Creating a software tool to reuse existing decoders

Shanna-Shaye Forbes

Academic Affiliation, Fall 2005: Senior, University of Texas at Austin

SOARS® Summer 2005

Science Research Mentor: Anne Wilson
Writing and Communication Mentors: Kirsten Butcher, Sonal Bhushan

ABSTRACT

The Local Data Manager (LDM) was created by Unidata at the University Corporation for Atmospheric Research (UCAR) to relay atmospheric data in near real time to its community of universities and research facilities. Although the LDM successfully delivers data to its community, Unidata is considering other alternatives with more advanced options for data relay. One alternative to the LDM, the Network News Transfer Protocol-based InterNet News (INN), was shown to successfully deliver data. Since the interface that INN provides to these programs is different from the LDM’s interface, a new piece of software to pass INN articles to existing decoders was required. Decoders are used to transform data into a variety of formats for purposes such as visualization and input to other programs. This project’s aim was to create a software tool called a decoder wrapper that allows data to be decoded via INN just as it is decoded via LDM. The decoder wrapper was created in C and then upgraded to an object oriented design to facilitate code comprehension and future modification. The created decoder wrapper will facilitate the use of both LDM and INN on the relay network, by putting INN’s data products into a format desired by INN users. It will also facilitate further testing within Unidata of INN’s capabilities for data relay.

This work was done under the auspices of the Significant Opportunities in Atmospheric Research and Science (SOARS®) program of the University Corporation for Atmospheric Research, with funding from the National Science Foundation, the U.S. Department of Energy, the National Oceanic and Atmospheric Administration, and Goddard Space Flight Center, NASA.
Section 1. INTRODUCTION

Unidata exists to serve a community of universities and research facilities. This community has a pressing need for earth observational data, such as tornado warnings, watches, earthquake information, and model output, to be delivered in a timely fashion. In response to this community need, Unidata created the Local Data Manager (LDM) to “select, capture, manage, and distribute arbitrary data products” (Unidata-LDM, 2005) via its Internet Data Distribution (IDD) network. The LDM has proven to be extremely successful and is predicted to continue relaying data successfully for the next five years. Unidata, however, expects to see significant increases in data volume within the next ten years, which may require a different method of data relay. In particular it is presumed that a combination of push and pull data relay will be necessary. Push based data relay is based on a subscription list and pull based data relay is done on demand.

INN, an open source news server based on network news transfer protocol, is currently being explored as a possible alternative to the LDM in the future. INN supports push based transmission, at least as well as the LDM, and has many more features that include pull-based retrieval, longer term storage, metadata support, automated routing, support for browsing metadata and many other features. These additional features of INN could be beneficial in providing access to data. The actual benefits can be determined after Unidata is able to incorporate INN nodes into the IDD.

To enable both INN and LDM to interact on the IDD, three tools are needed: an LDM to INN bridge, a decoder wrapper to enable INN to use existing decoders, and an INN to LDM bridge. Although a LDM to INN bridge (a software tool to retrieve data from LDM and send it to INN) exists, an INN to LDM bridge (a software tool to retrieve data from INN and send it to LDM), which would enable two way communication between LDM and INN, does not yet exist. This project created the decoder wrapper which sends INN data products to LDM decoders so that INN users can obtain INN data products in a usable format. It also allows a site to retrieve and use data from an LDM or INN site, and to relay to another INN site, but not yet to another LDM site.

This remainder of this paper is divided into four sections. First, background information about INN and LDM is presented. Next, a discussion of the implementation method is provided. We then provide an overview of the newly created software tool’s performance. Finally, the paper concludes with a summary of the impacts of the newly created software tool and also discusses possible future developments.

Section 2. BACKGROUND

This section discusses the Internet Data Distribution system, the Local Data Manager and also InterNet News.

2.1 Internet Data Distribution System (IDD) - A private Internet-based network for delivery of data in near real time

The IDD is originally “one of the first systems to use the Internet as an underlying framework for delivering data,” (Baltuch, 1997) and was nationally implemented in 1996.
The IDD “ingests data at a source site and then ships via the LDM via the Internet to one of its downstream servers” (Baltuch, 1997).

The IDD is a cooperative effort. It is heavily dependent on the willingness and the ability of other sites to relay data streams to a fixed number of other streams. The IDD allows users to ‘subscribe’ to certain sets of data products and IDD servers then deliver the requested data to the local servers as soon as they are available from the source” (Domencion, 2003).

