One Year with NOAAPORT

A guide to reception equipment and processing software

Jose F Nieves

Department of Physics, University of Puerto Rico, Rio Piedras, Puerto Rico

NOAAPORT is a system that provides to end users a one-way communication of environmental data via C-band satellite. This article describes the equipment that is required to receive this data, the software to process it and how it can be used.

Introduction

NOAAPORT is the name of a system of the US National Weather Service (NWS) that provides end-users with a one-way broadcast communication of weather and environmental data via C-band satellite. The weather data is collected by GOES geostationary satellites and various NWS facilities and is then processed to create the so-called *products*. These products are then transferred to the Network Control Facility (NCF), which routes them to the Master Ground Station through one of four *NOAAPORT channels* for uplink and then broadcast to the end receiving stations (figure 1).

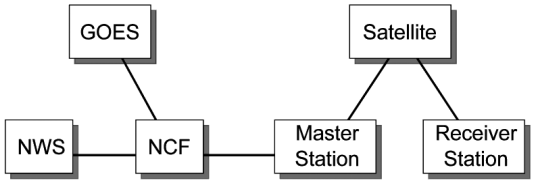


Figure 1 - Block diagram illustrating the ow of data in the *NOAAPORT* system.

During the first months of 2005, the final steps were completed to replace the old system with the new one that is based on Digital Video Broadcast-Satellite (DVB-S) and open standards. For example, the radar and satellite products are broadcast in compressed form using the open source software algorithm and library, *ZLIB*. The open specifications of the system allows an end user to put together a receiving station from off-the-shelf parts—for a fraction of what it used to cost a few years ago.

I am, by training and profession, a theoretical high energy physicist. My interest in the *NOAAPORT* technologies was aroused as a result of having served as advisor of one of our students, who was, and still is, employed by the local office of the NWS, but wanted to finish his Master's degree thesis. He had external advisors to guide him in his project, and my responsibility was officially to serve as his sponsor in our department.

I became aware of the evolution of *NOAAPORT* sometime during 2004. Early in 2005 I learned from some discussion groups that the signal could be received in my location (San Juan, Puerto Rico), although nobody was using it. Very little was generally known about the system and it was not clear, at least not to me, how to proceed from there. That is when I decided to put my own dish. I convinced myself that the signal was being received and was usable, and then, in March 2005, embarked on the project to write the software to receive and process the data. On April 9, 2005 I registered the *noaaport.net* domain and web site and, a few days later, announced the first experimental version of the *NOAAPORT Broadcast System Processor* (NBSP) software.

Since then, for almost one year now, I have been working constantly on the development of the system. It has reached a mature level that allows it to be used in several ways and in a variety of circumstances. In the rest of this article I will describe, based on my own experiences, the equipment that is required to receive the *NOAAPORT* data, the computer hardware and software required to process it, and how these can be used.

Receiver Equipment

The *NOAAPORT* signal is transmitted on a C-band frequency from the GE Americom GE-4 satellite, also known as AMC4, which is located at 101° west longitude. To receive the signal, a C-band dish, with its associated LNB and an appropriate DVB-s receiver are required.

Dish

At the moment, I use a 10 foot diameter *Sami* mesh dish. There have been lots of discussions regarding the possible dish types and sizes to use. The consensus seems to be that the smallest dish that can be used reliably is the 10 foot diameter one. A question that is often asked is whether a 7 foot dish could be used. Reports from those who have tried and tested such dishes show that they do not give a signal that is strong and stable enough, mainly due to interference from adjacent satellites. Regarding the type, a solid dish should perform better than a mesh one.

This type of mesh dish used to be common in the US before the smaller Ku-band dishes became popular. There are reports from people who have acquired a used dish from someone who essentially wanted to clean his backyard. I bought mine from a local satellite dealer for about $500. The 10 foot dish was the largest model that they had in stock, and although I was not completely sure it would work, I had no other choice at that time but to try. Therefore, while a larger and/or solid dish should perform even better, my own experience is that the 10 foot mesh dish does yield a very stable signal. While the LNB is of course a central piece, which particular model is used is not as critical as choosing the dish size. I bought the best LNB that my local dealer had at the time for about $100. I also bought an actuator (motorized arm), for about $100, to be able to move the dish by remote control.

Receiver

The other required component is a DVB-s receiver. While in principle there are various possibilities, the best choice in terms of the performance/cost ratio is the *S-75* receiver from *Novra*, a Canadian company (figure 2). This receiver, which costs around $350, is available from satellite equipment dealers, and it can also be ordered directly from the company, which is what I did. There are are receivers that are more expensive, used mostly in commercial *NOAAPORT* installations, and there are also PCI cards that can be installed inside a computer.

There are many varieties of PCI card, some expensive and some considerably cheap. But there are several problems with such cards. First, they require drivers and one has to be sure that the vendor supports the operating system that one

Figure 2 - The Novra S75 receiver measures just 5¼ inches wide

intends to use. Secondly, it is not advised to install the card in the same computer where the *NOAAPORT* processing software will be installed. Processing the *NOAAPORT* data is very resource consuming, and using the same computer as the receiver and processor is a recipe for disaster. The PCI card could be an option only if one is to use a dedicated computer as the receiver. However, in this case I would opt for one of the commercial receivers, which are possibly built by using one such card in a small dedicated computer, with a custom operating system, all integrated and well tested.

Dish Installation

The big hurdle was the installation and alignment of the dish. It was to be installed on the roof of the three-story building where I live, which is in the university campus, a few hundred meters from my office. I had contacted a dish installer to help me, but all I wanted from him was to install the pole on the roof to make sure it was absolutely upright and fixed. That saved me $200. Actually he helped me a little more, by assembling the dish and mounting it on the pole. All the adjustments and alignment remained to be made, the next day.

I had been looking for instructions about how to align the dish, and out of the many that I found, one stood out. The website

http://www.geo-orbit.org

is full of useful information, and in particular the page

http://www.geo-orbit.org/sizepgs/tuningp2.html

had, I thought, the clearest explanations. That night, after having seen the dish, all the diagrams started to make much more sense. I spent a few hours making my own diagrams and angle calculations, and writing down which angles I had to measure and set.

I had bought an inexpensive receiver for free TV channels, and also a box to control the actuator, both from an on-line store. The next day, I took the control box to the roof and connected it with a 100 ft long power cable to the first floor. I followed the instructions on that web page almost verbatim, complemented by my own notes. Essentially, it involves moving the dish to its highest point along the arc, measuring all the angles and adjusting the dish there. If that is done correctly, using the actuator, the dish will track all the satellites along the arc. After completing those steps, I moved the dish full swing back and forth with the actuator a couple of times and remeasured and readjusted the angles once more.

I threw the cable from the dish to the TV receiver in the first floor then moved the dish to point in the direction of the AMC-4 satellite. I went downstairs to program the receiver, set it to one of the strongest TV channel signals from that satellite, and began to play with the actuator control box. After a few moves, I was watching TV. With the help of a signal, meter I tweaked the dish a little and readjusted the LNB to optimize the signal.

Receiver Configuration

The *Novra* receiver has two connectors on the back, one for the cable from the dish and the other for an *ethernet* port. The configuration is accomplished by a simple program provided on a CD that runs on *Windows*. The parameters to be set are the *frequency* and *symbol rate* of the *NOAAPORT* signal, and the packet identifiers (PIDs) for the four *NOAAPORT* channels, 101 - 104.

The ***symbol rate*** is 6.349 Msps (megasymbols per second). The downlink frequency of the satellite is 3956.5 MHz, but that is **not** the number to enter in the configuration screen. A typical C-band LNB local oscillator (LO) frequency is 5150 MHz. The LNB converts the downlink frequency to the receiver frequency as follows:

5150 - 3956.5 = 1193.5

and that is the value that must be entered in the *Novra* receiver. I mention this because I have seen at least one PCI card that requires the downlink frequency to be entered rather than the receiver frequency in the configuration program. With this configuration in place, the software receive application must be set up to receive packets from the following multicast address/port combinations:

- 224.0.1.1 - 1201
- 224.0.1.2 - 1202
- 224.0.1.3 - 1203
- 224.0.1.4 - 1204

which correspond to the four *NOAAPORT* channels.

Connection to the Network

The receiver can be connected to the processing computer in two ways. In one case, with a computer that is equipped with two ethernet ports, one port can be connected directly to the receiver with a crossover ethernet cable and the second port connected to the rest of the network. The other way is to connect both the computer and the receiver to an *ethernet* switch. Since I use my system for development purposes, and that requires various computers to be receiving the signal simultaneously, I use the second option. Thus, I created a sub-network where I have several computers and the receiver connected to the same switch. By means of a very small application that I had written just for this purpose, I verified that the packets were being received in the four channels in all the computers.

Computer

Since the choice of the computer must be dictated by the software that one plans to use, let us discuss first the software options.

Software

The software I use is *NBSP* (*NOAAPORT* Broadcast System Processor), which I have been developing for this purpose. Before I started the development of this program, the only software generally available for receiving and processing the *NOAAPORT* data was the *LDM* (Local Data Manager) package from *Unidata* (Unidata Program at the University Corporation for Atmospheric Research, housed in the University of Colorado. *http://www.unidata.ucar.edu*). To be usable with *NOAAPORT*, *LDM* is complemented by installing the *Gempak* package which is also maintained by *Unidata*.

