Technical Report NEESgrid-2004-29
www.neesgrid.org

Draft Whitepaper Version: 1.0.1
Last modified: September 7, 2004

Connecting your DAQ to the Data Turbine

Paul Hubbard[1]

[1] Argonne National Laboratory

Feedback on this document should be directed to hubbard@mcs.anl.gov

Acknowledgment: This work was supported primarily by the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program of the National Science Foundation under Award Number CMS-0117853

Table of Contents

1. Summary

This document explains how to interface your data acquisition system (DAQ) to the Data Turbine as used in the NEESGrid software distribution. It covers the reference software, direct interfacing, diagnostics and related issues.

2. Revision History

Initial release 1.0 August 25 2004.
1.0.1 September 7, 2004 – corrected TR number from 39 to 29.

3. Introduction – the Role of the Data Turbine in NEESGrid

The Data Turbine, sometimes referred to as 'RBNB' for Ring-Buffered Network Bus, is a commercial Java-based program that handles data streaming, archiving, synchronization, plugins, and many other features. It's free for non-commercial use, and we've negotiated and OEM redistribution license and some enhancements as well. It's a replacement for the NSDS in v3.0.

Due to its power and flexibility, is continues to assume a larger role in the NEESGrid code. For example, we now have a program that will assemble video into a DAQ-synchronized Quicktime movie based on DAQ events!

Basically, all streaming data in NEESGrid goes through the data turbine. There's an exception for control, in that NTCP does not use the turbine yet, but that's changing too.

There are several other reference documents covering the Data Turbine, and I encourage you to read them.

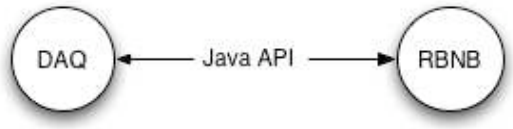
4. Terminology

The NEESgrid reference platform is LabVIEW from National Instruments. We provide example code in CVS that implements the functionality in this document.

For this document 'driver' means our Java program called DaqToRbnb.

5. Direct Connect Method

There are two methods for streaming data, depending on how your system is designed. If you have code in Java, or can call it (e.g. Matlab), then using the Creare API is probably the easiest. We'll call this the 'direct' method:



For direct connections, please see the Creare documentation at http://outlet.creare.com/rbnb/WP/index.html

Please also see http://www.mcs.anl.gov/neesgrid/dtpatterns/ for more information on naming conventions, metadata and such. This will help ensure that your code plays well with the other DAQ implementations and data consumers.

6. Indirect connect 1: Custom driver

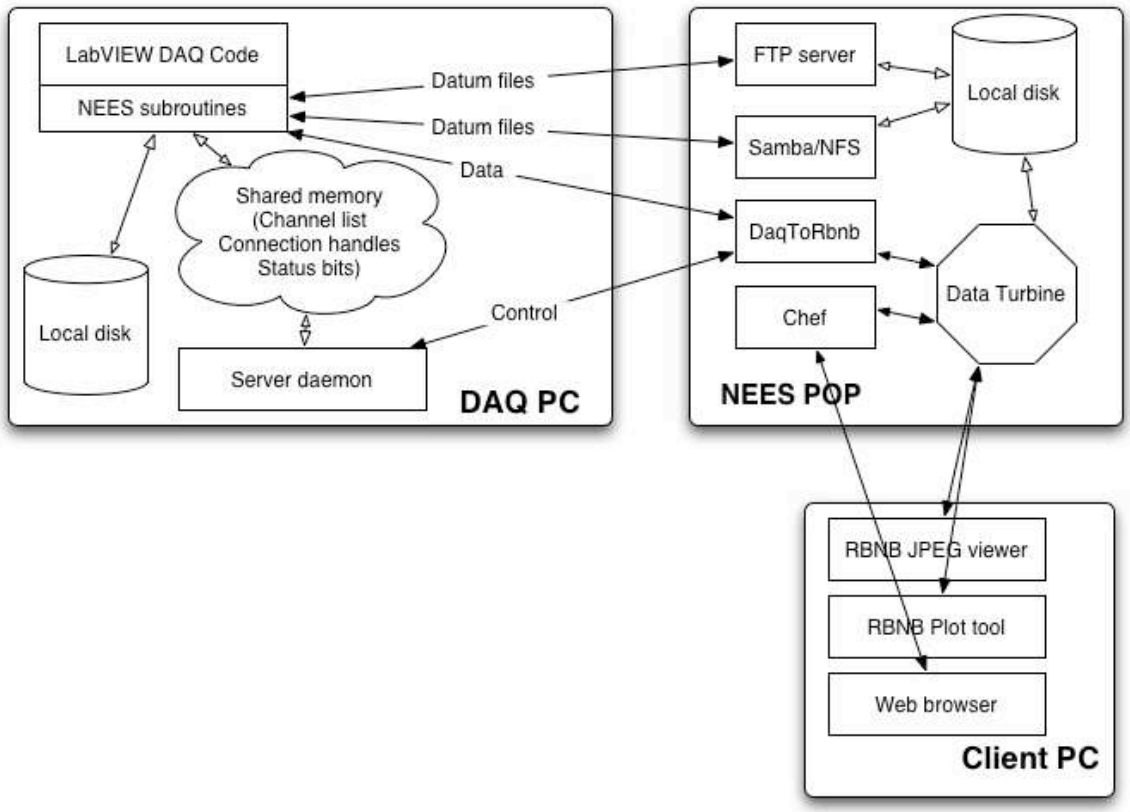
If you can't call the Creare API, then you have to have a process in the middle to mediate. Something like this, as used with http://www.mcs.anl.gov/neesgrid/orb2nsds

This relies on you writing a driver program that knows both your local protocol and the Creare API. The same naming conventions as 'Direct Connect' apply here as well.

