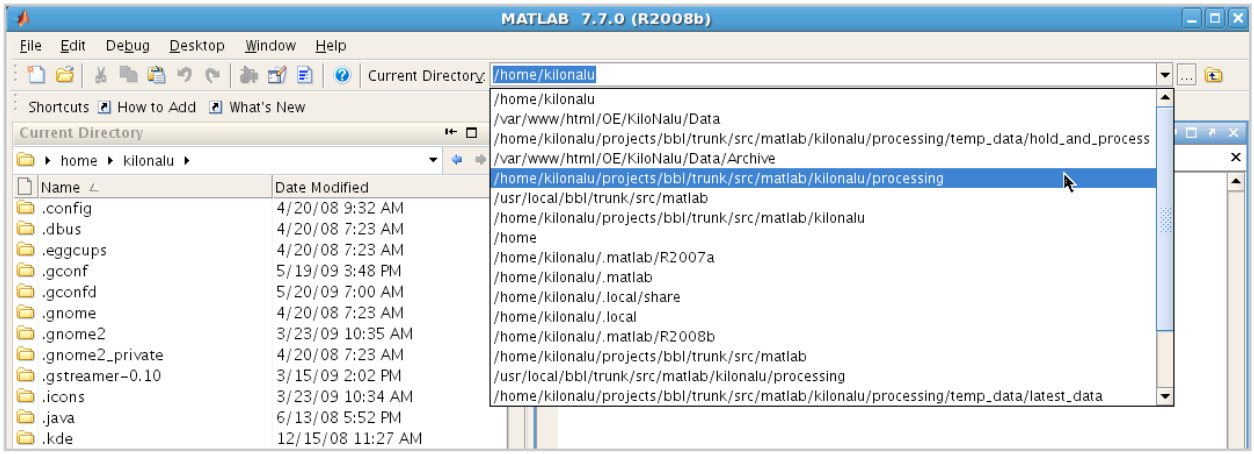
The Matlab prompt will show up in the terminal, and then start the scheduler script for the desired plotter. For instance, to start the 10m FLNTU plotting, enter:

```
>> schedule_KN0101_010FLNT010R00_processing
```

This will call two Matlab classes (Configuration.m and DataProcessor.m), and will use a Matlab timer to run the DataProcessor.process() function based on the values set in the Configuration class. Do this for each of the plotters separately in terminal tab windows. Each scheduler Matlab script follows the naming convention of `schedule_SOURCENAME_processing.m`. The list of scheduler scripts includes:

Instrument Description	Matlab Scheduler Script Name
10m FLNTU	schedule_KN0101_010FLNT010R00_processing.m
10m SBE 37	schedule_KN0101_010SBEX010R00_processing.m
01m Ala Wai NS01 CTD	schedule_AW01XX_001CTDXXXXR00_processing.m
01m Ala Wai NS02 CTD	schedule_AW02XX_002CTDXXXXR00_processing.m
01m Waikiki NS03 CTD	schedule_WK01XX_001CTDXXXXR00_processing.m

The 20m 1200 kHz ADCP plotting code is handled slightly differently. To start this instrument plotter, double-click on the Matlab icon on the kilonalu user's remote desktop. This will open up Matlab in it's graphical mode. In the Current Directory dropdown at the top of the window, change directories to `/home/kilonalu/projects/bbl/trunk/src/matalb/kilonalu/processing`. This directory contains the Matlab m-files to start the ADCP proccessing.



In the matlab command window, start the ADCP scheduler by typing:

```
>> KN_RT_2007
>> prevtim = 1;
```

This will resume the processing on the next twenty-minute interval using the summary data that are cached.

1.7.3. Stopping the Instrument Plotting Code

For each of the open terminal tabs with a running version of Matlab, stop the processing by exiting Matlab:

```
>> exit
```

Likewise, do the same for the ADCP processing in the Matlab window. The scheduler will be stopped for the instrument plotter running in each particular instance of Matlab.