Gempak is a large collection of programs for decoding and displaying the received data. *LDM* and *Gempak* form a complete and powerful combination, and it can be said that it is the de facto standard. For various reasons, I would not be content with that setup. *LDM* is a general-purpose data capture and distribution system, independent of how or where the data comes from. As already mentioned, for the

NOAAPORT application, it must be coupled to a tool that gets the data from the receiver and sends it to the *LDM* system for processing, following the *LDM* protocol. Thus, while that combination yields a complete *NOAAPORT* receiver software, it is what I call *Gempak-centric*; that is to say, the assumption is that you will be looking at the data using the *Gempak* tools, and therefore the output of the whole system is governed by the *Gempak* tools' formats. For example, satellite images are not stored directly as png, gif or jpg files that can be viewed with ordinary programs, including a web browser. That requires further processing and/or configuration. Furthermore, *LDM* and *Gempak* run only on UNIX or Linux operating systems.

I wanted something that was not focused on *Gempak,* and which could output the files in ready-to-read formats that could be viewed and distributed using standard tools. I also wanted to incorporate facilities for distribution by various means (e.g., email, web) and to produce files that could be imported by other applications such as *Digital Atmosphere*. Trying to adapt the *LDM-NOAAPORT* system to do something it was not designed for seemed like the wrong path to follow. For example, one has to learn a new and rather specific language syntax to modify the *LDM* configuration file *pqact.conf*. I wanted an extensible system with support for a run-control script facility that was based on a full, general-purpose language like *TCL* and *Perl*.

Therefore, *NBSP* was designed from the ground up with these requirements in mind. In general, files would be stored in their original form as they were sent. Text files (e.g. bulletins) could be viewed with any text editor, satellite images with png/jpg/gif viewers, such that they could be sent through the network, by email, by news software, and similar means, without needing further tools to process them. If anyone was to be content with just getting the text files and satellite images and putting them up in a web site, it could be done without installing any other external programs. Output could also be in a *Gempak* format mimicking what *LDM* would do, so nothing is lost since all could be done at the same time.

To meet the goals it was decided to code *NBSP* using what is called a '*thread programming model*'. Each thread executes a line of instructions that are independent of those in other threads. For example, there are four reader threads, each one reading the data from just one of the *NOAAPORT* channels: a processing thread, a network server thread and so on. I will explain later in more detail what *NBSP* can do and how it can be used. For the moment it suffices to mention that the threaded model is very effective when the program runs in computers with more than one CPU, since the threads can run in parallel.

When designing any *NOAAPORT* processing software, as well as choosing the computer hardware, it is useful to keep in mind the rate at which data is received from the *NOAAPORT* system. On average, about 20-30 complete products are received per second, summing about 500-1000 kb/second. Even without any significant post-processing, collecting all the frames for all the products at this rate already puts non-trivial requirements on both the computer software and hardware; these of course increase further very rapidly once the data is collected and saved, depending on the post-processing involved.

Hardware

Either with ldm-gempak or NBSP, it is recommended to setup one processing computer, and one or more client computers able to access the data for analysis and display. In the case of the ldm-gempak setup, the client computers would also run Gempak. With NBSP, the clients can also be windows PC's running, for example, Digital Atmosphere or similar programs.

For the reasons mentioned above, my production *NOAAPORT* server is a modern dual-processor computer running the *FreeBSD* operating system and the *NBSP* software. Because it is configured to process and distribute the files in several different ways simultaneously, it has various disks connected to a RAID card. It's cost was around $2000. In the *NOAAPORT* sub-network I have various client computers, including a workstation with *Gempak*, a *Windows* PC, and sometimes a laptop. Because my system is mostly a development system, apart from the development machine I have a one-processor server that I use for stress-testing the software during the development cycle. At present, the main server processes the files and saves them simultaneously in a *Gempak* format, in a format suitable for importing into *Digital Atmosphere*; in a plain text or image format for web browsing; and feeds them to an NNTP server process for distribution to the outside world. A less expensive server could be used but it would not be possible to enable all these features at the same time.

NBSP

Shortly after installing the receiving equipment, I started to design the *NBSP* software. In march 2005, a student strike in our University disrupted the classes for a couple of weeks. I decided to take all that month off my regular research duties to code a prototype of *NBSP*. As already mentioned in the introduction, I announced the first experimental version in April that year. Since then I have been gradually improving the internal workings of the program in many ways, while maintaining the original set of features and goals. Below I describe the features supported and the current status of the program, and then how to set it up and use it.

Features

NBSP is free and distributed under a Berkeley style license. It is developed in a machine using *FreeBSD*. Whenever a new version is ready, I make available compiled binary packages for *FreeBSD*, the Fedora Core distribution of *Linux* and the *Cygwin* environment in windows. The *FreeBSD* and *Linux* packages are installed with the native package management tools of those operating systems. I support the package installations but do not support installation from the sources. The software and the manual are available from the web site:

http://www.noaaport.net.

Any *NOAAPORT* software must be able to receive the data and save it in some form. Before transmission, the *NOAAPORT* system divides each product into a set of frames, each being no larger than 5150 bytes. Some products fit into just one frame: but most products take several or many frames and a few are comprised of several thousand frames. The main job of the receiving software is to collect all the frames that comprise a product, build the product and decide what to do with it.

When *NBSP* starts, it spawns various threads, each with a specific function. Each of the four reader threads begins to read frames received from one of the *NOAAPORT* channels. When any of the readers receives the final frame of a product, it notifies this and passes all the frames to a separate thread for further processing. There is only one processor thread. The processor verifies that all the frames have been received, builds the product, saves the raw data file to the *spool* directory, then notifies two other threads that a new product has been received. One of these is a network server that listens to network connections from clients requesting data. For lack time and space I will not describe this capability any further here. The other thread is a *filter server*.

A *filter* is just a distinctive name for a program that can take a raw data file and process it further. A filter typically looks at the data file and, according to the data type, it passes the data through yet another program or *decoder* to decode it and

produce a human viewable file. Thus, a filter can be thought of as a 'decoder dispatcher'. Any number of filters can be enabled in *NBSP*. When the filter server is notified that a new product has been saved to the spool directory, it looks at the list of enabled filters and executes each of them.

NBSP comes with several filters installed, although not all of them are enabled by the default package installation. For example, the *Gempak* filter takes the raw data files and outputs the processed data in what I call a 'Gempak-friendly' format. That is, the files are saved in a directory layout and in the file format that the *Gempak* tools expect. However, that filter is not enabled by default because it requires the installation and configuration of the *Gempak* decoders. Nevertheless, if that prerequisite is met, enabling the *Gempak* filter is a matter of changing one optional parameter in the *NBSP* runtime configuration file. For similar reasons, several other filters are installed but not enabled by the default package installation.

There are two filters that are enabled by the default installation. The *Rstfilter* processes text products and satellite image data. The *Dafilter* processes and outputs the data in a format that can be imported by the *Digital Atmosphere* (DA) program. If the facilities provided by these two filters are not desired or needed, either one or both can be disabled by changing a parameter in the *NBSP* runtime configuration file. For the purposes of this article, let us assume that *NBSP* has been installed in its default configuration by the package management tools, so that those two filters are enabled. What do we get?

Default Configuration

In the default configuration the processor saves the raw data files in the directory

/var/noaaport/nbsp/spool

The *Rstfilter* populates the directories

/var/noaaport/data/nbsp/txt

/var/noaaport/data/nbsp/sat

while the *Dafilter* populates various subdirectories under

/var/noaaport/data/digatmos

There are various ways in which these data can be used. Let's look at some of them. At this point I will go to my laptop, named *CPQ*, which is connected to the same network switch as the NBSP server (named 'Diablo'). Figure 3 shows a view of the contents of the

/var/noaaport/data

directory referred to above. Remember, all these directories are in the server, and the files are saved there as well. We are viewing them from the laptop, as if they are saved in the laptop's disk. For all practical purposes, they behave like local files in the laptop.