2.2 The Local Data Manager (LDM) - *The software that drives the IDD*

The Unidata Local Data Manager is a collection of cooperating programs that select, capture, manage and distribute arbitrary data products. The system is designed for event-driven data distribution and is currently used in the Unidata Internet Data Distribution network. The LDM system includes network client and server programs and their shared protocols and supports flexible site-specific configuration” (LDM 6.4, cited 2005). The LDM is user configurable, event driven (handles data as it arrives) and also supports distributed processing. It is designed to run on a UNIX machine which has enough storage space for the data requested, a TCP/IP Ethernet connection, a standard C compiler, and Perl. The LDM currently handles “NEXRAD radar data, lightning data from the National Lightning Detection Network, GEOS satellite imagery” (Unidata-LDM), and many other arbitrary data types.

2.3 InterNet News (INN)

InterNet News (INN) is “an open source news server based on the Network News Transfer Protocol,” (Wilson, 2004). An INN network that delivers data retrieved from the IDD was created and demonstrated to successfully extract data from LDM and send to INN through the use of a software tool known as pqToInews. This network demonstrated that it is capable of delivering the entire contents of the IDD in a robust, reliable, and timely manner. INN uses NNTP protocol which includes support for automated routing via a flooding algorithm that automatically selects the fastest route to deliver data (Wilson, 2004). INN is freely available open source code maintained by a community of developers since the early 1990’s.

2.4 Decoders

A decoder transforms data from one format into another. For example, a decoder may take information in binary format and save it in netCDF format. A decoder reads from standard input, recognizes boundaries between data from different data products, and waits for more input when there is none. It should also use LDM conventions to log error messages. Unidata provides decoders for “NOAAPORT, METAR, ship/buoy, synoptic and upper air observations, and NCEP model GRIB messages.” These decoders include the gribtonc decoder, Perl decoders, GEMPAK LDM decoders, McIDAS-XCD decoders and LDM-McIDAS decoders.

2.4.1 Gribtonc decoder

Gribtonc converts gridded binary data (GRIB) products to netCDF files and stitches together multiple GRIB products into a single file. Gribtonc was developed as a model LDM in-
line decoder, and requires auxiliary CDL files that specify the netCDF file structure. The gribtonc decoder also requires netCDF and udunts libraries.

2.4.2 Perl decoders

Decoders for metars, buoy reports, surface reports, synoptic reports, and upper air reports were created in Perl as metar2nc, buoy2nc, sao2nc, syn2nc, and ua2nc respectively. These decoders were written and are fully supported by Unidata and require the netcdf-perl package.

2.4.3 GEMPAK LDM decoders

Gempack decoders are a set of real time decoders to convert radar from LDM to Gempak format as the data arrives. Thoroughly supported by the Unidata Program Center, some of the available decoders include DCUAIR to convert radiosonde upper air data and DCNLDN to convert NLDN lightning data.

2.4.4 McIDAS-XCD decoders

McIDAS decoders are a set of real-time decoders that convert data from the LDM into McIDAS format as the data arrives. A few of the available decoders are DMRAOD to convert mandatory and significant level upper air data, DMSYN to convert synoptic code, ship and buoy data, and DMSFC to convert SAO/Metar surface data.

2.4.5 LDM-McIDAS decoders

LDM-McIDAS decoders are LDM compatible, generate McIDAS data files, and read data from standard input. They can be told where to write data, invoked by LDM pqact program, and invoked to stand alone. Some of the available decoders are ldm-mcidas, nids2area, and nldm2md.

Section 3. METHODOLOGY

3.1 Predecoder wrapper development

Since INN requires development of an INN to LDM decoder wrapper, it was necessary to have INN and LDM running on a local UNIX machine. LDM 6.3 and INN 2.4.1 were installed on an Athlon based dual 2 GHz Linux machine with 3.6 GB of RAM, currently running Fedora Core 3. After downloading and installing both programs, they were configured to default settings. The LDM 6.3 was configured to the settings recommended by the Unidata Program Center (LDM’s creator) and INN was configured to use a single monolithic and a single logical buffer to make it similar to LDM. The Unidata LDM decoders were also installed in the INN directory because they would be used by the decoder wrapper.

3.2 Decoder wrapper requirements

To be successful the decoder wrapper had to integrate with INN. The decoder wrapper had to link with INN code to build and had to use the INN storage manager (a set of functions that allow access to INN buffers) to run. In addition the decoder wrapper had to filter on product ids, perform substitutions of matched substrings from product id into a template, and build
required directories dynamically. It also had to generate complete date information from day and hour time stamps, write to the standard input of decoders and run indefinitely.

3.3 Decoder Wrapper Tasks

The tasks of the decoder wrapper launched either by innfeed or from the command line are fairly straightforward and are listed below.