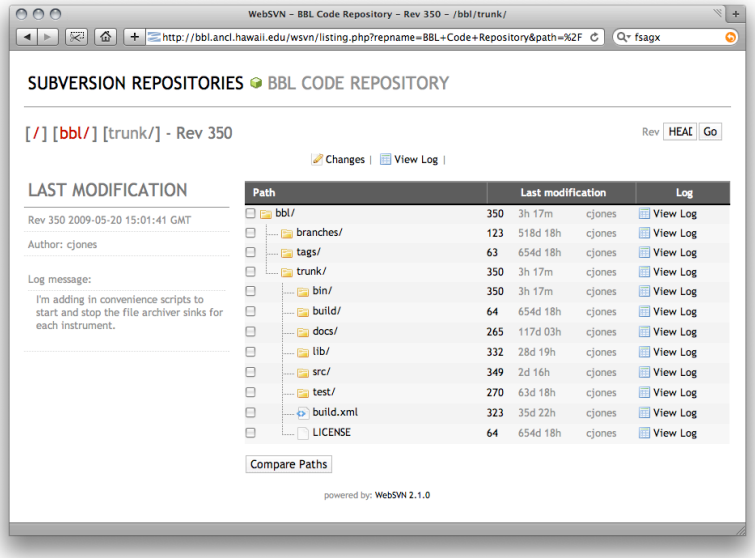
1.8. Managing the BBL Source Code and Installations

The source code for the instrument drivers, startup and stop scripts, and Matlab plotting code is all managed through a code versioning system called Subversion (see http://subversion.tigris.org/). The BBL source code Subversion repository is a shared code base that benefits from using a formal version control system to organize, document, and communicate changes to the source files. Along with the DataTurbine server software, the BBL source code is installed and maintained on both the shore station and campus servers. When changes are made to the source code repository, the new code can be updated on each of the servers using the subversion command `svn` installed on the Linux machines, or by using the graphical subversion client called eSVN. The BBL source code is installed in `/usr/local/bbl/trunk` on each server.

1.8.1. Working with the BBL Source Code Repository

The repository is a master copy of the source code, and is stored on the BBL server in the `/var/svn/bbl` directory. All changes to this directory are done using the various subversion commands or clients. The repository can be browsed by visiting http://bbl.ancl.hawaii.edu/wsvn. The code is open source, and downloadable by anyone, with the caveat that the copyright of the original authors be maintained, and that any modifications to the source code be made available to the public according to the GNU General Public License.

The Subversion system tracks all of the changes to the source code files, and stores the changes and each of the change log entries submitted by the person who made the particular change. The history of any given file can be viewed to get an understanding of who did what to a file, or to show when new files and features are added.



A screenshot of the BBL code repository web interface that allows browsing of the source code and version history of each file.

The entire source code repository can be downloaded to your local workstation using a Subversion client. On windows, a good program is TortoiseSVN (see http://tortoisesvn.tigris.org/). For the Mac, a good program is SCPlugin (see http://scplugin.tigris.org/), and for Linux, and good graphical client is eSVN (see http://esvn.sourceforge.net). The command line program (`svn`) is also available on Linux and Mac OS by default, and can be the quickest once the commands become familiar.

To make modifications to the source code, download the code (do a Subversion checkout, see below) using one of the clients above using the following repository URL: https://bbl.ancl.hawaii.edu/projects/bbl. We use a secure connection, and so you will have to accept the SSL certificate when the client connects and prompts you.

1.8.2. Modifying the BBL Source Code

Interacting with the Subversion server entails a hand full of operations for communicating with the server, committing changes to the master repository, updating your local copy of the repository, or getting information about selected files or directories. The operations are described below

1.8.2.1. Subversion checkout

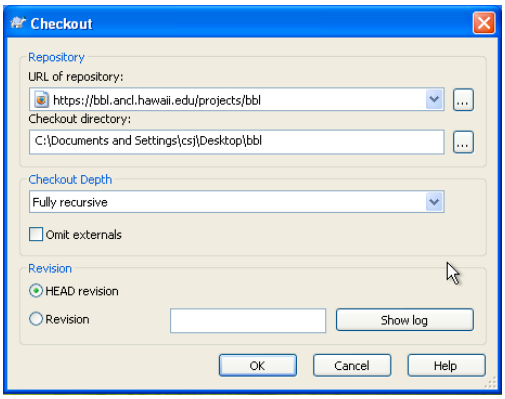
To get the download the source code from the repository, you perform an operation called 'checkout'. If you are on Mac OS or Linux as your desktop, you can open a terminal, cd to the desired work directory, and checkout the code using:

```
$ cd [checkout-directory]
$ svn checkout https://bbl.ancl.hawaii.edu/projects/bbl
```

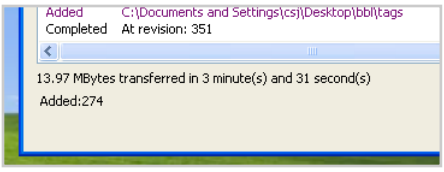
You may be prompted to accept the secure SSL certificate from the server, and you should coose the

'permanently' option. To checkout the code base from a graphical client such as TortoiseSVN (on Windows), install TortoiseSVN, restart your PC, and then right-click on the desktop, and choose the *SVN Checkout ...* menu item. In the next dialog box, enter the following values