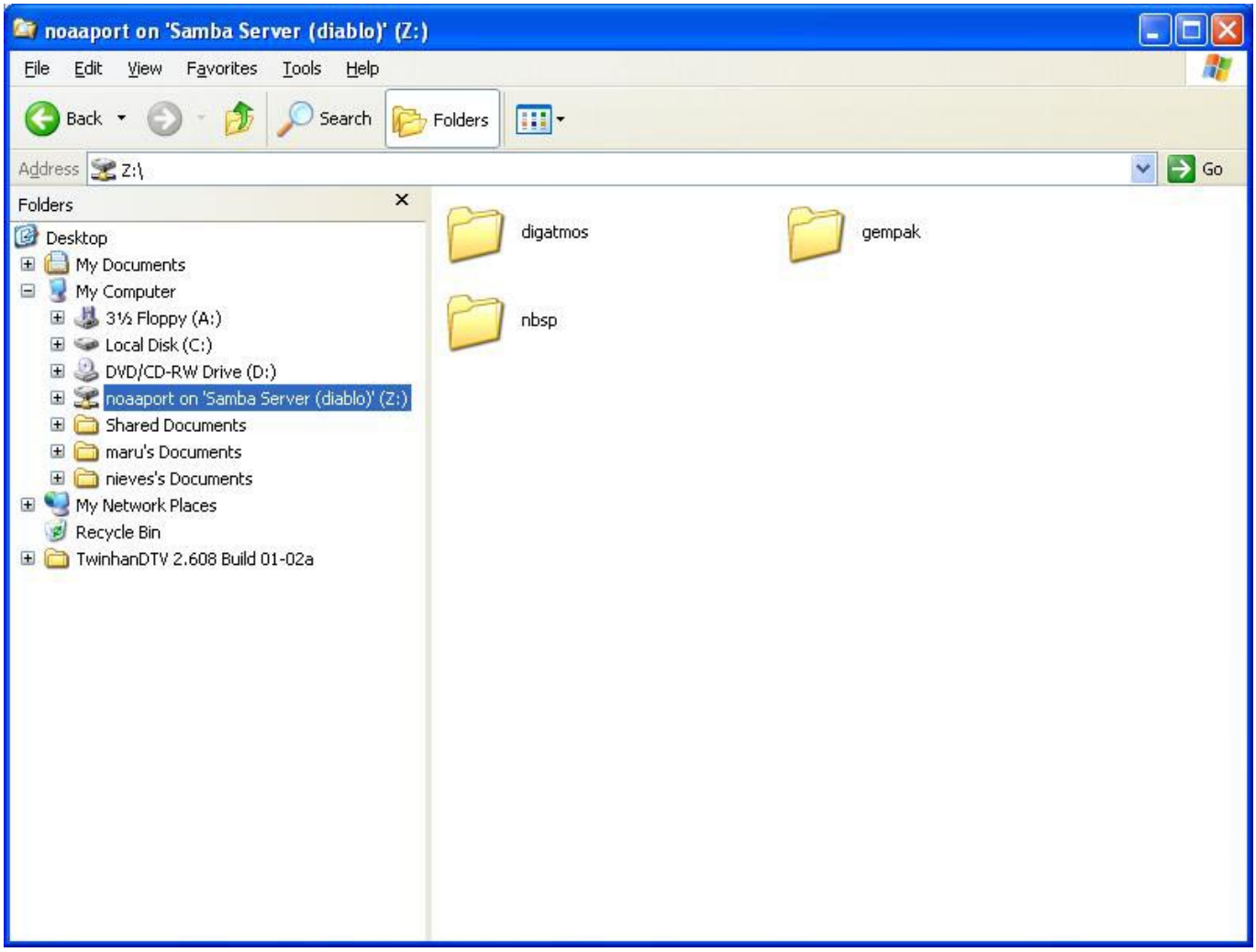
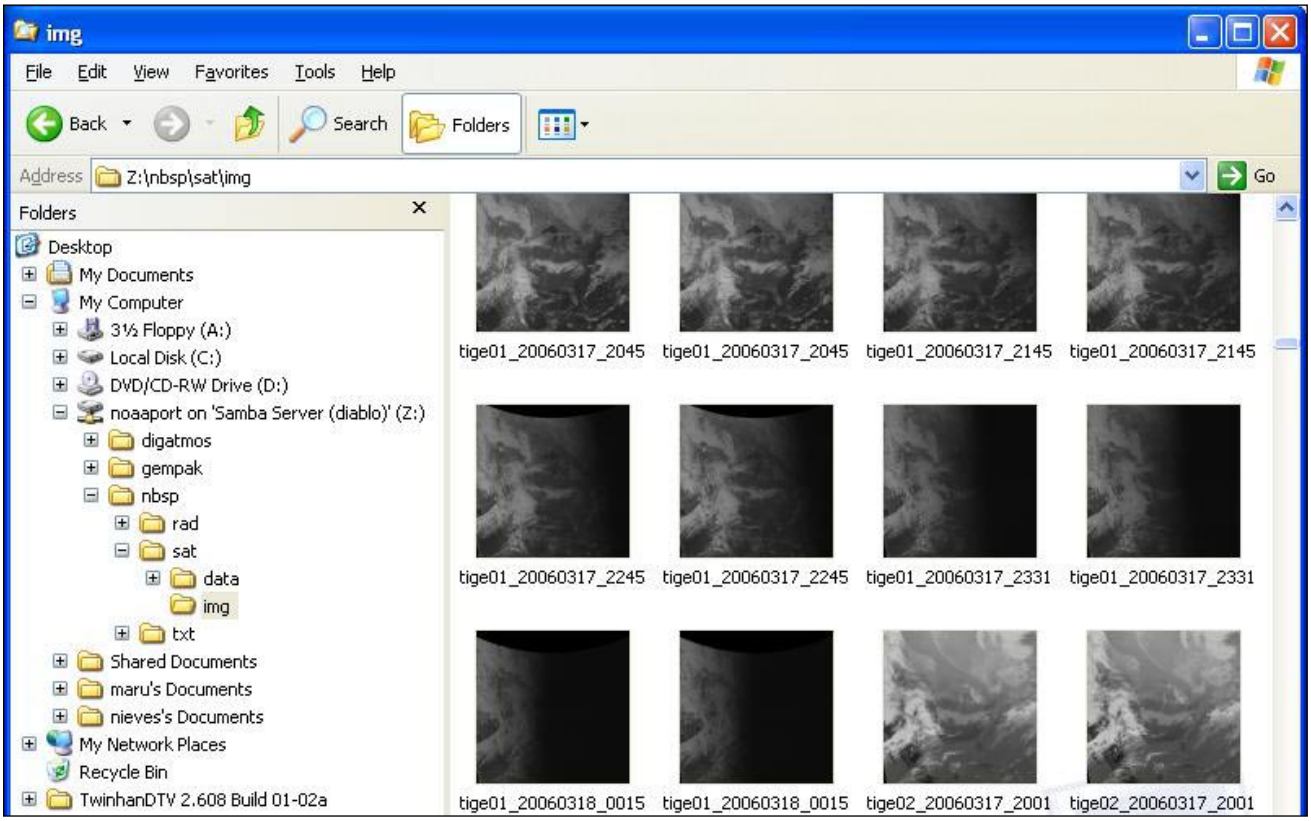


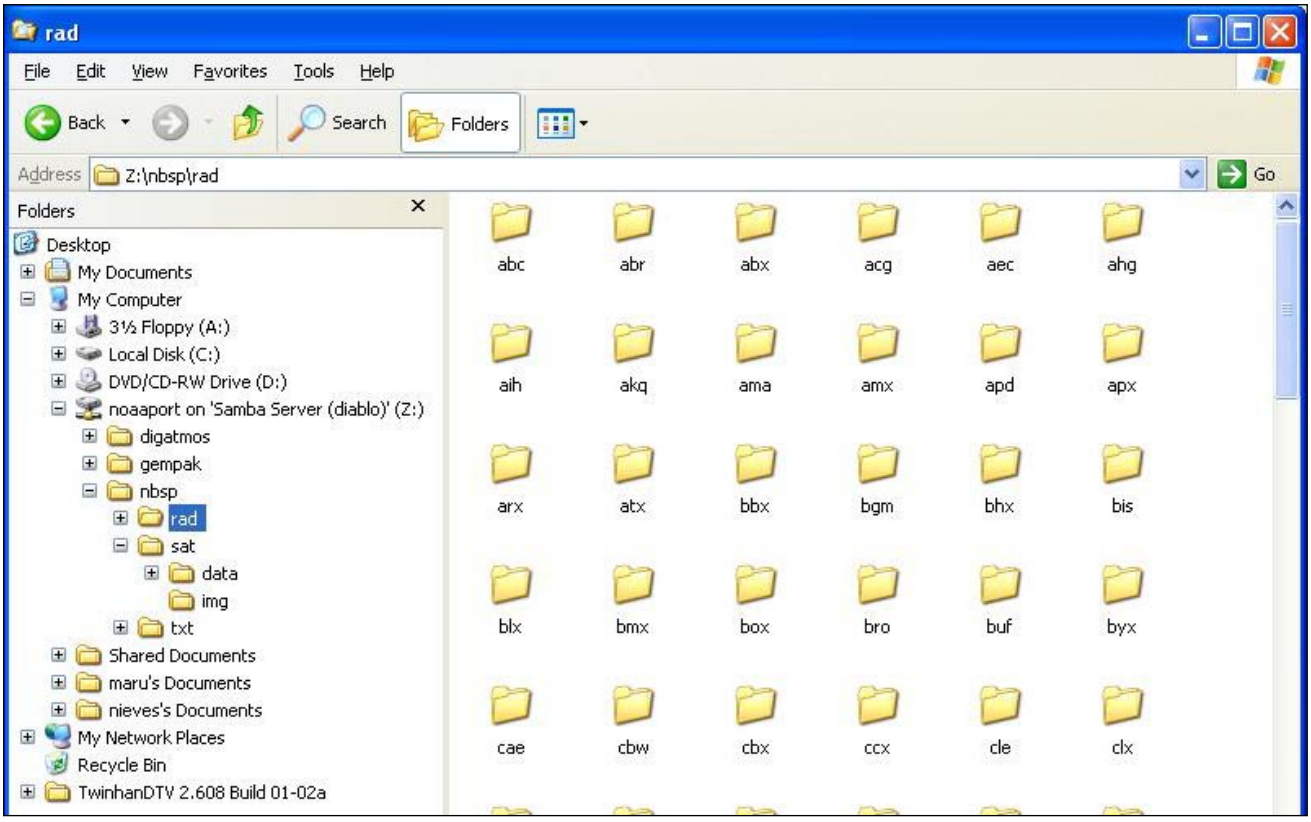
Figure 3 - A screenshot from the laptop showing the contents of the /var/noaaport/data directory in the server

What has been done here is to export, by means of the Samba software, the *NOAAPORT* data directories in the server. It is like a share in windows parlance. Once that is done, a large window to the data is opened. That directory is mounted in the laptop as drive Z. As figure 3 shows, the *NOAAPORT* data directory contains three subdirectories. The '*gempak*' subdirectory contains the data files in a format suitable for use with the *Gempak* tools; they can be used from a another machine where those tools are installed, running Linux or a UNIX variant.