The decoder wrapper should:
1. Retrieve from the command line
   a. the pattern against which the product ids will be filtered
   b. the name and location of the decoder to be called
   c. any arguments the decoder may take
2. Compile the retrieved pattern as a regular expression
3. Initialize the storage manager to enable article retrieval from INN
4. Get an article token
5. Retrieve the article’s product id and compare it to the regular expression pattern
   If there is a match:
   a. Perform any substitutions from the product id into the arguments to the decoder
   b. Dynamically create directories
   c. Retrieve article bodies
   d. Decode products buffer by buffer
   e. Pipe the decoded products to the appropriate decoders.
   f. Return to step four
   If there is no match:
   a. Return to step four

3.3.1 Regular Expressions

A regular expression is a text string to decode a search pattern. Regular expressions are used for pattern matching and also performing substitutions. To use a regular expression memory must be allocated, the expression must be compiled, compared to a string, and substitutions based on a template must be performed. Most programming languages provide support for matching arbitrary strings against regular expressions, including returning sub-strings of a matched string that can be used elsewhere.

3.4 Decoder Wrapper

The project was executed in two distinct stages. First a basic wrapper for the preexisting decoders was created, and then an object oriented approach was taken to facilitate code maintenance and to increase comprehension. Code for the decoder wrapper was written in C because it is a widely used standardized programming language, a C compiler is required to build INN, and also because an incompatibility exists between an INN header file and the C++ compiler. The C code was compiled with the GNU Compiler Collection (gcc), which “includes a front end for C, C++, Objective-C, Fortran, Java and Ada” (GCC, cited 2005).
The incompatibility occurred because there was a conflict with preexisting INN code and the C++ compiler. An INN header file storage.h included in the decoder wrapper code contained the C++ reserved words class and private. Since storage.h was created as a C header file the reserved words were not used in a C++ context. They were used as variable names inside two separate structures. Although renaming the variables in each structure from class to sclass and private to dprivate was considered, it was determined to be unacceptable. It would require modifying other fully functional and fully tested INN code that made use of the structures defined in storage.h.

Stage 1: Non object oriented design of the decoder wrapper

Stage 1 of the decoder wrapper, an unstructured prototype, was written to ensure that at the most basic level the decoder wrapper was fully functional. In addition, to provide a starting point, during stage 1 the decoder wrapper retrieved four mandatory arguments from the calling process or from the command line. In addition to being unmodularized, memory for regular expressions used for INN product id matching and substitution had to be allocated individually, set, compiled, and substituted independently. As shown in Figure 1, Stage 1’s implementation provided few abstractions, had little modularity and thus created barriers to understanding and maintenance. In fact the code in stage 1 would have proven very difficult to understand and maintain by anyone unfamiliar with the specific tasks of the decoder wrapper. After the decoder wrapper worked at this basic level, an object oriented approach was taken to the decoder wrapper design in Stage 2.

Figure 1. Synopsis of the unmodularized main used in stage 1 of the decoder wrapper.

Stage 2: Object oriented design of the decoder wrapper

Stage 2 of the decoder wrapper featured an object oriented design, and the code was also modularized to increase code comprehension and to facilitate future code modification.

To accomplish the object oriented design of the decoder wrapper, a regular expression aware string class was created to abstract the specific implementation of a regular expression
from the programmer. Although C is a procedural and not an object oriented language, a structure describing the regular expression string was created and functions to act on the structure were created to emulate class methods. As shown on the right side of Figure 2, functions were created to construct, destruct, compile, compare, and perform substitutions on regular expressions so that detailed knowledge of regular expressions and their implementation would be unnecessary for anyone using the decoder wrapper code. As an added safety net a check to see if the regexstring object being modified has been constructed is performed in all the ‘class’ functions, and in the substitute ‘class’ function a check for a match is done before any of the functions tasks can be performed.

In addition to the regular expression aware string class, the code was modularized to increase comprehension and also to ensure that specific tasks were being performed accurately. The abstracted modularization is shown in Figure 2 and should be much easier to comprehend and follow than code shown in Figure 1. As another improvement, the decoder wrapper code was modified to handle decoders having up to ten arguments. In addition because of the original modular structure of the substitution function in stage 1, no modification to the function was necessary in stage 2 to ensure that an arbitrary number of arguments would be correctly substituted into by a matching product id. This simply means that the substitute function is called multiple times instead of a fixed number of times.

Figure 2. Synopsis of the object oriented design of the regular expression string class and the modularization of the tasks of the decoder wrapper.

3.5 Inner workings of LDM and INN

The LDM receives data products from an upstream node and stores the data in its product queue. Based on its subscription list, it relays data products to downstream nodes and or sends its data products to decoders. LDM has a tool known as pqact that retrieves article bodies from the product queue, dynamically creates necessary directories and sends the data to the LDM decoders. The decoders then modify the format of the data to produce a user desired format. One of the commonly used formats is netCDF, which is the network Common Data Format. A visual representation of these actions is shown in Figure 3.
InterNet News receives information from an upstream node and stores the retrieved data in the INN buffer. INN then has the option to relay data to a downstream node and or act on the data in the buffer. Like the LDM, INN has a tool, infeed, to act on the contents of the buffer. Based on the entries in the newsfeeds configuration file infeed launches multiple instances of the decoder wrapper to filter on different types of data products. A visual representation of INN workings is shown in Figure 4.