- URL of repository: `https://bbl.ancl.hawaii.edu/projects/bbl`
- Checkout directory: `C:\Documents and Settings\[user]\Desktop\bbl`
- Checkout depth: `Fully Recursive (dropdown)`
- Revision: `HEAD revision (radio button)`



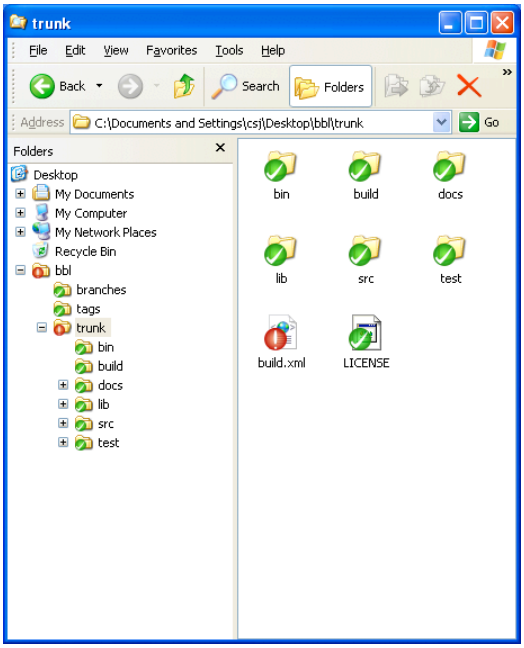
When you click *Ok*, all of the files in the repository will be downloaded to your checkout directory, and are known as your 'working copy'. When the download is complete, you should see a final message that the checkout operation completed:



Now, change directories into the new *bbl* directory that was created in order to work with the files. You'll notice three top-level folders - *branches*, *tags*, and *trunk*. The branches folder is used for experimental development of the main source code tree, and the tags folder is used to create stable snapshots of the main code tree, but at the moment we are not using those directories. The main code base is in the trunk folder (i.e the trunk of the tree of code). These names are conventions that come from versioning systems like Subversion.

1.8.2.2. Subversion status

You can get the status of any file or folder by issuing the status command (Mac OS and Linux), or in the graphical programs, the




```
$ svn status
```

status of each file or folder is shown based on the changing icon (green means that the file has not changed from the master repository version, red means it has been altered in your working copy). Try opening the *build.xml* file with your text editor (WordPad, or try downloading the free code editor named ConTEXT from http://www.contexteditor.org/). Make a minor change to the file, save it, and notice the change in the status icon, or issue the `svn status` command. To revert the

```
$ svn revert build.xml
```

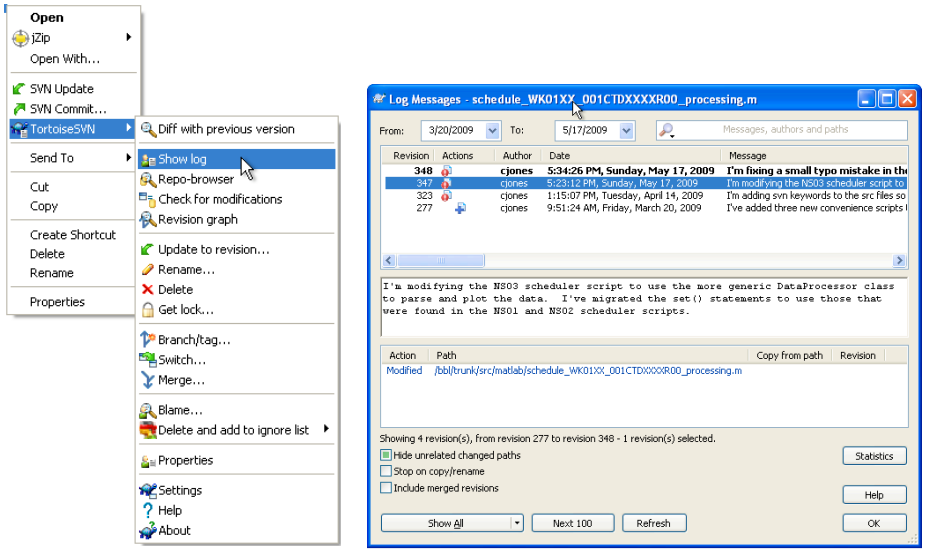
file back to the original, issue the `svn revert` command, or in Windows, right-click on the file and choose the *TortoiseSVN --> Revert ...* sub-menu item. This process is essentially the same using SCPlugin on the Mac (right click and choose *More --> Subversion --> Revert* menu item).