The NBSP subdirectory contains the text files as well as the radar and satellite images that have been created on-the-fly by the server. Figures 4 and 5 show how a quick browsing of those directories gives an overview of what is available in the server at any time. In the case of the radar images, since they are so many of them, they are not lumped together in one directory but instead they are sorted according to the radar site that emits them, which is identified by a three-letter name.



Figures 4 and 5 - the Quick Browse directories



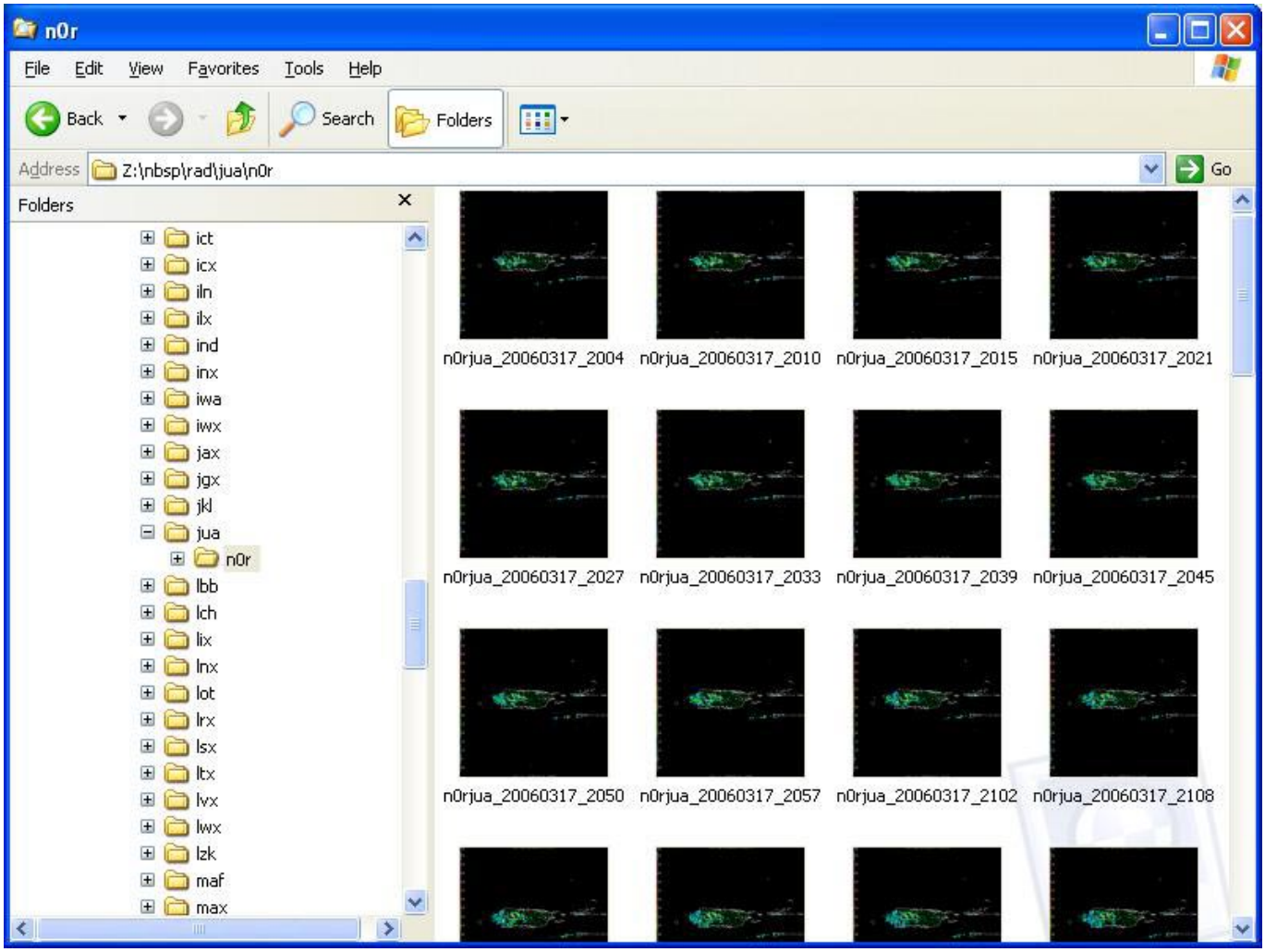


Figure 6 - A catalog of some of the radar images from the '*jua*' site in San Juan.

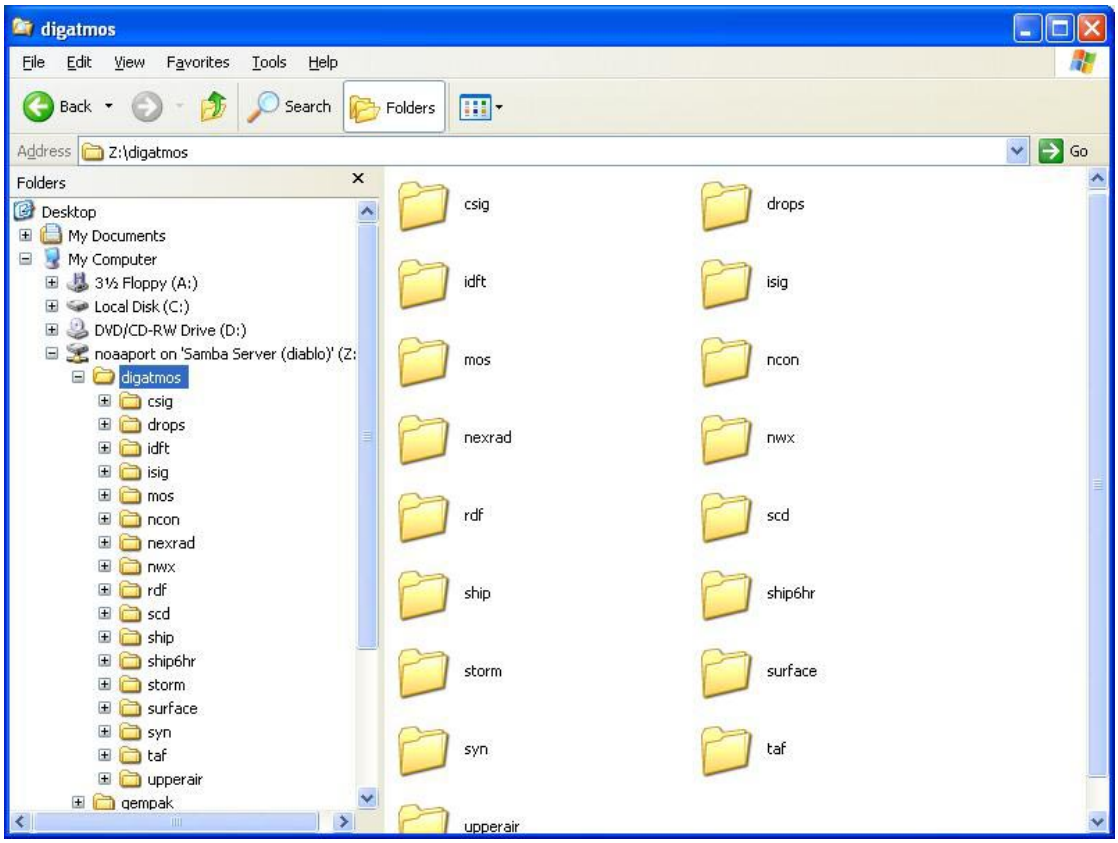


Figure 7

The '*digatmos*' directory, containing files saved in a format suitable for importing by *Digital Atmosphere*.

The other subdirectory of the *NOAAPORT* data directory is the '*digatmos*' directory, which contains the files saved in a format suitable for importing by *Digital Atmosphere*. There are no images in this directory, only data. Figure 7 shows the contents of this directory.

Working in Digital Atmosphere

Let us open the *Digital Atmosphere* program and process some of that data. In Figures 8 and 9 I am about to open a 'n0r' radar data file from the '*jua*' site and a surface data file, respectively.

Digital Atmosphere processess the radar data and displays the image shown in figure 10. Other types of radar data file can be procesed and displayed in a similar way.