INN’s newsfeeds configuration file makes it very easy to transition from the LDM’s pqact.conf configuration file. As demonstrated by Figures 5 and 6, to make an entry in the newsfeed file a user could copy a pqact.conf entry into the newsfeeds file and expect to make only slight changes to the entry. Apart from providing complete path information which may not be necessary and removing the PIPE command, the last five entries remain the same. The only
visible difference in the entries is the need to specify the complete path for the decoder wrapper executable. In INN the feedtype specification (DDS\|IDS) is handled via the newsgroup name which in this case is unidata.binaries.idsddplus because innfeed does the filtering automatically.

```
DDS\|IDS ^S(A....|P....|XUS8.) .... ([0-3][0-9])
  PIPE /local/ldm/decoders/metar2nc
  etc/metar.cdl
  data/pub/decoded/netcdf/surface
  (\2:yyyy)(\2:mm)
```

Figure 5. Excerpt from the pqact.conf file to call the decoder wrapper for the IDSDDPLUS feedtype with a METAR data pattern and arguments to the decoder

```
decoderWrapper:!*,unidata.binaries.idsddplus)
  :Tc,Wn:/usr/local/news/nldm/nldm-0.0.13/src/decoderWrapper/decoderWrapper \ 
  "^S(A....|P....|XUS8.) .... ([0-3][0-9])" \ 
  /opt/news/decoders/bin/metar2nc \ 
  /opt/news/decoders/etc/metar.cdl \ 
  /opt/news/var/DATA/newstuff \ 
  "(\2:yyyy)(\2:mm)"
```

Figure 6. Excerpt from the newsfeeds file to call the decoder wrapper for the IDSDDPLUS feedtype with the same METAR data pattern and arguments to the decoder as in Figure 5.

**Section 4. RESULTS AND DISCUSSION**

Because of the time constrains of the ten week research project a full scale test of the decoder wrapper was not possible. A small scale test was performed to ensure that all aspects of the decoder wrapper were functional.

Testing determined that the regular expression aware string class effectively allocated and deallocated memory, and also accurately performed date completion and substitution. Testing of the regular expression aware string class revealed that although the destructor is dependent on the programmer to call it to ensure proper memory management, the built in safety net to ensure that an unconstructed object could not be modified proved to be very effective.

To test the decoder wrapper we opted to match against a METAR data product pattern. METAR data is observational data at different locations across North America. The METAR specific regular expression “^S(A....|P....|XUS8.) .... ([0-3][0-9])” was constructed, compiled without errors and compared to the product ids received by INN. When a match was found a four step process was followed. First the contents of the data product was retrieved, next the command to launch the LDM decoder was successfully assembled to form the arguments passed to the decoder, then the assembled command was issued to the system, and finally the retrieved data product was piped to the decoder. As a simple check to see if the data was accurately sent from the INN buffer via the decoder wrapper to the to the LDM decoder, a

SOARS® 2005, Shanna-Shaye Forbes, 9
Unidata visualization tool known as the Integrated Data Viewer (IDV) was used to visualize the decoded METAR data.

As shown in Figure 7, IDV was able to display the data sent to the LDM decoders via the decoder wrapper. Thus we concluded that the decoder wrapper successfully invoked the decoder, completing the one way conversation between LDM and INN that the software tool pqToInews began. It is expected that INN is now able to retrieve the entire contents of the Internet Data Distribution network and to decode the data from its buffers for the user with existing LDM decoders.

![Visualization of the decoded METAR data. Data for one station is shown in the table on the right.](image)

**Figure 7.** Visualization of the decoded METAR data. Data for one station is shown in the table on the right.

### Section 5. CONCLUSION AND FUTURE MODIFICATIONS

Although a full scale test of all the Unidata decoders was not possible due to the time constraints of the research, we assume that since the command to call the Perl METAR LDM decoder was successfully assembled from an arbitrary number of arguments, that the same will hold true for all other decoders whether written in C, Perl, or shell script.

Future modifications to this research could include completing the dynamic creation of directories to replicate LDM’s directory structure on INN, and also launching full scale tests of the decoder wrapper with all of the LDM decoders.
References


UCAR Unidata, cited 2005. Internet Data Distribution. [Available at http://my.unidata.ucar.edu/content/software/idd/index.html]

UCAR Unidata, cited 2005. Public Bulletins. [Available at http://my.unidata.ucar.edu/content/software/ldm/ldm-6.3.0/basics/feedtypes/public_bulletins.html#public]

UCAR Unidata, cited 2005. LDM-6.4. [Available at http://my.unidata.ucar.edu/content/software/ldm/ldm-6.4.1/index.html]