1.8.2.3. Subversion log

You can view the entire history of a given file by running the `svn log` command. For instance, change directories to the main matlab source directory in `trunk/src/matlab`, and type:

```
$ svn log schedule_WK01XX_001CTDXXXXR00_processing.m
```

The output of this command will show all of the log messages associated with this file, along with their revision numbers and revision dates. To do the same in Windows, navigate to the same directory, right-click on the *schedule_WK01XX_001CTDXXXXR00_processing.m* file, and choose the *TortoiseSVN --> Show log* menu item.



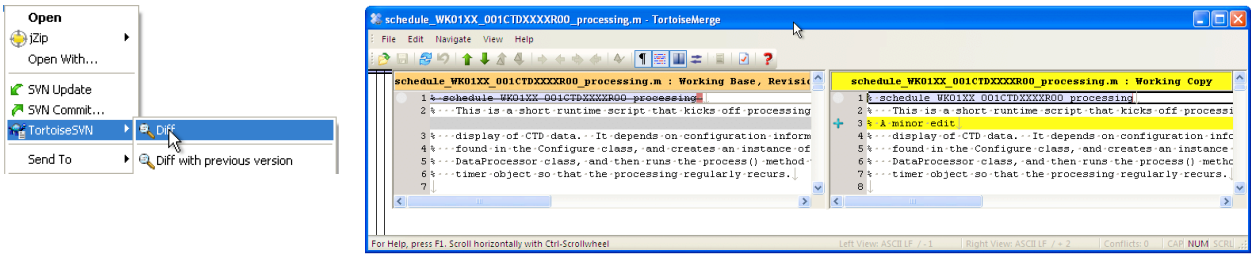
The resulting dialog box will allow you to explore the change history of the particular file. The graphical process is basically the same using SCPlugin on the Mac (right-click and choose the *More --> Subversion --> Log* menu item).

1.8.2.4. Subversion diff

To illustrate how to view the differences between a file in your working directory and the same file in the master repository, use your text editor (or Matlab) to change the *schedule_WK01XX_001CTDXXXXR00_processing.m* file. Then, if you're using the command line tools, issue the following command:

```
$ svn diff schedule_WK01XX_001CTDXXXXR00_processing.m
```

This will show the line-by-line differences of the file. Using TortoiseSVN, right click on the file and choose the *TortoiseSVN --> Diff* menu item. Likewise on the Mac with SCPlugin (*More --> Subversion --> Diff* menu item).



The dialog box will highlight line-by-line differences between the two versions (additions, deletions, etc.).

1.8.2.5. Subversion update

A subversion update operation merely updates your working copy (of a single file or whole folders) with the latest versions that have been committed to the master repository. When multiple people are working on the same code base, it's critical to keep your working copy up-to-date with the master versions. By doing so, you'll minimize conflicts and not overwrite someone elses changes. **Note: It's good practice to always run the Subversion update command to your local working copy of the code base before making any local changes and committing them to the master repository.** Subversion will flag conflicts, but it's good to avoid them in the first place. This is merely good etiquette in shared code development.

To update your working copy directory via the command line tools, issue the following commands:

```
$ cd [checkout-directory]/bbl/trunk
$ svn update
```

This will grab the latest versions of all of the files in the master repository, and will list which files have been added [A], which have been deleted [D], which have been updated [U], which have been merged with your local changes [G], and which are in conflict with your local working version [C]. If you see files that are conflicting, you'll need to reconcile the file differences manually in your file editor. To update in TortoiseSVN, right-click on the *[checkout directory]\bbl\trunk* folder and choose the *SVN Update* menu item. Using SCPlugin on the Mac, right-click and choose *More --> Subversion --> Update*.