Digital Atmosphere does not natively have the ability to process GOES satellite data to create an image. However, it does have a scripting capability which, in particular, allows any external program to be called and used as a DA extension if is suitably coded. What has been done to produce figure 11 (next page) is to use the *GINIview* program, which was written by David Taylor with this use in mind. A DA script first invokes *GINIview* with the appropriate parameters to produce and save the image, which is then loaded and displayed by DA.

The script can subsequently upload the image to a website if desired. Furthermore, the script can be scheduled to run periodically at any desired interval so that the entire process is run automatically without any manual intervention whatsoever.

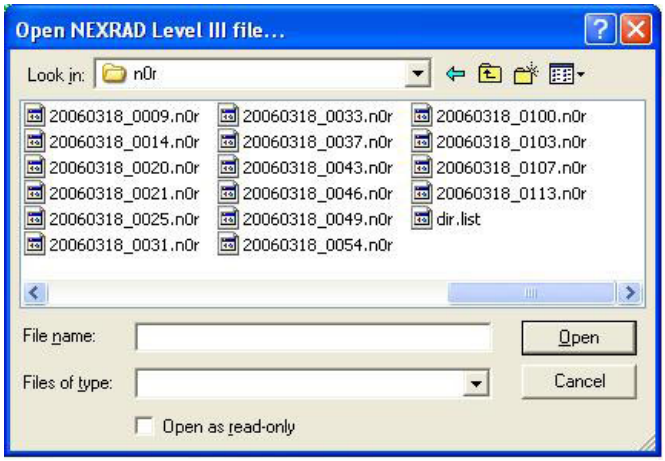


Figure 8 (upper right) - Opening a radar data file

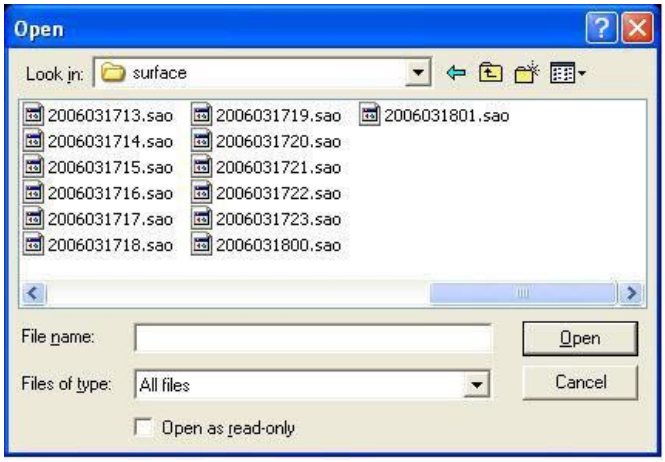


Figure 9 (lower right) - Opening a surface data file

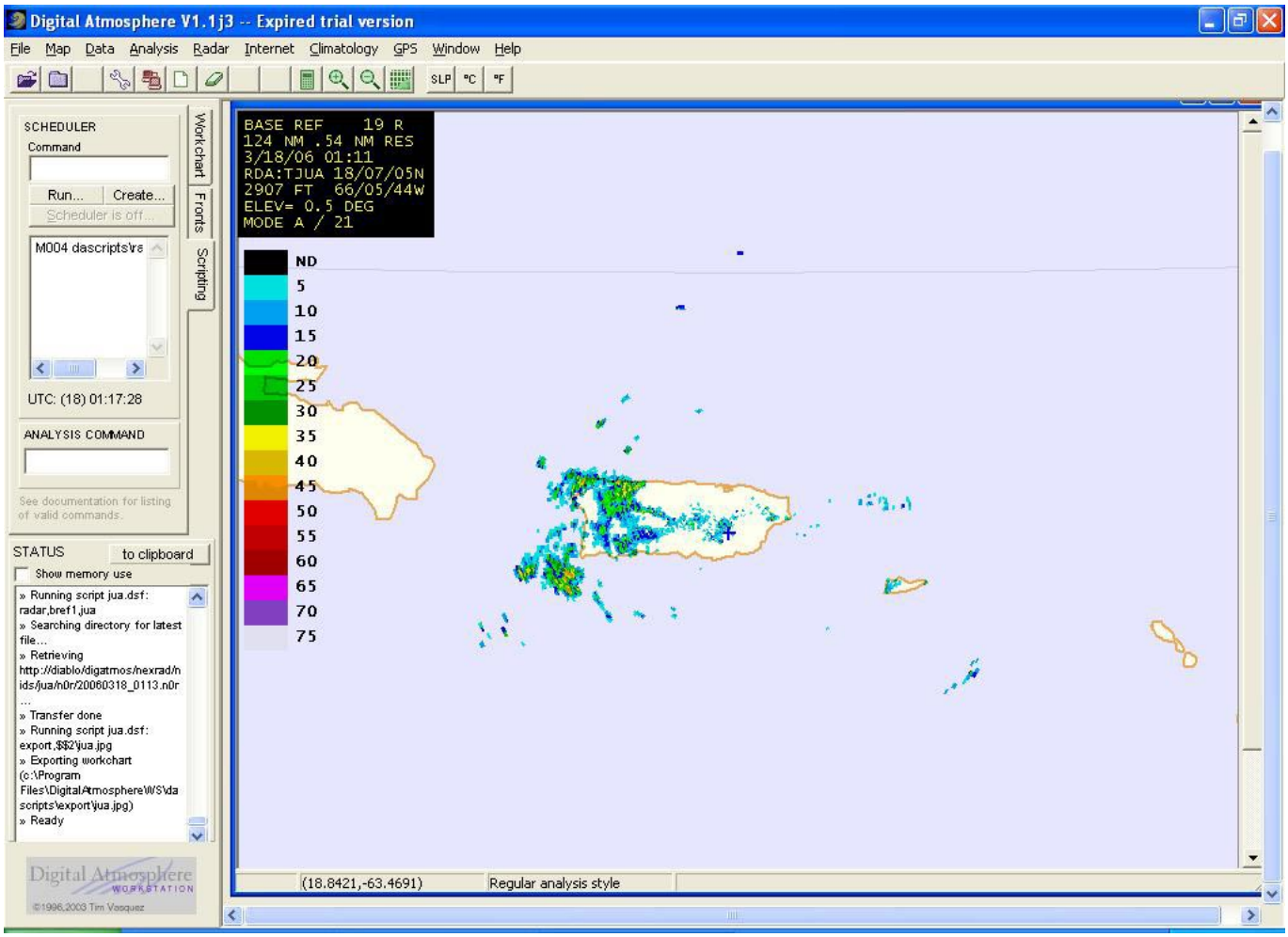


Figure 10 - A radar image processed in Digital Atmosphere

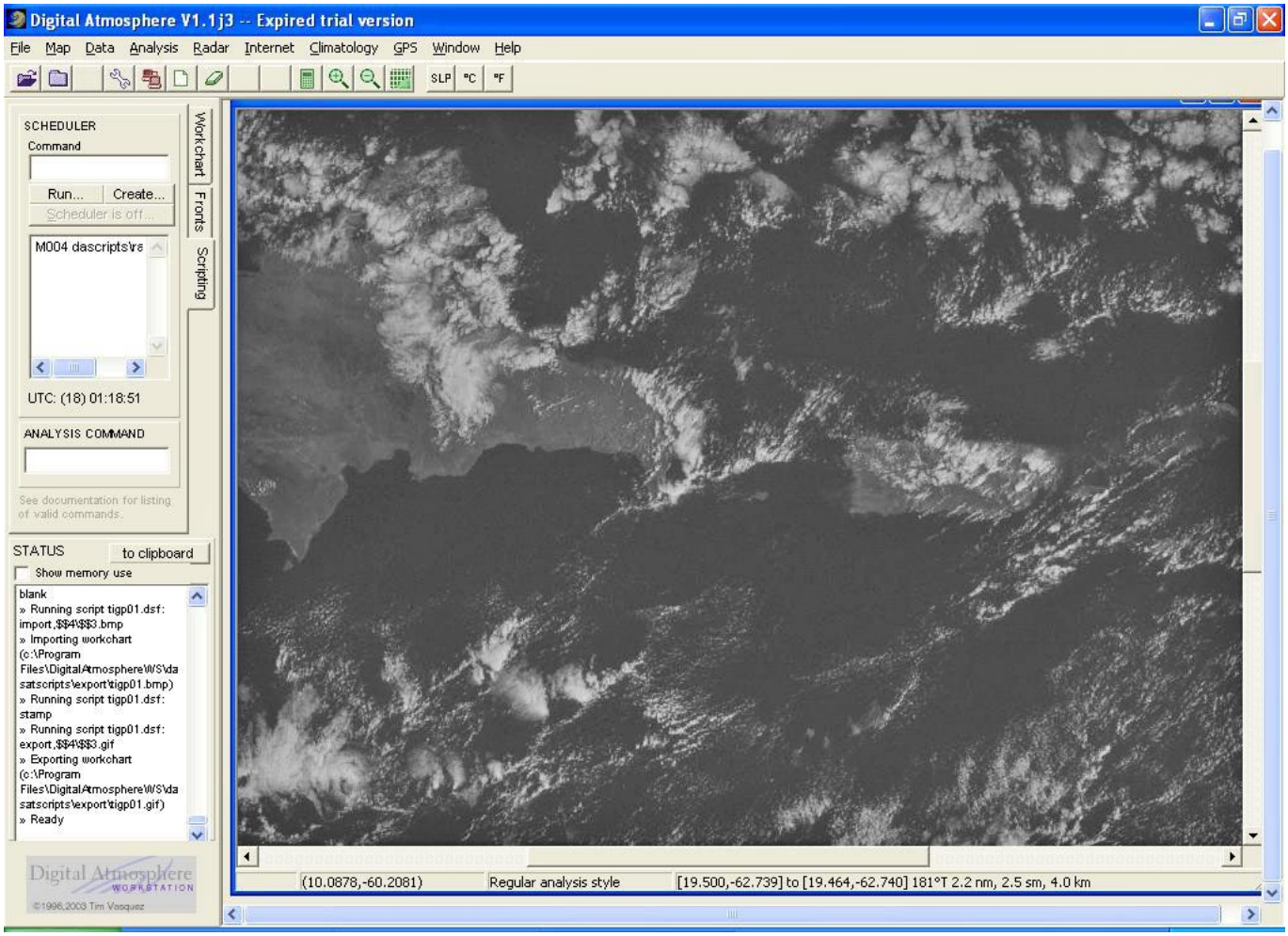


Figure 11 - The satellite image created from a GOES data file by a DA script that invokes the *GINIview* program