7. Indirect connect 2: DAQ protocol and DaqToRbnb

Within the indirect method, we have another category. The previous versions of the code used a simple ASCII over TCP protocol, and we've created a new Java program called 'DaqToRbnb' that uses the same protocol to write to the data turbine.

A full diagram of this in practice, as used with LabVIEW:

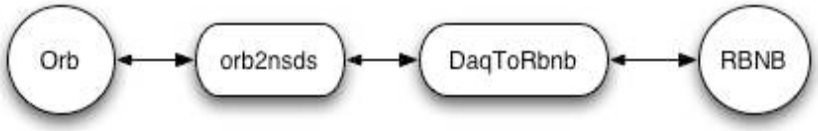


More information (protocol spec, validation tools, etc) about the DAQ protocol can be found at http://www.mcs.anl.gov/neesgrid/v3daq/daq-spec.html

If you are using the existing DAQ protocol, the DNDTester application, found at http://www.mcs.anl.gov/neesgrid/dndtester, will test and validate your DAQ implementation for correctness and completeness. Its use is strongly recommended.

8. Indirect connect 3: More Layers as Required

There are cases, such as orb2nsds, where two indirection layers are used:



I don't recommend it, since complexity is hard to debug and use, but it is possible.

This should give you some ideas as to the variety of methods you can use to connect to NEESGrid.

9. Code Access, Mailing List and Bugzilla Archive

CVS access to the NEESgrid code archive is documented at http://www.mcs.anl.gov/neesgrid/cvs.html. A Bugzilla archive for logging and tracking bugs and feature requests is available at http://bugzilla.ncsa.uiuc.edu/neesgrid. There is also a mailing list; instructions can be found at http://www.mcs.anl.gov/neesgrid/

More information about our use of the Data Turbine can be found at http://www.mcs.anl.gov/neesgrid/turbine/ and http://www.mcs.anl.gov/neesgrid/dtpatterns/

10. Post-experiment FTP and Samba uploads

There are two methods of remote data access, streaming and batch. So far, we've looked at the mechanisms for delivering streaming data. However, if you look at the DAQ examples, you will find the code to do post-experiment data uploads. We use the National Instruments Internet Toolkit, specifically the FTP transfer routines. Post-experiment, we FTP the data file to the NEES-POP or other specified destination. Once that's complete, we write a semaphore file named {datafile}.written. This triggers its transfer into the metadata handling system. The repository upload is handled by the NFMS upload agent.

11. Testing, Benchmarking and Profiling

Once you have assimilated the design and replaced portions of it, you will need to test your components. There are several programs in the archive to assist with this.

a) fake_daq.c A simplest-case data source, useful for testing the current driver and NSDS. Useful as an end-to-end test, and possibly for benchmarking. You can set the sample rate from the command line, and the number of simulated channels is a compile-time constant. Also useful for C/C++ programmers as a codebase when interfacing your DAQ to the NEESGrid.
b) NSDS Simulator.vi LabVIEW code to emulate a normal set of operations – query DAQ status, list channels, subscribe to a requested channel, and plot the streaming data as it arrives. A good test of end-to-end functionality, it also has the side benefit of plotting the data, which is often very telling.
c) NSDS Stress Tester.vi LabVIEW code that is created for benchmarking and stress testing. It subscribes to all listed channels, which is useful.
d) Stress test Fake DAQ.vi Generates as many channels of simulated data as you request.
e) DNDTester will exercise any DAQ implementation and test its protocol correctness and compliance.
f) RBNB Admin is a good way to view what channels are present in the Turbine, show listeners, and so forth.
g) RBNB Plot can also show live server statistics

Note that the Server Daemon and NSDS simulator programs display the commands and responses on their front panels; this is useful for checking and response command syntax.

Unfortunately, the Chef portion of the system does not yet have decomposition and testing tools made available. Until those are present, I recommend you use the above list to test your new components.

Profiling can take several forms: network utilization, CPU usage, sampling rate, etc. Different tools are appropriate for each; here are some I've found useful so far.

a) LabVIEW has an excellent profiler (Menu is Tools/Advanced/Profile VIs) that is quite useful in locating bottlenecks.
b) If you have Windows 2k, NT or XP, the Task Manager is useful for checking CPU, memory and network utilization.
c) On the NEES-POP, standard Unix tools such as top and lsof are invaluable for monitoring the driver and NSDS.

12. Where to Begin?

Having read this far, you are probably a bit adrift in new acronyms and design flotsam. Check out the fake_daq package from CVS, and browse through fake_daq.c. It implements much of what your system will need, and is extensively commented. This code is documented in the html subdirectory, with auto-generated Doxygen pages.

Note that you will also need to check out and compile the 'flog' message library and 'nsds-driver' before you can compile fake_daq. (nsds-driver has utility routines used by fake_daq)

Best of luck!