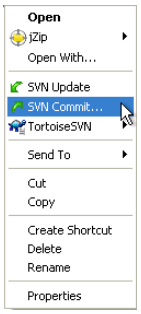
1.8.2.6. Subversion commit

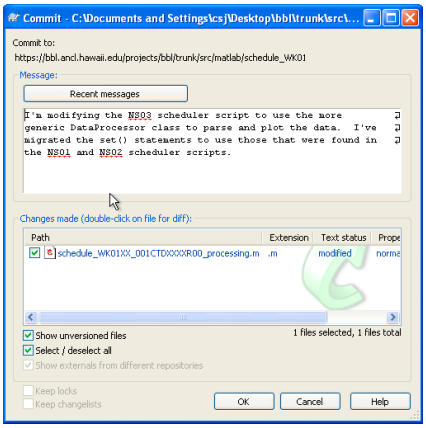
The Subversion commit operation is used to upload your changes from your local working copy to the master repository. This is often confused with Subversion update, but update pulls changes down to your machine, commit pushes changes up to the server. When you've finished modifying a source file to your liking, and it is stable (i.e. won't break the functionality of the overall code base), use the svn commit command along with a descriptive summary of the changes you've made. For instance, via the command line:

```
$ cd [checkout-directory]/bbl/trunk/src/matlab
$ svn commit schedule_WK01XX_001CTDXXXXR00_processing.m -m "I'm modifying
the NS03 scheduler script to use the more generic DataProcessor class to
parse and plot the data.  I've migrated the set() statements to use those
that were found in the NS01 and NS02 scheduler scripts."
```

Note: It's good practice to make changes, additions, or deletions to files in small, atomic, easy to understand steps. Try not to modify twenty unrelated files and commit them all at once, because it defeats the purpose of communicating the changes to others, and makes it difficult to revert the changes if something isn't correct. For instance, if you add a Matlab function, commit the function to the repository in a solitary commit with good documentation about its purpose, inputs, and outputs.

To perform the same commit above using TortoiseSVN, right-click on the schedule_WK01XX_001CTDXXXXR00_processing.m file and choose the *SVN Commit ...* menu item. This will bring up the commit dialog box, and enter your detailed commit message to document the changes. This process is the same using SCPlugin on the Mac, just right-click and choose *More --> Subversion --> Commit* menu item.



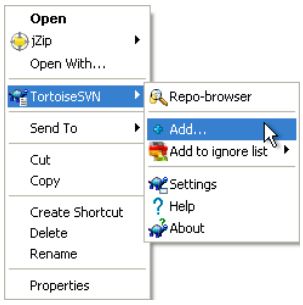


1.8.2.7. Subversion add

If you've created a new file that is part of the larger code base, use the Subversion add command to add it to your working copy. For instance, once you've finished writing a new `schedule_AW03XX_001CTDXXXXR00_processing.m` file, commit it to the repository with the following commands:

```
$ svn add schedule_WK02XX_001CTDXXXXR00_processing.m
$ svn commit schedule_WK02XX_001CTDXXXXR00_processing.m -m "I've added a new
scheduler script for the NS04 CTD located on Waikiki Beach.  The script
first creates a Configuration class, and calls the set() function on each of
the properties needed to configure the plotting for the instrument data
stream.  It then calls the DataProcessor class with the configuration
variable as it's input.  Finally, it creates a Matlab timer, and calls the
DataProcessor.process() function on a twenty minute schedule to process and
plot the newest data from the Data Turbine."
```

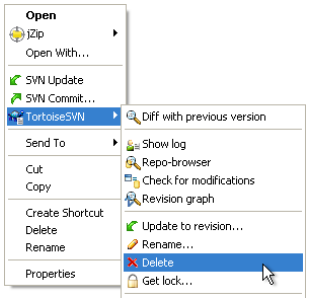
Again, the same can be done using TortoiseSVN. Right-click on the new file and select the *TortoiseSVN --> Add ...* menu item. The file icon should change to have a blue plus in front of it indicating that it has been added to your local working copy, but needs to be committed to the master repository. Right-click on the file again and choose the *SVN Commit ...* menu item to commit the changes, and add the commit message to the dialog box. The same process will work using SCPlugin on the Mac. Right-click on the file and choose *More --> Subversion --> Add*, and then, again, *More --> Subversion --> Commit*.



1.8.2.8. Subversion remove

Lastly, if you've added a file to the repository and decided that it was a mistake, or if a file is no longer pertinent to the code base, remove it using the Subversion remove command. For instance, in the case of the scheduler script above:

```
$ svn remove schedule_WK02XX_001CTDXXXXR00_processing.m
$ svn commit -m "I've mistakenly added this file and am
removing it."
```



Using TortoiseSVN, right-click on the file and choose *TortoiseSVN --> Delete*, and then right-click on the containing folder and choose the *SVN Commit ...* menu item. For SCPlugin on the Mac, right-click on the file, choose *More --> Subversion --> Remove*, and then *More --> Subversion --> Commit*.

1.8.3. Installing the BBL Source Code

On each of the servers, the source code should already be installed in `/usr/local/bbl/trunk`. In the event that the code needs updating, cd to this directory on each of the servers, and update the code with the following commands as the *kilonalu* user:

```
$ cd /usr/local/bbl/trunk
$ svn update
```

In the event that there are changes to the Java code that need to be installed, additionally use the following command:

```
$ ant clean compile
```

This will remove the old code and compile and install a new version of the Java code.