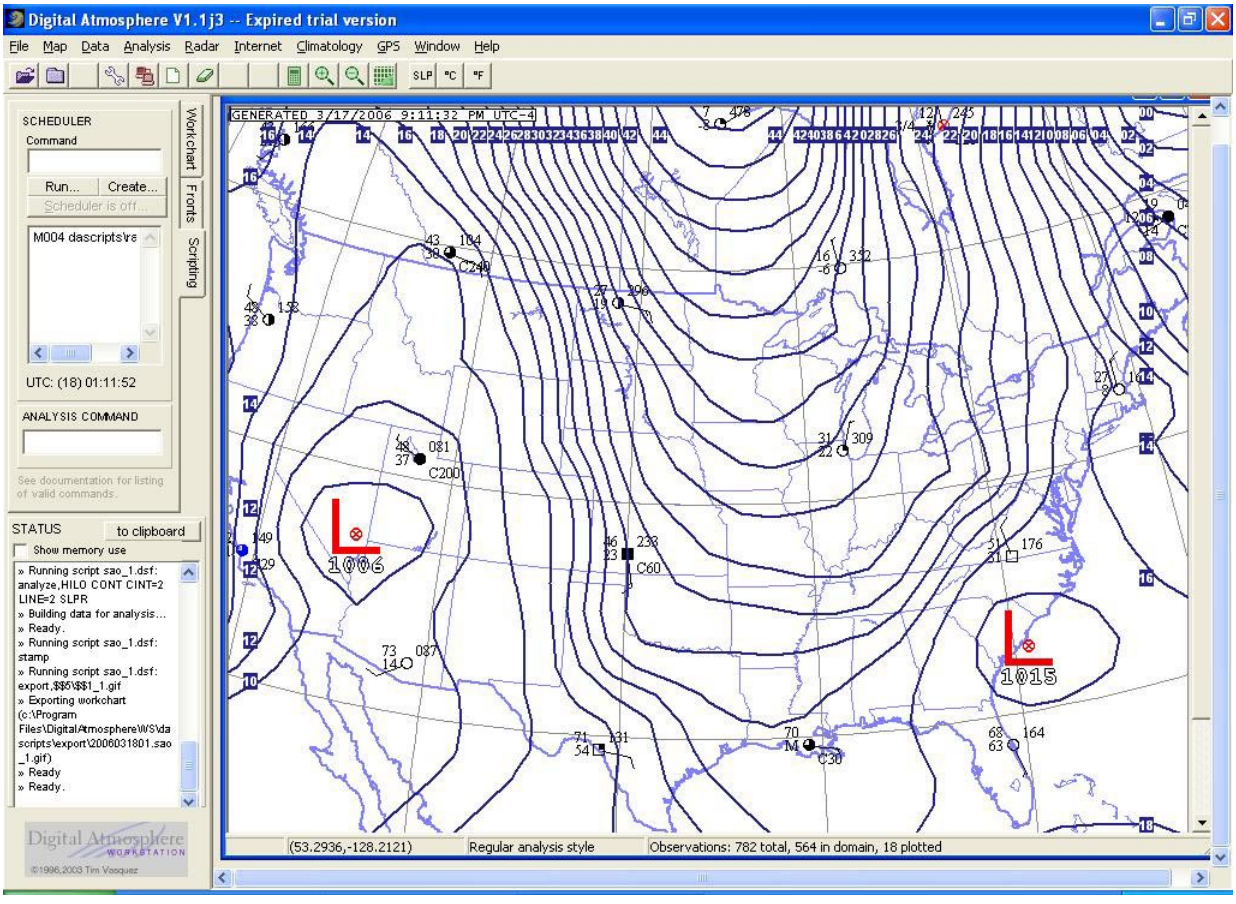


Figure 12 - A surface temperature and pressure map created from the data in the 'surface' subdirectory

The same thing can be done with any other kind of data that DA is able to process and analyze. Figure 12 (previous page, bottom) shows a surface temperature and pressure map which has been created from the data in the 'surface' subdirectory. As a matter of fact, readers will surely find more creative ways of using this kind of data, as in figure 13.

Figure 13 is also a surface map, but was created with the help of some DA scripting code that was mentioned in a forum, and which appeared on page 10 of the December 2005 issue of *GEO Quarterly*. I have included the script as an appendix to this article. The text in upper case is the code snippet that appeared in the forum message; the text in lower case is what I have added to complete the script.

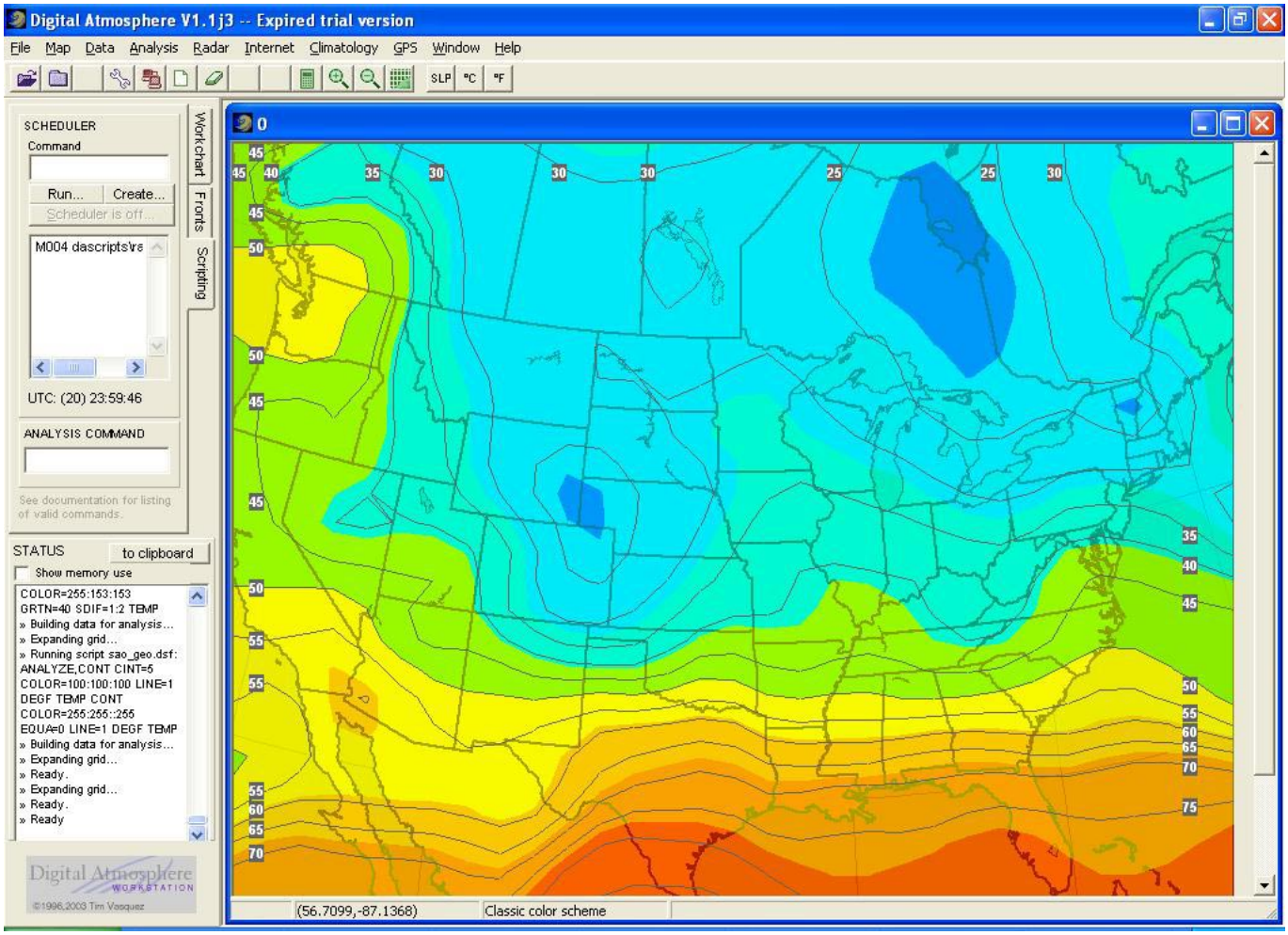


Figure 13 - Coloured surface map created using a Digital Atmosphere script (see appendix)

Advanced Configuration and Distribution

I like to have a global view of all the data that is available at any time. For this and other reasons, I have chosen NNTP as the protocol to use in NBSP for distributing the data files through the network. NNTP is a very efficient protocol for distributing files through the Internet and it existed long before the Web became popular in the form that it is today. It is widely used and there are NNTP server and client implementations for all computer architectures and operating systems. NBSP can distribute files by NNTP, through a *news* (NNTP) server. An NNTP server uses the concept of groups and articles. Each *NOAAPORT* product becomes an article that belongs to one (or more) group according to a predefined criteria. For example, some of the groups used in the NNTP server that I have setup are as follows:

- noaaport.kkkk.txt
- noaaport.sss.rad.n0r

which contain text bulletins and *n0r* radar images, respectively. Here and below, *kkkk* is any station ID, and *sss* stands for a radar site. The groups

- noaaport.raw.kkkk.txt
- noaaport.kkkk.data
- noaaport.raw.rad.sss.{n0r,...}

carry the raw data files corresponding to the text bulletins, data sent as text (e.g., METAR reports), and the compressed radar data files. Two special-purpose groups

- noaaport.misc.fadm,urgentg

carry administrative messages from the *NOAAPORT* system, and messages (weather bulletins) tagged as urgent. Then there are several groups that contain data that can be used directly by applications, and in particular *DigitalAtmosphere* (see below)

- noaaport.appdata.da.sao
- noaaport.appdata.da.syn
- noaaport.appdata.da.sb
- noaaport.appdata.da.nwx.spc.day1
- noaaport.appdata.da.nwx.spc.day2
- noaaport.appdata.da.nwx.spc.day3
- noaaport.appdata.da.nwx.hpc.fronts

There are two ways in which any other host can use this system: as an end-point client, or as a slave server.

As an end-point client, all it involves is the use of a news-reader software, such as *Tin* in UNIX, and *Outlook Express* in a Windows machine or any other from the many available. A

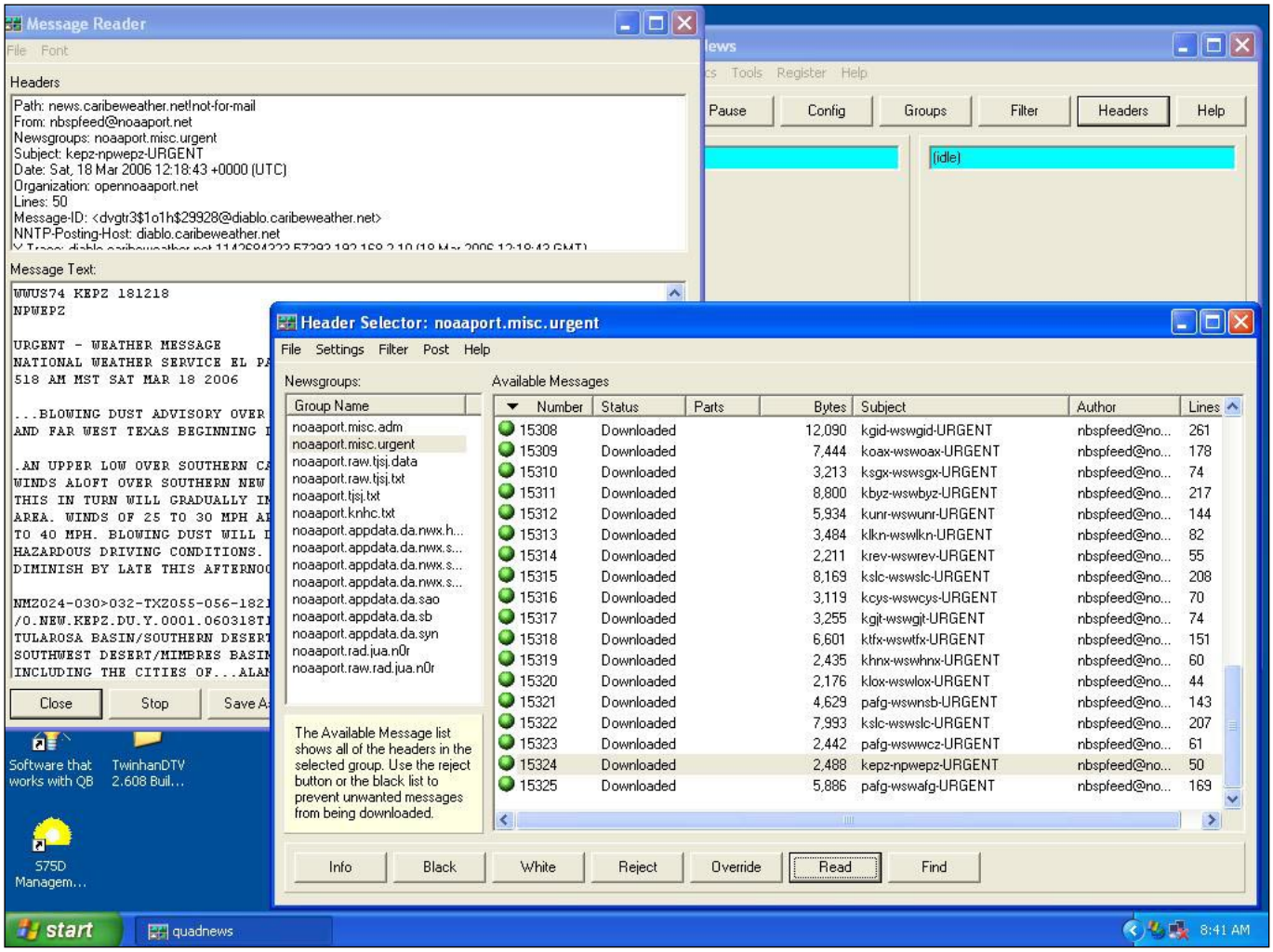


Figure 14 - Quadsucker in action

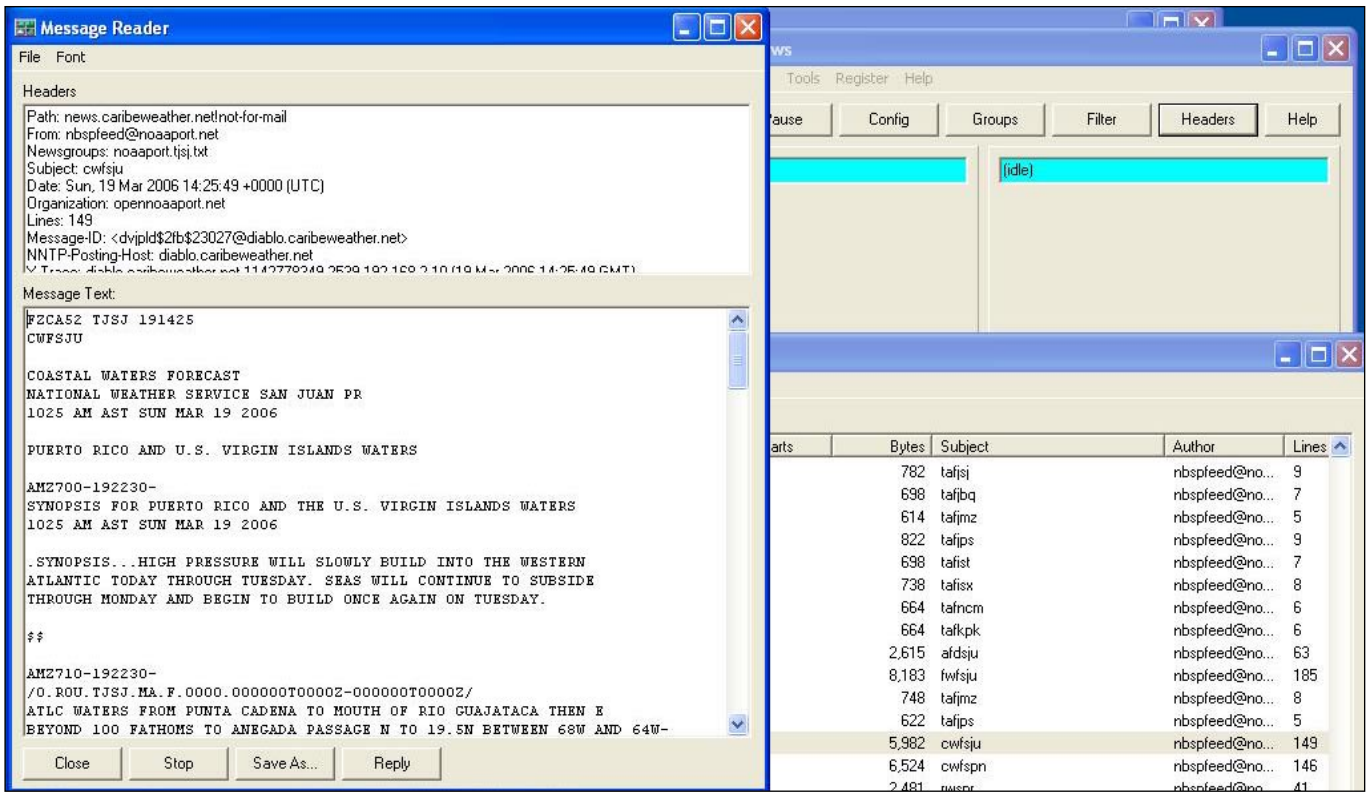


Figure 15 - A coastal waters forecast issued by the San Juan ofice of the NWS, and which was carried by the group *noaaport.tjsj.txt*

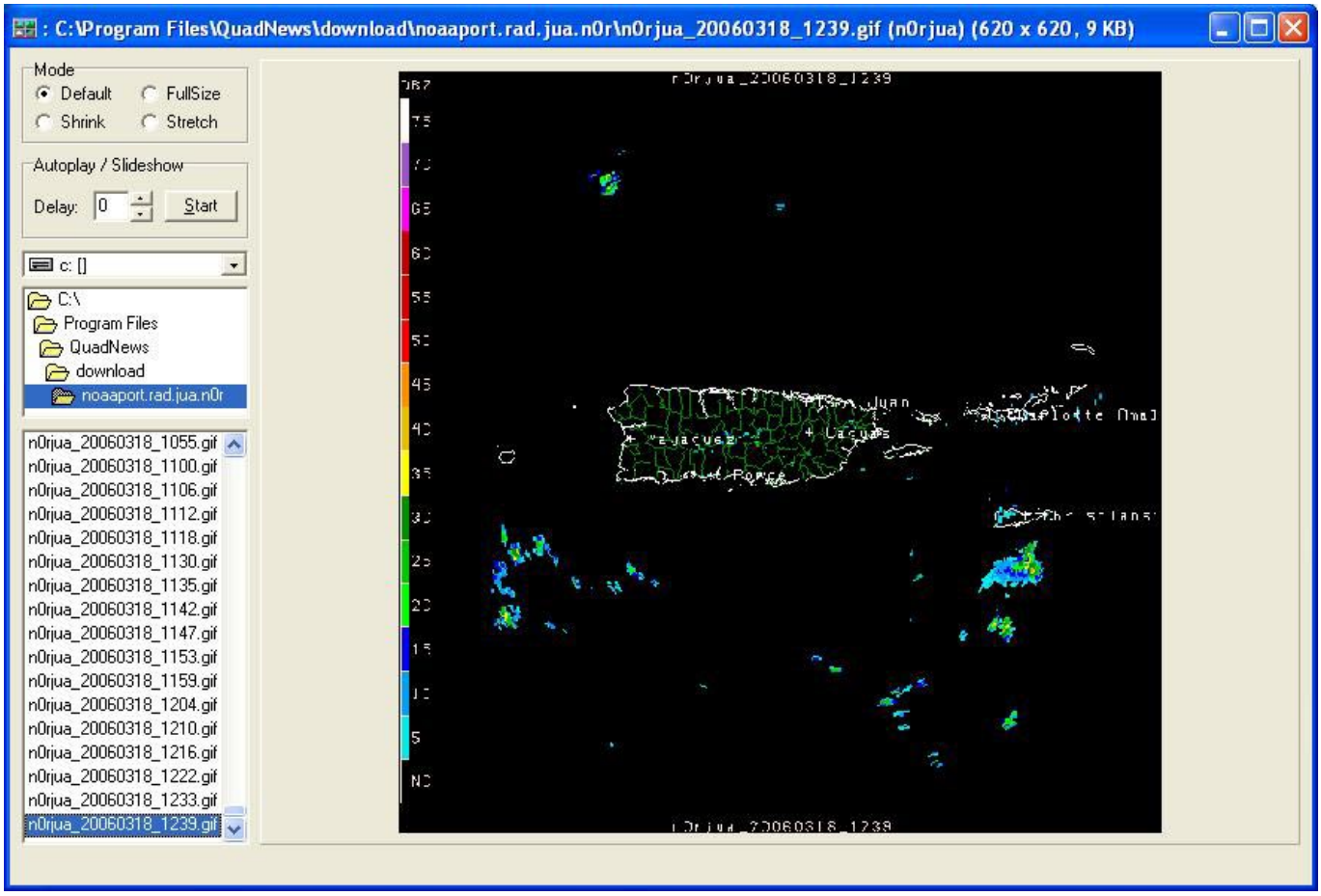


Figure 16 - A radar image and the catalog from the 'jua' site, carried by the group *noaaport.jua.rad.n0r*

particular one that I have tested under Windows, which is efficient, simple and inexpensive, is a program called *QuadSucker/News*, available from

http://www.quadsucker.com/quadnews.

After configuring the client program to use our news server as the NNTP server, anyone of the above groups can be subscribed. The text files and the radar (gif) images can be viewed from the news-reader itself, while the raw data files can be saved to disk and then imported by special-purpose programs, such as *Digital Atmosphere*, which we have tested successfully in this way.

Figure 14 (previous page) shows *QuadSucker* in action. The topmost window shows the groups to which I have subscribed myself, out of the groups mentioned above. When I select one particular group from the group list on the left (the *noaaport.misc.urgent* in this example), the list of articles in that group is displayed on the right. Then, selecting one of these displays it in another window, as shown on the partially covered window on the left. Figure 15 similarly shows a coastal waters forecast issued by the San Juan office of the NWS, and which is carried by the group *noaaport.tjsj.txt*. Figure 16 shows a radar image and the catalog from the '*jua*' site, carried by the group *noaaport.jua.rad.n0r*.

Once the groups to subscribe are configured in the client, the program can take care of downloading the newer articles as they become available from the server, without manual intervention. This capability, together with the global overview of the data that this distribution method offers, is a very useful feature of this system.

A slave server involves setting up a news server in the slave machine, with a feed from a master server. A slave server is then able to relay the data to news clients as described above, as if the slave were receiving the data directly from the dish, when in fact it is not. To the client it makes no difference, and in fact it cannot tell.

Conclusions

It is not possible to discuss in detail all the configuration options of NBSP in an article of this type. Thus, I have limited myself to mention the salient and simple capabilities that I find most useful for common uses. Although I have chosen to use Digital Atmosphere as the example client application, there are several others available, such as the *GRlevel3* program (http:/www.grlevelx.com) which is becoming very popular, and of course *Gempak*, among others.

My intention has been to illustrate, within the scope of this article, the essential features of the *NOAAPORT* system in general, and of NBSP in particular. The open nature of this system, both the specifications and software, allows it to be extended in ways that are limited only by our imagination.

Acknowledgments

I would like to thank Stonie Cooper, of Planetary Data Inc, who has always been very helpful with my questions about *NOAAPORT*.

APPENDIX: Digital Atmosphere SCRIPT FOR Figure 13

```
set,1,@FFZ@MMZ@DDZ@HHZ.sao
set,2,z:\digatmos\surface
set,3,c:\Program Files\DigitalAtmosphereWS\data
set,4,c:\Program Files\DigitalAtmosphereWS\dascripts\maps\us.dmf
set,5,c:\Program Files\DigitalAtmosphereWS\dascripts\export
#set,6,ftp://user:passwd@host.domainname.com/path/to/directory
blank
load,$$4
ingest,$$2\$$1
####
ANALYZE,STOR=1 TEMP
#-----------------------------------
ANALYZE,STOR=2 SDVC=1:1000
#-----------------------------------
ANALYZE,OVER FILL COLOR=000:239:255 GRTN=0 SDIF=2:1
ANALYZE,OVER FILL COLOR=000:255:209 GRTN=0 SDIF=1:2
ANALYZE,OVER FILL COLOR=000:154:255 GRTN=5 SDIF=2:1
ANALYZE,OVER FILL COLOR=159:255:000 GRTN=5 SDIF=1:2
ANALYZE,OVER FILL COLOR=000:080:255 GRTN=10 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:255:000 GRTN=10 SDIF=1:2
ANALYZE,OVER FILL COLOR=082:000:255 GRTN=15 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:204:000 GRTN=15 SDIF=1:2
ANALYZE,OVER FILL COLOR=179:000:255 GRTN=20 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:161:000 GRTN=20 SDIF=1:2
ANALYZE,OVER FILL COLOR=242:000:242 GRTN=25 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:097:000 GRTN=25 SDIF=1:2
ANALYZE,OVER FILL COLOR=204:000:204 GRTN=30 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:014:014 GRTN=30 SDIF=1:2
ANALYZE,OVER FILL COLOR=153:000:153 GRTN=35 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:110:110 GRTN=35 SDIF=1:2
ANALYZE,OVER FILL COLOR=122:000:116 GRTN=40 SDIF=2:1
ANALYZE,OVER FILL COLOR=255:153:153 GRTN=40 SDIF=1:2
# basemap
ANALYZE,CONT CINT=5 COLOR=100:100:100 LINE=1
ANALYZE,CONT COLOR=255:255::255 EQUA=0 LINE=1
####

#stamp
#export,$$5\$$1fi1.gif
#upload,$$6/$$1fi1.jpg,$$5\$$1fi1.jpg
```